Evaluatie en verbetering van foutconvergerende technieken voor IP netwerken

Evaluating and improving failure convergence schemes in IP networks

Pim Van Heuven

Promotor: Prof. Dr. Ir. Piet Demeester

Proefschrift tot het behalen van de graad van

Docotor in de Toegepaste Wetenschappen: computerwetenschappen

Universiteit Gent

Faculteit Toegepaste Wetenschappen

Vakgroep Informatietechnologie

Academiejaar 2002-2003

Dankwoord

Als u het mij toestaat, zou ik het behalen van een doctoraat willen vergelijken met een lange wielerrit. Deze rit, en dit zal ieder kunnen bevestigen die deze rit reeds heeft afgelegd, eindigt na een laatste steile beklimming: het schrijven van het doctoraat. Ondanks het individuele karakter van schrijven, kan het behalen van een doctoraat, als ik de beeldspraak nog wat verder doortrek, het best worden beschouwd als een teamsport. Vandaar dat ik aan het einde van mijn inspanning een aantal teamleden wil bedanken.

Eerst en vooral dank ik mijn promotor Piet Demeester voor het vertrouwen dat hij mij heeft geschonken en voor het feit dat er op de belangrijke momenten steeds een plaatsje vrij was in zijn drukke agenda.

Het IST Tequila project heeft zonder twijfel een belangrijke rol gespeeld in mijn doctoraat. Ik dank dan ook al mijn collega's voor de interessante tijd, niet enkel op technisch gebied maar ook op persoonlijk vlak. Ik dank in het bijzonder Eleni Mykoniati, David Griffin en Panos Georgatsos voor hun eerlijkheid en oprechtheid, ook in moeilijker tijden (cheers!), maar ook mijn IBCN-collega's Steven Van den Berghe en Jan Coppens. Daarnaast dank ik Steven nog voor het nalezen van mijn doctoraat en Jan voor het werk dat hij heeft verzet tijdens zijn licentiaatsthesis. Ook Erik Van Breusegem ben ik dank verschuldigd voor het analyseren van de Zebra-broncode in het kader van zijn thesis.

Didier Colle wil ik bedanken voor het kritisch nalezen van mijn thesis en het meedenken rond FTCR (over FTCR zal u nog veel te weten komen als u dit werk verder leest). Filip De Turck dank ik voor de medewerking aan de MPLS-sectie van de Wiley Encyclopedia of Telecommunication.

Als ik wat verder terugdenk dan dank ik Henk Neefs, die mijn thesisbegeleider was, zonder hem was ik waarschijnlijk nooit aan een doctoraat begonnen. Ook Mike Vogeleer dank ik omdat hij me bij het toetreden tot onze onderzoeksgroep op het goede pad heeft gezet.

Hilary Kerslake wil ik danken voor het taalkundig doorworstelen van mijn doctoraat. Ik besef maar al te goed dat dit geen sinecure is voor een leek.

Het spreekt vanzelf dat elke inspanning moet worden afgewisseld met ontspanning. Sta mij even toe een groet uit te brengen aan de vaste slachtoffers van dienst: Toem, Tom en Zita. Daarnaast wens ik Mark nog veel succes met zijn doctoraat.

Een nieuwe uitdaging staat al voor de deur: een eigen vennootschap. Danny Cools mijn toekomstig medezaakvoerder bedank ik alvast voor het vertrouwen.

Ten slotte rest mij nog mijn ouders te bedanken voor het leveren van het genetisch materiaal, voor de opvoeding en de steun, soms op afstand, soms heel concreet en direct.

Aan iedereen die ik opgenoemd heb, maar ook aan degenen die ik spijtig genoeg vergeten ben:

Bedankt!

Pim Van Heuven, 16 juni 2003

Abbreviations

ACTS	Advanced Communications Technologies and Services
AF	Assured Forwarding
API	Application Program Interface
AS	Autonomous System
ATLANTIS	Test Lab for Advanced Network Technologies and Integrated Services
ATM	Asynchronous Transfer Mode
BA	Behaviour Aggregate
BGP-4	Border Gateway Protocol, version 4
BSD	Berkeley Software Distribution
CBS	Committed Burst Size
CLI	Command Line Interface
CR	Constraint-based Routing
CR-LDP	Constraint-based Routing-Label Distribution Protocol
DS	Diffserv
DSCP	Diffserv Code Point
ECMP	Equal Cost Multi-Path
ECN	Explicit Congestion Notification
EF	Expedited Forwarding
EGP	Exterior Gateway Protocol
E-LSP	EXP inferred PSC LSP
FAQ	Frequently Asked Questions
EXP	EXPerimental (field)
FEC	Forward Equivalence Class

FIB	Forwarding Information Base
FIS	Failure Indication Signal
FR	Frame Relay
FRS	Fault Recovery Signal
FSL	FTCR Switch LSR
FTCR	Fast Topology based Constrained Rerouting
FTN	FEC To NHLFE
GPS	Global Positioning System
G-MPLS	Generalised-MPLS
IBCN	Intec Broadband Communication Network
ICMP	Internet Control Message Protocol
ITHACI	Internet and the ATM: Experiments & Enhancements for Convergence and Integration
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
ILM	Incoming Label Map
INTEC	(department of) Information Technology
IP	Internet Protocol
IPv4	Internet Protocol version 4
IS-IS	Intermediate System – Intermediate System
ISO	International Standards Organisation
ISP	Internet Service Provider
IST	Information Society Technologies
ITU	International Telecommunications Union
LAN	Local Area Network
LDP	Label Distribution Protocol
LER	Label Edge Router
LIB	Label Information Base

LSA	Link State Advertisement
L-LSP	Label only inferred PSC LSP
LSP	Label Switched Path
LSR	Label Switch Router
MAC	Media Access Control
MPLS	MultiProtocol Label Switching
MF	Multi Field
MIB	Management Information Base
MPLS	Multi Protocol Label Switching
NHLFE	Next Hop Label Forwarding Entry
NIC	Network Interface Card
NLRI	Network Layer Reachability Information
NTP	Network Time Protocol
OA	Ordered Aggregate
OAM	Operations And Maintenance
OMP	Optimal Multi-Path
OSI	Open Systems Interconnect
OSPF	Open Shortest Path First
PHB	Per Hop Behaviour
PML	Protection Merge LSR
PRC	Partial Route Computation
PSC	Per hop behaviour Scheduling Class
PSL	Protection Switch LSR
QoS	Quality of Service
RED	Random Early Detection
RFC	Request For Comments
RIB	Routing Information Base
RIP	Routing Information Protocol

RSVP	Resource reSerVation Protocol
RSVP-TE	RSVP for Traffic Engineering
SLS	Service Level Specification
SNMP	Simple Network Management Protocol
TB	Token Bucket
TCP	Transmission Control Protocol
TE	Traffic Engineering
TEQUILA	Traffic Engineering for Quality of Service in the Internet, at Large Scale
ToS	Type of Service
UDP	User Datagram Protocol
VPN	Virtual Private Network
VoIP	Voice over IP
WG	Working Group
WWW	World Wide Web

Table of contents

I.Nederlandse Samenvatting

I.1 Inleiding

In de loop van de jaren is het Internet een alomtegenwoordig hogesnelheidsnetwerk geworden dat bedrijfskritisch is. Fouten van korte duur in het netwerk lijden tot het verlies van grote hoeveelheden gegevens en het aantasten van de netwerkdiensten. Indien de fout langer duurt, dan heeft dit grote financiële gevolgen. Het is daarom van primair belang dat netwerkfouten worden vermeden. Dit is slechts mogelijk in beperkte mate. Vermits fouten niet te vermijden zijn, is het van belang om hun impact zo klein mogelijk te houden.

Dit werk behandelt de manier waarop netwerken in staat zijn zichzelf te herstellen na een fout. Technieken die men aanwendt om netwerken fouttolerant te maken worden foutconvergerende technieken genoemd. Het Van Dale groot woordenboek der Nederlandse taal definieert convergentie als: “het convergeren of convergent-zijn; samenkomst in een punt” [1]. Foutconvergentie aldus beschouwd is een groep van operaties, mogelijk gedistribueerd uitgevoerd, die als gemeenschappelijk doel hebben het netwerk te herstellen na een fout.

Het onderzoek naar methoden om netwerken meer fouttolerant te maken is een zeer uitgebreid en actief onderzoeksdomein. Het is daarom van belang dat we vooraf het onderwerp van dit werk afbakenen. Dit werk analyseert en evalueert de operationele procedures die worden toegepast na een netwerkfout. Het behandelt dus de acties die worden ondernomen direct nadat een fout wordt ontdekt. Dit werk behandelt niet hoe een netwerk gepland moet worden om meer fouttolerant te zijn. Bovendien zullen we slechts vluchtig bestuderen hoeveel extra middelen moeten worden ingezet om het verkeer in het netwerk tegen fouten te beschermen. We beperken ons ook tot IP netwerken, netwerken met meerdere lagen worden niet behandeld. Wat we wel beschouwen is het gebruik van “verscheidene protocol labelschakelen” (MPLS, *Multi-Protocol Label Switching*). Zoals we zullen zien biedt MPLS interessante foutconvergerende technieken aan IP netwerken aan.

Zoals reeds vermeld zullen we bestaande foutconvergerende technieken evalueren en verbeteren. Daarnaast stellen we ook een eigen ontwikkelde foutconvergerende techniek voor: snelle topologisch gebaseerde herroutering met beperking (FTCR, *Fast Topology based Constrained Rerouting*).

I.2 Overzicht

In deze sectie geven we een overzicht van dit werk. Na dit overzicht beschrijven we MPLS en de toepassingen ervan. Dit is een vertaling en samenvatting van onze bijdrage rond MPLS aan de “Wiley Encyclopedia of Telecommunication” [2].

Vervolgens geven we een overzicht van de bestaande foutconvergerende technieken. Deze zijn IP routering en MPLS routering en protectie.

Daarna beschrijven we FTCR, een foutconvergerende techniek ontwikkeld door de auteur. Het idee van FTCR was eerst gepubliceerd in 2000 in [3] als een nieuw foutconvergerende techniek voor MPLS netwerken. Latere publicaties [4, 5, 6] vergelijken de capaciteitsvereisten van de verschillende technieken, onderzoeken hoe de foutconvergerende technieken voor elektrische MPLS kunnen worden omgezet naar optische MPLS en beschrijven datacentrische optische netwerken en hun fouttolerantie. In deze publicaties speelt FTCR steeds een belangrijke rol.

De sectie rond FTCR behandelt hoe FTCR de trafiek in het netwerk rond fouten herrouteert. We behandelen eerst enkelvoudige fouten op een kortste pad LSP. Daarna beschouwen we enkelvoudige fouten op expliciet gerouteerde LSPs en vervolgens meerdere fouten. Daarnaast bekijken we hoe FTCR de trafiek terug op het kortste pad routeert, nadat een fout is hersteld.

De sectie rond FTCR wordt gevolgd door een sectie over de evaluatie van de reeds besproken foutconvergerende technieken. De criteria die worden gebruikt voor deze evaluatie zijn: de snelheid van convergentie, de schaalbaarheid, de stabiliteit en de extra hoeveelheid capaciteit die nodig is om een netwerk fouttolerant te maken. Voor de evaluatie van de convergentiesnelheid maken we gebruik van een eigen ontwikkeld routerplatform. De beschrijving ervan werd gepubliceerd in [7]. De resultaten werden gedeeltelijk gepubliceerd in [8]. Het werk rond de extra capaciteitsvereisten werd gepubliceerd in de reeds vermelde publicaties [4, 5, 6].

De daaropvolgende sectie behandelt de recente vooruitgang geboekt in foutconvergerende technieken. Daarbij kijken we vooral naar de vooruitgang met betrekking tot linkstatusrouteringsprotocollen en hoe deze technieken kunnen worden toegepast op FTCR. Daarnaast worden ook technieken voorgesteld die de convergentie van FTCR kunnen versnellen.

De laatste sectie vat de belangrijkste conclusies samen en geeft enkele slotbemerkingen.

I.3 MPLS en zijn toepassingen

I.3.1 Inleiding

MPLS speelt een belangrijke rol in dit werk omdat het toelaat nieuwe foutconvergerende technieken te implementeren in IP netwerken. Om deze technieken uit te leggen is het nodig MPLS voldoende toe te lichten.

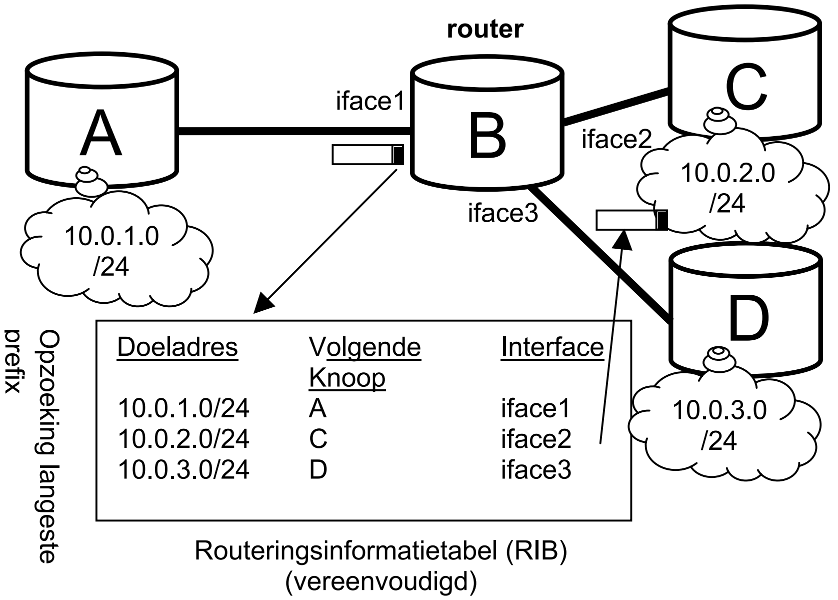
We lichten eerst de belangrijkste concepten met betrekking tot het versturen van pakketten in IP en MPLS netwerken uit. Vervolgens bekijken we naast de meer geavanceerde onderwerpen rond het versturen van pakketten in MPLS ook de manier waarop de paden worden opgezet (signalering). Een Engelstalige, meer uitgebreide, versie van de tekst in deze sectie werd reeds gepubliceerd in [2].

I.3.2 Het versturen van pakketten in IP en MPLS netwerken

Het Internet Protocol (IP) dat de basis vormt van het Internet is een connectieloos protocol [9, 10, 11, 12]. Dit wil zeggen dat elke knoop (*router*) in het netwerk de hoofding (*header*) van elk pakket onderzoekt alvorens die verder te sturen over het correcte pad. Deze beslissing wordt onafhankelijk genomen door elke router aan de hand van de routeringstabel (*routing table*) met als gevolg een knoop per knoop versturing van de pakketten in het netwerk. De routeringstabel bevat informatie over de volgende knoop en de uitgaande interface voor elk bestemmingsprefix.

In MPLS netwerken omzeilt men het verzenden van pakketten knoop per knoop, zoals in IP netwerken, door gebruik te maken van labelgeschakelde paden (LSP, *Label Switched Path)*. Hierbij wordt elk pakket voorzien van een label. Het oorspronkelijk doel van MPLS was om het verzenden van pakketten te versnellen [13, 14, 15]. Tegenwoordig gebeurt het verzenden van pakketten in IP netwerken even snel als in MPLS netwerken dus dit initiële voordeel van MPLS is niet meer geldig. Toch biedt MPLS nog steeds een aantal voordelen ten opzichte van IP. Ten eerste laat MPLS beter toe het netwerkverkeer te optimaliseren binnen het netwerk [16, 17]. Het aanbieden van een netwerk met private communicatie over het publieke Internet een zogenaamd virtueel privaat netwerk (VPN, *Virtual Private Network*) wordt door MPLS vereenvoudigd [18, 19]. Dankzij MPLS is het vaak ook mogelijk om meerdere protocol-lagen te elimineren en te vervangen door MPLS [20, 21, 22]. Daarnaast zijn er natuurlijk de foutconvergerende technieken die MPLS biedt [3].

In MPLS worden de paden opgezet door informatie in verband met labels op te slaan in elke router van het pad. Deze labels zijn entiteiten met vaste lengte die enkel lokaal betekenis hebben. In MPLS worden deze labels gebruikt om pakketten te versturen. Laten we nu even het versturen van pakketten in IP netwerken en MPLS netwerken vergelijken.

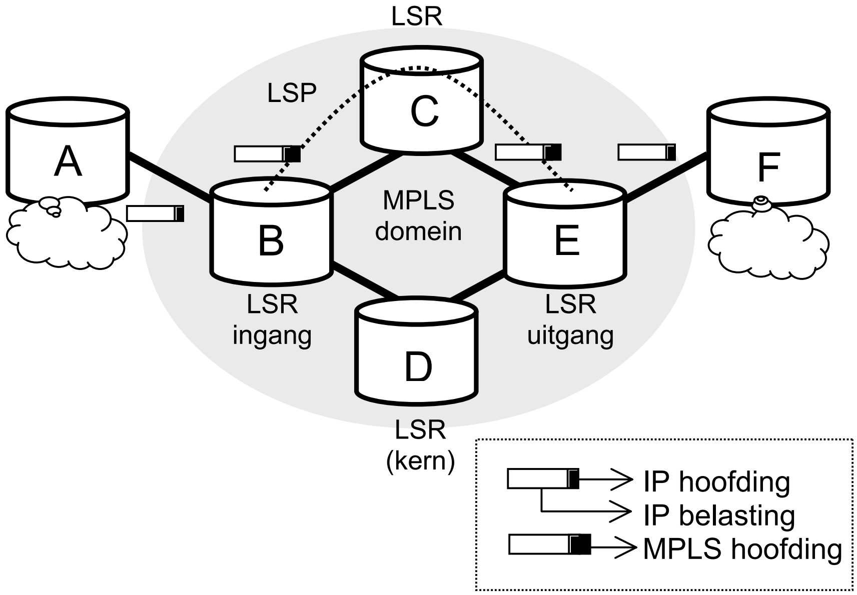

router
A
B
C
D
iface1
iface2
iface3
10.0.1.0 /24
10.0.2.0 /24
10.0.3.0 /24
Opzoeking langeste prefix
Routeringsinformatietabel (RIB) (vereenvoudigd)

Doeladres	Volgende Knoop	Interface
10.0.1.0/24	A	iface1
10.0.2.0/24	C	iface2
10.0.3.0/24	D	iface3

Figuur 1: Deze figuur illustreert hoe een pakket afkomstig van knoop A door knoop B verder wordt gestuurd naar knoop D. Hiervoor gebruikt knoop B de informatie die hij vindt in zijn routeringsinformatietabel.

In IP netwerken worden pakketten knoop per knoop verstuurd. Dit wil zeggen dat de beslissingen die moeten worden genomen tijdens het versturen van een pakket door elke knoop op zich wordt genomen. Hiervoor wordt gebruik gemaakt van de routeringsinformatietabel (RIB, *Routing Information Base*). Figuur 1 illustreert de routeringsinformatietabel van knoop B in een klein netwerk bestaande uit vier knopen: A, B, C en D. Wanneer knoop B een pakket ontvangt zal hij de uitgaande interface en het adres van de volgende knoop opzoeken in de routeringsinformatietabel. Om het aantal toegangen in de routeringsinformatietabel te reduceren worden deze toegangen geaggregeerd aan de hand van hun doeladres. Dit is mogelijk door niet de volledige adressen op te slaan maar enkel het meest significante gedeelte gevolgd door de lengte ervan (0 tot 32 bits). Een dergelijk partieel adres wordt een prefix genoemd en genoteerd als *a/n*, waarbij *a* het adres is en *n* de lengte. Tijdens het opzoeken van een doeladres wordt enkel het meest significante deel van het adres vergeleken met de toegangen in de routersinformatietabel. Daarbij wordt de toegang gekozen die het meest specifiek is, de toegang dus met de langste prefix die overeenkomt met het doeladres (*longest prefix match*). Een dergelijke opzoeking die rekening houdt met de langste prefix is complexer dan een opzoeking die gebruik maakt van adressen met vaste lengte [12].

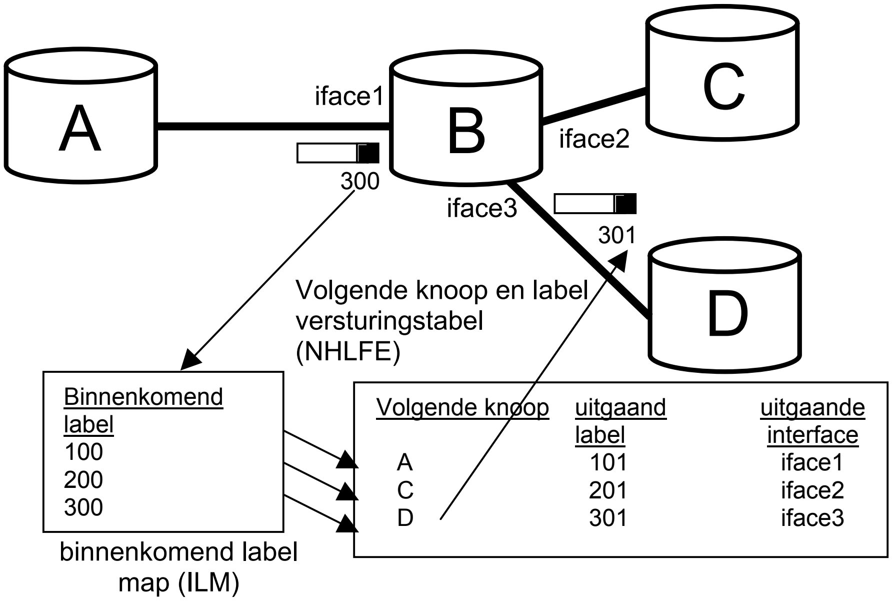
Een belangrijke eigenschap van het versturen van pakketten in IP netwerken is dat elk pakket met eenzelfde doeladresprefix op eenzelfde manier wordt verzonden. Een verzameling van pakketten die op dezelfde manier kan worden verstuurd wordt een klasse van gelijkaardig verstuurbare pakketten genoemd (FEC, *Forwarding Equivalence Class*). Doordat het versturen van pakketten in IP netwerken gewoonlijk gebaseerd is op het doeladres zal een FEC meestal overeenkomen met een doeladresprefix. Het is daarbij belangrijk op te merken dat een pakket in elke knoop opnieuw in de juiste FEC moet worden ingedeeld om de volgende knoop en de juiste uitgaande interface te vinden.



Figuur 2: Een MPLS netwerk bestaande uit de vier knopen B, C, D en E. De knopen A en F behoren niet tot het MPLS domein. Een LSP is opgezet vanuit B (de ingang) langs C (kern) naar E (uitgang). Over de LSP worden de pakketten met behulp van labels verzonden.

Het versturen van pakketten in MPLS netwerken is niet gebaseerd op doeladresprefixen maar op labels [23]. Deze labels hebben een vaste lengte en zijn slechts significant binnen één knoop. Daar deze labels enkel binnen een knoop betekenis hebben moeten ze in iedere knoop worden veranderd, dit wordt labelschakeling genoemd (*label switching)*. Binnen een MPLS netwerk worden de labels verdeeld door een labeldistributieprotocol (*label distribution protocol)*. Routers in een MPLS domein worden labelschakelende routers genoemd (LSR, *Label Switching Routers,* zie Figuur 2). Een aaneenschakeling van labels wordt een labelgeschakeld pad genoemd (LSP, *Label Switched Path*). Een LSP wordt opgezet vanuit de ingang door de kern van het netwerk naar de uitgang.

Pakketten die tot een bepaalde FEC behoren worden op een LSP gemapt. Het bepalen van de FEC van een pakket moet enkel gebeuren aan de ingang van de LSP. Dit in tegenstelling tot in IP netwerken waar dit in elke knoop moet gebeuren. Dit laat een flexibele mapping van een FEC op een LSP toe. De mapping kan bijvoorbeeld gebaseerd zijn op zowel het doeladres als het bronadres van het pakket. Merk op dat een LSP een unidirectioneel pad is.


A
B
C
D
iface1
iface2
iface3
300
301
Volgende knoop en label versturingstabel (NHLFE)
binnenkomend label map (ILM)

Binnenkomend label
100
200
300

Volgende knoop	uitgaand label	uitgaande interface
A	101	iface1
C	201	iface2
D	301	iface3

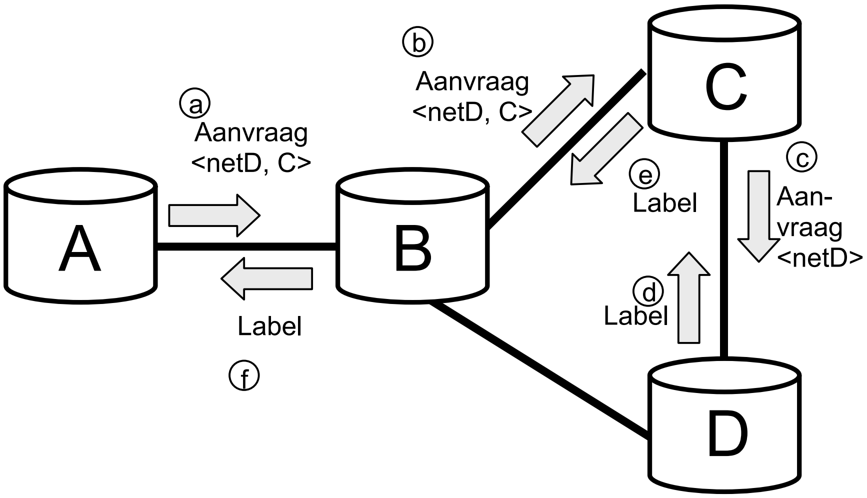
Figuur 3: Labelschakeling in MPLS netwerken. Het binnenkomend label wordt opgezocht in de ILM. Vanuit de toegang in de ILM wordt een toegang gevonden in de NHLFE. De NHLFE toegang bevat de nodige informatie om het pakket verder te sturen met inbegrip van het uitgaand label.

De labelschakeling is gebaseerd op twee tabellen (zie Figuur 2): een eerste met de binnenkomende labels (ILM, *Incoming Label Map*) en een tweede tabel met de volgende knopen en de uitgaande labels (NHLFE, *Next hop label forwarding entry*). De NHFLE bevat de nodige informatie om een pakket verder te sturen

[24]. Door de ILM en de NHFLE te verbinden kan men aan labelschakeling doen. Aan de ingang van de LSP moet een FEC worden gemapt op een NHLFE. Alle pakketten die tot die FEC behoren zullen dan over de LSP worden verstuurd. In de volgende sectie bekijken we hoe een LSP wordt opgezet.

I.3.3 Het opzetten van een LSP

Er bestaan twee soorten van LSPs: de knoop per knoop gerouteerde LSPs en de expliciet gerouteerde LSPs. Een knoop per knoop gerouteerde LSP wordt opgezet volgens het kortste pad vanaf de ingang van de LSP naar de uitgang van de LSP. Dit kortste pad wordt bepaald door de IP routering. Een expliciet gerouteerde LSP is een pad dat geheel of gedeeltelijk is gespecificeerd wanneer het wordt opgezet. Zo'n LSP kan dus afwijken van het kortste pad van de ingang naar de uitgang van de LSP.



Figuur 4: Het opzetten van een expliciet gerouteerde LSP vanuit knoop A naar knoop D via knoop C. Knoop A kan het pad van de LSP geheel of gedeeltelijk bepalen door knopen mee te geven met de aanvraag. Deze knopen zullen dan worden doorlopen bij het verwerken van de aanvraag. De labels worden bepaald in de omgekeerde richting als de aanvraag, Elke knoop bepaalt het label dat hij wil ontvangen. Daar de ingang geen label ontvangt zal hij ook geen label kiezen.

Een LSP wordt opgezet met een labeldistributieprotocol. Voorbeelden van dergelijke protocollen zijn LDP (*Label Distribution Protocol*) [25, 26, 27], CR-LDP (C*onstraint-based routed LDP*) [28, 29] en RSVP-TE (*ReSource reserVation Protocol for Traffic Engineering*) [30, 31, 32, 33]. Enkel de twee laatst vermelde protocollen ondersteunen expliciet gerouteerde LSPs. Het grootste verschil tussen CR-LDP en RSVP-TE is dat de informatie met betrekking tot de LSPs CR-LDP permanent is waar deze in RSVP-TE slechts van tijdelijke aard is zodat ze regelmatig moet worden vernieuwd.

Het opzetten van een LSP gebeurt in twee fasen (zie Figuur 4). In de eerste fase wordt een aanvraag verstuurd vanuit de ingang van de LSP naar de uitgang. Daarna zal de uitgang een antwoord versturen naar de ingang. De labels worden meegegeven in het antwoord. Het is dus zo dat de uitgang het eerst een label kiest en dat vervolgens opstuurt. Elke knoop tussen de uitgang en de ingang ontvangt een label en verstuurt vervolgens een label dat hij zelf kiest. De ingang van de LSP zal enkel een label ontvangen maar kiest er zelf geen.

Het pad van een LSP kan gestuurd worden door bij de aanvraag één of meerdere knopen mee te geven die de LSP moet doorlopen. Indien er geen knopen worden meegegeven dan verkrijgt men een knoop per knoop gerouteerde LSP. Figuur 4 toont hoe de expliciet gerouteerde LSP van A naar D via C wordt opgezet. Merk op dat alle knopen de labels die ze versturen en ontvangen opslaan in respectievelijk de NHLFE en ILM tabellen. Bovendien zal er in knoop A een mapping van een FEC naar de LSP nodig zijn om trafiek op de LSP te mappen.

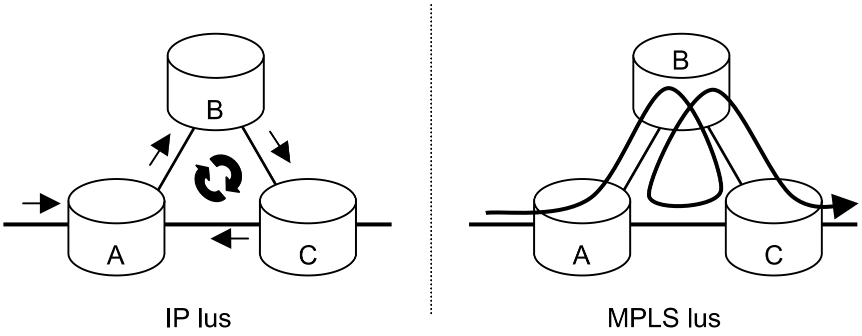
Zowel CR-LDP als RSVP-TE laten toe om meer informatie mee te geven bij het opzetten van een LSP. Het is daarbij mogelijk de gewenste bandbreedte van de LSP kenbaar te maken. Als ook de prioriteit waarmee de LSP moet opgezet worden of waarmee hij eventueel afgebroken kan worden. Het is ook mogelijk een LSP een bepaalde kleur te geven. Het is dan mogelijk om bijvoorbeeld te eisen dat een LSP enkel door het gedeelte van het netwerk loopt dat gelijkaardig gekleurd is. Deze bijkomende functionaliteit van CR-LDP en RSVP-TE is belangrijk om de trafiek zo optimaal mogelijk over het netwerk te verdelen (*traffic engineering*) [16, 17].

In de volgende secties gaan we nog enkele facetten van MPLS nader toelichten. We beginnen eerst met de beschrijving van de verschillen tussen lussen in IP en MPLS netwerken. Vervolgens lichten we toe hoe het gebruikte aantal labels kan worden verminderd.

I.3.4 Lussen

Zoals we later zullen zien kunnen lussen in zowel IP als in MPLS netwerken voorkomen. In MPLS is het mogelijk om te vermijden dat een LSP met een lus wordt opgezet [34, 35]. Er is echter een groot verschil tussen een lus in een IP netwerk en een MPLS netwerk (zie Figuur 5).

Vermits het versturen van een pakket in een IP netwerk gebeurt op basis van het doeladres is het onmogelijk om een lus te verlaten eenmaal men daarin is verzeild. Men ziet op de figuur dat knoop A elk pakket verder stuurt naar C onafhankelijk of dit pakket A bereikt via de lus of niet. In de praktijk zal het pakket dat de lus bereikt een aantal maal de lus doorlopen om tenslotte te worden verwijderd.



Figuur 5: Lussen in IP en MPLS netwerken verschillen danig.

In MPLS is het mogelijk dat een LSP wordt opgezet die een lus bevat. Dit hoeft echter niet zulke drastische gevolgen te hebben als in IP netwerken. In het voorbeeld doorloopt de LSP het segment A-B-C eenmaal om vervolgens verder te lopen. Dit is mogelijk omdat het versturen van pakketten in MPLS netwerken gebaseerd is op labels. Het is duidelijk dat de negatieve gevolgen van een dergelijke lus veel kleiner zijn dan bij een lus in een IP netwerk.

I.3.5 Het aantal gebruikte labels verminderen

In deze sectie bekijken we hoe het aantal labels dat binnen een netwerk of zelfs binnen een bepaalde knoop wordt gebruikt kan worden verminderd. We zullen drie mechanismen bespreken. De eerste methode bestaat erin labels samen te voegen, de tweede methode tunnelt een aantal LSPs in elkaar en tenslotte bespreken we ook hoe de voorlaatste hop geen label hoeft te gebruiken. Maar eerst vermelden we dat een label een bepaald bereik heeft.

Het bereik van labels
Het bereik van een label kan beperkt worden tot een bepaalde interface (per interface label, *per interface label space*) of de label kan globaal voor heel de router gelden (*platform wide label space*) [24]. Indien de labels over de gehele router gelden dan heeft dit het voordeel dat een inkomend label zal herkend worden gelijk over welke interface het binnenkomt. Dit kan handig zijn wanneer het pad van een LSP verandert. Het nadeel is dat er minder labels beschikbaar zijn omdat de waarden niet worden opgesplitst per interface.

Het samenvoegen van labels

Om het aantal labels dat wordt gebruikt binnen een router te verminderen is het soms mogelijk een aantal binnenkomende labels samen te nemen en slechts één uitgaand label te gebruiken. Dit is enkel mogelijk als de FEC van de verschillende LSPs dezelfde zijn. In de praktijk betekent dit dat de LSPs dezelfde bestemming en pad naar bestemming moeten hebben. Merk op dat er geen onderscheid meer mogelijk is tussen de verschillende LSPs die samengevoegd zijn.

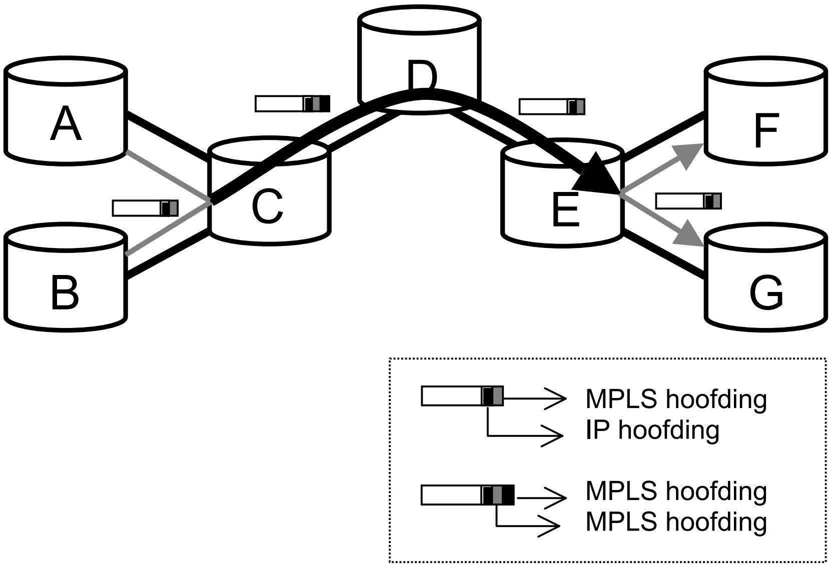
Voorlaatste knoop ploffing

Zoals we reeds hebben besproken, worden in MPLS labels gebruikt om te bepalen hoe een pakket verder moet worden gestuurd. Het is echter niet nodig om een label te gebruiken tussen de voorlaatste en de laatste knoop. De voorlaatste knoop neemt deze beslissing aan de hand van het label dat hij heeft ontvangen en de laatste knoop moet enkel het pakket ontvangen en niet meer verder sturen. Indien er geen label gebruikt wordt tussen de voorlaatste en de laatste knoop dan noemen we dit voorlaatste knoop ploffing (*penultimate hop popping*).

Het tunnellen van LSPs

We hebben reeds gezien dat het aantal labels verminderd kan worden door verscheidene labels samen te nemen. Een andere methode is het tunnellen van verschillende LSPs in een andere LSP (zie Figuur 6).

Het is belangrijk om even stil te staan bij de manier waarop een pakket wordt verstuurd over een dergelijke getunnelde LSP. We nemen als voorbeeld het LSP B-C-D-E-G. Een pakket over die LSP wordt natuurlijk verzonden in knoop B. Knoop B voegt een MPLS hoofding toe die het label bevat dat knoop C hem gezonden heeft. Daarna stuurt knoop B het pakket verder. Wanneer knoop C dit pakket ontvangt dan zal hij het binnenkomende label vervangen door het label dat hij heeft ontvangen van *knoop E*. Verder zal hij ook een extra hoofding toevoegen die het label bevat dat hij heeft ontvangen van *knoop D*. Op dit moment bevat het pakket dus twee labels. Dit wordt labelstapeling genoemd (*label stacking*). Vervolgens zendt knoop C dit pakket verder. Merk op dat knoop D de voorlaatste knoop is van de LSP C-D-E. Om LSPs te tunnellen is het nodig dat de LSRs werken met voorlaatste knoop ploffing. Knoop D zal dus het label dat hij naar knoop C heeft gestuurd verwijderen (merk op dat dit het bovenste label is) en het pakket verder sturen. Op dit moment zal het pakket slechts één label bevatten namelijk het label dat knoop C heeft ontvangen van knoop E. Knoop E herkent natuurlijk dit label en zal het vervangen door het label dat hij heeft ontvangen van knoop G.



Figuur 6: Het tunnellen van LSPs. Deze figuur is illustreert hoe de LSPs A-C-D-E-F en B-C-D-E-G worden getunneld in de LSP C-D-E.

Zoals we zien vereist het tunnellen van LSPs dat we zowel gebruik maken van voorlaatste knoop ploffing en labelstapeling. Merk op dat in dit geval er slechts over één link gebruik wordt gemaakt van twee labels. Indien de LSP die wordt gebruikt om andere LSPs in te tunnellen langer is dan zal er ook langer gebruik moeten gemaakt worden van twee labels tegelijk.

In de volgende sectie bekijken we de belangrijkste foutconvergerende technieken voor IP en MPLS netwerken.

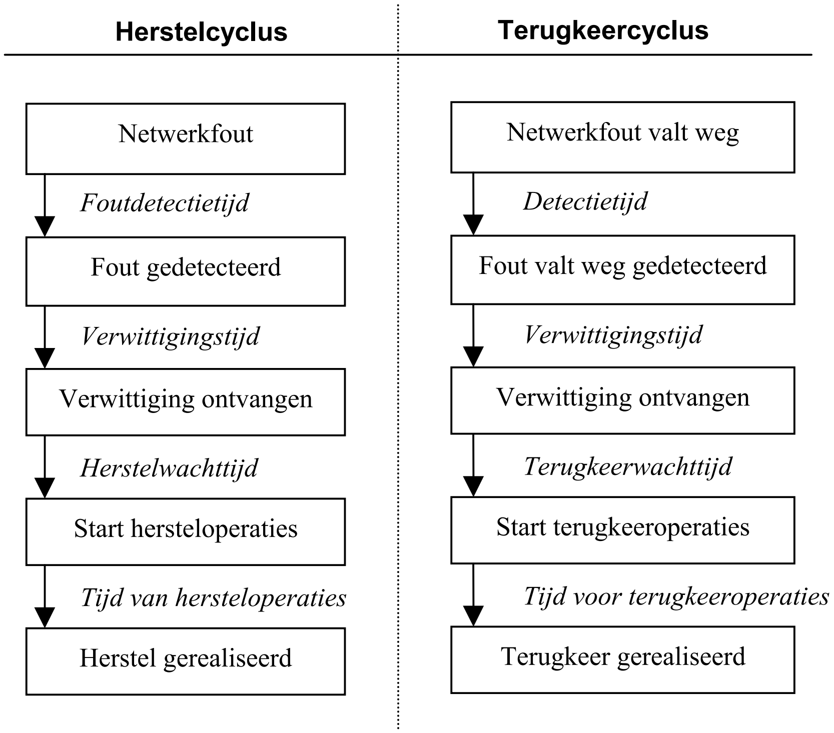
I.4 Foutconvergerende technieken

I.4.1 Inleiding

In deze sectie onderzoeken we drie belangrijke foutconvergerende technieken. Eerst onderzoeken we hoe fouten worden behandeld in IP netwerken. Daarvoor gebruiken we het Open Shortest Path First (OSPF) [36] protocol als voorbeeld. Dit wil zeggen dat we een ander veel gebruikt protocol Intermediate System-Intermediate System (IS-IS) [37, 38] niet zullen bespreken. De principes die we uitleggen voor OSPF zijn in principe overdraagbaar naar IS-IS. Naast OSPF en IS-IS wordt soms ook het Routing Information Protocol (RIP) [39, 40, 41] gebruikt binnen een netwerk. RIP convergeert echter zeer traag na het optreden van een fout. RIP wordt minder en minder gebruikt en is zeker niet toepasbaar in grotere netwerken. Vandaar dat we RIP niet verder beschouwen. Naast de routeringsprotocollen die binnen één netwerk worden gebruikt is er Border Gateway Protocol (BGP) [42] dat tussen de verschillende autonome systemen wordt gebruikt. Ook BGP beschouwen we niet verder.

Naast de foutconvergerende technieken voor IP netwerken gaan we ook kijken naar dergelijke technieken in MPLS netwerken. De eerste categorie is het herrouteren in MPLS, daarnaast kijken we ook naar protectie in MPLS.

Zowel herroutering als protectie doorlopen beide dezelfde fasen wanneer ze een fout herstellen. Tijdens de eerste fase wordt de fout vastgesteld. Nadat de fout is ontdekt moeten de relevante partijen worden geïnformeerd van dit feit. Het kan zijn dat de fout enkel lokaal gemeld moet worden maar het kan ook zijn dat alle knopen in het netwerk verwittigd moeten worden. Daarna kan er eventueel een tijdje worden gewacht voor de herstelacties worden gestart. Dit kan nodig zijn om zich ervan te vergewissen dat de fout persistent is of om de stabiliteit te verhogen. Vervolgens worden de hersteloperaties uitgevoerd. Deze herstelcyclus is geïllustreerd in [43].



Figuur 7: De herstelcyclus en de terugkeercyclus

Naast een herstelcyclus bestaat er ook een terugkeerfase. Deze fase wordt niet gekenmerkt door een nieuwe fout maar door het feit dat een fout niet langer geldt. Wanneer een fout wegvalt, zal dit opnieuw gemeld moeten worden. Wederom wordt er meestal gewacht voor de hersteloperaties worden aangevangen analoog aan het wachten in de herstelcyclus. In de terugkeercyclus is het nog belangrijker om lang genoeg te wachten zodat het netwerk hersteld is, indien er niet lang genoeg gewacht wordt zal er onnodig trafiek verloren gaan.

Zoals we hebben gezien begint de herstelcyclus met een netwerkfout. In de volgende sectie bekijken we hoe netwerkfouten kunnen worden ontdekt.

I.4.2 Foutdetectie en foutverwittiging

Hardware gebaseerde foutdetectie
De snelste manier om een fout te ontdekken is wanneer de gebruikte hardware dit ondersteunt. Bijvoorbeeld een Pakket over Sonet (POS) interface kan detecteren wanneer de drager (de link) tussen twee kaarten verbroken wordt. Deze detectie gebeurt zeer snel (in de orde van 10ms) maar ze ondersteunt niet alle foutoorzaken. Wanneer het signalisatiegedeelte van een knoop faalt zal dit niet worden ontdekt. Daarnaast zijn er ook interface types die geen hardware gebaseerde foutdetectie ondersteunen. Er zijn dus complementaire technieken nodig.

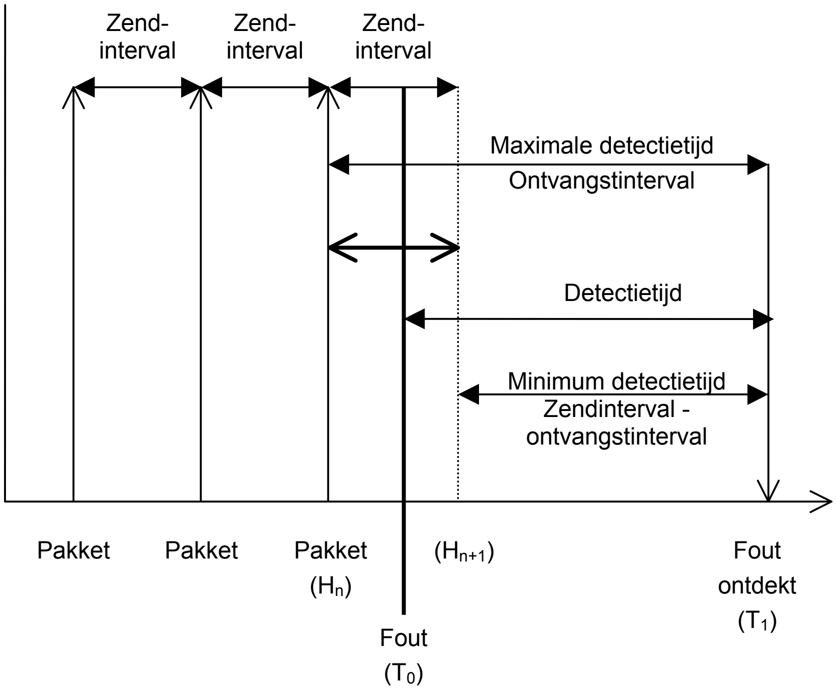
Algemeen hartslag mechanisme
In deze sectie beschrijven we een techniek die algemeen wordt gebruikt om fouten te detecteren. De techniek is unidirectioneel omdat er één zender en één ontvanger is.

De zender en de ontvanger spreken af dat de zender elk *zendinterval* een pakket stuurt naar de ontvanger. Indien de ontvanger geen pakket heeft ontvangen binnen een *ontvangstinterval* zal deze concluderen dat er een fout is opgetreden. Het is dus belangrijk dat het ontvangstinterval groter is dan het zendinterval om te vermijden dat er valse positieven optreden. Bijvoorbeeld in het OSPF protocol heet het zendinterval het *Hello Interval* en het ontvangstinterval het *Routerdead Interval* en hun waarden zijn respectievelijk 10s en 40s [36].

De detectietijd, namelijk de tijd tussen het optreden van de fout (T_0) en het detecteren van de fout (T_1), is afhankelijk van het grootte van het zendinterval en het ontvangstinterval (zie Figuur 8). Indien de fout onmiddellijk na het verzenden van een hartslagpakket (H_n) optreedt dan bedraagt de detectietijd bijna een volledig ontvangstinterval. Indien de fout optreedt juist voordat het volgende hartslagpakket moet worden verstuurd dan is detectietijd bijna een volledig zendinterval kleiner.

$$\text{ontangstinterval} - \text{zendinterval} \leq T_{detect} = T_1\text{-}T_0 \leq \text{ontvangstinterval} \qquad (1)$$

We zien duidelijk in formule (1) dat de detectietijd afhangt van de ontvangstinterval. Om de detectie te versnellen kunnen we dus het ontvangstinterval verkleinen. Indien we echter het ontvangstinterval verkleinen moeten we ook het zendinterval verkleinen om te vermijden dat er vals positieve detecties optreden.. In de praktijk heeft het ontvangstinterval een ondergrens omdat de netwerkbelasting en de verwerkinglast van de pakketten anders te groot wordt.



Figuur 8: De detectietijd van het algemeen hartslag mechanisme hangt af van wanneer de fout optreedt ten opzichte van wanneer het laatste pakket is verzonden.

Dit algemeen hartslagmechanisme wordt gebruikt in zowel OSPF, IS-IS en LDP.

De bandbreedte verminderen

Zoals we reeds hebben gezien is het moeilijk om het ontvangstinterval zeer klein te nemen omdat het zendinterval dan ook kleiner moet worden wat dan weer als gevolg heeft dat de gebruikte bandbreedte toeneemt. Het is echter ook mogelijk om het ontvangstinterval te verkleinen door gelijk welk pakket impliciet als een hartslagpakket te beschouwen. Dit heeft als voordeel dat de detectie sneller kan gebeuren zonder de bandbreedte te verhogen maar heeft als nadeel dat niet alle fouten worden ontdekt. Stel bijvoorbeeld dat de controlelaag van een knoop faalt maar niet het onderdeel dat binnenkomende pakketten verder stuurt. Een dergelijke fout zal niet worden ontdekt indien de falende knoop voldoende pakketten verder zendt.

Voor meer informatie over foutdetectie verwijzen we naar ons onderzoek in [44] Naast de beschrijving van foutdetectie met behulp van impliciete hartslagpakketten geeft dit werk ook experiementeel bepaalde foutdetectie- en convergentietijden voor Linux gebaseerde netwerken.

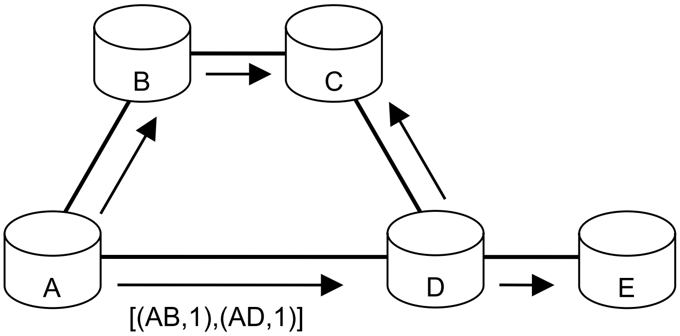
Foutverwittiging
Soms is het noodzakelijk om andere knopen te informeren over een fout die lokaal wordt gedetecteerd. Een dergelijk foutverwittigingsmechanisme kan onderdeel zijn van de convergentietechniek of het kan een apart mechanisme zijn. We gaan niet verder in op foutverwittiging maar we veronderstellen dat steeds een adequaat mechanisme wordt gebruikt.

In de volgende secties beschouwen we de belangrijkste foutconvergerende technieken voor IP en MPLS netwerken. We beginnen met OSPF.

I.4.3 Open Shortest Path First

Het Open Shortest Path First (OSPF) routeringsprotocol behoort samen met IS-IS tot de familie van de linkstatusrouteringsprotocollen [12].

In de opstartfase beginnen de routers in het netwerk met het verspreiden van linkstatuspakketten in het netwerk (*Link Status Advertisement, LSA*). Een linkstatuspakket bevat informatie over elke interface van de router samen met de kost die aan deze interface is toegewezen. Figuur 9 toont dit proces voor router A.



Figuur 9: Het verspreiden van de lokale informatie van router A met behulp van een linkstatuspakket doorheen het netwerk.

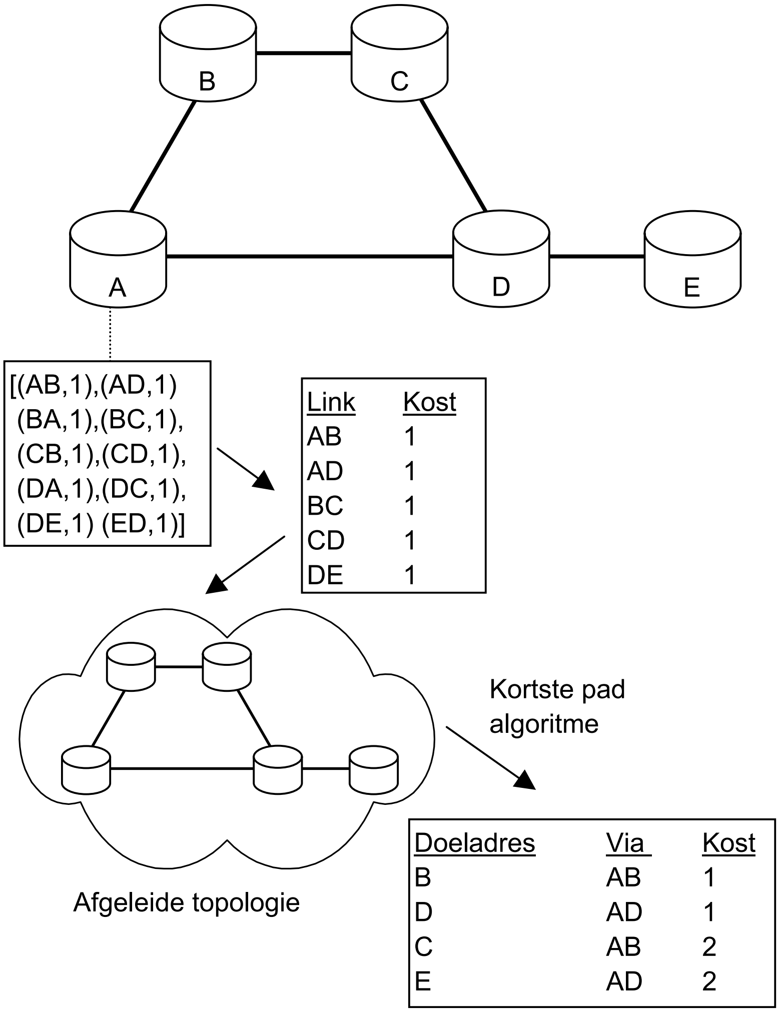
Elke router in het netwerk zal hetzelfde doen en tegelijk er ook voor zorgen dat de linkstatuspakketten van de andere routers doorheen het netwerk vloeien (*flooding*). Wanneer een router een linkstatuspakket ontvangt dat hij nog niet heeft gezien zal hij een kopie ervan opslaan in zijn *linkstatusdatabank* (*link state database*) alvorens het verder te sturen.

Nadat alle routes in het netwerk hun lokale informatie hebben verspreid en de linkstatuspakketten van alle andere routers hebben ontvangen is het mogelijk om uit deze informatie de topologie van het netwerk te construeren. Vervolgens kan men deze topologie gebruiken om de kortste paden te berekenen binnen het netwerk aan de hand van het Dijkstra kortste pad algoritme (*Shortest Path First, SPF*) [45]. Nadat de kortste paden bekend zijn kan deze informatie worden gebruikt om de volgende knoop voor elk doeladres in de routeringstabel in te vullen (Figuur 10). Merk op dat elke router dit afzonderlijk doet voor elk doeladres in het netwerk. Het is daarom van kritiek belang dat iedere router hetzelfde beeld heeft van de topologie van het netwerk.

We hebben reeds gezien dat een variant van het algemeen hartslagmechanisme in OSPF wordt gebruikt om fouten te detecteren. Stel nu dat een fout optreedt, de router die deze fout detecteert zal een nieuw linkstatuspakket verzenden waarin de kost van de falende link op oneindig wordt gezet. Dit is het signaal voor de andere routers om deze link niet langer te gebruiken. De volgende kortste pad berekening zal deze link dan ook niet meer gebruiken. Dit heeft als gevolg dat ook in de routeringstabellen de falende link niet meer wordt gebruikt en dat de trafiek niet langer over de fout loopt.

Het berekenen van de kortste paden in het netwerk is vrij belastend. Daarom worden de kortste pad berekeningen beperkt. Bijvoorbeeld in de Zebra implementatie van OSPF [46] wordt er voor elke kortste pad berekening een aantal seconden gewacht (de SPF vertraging, *SPF delay*). Bovendien ligt de minimale tijd tussen twee opeenvolgende berekeningen nog hoger (de SPF wachttijd, *SPF hold-down*). Deze twee timers worden samen de wachttijden van OSPF genoemd.

Merk op dat de nieuwe linkstatuspakketten steeds worden opgenomen in de linkstatusdatabank maar dat de berekening van de kortste paden even wordt uitgesteld om zo veel mogelijk linkstatuspakketten te betrekken in elke kortste pad berekening.



[(AB,1),(AD,1)
(BA,1),(BC,1),
(CB,1),(CD,1),
(DA,1),(DC,1),
(DE,1) (ED,1)]

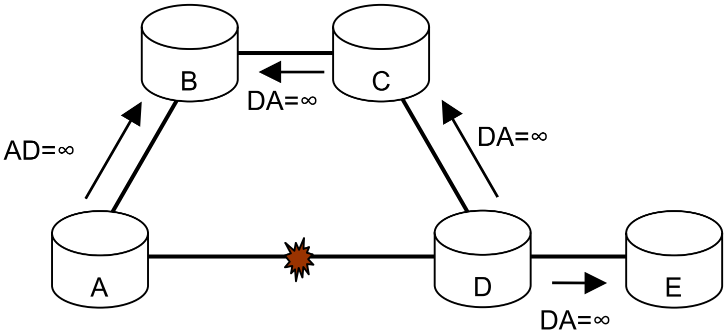
Link	Kost
AB	1
AD	1
BC	1
CD	1
DE	1

Doeladres	Via	Kost
B	AB	1
D	AD	1
C	AB	2
E	AD	2

Figuur 10: In OSPF worden de routeringstabellen berekend door het kortste pad algoritme toe te passen op de linkstatusdatabank.

Het is belangrijk op te merken dat een fout meestal wordt ontdekt door meer dan één router. Bijvoorbeeld wanneer de link AD faalt zal zowel knoop A als knoop D deze fout ontdekken (zie Figuur 11). Als gevolg daarvan zal knoop B zowel een linkstatuspakket van knoop A als van knoop D ontvangen. Indien knoop B even wacht voor hij de kortste pad berekening doet wordt er vermeden dat beide pakketten elk voor een kortste pad berekening zorgen.

Knoopfouten worden ontdekt doordat elke router die verbonden is met de koop merkt dat zijn link met die knoop faalt. Elk van deze routers zal een linkstatuspakket verzenden met de kost naar de falende router gelijk aan oneindig. Dit zal leiden tot de situatie waarin geen enkele router een link gebruikt van de falende router waardoor de router niet langer wordt gebruikt.



Figuur 11: Een fout wordt meestal door meer dan één router gedetecteerd, het is daarom van belang het ontvangen van een linkstatuspakket niet direct te laten volgen door een kortste pad berekening omdat dit leidt tot overbodige berekeningen.

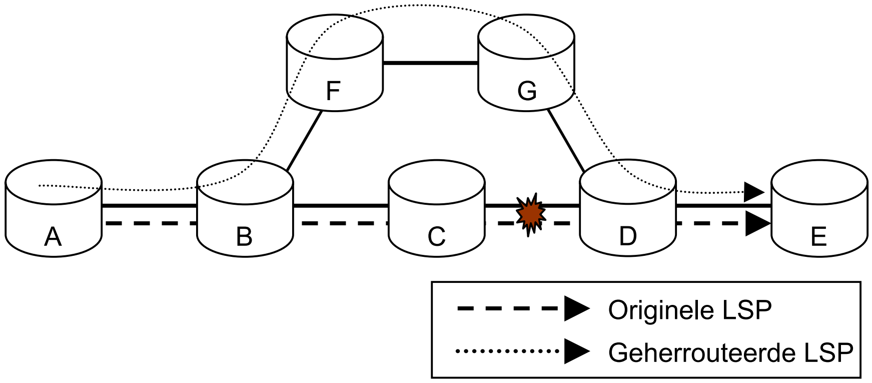
Tot zover hebben we ons toegespitst op de herstelcyclus van OSPF. De terugkeercyclus is vergelijkbaar met de herstelcyclus. De herstelcyclus begint nadat een fout hersteld is, als gevolg daarvan zullen de verbonden routers elkaar terugvinden en opnieuw OSPF pakketten beginnen uit te wisselen. Deze initiële uitwisseling loopt onder leiding van het Hello protocol. Daarna zal informatie over de nieuwe link opnieuw worden verspreid over het netwerk zodat de routers deze kunnen gebruiken in de volgende kortste pad berekening zodat de link tenslotte ook gebruikt zal worden in de nieuwe routeringstabellen.

I.4.4 MPLS herroutering

Herroutering in MPLS is gebaseerd op de herroutering in IP. Dit wil zeggen dat het IP routeringsprotocol verantwoordelijk is om nieuwe routeringstabellen te berekenen nadat een fout is opgetreden. Hoe dit gebeurt hebben we reeds geïllustreerd in de vorige sectie aan de hand van het OSPF protocol.

Het herrouteren van een LSP komt neer op het aanpassen van het pad van de LSP zodat dit pad terug overeenkomt met de routeringstabellen. Dit is geïllustreerd in Figuur 12 aan de hand van een kortste pad LSP van knoop A naar knoop E. We zien dat link CD faalt waardoor de LSP wordt aangetast. Als gevolg van de fout zal de IP routering nieuwe routeringstabellen installeren in het netwerk. Zo ook in knoop B waar de volgende knoop voor het doeladres van de LSP, knoop E, verandert van C naar F. Wanneer dit gebeurt zal B een nieuwe

LSP opzetten vanuit B naar E en vervolgens het oude gedeelte vervangen door het nieuwe.



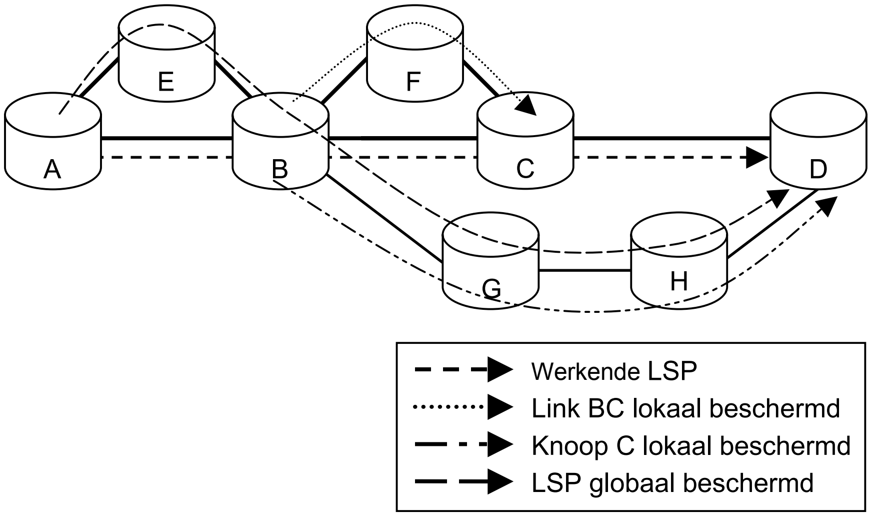
Figuur 12: Herroutering van de LSP A-B-C-D-E door knoop B na een fout op link CD

De aanzet van het herrouteren van de LSP wordt gegeven door een verandering van de volgende knoop ergens op het pad van de LSP. In LDP is het nodig om te kunnen detecteren dat een volgende knoop is veranderd om de LSP te kunnen herrouteren. In RSVP-TE ligt dit anders omdat de staat niet vast is en dus regelmatig moet worden ververst. Wanneer de volgende knoop is veranderd en de staat van de LSP wordt ververst dan zal deze automatisch worden geherrouteerd. De LSPs worden ververst binnen het interval [0.5,1.5]R waarbij R=30s de standaard waarde is [31]. Dit kan leiden tot zeer grote vertraging van de herroutering. Het is echter ook mogelijk zoals bij LDP actief de routeringstabellen te monitoren en de LSPs onmiddellijk te verversen indien deze wijzigen.

I.4.5 Protectie in MPLS

Het herrouteren van LSPs is sterk afhankelijk van de herroutering in IP. MPLS biedt echter een alternatief om trafiek binnen het netwerk te beschermen, protectie genaamd. In protectie wordt een herstel-LSP opgezet samen met de LSP die onder normale omstandigheden wordt gebruikt (de werkende LSP).

Indien protectie wordt gebruikt is het bijvoorbeeld mogelijk om de LSP A-B-C-D-E als werkende LSP te gebruiken en de LSP A-B-F-G-D-E uit Figuur 12 als herstel-LSP. Wanneer een fout optreedt op de link BC of link CD of in de knoop C dan kan er overgeschakeld worden van de werkende naar de herstel-LSP.



Figuur 13: Lokale link- en knoopprotectie en globale protectie.

We onderscheiden lokale en globale protectie. In lokale bescherming wordt de werkende LSP beschermd tegen het falen van een knoop of een link (zie Figuur 13). In globale protectie wordt een LSP over zijn gehele lengte zo goed mogelijk beschermd tegen alle mogelijke fouten.

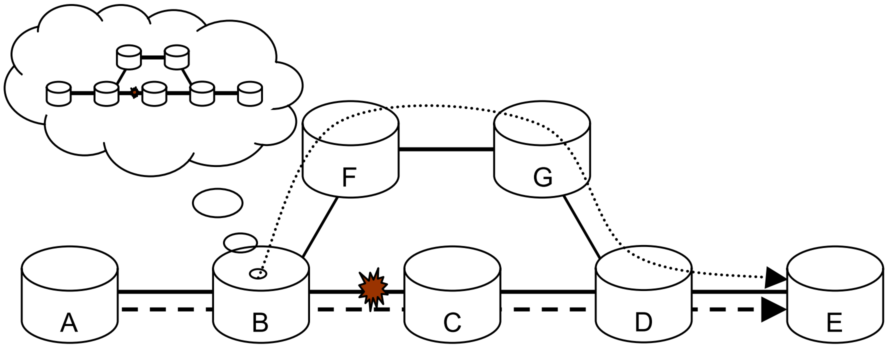
Het gebruik van lokale protectie en in zekere mate ook globale protectie kan leiden tot een zeer groot aantal herstel-LSPs. Het is daarom aan te raden om meerdere werkende LSPs te beschermen met behulp van één herstel-LSP indien mogelijk. Dit wordt gewoonlijk genoteerd als N:1 protectie, waarbij N het aantal werkende LSPs is. Het aantal herstel-LSPs kan ook worden verminderd door de technieken die werden voorgesteld in sectie I.3.5 toe te passen. Het is bijvoorbeeld mogelijk een aantal herstel-LSPs te tunnelen binnen een andere LSP. Een andere techniek bestaat erin een aantal herstel-LSPs samen te nemen op hun bestemming.

Hoewel protectie meestal gepaard gaat met off line berekende herstel-LSPs hoeft dit zeker niet altijd het geval te zijn. Tijdens het opstellen van de werkende LSP kan er worden opgegeven dat deze moet worden beschermd. De individuele routers op het pad kunnen dan tijdens het opzetten van de werkende LSP ook lokale herstel-LSPs opzetten [47].

I.5 FTCR

I.5.1 Basiswerking

In deze sectie beschrijven we Fast Topology based Constrained Rerouting (FTCR) een convergentietechniek ontwikkeld door de auteur [4, 5, 6, 8].



Figuur 14: Deze figuur illustreert hoe de linkstatusdatabank wordt gebruikt in FTCR om de herstel-LSP te berekenen.

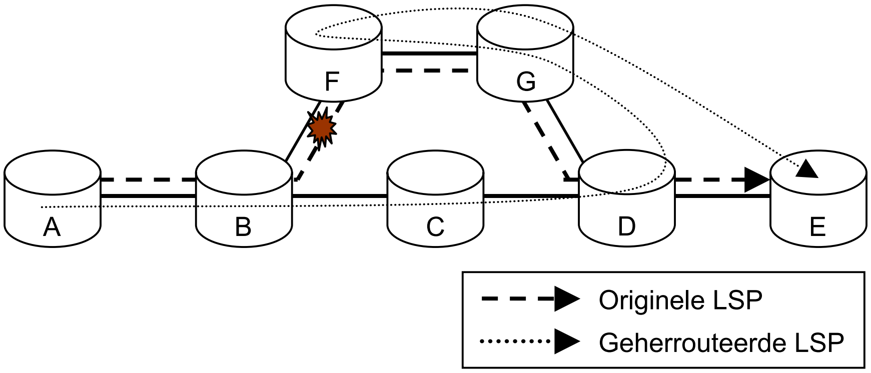
We zullen de werking van FTCR illustreren aan de hand van een voorbeeld (zie Figuur 14). We gebruiken dezelfde topologie en werkende LSP als in Figuur 12, deze keer faalt de link BC. Wanneer knoopt B merkt dat de link BC faalt kan hij een herstel-LSP berekenen aan de hand van zijn linkstatusdatabank. Zoals we hebben gezien in de sectie rond OSPF bevat de linkstatusdatabank voldoende informatie om de topologie van een netwerk op te bouwen. Door nu in deze topologie de falende link te verwijderen kan men met behulp van het kortste pad algoritme een herstel-LSP berekenen. Het opzetten van de herstel-LSP vormt wel een probleem omdat de routeringstabellen in het netwerk de fout nog niet in acht hebben genomen. Het is nochtans mogelijk de herstel-LSP op te zetten indien alle knopen langs de LSP expliciet worden meegegeven bij het opzetten ervan. Vermits elke knoop wordt meegegeven is het niet nodig dat de routeringstabellen accuraat zijn.

Het herrouteren van een LSP met behulp van FTCR zal sneller zijn dan gebruik te maken van MPLS herroutering omdat het opzetten van de herstel-LSP simultaan gebeurt met het hernieuwen van de routeringstabellen.

De knoop die de herstel-LSP opzet wordt de FTCR Switch LSR, de FSL, genoemd. De FSL kan zich onmiddellijk stroomopwaarts van de fout bevinden (lokale FSL) zoals reeds geïllustreerd, maar de FSL kan zich ook aan de ingang van de LSP bevinden (ingangs-FSL) of ergens tussenin. Een belangrijke geval van deze laatste vorm is wanneer de FSL daar wordt geplaatst zodat er geen lus op de herstel-LSP optreedt maar toch zo dicht mogelijk bij de fout (dichtstbijzijnde lusvrije FSL). Dit zijn de drie belangrijkste FSL-selectiemodes. We veronderstellen steeds, tenzij anders vermeld, dat de lokale FSL selectiemode wordt gebruikt.

I.5.2 Het herrouteren van expliciete LSPs

Het herrouteren van een expliciet gerouteerde LSP is moeilijker dan een kortste pad LSP vermits een deel of het gehele pad van de LSP vast ligt. Er zijn een aantal mogelijkheden om een expliciete LSP te herrouteren maar deze zullen het originele pad moeten wijzigen. Eén mogelijkheid bestaat erin het originele pad zoveel mogelijk te herbruiken. Dit is geïllustreerd in Figuur 15 waar de expliciete LSP A-B-F-G-D-E is geherrouteerd door een nieuw pad te zoeken vanuit de FSL naar de volgende hop in de LSP specificatie (F). Het is duidelijk dat dit pad verre van ideaal is. Men kan natuurlijk eerst de lus elimineren voor men de herstel-LSP opzet. Een andere mogelijkheid om de LSP te herrouteren bestaan erin het kortste pad te gebruiken van de FSL naar de uitgang van de LSP. Tenslotte kan men soms hetzelfde algoritme gebruiken dat de werkende LSP heeft berekend maar ditmaal op de nieuwe topologie.

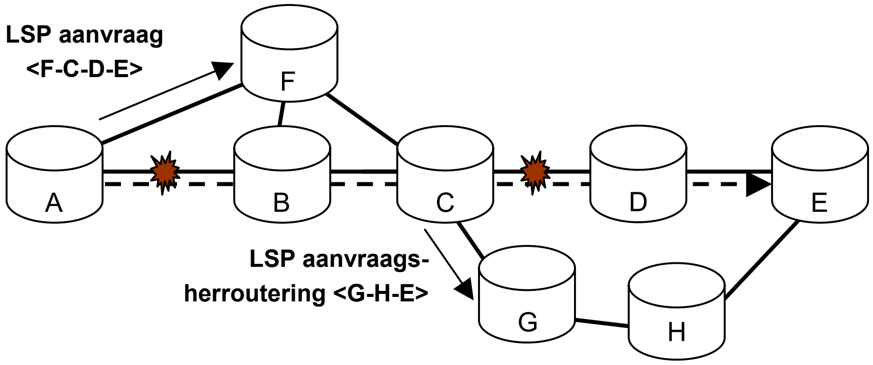


Figuur 15 : Herrouteren van de expliciete LSP A-B-F-G-D-E

I.5.3 Meerdere fouten herstellen

We hebben reeds behandeld hoe een kortste pad en expliciet gerouteerde LSPs kunnen worden hersteld met FTCR. In deze sectie beschouwen we hoe meerdere fouten op een LSP kunnen worden hersteld. Het is belangrijk om op te merken dat meerdere fouten op één LSP betekent dat er een fout optreedt en vervolgens nog minstens één fout vóór de IP herroutering heeft plaats gevonden.

Wanneer er een fout optreedt zal de FSL de werkende LSP herrouteren door een aanvraag voor een herstel-LSP te verzenden. Stel nu dat er een nieuwe fout optreedt, ditmaal op het pad van de herstel-LSP. Het is duidelijk dat de herstel-LSP niet meer geldig is. Er is dan een nieuwe FSL, de FSL verantwoordelijk voor de nieuwe fout. We gaan er even vanuit dat deze FSL direct stroomopwaarts van de fout ligt. Deze FSL kan de aanvraag van de eerste FSL gaan herrouteren en hiervoor een nieuwe herstel-LSP opzetten. Dit is geïllustreerd in Figuur 16. Eerst stuurt A een aanvraag voor de herstel-LSP na de fout op link AB, daarna zal C deze aanvraag herrouteren om de fout op link CD te vermijden.



Figuur 16: Het herrouteren van een LSP indien meer dan één fout optreedt.

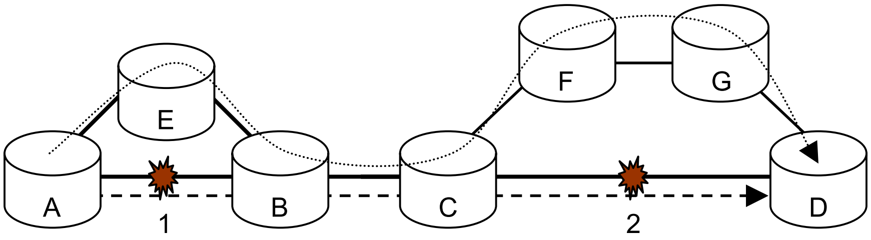
Als de ingang van de LSP steeds de FSL is dan zal knoop A tweemaal een herstel-LSP trachten op te zetten. De eerste keer zal dit mislopen omdat de link CD ook gefaald heeft, de tweede maal weet knoop A dat CD niet bruikbaar is en zal deze link dan ook vermijden.

Stel nu dat de fouten in omgekeerde volgorde optreden namelijk dat link CD faalt voor link AB, we beschouwen opnieuw de lokale FSL mode. In dit geval zal knoop C de werkende LSP herrouteren over het pad G-H-E. Op het moment dat link AB faalt zal knoop A een aanvraag voor de herstel-LSP F-C-D-E verzenden (merk op dat A niet weet dat link CD heeft gefaald). Opnieuw zal knoop C deze aanvraag herrouteren over het pad G-H-E. In beide gevallen is de herstel-LSP dezelfde, de volgorde van de fouten heeft dus geen invloed op het pad van de herstel-LSP.

Het herrouteren van meerdere fouten tegelijk werkt in het algemeen omdat er steeds één FSL verantwoordelijk is voor een bepaalde fout op een LSP. Deze FSL zal ofwel de werkende LSP herrouteren ofwel de LSP aanvragen die hij ontvangt en die over de fout lopen herrouteren.

I.5.4 Terugkeer in FTCR

De terugkeer cyclus in IP routering begint vanaf het moment dat een fout is niet meer van kracht is. Terugkeer in FTCR kan pas gebeuren nadat de IP herroutering is volbracht. Nadat de IP routering is volbracht zal de originele LSP opnieuw moeten worden opgezet. Figuur 17 toont hoe de kortste pad LSP A-B-C-D tweemaal geherrouteerd is geweest, eenmaal voor een fout over link AB en eenmaal voor een fout over link CD.



Figuur 17: Terugkeer in FTCR hangt af van de terugkeer van het IP routeringsprotocol.

In het geval van een kortste pad LSP is de terugkeer eenvoudig omdat deze LSP enkel gekenmerkt wordt door het doeladres wat steeds gekend is. Indien echter een expliciete LSP geherrouteerd wordt moet het volledig pad worden opgeslagen omdat tijdens de terugkeercyclus dit pad gekend moet zijn.

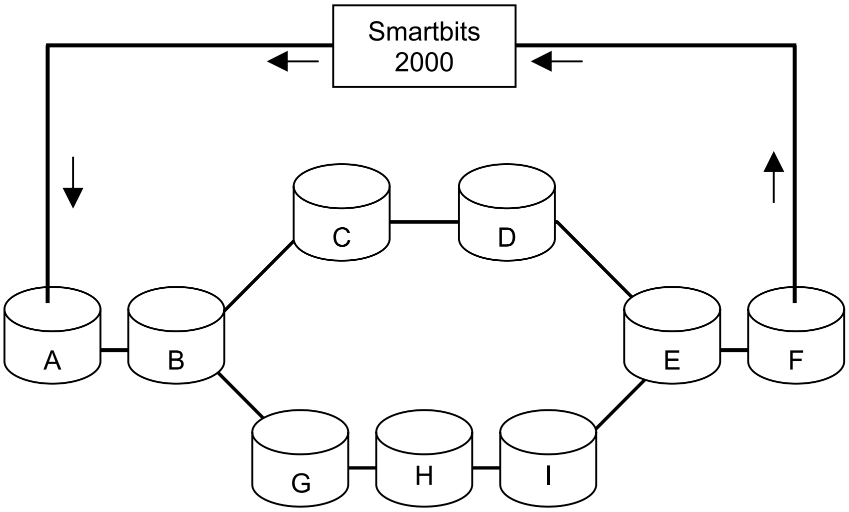
I.6 Evaluatie van foutconvergerende technieken

In deze sectie evalueren we de convergentietechnieken over een aantal criteria. Naast de convergentietijd kijken we ook naar de schaalbaarheid, de stabiliteit en de hoeveel extra middelen die nodig zijn om een bepaalde hoeveelheid trafiek te beschermen.

I.6.1 Convergentietijd

Test netwerk en meettechniek
We maken gebruik van een netwerk van Linux routers [48, 49, 50, 51]. Het alternatief is het gebruik van commerciële routers maar deze zijn niet eenvoudig aanpasbaar. Daarom is het onmogelijk om FTCR erop te implementeren. Vandaar dat dit alternatief niet werd weerhouden.

De routers worden met elkaar verbonden via het testnetwerk maar daarnaast ook met een controlenetwerk. Het controlenetwerk wordt gebruikt om de tests te orchestreren. Daarnaast wordt het netwerk verbonden met een Smartbits 2000 netwerktestapparaat [52]. De Smartbits is in staat zorgvuldig getimed, genummerde pakketten te sturen doorheen het testnetwerk (zie Figuur 18).



Figuur 18: De Smartbits stuurt genummerde testpakketten doorheen het testnetwerk.

De convergentietijd wordt bepaald door het aantal gemiste pakketten te vermenigvuldigen met de tijd tussen het verzenden van de pakketten:

convergentietijd = (aantal pakketten verloren ± 1)* (1 / snelheid) (2)

Formule (2) neemt reeds de absolute fout van één pakket in rekening. Figuur 19 toont dat de gemeten convergentietijd één eenheid hoger of lager kan liggen dan de reële convergentietijd. Deze fout stemt overeen met de absolute fout op de meting. In de experimenten die we uitvoeren komt de convergentietijd overeen met meer dan drie pakketten. De relatieve fout zal dus veel lager liggen dan de 50% die we krijgen in het voorbeeld van Figuur 19. In de praktijk krijgen we een relatieve fout tussen de 0.01% en de 5.6% wat acceptabel is.

	Experiment	**Gemeten waarde**	**Reële Waarde**
A	• (x x) • • •	**2**	**~2**
B	• (• x •) • •	**1**	**~2**
C	• (x x x) • •	**3**	**~2**

Figuur 19: De absolute fout op de meting

Er is echter een tweede oorzaak voor fouten op de metingen. De meetmethode veronderstelt dat de meetpakketten die door de Smartbits worden verstuurd met een constante snelheid door het netwerk worden verstuurd. Dit veronderstelt ten eerste dat de pakketten met een constante vertraging worden verstuurd. Dit is geen probleem omdat de Smartbits de pakketten met voldoende precisie kan versturen (400ns). Het bufferen van pakketten in de routers zorgt echter voor fluctuaties in de snelheid waarmee de pakketten het netwerk doorkruisen.

Cumulatieve vertragingsdistributie

<0.2	<0.5	<1	<2.5	<5	<10	<25	**ms**
0	99,32	99,40	99,62	99,89	99,99	100	**%**

Figuur 20: De vertraging die de meetpakketten ondervinden wanneer ze het netwerk doorkruisen.

Figuur 20 toont de vertraging van de pakketten indien ze worden verstuurd met een tussenpauze van 1ms. Uit de figuur leren we dat het merendeel van de pakketten een kleine vertraging ondervinden. 99,40% van de pakketten ondervinden een vertraging die lager ligt dan de absolute fout van 1ms. De overige pakketten kunnen echter een vertraging oplopen tot 25ms. De kans dat een pakket significant wordt vertraagd (>10ms) bedraagt 0,01%. Opdat een dergelijk pakket een invloed kan hebben op de convergentietijd moet het overlappen met het begin of het einde van de convergentieperiode. Indien de convergentieperiode bijvoorbeeld 50ms bedraagt dan is de kans dat een pakket de meting van de periode beïnvloedt 0.2%. Indien we verder opmerken dat elk experiment 100 maal wordt herhaald dan is het duidelijk dat significant vertraagde pakketten geen grote invloed hebben op de gemiddelde convergentietijd. We zijn echter ook geïnteresseerd in de minimale en de maximale convergentietijd. Opnieuw vormt dat geen probleem omdat een dergelijke anomalie eenvoudig kan worden opgemerkt in een reeks van 100 metingen.

Parameters

In de experimenten onderzoeken we de convergentietijd van OSPF met directe foutdetectie, RSVP herroutering door verversing van staat, RSVP herroutering getriggerd door de OSPF herrroutering, FTCR en protectie.

OSPF en RSVP hebben een aantal configuratieparameters deze worden getoond in Tabel 1. Daarnaast vermeldt deze tabel ook dat elk experiment 100 maal wordt uitgevoerd en dat het meetinterval 1ms bedraagt.

De topologie van het netwerk gebruikt in de experimenten is deze van Figuur 18.

Tabel 1: De algemene en de OSPF en de RSVP parameters gebruikt in de experimenten.

	Parameter	Waarde
a	Aantal metingen	100
b	Meetinterval	1ms
c	OSPF Hello interval	10s
d	OSPF Router Dead interval	40s
e	OSPF SPF vertraging	5s
f	RSVP ververs periode (R)	30s
g	RSVP herstelwachttijd	2s

Resultaten

In deze sectie geven en bespreken we de resultaten van de experimenten [8]. We zien onmiddellijk op Figuur 21 dat er enorme verschillen bestaan tussen de technieken, van die aard zelfs dat de grafieken voor FTCR en protectie niet zichtbaar zijn op de lineaire schaal van Figuur 21a. We bespreken de resultaten nu één voor één. Daarbij maken we gebruik van de convergentiecyclus uit sectie I.4.1.

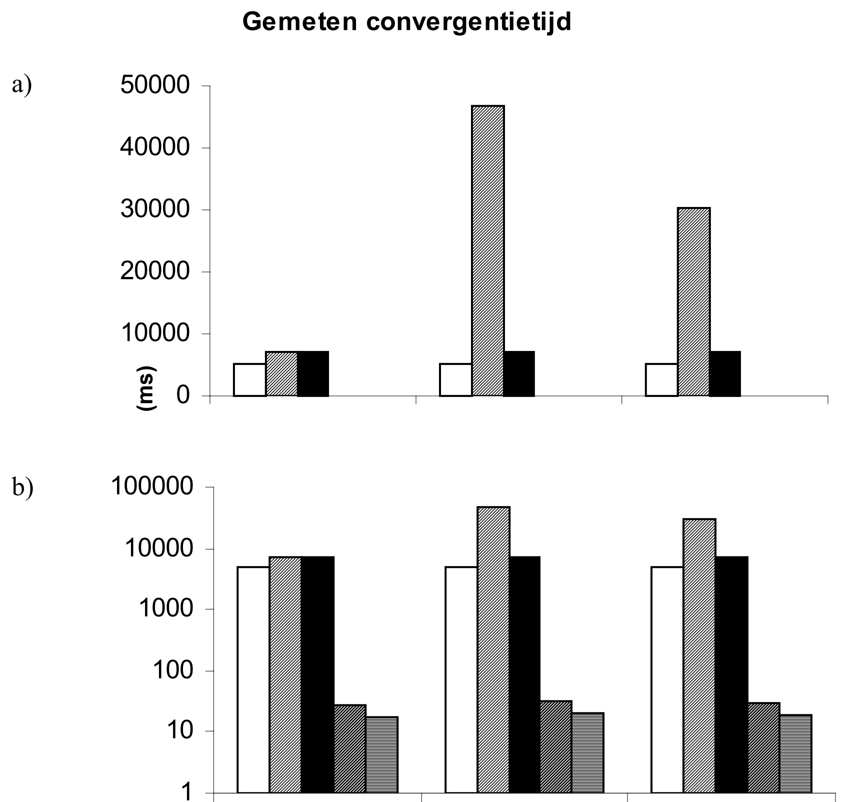
Voor elke techniek die we bespreken geven we een tabel met de theoretische convergentiecyclus en de convergentiecyclus waarbij bepaalde waarden, waar mogelijk, gesubstitueerd zijn met deze uit Tabel 1. Onbekende factoren worden voorgesteld door een Griekse kleine letter.

We zien dat in de derde rij van Tabel 2 de herstelwachttijd 5s bedraagt. Deze waarde wordt gevonden door de SPF vertraging in de tweede rij te substitueren met de waarde uit Tabel 1. Merk op dat de SPF wachttijd geen rol speelt omdat er slechts één SPF berekening moet worden uitgevoerd. Daarnaast zien we dat de overige factoren die de convergentietijd beïnvloeden onbekend zijn.

Tabel 2: Convergentiecyclus van OSPF met directe foutdetectie

Foutdetectietijd	Verwittigings-tijd	Herstelwacht-tijd	Hersteloperaties
κ	δ	[SPF vertraging, SPF wachttijd]s	SPF (ε), aanpassen RIB (γ)
κ	δ	5s	ε+γ

In Figuur 21 zien we dat de convergentietijd iets meer dan ongeveer 5s bedraagt. We zien dus dat de theoretische convergentietijd overeenkomt met de experimentele convergentietijd. Daarnaast zien we dat de tijd die nodig is voor de detectie van de fout en het berekenen en het aanpassen van de routeringstabellen klein is (<27ms).



	Min	Max	Avg
□ OSPF	5013	5027	5018,45
▨ Softstate RSVP	7031	46803	30348,00
■ Getriggered RSVP	7030	7033	7031,43
▨ FTCR	28	31	29,60
▦ Protectie	18	20	18,59

Figuur 21: De convergentietijden van de verschillende technieken: a) lineaire schaal b) logarithmische schaal. De minimale, maximale en gemiddelde waarden van de convergentietijden zijn gegeven voor 100 experimenten.

Tabel 3: Convergentiecyclus van RSVP herroutering door verversing van staat.

Foutdetectietijd	Verwittigings-tijd	Herstelwachttijd	Herstel-operaties
IP convergentie	IP convergentie	RSVP herstelwachttijd + [0,5-1,5]R	Opzetten LSP (ζ)
IP convergentie	IP convergentie	2s + [15-45]s	Opzetten LSP (ζ)

De theoretische convergentietijd voor RSVP herroutering door verversing van staat is gegeven in Tabel 3. De foutdetectie en verwittiging gebeurt in MPLS herroutering onder invloed van de IP herroutering. De herstel-LSP wordt opgezet van zodra de originele LSP wordt ververst en dus niet onder impuls van de gewijzigde routeringstabel. Indien er een wijziging in de routeringstabel wordt vastgesteld tijdens de verversing van de LSP dan wacht men even (de herstelwachttijd) voor de LSP wordt ververst. Een LSP wordt gemiddeld elke 30s (R) ververst maar deze concrete waarde varieert tussen de 15s en de 45s. We kunnen dus verwachten dat de convergentietijd tot 2s + 45s hoger ligt dan bij IP herroutering plus de tijd die nodig is om de herstel-LSP op te zetten. Deze analyse wordt bevestigd door Figuur 21. Enkel de hoogst mogelijke theoretische convergentietijden worden niet gehaald. Dit komt door de zwakke implementatie van verversingsinterval waarbij de hoogste waarde slechts 42s in plaats van 45s bedraagt (gecontroleerd over 100 experimenten). Toch is dit voldoende om aan te tonen dat de convergentietijd sterk varieert en potentieel heel hoog kan zijn in vergelijking met OSPF herroutering.

Tabel 4: Convergentiecyclus RSVP herroutering getriggerd door gewijzigde routeringstabel.

Foutdetectietijd	Verwittigings-tijd	Herstelwachttijd	Herstel-operaties
IP convergentie	IP convergentie	RSVP herstelwachttijd	Opzetten LSP (ζ)
IP convergentie	IP convergentie	2s	Opzetten LSP (ζ)

Zoals we reeds hebben besproken is het ook mogelijk de herroutering van een LSP onmiddellijk te starten wanneer de routeringstabellen worden aangepast door de IP routering. Op deze manier gebeurt de convergentie veel sneller omdat deze niet meer afhangt van de verversingsperiode van RSVP. Figuur 21 toont duidelijk aan dat dit een positieve invloed heeft op de convergentietijd. De convergentietijd van RSVP met getriggerde herroutering is ook veel beter voorspelbaar. De convergentietijd ligt nog steeds hoger dan deze van OSPF omdat er een herstelwachttijd is van 2s. Het herrouteren van de LSP wordt wat vertraagd om de routeringstabellen in het netwerk te laten stabiliseren.

Tabel 5: Convergentiecyclus van FTCR

Foutdetectietijd	Verwittigings-tijd	Herstelwacht-tijd	Hersteloperaties
κ	η	0s	Berekening herstel-LSP (θ), opzetten LSP (ζ)

Zoals we reeds hebben besproken worden de hersteloperaties in FTCR onmiddellijk gestart na het detecteren van de fout. Een LSP wordt hersteld door het berekenen en het opzetten van de herstel-LSP vanuit de FSL. In deze experimenten gebruiken we lokaal herstel. Figuur 21b toont duidelijk aan dat FTCR aanzienlijke sneller is dan MPLS herroutering en OSPF herroutering.

Herstel met behulp van protectie gebeurt, zoals in FTCR, ook onmiddellijk na het detecteren van de fout. Het verschil tussen FTCR en protectie is dat bij protectie de herstel-LSP reeds vooraf is opgezet. De enige actie die dus nog moet worden ondernomen is het overschakelen van de originele LSP naar de herstel-LSP (zie Tabel 6). Dit zorgt voor nog snellere convergentietijden dan FTCR (ongeveer 10ms sneller).

Tabel 6: Convergentiecyclus van protectie

Foutdetectietijd	Verwittigings-tijd	Herstelwacht-tijd	Hersteloperaties
κ	η	0s	Aanpassen LIB (ι)

Het verschil tussen de convergentietijden van FTCR en protectie zal echter toenemen als het netwerk in grootte toeneemt omdat zowel de rekentijd als de tijd die nodig is om de herstel-LSP op te zetten toeneemt.

Besluit

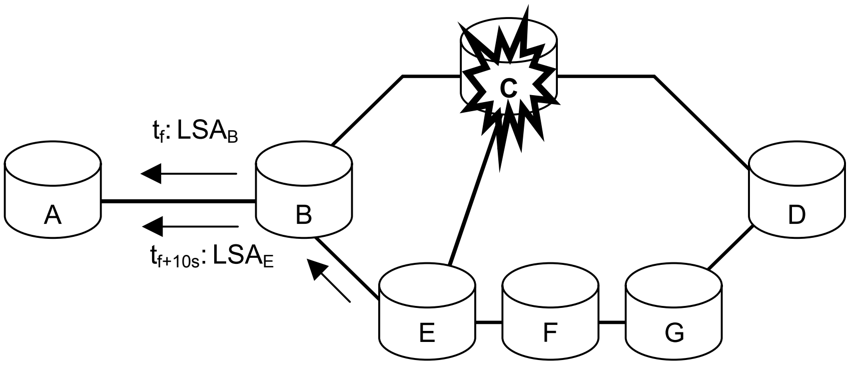
We kunnen drie categorieën van convergentietechnieken onderscheiden. De snelste technieken zijn protectie en FTCR. OSPF herroutering en getriggerde RSVP herroutering zijn twee orden van grootte trager maar hebben een voorspelbare convergentietijd. MPLS herroutering door de verversing van staat is potentieel zeer traag en kan dus best worden vermeden indien mogelijk. Te meer daar het geen voordelen biedt ten opzichte van getriggerde herroutering.

I.6.2 Stabiliteit

De convergentietijd van OSPF en MPLS herroutering wordt grotendeels bepaald door de herstelwachttijd. Deze wachttijden verhogen de convergentietijd maar kunnen niet zomaar worden geëlimineerd omdat ze de stabiliteit moeten garanderen. In deze sectie leggen we uit waarom de herstelwachttijden nodig zijn in OSPF en MPLS herroutering en waarom ze niet nodig zijn in FTCR.

OSPF

OSPF gebruikt SPF vertraging en SPF wachttijd om het aantal kortste pad berekeningen te beperken (zie sectie I.4.3). Indien een knoop in het netwerk faalt zal elke router verbonden met die knoop het falen van de link met de falende knoop detecteren. Als gevolg daarvan zal elke router direct verbonden met de falende knoop een nieuw linkstatuspakket verzenden waarbij de kost van de link oneindig is. Dit kan leiden tot een groot aantal linkstatuspakketten op korte termijn. Het is dan niet aan te raden om direct bij het binnenkomen van een linkstatuspakket direct een kortste pad berekening te doen omdat nog niet alle linkstatuspakketten zijn ontvangen. De SPF vertraging en de SPF herstelwachttijd dienen dus om het aantal onnodige kortste pad berekeningen te beperken. Het introduceren van deze wachttijden zal echter steeds de convergentietijd doen toenemen.



Figuur 22: Het tijdsverschil tussen linkstatuspakketten van knoop E en knoop B bij het falen van knoop C kan oplopen tot 10s.

De optimale waarden van SPF vertraging en SPF herstelwachttijd zijn zeer moeilijk te bepalen. Het is dan ook eenvoudig aan te tonen dat de standaard waarden van de Zebra OSPF implementatie niet ideaal zijn.

Stel dat knoop C faalt, zowel knoop B als knoop E zullen dit detecteren (zie Figuur 22). Stel dat het Hello protocol wordt gebruikt om de fout te detecteren. Het verschil tussen de tijd dat knoop B het falen van knoop C detecteert en de tijd dat knoop E dit doet kan oplopen tot 10s (zie formules (1, 2, 3)).

$T_{detect} = T_1 – T_0 =$ [ontvangstinterval– zendinterval, ontvangstinterval] (2)

$T_{detect} = T_1 – T_0 = [40 – 10, 40] = [30, 40]$ (3)

Doordat de detectietijd 10s kan verschillen is het ook mogelijk dat knoop A de linkstatuspakketten van B en E met een tussenpauze van 10s ontvangt. Knoop A zal, na het ontvangen van het linkstatuspakket van B, 5s (SPF vertraging) wachten alvorens de kortste pad berekening te starten. In dit geval zal dit niet voldoende zijn om het linkstatuspakket van knoop E te ontvangen. Als gevolg daarvan zal pas de volgende kortste pad berekening, 10s later (SPF wachttijd), het linkstatuspakket van E in rekening brengen.

Niet enkel het verschil in detectietijd kan zorgen voor grote verschillen tussen het ontvangen van de linkstatuspakketten van eenzelfde fout. Ook door vertraging in het netwerk of het verliezen van een linkstatuspakket, waardoor het opnieuw moet worden verzonden, kan leiden tot grote vertragingen.

De stabiliteit van OSPF komt ook in gevaar tijdens linkstatuspakketstormen (LSA storm). Tijdens een LSA storm worden onder invloed van een groot aantal wijzigingen in het netwerk zeer veel linkstatuspakketten verzonden. Dit kan als gevolg hebben dat de pakketten niet tijdig worden bevestigd waardoor ze opnieuw moeten worden verstuurd waardoor er nog meer pakketten worden verstuurd. Bij een voldoende grote linkstatuspakketstorm kan deze situatie blijven duren zodat het netwerk totaal onstabiel wordt. Een linkstatuspakketstorm kan bijvoorbeeld te wijten zijn aan het falen van een groot aantal knopen tegelijk. De stormen kunnen voorkomen worden door de timers in het OSPF netwerk te vergroten maar dit heeft natuurlijk als gevolg dat de convergentietijden stijgen.

In het algemeen kan men stellen dat er een wisselwerking bestaat tussen de stabiliteit en de convergentietijden in OSPF.

MPLS herroutering

De stabiliteit van MPLS herroutering ligt typisch hoger dan deze in OSPF. Dit komt doordat er een extra wachttijd timer wordt gebruikt (de RSVP herstelwachttijd in de experimenten van sectie I.6.1). Met behulp van deze herstelwachttijd probeert men ervoor te zorgen dat de routeringstabellen stabiel zijn alvorens de herstel-LSP op te zetten. Uit Figuur 22 blijkt echter wel dat de standaard waarde van 2s niet altijd voldoende is (in dit geval is er meer dan 10s nodig). Toch zal de standaard timer al een aantal fluctuaties die optreden tijdens het herrouteren in IP kunnen opvangen.

Indien de routeringstabellen nog niet stabiel zijn na 2s zal dit echter stabieler worden opgevangen door MPLS herroutering omdat deze een herstel-LSP opzet. Indien bijvoorbeeld de routeringstabellen de trafiek in een zwart gat sturen dan zal de herstel-LSP niet worden opgezet simpelweg omdat deze niet kan worden opgezet. Er kan gesteld worden dat het pad van een LSP wordt getest tijdens het opzetten ervan.

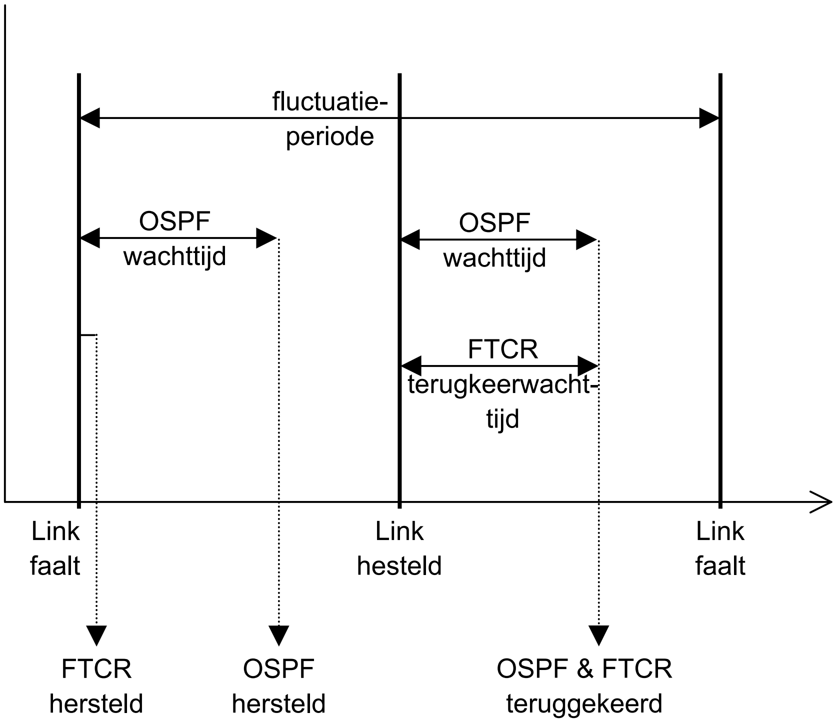
Men kan besluiten dat MPLS herroutering stabieler is dan IP herroutering door het gebruik van een extra wachttijd en door de padgeoriënteerde natuur van MPLS.

Protectie

Protectie is in principe zeer stabiel omdat er tijdens het optreden van een fout enkel moet worden overgeschakeld naar een vooraf opgezette herstel-LSP. Het ondersteunen van meerdere fouten is echter moeilijk met behulp van protectie. Meerdere fouten ondersteunen vereist dat niet enkel de werkende LSPs maar ook de herstel-LSPs worden beschermd. Daarnaast moeten ook fouten kunnen worden ontdekt op de herstel-LSPs. Wegens de complexiteit ervan worden meerdere fouten niet altijd ondersteund door protectie. Indien meerdere fouten niet worden ondersteund dan kan de techniek niet als stabiel worden beschouwd. Daarnaast worden soms ook niet alle enkelvoudige foutscenario's ondersteund omdat dit te veel herstel-LSPs vereist. Dit leidt opnieuw tot een verminderde stabiliteit.

FTCR

FTCR convergeert veel sneller dan OSPF omdat in FTCR geen herstelwachttijden worden gebruikt. Het is dan ook de vraag of dit de stabiliteit van FTCR niet in gevaar brengt. In de vorige paragraaf hebben we verklaard waarom OSPF wachttijden nodig heeft. FTCR heeft deze niet nodig omdat er slechts één FSL is die verantwoordelijk is voor een bepaalde fout op een LSP. In OSPF is dit niet het geval omdat een fout minstens door twee knopen wordt ontdekt en deze knopen ook beide acties zullen ondernemen als gevolg daarvan. Stel dat bijvoorbeeld een link faalt dan zullen in OSPF beide eindpunten een nieuw linkstatuspakket versturen. De routers in het netwerk zullen deze informatie dan gebruiken om nieuwe routeringstabellen te berekenen. In FTCR zal enkel de knoop direct stroomopwaarts van de fout een herstelactie ondernemen met betrekking tot de aangetaste LSPs. Waar in OSPF de herstelacties worden gedistribueerd, zullen bij FTCR deze centraal gebeuren in de FSL. Doordat er geen gedistribueerde herstelacties zijn is er ook geen nood aan de impliciete synchronisatie door expliciete wachttijden. Deze vaststelling geldt voor alle FSL selectiemodes van FTCR maar ook indien meerdere fouten optreden. Indien er meerdere fouten optreden zal er nog steeds slechts één FSL verantwoordelijk zijn voor een bepaalde fout op een gegeven LSP ondanks het feit dat er meerdere fouten optreden op de LSP.



Figuur 23: De fluctuatieperiode van een link die alterneert tussen werkende en falende status. Het herstel en de terugkeer van OSPF en FTCR zijn geïllustreerd.

Het snelle herstel van FTCR betekent ook niet dat FTCR onstabiel is bij het fluctueren van een link tussen werkende en falende status. Dit komt omdat FTCR tijdens de terugkeercyclus afhankelijk is van de terugkeer in IP. Stel dat een link zeer sterk fluctueert tussen werkende en falende status dan zal in OSPF de fluctuaties van de routeringstabellen in het netwerk worden beperkt door de wachttijden. Doordat FTCR voor de terugkeer afhangt van OSPF zal de fluctuatieperiode gelijk zijn aan deze van OSPF. Dit is geïllustreerd in Figuur 23 waar de link iets trager fluctueert dan twee maal de OSPF wachttijd. Deze figuur toont ook dat FTCR even traag fluctueert als OSPF omdat er bij de terugkeer wordt gewacht op OSPF. Het is echter duidelijk dat het herstel, wat veel kritischer is dan de terugkeer, veel sneller gebeurt in FTCR dan in OSPF. Men zegt dat FTCR snel reageert op slecht nieuws en traag op goed nieuws. Deze positieve eigenschap van FTCR heeft OSPF niet omdat het even snel op slecht nieuws als op goed nieuws reageert.

I.6.3 Schaalbaarheid

In deze sectie kijken we naar de schaalbaarheid van de convergentietechnieken in functie van de grootte van het netwerk en het aantal LSPs dat kan worden ondersteund.

OSPF
De schaalbaarheid van OSPF wordt beperkt door de maximale grootte van de linkstatusdatabank. Zowel de opslagruimte en de hoeveelheid linkstatus-pakketten dat moet worden verspreid over het netwerk neemt toe als de linkstatusdatabank in grootte toeneemt. De grootte van de linkstatusdatabank hangt af van het aantal knopen in het netwerk maar ook met het aantal externe routes dat wordt toegevoegd. Externe routes worden soms toegevoegd om steeds de ideale uitgang in het netwerk naar een dergelijke route te gebruiken. Daarnaast neemt de duur van de kortste pad berekening ook toe naarmate de grootte van de linkstatusdatabank toeneemt (O(n.log(n)) of $O(n^2)$).

Het is mogelijk om een OSPF netwerk op te delen in meerdere subnetwerken met één centraal ruggengraatnetwerk. Op die manier zullen meerdere, kleinere linkstatusdatabanken worden gebruikt met als nadeel dat de verschillende subnetwerken moeten worden geconfigureerd en onderhouden [36, 53].

MPLS herroutering
Zoals de herroutering in MPLS afhangt van het IP routeringsprotocol wordt de schaalbaarheid natuurlijk ook beperkt door het IP routeringsprotocol. Daarnaast wordt de schaalbaarheid ook bepaald door de MPLS implementatie. Er zijn namelijk een beperkt aantal LSPs dat simultaan kan worden ondersteund. Vooral bij RSVP kan dit voor beperkingen zorgen omdat in RSVP continu de staat met betrekking tot de LSPs wordt ververst. Bij een groot aantal LSPs kan dit zorgen voor een aanzienlijke hoeveelheid trafiek. Bijvoorbeeld met 10.000 LSPs over één link en een standaard verversperiode van 30 seconden bedraagt de gebruikte bandbreedte 600kb per seconde. LDP heeft dit probleem niet omdat de staat vast is en niet meer wordt herhaald. Gelukkig bestaan er extensies die verversingsactiviteit van RSVP drastisch reduceren zodat de schaalbaarheid in de buurt komt van deze van LDP [54].

Een ander facet van de schaalbaarheid is het aantal LSPs dat simultaan kan worden opgezet. In principe moet elke implementatie robuust genoeg zijn om een zeer groot aantal LSPs tegelijk op te zetten. De tijd die nodig is om een LSP op te zetten zal wel toenemen indien er veel LSPs tegelijk worden opgezet.

Protectie
De schaalbaarheid van protectie zal vooral afhangen van het aantal LSPs dat simultaan opgezet kan zijn. In protectie is dit veel meer kritiek dan in MPLS herroutering omdat de herstel-LSPs vooraf worden opgezet waardoor er dus meer LSPs simultaan bestaan. Het voordeel van protectie ten opzichte van MPLS herroutering met betrekking tot de schaalbaarheid is dat de snelheid waarmee een LSP wordt opgezet en het aantal LSPs dat simultaan kan worden opgezet niet meer kritiek is omdat de herstel-LSPs vooraf worden opgezet. Natuurlijk moet de juiste herstel-LSP nog steeds worden gekozen bij een fout wat bij een groot aantal aangetaste werkende LSPs tot eventueel schaalbaarheidsproblemen kan leiden.

FTCR
FTCR vereist dat er een linkstatusdatabank wordt onderhouden in het netwerk. Daarnaast wordt de schaalbaarheid ook beperkt door het aantal LSPs dat simultaan kan worden ondersteund en opgezet. De schaalbaarheid van FTCR is dus vergelijkbaar met deze van MPLS herroutering indien er daarbij een linkstatusrouteringsprotocol wordt gebruikt.

I.6.4 Extra capaciteit van een beschermd netwerk

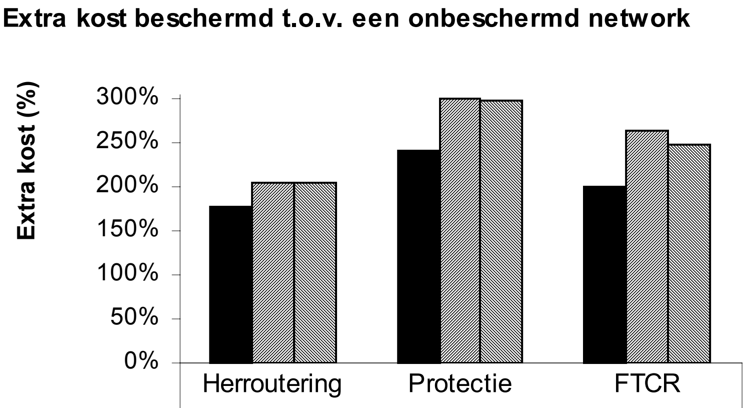
Om tijdens foutsituaties alle trafiek in het netwerk te kunnen blijven versturen moet er in het netwerk extra capaciteit worden voorzien. De hoeveelheid extra capaciteit die zo moet worden voorzien hangt af van convergentietechniek. De volgende sectie beschrijft het simulatiemodel dat werd gebruikt om de extra capaciteit te bepalen, de sectie daaropvolgend geeft de resultaten.

Simulatiemodel
In de simulaties wordt er verondersteld dat elke link bidirectioneel is maar dat de capaciteit ervan en de vraag erover unidirectioneel is. Daarnaast veronderstellen we dat de capaciteit van een link elke willekeurige positieve waarde kan aannemen. Verder veronderstellen we dat de kost van een link recht evenredig is met de lengte en de capaciteit ervan. De totale kost van het netwerk is de som van de links [55].

Voor de convergentietechnieken vergelijken we de kost van een onbeschermd netwerk met de kost waarbij het netwerk is beschermd tegen een willekeurige enkelvoudige fout. Herroutering en MPLS herroutering moeten daarbij niet afzonderlijk worden beschouwd omdat zij topologisch de trafiek op dezelfde manier herrrouteren met als gevolg dat de capaciteitsvereisten dezelfde zijn. We beschouwen naast OSPF herroutering, lokale protectie en FTCR met lokaal herstel.

We gebruiken het e.spire netwerk als voorbeeldtopologie in deze studie [55]. Het netwerk bestaat uit 44 knopen en 57 links. Het gewicht van de link wordt proportioneel genomen met zijn afstand, het gewicht van een knoop wordt bepaald door de grootte van de stad waarin het zich bevindt. Hoe groter de stad des te groter het gewicht en des te groter de vraag van en naar de knoop. De vraag in het netwerk wordt willekeurig bepaald maar er wordt rekening gehouden met het gewicht van de knopen en de links. De resultaten worden gegeven voor tien willekeurige vraagmatrices. De vraag in het netwerk wordt steeds zo goedkoop mogelijk gerouteerd door het netwerk (door gebruik te maken van het kortste pad algoritme).

Resultaten
Het eerste wat opvalt in Figuur 24 is dat het beschermen van het netwerk steeds op zijn minst een verdubbeling van de netwerkkost met zich meebrengt.



	Herroutering	Protectie	FTCR
■ Symmetrisch	178%	242%	200%
▨ Naar 1 bestemming	205%	301%	263%
▧ Vanuit 1 bron	205%	299%	248%

Figuur 24: De extra kost van een netwerk dat is beschermd tegen een enkelvoudige fout ten opzichte van een onbeschermd netwerk.

Verder zien we dat herroutering de goedkoopste techniek is gevolgd door FTCR en daarna protectie. Herroutering is de goedkoopste techniek omdat herroutering de trafiek in het netwerk globaal herrouteert en dus ook globaal kan spreiden. Dit in tegenstelling tot lokale protectie dat de trafiek lokaal beschermt. FTCR heeft stroomopwaarts het lokale gedrag van protectie maar stroomafwaarts het globale gedrag van herroutering met als gevolg dat de extra capaciteitskost van FTCR tussen deze van herroutering en protectie zit.

Verder zien we dat een asymmetrisch trafiekpatroon meer capaciteit vereist. Daarnaast zien we dat enkel FTCR meer capaciteit vereist indien de trafiek naar één bestemming vloeit in plaats van dat alle trafiek van één bestemming komt. Dit valt te verklaren door het asymmetrisch karakter van FTCR. De FSL zal alle trafiek over eenzelfde pad naar de bestemming sturen waardoor er een zeer geconcentreerd trafiek patroon ontstaat.

I.7 Verbetering van foutconvergerende technieken

In deze sectie kijken we naar recente verbetering in linkstatusroutering, hoe deze technieken kunnen worden toegepast op FTCR en hoe FTCR verder kan worden verbeterd.

I.7.1 Linkstatusroutering

We bekijken de recente verbeteringen die worden voorgesteld om de convergentietijden van linkstatusrouteringsprotocollen te verbeteren.

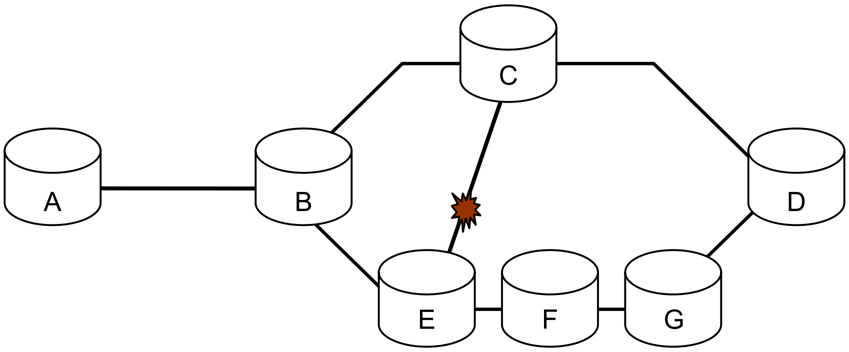
Bij de implementatie van de eerste linkstatusrouteringsprotocollen, rond 1990, concentreerde men zich vooral op de correctheid van de implementatie en niet op de snelheid van de convergentie. In een tweede fase werd de convergentiesnelheid wel belangrijk maar eigenlijk vooral in de marketing omdat de versnelde convergentie vaak gepaard ging met een verlaagde stabiliteit en dus onbruikbaar was in realistische netwerken. Vanaf 1995 begonnen de IP netwerken zeer groot en bedrijfskritisch te worden dus de stabiliteit was het eerste belang. Deze stabiliteit werd behaald door zeer lange timers te gebruiken wat natuurlijk een negatief effect heeft op de convergentietijd. Meer recent is er een hernieuwde interesse naar verlaagde convergentietijden onder invloed van het stijgende belang van interactieve services zoals telefonie over het Internet en de snelle convergentietijden van MPLS protectie [56].

Er zijn een aantal technieken die de convergentie kunnen versnellen. De belangrijkste is misschien wel eenvoudigweg het gebruik van hardwarematige foutdetectie. Met de standaard timers van OSPF duurt de detectie van een fout [30-40]s (zie Figuur 8) terwijl dat bij hardwarematige foutdetectie slechts enkele milliseconden duurt. Naast het gebruik van hardwarematige foutdetectie kan de convergentietijd ook versneld worden door het gebruik van incrementele kortste pad berekeningen en exponentiële timers.

Incrementele kortste pad berekeningen [57, 58, 59] zijn efficiënter dan gewone kortste pad berekeningen omdat ze rekening houden met de locatie van de verandering in de topologie van het netwerk. Een gewone kortste pad berekening zal steeds alle routes herberekenen ongeacht of dit noodzakelijk is of niet. Het is eenvoudig aan te tonen dat dit zeker niet altijd nodig is.

Stel dat de link CE faalt in het netwerk van Figuur 25. Bij een gewone kortste pad berekening zullen alle paden opnieuw worden berekend in alle knopen. Dit is niet nodig omdat bijvoorbeeld de kortste paden in de knopen A en B niet veranderen. In knoop E zullen de kortste paden naar C en D wel veranderen onder de invloed van de fout, maar de andere kortste paden niet. Opnieuw zal de incrementele kortste pad techniek ervoor zorgen dat knoop E enkel de nodige paden herberekent.

Incrementele kortste pad berekening kan leiden tot een tienduizendvoudige versnelling van de padberekening. De versnelling zal echter afhangen van de locatie van de fout, indien de fout zich in het centrum van het netwerk bevindt dan zal de versnelling kleiner zijn omdat de link of knoop dan praktisch voor elk pad wordt gebruikt.



Figuur 25: Incrementele kortste pad berekeningen vermijden dat overbodige kortste pad berekeningen in knoop A en B worden uitgevoerd.

Naast incrementele kortste pad berekeningen kan de convergentietijd ook worden ingekort door het gebruik van exponentiële timers op de wachttijden. Het idee hierbij is dat korte timers goed zijn voor de convergentietijd maar eveneens slecht voor de stabiliteit waarbij het gebruik van een vaste timer dus geen goed idee is. Bijvoorbeeld een korte wachttijd zal leiden tot snelle convergentie maar instabiel gedrag indien een link fluctueert tussen werkende en falende status. Het idee is de initiële waarden voor de timers klein te nemen maar indien de timers kort na elkaar aflopen de waarden ervan exponentieel, met een bepaald maximum, te laten toenemen. Dit heeft als gevolg dat fouten snel worden behandeld indien ze niet recentelijk voorkwamen. In het geval van een fluctuerende link zullen de routeringstabellen initieel snel mee fluctueren maar al snel zullen ze een stabiel gedrag vertonen omdat de wachttijden sterk stijgen. Het voordeel van exponentiële timers ligt er dus niet zo zeer in dat de convergentie altijd sneller gebeurt maar eerder dat de initiële waarden van de timers klein kunnen worden genomen zonder de stabiliteit op langere termijn in gevaar te brengen.

Door het toepassen van exponentiële timers en incrementele kortste pad berekeningen is het mogelijk convergentietijden onder de seconde te halen zelfs in grote netwerken. Snellere convergentie is niet mogelijk in grote netwerken omdat de rekentijd en de verwittigingstijd te hoog worden [60].

I.7.2 FTCR

De technieken die worden aangewend om de convergentie te versnellen in linkstatusroutering kunnen ook worden gebruikt in FTCR. FTCR heeft inderdaad ook baat bij incrementele kortste pad berekeningen bij het berekenen van de herstel-LSPs. Vermits de herstelcyclus van FTCR geen gebruik maakt van wachttijden zijn exponentiële timers hier niet toepasbaar. De terugkeercyclus onder invloed van de terugkeer in IP kan echter baat hebben bij exponentiële timers. Indien de terugkeer sneller gebeurt is de herstel-LSP minder lang kwetsbaar met als gevolg dat er minder kans is op meerdere fouten.

De convergentietijden in grote netwerken blijven zoals bij linkstatusroutering beperkt tot ongeveer één seconde. Snellere convergentie is niet mogelijk omdat de reken- en verwittigingstijden te groot zijn. In FTCR kan echter de convergentie versneld worden door de herstel-LSPs vooraf te berekenen. In FTCR met lokaal herstel betekent dit dat elke router, dus elke potentiële FSL, een nieuw pad berekent voor elke potentiële fout, dus elke uitgaande interface, voor elke LSP. In FTCR met herstel aan de ingang betekent dit dat de ingangs-LSRs een nieuw pad berekent voor elke LSP die er begint en voor elke potentiële fout op deze LSP. Het is belangrijk op te merken dat het berekenen van de paden van de herstel-LSPs in de achtergrond gebeurt en dus geen invloed heeft op de normale werking van de router. Indien het pad van een herstel-LSP vooraf berekend is dan kan de herstel-LSP sneller worden opgezet wanneer een fout optreedt.

Zelfs met het gebruik van vooraf berekende herstel-LSPs moet de herstel-LSP nog steeds worden opgezet wat de convergentietijd doet toenemen vooral in grote netwerken. Men kan natuurlijk de herstel-LSP zowel vooraf berekenen als vooraf opzetten. Merk op dat dit nog steeds in de achtergrond gebeurt zodat de techniek niet te vergelijken is met on line protectie. Het aantal opgezette herstel-

LSP kan hierdoor sterk toenemen. Het is daarbij aan te raden slechts een maximum aantal herstel-LSPs te voorzien. Daarnaast is het praktisch onmogelijk om in grote netwerken de werkende LSPs te beschermen tegen meerdere fouten. Het is dan ook aan te raden om in meerdere foutenscenario's te betrouwen op FTCR herroutering.

I.8 Besluit

MPLS is een techniek die labelgeschakelde paden aanbiedt aan IP netwerken. Deze LSPs kunnen bijvoorbeeld worden gebruikt om de trafiek in het netwerk beter te verdelen, om virtueel private netwerken aan te bieden of om meerdere protocol-lagen te elimineren. Daarnaast laat MPLS toe nieuwe convergentietechnieken te gebruiken die niet zomaar implementeerbaar zijn in zuivere IP netwerken. Voorbeelden daarvan zijn protectie en FTCR.

De convergentietechnieken die behandeld werden zijn IP herroutering, MPLS herroutering, FTCR en protectie. Naast het beschrijven van deze technieken zijn deze technieken ook geëvalueerd over een aantal criteria (zie Tabel 7).

Tabel 7: Overzicht van de eigenschappen van de behandelde convergentietechnieken. ++: zeer goed, +: goed, 0: neutraal, –: minder goed, – –: slecht.

	OSPF herroutering	MPLS herrroutering	FTCR	Protectie
Convergentie-tijd	–	–(–)	+(+)	++
Stabiliteit	–	0	+	+
Schaalbaar-heid	++	+	+	–
Capaciteits-vereisten	++	++	+(+)	–

Protectie convergeert het snelste na een fout omdat er enkel moet worden overgeschakeld van de werkende naar een vooraf opgezette herstel-LSP. FTCR is sneller dan herroutering omdat het niet wacht op de routeringstabellen en er geen herstelwachttijd wordt gebruikt. Dit is mogelijk omdat het herstel centraal gebeurt waardoor er geen synchronisatie nodig is. Toch blijft FTCR trager dan protectie omdat de herstel-LSP moet worden berekend en opgezet. MPLS herroutering is trager dan herroutering in IP omdat MPLS herroutering een extra herstelwachttijd gebruikt.

De extra wachttijd die MPLS herroutering gebruikt ten opzichte van IP herroutering vertraagt inderdaad de convergentie maar aan de andere kant neemt de stabiliteit er door toe. Daarnaast zijn de MPLS convergentietechnieken stabieler omdat ze gebruik maken van LSPs die enkel kunnen worden opgezet als hun pad geldig is. FTCR is stabieler dan MPLS herroutering omdat de herstelwachttijd, die gebruikt wordt in MPLS herroutering om er voor te zorgen dat de routeringstabellen geldig zijn, niet altijd volstaat. In FTCR is er geen herstelwachttijd nodig omdat de herroutering centraal gebeurt. De stabiliteit van FTCR wordt verder verbeterd doordat meerdere foutenscenario's worden ondersteund door LSP en LSP-aanvraag herroutering. Protectie is ook zeer stabiel omdat de hersteloperaties zeer eenvoudig en statisch zijn. Tijdens het herstel wordt er enkel overgeschakeld naar de herstel-LSP. De stabiliteit van protectie wordt gelimiteerd door het moeilijk ondersteunen van meerdere fouten.

De schaalbaarheid van linkstatusroutering wordt voornamelijk bepaald door de grootte van de linkstatusdatabank. Deze linkstatusdatabank moet worden onderhouden door het verspreiden van de linkstatuspakketten en hij moet ook worden opgeslagen in de routers. Ook de complexiteit van de kortste pad berekeningen hangt af van de grootte van de linkstatusdatabank. Vandaar dat er een bovengrens is op de grootte ervan. MPLS herroutering en FTCR schalen iets minder goed dan IP herroutering omdat de schaalbaarheid afhangt van de grootte van de linkstatusdatabank en de schaalbaarheid van de MPLS implementatie. De schaalbaarheid van de MPLS implementatie hangt af van het aantal LSPs dat kan worden ondersteund en hoeveel LSPs tegelijk kunnen worden opgezet. Protectie schaalt minder goed omdat het aantal herstel-LSPs vaak wordt beperkt door het maximum aantal LSPs. Deze bovengrens kan snel worden bereikt indien de werkende LSPs worden beschermd tegen meerdere fouten. Door het intelligent samen nemen van de herstel-LSPs kan de schaalbaarheid sterk worden verbeterd. Indien de LSPs slechts tegen enkelvoudige fouten moeten worden beschermd dan kan de schaalbaarheid van protectie als goed worden beschouwd.

OSPF herroutering, MPLS herroutering en FTCR met herroutering aan de ingang van de LSP vragen het minste extra capaciteit indien het netwerk beschermd wordt tegen enkelvoudige fouten. Lokale protectie heeft meer extra capaciteit nodig omdat de fout lokaal wordt hersteld zodat er geen globale spreiding mogelijk is. FTCR met lokaal herstel situeert zich tussen OSPF herroutering en lokale protectie omdat het stroomopwaarts het lokale gedrag heeft van protectie en stroomafwaarts het globale gedrag van herroutering.

FTCR kan beschouwd worden als een herrouteringstechniek die werkelijk gebruik maakt van de padgeoriënteerde eigenschappen van MPLS. MPLS herroutering doet dit niet omdat de LSPs enkel worden gebruikt voor het verzenden van pakketten, de routering hangt nog steeds af van de routeringtabellen. Daarnaast biedt FTCR ook de nodige flexibiliteit. Voorbeelden daarvan zijn de keuze uit verschillende algoritmen voor het herrouteren van ER-LSPs en de verschillende FSL selectie modes. De belangrijkste bron van flexibiliteit is waarschijnlijk de mogelijkheid om de herstel-LSP vooraf te berekenen en op te zetten, de herstel-LSPs vooraf te berekenen en op te zetten wanneer de fout optreedt of de herstel-LSPs pas te berekenen en op te zetten wanneer de fout optreedt. FTCR kan dus gezien worden als een techniek die de beste eigenschappen van herroutering en protectie combineert waarbij een grote vorm van flexibiliteit wordt gegeven.

De verschillende vormen van herstel in FTCR zijn interessant in een netwerk met differentiatie van diensten zoals in DiffServ netwerken. De meer premium diensten kunnen dan worden beschermd door vooraf opgezette herstel-LSPs en de minder belangrijke diensten door het berekenen en opzetten van herstel-LSP tijdens het optreden van de fout.

Chapter 1

Introduction

1.1 Scope

The Internet has evolved into a high speed and mission critical network. Even a short disruption in its network operation causes severe loss of data and disrupts interactive services. Longer network outages lead to large monetary losses. Unfortunately network failures are inevitable. Using robust equipment and physically protecting that equipment to prevent failures is only effective to a certain extent. It is important to prevent failures but it is equally important to limit the negative effects of failures by recovering the network from these failures as good as possible. An important, but certainly not the only factor, is the time it takes before the network becomes operational again after a failure.

The subject of this work is how networks are able to recover from failures. Techniques that are used to recover a network after a failure are called *failure convergence schemes*. The Merriam-Webster dictionary [61] defines c*onvergence* as “the act of *converging* and especially moving toward union or uniformity”. And *to converge* in turn is defined as “to tend or move toward one point or one another: come together”. As such, failure convergence can be regarded as a set of operations, possibly distributed, which have the overall goal of recovering the network after a failure.

Supporting failure convergence in networks to improve their reliability is a very broad and active research field and as such it is important to give the scope of our work up front. This work is about the operational procedures of failure convergence. It describes and analyses the recovery operations the network undertakes when a failure occurs. Therefore, its main topic is not about planning how a network can be made more fault-tolerant or how much resources are needed to protect the traffic in the network. Although we will briefly touch upon the latter. We limit our scope also to IP networks. That means that other technologies like the self-healing rings of SDH will not be considered and that multi-layer failure convergence is also out-of-scope [4, 5, 6, 62, 63]. We will investigate which failure convergence schemes are available to pure IP networks but also to IP networks that use Multi-Protocol Label Switching (MPLS). As we will see, MPLS has a lot to offer to IP networks in terms of failure convergence schemes.

The title of this work is “Evaluating and improving failure convergence schemes in IP networks”. This means that we will *evaluate* how the different convergence schemes for IP networks perform under different circumstances and with respect to different criteria. We will also discuss recent *improvements* to these convergence schemes. But the most important improvement is given by a new convergence scheme developed by the author called Fast Topology based Constrained Rerouting (FTCR).

1.2 Overview

In the first chapter after this introduction, we explain how MPLS works and describe its major applications. We start by comparing the forwarding in IP and MPLS networks. Later on we will explain the major concepts of the MPLS forwarding and signalling in more detail. Many of these concepts are used in the MPLS convergence schemes and are also used in the major applications of MPLS. These applications are also covered in the second chapter. We explain the applications of MPLS to illustrate that MPLS has other benefits besides the improved convergence schemes. The major part of this chapter has been published in [2].

The third chapter we will introduce the failure convergence schemes for IP and MPLS. We propose a generic recovery and reversion cycle which is based on the combination of the different cycles in [43]. In this chapter we will not only cover the different convergence schemes but we also look at failure detection and failure notification. One of the failure detection mechanisms that we cover has been published in [44]. After the discussion about failure detection and notification we look at several convergence schemes for IP and MPLS in more detail. We start with the IP convergence schemes: Routing Information Protocol (RIP) and Open Shortest Path First (OSPF). Afterwards we look at MPLS rerouting and Protection Switching (PS) two techniques used to recover traffic in MPLS networks.

While the third chapter covers existing convergence schemes, we will propose a new convergence scheme called Fast Topology based Constrained Rerouting (FTCR) in the fourth chapter. The idea of FTCR was first published in 2000 in [3] as a new convergence scheme for MPLS networks. Subsequent publications [4, 5, 6] compare the capacity of different convergence schemes including FTCR, investigate how different convergence schemes for electrical MPLS including FTCR can be ported towards optical networks and describes data centric optical networks and their survivability again including FTCR. In this chapter about FTCR we will look at how FTCR is able to recover a single failure on a shortest path LSP. Afterwards we investigate more complex scenarios including recovering failures on explicitly routed LSPs, recovering multiple failures and FTCR in revertive mode. The chapter concludes with the overall FTCR operations, an FTCR requirement matrix and the final conclusions.

In the fifth chapter we will evaluate the different convergence schemes covered so far over a number of criteria. The first criteria we use is the speed of convergence. The time to convergence of the different schemes is measured in a PC based router network. First we cover the design of the router platform which has been published in [7]. Afterwards we look at the measurement technique used and the error analysis. This is followed by the results of the measurements and the conclusions that can be drawn from them. These results have been partially published in [8]. We will also look at the scalability and stability of the convergence schemes. Finally, the amount of backup resources that is needed to protect the working traffic is investigated. The work related to the backup resource requirements has been published in [4, 5, 6].

In the sixth chapter we will look at the recent advancements in failure convergence times. We will focus mainly on new techniques in link-state routing and how these techniques can be applied to FTCR. Further on we look at how the convergence times of FTCR can be improved even more by introducing techniques from protection switching to FTCR.

The final chapter summarises the most important conclusions from this work and adds some personal remarks.

1.3 Publications

1.3.1 International Journal papers

1. I. Andrikopoulos, G. Pavlou, P. Georgatsos, N. Karatzas, K. Kavidopoulos, J. Rothig, S. Schaller, D. Ooms, **P. Van Heuven**, “Experiments and enhancement for IP and ATM integration: The IthACI project”, IEEE Communications Magazine, ISBN 0163-6804-01, Vol. 39, Nr. 5, May 2001, Pages 146-155.

2. D. Colle, **P. Van Heuven**, C. Develder, S. Van den Berghe, I. Lievens, M. Pickavet, P. Demeester, “MPLS recovery mechanisms for IP-over-WDM networks”, Photonic Network Communications, Kluwer Academic Publishers, Vol. 3, Nr. 1/2, January 2001, Pages 23-40.

3. D. Colle, A. Groebbens, **P. Van Heuven**, S. De Maesschalck, M. Pickavet, P. Demeester, “Porting MPLS-recovery techniques to the MPlS

paradigm", (Invited paper) Optical Networks Magazine, Special Issue on Protection and Survivability, Vol. 2, Nr. 4, July/August 2001, Pages 29-47.

4. D. Colle, S. De Maesschalck, C. Develder, **P. Van Heuven**, A. Groebbens, J. Cheyns, I. Lievens, M. Pickavet, P. Lagasse, P. Demeester, "Data centric Optical Networks and Their Survivability", IEEE journal of selected areas in communications, Vol. 20, Nr.1, January 2002, Pages 6-21.

5. S. Van den Berghe, **P. Van Heuven**, J. Coppens, F. De Turck, P. Demeester, "Distributed policy-based management of measurement-based traffic engineering: design and implementation", Future Generation Computer Systems, Vol. 19, Issue 2, February 2003, Pages 291-302.

1.3.2 International Conference papers

1. D. Colle, S. De Maesschalck, **P. Van Heuven**, P. Demeester, M. Pickavet, "Reliability consideration in WDM empowered IP networks", Combined COST 266 and COST 267 workshop Optical Signal Processing in Photonic Networks, April 5-7, 2000, Berlin, Germany, Pages 18-20.

2. P. Demeester, D. Colle, S. Demaesschalck, C. Develder, M. Pickavet, **P. Van Heuven**, "Resilience in IP over WDM based multilayer networks", (Invited paper) The 26st European Conference on Optical Communication (ECOC 2000), Tutorial, 3-7 September, 2000, Munich, Germany.

3. D. Colle, C. Develder, **P. Van Heuven**, M. Pickavet, P. Demeester, L. Raptis, G. Chatzilias, C. Mas, Y.I. Manolessos, J. Comellas, A. Rafel, J. Prat, J. Solé-Pareta, J. Moyano, S. Brunazzi, S. Rotolo, R. Stankiewicz, A. Gladish, "Recovery techniques for IP-over-WDM networks", (Invited paper) IP over DWDM conferentie (CD-Rom), 27-30 November 2000, Paris, France.

4. **P. Van Heuven**, S. De Maesschalck, D. Colle, S. Van den Berghe, M. Pickavet, P. Demeester, "Recovery in IP based networks using MPLS", IEEE Workshop on IP-oriented Operations & Management IPOM'2000, ISBN 83-88309-00-5, 2-4 September, 2000, Cracow, Poland, Pages 70-78.

5. **P. Van Heuven**, P. Demeester, "Defining the rationale for moving restoration and protection up to the MPLS layer", MPLS Forum June 12-13, 2000, Dublin, Ireland.

6. **P. Van Heuven**, P. Demeester, “Defining the rationale for moving restoration and protection up to the MPLS layer”, The Second Vision in Business summit on MPLS, June 14-16, 2000, Dublin, Ireland.

7. C. Develder, D. Colle, **P. Van Heuven**, S. Van den Berghe, M. Pickavet, P. Demeester, “Influence of recovery time on TCP behaviour”, MPLS World Congress "Building the New IP Architecture", 6-9 February 2001, Paris, France.

8. C. Duret, F. Rischette, J. Lattmann, V. Laspreses, **P. Van Heuven**, S. Van den Berghe and P. Demeester, “High Router Flexibility and Performance by Combining Dedicated Lookup Hardware (IFT), off the Shelf Switches and Linux.”, Networking 2002, 19-24 May, Pisa, Italy, poster prensentation accepted.

9. P. Demeester, D. Colle, S. Demaesschalck, C. Develder, M. Pickavet, **P. Van Heuven**, “Resilience in IP over WDM based multilayer networks”, (Invited paper) Optical Networks, 2000, December 7, 2000, Chalmers Teknikpark, Göteborg, Sweden.

10. D. Colle, C. Develder, **P. Van Heuven**, S. Demaesschalck, A. Groebbens, M. Pickavet, P. Demeester, “Resilience in IP-over-WDM networks”, (Invited paper) Proceedings of the 5th Working Conference on Optical Network Design and Modelling (ONDM 2001), 5-7 February 2001, Vienna, Austria.

11. **P. Van Heuven**, J. Coppens, S. Van den Berghe, P. Demeester, “RSVP-TE daemon for DiffServ over MPLS under Linux”, Linux Kongress, p141-155, 5-6 September 2002, Cologne, Germany.

12. **P. Van Heuven**, J. Coppens, S. Van den Berghe, D. Colle, P. Demeester, “Quantitative and Theoretical Analysis of Recovery Convergence in IP networks”, SoftCom 2002, p 209-213, 8-11 October 2002, Split, Kroatia.

1.3.3 Chapters in International Publications

1. **P. Van Heuven**, S. Van den Berghe, F. De Turck, P. Demeester, “Wiley encyclopedia of technology (MPLS section)”, Wiley and Sons, ISBN 0-471-36972-1, December 2002.

1.3.4 National Journal/Conference papers

1. **P. Van Heuven**, S. Van den Berghe, T. Aernoudt, P. Demeester, “Flexible and service aware networks with DiffServ and MPLS”, 1st FTW PHD Symposium, Interactive poster session, paper 88 (Proceedings available on CD-Rom), 5 December 2000, Gent, Belgium.

2. S. Van den Berghe, J. Coppens, **P. Van Heuven**, F. De Turck, P. Demeester, “Traffic Engineering For Large Scale QoS provisioning”, 3rd FTW PHD Symposium, Interactive poster session, 11 December 2002, Gent, Belgium.

Chapter 2

MPLS technology and applications

The MPLS technology plays an important role in this work because it offers additional functionality to IP networks. This additional functionality can be used to implement advanced failure convergence schemes that are not feasible in pure IP. In order to explain these convergence schemes quite detailed knowledge about the MPLS technology is required. Therefore, this section will introduce the MPLS technology and its applications. The first section of this chapter will give a quick overview of the most important concepts of IP and MPLS forwarding. Subsequent sections will focus on more advanced topics covering the forwarding, the signalling and the applications of MPLS. The majority of this chapter has been published in [2].

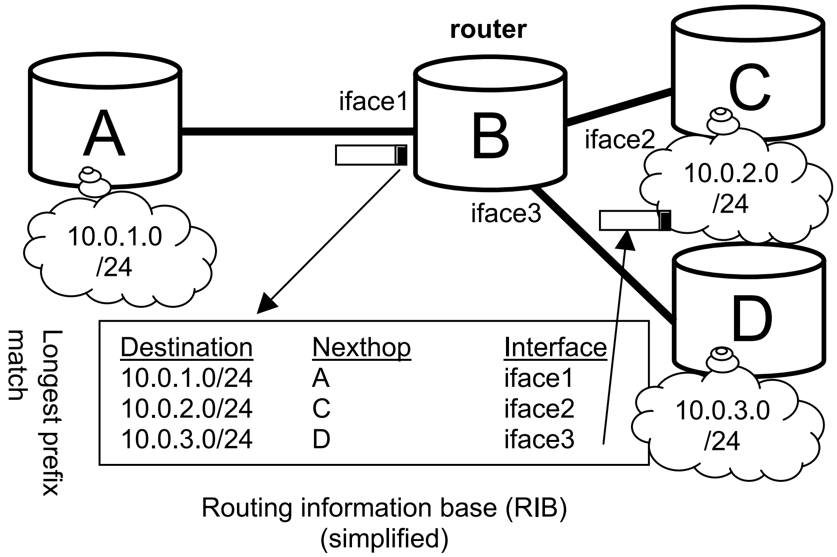
2.1 Introduction

The Internet Protocol (IP) is a connection-less networking layer protocol used to route IP packets over a network of IP routers [9, 10, 11, 12]. Every router in the IP network examines the IP packet header and independently determines the next hop based on its internal routing table. A routing table contains information about the next hop and the outgoing interface for the destination address of each IP address. In MPLS networks Label Switched Paths (LSPs) are set up between IP routers to avoid this IP forwarding in the intermediate routers. In MPLS IP packets are tagged with labels. The initial goal of label based switching was to increase the throughput of IP packet forwarding [13, 14, 15]. Label based switching methods allow routers to make forwarding decisions based on the contents of a simple label, rather than by performing a complex route lookup based on destination IP address. This initial justification for MPLS is no longer perceived as the main benefit, since nowadays routers are able to perform route lookups at sufficiently high speeds to support most interface types. However, MPLS brings many other benefits to IP-based networks, they include: traffic engineering, i.e., the optimisation of traffic handling in network [16, 17], Virtual Private Networks (VPNs), networks that offer private communication over the public Internet using secure links [18, 19], the elimination of multiple protocol layers [20, 21, 22] and improved failure convergence [3].

MPLS paths are constructed by installing label state in the subsequent routers of the path. Labels are fixed length entities that only have a local meaning. Labels are installed with a label distribution protocol. The MPLS forwarding of the IP packets is based on these labels. In the next subsection the IP and MPLS forwarding principles will be detailed followed by a subsection on the MPLS label distribution process.

2.1.1 Forwarding in IP and MPLS networks

In regular, non-MPLS, IP networks packets are forwarded in a hop by hop manner. This means that the forwarding decision for a packet traversing the network is based on the look up of the destination address in the local routing table (also called Routing Information Base, RIB). Figure 1 illustrates a network consisting of four routers: A, B, C and D. A simplified IP routing table of router B is shown. It consists of entries which map the destination network addresses of the IP packets to the IP addresses of the next hop and the router interface, which is connected to the next hop. When forwarding a packet, a router inspects the destination address of the packet (found in the IP header), searches through its local router table via a longest prefix match and forwards it to the next hop on the outgoing interface.



Destination	Nexthop	Interface
10.0.1.0/24	A	iface1
10.0.2.0/24	C	iface2
10.0.3.0/24	D	iface3

Figure 1: IP forwarding in a network consisting of four routers and three subnetworks

The destination addresses in this table are aggregated in order to reduce the number of entries in this table. These entries are aggregated by indicating the length of the significant part of the destination addresses (from 0 to 32 bits). If *n* is the length of address *a* then only the first *n* (most significant) bits of *a* are considered. The resulting partial address is called a prefix and is noted as a/n (e.g. 10.15.16.0/24). This aggregation of addresses has the drawback that searching through the table needs to be done with a *longest prefix* match. A longest prefix match is more complex than an exact match because the result of the search must be the entry with the longest prefix that matches the address [12].

An important characteristic of IP forwarding is that packets arriving at a router with the same destination prefix are forwarded equivalently over the network. A class of packets that can be forwarded equivalently is a Forwarding Equivalence Class (FEC). Because of the destination based forwarding of IP, FECs are usually associated with IP prefixes. The forwarding in an IP router can be restated as the partitioning of packets in FECs and assigning a next hop to each FEC. It is important to note that the determination of the FEC needs to be done in every hop for every packet and that it is based on the routing table.

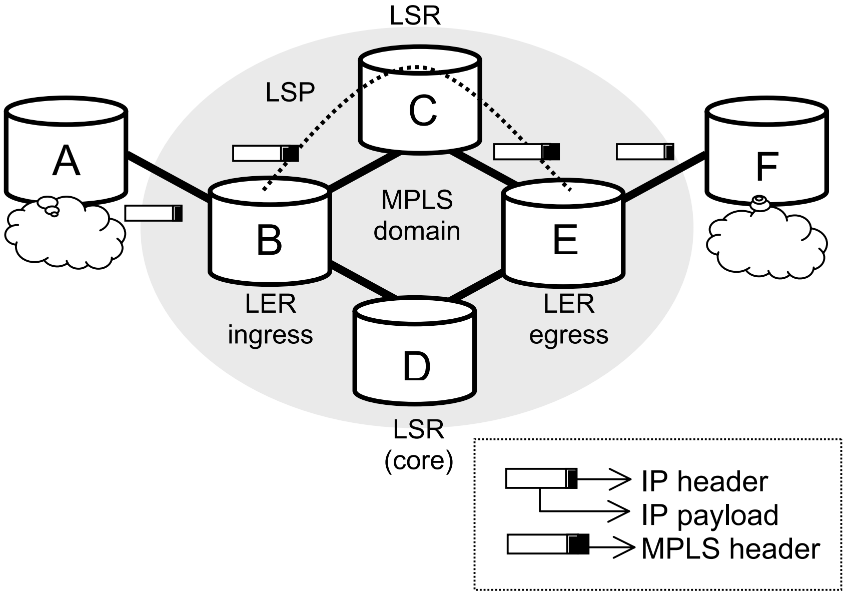
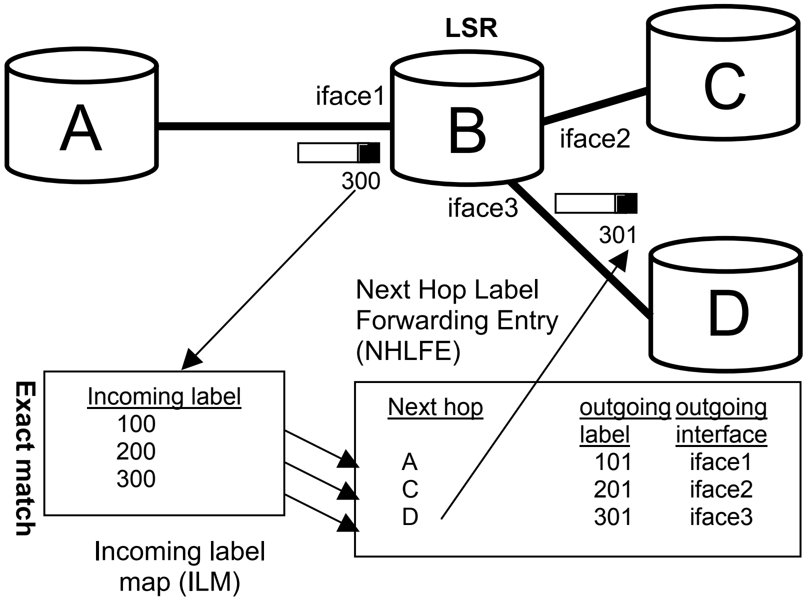


Figure 2: An MPLS domain that connects two external networks

On the other hand, MPLS forwarding relies on labels instead of prefixes to route packets through the network [23]. Labels are fixed length entities that have only a local meaning. Because a label only has a local meaning it can be different at every hop and therefore needs to be adapted before forwarding the packet. The process where the label is changed at every hop is called *label switching*. The labels are distributed over the MPLS domain by means of a label distribution protocol. MPLS routers are called Label Switching Routers (LSR) (see Figure 2) because they operate on labels rather than on IP prefixes when forwarding packets. The concatenation of these installed labels in the different LSRs is called a Label Switched Patch (LSP). An LSP is set up between the ingress LSR and the egress LSR, these edge LSRs are also called Label Edge Routers (LER). Label Edge Router is indeed a more correct name for a router that does not switch on labels but rather adds (ingress) or removes (egress) labels, however the term LSR is often used for both LERs and LSRs. We will use the term LSR to encompass both LSRs and LERs.

In MPLS packets belonging to a FEC are mapped on an LSP. Determining the FEC of a packet is only necessary at the ingress of the LSP. The segregation of packets in FECs needs only be done once, in the ingress router, and this segregation can also be based on more than the destination prefix of the packet. For example, it is possible to take both the source and the destination into account. Because LSPs are based on FEC-to-label binding makes that an LSP is a unidirectional path.



Incoming label
100
200
300

Next hop	outgoing label	outgoing interface
A	101	iface1
C	201	iface2
D	301	iface3

Figure 3: MPLS label switching in a network consisting of four Label Switch Routers

In the case of label switched routers, every router contains two tables: an Incoming Label Map (ILM) table that contains all the incoming labels the router has allocated and a table that contains all the necessary information to forward a packet over an LSP (see Figure 3). The latter table is populated with Next Hop Label Forwarding Entries (NHLFE). There is a mapping between the ILM and an NHLFE, mapping the incoming label to an output label, the outgoing interface and the next hop [24]. The router inspects the incoming label and consults the ILM table to find the right NHLFE. Before the packet is sent to the next hop, the label is switched to the outgoing label value.

Now that we have investigated how packets are forwarded via label switching in MPLS networks we will continue the discussion investigating how these labels are installed in the routers of the network.

2.1.2 Label distribution in MPLS networks

There are two different types of LSP, hop-by-hop routed LSPs or shortest path LSPs (SP-LSPs) that are set up according to the shortest paths in the network and explicitly routed LSPs (ER-LSPs) that are set up according to an arbitrary path specified during the setup. How these two types of LSPs are established is the topic of this subsection. However, we will not describe the label distribution process in its full detail nor will we discuss every possible distribution mode or distribution protocol. The goal of this section is to give a general overview, more details will be added in the subsequent sections of this chapter.

2.1.2.a Setting up a shortest path LSP

To distribute labels over the network and consequently set up an LSP a *label distribution protocol* is used. The LSP setup procedure typically consists of two steps: a request is sent to the egress of the LSP and then the response propagates back to the ingress. The first step will be denoted by the generic term “Label Request” whereas the second step will be denoted by the term “Label Mapping”. Figure 4 illustrates the label distribution process in a simple four node network. When LSR A wants to set up an LSP to network *netD*, it will send a Label Request to its next hop towards netD (step a, Figure 4). This next hop is determined by the local routing table of node A. The intermediate nodes from the ingress towards the egress (i.e. LSR B in the figure) will install state about the request and will then forward the request towards D according to their routing information base (step b). When the request reaches the destination of the LSP, the egress, this node will allocate a label for the LSP and stores this information in the Incoming Label Map (ILM). The LSR will then send a Label Mapping back to the previous hop. The Label Mapping message contains the label previously allocated by the LSR (step d). LSR B will then receive the Label Mapping from node D.

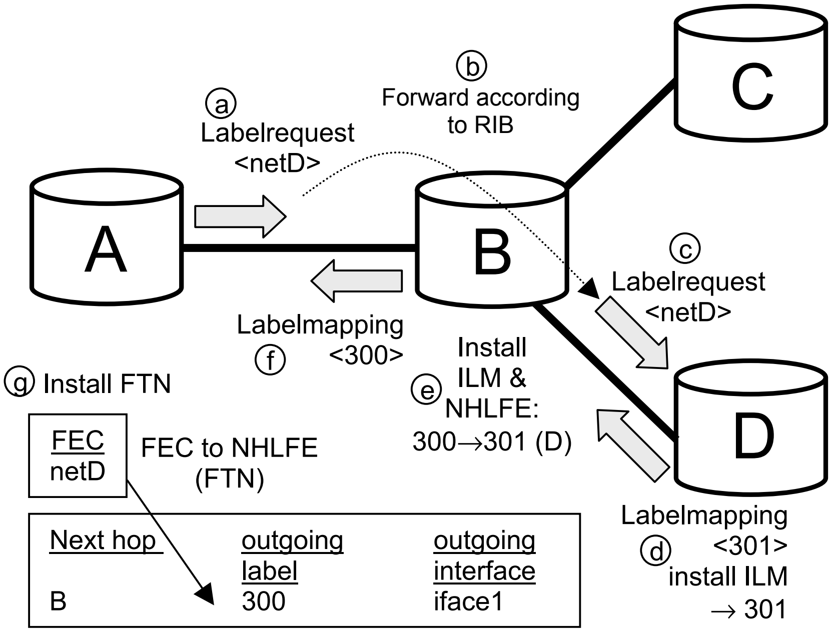


Figure 4: Example of hop-by-hop routed label distribution

The label contained in the Label Mapping will be used to make a Next-Hop Label Forwarding Entry (NHLFE). B will then, in its turn, allocate a label and store this label in its ILM. The information in the ILM (incoming label) and the NHLFE (outgoing label) is combined, effectively storing the information about the label switch (step e). After allocating the label and storing the relevant information, LSR B will send a Label Mapping to its previous hop (step f). Finally, the initiator of the LSP setup (node A) will receive the Label Mapping from its next hop. LSR A will store this information in a NHLFE. This ingress LSR will then map traffic to the newly established LSP by mapping a class of packets (FEC) to the LSP, which implies that traffic that belongs to this traffic class will be forwarded over the LSP. The FEC is thus mapped on the NHLFE (step g). All the FEC to NHLFE mappings are stored in the FEC to NHLFE map (FTN). The FTN is used by the ingress of an LSP to forward the packets belonging to a certain FEC over the LSP.

Because the request is forwarded according to the local RIB of the intermediate routers, the resulting LSP is called a hop-by-hop routed LSP. Another type of LSP is called an explicitly routed LSP (ER-LSP).

2.1.2.b Setting up an explicitly routed LSP

The real power of MPLS lies in the fact that paths can be set up with a great deal of flexibility. An example is an Explicitly Routed LSP (ER-LSP). Explicitly Routed means that some or all of the hops between the ingress and the egress of the LSP can be specified. Figure 5 illustrates this, LSR A sends a Label Request for *netD* and the Label Request explicitly states that the LSP should be routed along node C (step a). Node B will receive this Label Request and will forward it towards C although the shortest path towards netD is via node D (step b). When LSR C receives this Label Request it will remove itself from the Hop List in the Label Request and forwards the Label Request towards the destination. From then on the LSP setup continues as detailed in Figure 4. It is important to note that every node keeps state about the Label Request so that the Label Mappings can be sent to the correct previous hop, i.e. the hop it received the corresponding Label Request from.

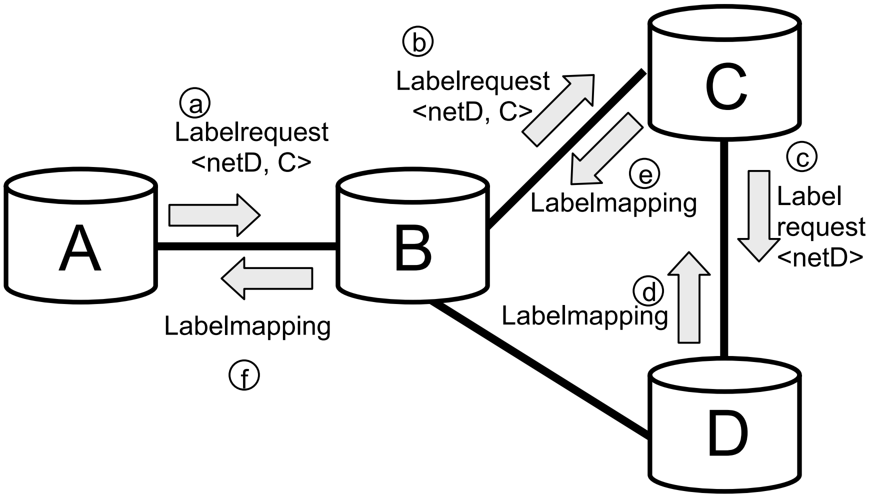


Figure 5: Example of explicit label distribution

2.1.3 Separation of forwarding and routing

In this work we will explicitly make distinction between forwarding and routing. Forwarding is the (data plane) operation where a packet is transferred over the network by the routers inside the network. Routing is a (control plane) operation that determines the paths that are used during forwarding.

In IP both the routing and the forwarding are based on the destination address. MPLS separates the forwarding from the route calculation by using labels to forward the packets. To distribute these labels over the domain and hence set up an LSP, MPLS uses a Label Distribution Protocol. LSPs can be set up according to the IP routing tables or the hops to be traversed can be explicitly specified. It is this separation of forwarding and routing in MPLS that enables most of the important applications of MPLS.

Another, though smaller, benefit of this clear separation of the forwarding and the routing/control parts of MPLS is that the discussion of them can easily be done independently too. First we will investigate the MPLS forwarding in more detail and then we will discuss the control architecture. However we finalise the introduction section with a brief look at the history of MPLS.

2.1.4 History

MPLS started out as a technique for IP over Asynchronous Transfer Mode (ATM [64, 65, 66, 67]) inter-working as a convergence of a number of “IP switching” schemes. IP switching is a technique that uses ATM hardware to forward IP packets. In contrast to ATM networks, in IP switching and MPLS networks the ATM hardware is administrated by IP and MPLS signalling protocols rather than ATM signalling.

There are, or rather were, a number of different IP switching implementations [13, 68]: Cisco Systems Tag Switching [69], IBM’s Aggregated Route based IP Switching (ARIS) [70], Toshiba’s Cell Switch Router (CSR) [71], Ipsilon Flow Management Protocol (IFMP) [72] and NEC's Ipsofacto [20, 73]. In order to standardise all these IP switching techniques a new IETF working group came to life back in 1997. The MPLS working group has since then been working on forming a common technology for IP switching [74].

MPLS contrasts with a number of other techniques for IP over ATM which are overlay techniques. Examples of overlay techniques are Multi-Protocol Over ATM (MPOA) [75, 76] and the work done in the IETF Internetworking Over Non-broadcast multiple access (ION) working group [77]. In ATM overlay networks there are two distinct networks: an ATM network and an IP network. This leads to the disadvantage of having to administer the two networks and the fact that the scalability is limited due to full meshed peering [15, 21, 78].

After this small overview of the history of MPLS we will address the forwarding layer in MPLS.

2.2 The forwarding layer of MPLS

This section starts with a summary of the most important MPLS forwarding concepts. After that it continues with a discussion about the other important issues and terminology with respect to the MPLS forwarding.

2.2.1 MPLS forwarding concepts

As we have seen in section 2.1.1, the MPLS architecture formalises three concepts with respect to the forwarding plane. First the Next Hop Label Forwarding Entry (NHLFE) is an entry that contains all the information in order to be able to forward a packet on a MPLS router. It contains the packet's next hop and the outgoing label. The NHLFE may also contain the data link encapsulation and the way to encode the label stack when transmitting the packet (we will address the label stack in the next section).

The Incoming Label Map (ILM) defines a mapping between an incoming label and one or more NHLFEs. It is used when forwarding labelled packets. If the ILM maps a particular label to more than one NHLFE exactly one NHLFE must be chosen before the packet is forwarded. Having the ILM map a label to more than one NHLFE can be useful to do e.g. load balancing over a number of LSPs. Since the ILM is used to forward *labelled* packets in a LSR it is typically used in a core LSR.

Finally the FEC-to-NHLFE Map (FTN) maps a FEC to one (or more) NHLFEs. It is used when forwarding packets that arrive unlabelled, but which are to be labelled before being forwarded. The FTN map is used in the ingress Label Edge Router.

2.2.2 Label merging

In order to reduce the number of outgoing labels, different incoming labels for the same FEC can use the same outgoing label. The LSR then merges different incoming labels to a single outgoing (label merging).

A link layer is capable of merging labels if more than one incoming label can be mapped to a single outgoing label. A merge-capable LSR only needs to store label information of every FEC. However some link layer technologies are not capable of merging labels (some ATM and Frame Relay implementations). In this case a one-to-one mapping between incoming and outgoing label is required. This has scalability implications because individual LSP state is needed for every ingress-egress pair traversing a given LSR (unless label stacking is used, see section 2.2.6).

2.2.3 Label spaces

Another point of difference between some link layer technologies is the scope of the labels. The MPLS architecture uses the term label space. Label spaces limit the scope of a label which means that the label is only valid and defined for a given label space. Two labels only match if both the label value and the label space are equal. This has the consequence that the same label value can be reused in a different label space. ATM and Frame Relay have a label space per interface which means that label A on interface 1 will be interpreted differently from label A on interface 2. On the other hand platform wide label spaces are not tied to a specific interface but are valid on the whole platform (i.e. router or host). Global label spaces have the advantage that if the incoming interface of an LSP changes (e.g. during rerouting) no action needs to be taken. Per interface label spaces reduces the number of used incoming labels per interface which is useful if labels are a scarce resource like in some ATM and Frame Relay implementations.

2.2.4 Penultimate-hop popping

Labels are used in MPLS to make forwarding decisions. However, no forwarding decisions have to be made at the last hop of an LSP, it only needs to send the packet towards the upper layer stack. This means that the next to the last hop, also referred to as the penultimate hop, can pop the top label of the LSP. When the penultimate hop pops the label and sends the packet without a label to the egress node this is called *penultimate-hop popping (PHP)*. Penultimate-hop popping has the advantage that there is no label overhead over the last link of an LSP. Another advantage of PHP is that it can simplify the MPLS forwarding operations in some circumstances as we will explain when we discuss the MPLS label operations and label stack in section 2.2.6.

2.2.5 Label encapsulation

It is apparent that “labels” constitute the centre of the MPLS architecture. However, the properties of the labels differ from the link layer technology on which MPLS is supported. Because of this close tie with the link layer technology, MPLS is sometimes called a layer 2.5 technology, situated between the link layer (layer 2) and the networking layer (layer 3) [79].

Two categories of data link layers can be distinguished, link layers that natively support fixed length entities and switch upon them and others that do not have such an entity [80]. Examples of such fields are the Virtual Circuit Identifier (VCI) field or Virtual Path Identifier (VPI) field or both fields (VPI/VCI) of ATM [35] (see Figure 6c). A second example is the Data Link Circuit Identifier (DLCI) from Frame Relay (FR) [81, 82] (see Figure 6b).

The second category, link layer technologies that do not natively support labels, encapsulate the MPLS labels by transmitting an additional header. This small header, called the *shim header*, is inserted between the link layer header and the networking header. The former way of encapsulating the MPLS labels is called link layer specific encapsulation, whereas the latter is called generic MPLS encapsulation. The shim header contains a label, three experimental bits, a bottom of stack indicator (BoS) and a Time-To-Live (TTL) field (Figure 6a). The *label* field (20 bits) is used to store the label value, the three experimental (*EXP*) can be used to support DiffServ over MPLS (DiffServ will be covered in detail in section 2.5.2) and/or Early Congestion Notification (ECN) [83] or other experimental extensions to MPLS forwarding. The *Bottom of Stack* bit is used to indicate the last shim header of the label stack (see section 2.2.6). Finally the *TTL* field is used to support the IP time-to-live mechanism (we refer to section 2.2.7).

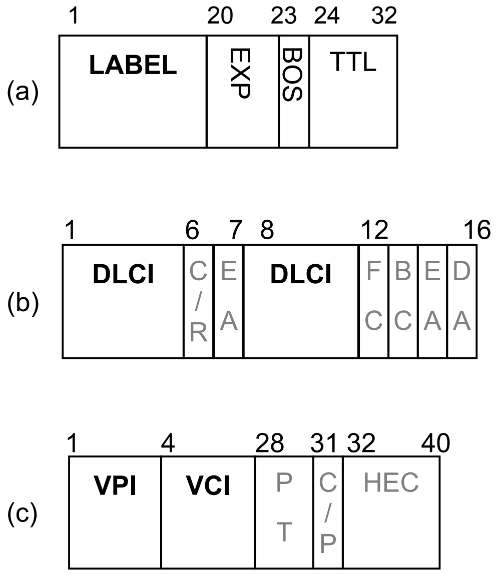


Figure 6: MPLS label encapsulation: a) generic MPLS encapsulation with the shim header, b) encapsulation in the Frame Relay DLCI, c) encapsulation in the ATM VPI/VCI

An example of a link layer technology that has a native label entity is ATM. In ATM based MPLS a label is a VPI, VPI/VCI or a VCI identifier. In ATM networks these identifiers are installed with User-to-Network Interface (UNI) [84] or Private Network-Node Interface (PNNI) signalling [85]. MPLS does not use ATM signalling because a label distribution protocol is used instead. When link layer specific label encapsulation is used the label stack is still encoded in the shim header but the shim header cannot be processed by the intermediate LSRs (only at the ingress and the egress). Therefore, the top label of the label stack is copied to the native label entity before the ATM or FR segment.

Similarly after the segment the current label value from the native label is copied to the top label of the label stack.

2.2.6 Label operations and label stacks

We have seen that the ingress LSR of an LSP sets the outgoing label in the packets before it forwards it. This label is then swapped by the next hop of the LSP. The labels continue to be swapped until the packet reaches the egress of the LSP where the label is removed and the packet is further processed.

The MPLS architecture defines a limited number of operations on MPLS labels: to replace the top label with a new label (label swap), to remove the top label (label pop) or to replace the top label and push a number of new labels [23]. The label swap operation is used by the core routers to replace the incoming label before forwarding a packet. The label pop operation is used by the egress of the LSP to remove the label before processing it further. The third operation (replace and push label) can be used by the ingress of an LSP to push a single label before forwarding the packet. However, the above definition also allows that more than one label is pushed. This means that multiple labels can be pushed on top of each other leading to a *label stack*. A label stack can be used to aggregate multiple LSPs in one top-level LSP (see Figure 7).

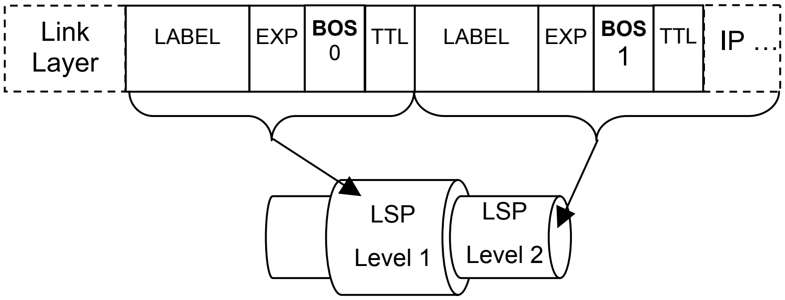


Figure 7: Using a label stack to aggregate LSPs

All operations on the label stack always occur on the top of the label stack so a node does not have to investigate the whole label stack as part of the MPLS forwarding operations. This also means that if an LSP is aggregated in another LSP then only the aggregated LSP is visible until the LSP is de-aggregated.

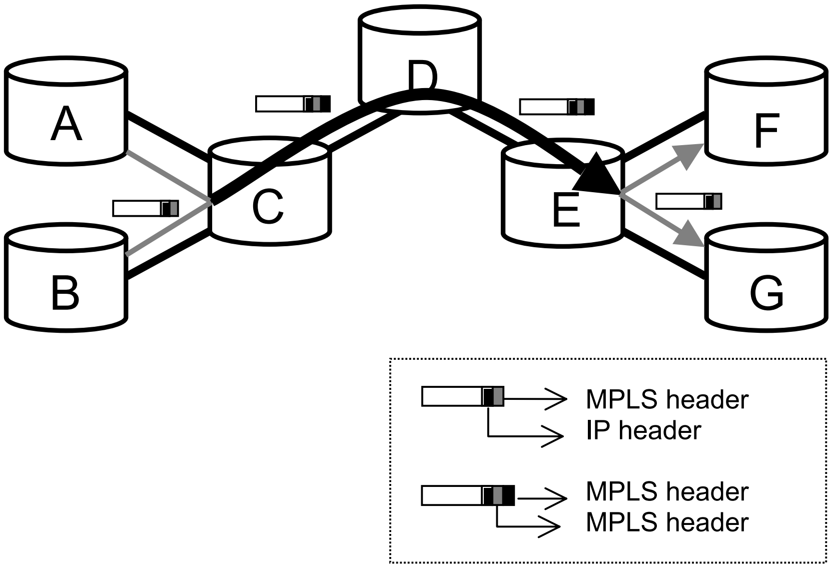


Figure 8: Aggregating LSPs in MPLS. The LSPs A-C-D-E-F and B-C-D-E-G are aggregated in the LSP C-D-E.

For example in Figure 8 the LSPs A-C-D-E-F and B-C-D-E-G are aggregated in the LSP C-D-E. We will now explain the forwarding over the aggregated LSP B-C-D-E-G. Node B pushes the outgoing label for the LSP B-C-D-E-G on the label stack and forwards it to node C. Node C removes the top label from the label stack and adds two labels. The first label is the outgoing label for the aggregated LSP B-C-D-E-G. Since the aggregated LSP is not visible over the aggregating LSP C-D-E the label should be the incoming label for node E. The second label that is pushed on the label stack is the outgoing label for the aggregating LSP C-D-E. Afterwards node C forwards the packet towards node D. Node D will inspect only the top label and swaps it with the outgoing label for the LSP C-D-E. Node E then pops the top label because the aggregating LSP ends here and swaps the incoming label with the outgoing label for the aggregated LSP so that the packet can be further forwarded towards node G.

Note that node E has to perform two actions, first pop the label of the aggregating LSP and then swap the label of the aggregated LSP. This is solved by penultimate-hop popping (see section 2.2.4). When penultimate-hop popping is used node D pops the label of the aggregating LSP because it is the second to last hop for this LSP. The label stack at node E then only contains the incoming label for the aggregated LSP. Node E then simple swaps this label before forwarding the packet towards node G.

As explained label stacking can be used to reduce the label state in core nodes, it is also used in MPLS VPNs (see section 2.6.3) and the local loop-back protection switching approach (see section 3.3.4.e).

2.2.7 Supporting the IP Time-to-live field

In regular IP networks, the time-to-live (TTL) field in the IP header is decremented at every hop. When the TTL reaches zero the packet is dropped. This mechanism is used to prevent packets from being forwarded forever in case of a network anomaly (e.g. a network loop). To support this mechanism in MPLS the TTL information must be available to the LSRs. Since the shim header contains a TTL field the LSRs that use generic MPLS encapsulation are able to decrement the TTL just like in regular IP forwarding.

When MPLS is supported by a link layer technology that uses its own native label entities, the LSR can only act on these native labels. Unfortunately, these link layer specific MPLS labels do not have a TTL field. It is therefore impossible to decrement the TTL at every hop. The solution is to compute the total TTL decrement for the non-TTL capable segment (the switching segment) at LSP setup time. When a packet arrives at the first hop of this segment the pre-computed TTL decrement is subtracted from the current TTL. If the result is positive then the TTL is written to the top entry of the label stack. A packet travelling through the LSP will have the correct TTL before the non TTL-capable segment and after that segment. The TTL will have a constant value in the segment (the same value as just after the segment). If there is no information about the hop count of the non-TTL capable segment the current TTL is decreased with one and the resulting value is written to the label stack. In any case appropriate actions must be taken on this result (e.g. dropping the packet if the TTL reaches zero) [86].

2.3 Label distribution modes

In the previous sections we investigated some important properties of labels and the label forwarding operations in MPLS. In this section we will examine how these labels are distributed over the network. The MPLS architecture allows for multiple methods for distributing labels. We already gave a basic overview of the label distribution process in Figure 4. The subsequent sections will describe the most important label distribution protocols for MPLS but we will introduce some important terminology first [23].

2.3.1 Down stream versus upstream allocation

Labels only have a local scope which means that they are defined in one node, if platform wide label spaces are used, or over one interface, if per interface label spaces are used. This has the advantage that allocation of the labels does not need to be controlled centrally.

For a given LSP, a core LSR has an upstream and a downstream neighbour. So in order to forward a packet with label switching it needs an incoming label and an outgoing label. With that in mind there are two possible approaches: the LSR supplies its upstream neighbour with a label and receives a label from its downstream neighbour or the other way around. When the LSR receives the label from his downstream neighbour this is called *downstream allocation*. MPLS typically uses downstream allocation.

2.3.2 Unsolicited distribution versus distribution on demand

As described in the previous section the labels are typically chosen (allocated) by the downstream LSRs. When these labels are distributed spontaneously (without request) this is called *unsolicited label distribution*. When the upstream LSR always sends a request to its downstream neighbour in order to obtain a label this is called *distribution on demand*.

Unsolicited label distribution has the advantage that it introduces less signalling overhead. However, distribution on demand is far more powerful than unsolicited distribution because it allows the ingress of the LSP to control certain parameters of the LSP. For example we explained in Figure 5 how explicit routedly LSP are set up with distribution on demand. We will examine more parameters that can be controlled by the ingress of an LSP when we discuss the different label distribution protocols.

2.3.3 Independent versus ordered control

When a regular IP router has calculated the shortest paths inside the network it knows the next hop towards each FEC in the network and installs a routing table entry for each of them. Although this process is driven by the routing protocol each router decides independently when to install a routing table entry.

Similarly in MPLS, when using *independent control* an LSR makes an independent decision to map and distribute a label when it recognises a new FEC. In *ordered control* with downstream label allocation the label mapping decision is co-ordinated by two simple rules: an LSR only maps a label to a particular FEC if it is the egress for that LSP or if it has already received a label binding from its next hop. This means that when ordered control is used together with downstream label allocation that the labels are allocated from egress to ingress. In Figure 4 and Figure 5 we illustrated ordered control.

2.3.4 Liberal retention versus conservative retention

Consider the situation where an upstream LSR has received and retained a label mapping from its downstream peer. When the routing changes and the original downstream peer is no longer the next hop for the FEC then there are two possible actions the LSR can take. The LSR can release the label, this is called *conservative retention* or it can keep the label for later use, which is called *liberal retention*.

Conservative retention (*release on change*) has the advantage that it uses fewer labels, liberal retention (*no release on change*) has the advantage that it allows for faster reaction to routing changes because it may already have a label over the new route.

We will return to the topic of retention modes when we discuss rerouting in MPLS (section 3.3.3.a).

2.3.5 Label use method

Labels can be used as soon as a LSR receives them (*use immediate*) or the LSR can only use a certain label if the corresponding LSP contains no loop (*use loop free*). MPLS supports both loop prevention (prevent an LSP with a loop from being set up, [34]) and loop detection mechanisms (detect if an LSP contains a loop, [35]).

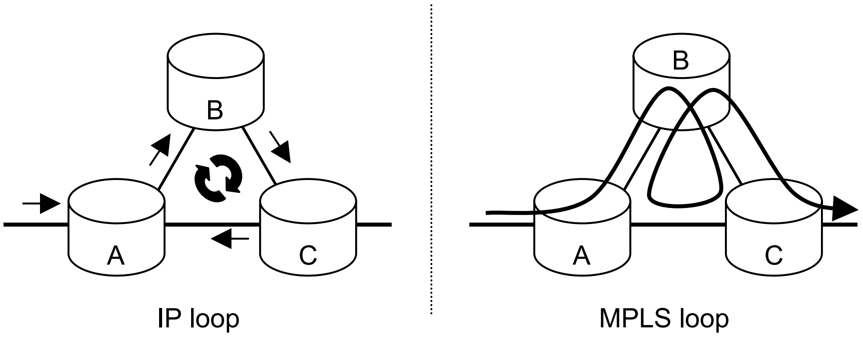


Figure 9: Loops in IP and MPLS. In IP when a packet enters a loop it will travel the loop until it is discarded or until the loop is removed. In MPLS, the effects of looping are typically limited where a packet traverses the loop only once.

It is important to illustrate the difference between a loop at the IP level and a loop at the MPLS level. Since the forwarding in IP is typically only based on the destination there is no way to leave an IP loop. For example, node A will always forward the packet towards node B whether it comes from inside or outside the loop (see Figure 9). The IP TTL mechanism is responsible to discard the packet after a number of iterations over the loop. In MPLS it is possible to set up an LSP which contains a loop. Unlike in IP this does not mean that the packets will travel forever in this loop rather they will simply leave it after one iteration because the forwarding is based on labels and not on the destination address.

MPLS loop prevention and detection are typically used in combination with shortest path LSPs which are routed according the IP routing protocols. Loop prevention and detection are not commonly used on explicitly routed LSPs. We will assume that the labels are used immediately so without loop prevention and detection.

2.4 The Label Distribution Protocols

In this section we will describe different protocols for distributing the labels over the MPLS network. We will start with the basic Label Distribution Protocol (LDP).

2.4.1 The Label Distribution Protocol (LDP)

The Label Distribution Protocol (LDP) [25, 26, 27] is the basic signalling protocol proposed by the IETF MPLS working group for hop-by-hop routed LSPs. In LDP labels are distributed for a given FEC. In LDP a FEC can be either an IP prefix or an IP host address. LDP gives the user a great deal of freedom how to set up the LSPs. LDP peers use TCP as the transport protocol for LDP messages. This ensures that these messages are reliably delivered and need not be refreshed periodically (LDP is therefore called a hard state protocol).

LDP discovery is a mechanism that uses *Hello messages* to enables potential LDP peers to discover each other (most of the time neighbours). When two LDP peers have discovered each other by exchanging Hello messages over a link they have built up a Hello adjacency. The Hello adjacency is monitored by exchanging Hello messages regularly.

The next step is to alter the formation of one or more Hello adjacencies between two LDP routers is to set up a LDP session between them. LSP session management allows LDP peers to negotiate about session parameters (e.g. on demand or unsolicited distribution). After this negotiation phase a LDP session is set up and the LDP messages can be exchanged (e.g. label request and label mapping messages for the distribution of the labels). The LDP session is monitored by requiring that LDP messages are regularly exchanged. If there are no messages to be exchanged over a period of time then a *KeepAlive* message is sent. The KeepAlive message is a dummy LDP message with the sole purpose of making sure that the LDP session is not torn down due to a lack of LDP messages.

Note that both the Hello and the KeepAlive messages are necessary because LDP is a hard-state protocol which means that if the MPLS related state does not change then there will be no signalling activity. This is obviously very efficient in terms of signalling overhead but it has the drawback that failures in the signalling layer are hard to detect. By requiring that Hello and LDP messages are regularly exchanged one creates a minimum signalling activity that can be used to monitor the adjacencies and LDP sessions.

LDP supports unsolicited and on demand distribution of labels, liberal and conservative label retention, independent and ordered control and the immediate or loop free use of labels (Figure 4 illustrates ordered control, downstream on demand label distribution).

Information in LDP messages is encapsulated in Type-Length-Value (TLV) structures. These TLVs are used for standard features but can also be used to extend LDP with experimental and/or vendor-private mechanisms. Constrained based Label Distribution Protocol (CR-LDP) is an extension to LDP and will be covered in the next subsection.

2.4.2 Constrained based Label Distribution Protocol (CR-LDP)

CR-LDP introduces a number of extensions to LDP in order to support MPLS traffic engineering [28, 29]. CR-LDP only supports the downstream on demand ordered label distribution. The additional functionality of CR-LDP compared to LDP is the possibility to set up constrained based LSPs, the support for traffic parameters, pre-emption and resource classes. We will now describe each of them in turn.

2.4.2.a Constrained based routes

CR-LDP allows to set up LSPs that defer from the shortest path by explicitly indicating the hops the LSP should traverse. There's a distinction between strict and loose hops. A strict hop on an LSP means that the next hop along the LSP should be that hop and that no additional hops may be present. A loose hop on an LSP simply requires that the hop is present on the path the LSP traverses. Hops are ordered which means that they should be traversed in the order they are specified.

For example to set up an LSP from node A to E one can just specify the destination E (Figure 10). This will result in a LSP from A over the shortest path B-C-D towards E. It is also possible to specify the LSP with node F as loose hop.

That will result in the LSP A-B-F-G-D-E or LSP A-B-F-C-D-E. Loose hops and strict hops can also be mixed. For example if we want to force the traffic over the B-F-G-D segment one could specify the loose hop F followed by the strict hop G. Another alternative is to specify all the hops on the LSP i.e. B-F-G-D-E. The hops can be strict or loose in this case. When all the hops of the LSP are specified, we call this a fully specified (strict or loose) ER-LSP.

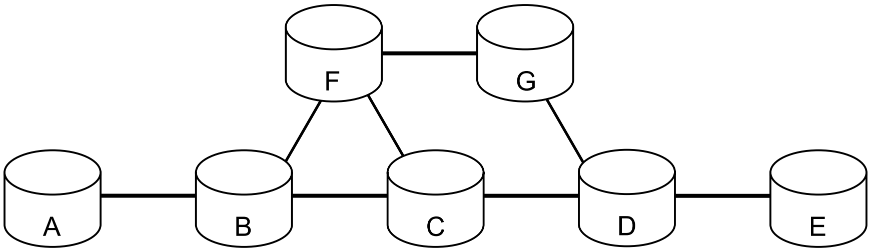


Figure 10: Example topology, when no hops are specified an LSP from node A to node E will be set up according to the shortest path.

CR-LDP supports the notion of abstract nodes. An abstract node is a group of nodes whose internal topology is opaque to the ingress node of the LSP. An abstract node can for example be denoted by an IP prefix or an Autonomous System number (AS).

2.4.2.b Traffic parameters

The traffic parameters of an LSP are modelled with a peak and a committed rate. The peak rate is the maximum rate at which traffic should be offered to the LSP (the offered rate). If resource allocation within the MPLS domain depends on the peak rate value then it should be enforced at the ingress of the MPLS domain.

The committed rate is the rate that the MPLS domain commits to the LSP. The extent by which the offered rate may exceed the committed rate is determined by the maximum excess burst size. There can also be a weight associated with the LSP; this weight indicates the relative share of the available bandwidth the excess bandwidth of the LSP receives.

Finally the frequency of an LSP indicates the granularity at which the committed rate is made available. It constrains the variable delay that the network may introduce and thus it constrains the amount of buffering that a LSR may use. There are three possible granularities: unspecified, frequent and very frequent. Very frequently means that no more than one packet may be buffered. Frequently means only a few packets may be buffered and unspecified means any amount of buffering is acceptable

CR-LDP also provides support for DiffServ over MPLS, as will be described in section 2.5.2.

2.4.2.c Pre-emption

An LSP can have two priorities associated with it: a setup priority and a hold priority. The setup priority indicates the relative priority an LSP receives to allocate resources (bandwidth) when set up. A LSP with a higher set up priority can be set up in favour of an existing LSP with a lower priority (it can pre-empt an existing LSP). The holding priority indicates how likely it is for an LSP to keep his resources after been set up.

2.4.2.d Resource classes

In CR-LDP one can specify which of the resource classes an LSP can traverse. A resource class is usually associated with a link so one can indicate which links are acceptable to be traversed by an LSP. Effectively this information allows for the network's topology to be pruned i.e. certain links cannot be traversed by the LSP. For example a provider might want to prevent continental traffic from traversing transcontinental links.

2.4.3 Extensions to RSVP for LSP tunnels (RSVP-TE)

The “Extensions to RSVP for LSP tunnels (RSVP-TE)” protocol [30] is an extension to the Resource Reservation Protocol [31], a signalling protocol originally developed for IntServ reservations [32]. First of all RSVP-TE extends RSVP with the possibility to set up LSPs and adds Traffic Engineering (TE) functionality [33].

2.4.3.a Path set-up

RSVP-TE is based on the RSVP protocol, which does not have support to set up LSPs. RSVP-TE has been extended to support this by introducing a new LSP session type and by defining two new objects: a label request object which is encapsulated in the downstream direction on the RSVP PATH messages and a label object, which is encapsulated in the upstream direction on the RSVP RESV messages. Labels are allocated in the upstream direction by the downstream nodes. In other words, RSVP implements a downstream on demand label distribution protocol. RSVP does not have direct support to detect the failure of a neighbouring node. To address this the Hello protocol has been developed, this protocol allows RSVP to detect the liveliness of its neighbours. RSVP also has a loop detection protocol to prevent setting-up LSPs that contain loops. This makes RSVP more or less functionality wise equivalent to LDP in downstream on demand mode with the important difference that RSVP is a soft-state protocol, this means that the state with respect to the LSP has to be refreshed periodically in the network. The advantage of a soft-state protocol is that the protocol acts more naturally to network changes while it typically requires more signalling overhead.

2.4.3.b Traffic Engineering

Other extensions to RSVP introduce the traffic engineering capabilities very similar to CR-LDP's functionality. Like CR-LDP, RSVP-TE supports the notion of explicitly routed paths whereby the (abstract) hops can be specified strict or loose. The approach to bandwidth and resource allocation differs fundamentally from the CR-LDP model. As mentioned before RSVP is a signalling protocol for IntServ so support for IntServ in RSVP-TE is naturally inherited from the base RSVP protocol. Like CR-LDP it is possible to indicate the setup and holding priority of the LSP. The resource class procedures for RSVP-TE are more powerful than those found in CR-LDP. In CR-LDP a link is eligible to be traversed by an LSP if the resource class of the link is part of the resource classes specified in the label request message. These lead to the procedure where any link can be used as long as the link is part of the resource class collection specified in the label request message. RSVP-TE also supports this include-any relationships between links and LSPs but also supports exclude-any and include-all relationships. It is not necessary to specify any of three relationships, but if set they must match for the link to be taken into account.

2.4.4 Border Gateway Protocol

The Border Gateway Protocol-4 (BGP-4) [42] is used to distribute routes across the Internet. These routes can be Inter-domain routes making BGP the sole Inter-domain routing protocol. By piggybacking label information on the BGP route UPDATE messages, BGP can be used to distribute the label mapped to that route [87]. A simple example of the use of BGP as a label distribution protocol is when two BGP peers are directly connected, then BGP can be used to distribute labels between them. A more important use of BGP as a label distribution protocol is the more common case where the BGP peers are not directly connected but belong to an MPLS domain that supports another label distribution protocol (e.g. LDP). BGP-4 and another label distribution protocol is used to administer MPLS VPNs (see section 2.6.3).

2.5 MPLS and Quality of Service

Although MPLS is not a Quality of Service (QoS) framework, it supports delivery of QoS. The following sections describe how the two major models for QoS in IP (IntServ and DiffServ) are implemented with MPLS.

2.5.1 Integrated Services

The integrated services (IntServ) architecture has the goal to provide end-to-end QOS (in the form of services) to applications. The IntServ QoS model has up to now defined two service types: *guaranteed service* (guaranteed delay and bandwidth [88]) and *controlled load* (QoS closely approximating that of an unloaded network [89]). The architecture uses an explicit setup mechanism to reserve resources in routers so that they can provide requested services to certain flows. RSVP is an example of such a setup mechanism but the IntServ architecture can accommodate other mechanisms. RSVP-TE as an extension of RSVP has natural support for both IntServ service types. An LSP with IntServ reservation is created just like any other IntServ reservation but additionally the MPLS specific LABEL_REQUEST and LABEL objects are piggybacked on the PATH and RESV message respectively.

CR-LDP does not have support for IntServ natively but it can support (a number of) IntServ flows over an LSP by setting the appropriate traffic parameters of the LSP. In order to guarantee the service received on the LSP, admission control and policing on the ingress is required.

2.5.2 Differentiated Services

In order to solve the IntServ scalability problem, Differentiated Services (DiffServ, [90, 91, 92, 93]) classifies packets into a limited number of *classes* and therefore does not need per-flow state or per-flow processing. The identified traffic is assigned a value, a DiffServ code point (DSCP, [94]). A DiffServ Behaviour Aggregate (BA) is a collection of packets with the same DSCP crossing a link in a particular direction. A Per-Hop-Behaviour (PHB), the externally observable forwarding behaviour, is applied to a behaviour aggregate [95].

The classification is usually based on multiple fields in the IP header (Multi-Field, MF classification) at the edge and based on the DiffServ CodePoint (Behaviour Aggregate, BA classification) in the core of the network (see Figure 11c).

An example PHB is Expedited Forwarding (EF, [96]) which offers low loss, low delay and low jitter with an assured bandwidth. This means that the collection of packets marked with the EF code-point traversing a link in a certain direction (BA) will receive low loss, delay, jitter and a assured bandwidth. The Assured Forwarding (AF, [97]) PHB group is a group of PHBs. A PHB of the AF group is noted as AFxy, where x is the class and y is the drop precedence. Packets belonging to a different AF class are forwarded separately. Usually more resources are allocated to the lower classes. Packets within a class that have a higher drop precedence will be dropped before packets with a lower drop precedence.

An Ordered Aggregate (OA) is the set of Behavior Aggregates that share an ordering constraint. This means that packets that belong to the same OA must not be reordered. When looking at DiffServ over MPLS it immediately becomes apparent that packets that belong to a certain OA must be mapped on the same LSP, otherwise this ordering constraint cannot be enforced. This is trivial if only one PHB is applied to the ordered aggregate. However, PHBs can be grouped in a Per-hop-behavior SCheduling group (PSC). A PSC is the set of one or more PHB(s) that are applied to a given OA. For example, AF1y is a PSC comprising the AF11, AF12 and AF13 PHBs. Combining the notion of OA and PSC means that in DiffServ over MPLS OA-PSC pairs will be mapped on LSPs. If the PSC contains more than one PHB this means that it must be possible for an LSR to enforce different PHBs to the packets that belong to the same LSP. This in turn means that the LSR must have some information in the packet header to determine the PHB to be applied, this information must be encapsulated in a way that is accessible to the LSR and thus must be part of the label or shim header. The next subsection will discuss how to encapsulate the PHB.

2.5.2.a Encapsulating the PHB

In IPv4 the “Type of Service” (ToS) field or in IPv6 “Traffic Class” field is used to encapsulate the DSCP. With generic MPLS encapsulation there is a mapping from the IP DSCP space to the EXP field of the shim header [98]. The DSCP field uses the 6 most significant bits of the 8 bits of these IP header fields. Since the DSCP field is 6 bits wide it can comprise 64 different values. However, the EXP field of the shim header is only 3 bits wide so it can only comprise 8 different values. This means that the mapping from DSCP to EXP value cannot be a one-to-one mapping. This is quite a problem because currently there are more than eight defined DSCP values (Best Effort, 12 AF values and EF). If the DiffServ domain uses less than 8 different DSCP values then the mapping between DSCP and EXP can be fixed over the domain. If the domain uses more than eight different code points then the mapping must be explicitly defined on a per-LSP basis.

When the EXP value is used to indicate the set of PHBs applied to an OA (the PSC) then we call this an EXP-Inferred-PSC LSP or E-LSP for short. This means that the PSC is inferred from the EXP value in the shim header (see Figure 11b).

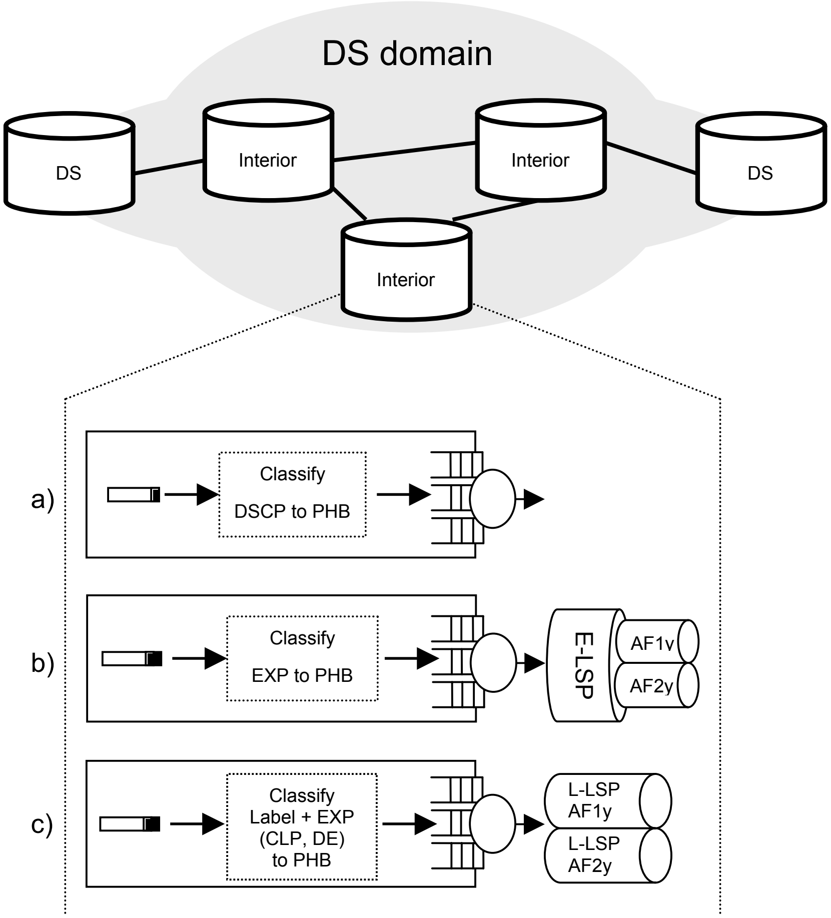


Figure 11: BA classification in DiffServ: a) classification on the DSCP value of the IP header, b) classification on the EXP bits of the shim header, c) classification on the label and the EXP bits (or the ATM CLP or FR DE bit)

This only works for LSRs that support shim headers but link layer specific labels do not have an EXP field. The solution is to set up a distinct LSP for each FEC and ordered aggregate (FEC, OA) pair and signal the PSC during the LSP setup. When the PSC of an OA contains more than one PHB these different PHBs still need to be enforced. The PHBs of the PSC only differ in drop precedence thus we need to encapsulate the drop precedence in the link layer specific label. In ATM only the Cell Loss Priority (CLP) bit can be used to encapsulate this information. Similarly the Discard Eligibility (DE) bit of Frame Relay can be used to encapsulated the drop precedence (shown in Figure 6). An LSP where the PSC is inferred from the label value is called a Label-Only-Inferred-PSC LSP or L-LSPs for short, meaning that the PSC is inferred from the label values as opposed to the EXP field value (see Figure 11c).

The use of L-LSPs is not restricted to link layer specific label encapsulating LSRs, it can also be used with generic MPLS encapsulation. The drop precedence is then encapsulated in the EXP field of the shim header and the PSC is still inferred from the label value.

2.5.2.b Allocation of bandwidth to L-LSP and E-LSPs

Bandwidth can be allocated to E-LSP and L-LSPs at setup time. When resources are allocated to an L-LSP the bandwidth is allocated to the PSC of the LSP, when bandwidth is allocated to an E-LSP then the bandwidth is associated to the whole LSP i.e. the set of PSCs of the LSP. Signalling bandwidth requirements of the LSPs can be useful in two ways. First of all, associating bandwidth to an LSP can be used to admit the traffic to the LSP based on the availability of resources. Secondly, the bandwidth allocation information can also be used to shift resources from certain PSCs to others. It is important to note that allocating resources to an L-LSP or E-LSP still honours the DiffServ principle that per-flow resource treatment is not needed because the resources are associated with the PSC classes.

2.6 Traffic Engineering

Traffic Engineering (TE) is generally defined as the performance optimisation of operational networks. (Other definitions focus more on the role of traffic engineering to offer efficient services to customers.) While TE is not strictly tied to multi-service networks and QoS, TE is definitively more complex and mission critical in multi-service networks. TE is probably considered as the most important application of MPLS networks [16, 17, 1315,99].

The next sections will first address the applicability of TE and then detail how these techniques are implemented in MPLS and how they relate to regular IP implementations.

2.6.1 Applicability

The optimisation of operational networks is typically achieved by the avoidance of congested routes, the resource utilisation of parallel links and routing policies using affinities.

2.6.1.a Avoiding congested routes

When certain network segments are congested while others are under-utilised, the network operator will want to route traffic away from the congested segments. In regular IP this can be done by modifying static link metrics [100, 101, 102] or using dynamic metrics [103, 104] but this is difficult due to the destination based forwarding of IP. In MPLS traffic can be routed away from the congested segments by setting up (explicitly routed) LSPs. Traffic can be mapped on an LSP not only based on the destination but virtually any classification can be used. A small number of LSPs can be used to route traffic away from the congested segments or a mesh of LSPs can be set up to distribute the traffic evenly over the network.

2.6.1.b Resource utilisation of parallel paths

Regular IP only calculates and uses a single shortest path from point A to point B. This limitation is addressed by Equal Cost Multi-Path (ECMP) extensions to routing that takes paths of equal cost into account and spreads the traffic evenly over the available paths with the same cost. Even more advanced is the Optimal Multi-Path (OMP) extension where paths with different cost are used and the traffic is spread according to the relative cost (a path with a higher cost gets a lower share of the traffic). The cost metric of OMP can be dynamic i.e. it can be based on the actual load and length of the path.

MPLS can be used to explicitly configure parallel paths. The calculation can be based on the on-line (routing) mechanisms like ECMP [36] or OMP [105, 106] or alternatively an off-line TE algorithm can compute the paths. Off-line LSP calculations can be based on the measured and forecasted traffic between the edge nodes of the networks (the traffic matrix) [107, 108, 109, 110, 111, 112].

2.6.1.c Routing policies

A network operator might want to exclude some types of traffic from certain links or force traffic on certain links. In MPLS this can be achieved by using the resource class procedures of RSVP-TE or CR-LDP. In IP traffic engineering extensions for OSPF or IS-IS have been defined to cope with resource affinity and routing policy procedures [104, 113].

2.6.2 Implementation considerations

MPLS traffic engineering allows gaining more network efficiency. But there’s no such thing as a free lunch. Efficiency can be gained by introducing more LSPs in the network but there’s a trade-off between the control granularity and the operational complexity associated with a large number of LSPs. In order for the traffic engineering to work properly it is necessary to obtain detailed information about the behaviour of LSPs (LSP monitoring). This is not a trivial task and can require significant resources. The assignment of resources to LSPs and the mapping of traffic to LSPs is another task that can be both time consuming (for the network operator or in terms of computing power) and prone to errors due to inaccurate or outdated traffic matrices. Path calculation is another additional task to perform in comparison with non-traffic engineered networks. Finally, the signalling overhead introduced by traffic engineering can cause additional overhead.

An alternative for a Traffic Engineered network is an over-provisioned network that always has enough capacity to transport the offered load. Even in over-provisioned networks monitoring is necessary to determine when to upgrade the network capacity.

2.6.3 Virtual Private Networks

A Virtual Private Network (VPN) is a network using secure links over the public IP infrastructure [18]. A VPN is a more cost-effective solution to a corporate extranet than a private network that consists of private infrastructure. In order to create the extranet the different sites have to be connected to each other through the provider's network (ISP network). The access points between a customer's site and the provider are called Customer Edge (CE) and Provider Edge (PE) respectively. The internal routers are called Provider (P) routers (see Figure 12). A VPN consists of number of sites that are connected through the ISP network. A participating site can be part of more than one VPN (like site CE4). If two VPNs have no sites in common then the VPNs can have overlapping address spaces. Since the addresses can overlap the routers need to interpret the addresses on a per-site basis by installing per-site forwarding tables in the PE routers. MPLS VPNs are set up with the combination of BGP-4 and another label distribution protocol (LDP, CR-LDP or RSVP-TE) (see section 2.4.4). The PE routers distribute labels associated with VPN routes to each other with BGP-4 [114, 115]. A VPN route is the combination of an IP prefix and a Router Distinguisher (RD). The RD allows to distinguish between common prefixes of the different VPNs. The other label distribution protocol is used to create a mesh of LSPs between the PE routers. The VPN route labels and the labels distributed by the internal label distribution protocol are used by the PE routers to forward packets over the VPN. The internal LSRs (P routers) only operate on the top label, the label distributed by the internal label distribution protocol, so they do not need to be aware of the BGP routes.

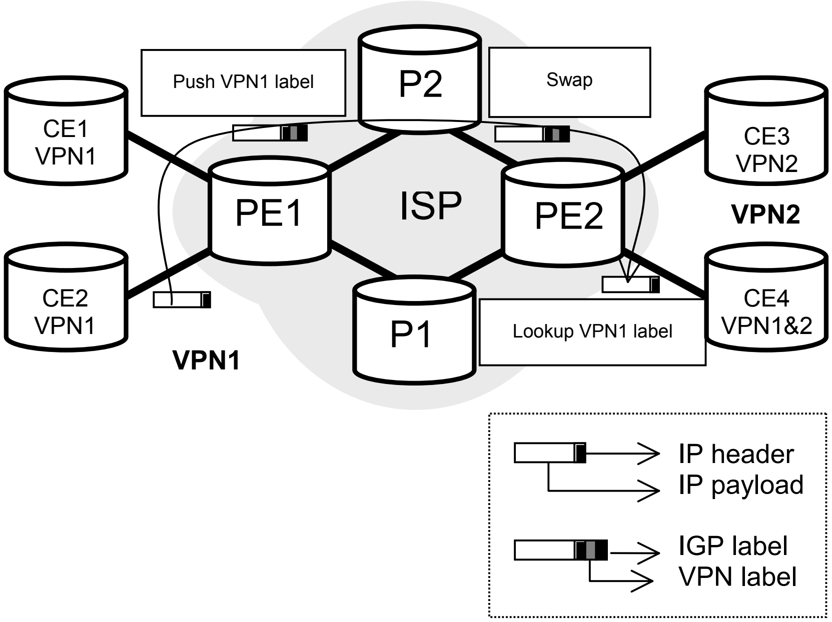


Figure 12: Example of an MPLS VPN and the forwarding of packets between the different interconnected sites.

Consider the example that a packet needs to be forwarded from CE2 towards CE4 over VPN1. The ISP network consists of BGP peers on the edge (PE1 and PE2) and interior LSRs (P1 and P2). A BGP node sends a packet to a certain VPN by looking up the label it has received from its BGP next hop and pushes this label on the label stack. For example PE1 pushes the label it has received from PE2 for VPN1. PE1 then looks up the label received from the internal label distribution protocol to PE2 and pushes this label on the label stack. The label stack then contains the BGP label for the VPN route (the VPN label) and on top of that the label for the BGP next hop (PE2). Then regular label switching is used to forward the packet to PE2. At PE2 the top label is popped and the VPN label pushed by PE1 is revealed. PE2 will then use this VPN label to look up the information to forward the packet to the next hop on VPN1 i.e. CE4.

2.6.4 Resilience

Regular IP typically recovers from network failures by rerouting. The routing protocol that is used to calculate the shortest paths in a network is able to detect network failures (or is notified of) and takes them into account when finding new routes after the failure. It typically takes some time before the routing protocol converges, that is before the network reaches a stable state after the failure.

When using MPLS, IP routing can also be used to restore the shortest path routed LSPs. This rerouting in MPLS depends on the new paths calculated by the IP routing protocol, this means that MPLS rerouting based on IP rerouting is slower than IP rerouting.

However, MPLS allows the use of more advanced resilience schemes. Protection switching is a scheme where a recovery path is pre-established. The recovery path can be node or link disjunct from the working path. (The working path is used as long as no failures are detected.) When a failure occurs the traffic is switched from the working path to the recovery path (the protection switch). The use of protection switching leads to a much lower convergence time. An additional advantage of protection switching over rerouting is that resources can be allocated in advance so that even after a failure the traffic over the LSP can still be serviced according to the predefined traffic parameters. Rerouting typically does not offer such guarantees unless the network is carefully planned. The drawback of protection switching is that it requires recovery paths that are pre-established leading to administrative and signalling overhead and a higher resource usage if the resources are dedicated (i.e. they cannot be used when the recovery path is not in use).

We will return to the subject of resilience extensively in the next chapters of this work.

2.7 Generalised MPLS (G-MPLS)

The original MPLS architecture considers only LSRs that are capable of recognising packet, cell or frame boundaries. However, the forwarding planes of a lot of devices do not have the property that it can recognise packet or cell boundaries, for example network equipment where the forwarding decision is based on time slots, wavelengths, or physical ports. In order to support MPLS on these devices the MPLS architecture is extended to explicitly support these devices called Generalised MPLS (G-MPLS) [116]. Also the control plane needs to be extended to encompass time-division (e.g. SONET and SDH [117]), wavelength (optical lambda's) [118] and spatial switching (e.g. incoming port or fibre to outgoing port or fibre). Protocol specific formats and mechanisms are specified for both RSVP-TE [119] and CR-LDP [120].

G-MPLS supports a generalised label request that contains how the LSP should be encoded (*LSP encoding type*), how the LSP should be switched (*switching type*) and an identification of the payload carried by the LSP (*generalised payload ID*). For example in order to carry Ethernet over a WDM network with Lambda switching capable LSRs the encoding type would be: *Lambda*, the switching type: *Lambda-Switch Capable* and the payload ID: *Ethernet.*

G-MPLS also supports elaborate bandwidth encoding, the possibility to suggest a label to the upstream node and bi-directional LSPs.

Another important feature of G-MPLS is that it is possible to separate the control channel and the data channel. So the signalling packets can be carried over another medium as the LSPs itself. This concept was introduced to support link bundling [121] in MPLS. In G-MPLS, the separation of control and data channel may be needed for other reasons. For example data channels that cannot carry in-band control information.

Finally G-MPLS has a more extended notification mechanism, better administration information and fault handling.

An important application domain for G-MPLS is optical lambda and fibre switching to carry IP or MPLS traffic because of the huge bandwidth potential these technologies offer.

2.8 Conclusions

The initial drive for MPLS was the performance gain achieved by using ATM switches to forward IP packets. Performance is no longer perceived as an advantage of MPLS over regular IP. MPLS does offer new functionality to IP networks. By separating the forwarding and the signalling layer and through its path oriented nature MPLS facilitates traffic engineering. More recently G-MPLS offers MPLS-like functionality to a new class of equipment including WDM based LSRs.

This evolution in MPLS also translates in the label distribution protocols for MPLS. The LDP, first label distribution protocol standardised, offers support for a lot of different label distribution modes but it does not offer any traffic engineering functionality. Both CR-LDP and RSVP-TE address this limitation of LDP by supporting traffic parameters, constrained routes, pre-emption and resource affinity procedures. BGP can also be used as a label distribution protocol typically to support MPLS VPNs. Finally, both G-MPLS extensions to CR-LDP and RSVP-TE are being developed.

The remainder of this work will focus on convergence schemes for IP and MPLS. MPLS supports protection switching which is not available to pure IP networks. It is important to realise that, as we have illustrated in this chapter, MPLS has more applications than protection switching, even though we will focus on resilience in the remainder of this work.

Chapter 3

Failure convergence in IP networks

In this chapter we investigate a number of failure convergence techniques for both IP and MPLS. We start by looking at the two major families of routing protocols in IP that are used for routing within a single autonomous system. These two families are the distance vector and the link-state routing protocols. Afterwards we will look at the convergence schemes in MPLS. We start by examining MPLS rerouting followed by a description of MPLS protection switching. The next chapter will investigate Fast Topology-based Constrained Rerouting (FTCR), a scheme developed by the author [3].

Before we start the discussion about the different convergence schemes we first have a look at the different phases in failure convergence and failure reversion. We start by giving a model of the generic cycles in failure convergence and failure reversion followed by a more in depth investigation of failure detection and failure notification.

This chapter will only describe the techniques and how they are able to converge after a failure. In the last chapter we will investigate the convergence times, the stability, scalability and backup requirements.

3.1 Failure recovery and reversion cycle

In this section we will describe a generic model to describe the different phases of both rerouting and protection switching convergence schemes. The model is loosely based on the models described in [122]. There are two models: one model for the recovery cycle and one model for the reversion cycle (see Figure 13 and Figure 14). The recovery cycle describes the actions a convergence scheme takes to handle a failure, the reversion cycle describes the actions taken after a failure has been cleared. When a node restores the traffic back to the preferred path after a failure has been cleared it is said that the node operates in *revertive mode*.

3.1.1 Failure recovery cycle

The recovery cycle starts when a network impairment occurs (see Figure 13). A network impairment can be a link failure, a node failure or the failure of a specific part of the router (e.g. a line card or control component). Detecting an impairment will always take some time. This *fault detection time* depends on the failure detection scheme that is used. Actual values range from a few milliseconds to a few seconds. When the fault has been detected the failure detection scheme notifies the router(s) by sending a Failure Indication Signal (FIS). The informed router might be a router directly attached to the failure or it might be a more remote router. Therefore, the time between the failure detection and the time that the router(s) receives the FIS, the *notification time*, can vary significantly. When a router is notified of a failure it might postpone the recovery actions. The reasons to wait are usually external factors e.g. to verify that another mechanism does not repair the fault. During the *hold-down time* the router keeps further actions pending and at the end of the interval it decides whether or not to start the recovery operations. If the FIS is cleared then it will probably not initiate the recovery operations. During the *recovery operation time* the router initiates the recovery actions, these actions usually involve other routers but thit is not always the case. When the recovery actions are complete the traffic is converged and the recovery cycle is complete.

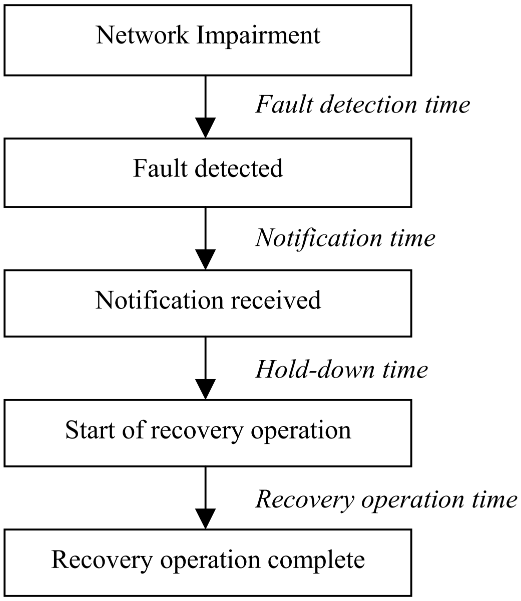


Figure 13: The failure recovery cycle describes the sequence of events starting from the network impairment and ending with failure convergence. The time between the individual events is important because they determine the overall time it takes from the impairment to failure convergence.

When the recovery cycle is completed that does not mean that all the negative effects of the failure have fully disappeared. For example, when the recovery path is longer than the original working path it will take some time after the convergence before packets start to arrive back at the destination. Another example is when TCP traffic has been affected by a failure it will take some time before the TCP streams flow at the same rate as prior to the failure [123].

In this chapter we will investigate different failure convergence schemes. A very important property of these schemes is the time the convergence cycle takes. Overall the recovery cycle needs to be as short as possible to reduce the network downtime to a minimum but not at the cost of the stability and scalability of the scheme.

3.1.2 Failure reversion cycle

The other cycle is the reversion cycle. When a fault is repaired it might be preferable to reverse back to the original working path. The reversion cycle starts when the network impairment is repaired.

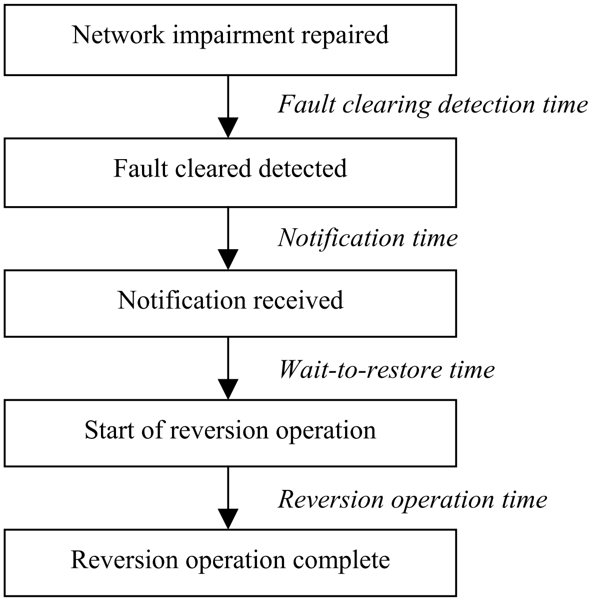

Network impairment repaired
Fault clearing detection time
Fault cleared detected
Notification time
Notification received
Wait-to-restore time
Start of reversion operation
Reversion operation time
Reversion operation complete

Figure 14: The reversion cycle describes the sequence of events starting from a fault clearance to the moment the reversion is complete.

After the *fault clearing detection time*, the time it takes to notice the fault clearance, the detection mechanism notifies the router(s) that a fault has been cleared. The *notification time* itself depends on the nature of the notification mechanism and the distance between the cleared fault and the router that initiates the reversion actions. When this router receives the notification it might wait for pre-configured time before it starts the reversion operations. This *wait-to-restore time* can be used to wait for external events. After the wait-to-restore time the router will start the reversion operations. The reversion operations take place during the *recovery operation time* and usually involve other routers too. Finally when the recovery operations are completed the traffic is restored to the preferred path.

The reversion cycle is not as time critical as the convergence cycle. During the convergence cycle the network is recovering from the network impairment and thus packet loss occurs. When the network has converged to a new path the connectivity is restored. When the failure is cleared it might be preferable to switch back to the preferred path but since connectivity is still available this process is less time critical. More important than the speed of reversion is to make sure no additional loss is introduced and that the packet reordering is kept as low as possible.

3.2 Failure detection and notification

In this section we will look at the first phases of the recovery and reversion cycles in more detail. We start this section with a description of different failure detection mechanisms and then look at failure notification.

3.2.1 Failure detection schemes

3.2.1.a Hardware based

Some link-layer technologies support failure detection natively. This means for example that when a cable breaks between two Network Interface Cards (NIC) this is immediately noticed in the data-link or physical layer of ISO Open Systems Interconnection (OSI) model [79]. Failure detection can be based on the detection of the lack of an electrical or optical signal. In order for such a failure detection mechanism to be useful there must be a way to notify the upper layer of this failure because a failure usually requires actions by the upper layers (e.g. convergence mechanisms).

Failure detection by the link-layer is very fast but it does not cover all possible error cases. For example when a control component of a router breaks this will not be noticed by such mechanisms. Another example is that when two routers are connected via a switch and a cable breaks only one router will notice the failure. The router behind the switch will not notice the failure. So while link-layer failure detection offers the fastest failure detection it does not cover all cases and other forms of failure detection are needed. In the next section we will describe a generic failure detection mechanism that is used in many protocols under various names, we will use the term heartbeat.

3.2.1.b A generic heartbeat mechanism

In this section we will describe a generic unidirectional mechanism that can be used to monitor a link or a control protocol peer in a given direction. Consider that we want to monitor the liveliness of a link. When a packet arrives on a link it is obvious that the link is operational at that instant in the receiving direction. But the reverse is not true, because no packet has arrived does not mean that the link has failed. However when no packets are received for a considerable amount of time one can suspect that a failure has occurred. But in order to prevent false negative reports one cannot just declare that the link has failed after an arbitrary time of inactivity. Such false negatives can be avoided if we can guarantee that at least one heartbeat packet is sent over a certain period of time. The sender is then responsible to send at least one packet within a pre-negotiated interval. Within this interval at least one message must be sent; this interval is called the *send interval*. If a sender agrees to send a packet at least every send interval the receiver can conclude that if no packet was received during this send interval that a failure has occurred. However in order to prevent false negatives due to transmission delay, packet loss or a transmission error in the packet, the send interval should be smaller than the interval used at the receiving side. Therefore, the receiving side uses a different interval: the *receive interval*. The receive interval is then the upper bound to detect a failure on the link. In the next section we will examine in detail what the lower and upper bounds are for the failure detection time.

The sender is required to send at least one heartbeat packet during the *receive interval* (see Figure 15). When the receiver receives a packet it will reset its receive timer associated with the receive interval. So when the receive interval is chosen sufficiently larger than the send interval then under normal conditions a router will always receive a packet within the receive interval and the receive interval timer will not expire. However during fault conditions no packet will be received and the receive timer will expire. When the receive timer expires a failure is assumed.

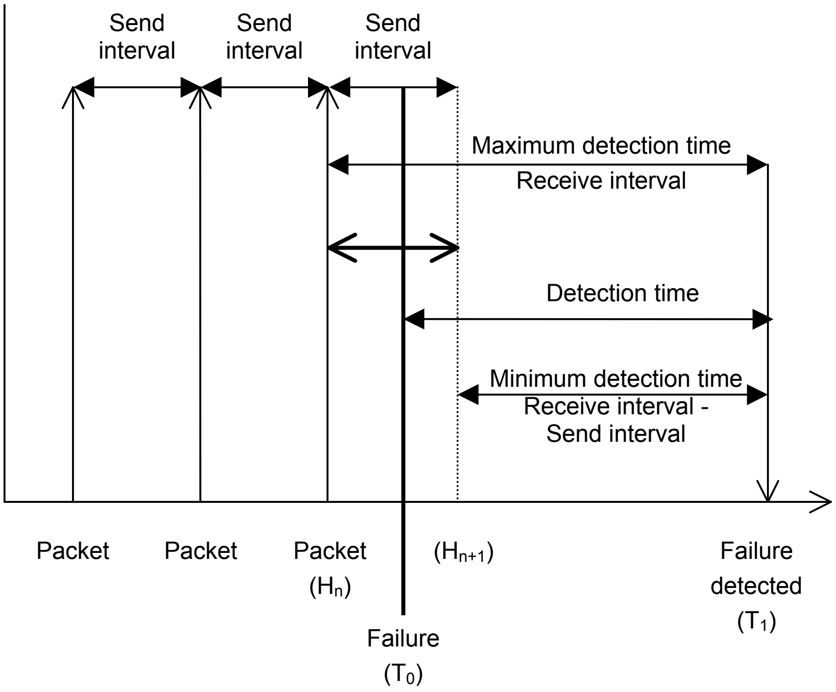


Figure 15: Failure detection with the generic heartbeat protocol. Heartbeat packets are sent every *send interval*. When no heartbeat packet is received for a *receive interval* amount of time a failure is detected. The actual detection time depends on when the last heartbeat packet was received with respect to the receive interval.

The time when a failure is detected (T_1) depends on when the last packet is sent (H_n) with respect to the time when the failure actually occurs (T_0). If the last packet is sent just before the failure (T_0 approaches H_n) then it will take almost a complete receive interval before the failure is detected. However if the last packet is sent almost a send interval time before the failure (T_0 approaches H_{n+1}) it will take less time to detect the failure. More specifically it will take up to a send interval less time to detect the failure. This leads to the detection time interval:

$$\text{receive interval} - \text{send interval} \leq T_{detect} = T_1\text{-}T_0 \leq \text{receive interval} \qquad (1)$$

This generic heartbeat mechanism is unidirectional. Bidirectionality can easily be achieved by running two instances of the protocol on each node. The node acts as the receiver in one instance and as sender in the other instance. Actually the fact that the mechanism is unidirectional should be considered a benefit because this makes the mechanism more flexible. Certainly in the context of LSPs which are intrinsically unidirectional. The second drawback of the mechanism is that the time to detect the failure cannot be decreased without increasing the overhead. We see in (1) that the detection time depends on the length of the receive interval, if the receive interval is decreased then the send interval needs to be decreased too in order to prevent false positives. This in turn means that the number of packets that are sent per second increases too and that the bandwidth that is used by the heartbeat protocol increases as well. When the receive interval is decreased without decreasing the send interval the protocol becomes less reliable because packet drops and delay can lead to false positives. Obviously the lower limit of the receive interval is the send interval but in practise the receive interval is taken at least three times higher as the send interval.

Despite these drawbacks this generic heartbeat mechanism is used in a number of protocols. As we will explain in section 3.3.2.b the link-state routing protocol Open Shortest Path First (OSPF) [36, 124, 125, 126] uses a very similar approach to detect failures. The send interval is called the *Hello interval* and the receive interval is called the *router dead interval.* Another link-state routing protocol called Intermediate System- Intermediate System (IS-IS) [37, 38] also uses a very similar concept to detect failures. The send interval is called the Hello interval, IS-IS does not use a receive interval but it states how much larger the receive interval is than the send interval. This parameter is called the *Hello multiplier*. Finally another example is the Hello mechanism of LDP (see section 2.4.1). LDP uses a slight variant because it only negotiates the receive interval, which is called the Hello interval, and the sender is responsible to make sure that a Hello packet is received in this interval.

In the next section we investigate how this mechanism can be improved by decreasing the detection time while limiting the overhead and without sacrificing the reliability.

3.2.1.c Decreasing bandwidth overhead

As we have seen in the previous subsection sending packets to monitor a link or control protocol is possible if a send and receive interval are negotiated. Decreasing the detection time can be done by decreasing the length of these intervals but at the cost of decreased reliability and increased bandwidth usage.

For example the OSPF hello protocol uses intervals in the seconds range. This means that HELLO packets are sent every few seconds. This does not induce a very high overhead in terms of processing or bandwidth usage. However the detection time is also quite high. The detection time can be decreased by decreasing the length of the send and receive interval. But since the Hello packets compete with other packets for link bandwidth there is obviously a practical lower limit to these intervals.

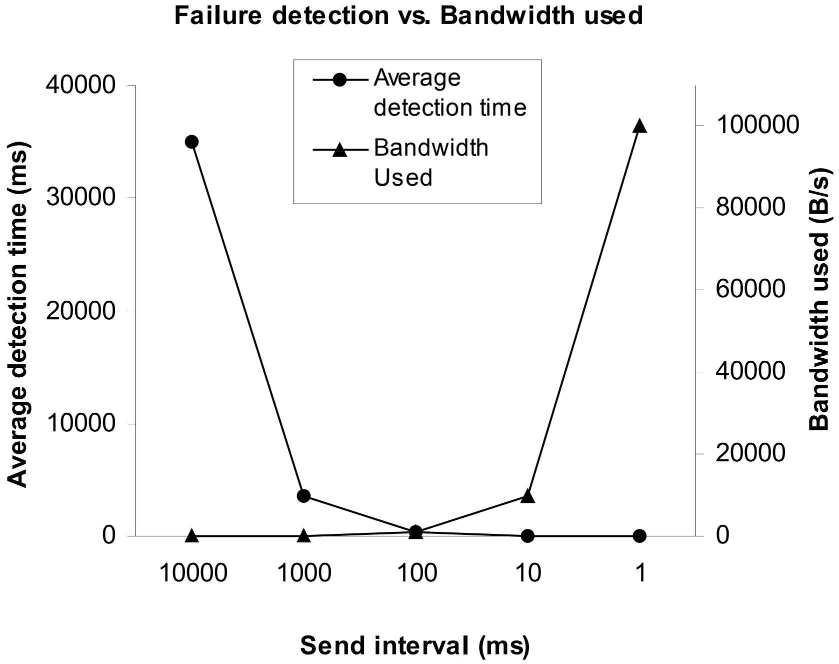


Figure 16: The failure detection time depends on the send interval. When the send interval is decreased, the detection time will also decrease but at the cost of additional bandwidth that used to send the heartbeat packets.

Figure 16 illustrates the fact that the detection time can be decreased by decreasing the send interval. A send interval of 10s corresponds with a standard Hello Interval (i.e. the send interval) of OSPF. In this figure we consider that the receive interval is four times higher than the send interval (like in OSPF). According to formula (1) this corresponds to an average detection time of 35s. The detection time can be decreased by decreasing the send interval and the receiver interval. This is shown by the descending line on the figure. However decreasing the send interval will influence the bandwidth used to send the heartbeat packets. If we consider that every heartbeat packet occupies 100B at the link layer then the ascending line shows the corresponding bandwidth used. For example when one heartbeat packet is sent every 10s the bandwidth used is 10B/s, however when one heartbeat packet is sent every 1ms then 100000B/s are used. The corresponding average detection time is then 3.5ms.

We will now describe how the generic heartbeat mechanism can be extended so that we can decrease the convergence time without wasting useable bandwidth. We start the discussion within the context of monitoring the status of a link followed by an example of the monitoring of a control component.

When we only want to monitor the status of the link there is no need to send specific packets since any packet is fine to assert that the link is operational. This means that we can reset the receive timer at the receiving side every time we receive *any* packet on the link instead of a specific monitoring packet. The sending node should monitor the outgoing packets on the link, if a packet is sent during the send interval the sender does not need to send a packet. However if no packet is sent during the send interval then a dummy packet will be sent as before. This means that the send and the receive interval can be decreased without sacrificing useable bandwidth. When the link is heavily used, packets will be sent regularly and the sending side decides that it does not need to send packets and no bandwidth is used. If the link is not heavily used then the sending side detects that no packets are sent during the send interval and will insert dummy packets so that the receiving side still receives at least one packet during its receive interval. We investigated this in more detail in [44] where the mechanism is called *link probe*. Experiments revealed that detection times of 150ms-300ms can be achieved with proof-of-concept implementations.

We see that this technique has the valuable property that it does not use bandwidth when the link is highly loaded but still offers fast failure detection. This technique can also be used to monitor the status of a control component. Instead of monitoring any packet that is sent over a link one needs to monitor the control packets that are sent. This is what the LDP keepalive mechanism does, when no LDP packet is sent for a configurable amount of time a keepalive packet is sent. The keepalive packet has no real semantics but it is used to keep the LDP session alive (see section 2.4.1).

If the mechanism is used to monitor control packets then it will be able to detect a broader range of failures since it is able to detect control component, link and node failures. However the amount of control packets is typically only a very small percentage of the total amount of traffic so the gain achieved is quite small. There is a trade-off between the amount and the types of failures that can be detected and the bandwidth that is saved. Since there is always a need to detect control component failures we cannot simply apply the mechanism solely to all packets because that makes it is impossible to detect control component failures.

The solution which we also proposed in [44] is to use the link probe mechanism as a fast link failure detection mechanism and to retain the existing protocol specific failure detection, this is illustrated in Figure 17. Another reason why the protocol specific mechanisms should be retained is that they typically serve other purposes besides the detection of control failures.

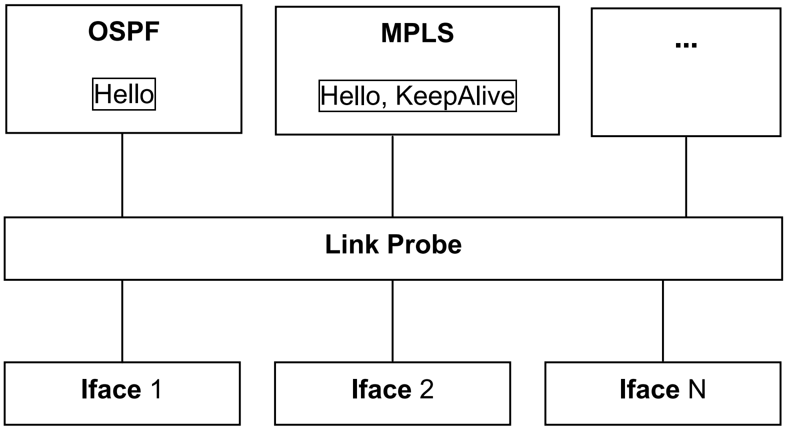


Figure 17: Link probe can be used in conjunction with existing protocol specific failure detection mechanisms.

Note that link probe offers little benefit if hardware-based failure detection is available. However, in that case the protocol specific detection mechanisms should be retained too.

3.2.2 Failure notification and failure clearance notification

In the previous sections we took a brief look at how failures can be detected. In this section we will look at how information about a failure event can be distributed. Failure notification is often needed to inform other nodes of a failure because the node that has detected the failure cannot repair the failure solely by itself. For example it might be necessary to notify the ingress of an LSP that a failure has occurred somewhere on the path of the LSP. Another example is that a router needs to notify its routing peers that a link has failed and that the current routing tables need to be recalculated.

However failure notification is often implicit. When a router detects a link failure it might send a notification message (explicit notification) or it might just stop advertising that link (implicit notification). Implicit notification has as drawback that the cause of failure is not known. For example when a router stops advertising a link this might be caused by a link failure but it might also be that the node has failed or even that the link in question has been disabled by the administrator. Explicit notification can indicate the type of the failure although that is not always the case.

Looking at MPLS, both LDP and RSVP-TE have an error notification mechanism to indicate that the egress of the LSP is (no longer) reachable. But both lack a specific encapsulation to notify what exactly is the type or the cause of the failure. Still we call any message that is sent to indicate that failure has occurred or that causes a recovery action a *failure indication signal (FIS)*. This means that a FIS can be for example a LDP notification message or RSVP-TE Path Err message. FIS messages can be implemented using a distinct protocol or they can be part of an existing protocol (as the previous examples show).

Failure notification and failure detection is often part of an Operations And Maintenance (OAM) framework. The ATM forum glossary [127] defines OAM as “A group of network management functions that provide network fault indication, performance information, and data and diagnosis functions.”. However an OAM framework for MPLS has not yet been standardised although effort is underway [128, 129, 130].

As we pointed out some messages in (CR-)LDP and RSVP-TE can indicate failure conditions. However for some convergence schemes these messages and their semantics are inadequate [131, 132]. For example sometimes the ingress of an LSP needs to be informed about a failure on the path of the LSP. In RSVP-TE the Path-Err message could be used but this message is usually sent in response to a Path message. Since a failure is not always associated with a Path message this creates a semantic mismatch. Another problem is that the object used to encapsulate information about the error condition, the Error-spec object, is not sufficient to convey the necessary information. Expanding the Error-spec is of course a possibility but since this leads to the problem of downward compatibility, [132] concludes that a new notification mechanism is more suitable and proposes the Reverse Notification Tree (RNT).

The reverse notification tree is a tree that is set up in the reverse direction of the LSP. LSPs can be merged which leads to a multipoint-to-point tree. Since the RNT follows the reverse direction this can lead to a point-to-multipoint RNT tree (see Figure 18).

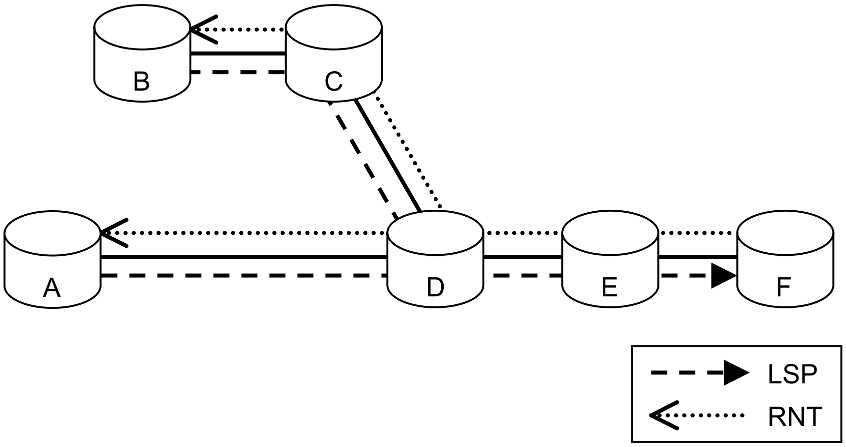


Figure 18: The Reverse Notification Tree (RNT) for the merged LSP A-D-E-F, B-C-D-E-F, C-D-E-F. When the link EF fails, node E sends a FIS upstream. Node D is then responsible to send a FIS to both node A and node C.

The information in the FIS message sent over the RNT contains at least the address of the failure detecting node. But every node is also responsible to verify which LSPs are affected by the failure before sending the FIS further upstream. For example when node E notices a failure on link EF it will send a FIS upstream with its own address in the failure detecting node field. When node D receives this FIS it is responsible to send this FIS upstream towards both A and C because node D is a merge point of the LSP. The FIS messages are not sent using a reliable protocol like TCP but rather the messages are repeated with a pre-configured frequency (the *FIS frequency*). The failure detecting node continues to send FIS messages until another timer expires (the *FIS duration*). When this timer expires the detecting node stops sending FIS messages and it is assumed that all the affected nodes are notified or that a high layer mechanism has converged the network. Note that the RNT is only used to send failure indications upstream. Downstream notifications are based on the MPLS signalling protocol.

When a failure occurs a FIS often needs to be sent to inform other routers in the network of the failure, similarly when a failure is cleared there needs to be a mechanism to inform the routers that the failure is cleared. The signal itself is usually called a *Fault Recovery Signal (FRS)*. For example when a router detects that a certain link is available again it will start to tell the other routers that this link can be used again. Like FIS messages, FRS messages can be implemented using a distinct protocol like RNT or they can be part of an existing protocol.

Now that we have covered failure detection and failure notification we start discussing the various failure convergence schemes for IP and MPLS networks.

3.3 Convergence schemes for IP and MPLS

In this section we describe different convergence schemes for IP and MPLS. Convergence protocols are really atypical protocols, they cannot rely on existence of routing tables because these protocols themselves are responsible to install the routing tables.

The convergence schemes for IP are usually referred to as routing protocols. The reason for this is that these schemes not only enable IP to converge from faults in the network but also that they install the working routes in the network when the network is first installed (cold boot). In this section we will only investigate routing protocols which operate within one autonomous system. These routing protocols are called Interior Gateway Protocols (IGP) in contrast with routing protocols that operate between autonomous systems that are called Exterior Gateway Protocols (EGP). We will investigate two important families of IGP routing protocols i.e. distance vector and link-state routing protocols. Some of the MPLS convergence schemes rely on the IP routing protocols. These schemes are called MPLS rerouting schemes. MPLS protection switching does not use IP routing to converge from network failures. Instead it relies on backup paths that are pre-established. Finally we propose a new scheme called Fast Topologybased Constrained Rerouting (FTCR), this scheme allows for fast convergence without the limited coverage or the high number of backup paths that protection switching typically introduces.

The IP routing protocols and MPLS signalling protocols are described in the relevant IETF Request For Comments (RFCs). However these documents usually only describe the external observable behaviour. Sometimes these specifications are not enough to get detailed insight into the working of these protocols. That is why we also looked at specific implementations of these protocols to get further insight [46, 133]. Since the main goal of the investigation is to gain insight into the working of these protocols with respect to convergence we won't examine for example the encapsulation of the various messages.

In this section we will start by examining the various intra-domain IP routing protocols before we examine the MPLS convergence schemes. We will only investigate intra-domain protocols, for information about inter-domain convergence with BGP we refer to [134].

This section starts by investigating distance vector routing in general and the Routing Information Protocol (RIP) in specific.

3.3.1 Distance vector routing

The first used interior gateway routing protocol of the Internet is simply called the Routing Information Protocol (RIP) [39, 40, 41]. RIP is a "distance vector routing" protocol based on a distributed implementation of the Bellman-Ford shortest path algorithm [45]. The first version described in RFC 1058 [135] documents the at that time current implementations for "routing". The RFC most notably describes the implementation of *routed* (route daemon) distributed with the 4.3 Berkeley Software Distribution (BSD), a popular Unix implementation at that time.

In the next section we will describe how distance vector routing protocols install the working routes and afterwards we look at how these routing protocols can converge after a failure. Subsequently we will address the timing and synchronisation issues with respect to distance vector routing. That section is followed by an explanation of the weaknesses of distance vector routing.

3.3.1.a Basic operation

Dynamic routing protocols like RIP initially require only local knowledge in each router i.e. the address of the router and the links that are attached to it. The routing protocol is responsible for the spreading of this local information across the network. This local information in combination with the source of the information can eventually be used to calculate the shortest paths in the network. Distance vector routing protocols and link-state protocols use a different approach to calculate these paths. Distance vector protocols achieve this goal by spreading distance vectors across the network. A distance vector is a vector of (node, cost) pairs. The cost associated with a node is the cost to reach it. Links have an administrative cost associated with them. This cost can for example be proportional to the monetary cost to use the link or it can be proportional to the length, bandwidth or delay of the link. The routing protocol tries to minimise this cost for a given path between two points in the network. In this example and further on we assume that the cost for each link is one. So the problem statement for finding a minimum cost path translates into finding a minimal hop count path. In the next section we will describe how RIP as a distance vector protocol achieves this goal. The RIP operations are first explained from the point where the network starts up (cold boot) and then we will investigate what happens when a link or node fails.

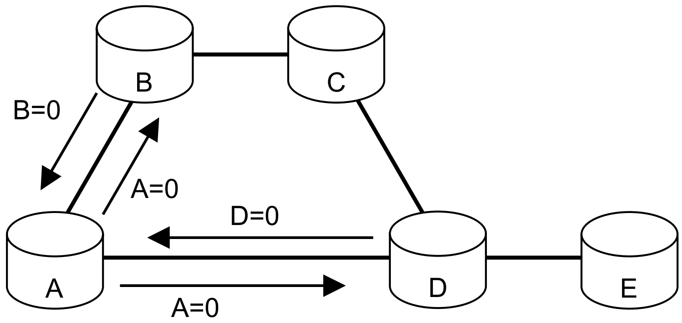


Figure 19: During the cold-boot of the RIP protocol, every router starts by announcing its local state first. Illustrated here are how node A, B and D announce their local state to each other.

When the network starts up each node will start to announce its local state. Consider that node A is the first to start announcing this information. It will start by constructing a simple distance vector with only its own address with a cost of zero and sending this vector over its attached interfaces (see Figure 19). When the directly attached nodes (node D and B) receive this distance vector they will reply with a distance vector with their own local knowledge i.e. their own addresses.

When a node has received non-local knowledge it will incorporate this information in all future distance vectors. As illustrated in Figure 20 we see that node B uses this recently learned information to spread a distance vector (A=1, B=0) to its other attached interface BC. Notice that node B has incremented the cost to reach node A with the cost of its local interface AB. This process is further illustrated in Figure 20 where the learned knowledge from the operations in Figure 19 are depicted in a box.

We see that node D sends the distance vector (A=1,B=2,C=1,D=0) towards node E. Node D has received two different costs to reach node A i.e. (A=0) from A and (A=2) from C. In order to minimise the overall cost a node will always use and announce the lowest cost it has learned.

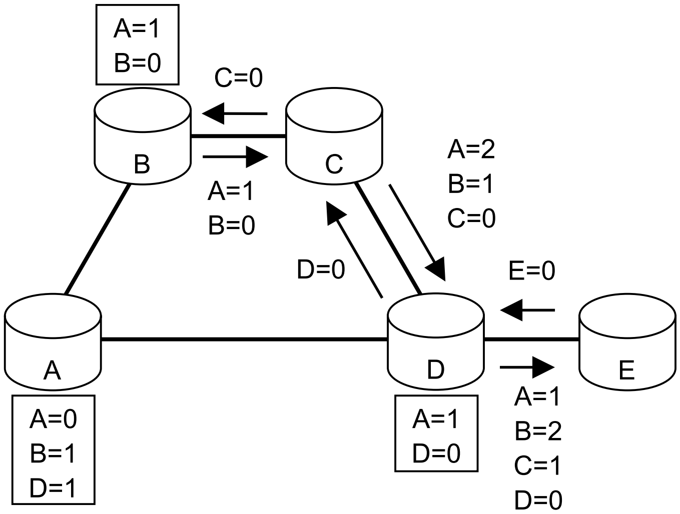


Figure 20: RIP routers do not only announce their local state but also the information that they have gathered during their operation. We show the interaction between nodes B, C and D. The gathered information in the routers before the illustrated interactions are shown in boxes.

All the nodes in the network will continue to announce their distance vectors and adjusting them if they receive a lower cost to reach a certain node. At any given time the distance vectors are also used to forward traffic in the network. For example when we look at Figure 20 we see that the routing tables in node A (the framed box) is only partially complete. But at a certain moment in time all the routers will have the lowest cost associated with all other nodes in the network. At this moment the spreading of distance vectors no longer changes the state in the nodes and we say that the network has converged. The forwarding on the routing tables i.e. the distance vector in the nodes is a simple operation. Up until now we didn’t mention the fact that the nodes also store the incoming interface of each of its retained entries in its distance vector. The incoming interface of the entry in the distance vector is used as the outgoing interface to reach the destination in the distance vector. For example Figure 21 shows the final routing tables after the network has converged. We see that node A uses interface AD to forward packets to node E because it has received the lowest cost to reach node E from node D over interface AD.

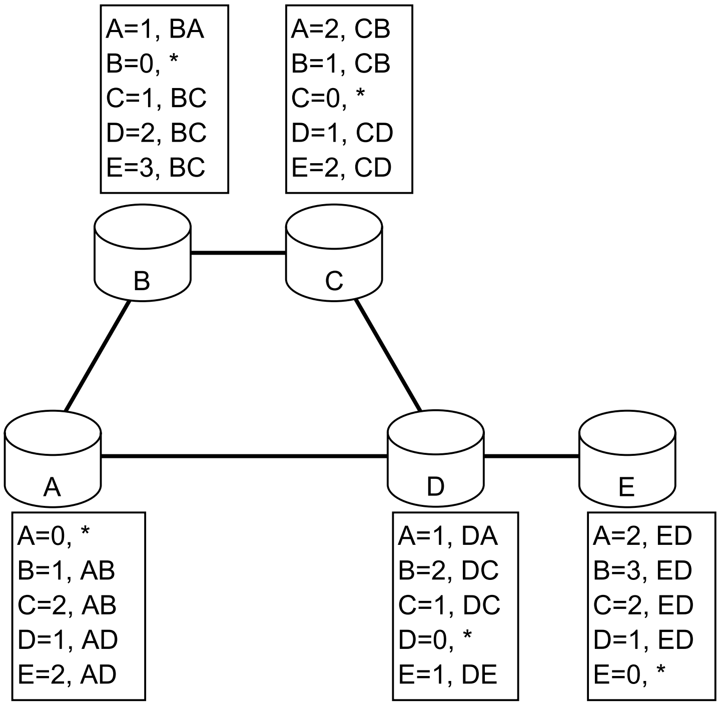


Figure 21: These routing tables are the result of the distributed RIP shortest path calculation in the network. Every node stores the cost and the outgoing interface towards all the destinations in the network.

3.3.1.b Failure convergence in distance vector routing

Now that we have explained how RIP, as an example of a distance vector routing protocol, can calculate the routing tables inside a network we will investigate how this protocol can handle link and node failures. When a node detects a link failure it will announce a new distance vector where the cost of the link is set to infinity. When a node receives a distance vector with a cost of infinity for a given destination it will never use it. This prevents new nodes from using the failing link but this does not solve the problem of existing nodes forwarding traffic towards the failing link because the new cost is always higher than the current cost. This is solved by requiring that when a node has received and retained an entry from another node he has to update to that node's new cost at all times (even if the cost is higher). So when a node announces the new cost of infinity all the nodes that used the original cost from that node have to update to the new cost of infinity. When there is another route to the destination this distance vector entry will soon be replaced by an entry with a lower cost from another node. When this happens the outgoing interface towards the affected destination changes and the traffic is routed around the failure. Note that this may take a couple iterations of flooding distance vectors in the network before this is achieved.

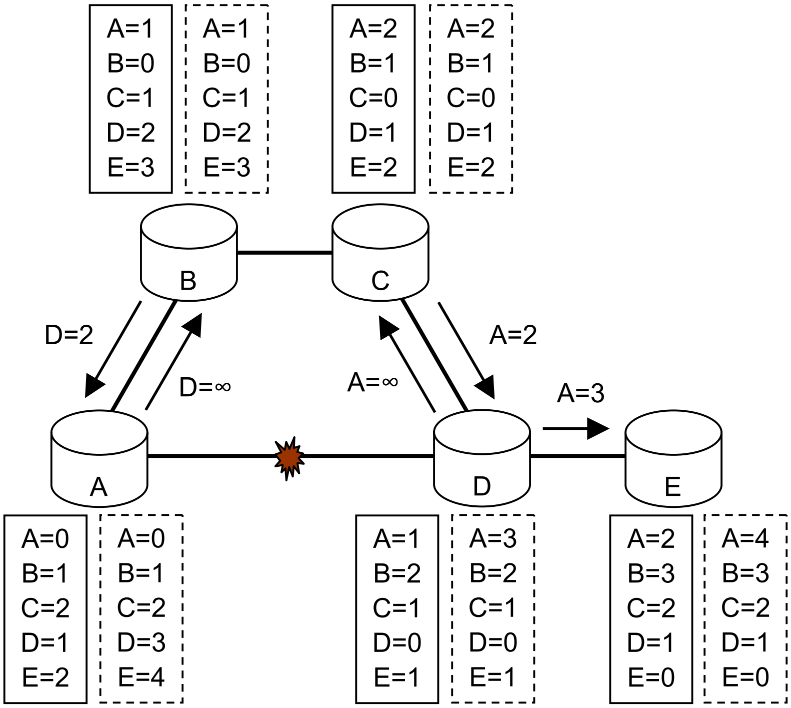


Figure 22: When a failure over a route is detected RIP will associate a cost of infinity with this route. The original route (solid box) will soon be replaced by another route (dashed box). For example node A associates a cost of infinity to its route to D after the failure. But soon thereafter it learns a new route from node B and uses that to reach D.

Referring back to the previous examples we illustrate the RIP recovery operations when the link AD fails (see Figure 22). First node A and D detect the failure and spread a new distance vector with the cost to reach node D and A respectively set to infinity. Node B will receive the distance vector from node A. Now consider for a moment that B uses node C to reach D (as in Figure 21). This means that node B does not need to update its distance vector for destination D. But B continues to send the same distance vector to its neighbours. When A receives this distance vector from B it will update its own distance vector and set the cost to reach D to 3. Node A now has a new route to reach node D that circumvents the failure on link AD.

In the meanwhile node C has received the new distance vector from D. Considering that node C uses node B to reach node A (again as in Figure 21), node C does not need to update its distance vector. It will continue to send its original distance vector to its neighbours including node D. When node D receives the distance vector from node C it will adjust its distance vector to reflect the new cost to reach A via node C. Finally node E will have to increase its cost to reach node A because the originator of the original entry to reach node A i.e. node D has increased its cost.

Up until now we have considered that node B and C did not forward traffic over the failing link AD. Now let's investigate what happens when they do use the failing link to forward traffic. Both the case where they do and the case where they do not use the failing link are possible because the cost of these routes is equal (for example the cost B-A-D equals the cost B-C-D).

When B and C forward traffic to D and A respectively over the failing link then the recovery operations are slightly different. Again A and D start by sending an infinite cost to reach D and A respectively. Since B uses A to forward traffic towards D it has to update its cost to reach D to infinity. Similarly node C will update his cost to reach A to infinity. Subsequently B and C send each other their distance vectors. Since B and C are respectively directly connected to A and D they spread a low cost path towards these destinations. As a result B and C will update their distance vectors to reach D and A via C and B respectively. Then node B and C send their updated distance vector to A and D so that they also have the correct path to D and A.

As we see in this case the convergence is a bit slower but the protocol is still able to converge. It is however important to note that the convergence time is not constant for a given network. This is a general problem with RIP and we will give other examples later on. But first the next section describes how RIP supports failure detection, node failures, multiple failures and the revertive actions.

3.3.1.c Failure detection and distance vector updates

Distance vector routing protocols do not use an explicit failure detection mechanism. External failure detection mechanisms can be used but they are generally not present. Instead RIP relies on the regular transmission of the distance vectors to monitor the routes and to be tolerant of packet loss. RIP requires that the distance vectors are announced regularly in the network, if no updates are received within the *time-out interval* the cost associated with these routes is set to infinity. In order to prevent time-outs during normal conditions, distance vector packets are sent every *update interval*. As such the time-out mechanism can be seen as a specific implementation of the generic heartbeat mechanism explained in section 3.2.1.b where the timers are associated with the routes in the distance vector.

Now consider that a link fails, the nodes that are connected to the failing link will be unable to announce their distance vectors to each other so eventually the routers will set the cost of routes learned via each other to infinity and forwarding over these routes will be avoided. The convergence will then proceed as explained in the previous subsection.

Notice that the timers are associated with the routes in the routing tables rather than directly associated with the RIP peers. So unlike the generic heartbeat mechanism explained in section 3.2.1.b, RIP is not able to directly detect that a link or peer has failed it will only detect that a route is no longer available.

As already explained a RIP router associates a time-out timer with every entry in its routing table. The detection time of a route failure depends on the time-out timer and the update interval. The lower these timers, the faster a faulting route is detected. However there is a lower limit to this timer. Obviously the time-out timer must be higher than the update timer plus the transmission delay of the link. Moreover the time-out timer must preferably be high enough so that the loss of a single distance vector packet does not lead to a false positive failure detection. In RIP the standard value of the update timer is 30 seconds and the time-out timer is 180 seconds. This means that up to six updates can be lost before the route associated with the timer is declared unreachable.

We have seen that failure convergence can require a significant number of distance vector exchanges. If the time between these distance vector exchanges is always equal to the update timer (standard 30 seconds) then obviously this leads to quite high convergence times. On the other hand, reducing the update timer increases the bandwidth and processing overhead without any benefit in normal working conditions. The solution is to trigger distance vector updates when a router receives a new or modified distance vector. *Triggered updates* will speed up the cold boot process and the convergence time while it does not increase the overhead during normal conditions [135, 136].

Finally, in order to prevent synchronisation, the actual value of the update timer varies between 15 and 45 seconds. This randomisation of the update time was introduced in RIP version 2 after the observation that every 30 seconds certain autonomous systems running RIP version 1 had peaks of congestion and packet loss. These effects where caused by the fact that the routers in the autonomous system started to get synchronised because of the fixed update time and triggered updates. Requiring a sufficiently large randomisation of the update time solves this problem.

3.3.1.d Node failures, multiple failures and reversion

In this section we address some of the aspects of convergence that we have not covered. These are node failures, multiple failures and the reversion operations.

When a node i.e. a router fails inside the network that router is no longer able to announce its distance vector to the other routers in the network. As a result all the routes that the failing router announced will time-out in the attached routers. The attached routers will start to announce a cost of infinity associated with these routes. This results in the situation where there is no route to reach the failing node with a cost lower than infinity. When this happens the failing node is effectively pruned from the network and the network now converges by calculating routes that circumvent the failing node.

We will now look how distance vector routing protocols converge from multiple failures. We define a *multiple failure event* as the event where one or more failures occur after a first failure but before the network has converged from the first failure. So the event where a failure happens, the network converges from that failure and the occurrence of a second failure does not count as a multiple failure event but rather as two consecutive single failures events. Regardless, multiple failures require no real special treatment by distance vector protocols. All the affected routes will time-out and the network will calculate new routes which circumvent the failure. Multiple failures can however lead to slow convergence in the network with routing anomalies during the transient period. This will be illustrated in the next section.

Finally we briefly address the reversion operations. When a link becomes operational again the attached routers will start to announce their distance vector over that link again. Other routers will pick up these updates and adjust their distance vector if this results in shorter routes. The result is that the link will be used again if that leads to shorter paths in the network, which is the correct behaviour. When a node goes back up this should be seen as a collection of links that are operational again and so the reversion operations are similar to the link failure case.

In the next section we will investigate the primary drawback of distance vector routing protocols namely slow convergence.

3.3.1.e Limitations and undesirable features

In the examples in the previous section we always took the cost of every link equal to one, as discussed before the cost of link can be an expression of for example the monetary cost or the delay, so a cost of one might not always be appropriate. However setting a high value may lead to undesirable behaviour as illustrated in the next example.

We use the same topology as in the previous examples but this time the cost of link AD has a high value such as 10 (see Figure 23). Because of its high cost the link will not be used to forward traffic in the network but when link CD fails link AD needs to be used because no other path is available for certain destinations (e.g. to reach node D from node A). Note that this might be exactly the reason for which this link has such a high cost i.e. that the link is only used when no other link is available. As we will see RIP will eventually route the traffic over the link AD but it will take a significant number of iterations before the network has converged.

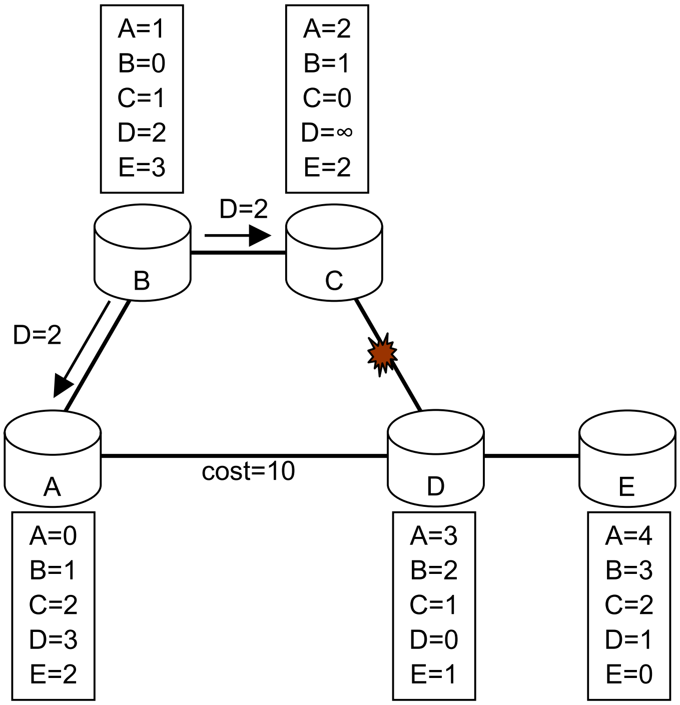


Figure 23: Convergence in RIP can require a significant number of iterations as illustrated by the *bouncing effect*. For example when a high cost metric, like 10, is used for link AD. When link CD fails, node B and A will need to count to 11 before link AB is used and D and E will finally be reachable from A, B and C.

When the link CD fails node C will set the cost to reach node D to infinity. When node B sends its distance vector to node C, node C will notice that node B has a shorter path towards D and will update its distance vector so that the cost to reach D is now 3. When node C in his turn sends its distance vector to B, node B has to update its cost to reach D too because it uses C to reach D. This cycle will be broken only when the cost to reach D in node B has increased to 11. Node A will then receive this cost from node B and will notice that it can reach node D on a cheaper path by using the link AD. After that node A will send his new distance vector to node B and node B will make the correct decision to reach node D via node A. When node B sends this new distance vector to node C, node C will also reach D via node B and node A and the network has finally converged.

The previous example illustrates that convergence can take a substantial amount of iterations because node B and C have to "count" till 10, i.e. the cost of the link AD. This highly undesirable effect of distance vector routing protocols is called the *bouncing effect* because the distance vector to reach D bounces between node B and C until a certain value is reached.

Another undesirable feature of distance vector routing protocols is counting to infinity. Consider now that not only link CD but also link AB fails. A similar counting effect will happen where the cost to reach A, D and E in B and C will increase with each iteration. This will in principle never stop. The solution to this problem is to stop this counting to infinity at an arbitrary but sufficiently high value. When the cost in node B and C has reached this value they will consider the cost infinite and declare the associated destinations unreachable. This value needs to be set high enough so that valid paths are not declared unreachable i.e. this value must be higher than the longest possible path in the network.

Fortunately the bouncing effect and counting to infinity can be solved by a simple *split horizon* test [40, 135]. The problems illustrated above stem from the fact that the nodes announce the cost that they have learned from a given node to that node itself. For example in the bouncing effect example node B announces to node C the cost to reach D as the cost it learned from C plus one. Obviously there is no reason for node B to announce to node C that it can reach a certain destinations via node C itself. The split horizon test therefore forbids that a node announces the routes that it has learned from a certain node to that node. This prevents the bouncing effect and counting to infinity between two nodes. In the bouncing effect example this would mean that node B will not announce to node C that it can reach node D with a cost of 2. Node C will then announce a cost of infinity to reach node D to node B and node B will have to update its distance vector accordingly. Afterwards node B will announce to node A that the cost to reach D is infinity. Node A will then favour link AD to reach D and will announce this cost to node B. Node B will then announce this new cost to node C and the network has converged.

The problem of counting to infinity can be solved with a more aggressive variation of the split horizon test called *split horizon with poisonous reverse* where the nodes announce all the destinations but will set the cost to infinity if the destination is reached via that node. This will immediately set the cost to infinity and will remain infinite if no other route is available thereby quickly marking the destination unreachable.

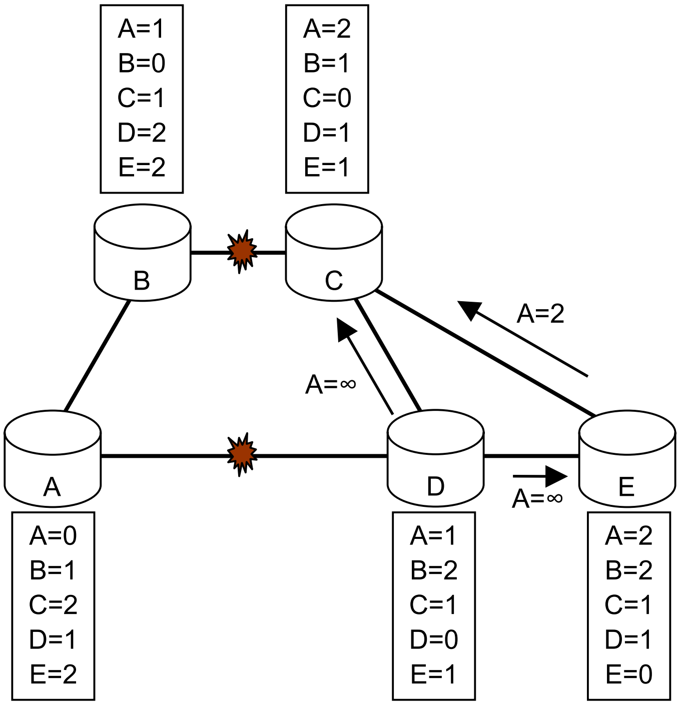


Figure 24: Even with the split horizon with poisonous reverse *three way counting to infinity* can occur. Nodes C,D and E will count to infinity before they declare A and B unreachable.

However neither the basic split horizon test or the split horizon with poisonous reverse can prevent all undesirable features of distance vector routing. Consider for example the topology depicted in Figure 24. When link AD fails node D will announce an infinite cost to reach A. Now consider that this update reaches C before it reaches E and that at the same time node E refreshes its announcements. Since node E uses poisonous reverse it will announce an infinite cost to reach A to node D and it will announce its real cost (A=2) to node C. While this sequence of events might seem unlikely it can happen when for example the update of the infinite cost of node D towards node E is lost due to a transmission error. Regardless of the fact that the above situation is common or not, it does lead to a new form of counting to infinity. Since node C has learned a cheaper way to reach A it will increase the cost and announce the cost to node D and node D in turn will increase the cost and announce the new route learned from node C. When node D announces the new cost to reach node A to node E, node E will update his distance vector and will announce this information to C. This process will continue, leading to a three-way counting to infinity.

While it might be possible to invent new schemes that can solve three-way counting to infinity it is general very difficult to prevent these anomalies in all cases. This is an intrinsic weakness of distance vector routing protocols. The fundamental problem is that Bellman-Ford is a distributed algorithm where every node has only limited knowledge about the network where it know its attached routers but has no knowledge about the topology of the network.

We have already seen that RIP is not a good choice for a convergence scheme in complex networks because RIP only supports networks of the limited size and because of the counting to infinity and bouncing effects. OSPF as an IP convergence scheme. Other problems of RIP are the lack of support for external routes and multiple metrics.

In the next section we will look at the routing protocols of the link-state family that are more complex but do not suffer from these problems.

3.3.2 Link-state routing

In the previous section we have explained RIP, a routing protocol from the distance vector routing protocol family. Distance vector routing protocols are really only suitable for small networks. This can be illustrated by referring to the large number of iterations needed to converge and the counting to infinity problems. But it is probably best illustrated by mentioning that the value set to infinity in RIP is 16. That means that networks with a diameter of more than 15 nodes cannot be supported. Another important drawback of RIP is that it can only support static metrics so dynamic metrics like link load cannot be used. In the remainder of this work we will only consider link-state routing protocols.

In this section we will introduce link-state routing protocols and we will illustrate their working by examining the Open Shortest Path First (OSPF) protocol [36]. Link-state routing protocols address the weaknesses of distance vector routing protocols but at the cost of a higher complexity. We will start with an overview of the basic operation of the OSPF protocol. Note that we will focus on OSPF rather than on IS-IS which is also a widely used link-state routing protocol. Research seems to be more focussed on OSPF (for example [12] covers

OSPF in detail and IS-IS just briefly), open implementations of OSPF are more widely spread [46] and OSPF is specifically designed for IP. For a comparison between IS-IS and OSPF we refer to [137, 138].

3.3.2.a Basic operation

We started the very first section on the basic operation of distance vector routing protocols by stating that these protocols start by spreading the locally available information in every router over the network. Exactly the same can be said about link-state routing protocols: they also start by spreading local information. The main difference is the way this information is stored and used. Distance vector routing protocols only retain and forward information about the routes they actually use i.e. the shortest route up to now. Link-state routing protocols on the other hand store every entry that they receive together with the source of this information. For example in Figure 25 node A starts by sending its locally available information. This information is then stored and further forwarded by every router in the network.

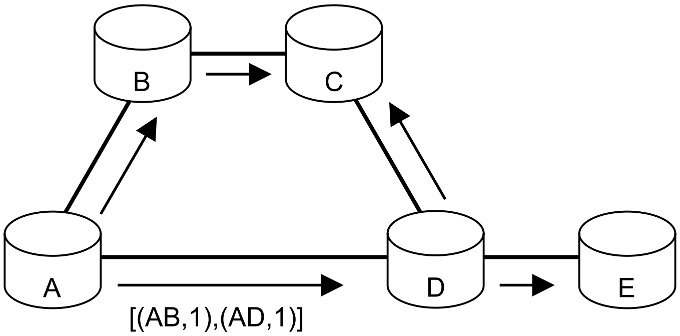


Figure 25: When the OSPF protocol starts up (cold boot) every router announces its local state in a link-state packet. These link-state packets are then flooded over the network. The link-state packet from node A and its flooding are illustrated.

Not only the procedures for storing and spreading information differ between link-state and distance vector routing but also the information itself. Link-state routers do not spread distance vectors; they announce their attached links and the cost to traverse this link. It is said that routers involved in a link-state routing protocol spread link-state packets also called *link-state advertisements* (LSA). A link-state advertisement contains a list of link identifiers and their associated cost. Each newly received LSA must be acknowledged, if the LSA is not acknowledged within the *retransmit interval* it will be retransmitted. The LSAs are also refreshed regularly (standard every 30 minutes [36]).

Figure 26 shows the state in each router after the first link-state advertisement by node A. Observe that every node has collected the same state information [(AB,1),(AD,1)], this information is stored in the *link-state database*. This is in clear contrast with distance vector routing where the routers only store their local routes and associated costs. When node D also advertises its local information all the link-state databases contain the following information:

```
[(AB,1),(AD,1),(DA,1)(DC,1),(DE,1)].
```

Subsequently every router in the network will advertise its local information. When all the routers have advertised their local information the link-state databases of the routers will contain an entry for every link in the network:

```
[(AB,1),(AD,1),(BA,1),(BC,1),(CB,1),
 (CD,1),(DA,1),(DC,1),(DE,1),(ED,1)].
```

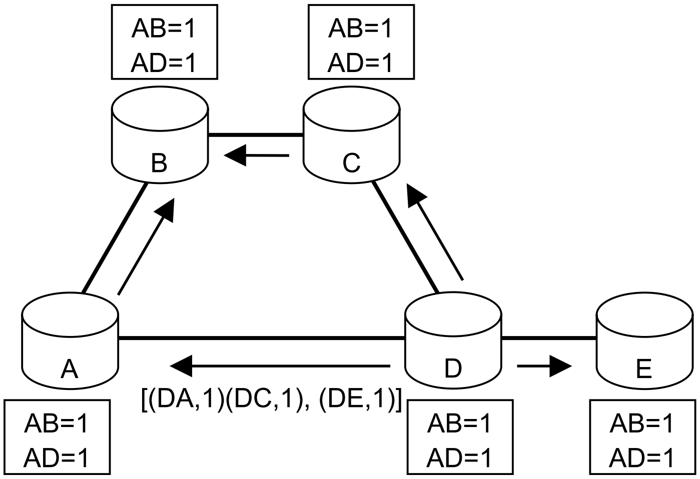
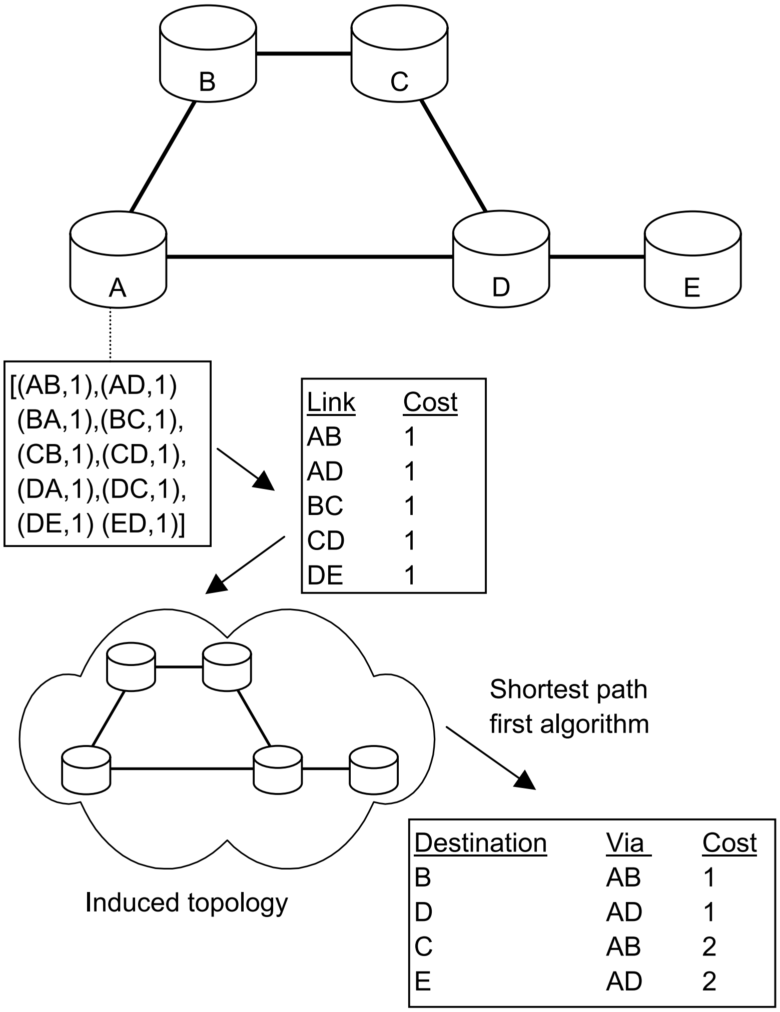


Figure 26: Every router stores information about every link in its link-state database. Illustrated here are how the link-state databases contain the information about node A’s links and the announcement by node D of its local state.

It is obvious that this spreading of link-state information as such does not lead to a distributed shortest path calculation like in the distance vector routing case. However, the gathered information can be used indirectly to calculate the shortest paths. From the link-state database the topology of the network can be constructed. It is then possible to calculate the shortest paths with the Dijkstra Shortest Path First (SPF) algorithm [45] on this topology (see Figure 27). The

Dijkstra algorithm will be used to calculate a shortest path tree with the router as root of the tree. After the calculation, the shortest paths from the router towards all the destinations in the network are known. After the calculation these paths are installed in the routing tables.



[(AB,1),(AD,1)
(BA,1),(BC,1),
(CB,1),(CD,1),
(DA,1),(DC,1),
(DE,1) (ED,1)]

Link	Cost
AB	1
AD	1
BC	1
CD	1
DE	1

Destination	Via	Cost
B	AB	1
D	AD	1
C	AB	2
E	AD	2

Figure 27: The OSPF routers use the information in the link-state database to induce the topology of the network. This topology and the metric information is then used to calculate the shortest paths in the network from the router towards all the destinations in the network. The result of the shortest path calculation is stored in the local routing table.

Instead of working directly on the routing tables, link-state routing uses an intermediate link-state database gathered from the link-state advertisements. Although the databases and the algorithm used is the same in every router, the result i.e. the routing tables are not the same because the routes are calculated from a different source.

Calculating the shortest paths on the link-state database only takes a small amount of time on current CPUs, for example [139] mentions hundreds of milliseconds up to one second for a 300 node network. As all the routers have the same database the calculated routes are coherent and loops do not occur when the network has converged. For example in our network, node A has calculated that in order to reach node E it has to forward the packet to node D. Node D has also calculated the shortest path to node E and will forward it on its local interface to node E.

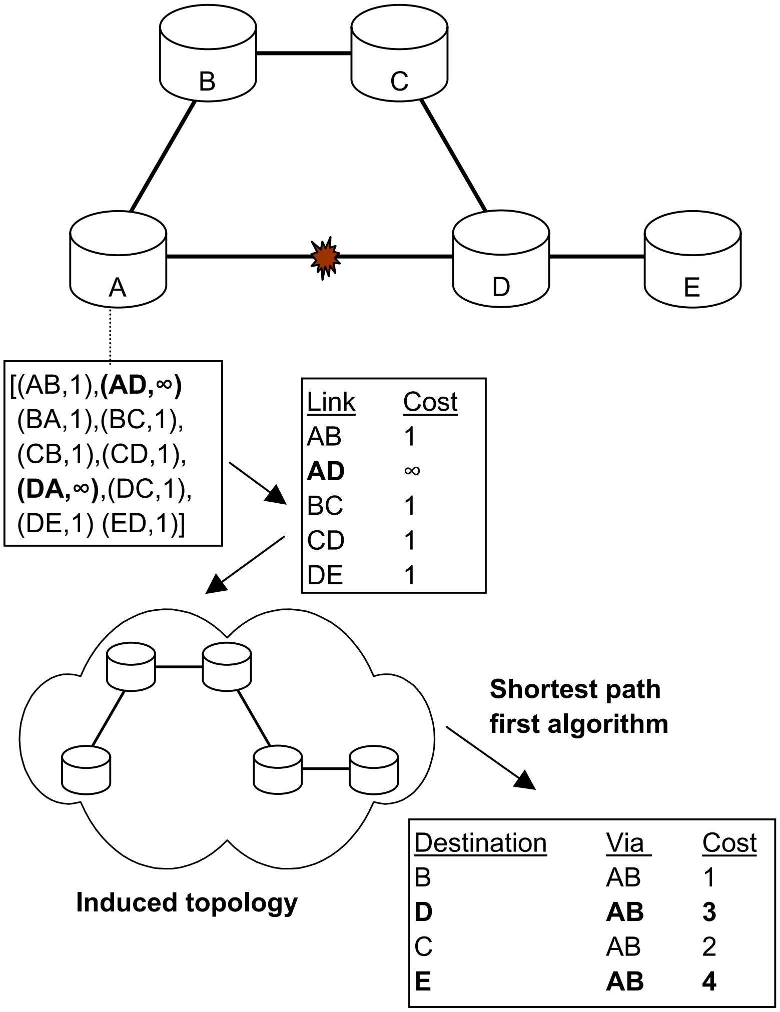
In the next section we will investigate how link-state routing protocols are able to recover from network failures.

3.3.2.b Failure convergence in link-state routing protocols

As explained link-state routing protocols use the topology information gathered via link-state advertisements to calculate the shortest paths. When a link or node fails in the network the topology view that the routing protocols have needs to be updated to reflect the new current topology. This is achieved by sending new link-state advertisements that withdraw the impairment from the induced topology. We will explain this process in detail in the next section within the context of a link failure, afterwards we will also explain how failures are detected in OSPF.

When a node detects a link failure, that node will transmit a new record that indicates that the link is down by setting the cost of the link to infinity. The routers then flood this new link-state packet across the network and store it in their link-state database. The new link-state databases will result in a new topology view where the failing link is missing. The Dijkstra algorithm will calculate new routes that do not use the failed link (see Figure 28). Finally when the new routing tables are installed the packets will be forwarded along the new routes that take the failure into account.

As we see the convergence operations in link-state routing protocols are relatively simple: update the topology, rerun the shortest path algorithm on the new topology and install the new routers. This makes link-state protocols like OSPF converge much faster than distance-vector protocols. This does not mean that during transient periods certain routing anomalies like routing loops cannot occur. During such transient periods the routers have different versions of the link-state database and thus have a different view of the network topology. As a consequence the result of the shortest path computation on these topologies in the different routers can be inconsistent which can lead to routing anomalies.



[(AB,1),**(AD,∞)**
(BA,1),(BC,1),
(CB,1),(CD,1),
(DA,∞),(DC,1),
(DE,1) (ED,1)]

Link	Cost
AB	1
AD	∞
BC	1
CD	1
DE	1

Destination	Via	Cost
B	AB	1
D	**AB**	**3**
C	AB	2
E	**AB**	**4**

Figure 28: When a failure is detected over a link, the link is withdrawn from the induced topology by announcing a cost of infinity associated with the link. The shortest path calculations will not take the failing link into account and the calculated routes will circumvent the failure. These new routes will then be installed in the routing tables. The actions of node A are illustrated for a failure on link AD.

Failure convergence in link-state routing protocols is pretty straightforward but how can we prevent that an old link-state packet that is still traversing the network reinstalls a recently removed link? For example an older link state packet from node A where the link AD still exists should never be used in preference of the newer link state packet that removes AD from the topology. This can be solved by adding an indication of the age to the link-state packets (by adding a sequence number or a time stamp) and applying correct flooding procedures. The routers should take care that an older link-state packet will never replace a newer entry in the link-state database and that only the newest link-state packets are forwarded. With these precautions older state will never replace newer state in the network.

Failure detection

Another aspect of failure convergence is failure detection. In distance vector routing a failure is detected by associating a timer with every entry in the routing table, when the timer expires the route is deleted and later replaced with a new route that takes the failure into account. OSPF also uses timers to detect failures but the timers are not associated with the routes but rather with a small protocol within OSPF. This protocol is called the *hello protocol* and it serves multiple purposes including neighbour detection, to elect the designated router and to check that links are operational. Here we will only address how the hello protocol is used to check that links are operational, we refer to the section about reversion below for more details.

Every router is required to send a hello packet during the *hello interval*. When a router receives a hello packet it will reset its *router dead interval* timer associated with the incoming interface. When the router dead interval is chosen sufficiently larger than the hello interval then under normal conditions a router will always receive a hello message and the router dead interval timer will not expire. However, during fault conditions no hello messages will be received and the router dead interval timer expires. When the router dead timer expires the associated link is declared dead and a new link state packet with a cost of infinity for that interface is sent.

Notice that the OSPF hello interval and router dead interval timers are very similar to the RIP update and time-out timers. In fact both RIP and OSPF implement a generic heartbeat mechanism as explained in section 3.2.1.b. The major difference is that OSPF associates these timers with a specific link while RIP associates the timers with the routes in the distance vector. There is also a major difference in granularity of the number of update packets i.e. in the RIP case the updates are proportionally to the network size while in OSPF the updates are proportionally to the number of nodes on a link. This translates into much lower standard timers. The hello interval timer is set to 10 seconds in OSPF [140, 141] while the update timer is set to 30 seconds in RIP [40]. The router dead interval timer is set to 40 seconds in OSPF [140, 142] while the time-out timer in IP is set to 180 seconds [40]. This means that the detection of a failure in RIP can take up to 180 seconds while this only takes up to 40 seconds in OSPF (see formula (1) in section 3.2.1.b).

Up to now we have not put any emphasis on the fact that a link failure is always detected by the two ends of the link. This has important consequences nevertheless. In Figure 29 we see when link AD fails both nodes A and D send new link-state packets. When node B receives the link-state update from node A it will adjust its link-state database and rerun the shortest path algorithm. The result of the shortest path algorithm will then be used to update its routing table. Only a few moments later the link-state update from node D forwarded via node C will arrive at node B. It is highly undesirable that these two link-state database updates lead to two shortest path computations and routing table updates. Shortest path computations and routing table updates are typically resource demanding and can disrupt traffic, even traffic that is unaffected by the failure. To reduce the frequency of shortest path computations and routing table updates some OSPF implementations like Zebra introduce two additional counters: the *SPF delay* and the *SPF Hold-Down* timer [46].

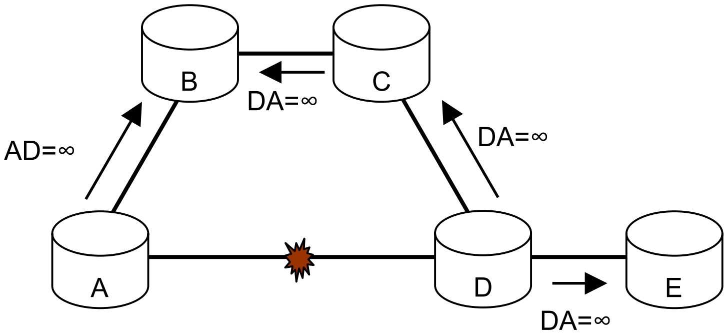


Figure 29: When a failure is detected the cost of the link is set to infinity and the new link-state packet is spread over the network. Since there are two endpoints of a link an updated link-state packet will be sent for each direction of the link.

The SPF Delay timer is the minimum time between receipt of a new link-state packet and the shortest path computation (see Figure 30). The SPF Hold-Down timer is the minimum time between two subsequent SPF calculations. When a new or updated link-state packet is received it is verified that no other shortest path calculation is scheduled. If there is another calculation planned then no new

SPF computation is scheduled and the link-state packet is incorporated in the link-state database (link-state packet *b* in Figure 30). Although unique link-state packets do not always lead to new SPF calculations they are always incorporated in the link-state database and subsequently used when calculating the new routes. When there is no calculation planned the shortest path calculation is delayed by the SPF delay (link-state packet *a* in Figure 30). However, if a previous calculation was done less than Hold-Down timer seconds ago then the calculation is delayed until Hold-Down seconds after the last calculation (link-state packet *c* in Figure 30). Note that the Hold-Down timer should be larger than the SPF delay for this scheme to have effect.

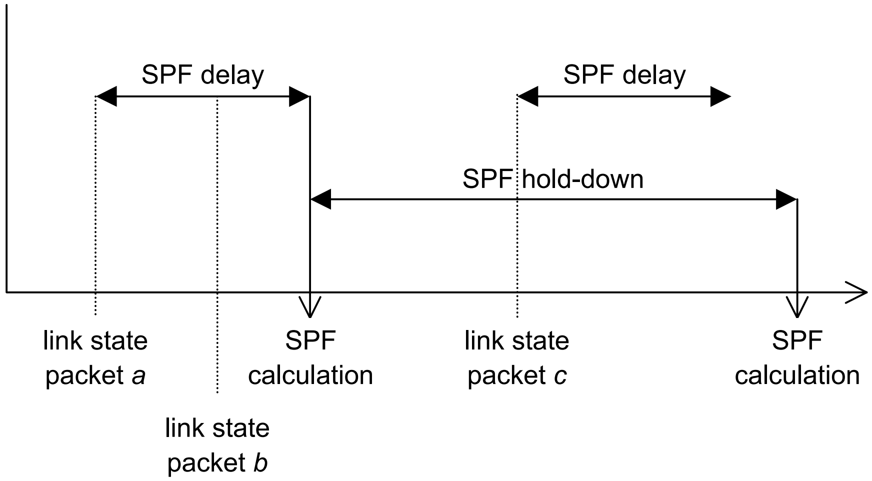


Figure 30: In order to reduce the number of SPF calculations two different intervals are used. The *SPF delay* is the minimum time between the receipt of a link-state packet and the resulting SPF calculation. The *SPF Hold-Down* interval is the minimum time between two consecutive SPF calculations.

Note that the hold-down time from the generic convergence cycle includes both SPF Delay and the Hold-Down time. We will use ‘Hold-Down’ (with capitals) to denote the OSPF specific delay while we use ‘hold-down’ for the generic delay.

Node failures and multiple failures

Node failures are detected indirectly by the hello protocol. When a node fails, all of the directly attached routers will no longer receive Hello messages. After the router dead interval these routers will advertise the links connected to the failing node with a cost of infinity. When these new link-state packets are flooded over the network the resulting topology based on the new link-state database will have all of the links of the failing node removed. The result is that the failing node is pruned from the network topology. The subsequent run of the shortest path first algorithm will calculate routes that no longer use the failing node and the network will converge after all the routers have recalculated their routing tables.

Seen from the perspective of a link-state routing protocol a node failure is nothing more than a number of simultaneous link failures. As explained in the previous section this does not require specific extensions. Similarly supporting multiple failures does not require extensions either. However, both node and multiple failures can lead to routing anomalies over relatively long periods of time since the failures are detected independently by the attached routers. We illustrate this later when we discuss the stability of OSPF (section 5.4.3).

Reversion

The reversion operations in OSPF are very comparable to the recovery operations. After the failure is cleared the cleared fault will be announced again through link-state packets so that the routers can integrate the node or link again in their link-state databases and subsequently use it in their shortest path calculations.

The major difference between the reversion and recovery operations is due to the first phase i.e. the fault clearance. This phase is quite different than the fault detection. As we have seen a failure is usually detected with the Hello protocol. We also mentioned that the Hello protocol is not only responsible for failure detection but also for neighbour detection and designated router election. We will explain these two functions in more detail because they have to be performed before a new link can be announced by the OSPF protocol.

Before a link can be used by the OSPF protocol its two end points (the OSPF routers) have to discover each other and establish bi-directional communication. If the link medium is broadcast capable or non-broadcast multiple access (NBMA) then a designated router needs to be elected. The designated router is the router responsible for announcing the link-state packets for the shared medium. In order to support fail-over a backup designated router is also elected. Note that there is no full mesh of adjacencies, every router has only an adjacency between the designated router and the backup designated router.

Another task that needs to be performed before between two new OSPF neighbours is to synchronise their link-state databases. OSPF requires that the link-state databases of adjacent routers are synchronised. This synchronisation process begins as soon as the routers attempt to bring up the adjacency.

It is clear that forming an adjacency and synchronising the link-state databases between two OSPF routers will take some time. This time obviously is a component of the reversion time.

As mentioned before the other operations during reversion are very similar to the convergence cycle. The major difference is the nature of the event (failure vs. fault clearance) rather than the actions triggered by the event (link-state advertisements, shortest path calculations and routing table updates). We will now look at the convergence cycle and the reversion cycle of OSPF.

3.3.2.c Convergence and reversion cycle

Now that we have described link-state routing in general and OSPF in detail we will look at how the different phases of the convergence and reversion fit into the convergence and reversion cycles of section 3.1.

Convergence cycle

The convergence cycle begins with the detection of the impairment. Usually the Hello protocol of OSPF is used to detect failures. As already mentioned in section 3.2.1.b, the Hello protocol is a specific implementation of the generic heartbeat mechanism with as the send timer the *hello interval* and as receive timer the *router dead interval*. The detection time of a generic heartbeat mechanism is given by:

T_{detect} = [receive interval– send interval, receive interval]

which corresponds to:

T_{detect_hello} = [router dead– hello interval, router dead]

for the Hello protocol.

Table 1 combines the different phases of the convergence cycle for OSPF.

Table 1: The convergence cycle for the OSPF routing protocol. The first row uses the name of the parameters while in the second row these parameters are substituted by the standard values according to the OSPF version 2 RFC [36] and the Zebra OSPF implementation [46]

Fault detection	Notification time	Hold-down time	Recovery operation
[router dead-hello interval, router dead]s	δ	[SPF delay, Hold-Down]s	SPF (ε) update RIB (γ)
[30,40]s	δ	[5,10]s	ε+γ

The notification time (δ) in OSPF determines how long it takes before the updated link-state have crossed the network. The time depends on the speed of the interfaces, the load on the network links and the load on the routers.

As we have seen in the previous section the hold-down time of the convergence cycle depends when the last SPF occurred. The hold-down time is at least *SPF delay* but can increase to *Hold-Down*.

Finally the recovery operations in OSPF encompass the SPF calculation itself and the updating of the routing tables (RIBs). The calculation of the shortest paths (ε) depends on the network size and connectivity, the speed of the processor and the efficiency of the implementation. The time required to update the routing tables (γ) depends on the size of the routing tables, the number of entries that need to be changed and the implementation efficiency. Note that routing tables usually are quite complex constructions typically optimised for reading rather than writing. Also the fact that a longest prefix match must be supported increases the complexity of updating the routing tables too.

This table does not really reflect it, but it is possible that convergence is not achieved after one SPF calculation. This can be caused by large delays in the transmission of the LSAs (large notification time) or because the large differences in failure detection time between nodes. With the standard timers there is a variation in the detection time of 10s while the hold-down time is just 5s. Therefore, it is possible that after 5s not every failure is detected and that the resulting SPF does not cover all failures. The second SPF will usually cover all failures although there is no theoretical limit on the number of SPF calculations that are necessary.

Reversion cycle

The reversion cycle is very comparable to the convergence cycle (see Table 2). The major difference is the failure detection and the clearing detection phases.

The operations during the clearing detection time encompass the time for the neighbours to detect each other, the time to elect the designated router and the backup designated router and the time to synchronise the link-state databases.

Again it can be the case that reversion is not achieved after one SPF. If that is the case, the first SPF is followed by a second SPF, Hold-Down seconds later. More SPF calculations may be needed, each of them will be performed Hold-Down seconds after another.

Table 2: The reversion cycle for the OSPF routing protocol. The first row uses the name of the parameters while in the second row these parameters are substituted by the standard values according to the OSPF version 2 RFC [36] and the Zebra OSPF implementation [46]

Clearing detection	Notification time	Wait-to-restore	Reversion operation
Bring up adjacency + synchronise LSDB	δ	[SPF delay, Hold-Down]s	SPF (ε) update RIB (γ)
κ	δ	[5, 10]s	ε+γ

3.3.2.d Conclusion

We now have explained the two most important routing protocols for IP, in the next section we will look at convergence in MPLS networks. We will start by investigating MPLS rerouting which is a straightforward extension of IP rerouting to MPLS.

3.3.3 MPLS rerouting

In the previous sections we explained rerouting in IP networks. In this section we will look at rerouting in MPLS networks. The first thing to remark is that MPLS rerouting uses IP routing to route the existing LSPs in the network. As we already explained there are two types of LSPs, shortest path LSPs that are routed according to the IP shortest paths and explicitly routed LSPs that are routed according to an arbitrary specified path. In the first subsection we will investigate how MPLS can set up and reroute LSPs according to the shortest paths and in the second subsection we will investigate the issues surrounding the rerouting of explicitly routed LSPs.

3.3.3.a Rerouting of shortest path LSPs

In this section we will investigate how shortest path LSPs are set up and rerouted in MPLS. We start the discussion without going into the details of the specific signalling protocols for MPLS. Subsequently we will address some of the specific issues with respect to LDP and RSVP-TE. Comparing LDP and RSVP-TE is interesting because LDP is a hard-state protocol while RSVP-TE is a soft-state protocol and as we will see this has influence on the speed of convergence (for a more in dept comparison between RSVP-TE and CR-LDP we refer to [143]).

The set up of a shortest path LSP is initiated by the ingress by sending a label request message towards the destination. Every router on the path forwards this message over its outgoing interface determined by its local routing table. Once the label request arrives at the destination, the egress will respond by sending a label mapping message back towards to ingress. Every router then forwards this packet on the reverse path back to the ingress. It is important to note that the reverse path is used and not the shortest path from egress to ingress because shortest paths in IP can be asymmetric [144].

For example in Figure 31 the working LSP denoted with the dashed line is set up by node A by sending a label request towards node E, when node E receives the label request it will send a label mapping back over the reverse path. The nodes on the reverse path will then send label mappings to their upstream peers until node A receives a label mapping from node B and the LSP is set up.

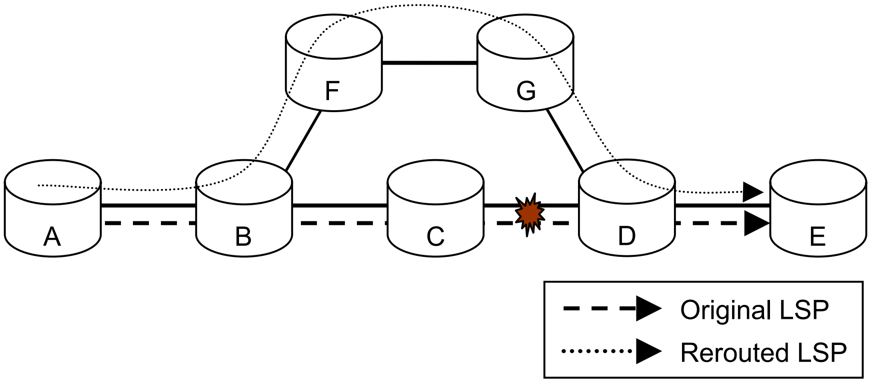


Figure 31: When node B detects that the next hop towards E has changed from C to F it will set up a recovery LSP over the new next hop. When this recovery LSP is set up the original LSP is rerouted.

We will now investigate the operations when a link fails. First of all the IP routing protocols running on the nodes will converge and new routing tables will be installed in the LSRs. For example when the link between node C and node D fails, the IP routing protocol will converge and will install new routing tables in the nodes. For node B this means that the next hop for the shortest path towards node E changes from node C to node F. When the MPLS signalling protocol notices this change it will send a label request via the new next hop towards the destination E. Again the egress, node E, will reply with a label mapping. When this label mapping is received at node B it will start switching the existing incoming label on the incoming interface AB to the new outgoing label over the interface BF. Node B will then use the new outgoing label towards node F and the LSP is rerouted around the failing link. It is important to note that the original LSP does not exist anymore in its original form because the rerouted LSP uses the original labels up to the point to where the LSPs detour (node B).

Moreover when the routers operate in release-on-change label retention mode (see section 2.3.4) node B and node C will release the original old outgoing labels. When the original labels are retained node D can merge the labels of the original working LSP and the rerouted LSP.

In next subsections we will investigate more closely how RSVP-TE and LDP reroute LSPs.

MPLS rerouting with RSVP-TE

We won't go into every detail regarding the rerouting of LSPs in RSVP-TE but we will focus on an important issue and that is how the routers are able to detect that the next hop has changed. RSVP-TE is a soft-state protocol which means that the RSVP equivalent of the label request and label mapping messages, the PATH and RESV messages respectively, need to be refreshed regularly in order to maintain the state in the routers [31]. The fact that the LSP is set up according to the shortest path implies that the forwarding of these messages is determined by the routing tables. When a failure occurs the IP routing protocol will introduce new routing tables and eventually the RSVP messages will be sent over the new path.

First the PATH message will be sent over the new path and then the RESV messages will take the reverse route setting up the recovery LSP. This is very similar to the above generic explanation of the MPLS rerouting. An important detail is that the set up of the recovery path is triggered by a refresh of the original working path. The convergence time thus depends on the refresh frequency. There is a trade-off between the convergence time and the scalability. Since the PATH and RESV refreshes are needed on a per LSP level it is not scalable to make the refresh period too short. On the other hand increasing the refresh period in order to increase the scalability leads to an increased convergence time. The standard value for the refresh period is R=30 seconds, in order to prevent synchronisation a refresh message is sent somewhere during a [0.5R, 1.5R] = [15,45] second time interval. The randomisation of the refresh message is done for similar reasons as for the RIP routing protocol (see section 3.3.1.b). This means that the time between the OSPF convergence, indicated by the installation of the updated routing tables, and the set up of the recovery LSP can be up to 45 seconds.

Typically the convergence time is even higher because a hold-down timer is used. When an LSP is refreshed, a check is made to verify that the next hop of the LSP has changed. If the next hop has changed then refreshing the LSP is briefly postponed to let the routing tables stabilise. A typical value of this hold-down timer is 2s. Note that it might be the case that the routing tables are updated fairly long ago. Since the LSP is refreshed every [0.5, 1.5]R seconds, the time between the update of the routing tables and the current refresh may be up to 45s. Even then a hold-down period is waited before the LSP is refreshed because there is no knowledge about when the routing tables were updated.

One can conclude that the soft-state mechanism that RSVP uses to adjust to network changes is an elegant solution to reroute an LSP but that this soft-state mechanism also introduces a significant delay to the convergence time.

MPLS rerouting with LDP

As explained in the previous section, RSVP uses a soft-state mechanism to reroute shortest path LSPs. LDP is a hard-state protocol so it does not use a comparable mechanism. LDP solves this problem by integrating more closely with the routing protocol. When the routing protocol has changed its routing tables, LDP expects to be informed of these changes [25]. When a next hop changes in the network, LDP will check which LSPs are affected by this change and will reroute these LSPs to take the new route into account. The benefit of this approach is that the convergence time is not delayed by a refresh period. The obvious draw back is that there needs to be an interface between the LDP signalling component and the routing protocol. This interface does not need to depend on the type of the routing protocol. The routing protocol only needs to inform the LDP component that the routing table has changed, the mechanism that was used to calculate these changes is irrelevant to LDP. Alternatively if there is no interface between the routing protocol and the LDP protocol then the LDP protocol can poll the routing tables regularly to check if something has changed. Even though this approach is less optimal than a direct communication with the routing protocol it is more scalable than the soft-state approach used by RSVP-TE. In RSVP-TE the frequency of the refresh messages affects the processing overhead in every router along the path of the LSP. Polling the routing tables in LDP only affects the local router and is only proportional to the size of the routing table and not proportional to the number of LSPs in the network. Regardless of the mechanism used, when the LSR detects that the next hop has changed the operations are similar to those in the general description of MPLS rerouting in the section 3.3.3.a.

Speeding up MPLS rerouting with RSVP-TE

We have seen how LDP requires a mechanism to actively detect the change of a next hop on the path of an LSP because it cannot depend on soft-state mechanisms. This mechanism can also be used by RSVP to speed up the LSP rerouting without sacrificing scalability. When RSVP is notified of a change in the routing tables it can force a refresh on all the affected LSPs and so the LSPs will be rerouted immediately. This speeds up the convergence time significantly without sacrificing scalability. This mode is called rerouting with *triggered refreshes* sometimes also called RSVP *fast reroute*. We deliberately refrain from using the latter term because it is too broad and it is often used for other mechanisms too. When RSVP rerouting solely depends on soft-state refreshes this is called *pure soft-state rerouting*.

3.3.3.b Explicitly routed LSPs

Rerouting shortest paths with MPLS is relatively easy because the route calculation is done by the IP routing protocol. The only responsibility of the LSRs is to notice changes in the routing tables and act upon these changes by setting up and switching over to a new downstream LSP. Supporting explicitly routed LSPs is more complex because these LSPs are not routed along the IP shortest paths so IP routing cannot be used to determine the new path towards to destination of the LSP. We now discuss rerouting ER-LSPs, we will start the discussion with ER-LSPs with only loose hops.

ER-LSPs are set up by specifying a number of hops that all need to be traversed by the LSP. When the LSP is set up, the ER-LSP is routed along these hops. IP routing is used to determine the path between the individual hops in the hop specification. When a failure occurs the path between these hops can change. If the ER-LSP is affected it needs to be rerouted in order to circumvent the failure. When a node detects that the route towards the next hop in the ER-LSP specification has changed it will set up a new downstream LSP that circumvents the failure. When this LSP is set up, the LSR will switch over to this new downstream part and the ER-LSP is rerouted. This process is quite similar to shortest path LSP rerouting with the exception that the route towards the next hop in the explicit hop specification is changed instead of the route towards the destination of the LSP.

Consider for example the explicitly routed LSP with loose hops A-B-F-G-D-E (see Figure 32). When the link BF fails the routing tables will converge and the outgoing interface in node B towards node F will change from the interface BF to interface BC. At the same time the routing tables in nodes C, D and G have also changed to take the failure into account. The label request message will then be forwarded over the path B-C-D-G towards the next hop in the LSP specification i.e. F. Node F receives the label request message with the explicit hop specification F-G-D-E. The next hop in the explicit hop specification is G so the label request message is further forwarded towards G. Similarly the label request is further forwarded to nodes D and E. Node E will reply to the label request message with a label mapping message that is forwarded along the reverse path of the label request. When the label mapping message arrives at node B the rerouted part of the ER-LSP: B-C-D-G-F-G-D-E is set up. At that time node B will switch over to the rerouted downstream LSP and the end-to-end LSP is rerouted to A-B-C-D-G-F-G-D-E. As we see this rerouted LSP contains the loop D-G-F-G-D which is highly undesirable.

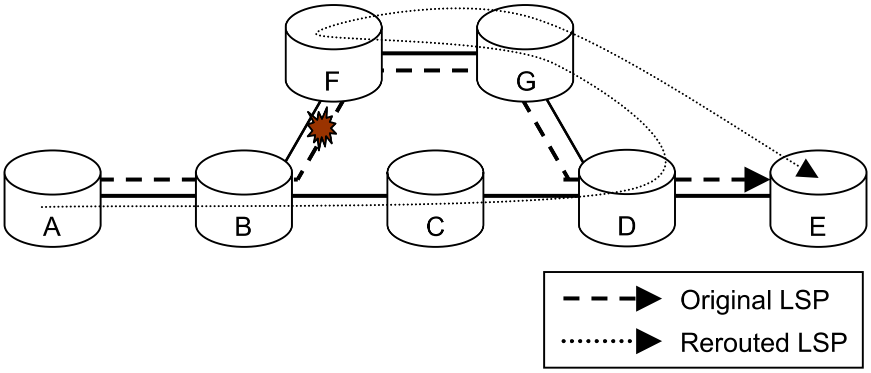


Figure 32: Rerouting an ER-LSP can lead to resource inefficiency such as unnecessary long paths and paths with loops.

This scheme works fine when the hops in the ER-LSP specification are loose. When the hops are strict this leads to problems because according to the definition of a strict hop no additional hop may be present between the strict hop and the previous hop. It is obvious from the above example that if hop F was specified strict in the original LSP specification that the LSP could not be rerouted.

There is another problem with rerouting ER-LSPs, consider for a moment that link BF and link GD both fail simultaneously. No form of rerouting can set up a rerouted LSP that obeys to the original LSP specification because the hops F and G are no longer reachable. In order to address this problem an additional flag can be used when setting up and ER-LSP. In RSVP-TE this flag is called *local protection desired* and when this flag is set then the LSP will be routed even if this means that the original hop specification is violated when doing so [30]. The mechanism that determines the new path of the rerouted LSP is not specified. A possible approach might be to remove the hops that are no longer reachable from the hop specification and to use IP rerouting for those hops that are still reachable. This will lead to recovery path A-B-C-D-E in the above scenario.

ER-LSPs with strict hops are more difficult to support since no additional hops may be present between these hops. As explained the local protection desired flag can be used to ignore the hop specification to reroute an LSP, this mechanism can also be used to reroute ER-LSPs with strict hops. Note that strict hops in an LSP specification are still useful even when the local protection desired flag is set. The local protection desired flag only states that the hop specification may be violated to *repair* the LSP. So the set up of the LSP will still fail if an additional hop is found between two strict hops of the ER-LSP specification. So using strict hops is still useful to verify that the path that is specified during the LSP set up is valid.

3.3.3.c Convergence and reversion cycle

When looking at the convergence and reversion in MPLS it is important to make a distinction between the pure soft-state rerouting as used by RSVP and the triggered rerouting as used by LDP and potentially RSVP.

Convergence cycle

The first thing to observe is that with MPLS rerouting the failure detection and notification depends on the IP routing protocol. The MPLS signalling daemon does not detect failure nor does it calculate new routes in the network. Rather the LSPs are set up according to the new routes calculated by the IP routing protocol.

Table 3: The convergence cycle for soft-state and triggered MPLS rerouting. Again the name and standard values of the parameters according to [31, 162] are shown.

	Fault detection	Notification time	Hold-down time	Recovery operation
RSVP soft-state	IP convergence	IP convergence	hold-down + [0,5-1,5]R	LSP setup (ζ)
RSVP soft-state	IP convergence	IP convergence	2s + [15-45]s	LSP setup (ζ)
MPLS triggered	IP convergence	IP convergence	hold-down	LSP setup (ζ)
MPLS triggered	IP convergence	IP convergence	2s	LSP setup (ζ)

The most interesting factor in the MPLS convergence cycle is the hold-down time. In pure soft-state rerouting the hold-down time depends on the next refresh messages of the LSP and a configured additional hold-down time. In RSVP the refresh time is randomised around the refresh period. Triggered MPLS rerouting of an LSP is performed under influence of the routing protocol. Potentially this could lead to a zero hold-down time. However, most implementations still maintain an additional hold-down period in order for the routing tables in the network to stabilise. This is certainly necessary for distance vector routing protocols since they work directly on the routing tables which means that fluctuations occur during the convergence period. But even link-state routing protocols can experience fluctuations in their routing tables just before the routing tables converge.

The last operation in the convergence period obviously is the setup of the LSP according to the new routing tables.

Reversion cycle

The reversion cycle in MPLS rerouting is identical to the convergence cycle seen from the MPLS layer. Both MPLS convergence and reversion changes the path of the affected LSPs induced by a change in the routing tables. Since the two cycles are identical we will not illustrate the reversion cycle.

3.3.3.d Conclusion

So far we have only looked at rerouting approaches for MPLS where the routes of the LSP are changed according to the IP routing tables when a failure occurs. The convergence time is always limited by the IP routing speed. In the next section we will look at a different approach to failure convergence called protection switching where the recovery LSPs are set up in advance.

3.3.4 MPLS Protection Switching

In this section we look at protection switching in MPLS networks. We start by explaining protection switching and its different modes. In the next section we will look at some of the implementation issues of protection switching.

3.3.4.a Introduction

Protection switching techniques differ quite considerably from rerouting schemes. The main difference is that during failure conditions the original working LSP is not replaced by a newly calculated LSP but that it is replaced by a pre-calculated and pre-established LSP (the *recovery LSP*). Recovery LSPs are set up in advance, therefore protection switching is sometimes also called a *make-before-break* technique. The LSR that switches over from the working LSP to the recovery LSP during fault conditions is called the *Protection Switch LSR (PSL)* (see Figure 34). The LSR that merges the recovery LSP back on the working LSP downstream of the failure is called the *Protection Merge LSR (PML)*.

When the PSL detects a failure it will switch over from the working LSP to the recovery LSP. For example when the link BC fails, node B (the PSL) will swap the outgoing label on link BC with the outgoing label on link BF so that the packets arriving on link AB are forwarded over link BF instead of link BC. The PML does not have an active role in the protection switch operation, the PML is only the topological location where the working LSP and the recovery LSP merge. In Figure 33 the PML is node D a node distinct from the egress of the LSP. This is only possible if the LSRs are merge capable. The role of the PML is to merge the incoming labels from the working LSP and the recovery LSP to a single outgoing label. If in a network the LSRs are not merge capable the PML will always be the egress of the LSP and the working LSP and recovery LSP are merged at the IP layer.

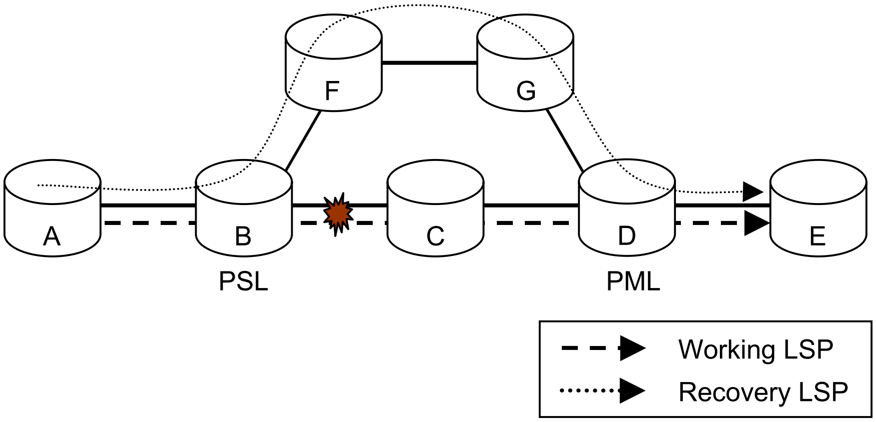


Figure 33: In MPLS protection switching the switching operation is performed by the *Protection Switch LSR (PSL)*. The *Protection Merge LSR (PML)* is the location where working and recovery LSP merge back together.

Notice that the recovery operations of protection switching, in contrast with MPLS rerouting, do not rely on IP routing to calculate a new route. Since IP routing is not used a distinct failure detection mechanism is needed. The mechanisms described in section 3.2.1 can be used including hardware based failure detection.

3.3.4.b Protection switching modes

There are two topological modes of protection switching: local repair and global repair (also called path repair).

In local repair the PSL is the LSR immediately upstream of the failure. For example in Figure 34 node B is the PSL if link BC or node C fails. Local repair has the advantage that the failure notification time is minimal. Within local repair there are two additional modes. Link protection protects a working LSP against a link failure and node protection also protects the LSP against a node failure. In Figure 34 the recovery LSP to link protect the working LSP against a failure on link BC is given by the nodes A-B-F-C-D. The recovery LSP to node protect the LSP against the failure of node C is given by the hops A-B-G-H-D.

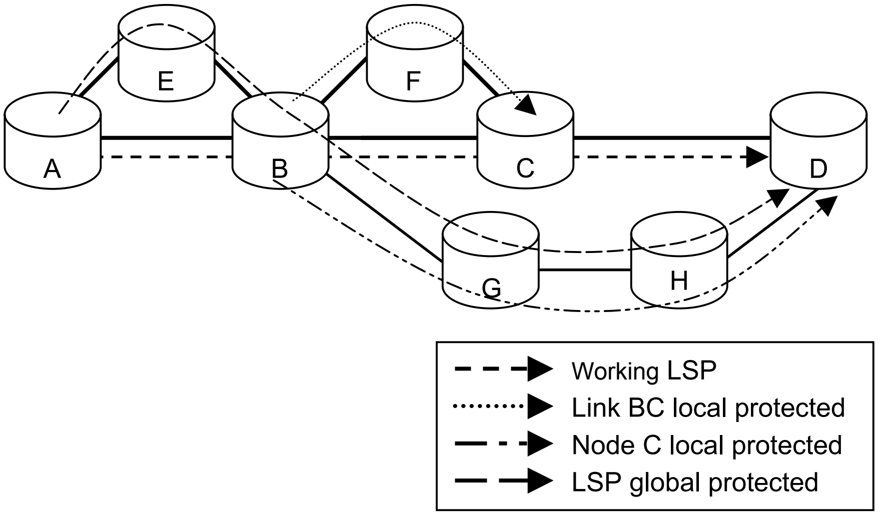


Figure 34: MPLS protection switching offers different types of failure protection. Illustrated here are how the working LSP is protected against a link failure on link BC, against a node failure on node C and globally.

Global repair tries to protect the working LSP against any link or node failure. This is accomplished by setting up a recovery LSP that is as much node and link disjoint as possible (in Figure 34 this results in the recovery LSP A-E-B-G-H-D). Global repair can recover more failures on the working path than local protection but at the cost of a slightly higher convergence time. Since the PSL is the ingress of the LSP a failure indication signal needs to be sent to the ingress which introduces additional latency. One can say that there’s a speed/coverage trade-off between local protection and global protection.

Up to now we have considered that the recovery LSP is dedicated to one working LSP. This is called *one-to-one* protection switching often denoted by 1:1. The main drawback of one-to-one protection switching is that it requires a lot of LSPs especially in local repair mode. This situation can be improved by reusing existing recovery LSPs to protect other working LSPs. For example every LSP that uses link BC can benefit from the local protection of this link over the path B-F-C (we will explain exactly how later on). In one-to-one protection, a recovery LSP over that path cannot be reused by another working LSP.

This drawback is addressed by *facility* protection switching denoted by N:1 where N LSPs are protected by one recovery LSP [122]. The obvious benefit of this approach is that the number of recovery LSPs is reduced. A drawback is that the recovery LSPs can no longer be determined on a per working LSP basis because it is shared among a number of other working LSPs and so it might be less than optimal for a given working LSP. Another drawback is that it is more difficult to determine the resources (e.g. bandwidth) to be assigned to the recovery LSP. Finally there is the N:M model where M recovery LSPs are used to protect N working LSPs (where typically M<N).

Another model that is traditionally used is the 1+1 model where traffic is sent over the working path and the recovery path at the same time. This has the benefit that if the working path fails the PSL does not need to take any action (it does not need to switch the traffic over to the recovery LSP because the traffic is already forwarded over the recovery LSP). The PML has to decide if it forwards traffic from the working LSP or the recovery LSP towards the destination. Another benefit of this approach is that reversion is also very easy to implement, when the working LSP is operational again the PML just forwards the traffic from that LSP again. A problem with this approach is that the resources need to be reserved bandwidth on both the working and recovery LSP simultaneously which is avoidable with 1:1 protection.

In our opinion the 1+1 model does not fit well with the model of label switching in MPLS (although [145] specifies it). 1+1 Protection switching requires changes to the forwarding plane since the PML must monitor the working and recovery LSPs and filter the duplicates before forwarding the packets over the outgoing interface. Normal MPLS forwarding operations act on label values and do not inspect the contents of the LSPs.

3.3.4.c Path calculation and set up

Since the recovery paths cannot be determined directly from the routing tables there needs to be some kind of path calculation algorithm that determines the paths that the recovery LSPs take. This is typically done by an off-line mechanism because determining the recovery paths on-line is difficult. For example a global recovery LSP needs to be as diversely routed (link and node disjoint) as possible to be as effective as possible. Determining such a recovery LSP on-line is not easy so off-line mechanisms are typically used. Similarly determining M different recovery LSPs for N working LSPs is also usually carried out by an off-line mechanism. Off-line mechanisms might require significant processing power and they might also require a considerable amount of time to run. A special case of an off-line mechanism is the manual input of the recovery LSP by the network administrator.

However techniques which calculate recovery LSPs on-line have been developed too [146]. In order to set up a recovery LSP on-line it is stated that the working LSP should be protected when the LSP is set up. The individual LSRs on the path of the LSP can then set up recovery LSPs based on the path that the working LSP takes, topology information and by running what is called a constraint shortest path first algorithm (CSPF) [146]. In the next section we will explain how CSPF works in combination with MPLS, afterwards we return to the topic of on-line recovery LSP set up.

In MPLS CSPF a single LSR calculates a recovery LSP by using a constrained shortest path algorithm on distributed information and then setting up an ER-LSP accordingly. The actual constraint in the calculation is for example that a certain link cannot be used. The calculation of this path is only performed by a single router in contrast with IP routing where all the routers calculate the shortest paths in the domain. By setting up a fully specified ER-LSP the route calculation does not need to be performed by other routers and the route does not need to be installed in the routing tables. The CSPF model is beneficial for a number of reasons. Since the calculations are done typically only at edge nodes (the ingress of the ER-LSP) this model achieves higher scalability because the complexity is lower and it is shifted to the edge of the network. Another advantage is that the CSPF model matches certain problem spaces better than distributed calculation. There is for example little reason to distribute the calculation of a single recovery LSP for a given working LSP over the network.

Let us now illustrate how CSPF can be used to determine a recovery LSPs on-line. For example suppose LSR A in Figure 34 wants to calculate a recovery LSP that circumvents link BC. All it has to do is remove the link BC from the topology information and calculate the shortest path from itself towards the destination of the working LSP. This results in the same recovery LSP as the one depicted in Figure 34: “Link BC local protected”. Calculating a node disjoint recovery LSP can be done similarly by removing the node from the topology information. Finally calculating a global recovery LSP can be achieved by increasing the metrics of the links used by the working LSP so the recovery LSP tries to avoid the working LSP.

The benefit of on-line calculated recovery LSPs is that they are more dynamic than off-line calculated recovery LSPs. Consider for example that a fault occurs on a *recovery* LSP. On-line mechanisms can easily re-calculate the recovery LSP while this is more difficult, or even impossible due to manual intervention, with off-line mechanisms.

3.3.4.d Preventing double booking

Protection switching pre-establishes the recovery LSPs before the failure occurs. So the working path and the recovery path exist at the same time. The paths of the working LSP and the recovery LSP can overlap. When resources are allocated on the working LSP and on the recovery LSP it is highly undesirable that these resources are double booked where they overlap.

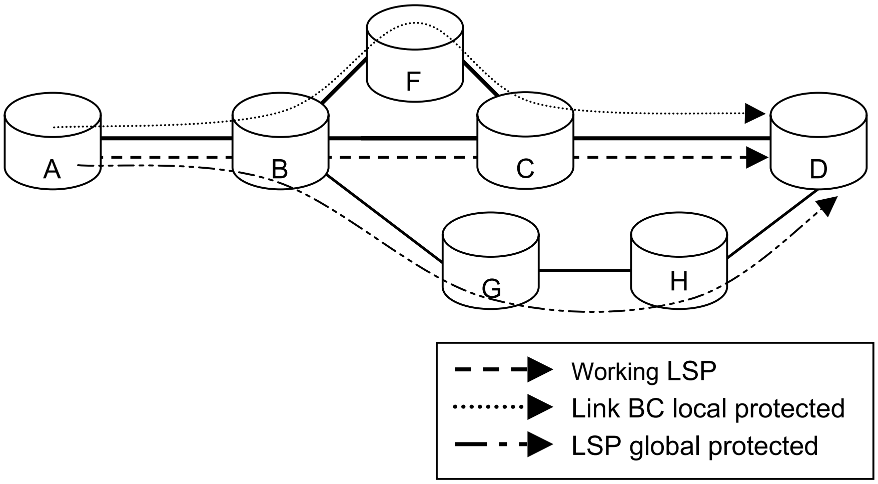


Figure 35: Special care must be taken to prevent resource inefficiency which can happen when resources are double booked on common segments of the working and recovery LSPs. In this example both the local protection LSP and the global protection LSP share common segments with the working LSP.

Double booking means that although the recovery path and the working path overlap and they will never be used at the same time the resources are still allocated twice. It is clear that double booking must be avoided. We deliberately use the generic term “resources” here but examples are labels, bandwidth and buffer space.

In Figure 35 we see that the working LSP and the global recovery LSP overlap over the link AB. Double booking should be prevented over that link. The figure also shows a local protection LSP that is set up end-to-end. Local protection LSPs can be supported by using end-to-end recovery LSPs or by using local detour LSPs. When end-to-end LSPs are used it is critical that double booking is avoided.

RSVP-TE prevents double booking with protection switching by using the RSVP *Shared Explicit (SE)* reservation style to support protection without double booking [31]. This reservation style dictates that a certain reservation is shared between a number of explicitly specified senders. The senders should be interpreted here as the ingress of the working and recovery LSPs. This mechanism elegantly solves the problem of double booking.

As we have explained it is important to prevent double booking. However it is equally important to note that reserved bandwidth can be reused if it is unused. For example, when bandwidth is reserved on a recovery LSP then this bandwidth can be used to carry other traffic during working conditions. The CR-LDP traffic parameter *weight* can be used to determine the relative share an LSP gets of this available bandwidth (see section 2.4.2.b).

3.3.4.e Bypass and loop-back tunnels

Facility recovery tries to reuse recovery LSPs for different working LSPs. This is possible if the working LSPs have a segment in common that requires protection and if there is an alternative route over that path. For example when we look at Figure 36 it is obvious that the LSPs A-B-C-D-E and F-B-C-D-E can both be node protected against a failure in node C or protected against failures on links CB or CD by using the recovery LSP B-H-I-D. This kind of recovery LSP is also called a *bypass tunnel* [122].

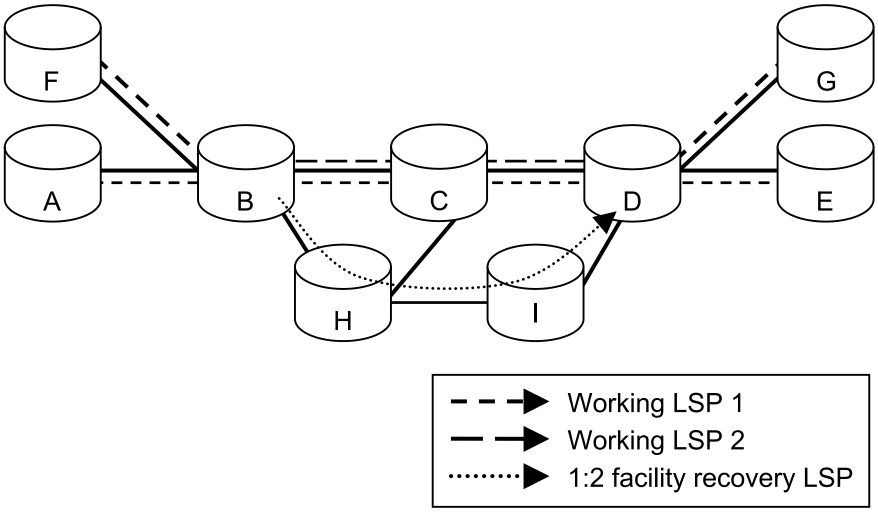


Figure 36: The two working LSPs can be protected over the segment B-C-D by a single bypass tunnel that takes an alternative path over the protected segment.

Now consider that this recovery LSP is used to recover both working LSPs and that a failure occurs on link BC. It is not possible to switch the original incoming label over to the outgoing label of the recovery LSP because the traffic would be terminated at the egress of the recovery LSP i.e. node D. This can be solved by tunnelling the original working LSPs in the recovery LSP. This is achieved by stacking the outgoing label of a recovery LSP on the current label stack (see section 2.2.6). This works when the recovery LSP only circumvents a single hop on the original working LSP, when penultimate hop popping is used (see section 2.2.4) and when the PML uses platform wide label spaces (see section 2.2.3). Let us illustrate this case first before we continue with the more generic solution.

Consider for a moment that the recovery LSP B-H-C is used instead of the recovery LSP B-H-I-D. When a failure occurs on the link BC the incoming label is swapped as normal with the original outgoing label for link BC but then the outgoing label on link BH is also pushed on top. Node H will inspect the incoming label and pop it since H is next to the last hop and penultimate-hop popping is used. The label stack now contains the label that was originally used on link BC. When node C operates with a platform wide label space it will recognise the label and forward the packet towards the destination as usual.

So far we have encountered two restrictions, first the bypass tunnel can only circumvent a single link and the merging node needs to operate with global label spaces. These two restrictions stem from the same requirement that the PSL (node B) needs to know the incoming label of the working LSP at the PML. When the bypass tunnel only circumvents a single link and when global label spaces are used then this incoming label at the PML is simply the outgoing label of the PSL.

In the more generic case the PSL needs to have a way to find the incoming label of the working LSPs at the PML (node D in the original example). In RSVP-TE the route an LSP takes can be requested and the resulting report contains both the hops that the LSP takes and the incoming labels along the LSP. This information can then be used at the PSL to set the correct outgoing label before pushing the label of the bypass tunnel to the label stack. For example when the bypass tunnel B-H-I-D is used node B will swap the incoming label on interface AB to the incoming label of node D on interface ID before pushing the label of the bypass tunnel on the label stack.

Another type recovery LSP is called a *(local) loop-back tunnel* and it is illustrated in Figure 37 [47]. The important benefit of a loop-back tunnel is that multiple failures can be protected with one LSP. For example the loop-back tunnel depicted in Figure 37 protects the working LSP against link failures on link BC, CD and DE and similarly against node failures on node C and D. When a failure occurs the traffic is switched over to the loop-back tunnel. For example when the link DE fails node D will switch over to the loop-back tunnel and the recovery path is D-C-B-G-H-I-E which means that the traffic travels back upstream before it takes a disjoint path to the egress of the working LSP. The drawback of this approach is that the latency on the recovery path is higher than on a dedicated recovery LSP. When a dedicated recovery LSP is used to locally protect link DE, then the recovery path is D-H-I-E, which is four hops shorter. The main benefit of loop-back bypass tunnels is that they can reduce the number of recovery LSPs.

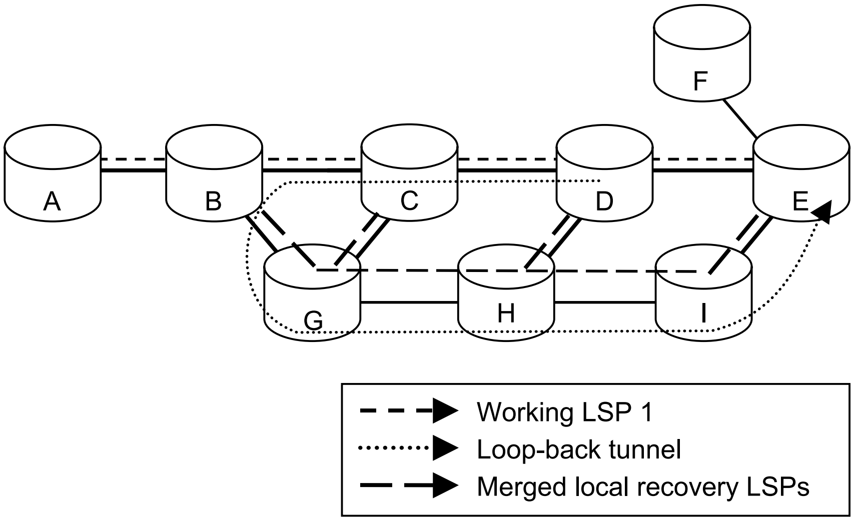


Figure 37: This figure shows two alternatives for protecting the segment B-C-D-E. The first alternative is to use a local loop-back LSP that is routed in the reverse direction of the protected segment and then takes an alternative path to the destination of the segment. The other alternative is to locally protect every link and node and to merge all these recovery LSP based on the destination.

Figure 37 also illustrates a highly merged LSP that is the result of protecting every link on the path B-C-D-E. All the recovery LSPs have node E as the destination so they can be merged based on the destination. This merged recovery LSP uses the same number of labels as the loop-back tunnel. But the merged recovery LSP offers shorter recovery paths and it does not require label stacking and label discovery. The main drawback of this approach is that the destination of all the recovery LSPs needs to be the same in order to be able to merge them. For example consider the working LSP A-B-C-D-E-F, this LSP cannot be protected by merging the same recovery LSPs because the destination of the protected LSPs is not the same. However when we use the same tunnelling technique as used in the loop-back and bypass tunnels this recovery LSP can be used to protect both working LSPs. Similarly, label stacking and label discovery are not needed to support loop-back and bypass tunnels if they are extended towards the egress of the working LSPs they protect. Of course this requires that the egress of the working LSPs is the same.

This list of protection switching techniques for IP and MPLS networks is far from complete. For example [147] proposes a scheme which uses virtual protection cycles. Virtual protection cycles are hybrid backup paths situated between a ring and a mesh. We only mention this effort briefly because it requires additional forwarding treatment on the protection cycle.

3.3.4.f Convergence and reversion cycle

In this section we will discus the convergence and reversion cycle of MPLS protection switching.

Convergence cycle

MPLS protection switching does not specify a failure detection scheme. So the failure detection can be based on any of the techniques mentioned in section 3.2.1.

More interesting is the notification time which depends whether local or global protection is used. When local protection is used then the notification time will be minimal since the node that detects the failure will also perform the protection switch. When global is used a notification message needs to be sent to the PSL which is typically the ingress of the LSP. Obviously this will increase the convergence time.

Protection switching does not need a hold-down timer if the failure detection mechanism can be relied upon. We consider the failure detection reliable when a failure event is always the proper trigger for the protection switch operation. When the failure detection is not reliable a hold-down timer can be used to ensure that the failure event remains valid over a period of time. Since we do not make any assumptions about the failure detection we will assume that the failure detection is reliable and so we do not include a hold-down time. When an unreliable failure detection mechanism is used a proper hold-down timer might be needed.

The final operation in the convergence period is the protection switch operation itself. In order to perform the protection switch the recovery LSP of the affected working LSP needs to be determined. When the recovery LSP is determined the PSL replaces the outgoing label of the working LSP in the LIB with the outgoing label of the recovery LSP. The time required to determine the proper recovery LSP for a working LSP typically depends on the number of working and recovery LSPs.

Table 4: The convergence cycle for protection switching in MPLS. The notification time depends on whether local or global protection is used.

Fault detection	Notification time	Hold-down time	Recovery operation
*	η	0s	Update LIB (ι)

Reversion cycle

The reversion operations in MPLS protection switching are identical to the convergence cycle. However before switching back to the preferred working LSP one must make sure that the working LSP is available. Since the working LSP was affected by a failure it might be necessary to set it back up before switching over to it.

Table 5: The convergence cycle for protection switching in MPLS. The notification time depends on whether local or global protection is used.

Clearing detection	Notification time	Wait-to-restore	Reversion operation
*	η	Verify working LSP Set up working LSP if necessary	Update LIB (ι)

3.3.4.g Conclusion

This concludes the section about MPLS protection switching. In this section we have spent a considerable amount of space discussing different types of recovery LSPs and the merging and tunnelling of them. A lot of attention went into techniques to reduce the number of recovery LSPs in a network because a recovery LSP only protects a working LSP to a limited extend. For example link protection protects an LSP only against a single link failure. Even global protection can be insufficient when multiple failures occur. These problems stem from the fact that the recovery LSPs need to be calculated and set up in advance for every failure scenario that needs to be covered (the *coverage*). To have full failure coverage using protection switching requires a lot of LSPs.

3.4 Conclusion

In this chapter we discussed several convergence schemes. We started out by giving a generic model for both the convergence cycle and reversion cycle. Afterwards we discussed existing failure detection schemes which can be software or hardware based. Software-based failure detection schemes typically are able to detect more failures than hardware-based failure detection schemes but they are much slower. Another drawback is that software based failure detection schemes introduce bandwidth and processing overhead. The bandwidth overhead can be reduced by relaxing the requirements on the liveness messages. Instead of requiring that protocol specific liveness messages are sent to monitor the status of the link it is also possible to monitor the status of the link with any packet. This decreases the overhead of useable bandwidth but at the cost that protocol specific failures cannot be detected anymore. Failure detection and convergence schemes are in theory orthogonal with respect to each other although for example OSPF typically uses the Hello protocol to detect failures. Note that the Hello protocol has other functions besides failure detection.

The first convergence scheme that we discussed was RIP. RIP is a simple protocol that uses distributed Bellmann-Ford algorithm to calculate the routing tables in the network. The network learns the best routes in the network by spreading distance vectors. The distance vectors can be regarded as routing tables. By selecting the best routes at each node and by only spreading this information the network eventually will converge to using the best routes only. The major drawback of RIP is that it can take a lot of iterations before the network has converged. Effects like counting to infinity and the bouncing effect can partially be solved by using split horizon (with poisonous reverse) and triggered updates. Even with these techniques anomalies can happen leading to prolonged convergence times. Another drawback of RIP is that it can only support networks of very limited size, that it only supports one metric, that multipath is not supported and that external routes cannot be injected.

The limitations of RIP are well addressed by the routing protocols of the link-state family most notably IS-IS and OSPF. Link-state routing protocols build up a map of the network and calculate the shortest paths upon it. This map, the link-state database, is populated by spreading link-state advertisements. By separating the information spread over the network and the route calculation, link-state protocols are able to achieve much higher stability and are able to support more requirements (larger network sizes, multiple metrics, multipath, external routes). However this comes at the cost of higher complexity. Despite the higher complexity, the protocol and the implementations have matured so that link-state routing protocols are the convergence scheme of choice for IP networks of significant size.

We then discussed failure convergence in MPLS networks. The first approach discussed was MPLS rerouting. MPLS rerouting depends on the IP routing protocol to calculate the working routes and to adapt the routes when failures occur. There are two variants of MPLS rerouting. Soft-state MPLS signalling supported by RSVP-TE refreshes the state of the LSPs regularly where the path of the LSPs is determined by the routing tables. When the routing tables change under the influence of the routing protocol the path of the LSPs will also take to new path after the next refresh. This main benefit of this approach is that the MPLS signalling is loosely coupled with the routing protocol. The main drawback is that the time between the update of the routing tables and the next refresh can be quite high. This problem is addressed by triggered rerouting that changes the path of the LSPs immediately after the changes are detected. This does require that the MPLS signalling component is notified of the changes in the routing tables. Both LDP and RSVP-TE support triggered MPLS rerouting.

MPLS offers an alternative convergence technique that is not available to IP networks called Protection Switching (PS). Protection switching is make-before-break scheme where the recovery LSPs are pre-established. This has the major benefit that when failures occur the recovery LSPs do not need to be determined and set up which decreases the convergence time significantly. There are two major topological modes of protection switching. In local protection switching a link or node is protected by a recovery LSP, while in global protection a path is protected by a recovery LSP. Supporting all possible failure scenarios (full coverage) requires a lot of recovery LSPs. Fortunately techniques exist to share recovery LSPs between different sets of working LSPs. These techniques depend on label stacking or merging labels on the destination. The calculation of the paths of the recovery LSPs can be done on-line or off-line.

So far we focussed on the basic operation and the ability to react to fault conditions of the convergence schemes. In the last chapter we will revisit our convergence time analysis and we will also investigate the stability, scalability and backup capacity requirements of each of the convergence schemes. But first we introduce another convergence scheme. This convergence scheme, specifically developed for MPLS, has been developed by the author.

Chapter 4

Fast Topology based Constrained Rerouting (FTCR)

4.1 Introduction

As we have seen, convergence schemes have a lower limit on their convergence time. Part of the convergence time is due to the time required to detect the failure. More relevant to this work are the other parts of the convergence cycle: the notification time, the hold-down time and the recovery operation time. These steps are the prime differentiation factors of the convergence schemes. The complexity and the length of these steps determine the convergence time. For example although we did not quantify it yet it is clear that the convergence time of OSPF will be higher than the convergence time of protection switching when the same failure detection mechanism is used. Similarly it is clear that MPLS rerouting will be slower to converge than OSPF rerouting since MPLS rerouting depends on OSPF rerouting.

In this chapter we will describe a MPLS rerouting scheme that like regular MPLS rerouting is designed to work in conjunction with OSPF. However the convergence with Fast Topology based Constrained Rerouting (FTCR) does not depend on OSPF. Instead of relying on OSPF for its convergence FTCR implements it own independent convergence cycle. This allows for much faster convergence then both OSPF and MPLS rerouting by using LSPs rather than routing tables to achieve convergence.

FTCR is a rerouting scheme which means that the recovery LSPs are determined when a fault is detected. In other words FTCR is not a make-before-break scheme. This makes it easier to support all possible failure scenarios as opposed to, for example protection switching, where it is difficult to support the full range of failure scenarios including multiple failures. The benefit of protection switching is that it can achieve much lower failure convergence times then OSPF and MPLS rerouting. We will investigate the relative speed of convergence of protection switching and FTCR in the next chapter.

FTCR is developed by the author with feedback from various people in the IBCN research group. The idea of FTCR was first published in 2000 in [3] as a new convergence scheme for MPLS networks. Note that the term FTCR had not been adopted at that time. The acronym FTCR was first introduced in [4] which describes a number of convergence schemes for MPLS in optical networks and compares the capacity requirements of them. Porting different convergence schemes for electrical MPLS, including FTCR, towards optical networks is described in [5]. Data centric optical networks and their survivability is the subject of [6]. In that article FTCR is proposed as a rerouting mechanism that speeds up the time to convergence compared to IP rerouting. Finally, the convergence times of the IP rerouting, MPLS rerouting and FTCR are compared quantitatively in [8].

This chapter starts with a description of FTCR rerouting a SP-LSP, followed by the FTCR architecture. Subsequently we will describe the modes of operation of FTCR. We will then investigate two more complex failure scenarios. We start by explaining how FTCR reroutes ER-LSPs followed by a description of how FTCR supports multiple failures. After we discussed the failure scenarios we look at reversion in FTCR. After discussing the failure convergence and the reversion we give the overall operations of FTCR. We will use flowcharts to describe the actions of FTCR in each of four different modes of operation FTCR can operate in. The chapter ends by describing the prerequisites of FTCR and drawing conclusions.

4.2 The basic operation of rerouting a shortest path LSP

In this section we describe FTCR which uses explicit-routed LSPs to reduce the convergence time by installing recovery LSPs before the routing protocol has converged. We will illustrate this with a simple scenario where a fault occurs on a shortest path LSP.

FTCR relies on the fact that if every hop of an LSP is explicitly specified then its setup does not depend on the existence of (valid) routing tables. Because of this property it is possible to set up a recovery LSP before the IP routing has converged. But to be able to set up such a recovery LSP every hop on the path needs to be determined. We cannot use the routing tables to find such a path simply because they are not valid directly after a failure. Determining these hops thus requires some kind of topology information. In networks that use a link-state routing protocol every router has the topology information accessible in his link-state database. The recovery path can then be calculated using the link-state database. We call the part of the network that has failed the *affected network* *part*. By removing the affected network part from the topology in the link-state database we get a correct view of the current topology after a failure is detected. We cannot use the link-state database directly because the failure might not be incorporated into the link-state database yet. We call the link-state database where the affected network has been removed the *adjusted link-state database*. The path of the recovery LSP can be determined by running the shortest path first algorithm on the adjusted link-state database. To run the shortest path algorithm we need not only topology information but also the cost metric of every link but this information is also available in the link-state database. The calculated path is then used to set up the recovery LSP towards the destination of the original working LSP. When this recovery LSP is set up the traffic is switched from the original working LSP to the recovery LSP like in regular MPLS rerouting. The LSR that initiates the recovery actions is called the FTCR switch LSR (FSL). This scheme is called Fast Topology based Constrained Rerouting because topology information is used to calculate the recovery LSPs that are constrained so that they circumvent the impairment and are used to speed up MPLS rerouting. The next section illustrates this process with an example.

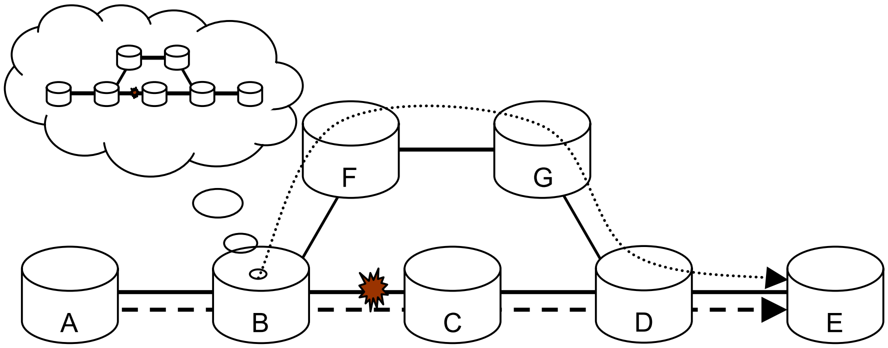


Figure 38: FTCR uses its local topology information and the information about the failure event to calculate and set up a recovery LSP before the routing tables have converged. In this example the recovery LSP B-F-G-D is set up to circumvent the failure on link BC.

When LSR B detects the failure on link BC (see Figure 38) it will check which LSPs are affected by this failure. In this case only the LSP A-B-C-D-E is affected and needs to be rerouted. The LSP rerouting is done by removing the link BC from the link-state database and running the SPF algorithm in node B on the adjusted link-state database. Only node B calculates the recovery LSP. The result of this computation is a path from node B to the egress of the LSP i.e. node E. After this computation a new route is known for the affected LSP. Note that it may be possible that such a route does not exist. If the new route does not exist then the LSR knows this and the existing LSP can be torn down. A notification message can be sent and the reason of the failure can be included in the message.

Suppose there is at least one path from the detecting node towards the destination according to the adjusted link-state database (as is the case in our example). After the calculation the recovery LSP B-F-G-D-E needs to be installed as soon as possible. Every hop on the path needs to be specified because the routing tables have not yet stabilised. After the recovery LSP has been set up this ER-LSP replaces the downstream part of the original LSP.

4.3 FTCR architecture

In the previous section we looked how FTCR can be used to reroute LSPs right after a failure has been detected. In this section we will briefly look at how FTCR needs to be integrated with the rest of the components in an LSR.

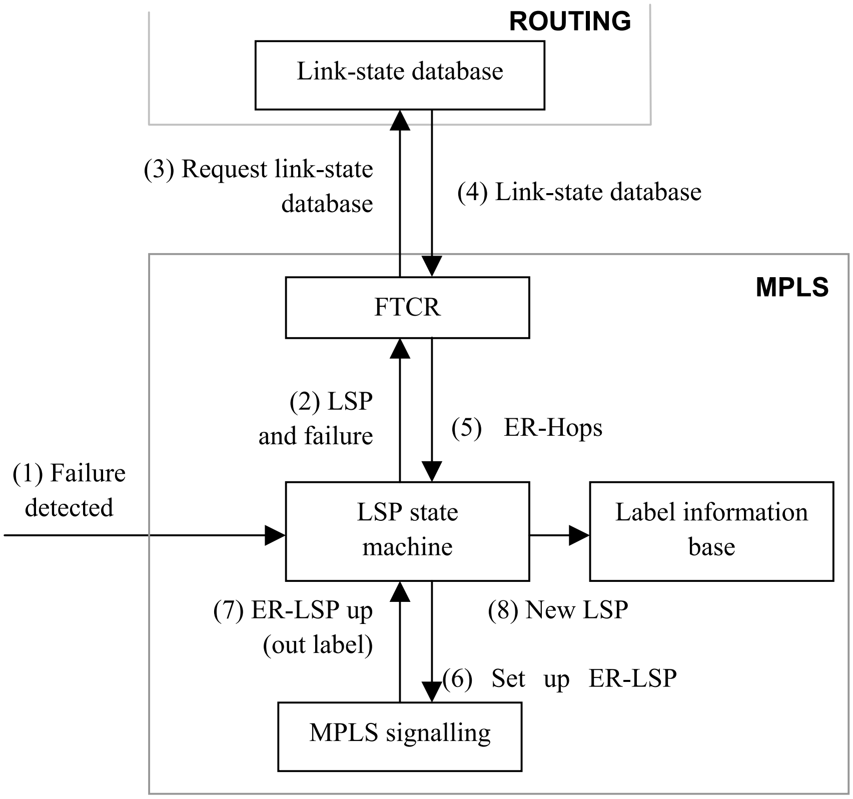


Figure 39: The FTCR architecture shows how the FTCR component interacts with the other MPLS components in this simplified view of an LSR.

When a failure is detected the LSR is notified of this failure (1), the LSP state machine will then look up if there are LSPs that are affected by the failure. If this is the case it will dictate the FTCR component to determine a new route for these LSPs that are affected (2). The FTCR component will request the topology information from the link-state database (3-4) and will use that information to calculate a new route for every affected LSP that takes the failure into account. The new routes for each of the LSP is then returned to the LSP state machine (5) so that it can set up the recovery LSPs (6-7). When the recovery LSPs are set up the new labels are installed in the label information base (8) and the traffic is restored. Note that only steps 2 through 5 are specific to FTCR.

4.4 Modes of operation

In this section we will look at the different modes of operation of FTCR. The first subsection describes the FTCR Switch LSR (FSL) selection modes which determines the topological location of the FSL on the LSP, the second subsection describes the failure presume modes which determines the assumptions the FSL makes about the types of failures.

4.4.1 FTCR Switch LSR selection modes

When we explained the basic operation of FTCR in section 4.1 we considered that the node directly upstream of the failure initiates the FTCR recovery actions (i.e. the FSL is the node directly upstream of the failure). This is only one possibility and it is called *failure-local repair* because failures are repaired locally. Failure-local repair has two main drawbacks. The first drawback is that it is possible that the failure detecting node is not able to recover the LSP and the second drawback is that failure-local repair can lead to *loops* (the doubling of a link or node). Both of these drawbacks are addressed by two other modes: *ingress repair* and *nearest non-looping repair*. The advantage of failure-local repair however is that it does not require actions from other nodes which makes it fast and it also does not depend on a notification mechanism.

In some cases the ingress of the LSP is a better or even the only choice for the FSL, this is called ingress repair. Sometimes the ingress of the LSP, the node that sets up the working LSP, is the only node that can calculate the recovery LSP. This restriction can be caused by the lack of an FTCR component in the core routers or the lack of information on how to reroute the LSP in the core routers (see section 4.5). Ingress repair is also a good solution for gradual deployment of FTCR in networks because only the edge routers need to be FTCR enabled. Finally pushing the FTCR calculations towards the edges of the network can increase the scalability of the network. The drawback of ingress repair is that a Failure Indication Signal (FIS) needs to be sent to the ingress of the LSP to notify the LSR of the failure. This increases the recovery time and the overhead slightly compared to failure-local repair. We do not specify or mandate a specific failure indication mechanism in combination with FTCR but for the sake of discussion we assume that a Reverse Notification Tree is used (RNT, see section 3.2.2).

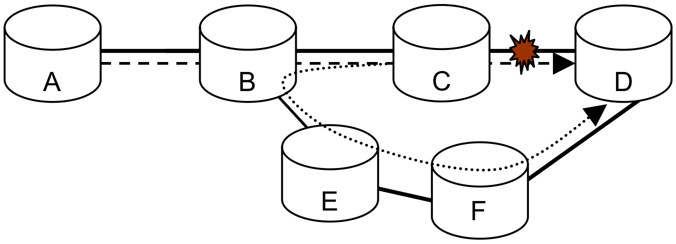


Figure 40: In failure-local repair mode the recovered LSP can contain loops as illustrated here (A-B-C-B-E-F-D). This is addressed by the nearest non-looping FSL selection mode. In this mode node C gives up the role of FSL and sends a FIS upstream. Node B will receive the FIS and will act as the FSL since the recovery LSP from node B (A-B-E-F-D) does not contain any loops.

In the previous section we addressed the first problem of local repair namely the inability of the failure detecting node to act as the FSL. The second problem of failure-local repair is that it can lead to limited looping as illustrated in Figure 40.

Consider the working LSP A-B-C-D. When the link CD fails and node C acts as the FSL in failure-local repair mode the recovery LSP C-B-E-F-D is calculated. This results in the new working LSP A-B-C-B-E-F-D that contains the loop B-C-B. Note that the creation of loops is not specific for FTCR. If we consider the ER-LSP A-B-C-D, when MPLS rerouting is used to reroute this LSP the same loop occurs. However in FTCR this problem can be addressed by the nearest non-looping FSL selection mode. In this mode the FSL is the first hop upstream of the failure detecting node that when acting as a FSL, the recovery path does not contain loops. The nearest non-looping FSL can be the detecting node, the ingress or a LSR residing somewhere between the ingress and the detecting node.

The determination of the FSL when the failure is detected is again based on the link-state database. The node that detects the failure starts by calculating a recovery LSP. If that recovery LSP contains a loop according to the link-state database then a FIS will be sent upstream towards the previous hop of the LSP. This hop will receive the FIS and will start the recovery operations. The node will again calculate a recovery LSP and check it against its link-state database for potential loops. If the recovery LSP does not contain a loop the recovery LSP is installed, if the calculated recovery LSP does contain a loop then again a FIS is sent upstream. This process is repeated until a node has calculated a recovery path that does not contain a loop. Ultimately this node might be the ingress of the LSP which is always capable of calculating a non-looping recovery LSP (unless the ingress is unreachable and thus the network is no longer connected). Since the FSL has topology information it is also possible to send the FIS directly towards the nearest non-looping FSL instead of forwarding the FIS hop-by-hop and testing for loops at every hop.

Note that selecting the FSL is a trade-off between convergence time and resource efficiency. Placing the FSL at the detecting node reduces the convergence time (no FIS needed) but it might lead to inefficient resource usage (i.e. loops).

4.4.2 Failure presume modes

Depending on the failure detection mechanism FTCR may not know what the cause of a failure is. For example if the generic heartbeat mechanism is used (section 3.2.1.b) to monitor a link and it detects that no packets has been received during the normal receive interval it will conclude that a failure has occurred. This assumption is correct but the cause of the failure is not known. It is possible that the link has failed but it is also possible that the node at the other end of the link is down and that it has taken down all its attached links. To handle node failures properly during FTCR rerouting all the attached links of a router need to be removed from the link-state database before calculating the new recovery LSP.

Note that link-state routing protocols do not suffer from the problem that the type of failure is not known. These routing protocols will distribute the information about all failed links throughout the network. So if a node fails its attached links will be declared dead and the routing protocol will route around all the affected links to calculate the new working paths. The drawback of this approach is that the flooding of this information takes time and thus increases the convergence time.

Following from the fact that FTCR cannot always rely on the fact that the type of the failure is known four different failure presume modes exist.

The first mode is called *failure-type-known*, this mode is the easiest to support, in all cases the FSL is notified of the failure and its type. This also means that every time the most efficient decision can be taken. If the type of failure is not known the three other modes are applicable.

In *assume-node-failure* the detecting node always assumes that a node has failed. This is a safe assumption because FTCR will always calculate correct working paths but at the cost of being too conservative. It is possible that FTCR decides that there is no recovery possible, based on the incorrect assumption that a node has failed but that in reality a link has failed and that a recovery path is actually possible. It is also possible that FTCR calculates longer recovery paths than necessary. Consider for example that the link AB fails in assume-node-failure mode, FSL A will calculate the recovery LSP A-E-F-G-C for the working LSP A-B-C (see Figure 41). If the failure type was know or if the *presume-link-failure* mode was used the recovery LSP would be A-D-B-C which is obviously shorter. Assume-node-failure is suited when node failures are common.

Alternatively in *assume-link-failure* mode the FSL assumes that every failure is a link failure. This can lead to incorrect recovery paths if a node fails. For example when the node B fails in our previous example. If node A operates in assume-link-failure it will assume that the link AB has failed and it will reroute the working LSP over the path A-D-B-C which is obviously incorrect. Still assume-link-failure can be suitable when link failures are far more common than node failures.

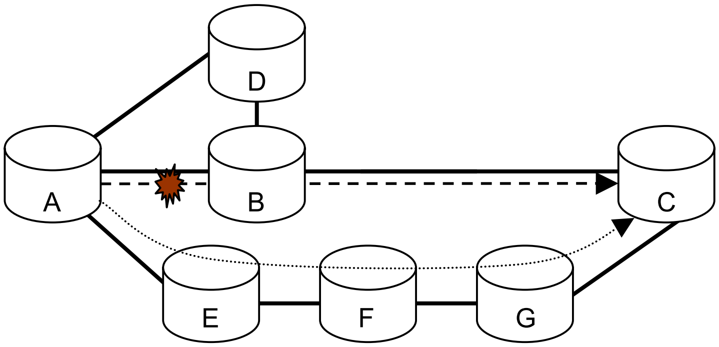


Figure 41: In assume-node-failure mode the FSL presumes that all failures are node failures. When node A detects a failure on link AB, connecting node A with node B, node A will presume that all the links of node B have failed. As a consequence the links DB and BC will not be used by the recovery LSP and a longer path over the link AE is used.

Finally in *assume-worst-case-working* mode the detecting node assumes that a node failure has occurred unless this does not lead to a recovery LSP. This mode is particularly interesting when node failures are common and when a failure is detected by the penultimate hop. In assume-node-failure mode this would automatically lead to the assumption that the LSP cannot be recovered. For example when node B detects a failure on its interface link BC. If node B operates in assume-node-failure mode it will assume that rerouting the working LSP is impossible since the egress of the LSP has failed. However in assume-worst-case-working mode the FSL will still try to recover the LSP in this case by assuming that the link has failed. If that fails because the egress node has indeed failed no real harm is done. If on the other hand it was the link that failed and another path between the detecting node and the egress exists then the recovery LSP will be set up and the LSP is recovered.

Yet another idea might be to try a node disjoint path first and if that fails try a link disjoint path for the recovery LSP. This scheme has the drawback that detecting that an LSP set up has failed is slow. This can be solved by setting the two alternatives in parallel and tear down the least preferred one if the two succeed.

So far we have discussed the basic operation of FTCR, its architecture and its different modes of operation. In the next section we will look at more complicated forms of rerouting like multiple failures and the reversion operations but first we will look at how ER-LSPs can be rerouted.

4.5 Rerouting Explicitly Routed LSPs

As we have seen FTCR can reroute shortest path LSPs by calculating a new path from the FSL towards the destination using the adjusted link-state database. This section describes the rerouting of Explicitly Routed LSPs.

4.5.1 Problem statement

We have seen that ER-LSPs are an important tool to implement traffic engineering for load sharing, congestion avoidance and routing policies (see section 2.6). The explicitly routed path taken by an ER-LSP usually does not follow the shortest paths as calculated by the routing protocol. Hence the reason to specify the hops that are to be traversed by the LSP. The actual path taken by the ER-LSP can be calculated by an off-line mechanism or it can be a distributed calculation. Rerouting an LSP can be considered as changing the original route of the LSP in order to circumvent a failure. This is mostly in conflict with the original specification of an ER-LSP. A trivial example of this is a fully specified strict routed LSP. Still, rerouting an ER-LSP usually is preferable even though it violates the hop specification. This is illustrated by the existence of the local protection required flag of RSVP-TE (see section 3.3.3.b). In the next section we will give four different approaches to FTCR rerouting of ER-LSPs.

4.5.2 Different approaches

There are four approaches to FTCR rerouting of ER-LSPs. We will describe these four approaches in the context of a single link failure on a fully specified strict ER-LSP. FTCR operates in *presume-link-failure* and *failure-local-FSL* modes. A subsequent subsection will describe the issues with the other FTCR modes, with partially specified ER-LSP and with ER-LSP with loose hops.

The calculation of the recovery LSP of an ER-LSP is preferably based on the same algorithm that calculated the working ER-LSP. Of course the calculation should be based on updated topology information (e.g. applied on the adjusted link-state database). Calculating the recovery LSP with the same algorithm might not be possible though. An obvious example is an ER-LSP that is specified through user intervention. But even if the ER-LSP is algorithmically calculated, it might not be possible to calculate the recovery ER-LSP at the FSL. First of all, the algorithm might be computationally too expensive. It might for example be based on analysis of the traffic matrix, forecasted demand, monitoring information etc [107, 108, 109, 110, 111, 112]. Also certain information might not be available at the FSL or the route calculation software might not be part of the FSL node's software. The solution to the latter two problems might be to use the *ingress FSL* selection mode. Still it is not possible to assume that either the original algorithm can be run or that it can be run in a sufficiently short time interval. So other alternatives are needed.

A possible solution is to reroute the destination of the LSP over the shortest path on the adjusted link-state database. Notice that the resulting action is the same as with SP-LSP FTCR rerouting. The benefit of this approach is that the resulting recovery LSP is easy to calculate and that the path from the detection node towards the destination has a minimal cost. However this is not an ideal solution to the problem if we take into account that ER-LSPs are a means for traffic engineering. For example the whole purpose of the LSP might be to circumvent the shortest path to balance the load more evenly over the network.

An alternative to using the shortest path to reroute the ER-LSP is to reuse the original hop specification. This is only possible to a certain degree because the network topology has changed. Certain hops may no longer be reachable while others can only be reached over a longer path. But still a recovery path can usually be found by removing the nodes that are no longer reachable and by inserting hops where necessary to reach the hops in the original specification. Remember that every hop on the recovery path needs to be specified because we cannot rely on the availability of valid routing tables. This approach can lead to very inefficient paths since we might traverse links twice: the first time to reach a hop in the original specification and the second time to reach the next hop in the specification (we will give an example of this later on).

The fourth approach improves upon the third approach by eliminating loops in the recovery path. This means that all sub-paths that are loops should be pruned from the result of approach three. Similar to the third approach, unreachable hops should be removed from the hop list and additional hops need to be inserted. This approach tries to combine the goals of the previous two approaches by creating a recovery path that tries to stay as close as possible to the original ER-LSP without creating avoidable inefficiency in the network. A drawback of this approach is that although it tries to stay close to the original path specification it has no knowledge about the algorithm that is used to calculate the working path so it might violate the original goal more than approach two.

Let us now illustrate the four approaches with an example (see Figure 42). We consider that the algorithm that is used to calculate the working path calculation tries to avoid the links on the shortest path. For an LSP from node A to node E the algorithm will yield the ER-LSP A-F-G-C-H-I-E). Now consider that link AF fails. We will investigate the results of the different FTCR rerouting approaches.

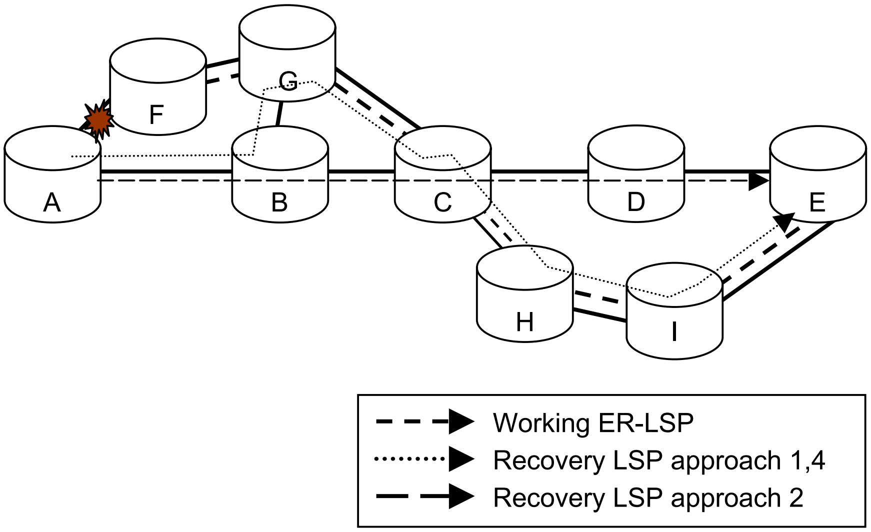


Figure 42: The different FTCR ER-LSP rerouting approaches illustrated. Approach 1 and 4 result in the same recovery LSP. The recovery LSP from approach 3 is not illustrated but the path is A-B-G-F-G-C-H-I-E.

The first approach uses the original algorithm to calculate the recovery LSP on the adjusted link-state database which leads to the recovery LSP A-B-G-C-H-I-E. The second approach calculates the shortest path from node A to node E so the recovery LSP is A-B-C-D-E. The third approach tries to reuse the original ER-LSP specification as much as possible. The first hop in the specification, node F, is no longer directly reachable so we need to insert additional hops in the original specification to create the fully specified ER-LSP. These hops are B-G and so the resulting recovery LSP is A-**B-G**-F-G-C-H-I-E. (The nodes in boldface are the inserted hops in the hop list.). We see that the recovery LSP from the third approach contains the loop G-F-G. The fourth approach optimises this recovery LSP by removing the loop so the result is the recovery LSP A-B-G-C-H-I-E.

The first approach is always preferable but, as we explained, might not be applicable all the time. The fourth approach might be preferable over the second approach and especially the third. The second approach routes the traffic in the example over the links C-D-E which is probably not ideal given that the working LSP was explicitly routed away from this link (notice that the link AB cannot be avoided). The third approach can create substantial longer paths. The first and the fourth approach have the same result in this example but this might not be the case if a different algorithm is used to calculate the working paths.

4.6 Support for Multiple failures

4.6.1 From explicit routed LSPs to multiple failures

Routing protocols typically offer good support for multiple failures. They are able to handle multiple failures by spreading information about all the failures and by calculating the new shortest path accordingly. Supporting multiple failures with protection switching is more difficult. Since all the recovery LSPs need to be pre-established it is difficult to support all failure scenarios. This is particularly true for multiple failures. When a failure happens on the working LSP a recovery LSP is selected and the traffic is switched over to the recovery LSP. In order to support multiple failures the recovery LSP needs to be protected by another unaffected recovery LSP. We will now investigate how FTCR supports multiple failures.

In the previous section we described the rerouting of ER-LSPs within the context of ER-LSP that are specified as a means for traffic engineering. Another aspect of ER-LSP rerouting is the support for multiple failures in FTCR. FTCR reroutes an LSP by setting up a recovery ER-LSP, if another failure occurs on the recovery LSP, FTCR needs to recover that recovery ER-LSP. This requires ER-LSP rerouting as explained in the previous section. However in order to support multiple failures with FTCR additional functionality is needed.

When a link fails, the FSL will then send an ER-LSP request towards the destination to recover the LSP. The FSL calculates this path on its local link-state database. Consider that another failure occurs shortly after the first failure, the FSL of the first failure does not have information about the second failure and does not take it into account when rerouting the LSP. If the second failure occurs on the path of the recovery LSP then that recovery LSP is obviously affected by it and cannot be used. This can be handled by the FSL of the *second* failure by rerouting the ER-LSP *request*. So in order for FTCR to cope with multiple failures we need to support ER-LSP rerouting and ER-LSP request rerouting. Note that we need to support ER-LSP request rerouting to properly handle the occasion where a FSL detects a failure and a second failure happens immediately afterwards on the path of the recovery LSP even before the recovery LSP has been set up. If a failure occurs after the recovery LSP has been set up the second failure will be recovered with an ER-LSP reroute.

As described above the FSL should not only reroute existing LSPs on failure events but they should also reroute the ER-LSP requests. If a FSL receives an LSP request routed over his directly downstream affected part he will need to reroute that request over a new path calculated using his adjusted link-state database. The four different approaches of ER-LSP rerouting (see section 4.5.2) are again applicable to ER-LSP request rerouting. However the same approach should be used for both LSP and LSP request rerouting. Note that LSP request rerouting also solves the problem of setting up an LSP during a failure condition.

An example

We will now illustrate how multiple failures are handled in FTCR. We reuse the same topology as in the ER-LSP rerouting example in the previous subsection. As routing approach for the working path we take the shortest path first algorithm. The working LSP is an LSP from node A to node E (A-B-C-D-E). Now consider that both link AB and link CD fail, we will now examine the different timings i.e. link AB fails before link CD and visa versa. We start by examining what happens when the upstream link fails before downstream link i.e. when link AB fails before link CD.

When link AB fails node A will send a request for the recovery ER-LSP A-F-C-D-E to circumvent the failure on link AB (see Figure 43). Now when the link CD fails there are three sub-cases (example 1-1 and 1-2). If the recovery ER-LSP from node A has been set up when the second failure occurs, node C (the FSL for the second failure) will reroute this recovery ER-LSP. This results in the recovery ER-LSP C-G-H-E and the resulting LSP will be A-F-C-G-H-E.

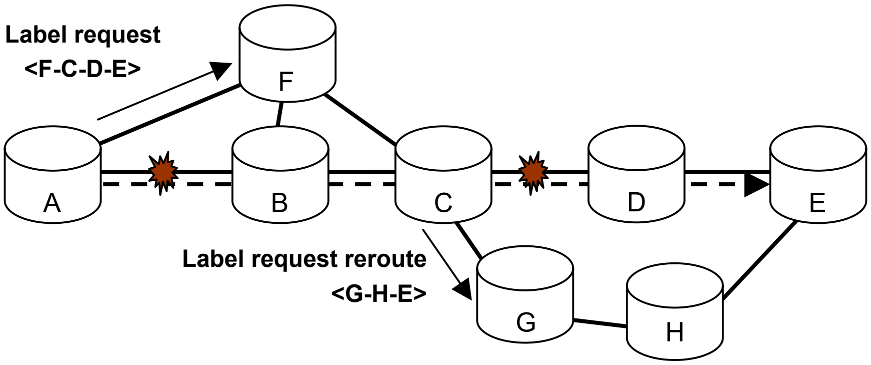


Figure 43: Multiple failures are supported in FTCR by having more than one FSL, one per failure. Illustrated here is that node A and B are FSL for the failures on link AB and CD respectively. Here node C needs to reroute the label request of node A in order to circumvent the failure on link CD.

In the other sub-case the ER-LSP from node A has not been set up when the second failure occurs. When the request from node A reaches node C, it has been notified of the second failure. Node C acts as an FSL for the second failure and uses its adjusted link-state database to calculate a new path for the ER-LSP request. In this example this will lead to ER-LSP request reroute over the nodes C-G-H-E leading to the same recovery LSP A-F-C-G-H-E as in the first sub-case (example 1-2).

The third sub-case is where node C receives the label request from node A and the link CD has failed but the failure is not yet detected. Node C will then send the label request over link CD and it will be lost. The label request will remain pending since it will not be acknowledged with a label mapping. When the failure is detected node C will reroute the label request from node A along G-H-E which results in same the recovery LSP as in the other sub-cases (A-F-C-G-H-E). This process is called pending request rerouting and we will return to this topic after the example.

Now let us examine what happens when the downstream failure happens before the upstream failure. In this second case we examine the event when link AB fails after link CD. When link CD fails node C will reroute the working LSP to A-B-C-G-H-E. This is a regular FTCR LSP reroute. Now there are again two sub-cases regarding the timing of the second failure (example 2-1 and 2-2). When the link-state database of node A already took the failure of link CD into account when link AB fails then node A can properly circumvent both failures. In the example this means that node A will set up the recovery LSP A-F-C-G-H-E. If this is not the case we have to rely on node C to reroute the ER-LSP request of node A properly. In this case node A will try to reroute the original LSP over the path A-F-C-D-E. When this request reaches node C it will reroute the request over C-G-H-E and the resulting recovery LSP is again A-F-C-G-H-E. Note that if FSL C uses approach three to reroute ER-LSPs that the recovery LSP will be A-F-C-G-H-E-D-E. Note that in this example both FSLs are located on the working LSP but this is not always the case.

We have given an example of rerouting multiple failures with FTCR. Rerouting an LSP when multiple failures occur works in general because there is always a node acting as the FSL for a failure that will either reroute the current LSPs or will reroute the ER-LSP requests. Yet there is a small period between the failure event and its detection where the recovery still can fail. This is also true when setting up an LSP just after a failure but before it is detected so it is not really FTCR specific. It is however related to recovering from multiple failures so we will address how this problem can be solved by FTCR.

4.6.2 Rerouting pending label requests

When a recovery LSP request is sent over a failure that has not been detected the ER-LSP will not be set up and the request remains pending. When the FSL is notified of the failure, the FSL needs to reroute the pending LSP request by calculating a new route for it. This is called pending label request rerouting and it was illustrated previously in the example. In the example we explained a label request remains pending because the request was sent over a faulty link. It is also possible that the label request remains pending because the label mapping is lost when it propagates back from the egress towards the FSL of the yet to be detected failure. These two scenarios are illustrated in Figure 44.

When a node is notified of a failure on a link over which a label request is pending it will need to reroute that pending request. The rerouting of a pending request is again based on the failure information and the topology information (for more detailed information about pending request rerouting we refer to section 4.8.1).

Now, let us have a look at the result of a rerouted pending label request. As stated before pending label requests happen between the time a failure occurs and the time the failure is detected. In this time interval the LSP request is sent over an incorrect path. When the failure is detected this situation is corrected and the request is rerouted over the correct path. As far as the actual result of the pending request reroute is concerned this rerouted label request is equivalent to a rerouted label request that would have happened if the label request arrived after the failure has been detected. The only difference is a difference in timing since pending label requests can only be rerouted after the failure is detected. In further discussions we do not make a distinction between label request rerouting and pending label request rerouting unless specifically stated.

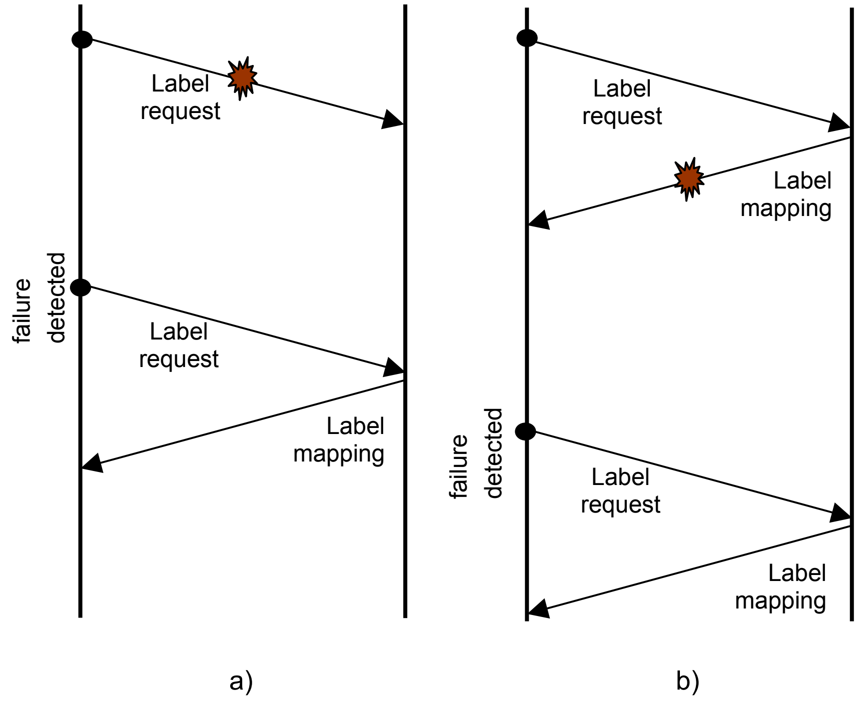


Figure 44: Pending label request rerouting. When a failure happens shortly after a label request has been sent, the failure can affect the LSP set up: a) the label request can be lost, b) the label mapping can be lost.

4.6.3 Generalising

So far we have covered a number of cases of rerouting multiple failures in FTCR (two cases with each two sub-cases). We will now formalise these cases and investigate if there are any significant cases that have not been covered and in which FTCR multiple failure rerouting specified so far may not suffice. Note that we will investigate failure-local repair only, the discussion for ingress repair and nearest non-looping is postponed till later.

The notation used in the rest of this subsection is summarised in Table 6. The first two symbols are used to describe the failure event and the detection by a node of this failure respectively. The detection by a node of a failure might be direct via a failure detection mechanism or indirect via a FIS message or via link-state updates. The table further shows a symbol for a label request and for the rerouting of a label request. Finally the symbol L is used to indicate that the LSP has been restored. The symbol always applies to the last label request.

Table 6: This tables shows the notation used for the different events during FTCR rerouting with multiple failures.

Notation	Description
F_1	Failure 1
d_{A1}	Node A detects failure 1
LR_{A1}	Node A requests an LSP to circumvent failure 1
R_CLR_{A1}	Node C reroutes the label request from node A
L	LSP set up and traffic restored

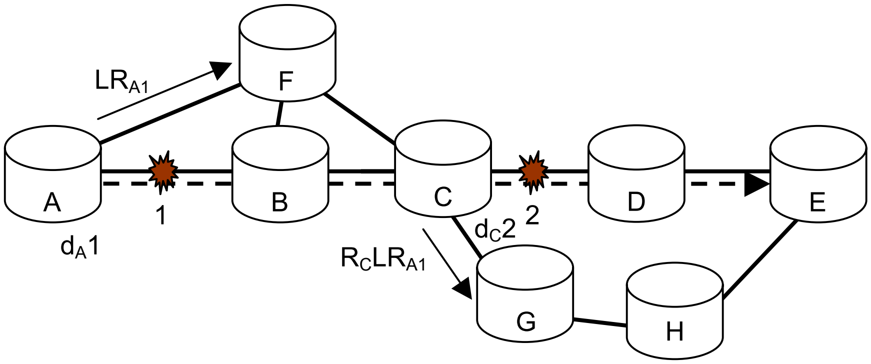


Figure 45: This figure illustrates some of the notation introduced in Table 6 for the example given before where node C needs to reroute the label request of node A to circumvent the failure on link CD.

Figure 45 uses this notation to illustrate some of the events of the first case and the second sub-case where the recovery LSP from node A is requested and the request is subsequently rerouted by node C to circumvent the second failure on link CD.

The first two examples and their sub-cases are noted in Table 7. The second column contains the events that happen during each of the examples and the third column gives a simplified explanation of the actions that are taken accordingly.

Table 7: Events and actions for multiple failure recovery in FTCR for the previous examples. The Events column contains the external events like failures and the resulting actions by the LSRs while the Actions column only contains the (simplified) actions by the LSRs.

	Events	Actions
Ex1-1	F_1 d_{A1} LR_{A1} L F_2 d_{C2} LR_{C2} L	FTCR reroute in A, FTCR ER reroute in C
Ex1-2	F_1 d_{A1} LR_A F_2 d_{C2} $R_C LR_{A1}$ L	FTCR reroute in A, FTCR request reroute in C
Ex2-1	F_2 d_{C2} LR_{C2} L d_{A2} F_1 d_{A1} $LR_{A1,2}$ L	FTCR reroute in C, FTCR reroute (2 failures) in A
Ex2-2	F_2 d_{C2} LR_{C2} L F_1 d_{A1} LR_{A1} $R_C LR_{A1}$ L	(FTCR reroute in C) FTCR reroute in A, FTCR request reroute in C

Now Table 8 simplifies Table 7 further by only looking at the FTCR actions and splitting them up in two columns.

Table 8: The recovery actions by the LSRs as a result of the failure scenarios FTCR action sequences for the examples

	Action 1	Action 2
Ex1-1	Reroute in A	Reroute in C
Ex1-2	Reroute in A	Request reroute in C
Ex2-1	Reroute in C	Reroute in A
Ex2-2	Reroute in A	Request reroute in C

The first observation is that the “Action 1” column always contains a simple reroute (R). The reason for this statement is simple, since this action is performed first, it is a reaction to a failure event. This means that it can only be a normal reroute operation since the alternative, request rerouting, is a reaction to incoming label request (this is a *temporal constraint* that dictates that the first action should be a simple reroute). The second observation is that the recovery actions done by node A are all regular rerouting. This observation can be explained by looking at the position of node A in the network. Since node A is the most upstream FSL in the network this means the node A will not receive incoming label requests from a downstream FSL that needs to be rerouted (this is a *topological constraint* that states that the most upstream FSL performs only regular rerouting). When we combine these two observations we conclude that all the possible event sequences are given by Table 9.

Table 9: This table gives all the distinct action sequences for the different failure scenarios with two failures.

Action 1	Action 2
Reroute in A	Reroute in C
Reroute in C	Reroute in A
Reroute in A	Request reroute in C

We have given the possible event combinations for two failures. What happens when more than two failures occur? This leads to a significant number of action sequences. But the temporal and topological constraints still apply so LSP rerouting and request rerouting are sufficient to support SP and ER-LSP rerouting even with more than two failures. For example when three failures occur on an LSP then the three FSLs will either reroute the working LSP or the recovery LSP (or its request).

4.6.4 Nearest non-looping and ingress repair

We conclude this section by investigating multiple failures in ingress repair and nearest non-looping repair modes.

In the previous sections we described how ER-LSP, ER-LSP rerouting request and LSP request rerouting are used to support multiple failure recovery in FTCR. So far we only covered the failure-local repair mode. Ingress repair and non-looping repair modes are more complex to support because they require that a FIS is sent upstream towards the ingress of the LSP. The problem is not the forwarding of the FIS itself because this is also required to support the rerouting of single failures in FTCR. The problem is that the reverse path that the FIS takes towards the FSL can be affected by a failure.

For example if you look at the example in Figure 46 and consider that link DE fails before link BC with ingress repair. When node D detects a failure on the link DE it will sent a FIS towards the ingress of the LSP (node A) since it is running in ingress repair mode. Node C will forward the FIS further upstream towards node A. Now consider that at the time node C receives the FIS from node D, link BC already has failed. If node C forwards the FIS over link BC the FIS will be lost, the ingress will not be notified of the failure and the LSP will not be recovered.

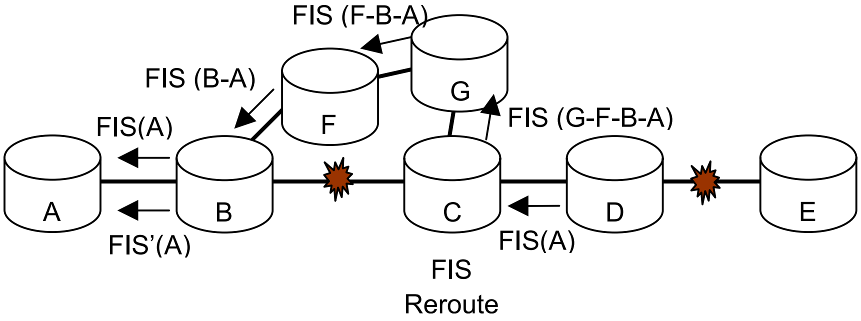


Figure 46: When a failure is detected in ingress repair mode (and sometimes in nearest non-looping mode) a FIS is sent upstream. When multiple failures occur it is possible that the path that the FIS takes upstream is affected by another failure. In order to circumvent this failure the path taken by the FIS may need to be adjusted and explicitly specified. This is illustrated for the FIS originated by node D and send towards node A. Node C needs to adjust the path of the FIS to circumvent the failure on link BC. At the same time node B also sends a FIS (denoted FIS') for the failure on link AB.

In order to forward FIS messages properly during failure conditions we need yet another form of FTCR rerouting called *FIS rerouting*. It is obvious that the FIS should be forwarded over the path G-F-B-A. This path can again be calculated on the link-state database of the FSL i.e. node C. In order for this to work it is required that the path of the FIS can be explicitly specified. Just forwarding the FIS over the correct outgoing interface is not enough because the next hop or another upstream node might send the FIS back. For example when node C detects that the FIS towards node A needs to be rerouted, it is not sufficient to forward the FIS over the interface CG because node G can send the FIS back. Therefor node C forwards the FIS according to the explicit path G-F-B-A (see Figure 46).

Notice that the RNT (see section 3.2.2) does not support explicit routes in the FIS messages. However the *Notification message*, the actual FIS message sent over the RNT, that is proposed in [132] can easily be extended to encompass an explicit route object (ERO). The ERO can then be used to pin the route of the

FIS messages during multiple failure events (as illustrated in Figure 46). This would mean that the RNT changes when multiple failures occur on the working path.

Notice that if a path towards the ingress does not exist then the ingress cannot be notified of the failure of the LSP and the LSP cannot be recovered. But if such a path does not exist then that means that the network is no longer connected so recovering the LSP is no longer possible at all.

As final note we remark that similarly to pending request rerouting, it might be beneficial to re-send a FIS when a failure is notified directly after a FIS has been sent. For example when node C has forwarded the FIS of node D towards A over the interface BC and immediately afterwards detects a failure on that interface. In the event a RNT is used this is done automatically because the originator of a FIS message i.e. node D regularly resends the FIS.

Now that SP-LSP and ER-LSP rerouting and rerouting multiple failures have been covered we will look at the reversion operations of FTCR in the next section.

4.7 Support for Revertive mode

We start the discussion about reversion operations in FTCR in the context of a single failure on a shortest path LSP and extend the discussion to include explicitly routed LSPs and multiple failures after that. Note that the term FTCR switch LSR (FSL) can be used to denote routers that switch from working paths to recovery paths but also for routers that switch from recovery path back to the original working path. So the term FSL is still applicable in this section.

4.7.1 Single failure on a shortest path LSP

The reversion operations start when the fault on the original working LSP clears. The switch from the recovery LSP back to the original working LSP can only happen when the original LSP has been restored. Most probably the original LSP has been torn down because of the failure on its path. FTCR does not mandate that the part of the LSP downstream of a failure will be torn down but it most probably will be torn down. For example in RSVP-TE the LSP will be torn down due to lack of refreshes and in CR-LDP the LSP will be torn down because of the lack of HELLO or KEEPALIVE messages. Since the original LSP has been torn down we need to set it back up and in order to set it back up we need the specification of that LSP.

Once we have received the failure clearance through the receipt of a FRS we can try to set up the original LSP. However when a failure has been cleared that does not mean that the routing tables have been restored. The set up of shortest path LSPs depends on valid routing tables. It is recommended that a timer postpones the set up of the original LSP after the failure has been cleared. This *wait-to-restore timer* should be high enough to encompass for the convergence time of the IP routing tables. When the FTCR is notified of the new routing tables that are the result of the clearance of the fault then this is the trigger for the reversion operations. If this is not the case then a high enough wait-to-restore timer should be used. The value should be at least as high as the maximum convergence time. Notice that the reversion is usually not as time critical as recovering from a failing working path.

It is not mandatory that a wait-to-restore timer is used. When the failure clearance is given the FSL can try to set up the LSP immediately but this will likely fail the first time. Repeated tries will eventually succeed. There is a small chance that the LSP has been set up along a suboptimal path due to routing anomalies possible during the convergence time interval. After the anomalies have been corrected MPLS rerouting will adjust the path of the LSP to the shortest path. This additional rerouting can lead to extra packet reordering and can be avoided by using the wait-to-restore timer. One can take it one step further and try to set up working LSPs continuously while the recovery LSP is used. Again the same problems apply so the only application is when no failure clearance signal is available. In the further discussion we will consider that a wait-to-restore time depends on the trigger received from the routing protocol. Note that in this case strictly speaking there is no wait-to-restore timer but a wait-to-restore trigger. We will speak of a wait-to-restore timer nevertheless and we assume that this case covers both the wait-to-restore timer and wait-to-restore trigger cases.

When FTCR has rerouted around a single failure then a single FSL can revert back to the original LSP. Once the original working LSP has been set up the traffic can be switched back to the original working LSP at this FSL. This normally will not introduce packet loss certainly when a proper wait-to-restore timer is used. It can introduce packet reordering for example when the original LSP is shorter then the recovery LSP. When the traffic is switched back to the original working LSP the recovery LSP can be torn down. The resulting LSP end-to-end is then routed along the shortest path and cannot be distinguished from a normal shortest path LSP anymore.

In the next subsection we will describe the reversion operations for a single failure on an ER-LSP, multiple failures are handled after that.

4.7.2 Single failure on an explicitly routed LSP

When FTCR reroutes an LSP it replaces the downstream part of the original LSP by an ER-LSP that circumvents the failure. In revertive mode this downstream part needs to be restored. To be able to restore the downstream part of an ER-LSP the ER hops need to be stored when the ER-LSP is rerouted. This is not needed for shortest path LSPs because they are only determined by their destination. Another difference is that the wait-to-restore timer is not needed to reverse a fully specified ER-LSP because setting up a fully specified ER-LSP does not depend on valid routing tables. Notice that this is the basic mechanism on which FTCR operates. However this does not mean that the wait-to-restore can be removed for every ER-LSP, only the fully specified ones. When rerouting a SP-LSP, FTCR always uses a fully specified ER-LSP as the recovery path. Now when reverting an ER-LSP we cannot rely on the fact that the working ER-LSP is fully specified so the optimisation to skip the wait-to-restore is not generally applicable. Furthermore removing the wait-to-restore timer can cause instability (we discuss the stability of FTCR in the next chapter section 5.5.4). Still it is possible to use this optimisation when the FSL knows that all the ER-LSPs are fully specified. The FSL can test whether or not the ER-LSP is fully specified by examining the ER hop list and the link-state database. If according to the link-state database every hop in the ER hops specification is directly connected to its predecessor then the LSP is fully specified and the wait-to-restore timer can be skipped otherwise the wait-to-restore timer should be used. The practical use of this optimisation can be questioned because the reversion operations are in general not so time critical.

4.7.3 Multiple failures on a shortest path LSP

When more than one failure occurred on the working path there is typically more than one FSL. Because the FSLs act independently there is no communication or synchronisation between them. Similarly in revertive mode no co-ordination between the FSLs is needed. This does not imply that packet loss or additional packet reordering is introduced. We will illustrate this with an example in the next paragraph (see Figure 47).

We consider that the original working LSP A-B-C-D has been recovered twice, once to circumvent a failure on link AB (failure 1) and a second time to circumvent a failure on link CD (failure 2). The resulting recovery LSP is then A-E-B-C-F-G-D.

We will examine the case where the downstream failure (failure 2) is cleared first. When node C receives the clearance for the failure 2, it will start the wait-to-restore timer and when the timer expires it will set up a shortest path LSP towards node D. When this LSP has been set up it will switch over the traffic to this LSP. The resulting LSP at this moment is A-E-B-C-D. Then node A receives the failure clearance for failure 1. After the wait-to-restore timer it will send a request for a shortest path LSP towards the destination. This will result in the shortest path LSP end-to-end i.e. A-B-C-D.

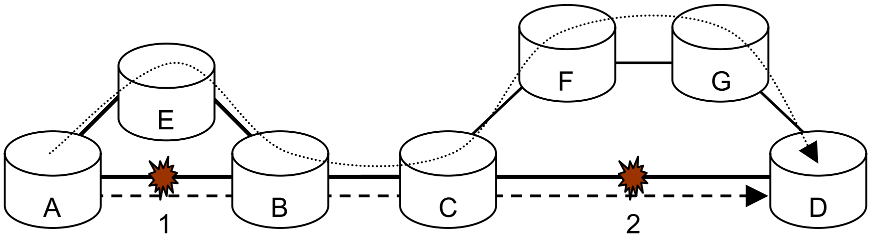


Figure 47: This figure shows the recovery LSP after the working LSP has been recovered from two failures. When the faults are cleared the working LSP will be restored through the reversion operations.

Now we will investigate what happens when the upstream failure (failure 1) is cleared first. First node A receives the clearance for failure 1, after the wait-to-restore timer it will set up a shortest path LSP towards node D. When node C receives this request it will forward the request over the shortest path F-G-D because the failure on link CD has not cleared yet. This results in the LSP A-B-C-F-G-D. Now when node C receives the failure clearance for the link CD it will restore the original downstream part of the LSP by sending a request for a shortest path LSP towards the destination. When that LSP is set up node C will switch over to the newly established LSP. The resulting LSP is then A-B-C-D.

In these two cases FTCR in revertive mode operates fine, but what happens when the partially reversed LSP, i.e. the LSP that has been set up to take into account the clearance of a first fault, is not routed over the second FSL? Again we illustrate this with an example (see Figure 48).

The shortest path LSP from node A to node D is A-B-C-D. When the link between node A and B fails the LSP is rerouted over A-E-B-C-D. When the link between node C and node D fails the LSP is routed over the nodes A-E-B-C-F-G-D (Notice that we consider the failure local FSL selection mode).

Now consider that the failure on link AB is cleared, after the wait-to-restore time node A will set up a shortest path LSP towards the destination node D. This LSP will *not* be routed along node C, instead it will be routed along the path A-B-H-I-D. This path is the result of shortest path routing, since the wait-to-restore timer has expired the routing tables have incorporated the link AB back in the network.

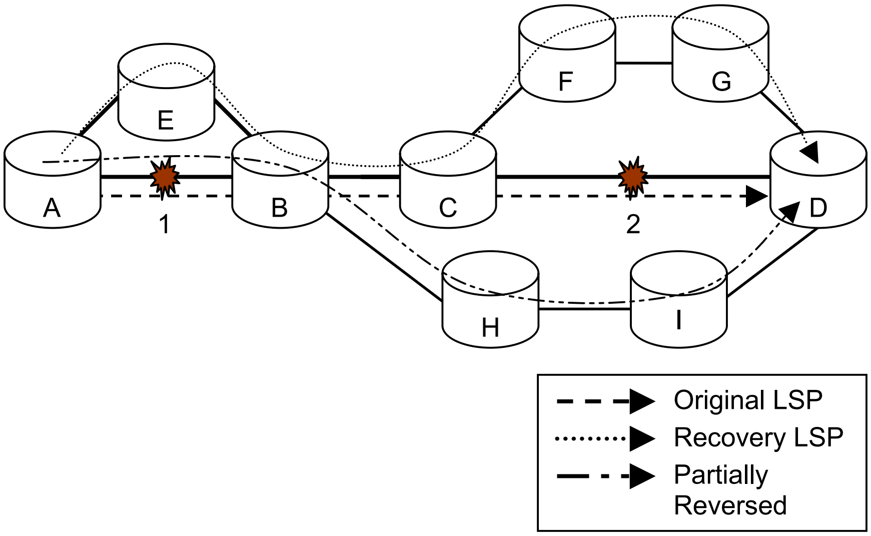


Figure 48: This figure illustrates the recovery LSP for the working LSP after two failures have occurred. When the fault 1 clears the LSP is partially reversed back to the original working shortest path. As illustrated here that may mean that this partially reversed LSP no longer is routed over the downstream FSL.

Now when the failure on link CD clears node C will not reroute the LSP because it is no longer routed through that node. Instead normal MPLS rerouting will reroute the LSP along the shortest path A-B-C-D. As in the previous two cases there are two reversion actions but in this case only one is performed by an FSL (node A) the other one is performed by regular MPLS rerouting under influence of the changed next hop in node B.

4.7.4 Multiple failures on an explicitly routed LSP

In revertive mode shortest path LSPs constitute a more complex scenario to support than fully specified explicitly routed LSPs. As explained in the previous section there is no need for the wait-to-restore timer when dealing with a fully specified ER-LSPs. Also the event where the partially reversed LSP is set up according to a path that does not contain the downstream FSL is less common with fully specified ER-LSPs. It is however still possible that a FSL is not on the recovery path for example when the FSL is disconnected from the rest of the network.

Regardless of the small differences the reversion operations for multiple failures on explicitly routed LSPs are similar to these for reversion of multiple failure on shortest path LSPs.

4.7.5 Ingress repair and nearest non-looping repair

We conclude the section about the reversion operations in FTCR with ingress repair and nearest non-looping repair. In the ingress FSL selection mode the ingress is always the FSL. When a failure recovery signal is received by a node it should send the FRS upstream towards the ingress of the LSP. It is possible that certain nodes need to consult their link-state database to find a route towards the ingress (FIS/FRS rerouting) when other failures are not yet cleared. When the ingress receives the FRS it will take the fault clearance into account and after the wait-to-restore timer it will send a new LSP request towards the destination. The LSP will most probably not be routed over a failing link because the link-state database has been updated during the wait-to-restore time. However if the label request is routed over a fault then the regular request rerouting mechanism will cope with it. Similar observations are valid for nearest non-looping FSL.

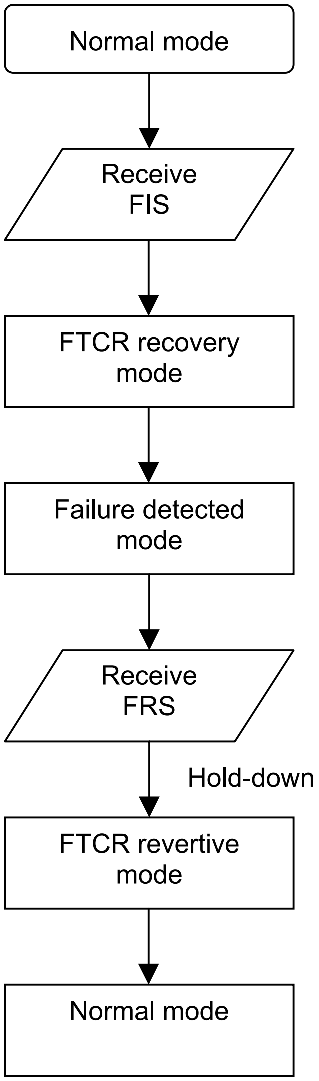
So far we described FTCR in individual situations this might give the idea that supporting the different modes under different failure and reversion conditions is very complicated. However in the next section we will give the general FTCR operations within a single router for all these cases.

4.8 FTCR operations

We started the FTCR section by describing the basic operation of FTCR, afterwards we have described all the necessary extensions to support ER-LSPs and multiple failure rerouting and reversion. Now we will investigate the overall operation of FTCR i.e. how does a single node support LSP rerouting, LSP request rerouting, FIS rerouting and the respective reversion operations?

The FTCR router can be in four different operational modes. This is illustrated in Figure 49 with a flow chart [148, 149]. The first mode is the *normal mode* where no special actions are taken and the router acts as a normal LSR. When a failure is detected the router enters the *FTCR recovery* mode and starts to recover all the affected LSPs. When all the affected LSPs are recovered the router enters the *failure detected mode*. In this mode the router will reroute LSP requests and FIS messages to prevent these messages from being sent towards the failure. When eventually the failure clears the router enters *FTCR revertive* mode and will start the revertive actions for every LSP that has been FTCR recovered before. When all the LSPs have been restored to their original state, from the point of view of that particular router, the router will enter normal mode again.

Note that Figure 49 illustrates that FTCR reacts immediately to a FIS entering the failure detected mode but in contrast waits for the down-down timer after a FRS before entering the revertive mode. In the next chapter, section 5.4.6 we will illustrate that this property imposes both fast recovery and good stability properties to FTCR.



The router acts as a normal LSR.

The router is notified of a failure (local or remote)

The router recovers the affected LSPs, pending label requests and FIS are rerouted.

The router reroutes LSP requests and FIS messages.

The router is notified that a failure has cleared (local or remote)

The FTCR recovered LSPs are restored to their original state.

The router acts as a normal LSR again.

Figure 49: An FTCR router can be in four different modes of operation: the normal mode, the FTCR recovery mode, the failure detected mode and the FTCR revertive mode.

We will now investigate each of these modes in more detail, with the exception of the normal mode which does not need any further explanation.

4.8.1 The recovery mode

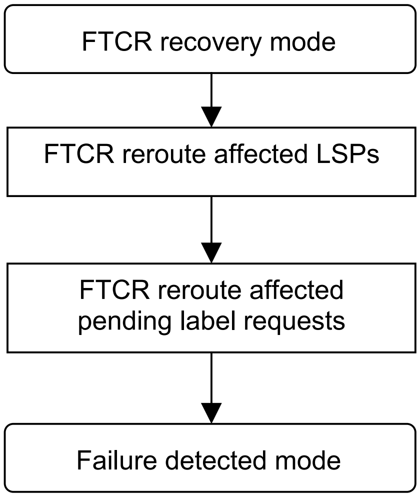

FTCR recovery mode
FTCR reroute affected LSPs
FTCR reroute affected pending label requests
Failure detected mode

Figure 50: In the FTCR recovery mode the LSR recovers the affected LSPs and the affected pending label requests and then enters the failure detected mode.

The FTCR recovery mode defines all the actions that need to be performed when a failure is detected (see Figure 50). When a failure is detected the affected LSP needs to be rerouted but also the pending label requests. In the rest of this section we will explain these operations in more detail. When the recovery actions are completed the router enters the failure detected mode.

4.8.1.a Rerouting the affected LSPs

To recover an LSP the router starts checking if the LSR is the FSL for that LSP (see Figure 51). When the node uses the ingress FSL selection mode and the LSR is not the ingress of the LSP a FIS is sent towards to ingress of the LSP and the next affected LSP is processed. If the LSR operates in nearest non-looping mode or failure-local mode or if the LSR is the ingress of the LSP then the recovery path is calculated (see Figure 52).

The recovery path calculation starts by retrieving the topology information as reported by the link-state database. Depending on the failure presume mode the local copy of the topology is pruned to take into account the current failure (or more correctly the presumed failure). Depending on the type of the affected LSP a new path is calculated on the adjusted link-state database. If the affected LSP is a shortest path LSP then the regular shortest path first algorithm is used. If the affected LSP is an explicitly routed LSP then the calculation depends on the approach taken (see section 4.5.2). Note that the shortest path tree on the adjusted network only needs be calculated once by the Dijkstra algorithm.

After the recovery path has been calculated further operations depend again on the FSL selection mode (we return to Figure 51). In nearest non-looping FSL mode a simple loop detection mechanism is triggered. If the outgoing link of the recovery LSP is the same as the incoming link of the affected LSP a loop is detected and a FIS is sent towards the new FSL. Note that the previous hop of the affected LSP is also the next hop of the locally calculated recovery LSP in this case.

In failure local repair mode or when no loop is detected the original LSP is stored. The original LSP specification needs to be stored so it can be retrieved in revertive mode. The router will then send a label request for the recovery LSP. When the recovery LSP is set up (the router receives a label mapping for the LSP) it will install the recovery LSP as the new working LSP and the traffic on the affected LSP is switched over.

When all the affected LSPs have been recovered the router will continue to reroute the pending label requests.

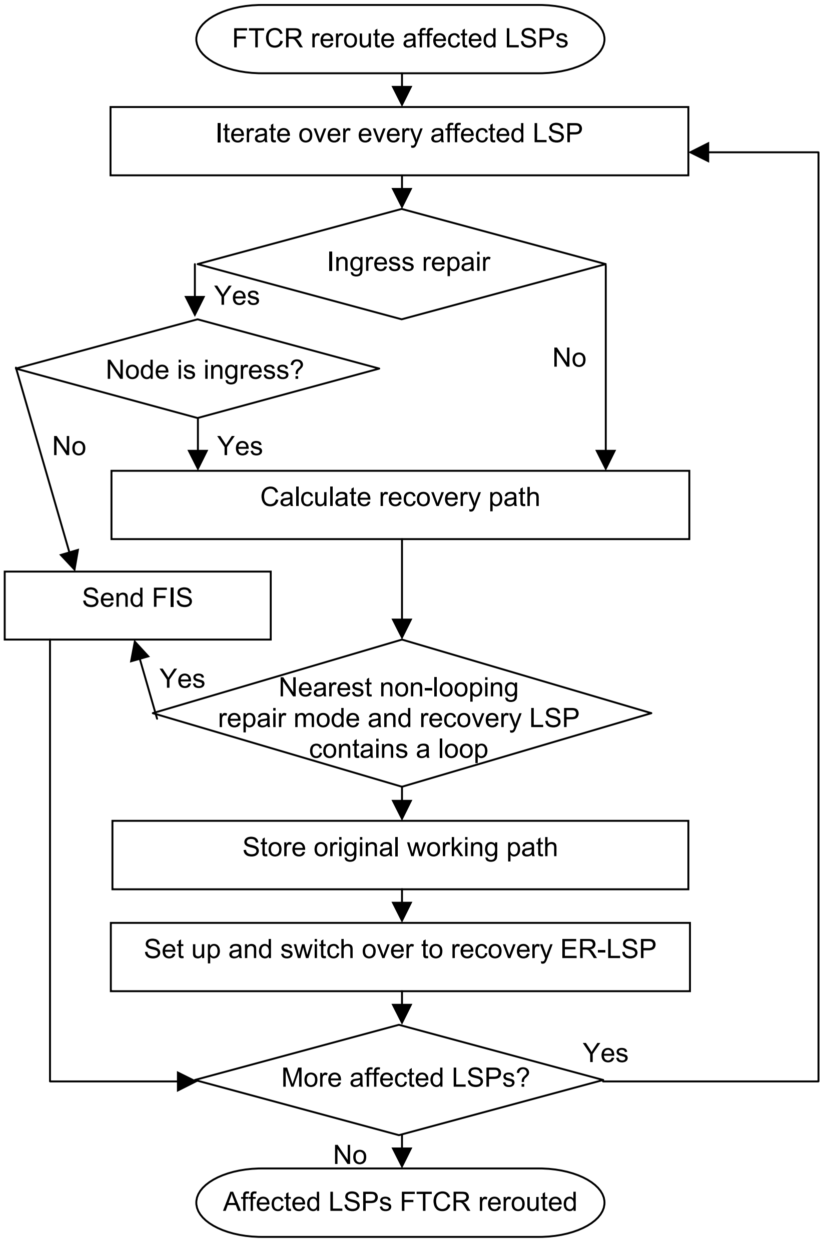

FTCR reroute affected LSPs
Iterate over every affected LSP
Ingress repair
Yes
No
Node is ingress?
No
Yes
Calculate recovery path
Send FIS
Yes
Nearest non-looping repair mode and recovery LSP contains a loop
Store original working path
Set up and switch over to recovery ER-LSP
More affected LSPs?
Yes
No
Affected LSPs FTCR rerouted

Figure 51: The recovery of an LSP depends on the FSL selection mode. Note that the original LSP specification needs to be stored so that it can be used in the reversion operations.

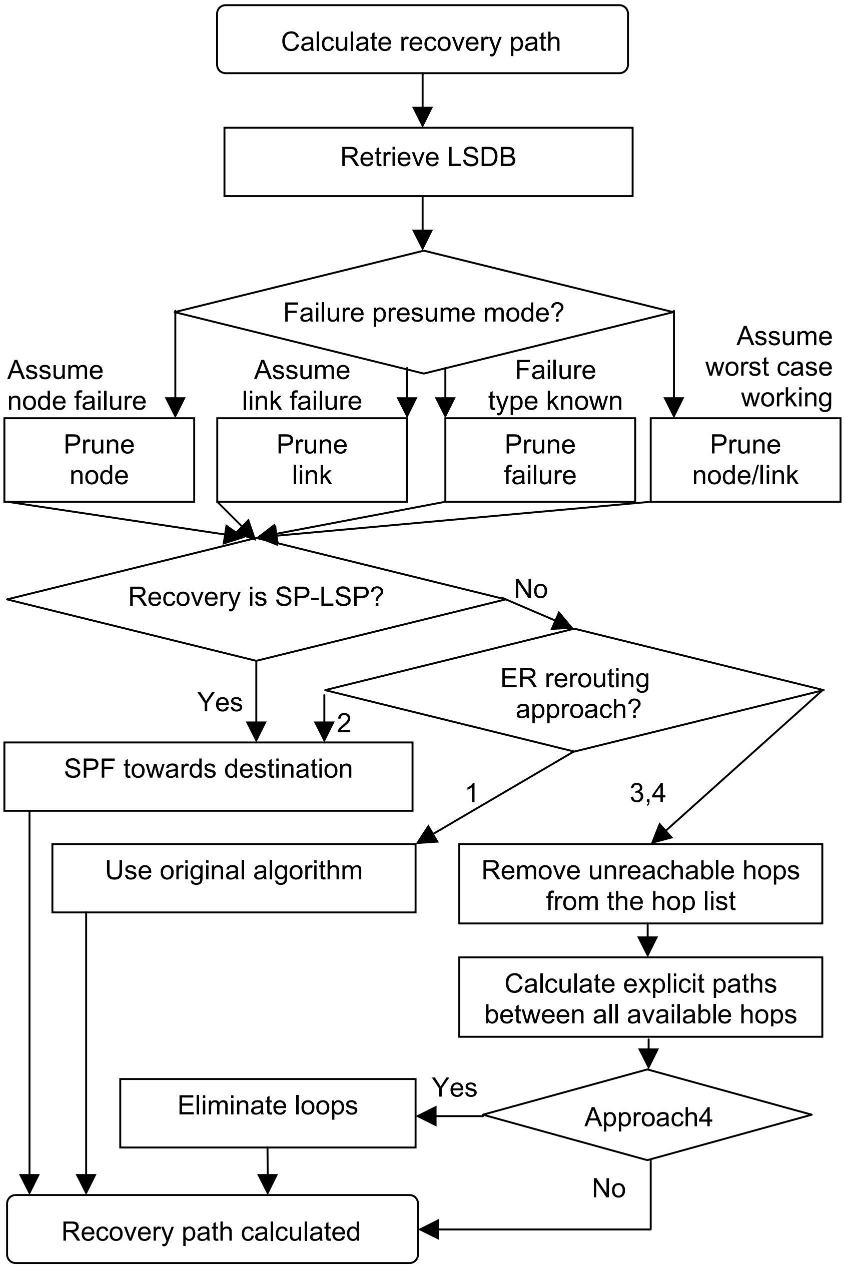


Figure 52: FTCR route calculation depends on the failure presume mode, the type of the LSP and the ER-LSP routing approach for ER-LSPs.

4.8.1.b Rerouting a pending label request

Rerouting a pending label request is very similar to rerouting a label request. The pending label request is first cancelled to inform the downstream LSRs of the fact that the original LSP request is no longer needed. Note that not all downstream routers will be reachable. However those that are not reachable did not receive the label request in the first place so this does not pose a problem. Subsequently the pending request is rerouted just like an incoming label request. Rerouting a label request will be explained in the next section, when we cover the failure detected mode.

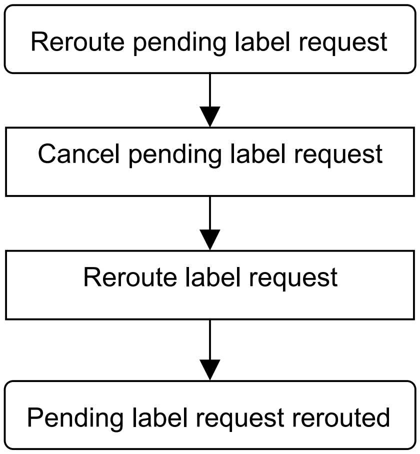

Reroute pending label request
Cancel pending label request
Reroute label request
Pending label request rerouted

Figure 53: Rerouting a pending label request is done by cancelling the pending label request and then rerouting that label request as if it just arrived.

4.8.2 The failure detected mode

The failure detected mode differs from the normal mode in two aspects. The first aspect is that some label request message may need to be rerouted as they are received i.e. a new path has to be found because the path normally taken is routed over the failure (see Figure 54). The second aspect is that FIS message also may need to be rerouted since it may not be possible to forward the message according to the normal path because that path runs over the failure (see Figure 55).

4.8.2.a LSP request rerouting

The first step when receiving a label request during failure detected mode is to verify whether or not the label request is routed over a failure. If this is not the case the request can be forwarded as usual. If the request cannot be forwarded then the action depends on the FSL selection mode. In ingress repair mode a FIS is sent towards the ingress so that the ingress can send a new label request that takes the failure into account. In the two other modes a new path is calculated to forward the LSP request. In nearest non-looping mode a check is made to verify that the new route does not contain a loop. If the path does contain a loop then a FIS is sent upstream (the previous hop of the label request). If the path does not contain a loop or the FSL mode is failure local repair then the LSP request is sent over the newly calculated path.

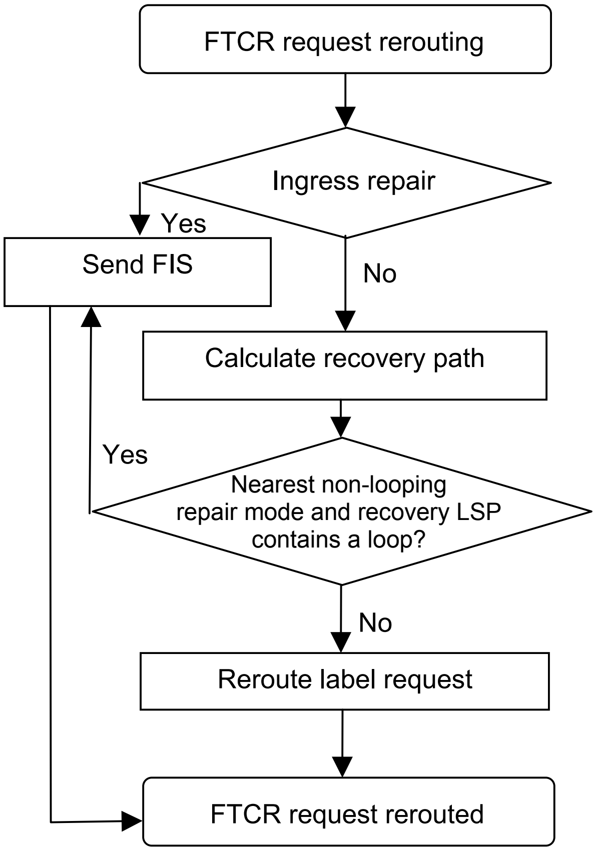


Figure 54: FTCR label request rerouting again depends on the FSL selection mode. If the router is not the FSL for the LSP a FIS is sent upstream towards the FSL. Otherwise the label request is rerouted by calculating a new path on the adjusted link-state database.

4.8.2.b FIS forwarding and rerouting

In failure detected mode the FSL must monitor the incoming FIS messages. Note that even though the FIS rerouting is explained in this section this does not mean that the transmission of the FIS messages that are generated during the FTCR recovery mode are postponed till the failure detected mode.

Forwarding a FIS towards the FSL is normally done via the reverse path of the LSP. This can obviously fail when multiple failures occur in ingress or non-looping repair mode. In these modes it is possible that a router receives a FIS but is unable to forward it over the normal path because of a local failure.

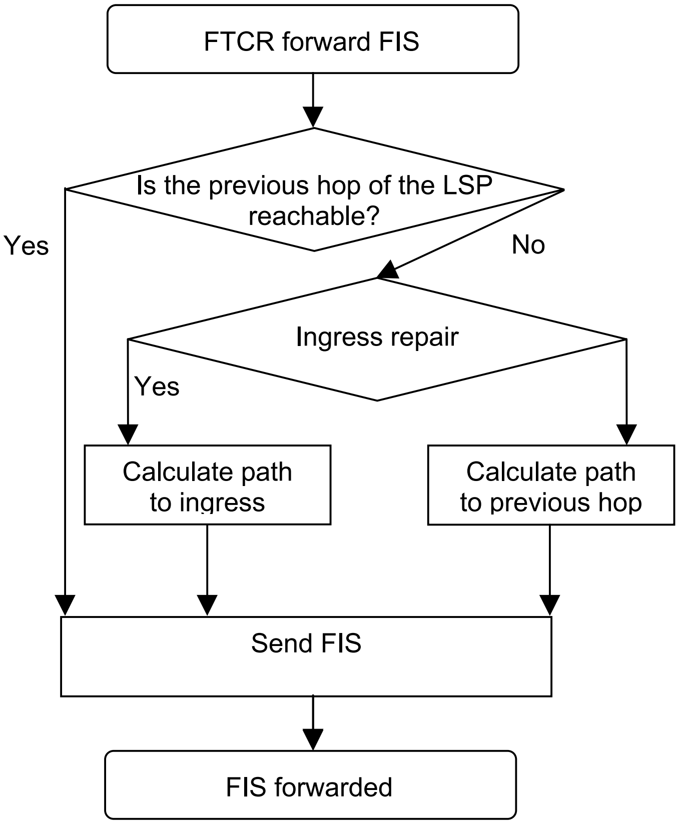

FTCR forward FIS
Is the previous hop of the LSP reachable?
Yes
No
Ingress repair
Yes
Calculate path to ingress
Calculate path to previous hop
Send FIS
FIS forwarded

Figure 55: Forwarding a FIS depends on whether or not the next hop is reachable. If the next hop is reachable the FIS is sent like a normal packet if this is not the case a new route is calculated and the FIS is sent with an explicit path.

When the FSL receives a FIS it will check if the normal next hop of the FIS is reachable (see Figure 55). If the next hop is reachable then the FIS is forwarded as usual. If the next hop is not reachable then a new route to the destination of the FIS is calculated. The destination of the FIS is the ingress of the LSP in ingress repair mode and is the previous hop in nearest non-looping mode. Actual calculation is again based on the adjusted link-state database using the shortest path first algorithm. After the calculation the FIS is sent with an explicit route towards the destination. Note that not all FIS mechanisms support explicit routes. If this is not case then supporting multiple failures with that FIS mechanism is not possible in ingress and nearest non-looping repair.

Note that when the destination of the FIS receives the FIS it will enter the FTCR failure detected mode as explained in Figure 49.

4.8.3 The revertive mode

An LSR enters the FTCR revertive mode after the wait-to-restore timer expires.

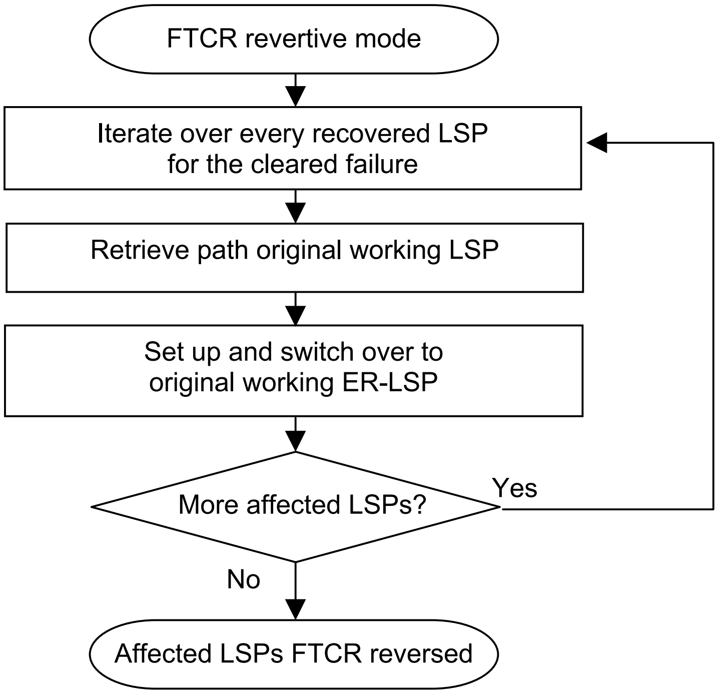

FTCR revertive mode
Iterate over every recovered LSP for the cleared failure
Retrieve path original working LSP
Set up and switch over to original working ER-LSP
More affected LSPs?
Yes
No
Affected LSPs FTCR reversed

Figure 56: FTCR reverses an LSP by retrieving the original LSP specification and setting up the LSP. A wait-to-restore delay between the moment the FRS is received and the start of the LSP set up is used to make sure the routing tables have converged.

The wait-to-restore timer is started when a FRS is received (see Figure 49). In the revertive mode the FSL should revert all the LSPs that have been recovered back to their original specification. The procedures of the reversion operations are detailed in this section.

Again the operations depend on the FSL selection mode, in ingress repair the FRS is forwarded towards the ingress of the LSP. Otherwise every affected LSP is reverted. Restoring an LSP starts by fetching the original LSP specification. Before the revertive mode is entered and the original LSP can be set up it is important to wait for the wait-to-restore timer. The wait-to-restore timer guarantees that the routing tables have converged and that partially specified ER-LSPs and SP-LSPs can be set up. When the wait-to-restore timer expires a request for the original LSP is sent and when the LSP is set up, the traffic is switched back from the recovery LSP to the working LSP.

When all the LSPs have been restored to their normal working path the router returns to normal mode.

4.9 Prerequisites for FTCR

In this section we will look at the requirements of FTCR. Every failure convergence scheme has certain requirements. FTCR has a few typical requirements. The requirements of FTCR depend on the FSL selection mode and whether or not ER-LSP and multiple failures need to be supported. In this section we will sum up the different requirements of FTCR and describe the reason for each of these requirements. When a certain requirement is not met that does not necessarily mean that FTCR does not operate but it might mean that a certain feature or mode of FTCR is not available. In section 4.9.5 we will give a matrix of the requirements of the different FTCR modes. But we start this section by describing the most important requirements of FTCR in more detail.

4.9.1 MPLS and explicitly routed LSP support

FTCR is a rerouting scheme specifically for MPLS, it cannot operate without the possibility to set up recovery LSPs. It is theoretically possible to invent a similar approach that does not use recovery LSPs. In this scheme the FSL intercepts all IP packets that would be forwarded over the failing link and it inserts a source route in every IP header calculated on the adjusted link-state database. This convergence scheme would introduce a severe performance and overhead penalty to the forwarding path of the rerouted packets. Another problem is that IP source routing is usually disabled due to security issues.

As stated before FTCR requires LSP support, moreover it requires fully specified ER-LSP support. Not all MPLS signalling protocols support ER-LSP set up, most notably LDP. The other major signalling protocols CR-LDP and RSVP-TE do support ER-LSPs. To compare FTCR with the other MPLS convergence schemes, MPLS rerouting only requires SP-LSPs support and protection switching requires ER-LSP support.

4.9.2 Link-state database

Another important requirement of FTCR is the ability to access a link-state database. Link-state routing protocols like OSPF and IS-IS maintain a link-state database but distance vector protocols like RIP and IGRP do not. That implies that distance vector routing protocols cannot be used to support FTCR. FTCR does not only require that a link-state database is maintained on the MPLS domain but also that an interface exists to access that link-state database.

The requirement to have access to a link-state database must be relaxed though. FTCR does not require that the link-state database is up-to-date all the time. When a failure occurs it will take some time before the link-state databases in the network have converged. Even in the node that detects the failure it can take some time before the failure is incorporated in the link-state database. FTCR does not require that recent failures already have been taken into account when it requests the link-state database. If the link-state already has been adjusted to take the failure into account then FTCR can run the shortest path algorithm directly on this link-state database. If this is not the case it will make a copy of the link-state database, remove the failure from the database and run the shortest path algorithm on the copy. It is important to note that FTCR does not alter the link-state database used by the routing protocol directly because this will leave the link-state database in an inconsistent state for the routing protocol.

FTCR uses the link-state database only for topology information. It does not use the route calculation component of the link-state routing protocol. This means that it can support non-shortest path LSPs. This also means that the IP routing tables are not required for the recovery actions. Some network operators prefer that any connectivity in the network is explicitly configured through LSPs and that no routing tables are available. In this scenario all the LSPs are fully specified. Since FTCR only requires the topology information in the link-state database and not the availability of routing tables this model is supported with FTCR. Note that the routing tables still contain the locally available information i.e. the local interfaces.

Another observation is that FTCR’s topology requirements can be fulfilled by other means than a link-state database. A mechanism specific to FTCR could be developed for the sole purpose of learning and spreading topology information inside the network (see [150, 151, 152, 153] for references about topology gathering techniques). Since link-state routing protocols are quite ubiquitous and fulfil the requirements completely this path has not been investigated.

Although the requirement of a link-state database is somewhat relaxed this requirement might be prohibitive in some set-ups. The requirement of FTCR to have access to a link-state database is quite typical for FTCR but on the other hand some router platforms only support certain features in combination with certain routing protocols too.

4.9.3 Extensions to MPLS signalling

FTCR does not rely on MPLS signalling extensions. FTCR can however benefit from signalling extensions to indicate the type of recovery mechanism to be used on a per LSP or even per sub-LSP level. If these signalling extensions are absent then the recovery mechanism can be configured on a platform wide or per interface basis. For example a node can be configured to recover all the LSPs with FTCR, or it can be configured to recover all the LSPs over a specific interface with FTCR or to not use FTCR at all. If more flexibility is needed then the recovery mechanism needs to be signalled when the LSP is established.

4.9.4 Failure detection and extended failure indication signal

Every convergence scheme requires that failures are acted upon one way or the other. So a failure detection mechanism implicit or explicit of some sort is always needed. Some schemes like OSPF define their own mechanism. FTCR does not define a failure detection and failure notification scheme itself. For example FTCR can use the failure detection scheme implemented by OSPF for failure detection and it can use the MPLS signalling protocol messages to send FIS messages. However in order to support multiple failures in ingress or nearest non-looping mode, one has to require that the route of the FIS messages can be explicitly specified and not every FIS mechanism supports that. If the FIS mechanism does not support explicit routing then either failure local repair should be used or multiple failures cannot be supported in these modes.

4.9.5 FTCR requirements matrix

In this section we will look at all the requirements of FTCR and investigate in which modes these are needed. We investigate the requirements where we only support SP-LSP rerouting, where we support ER-LSP rerouting and where we support multiple failure rerouting in each of the three FSL selection modes: Failure Local Repair (F), Nearest non-looping Repair (N) and Ingress Repair (I) (see Table 10).

As we described in sections 4.9.1 and 4.9.2 FTCR requires ER-LSP and link-state database (LSDB) support. FIS notification is only needed when the FSL is not directly affected by the failure. This means that failure notification is required in nearest non-looping repair and ingress repair. FIS rerouting is needed to support multiple failures in nearest non-looping and ingress repair because the path from the fault-detecting node to the FSL may need to be avoided because of another failure. Request rerouting is needed to support multiple failures in failure local and nearest non-looping repair modes. Request rerouting is not needed in ingress repair mode because the ingress of an LSP will never intercept a label request of an upstream router since it is the most upstream router of the LSP (topological constraint). The next observation is that loop detection is only needed to support nearest non-looping repair. ER-LSP rerouting is of course needed in ER-LSP reroute mode but also to be able to reroute the explicitly routed recovery LSP when multiple failures occur. The last two requirements in Table 10 i.e. pending request rerouting and reversion operations need a more in depth investigation.

Table 10: The matrix summarises the relationship between the FTCR functionality in the different FSL selection modes (top row) and the requirements (first column).

	SP-LSP rerouting			ER-LSP rerouting			Multiple Failure rerouting		
	F	N	I	F	N	I	F	N	I
ER-LSP	X	X	X	X	X	X	X	X	X
LSDB	X	X	X	X	X	X	X	X	X
FIS notification		X	X		X	X		X	X
FIS rerouting								X	X
ER-LSP reroute				X	X	X	X	X	X
Request reroute							X	X	
Loop detection		X			X			X	
Pending RR	O	O	O	O	O	O	X	X	X
Reversion	X	X	X	O	O	O	O	O	O

O: Optional
X: Mandatory

Pending request rerouting is optional when repairing single failures. Repairing a pending label request means that a label request has been sent over a path that contains a failure. This is possible because the failure was not detected at the time the label request was sent. Since the path of the LSP request contains a failure the LSP will not be set up and the request remains pending. Note that this is possible with all failure convergence schemes. In FTCR it is however possible to correct the pending label request and reroute the request over a path that circumvents the failure. It is not mandatory to reroute label request to support single failures modes but it will speed up the convergence if it is supported.

Rerouting pending label requests in the face of multiple failures is different from pending request routing as described in the previous paragraph. In case of a single failure the label request is the result of an ordinary LSP set up but when multiple failures occur the label request can be the result of a recovery action. Such a pending label request can happen when an FSL sends a label request for a recovery LSP over a path where a second failure has occurred but has not been detected yet. If the label request from the upstream is not rerouted by the downstream FSL when the failure is detected then the convergence time will be in the same order as normal MPLS rerouting and thus the benefits of FTCR are lost. From that reasoning pending request rerouting is mandatory in multiple failure scenarios to support fast convergence.

So far recovering single failures on shortest path LSPs had the least requirements. However it does have a specific requirement that is only mandatory in this mode. Reversion is mandatory to support if only SP-LSP can be rerouted. When FTCR can only reroute shortest path LSPs it cannot reroute the recovery ER-LSPs. So the recovery LSPs need to reversed back to the shortest paths as soon as the shortest path is available again in order to recover other potential failures in the future. Note that these failures do not count as multiple failures according to our definition in section 3.3.1.d.

If reversion is not supported that would mean that the recovery ER-LSPs will never be reversed back to the shortest path. That in turn would lead to the situation where a certain failure can only be recovered from once. When all possible failures have occurred once, the network has no resilience anymore.

4.10 Convergence and reversion cycle

As we did with the other convergence mechanisms in the previous chapter we will also look at the convergence and reversion cycles for FTCR.

4.10.1 Convergence cycle

Like protection switching, FTCR does not require a specific failure detection mechanism. The notification time depends on the FSL selection mode. In failure local repair the notification time will be minimal since the failure is repaired by the node that detects the failure. In ingress mode the notification depends on the distance between the failure and the ingress. In nearest non-looping FSL selection mode the failure notification time is increased by the subsequent loop detection algorithm run on all the nodes between the failure detection node and the FSL.

Similar to protection switching no hold-down time is used when a reliable failure detection mechanism is used. (See section 3.3.4.f for a discussion about unreliable failure detection and hold-down timers in protection switching.)

FTCR recovery operation calculates the recovery LSPs based on the adjusted link-state database. The time required depends on the network size and the efficiency of the implementation. When the recovery LSPs are calculated they will be set up. Note that Dijkstra calculates a shortest path tree, so the Dijkstra shortest path algorithm only needs to be calculated once, not for every recovery LSP.

Table 11: illustrates the convergence cycle of FTCR. The notification time depends on the FSL selection mode.

Fault detection	Notification time	Hold-down time	Recovery operation
*	η	0s	Calc. recovery LSP (θ) LSP setup (ζ)

4.10.2 Reversion cycle

The reversion cycle of FTCR differs from the convergence cycle mainly because FTCR needs a wait-to-restore timer in order to make sure that the IP routing tables have converged before the preferred working path is set up.

Another important difference between convergence and reversion in FTCR is that during reversion no LSPs need to be calculated since the working LSPs were stored before they were rerouted so they can be set up according to their stored specification.

Table 12: illustrates the reversion cycle of FTCR. The notification time again depends on the FSL selection mode.

Clearing detection	Notification time	Wait-to-restore	Reversion operation
*	η	IP convergence	LSP setup (ζ)

4.11 Conclusion

This concludes the chapter about FTCR. As we have seen, FTCR is a convergence technique for MPLS that speeds up convergence by setting up the recovery LSPs even before the IP routing tables have converged.

It is possible to set up a recovery LSP even when the routing tables are not valid if every hop of the recovery LSP is specified. This recovery LSP can be calculated on the link-state database taking the failure information into account.

FTCR also supports rerouting ER-LSPs and multiple failures. Multiple failures are supported by having one recovery operation per failure per affected LSP. Since there is always one LSR responsible for one downstream failure. That LSR will either do the recovery operation itself or notify the FSL that a recovery action needs to be performed. Supporting multiple failures not only requires rerouting of LSPs but also of LSP requests. But even with LSP and LSP request rerouting FTCR may not be able to offer faster convergence than the IP routing protocol. This happens when an LSP request is rerouted over a fault that has not been detected. The result is that the LSP request remains pending. However as soon as the failure is detected the LSP request can be rerouted over a valid path. This process is called pending label request rerouting and it is the final step required to offer fast FTCR convergence during multiple failure events.

FTCR is a convergence scheme specifically developed for MPLS that optimises MPLS rerouting instead of relying completely on IP convergence. This makes FTCR faster to converge after a failure than both OSPF and MPLS rerouting. However during reversion FTCR synchronises back with OSPF by depending on routing tables to set the working LSPs back up. In the next chapter we will see that this has important consequences for FTCR's stability.

In the next chapter we will compare FTCR with the other convergence schemes that we covered so far. We will look at the convergence times, the stability and scalability and backup requirements. In the chapter after that we will look at some of the recent proposed improvements to the existing convergence schemes and how FTCR relates to these advancements.

Chapter 5

Evaluating convergence schemes

5.1 Introduction

In this chapter we will evaluate the convergence schemes OSPF, MPLS rerouting, FTCR and protection switching in different ways.

In the first two sections we will describe the experimentation and measurement platform (the testbed) that we use to determine the convergence times of the different convergence techniques. We use the testbed to experimentally measure the convergence times and to increase our insight into the convergence mechanisms. The first section starts with an introduction of the testbed and the measurement techniques. We will pay special attention to the error analysis of the measurements. The second section starts with the description of the individual tests, the experimentation parameters, the results and the conclusions that can be drawn from them.

In the third and fourth section of this chapter we will investigate two less tangible properties of convergence techniques i.e. stability and scalability.

In the final section we will investigate another important property of convergence techniques: the backup requirements. The backup requirement of a convergence technique is the amount of bandwidth that must be available in the network to be able to recover a certain capacitated demand in the network during failure conditions. As we will see the amount of backup capacity depends on the convergence technique used to recover the working demand.

As always we will summarise the most important findings of this chapter in the conclusions section.

The results of this chapter were published in several publications as indicated in section 4.1. The design of the software platform that is used for the experiments was published in [7].

5.2 The Test network

In this section we investigate the time of convergence for the different convergence schemes we have described so far: OSPF, soft-state MPLS rerouting, triggered MPLS rerouting, protection switching and FTCR.

We will take two approaches towards determining the time of convergence of the convergence mechanisms. We will compare the theoretical analysis of Chapter 3 and Chapter 4 with the convergence time determined inside a test network.

Using two independent approaches to determine the convergence time has several benefits over determining the convergence time with only one technique. For example the theoretical analysis needs to be conform to the measurements and the other way around. The convergence times determined experimentally also allow us to get an idea of the time values of parameters which are not fixed (the parameters denoted with Greek letters). However these values should be interpreted within the limitations of the experiments which are conducted inside a test network.

5.2.1 Experimentation platform

In this section we will describe how the convergence times of the different convergence schemes are measured. We will describe the experimentation platform used to conduct the experiments and how the measurement platform is used to measure the convergence times.

5.2.1.a Router platform

When conducting experiments in a test network there are typically two choices for the router equipment to use. The first alternative is to use commonly available commercial router platforms. The main benefit of this approach is that these routers create a good reference because they are well known and have proved their interoperability and track record. The main drawbacks are that they are quite expensive and that it is very hard to extend their functionality. Since we want to evaluate FTCR this option was not chosen.

An alternative to using commercial routers is to use routers based on the PC architecture: PC based routers. PC based routers are flexible and cheap in comparison to commercial routers. The main drawback is that PC routers typically do not come as an integrated product. This means that the software for the required router functionality needs to be installed and configured. Some functionality is also not readily available for the PC router platform and may need to be developed. Of course the fact that functionality can be added very easily is the main strength of the PC router.

In [154] we discuss a hybrid between the two alternatives where the control component of the router is based on the PC architecture but where the forwarding is done by a very fast lookup engine called the Internet Fast Translator (IFT). Previously we also published a different hybrid design based on a PC control architecture and a ATM switch which is capable of unicast and multicast MPLS [20].

In our experiments we have chosen for a pure PC router platform. The platform used for the experiments consists of a collection of 10 Linux PC based routers arranged according to the topology depicted in Figure 57.

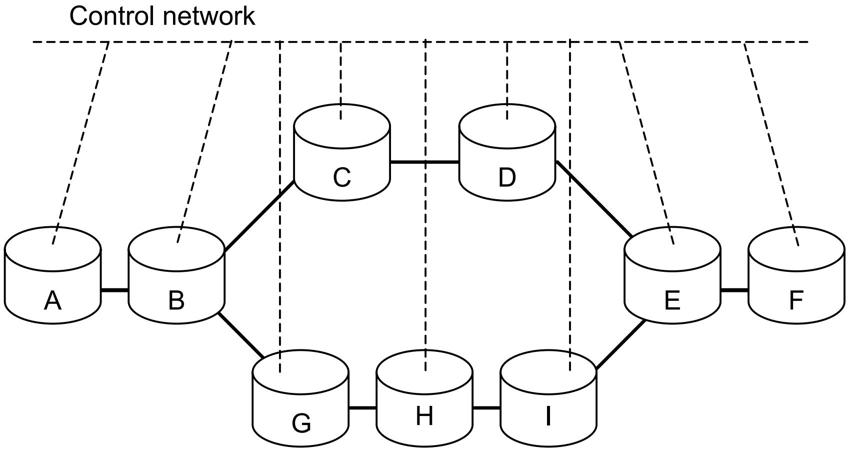


Figure 57: The topology of the PC router network used for the experiments. The routers are connected with each other via the test network but they are also connected to a control network.

The test network connects the routers point-to-point to each other. The routers are also connected to a control network. The control network is used to install and configure the routers but also to orchestrate the tests. Special care has to be taken that test traffic does not use the control network.

Unlike commercial routers, which are the combination of dedicated routing hardware and software, a PC router is really just a normal PC usually equipped with additional network interface cards (typically four). This means that all the software needs to be selected and installed. In the next section we will describe the software platform based on the Linux operating system.

5.2.1.b Software platform

In this section we will give a fairly high level overview of the software components used in the experiments. We refer to Appendix A for the details about each component. In this section we will look at four components. We start by introducing the operation system, afterwards the routing daemon and the MPLS signalling daemon and finally the control architecture.

Operating System

We selected GNU/Linux operating system [48, 49] usually simply referred to as Linux. Linux is a freely available UNIX-type operating system. Originally developed only for the Intel PC architecture but currently ported to both small embedded systems and high-end mainframe systems. Linux supports a wide range of Network Interface Cards (NIC), has a high performance network stack and has advanced queuing and scheduling capabilities.

Linux is distributed under the conditions of the GNU (GNU is Not Unix) [50] General Public License (GPL [51]) which means that the full source code is available to its users. This allows anyone to find out how the system works and to trace and to remove any bugs. But more importantly this gives the user the ability to add functionality to the Linux kernel. A prime example in the light of this work is the support for MPLS as provided by [155]. In his work James R. Leu implemented an MPLS stack for Linux loosely based on the IPv4 stack.

Note that Linux is not the only free and open Unix implementation available. The most notable alternatives are the operating systems of the BSD family (OpenBSD [156], FreeBSD [157] and NetBSD [158]). Linux was chosen over any of the BSD alternatives because of the author’s and the research group’s familiarity with Linux and the availability of a MPLS implementation for Linux.

We will now investigate some of the key properties of Linux. A Linux system typically consists of the Linux *kernel* and a number of *user space* programs and *daemons*. The kernel is the abstraction layer used to shield the system resources from the user space programs. User space is a term specifically used to denote that a certain program is not part of the Linux kernel but rather runs under the provisions of the kernel.

A *daemon* is a program that does certain jobs in the background. Daemons typically do not interact with the user directly nor do they output directly to the screen but rather add log entries to their own log files or to the system-wide log files. Examples of daemons are a web server, a firewall, a network proxy, a routing protocol daemon and a MPLS signalling daemon. It is important to note that a daemon does not run in kernel space although they sometimes are more closely tied (and dependent on) the kernel.

Linux is a true pre-emptive multi-tasking operation system kernel. All processes run entirely independently of each other in a separate address space. Pre-emptive multi-tasking means that the kernel can take away the processor from a certain user space process. Pre-emptive multi-tasking is more advanced than co-operative multi-tasking where a process has to explicitly yield the processor. Most modern operation systems are pre-emptive multi-tasking. In Linux only the user space processes are pre-emptive, the kernel itself is not pre-emptive. This means that the kernel will complete a certain operation unless it yields the processor itself. Certain kernel operations can be relatively long and during these operations the kernel does not perform any other work which can lead quite high response times. For example [159] cites response times to an audio application of more than 20ms during disk writes. Certain kernel operations take even more time, [160] cites 50ms for setting device parameters up to 500ms for scrolling on a frame buffer console. Linux is not a hard real-time operating system which means that it cannot guarantee real-time bounds on scheduling delays. Although real-time variants of Linux exist, for example RTLinux [161], these should actually be regarded as distinct operating system kernels. As such the MPLS functionality is not easily ported over to them. As a consequence we are not able to use a real-time operating system. We will see that this has its consequences later on.

The most important modification to the Linux kernel used in the experiments is the addition of the MPLS stack.

Routing daemon

We selected the GNU Zebra routing platform [46]. Zebra is a suite of routing protocols that manages TCP/IP based routing protocols. It supports the BGP-4 protocol as well as RIPv1, RIPv2 and OSPFv2. Zebra software offers true modularity in the sense that it has a process for each routing protocol. The availability of the source code has proved to be very useful, not necessarily to modify the source but rather to complement the information in the relevant RFCs on the routing protocols. We verified the Zebra OSPF implementation with respect to the RFC specification in [153]. We selected Zebra as the routing daemon because it is the most important and most active open source routing platform.

The only modification done to Zebra is that the precision of the time stamping in the log files has been improved from second to millisecond granularity.

MPLS signalling daemon

Unlike the operating system and routing protocol software components, where high quality implementations that supported all the requirements were available, there was no such MPLS signalling daemon. There was an LDP daemon developed in house and the LDP daemon from the MPLS-Linux project [155] but both where LDP-only daemons that did not support explicitly routed LSPs.

At that moment there was also a port of the ISI Intserv RSVP implementation [162] to Linux by Alexey Kuznetsov [163]. But that RSVP daemon does not support MPLS. There was also the Nistswitch version 2.0 daemon for FreeBSD by USC [164] supporting MPLS and explicitly routed LSP but only under FreeBSD.

So there were several daemons but neither of them offered ER-LSP support under Linux. It was decided to opt for a RSVP-TE daemon rather than an CR-LDP daemon because of the increasing popularity of RSVP-TE over CR-LDP and the fact that the author was more familiar with CR-LDP so working towards a RSVP-TE solution would be a good exercise to get more familiar with RSVP-TE. Note that implementing a RSVP-TE daemon from scratch was not considered an option so the existing daemons had to be reused.

It was decided to combine the IntServ RSVP daemon for Linux and the MPLS daemon for FreeBSD so that the MPLS and ER-LSP functionality of the Nistswitch daemon would be available under Linux. This was possible since they are both based on the ISI RSVP implementation. The MPLS specific code in the Nistswitch daemon was obviously written for FreeBSD. This does not create a problem for the bulk of the code since both Linux and FreeBSD are Unix-like operating systems. Only the part where the daemon interacts with the kernel to install the MPLS forwarding state needed to be rewritten because it is kernel specific. Fortunately MPLS support for the Linux kernel code was available through the MPLS-Linux project [155]. The kernel specific code in the daemon needed to be rewritten to use the MPLS-Linux MPLS code under Linux. After the integration of the two daemons and the kernel code support for DiffServ over MPLS was also added.

The resulting daemon in combination with the necessary kernel and user space patches supports the setup of shortest path and explicitly routed E-LSPs and L-LSPs (see section 2.5.2). The daemon in return was publicly released through the "RSVP-TE daemon for DiffServ over MPLS under Linux" [133] project. The author has been leading the open-source effort since summer 2001 and a considerable number of participants from vendors, research institutes and universities all over the world have joined the project. Appendix B, first published in [7], describes the architecture of the daemon and the benefits and drawbacks of open-sourcing a project like this in more detail.

Control architecture

As illustrated in Figure 57, each node of the test network is not only connected to its routing peers but also to a control network. The control network's purpose is to have an out-of-band administration and management channel to operate and administrate the test network. For example, a typical scenario in the experiments disables an interface of the test network and then measures the time it takes the test network to route around the failure (the disabled interface). In order to conduct such an experiment a control channel must exist to disable the interface of the *test network* and at the same time remain connected to the *control network*.

In order to conduct a single test scenario several operations need to be performed on the routers in the test network. Most of these operations are typically performed interactively from the Command Line Interface (CLI). A lot of these operations need to be performed on every router in the test network. For example before conducting an MPLS experiment the MPLS signalling daemons need to be started on every router. Given that we want to repeat the experiments numerous times it is clear that manually configuring and running each experiment is not realistic and very prone to error. To address these issues an experiment automation framework has been developed based on *Expect* [165]. Appendix C describes this framework in more detail. Note that the experiment automation framework uses the control network to connect to the routers in the test network.

In the next section we will describe the measurement platform that is used to measure the convergence times in the test network.

5.2.2 Measurement platform

In this section we will describe a few techniques that are typically used to conduct experiments inside a network. We will start by explaining how *ping* can be used to get a crude estimation of the convergence time. After that we will look at other alternatives which are more suitable in the context of precise convergence time measurements in MPLS networks. Subsequently we will compare the alternative and adopt a single measurement scheme. In the final section we will investigate the precision of the implementation of this scheme.

5.2.2.a Using Ping

Basic connectivity tests in IP are often conducted with the ‘ping’ tool [166]. Ping sends ICMP REQUEST packets to a certain destination. The destination replies by sending an ICMP RESPONSE back to the sender [11, 167]. The sender on receipt of the response knows the two-way delay, the number of hops between source and destination (from the TTL decrement) and a rough estimate on the packet loss. The ping tool can be used to get a crude estimation of the convergence time in IP by pinging a destination in the network and then introducing a fault. During the time of convergence the ICMP REQUEST messages will not reach the destination and/or the ICMP RESPONSE message will not be able to reach the sender. The maximum time between two consecutive successful ICMP request/response pairs (pings) or the number of lost pings are an indication for the time of convergence. Using ping as a measurement technique has a number of drawbacks especially within the context of MPLS.

A first problem with using ping is the low rate of the ICMP packets, only one packet per second. The rate of the ping packets determines the absolute error on the measurement and an absolute error of one second is definitely not an accurate measurement in this context. Increasing the rate easily solves this problem and some of the more advanced ping implementations support this.

A more fundamental problem is that ICMP based measurements are intrinsically bi-directional and LSPs are unidirectional. This means that you cannot measure the time to recover a single LSP but rather the time required to recover a single LSP and the reverse IP path or two LSPs bundled in together in the two opposite directions. The fact that routes in IP networks can be asymmetric introduces more problems. So we can conclude that unidirectional or one-way measurements are necessary.

5.2.2.b One-way measurements

A general problem with time measurements over a different number of nodes is time synchronisation [168]. For instance consider that we want to measure the time it takes a node to perform a protection switch over an LSP measured from the time the failure occurs (see Figure 58). The factors that contribute to this time are (1) the time to detect the failure, (2) the time it takes for the FIS to reach the PSL and (3) the time to determine and perform the correct protection switch action.

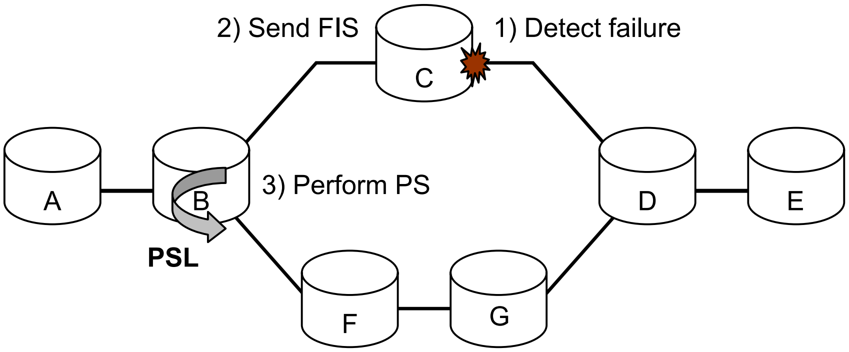


Figure 58: The different phases of convergence with protection switching. Measuring the time between the failure event and the protection switch operation at the FSL requires clock synchronisation at node C and node B.

Consider that we simulate a link failure on link CD by disabling the interface on node C. When we disable the interface on node C at instance t_1 and measure the time t_2 when the protection switch is performed at the PSL (node B) then the time to converge is given by $t_2 – t_1$. Since we measure time instances on two distinct machines the clocks on the machines need to be synchronised.

There are two main solutions to synchronise clocks on routers. One solution is to use a Global Positioning System (GPS) which allows a router to retrieve precise location and time information from a set of satellites. The timing information offers a 340ns precision 95% of the time for the Standard Positioning Service (SPS) [169]. The main drawback of the use of GPS is that it requires extra hardware and that sometimes the location of the routers can prohibit its use.

The other solution is to use the Network Time Protocol (NTP) [170]. NTP is network protocol used to synchronise clients or servers to other servers or reference time sources. The drawback of NTP is that there’s an error introduced by the transfer delay of the NTP packets. Another drawback is that the individual clocks still drift relative to each other so regular synchronisation is necessary. Still NTP delivers millisecond precision in LAN segments. Despite the fact that time synchronisation can be achieved it still is desirable to not rely on it if possible.

We now return to the example in Figure 58. There are additional drawbacks to measuring the time between the network impairment and the resulting convergence action (i.e. the link failure and the protection switch operation). The first drawback is that it is difficult to verify that the convergence action is correct. Verifying the correctness requires that the new outgoing label and interface after the protection switch are the outgoing label and interface of the recovery LSP and that the recovery LSP is properly set up. When another convergence scheme is used, for example IP routing, then the verification of the convergence will be different which is obviously a drawback. This leads to the third drawback of this measurement scheme. Not only does the verification of correct convergence differ for each convergence scheme but also the manner in which the convergence time is measured. For example the analogue for measuring the convergence time of IP routing requires monitoring the routing table in node B (the PSL in Figure 58). When FTCR is used in ingress FSL selection mode then this scheme requires that we measure when the recovery LSP is set up and switched over to.

Of course a measurement scheme that can measure the convergence time and the correctness of every convergence scheme the same way is preferable over a scheme that must be adapted ad hoc for each recovery scheme.

To conclude, as we have seen measuring the start (the network impairment) and the final convergence operation (e.g. protection switch) has numerous disadvantages: it requires time synchronisation, it is difficult to verify that convergence has occurred and it is not generic since it needs adaptation for every convergence scheme.

In the next section we describe a measurement scheme that does not require time synchronisation and that can be used to measure the different convergence times for all the schemes under investigation without modification.

5.2.2.c One-way measurements without time synchronisation

We discussed two measurement techniques: using ping and measuring the start and the end of the convergence cycle. The approach adopted here can be described as a high-speed unidirectional version of ping.

The measurement approach relies on frequently sending sequence numbered packets from a given source to a destination. After a random delay a network impairment is introduced. During the time of convergence the packets sent from the source towards the destination will be lost because the network has not converged yet. The convergence time can then be determined by measuring the number of packets missed at the receiver side. The number of missed packets is determined by looking at the gap in sequence numbers. The convergence time can then in turn be determined by multiplying that number by the rate at which the packets were sent.

$$\text{convergence time} = \text{loss count} * (1 / \text{rate}) \qquad (2)$$

This scheme is unidirectional because packets are only sent in one direction. Moreover the scheme does not rely on time synchronisation because determining the missed packets does not require any timing at all. The correctness of the measured convergence time does depend on the rate at which the packets are sent at the sender and received at the destination, which should be constant over the experiment. The scheme is applicable to every convergence scheme and it can measure the time of convergence between a given source and destination without modification.

It is important to note that this scheme measures the time of the recovery cycle as defined in section 3.1.1. In that section we stated that the recovery cycle ends when the last recovery action is finished. However that does not mean that all the negative effects of the network impairment have been resolved after the last recovery action. This is illustrated in the next paragraph.

When a failure occurs, packets that already have traversed the failing link or node will continue to flow to the destination (Figure 59a). After the last packet has been received at the destination no more packets will be received until the network has converged and the packets flow over the new path to the destination (see Figure 59b).

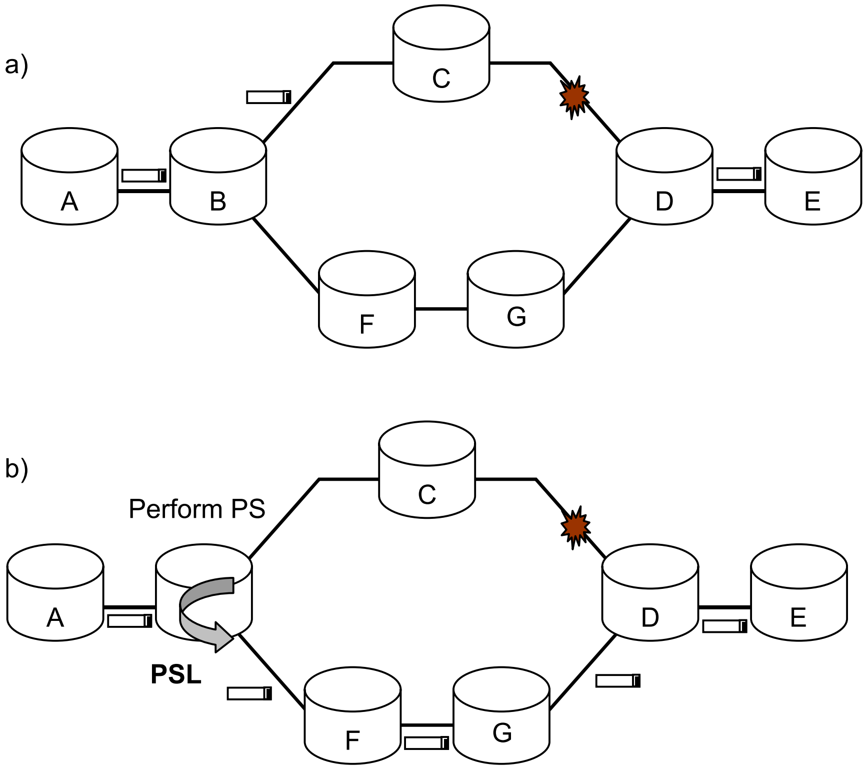


Figure 59: The forwarding of the test packets before and after the network has converged illustrated here with protection switching. a) When the failure occurs but before the network has converged only packets that have traversed the failing link will continue to reach the destination. b) When the network has converged, the packets start to arrive at the destination again over the recovery path.

The packet loss as measured by this scheme is only a measure for the convergence time since it does not take into account the difference in latency between the working path and recovery path. It does take the distance between the failure and the PSL into account which is important because it represents the amount of traffic that is lost because it was sent over the original working path after the failure but before the protection switch operation.

As an alternative for determining the packet loss as a measure for the convergence time it is also possible to measure the maximum delay between two consecutive packets at the receiving side. During the convergence period this delay will be very high since no packets will be received until the network has converged and the packets start to arrive at the destination again. This measure is a good indication for the convergence time from the user point of view. The measured time frame can be seen as the service interruption time from a user point view (f.i. consider that the test packets are video packets for a stream that a user is watching at the destination node). Measuring the maximum delay at the receiving side also measures the difference in delay between the working and the recovery path and thus has a higher dependency on the network topology. In the above example there is an additional delay introduced by the increased length of the recovery path from the PSL to the destination (one additional hop and link). Note that in some occasions the additional latency and length may be negative because the recovery path is shorter than the working path.

The main drawback of measuring the number of missed packets or the maximum delay is that the reversion time cannot be measured. These schemes will only measure the disrupted traffic when the path is changed from the recovery path back to the preferred path. Measuring the disruption during reversion is an interesting experiment, if however the actual reversion time needs to be measured then the individual events need to be measured.

5.2.2.d Comparing the alternatives

Table 13 compares three different measurement approaches to determine the convergence time. The first approach, explained in the previous section, determines the convergence time by measuring the time of the first and last step in the convergence cycle. The second approach measures the delay between the consecutive test packets and determines the convergence time by using the maximum delay value. The third approach determines the convergence time by multiplying the packet loss with the reciprocal of the rate.

Note that we do not include ping in this comparison because ping is bi-directional and thus not useful in the context of MPLS.

Measuring the time of the first and the last event of the convergence cycle has the drawback that the measurement implementation needs to be adapted for every scenario. Another drawback is that this scheme requires time synchronisation and that verification of successful convergence is weak. The major benefit of this approach is that it can measure the reversion time.

Measuring the maximum delay or the packet loss do not suffer from the drawbacks that measuring the individual events does. However measuring the maximum delay has the drawback that the delay must be low enough during working conditions and that the rate of the measurement packets must be precise enough. Similarly, measuring the packet loss requires that there is no packet loss during working conditions and that the rate is precise enough.

Table 13: comparison of the different convergence time measurement schemes.

	Measuring events	Max delay	Packet loss
Generic	no	yes	yes
Failure detected by	local detection	high delay	packet loss
Requirements	time sync.	low delay/ fixed rate	no loss/ fixed rate
Convergence verification	weak	strong	strong
Convergence time determined by	timing events	max delay	packet loss * rate
Reversion time determined by	timing events	n/a	n/a

5.2.2.e The adopted measurement platform

We have adopted the packet loss based convergence time measurement technique for conducting the convergence time measurements. Measuring the convergence events was deemed too error prone since it needs to be adjusted for every convergence scheme and there is little proof of convergence. Measuring the packet loss was chosen over measuring the maximum delay because this model was easier to support with the chosen measurement hardware. As a final note we like to remark that we have compared the results of measuring the maximum delay and the packet loss with a software-based solution. The results were in line with each other but the absolute errors on the measurements were too large for the measurements on the faster convergence schemes. As a result a hardware-based approach was investigated.

First we will describe the role of the measurement hardware and exact setup of the convergence measurements before we examine the precision of the adopted measurement scheme given the intrinsic limitations of real experiments.

Measurement setup

We use the Smartbits 2000 (SMB2000) [52] hardware measurement platform. The Smartbits is equipped with Fast Ethernet Smartcards [171]. Smartcards generates, monitors, and captures data at or beyond wire speed. We send data using the “Continuous Mode” where a constant stream of packets is sent at a user selected inter-packet gap. At the end of the experiment the statistics are gathered. The statistics come in the form of the following counters: packets successfully transmitted, valid packets received and the latency and a latency distribution of the packets. Time measurements on the Smartbits 2000 have a precision of 100ns [172] which is more than sufficient for our experiments.

The experiments are controlled via an application that uses the SmartLib [173, 174]. This application sends a stream of test packets sent with a configurable rate during a configurable amount of time. After the experiment, statistics both from the sending Smartcard and the receiving Smartcard are collected. This allows us to determine the packet loss and the latency distribution of the packets during the experiment.

When the Smartbits is attached to a network as illustrated in Figure 60 it allows to collect statistics of the test traffic sent over the network. We will use this setup to determine the convergence time of the various convergence schemes. However before the experiments are conducted it is important that the requirements of this measurement scheme are validated.

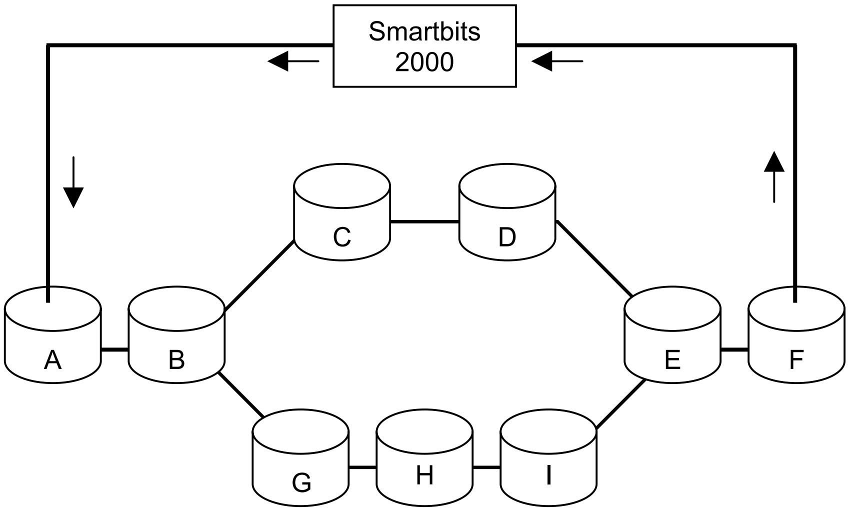


Figure 60: The test network and the test equipment used to conduct the convergence time experiments.

As we have seen in Table 13 the measurement scheme requires that there is no packet loss in the network and that the rate is constant over the experiment. This can easily be verified by running multiple initial tests over the network. Note that meeting these requirements depends on the rate at which the test packets are sent. For example when the packets are sent near to or even above the link rate the requirement that there is no packet loss can obviously not be met.

After the verification that the requirements are met, the experiments can be conducted. Each experiment starts by sending test packets then introducing a network impairment and then collecting the statistics after a suitable long time. It is important that the time during which the packets are sent is longer than the convergence time plus the random time that is waited before the impairment is introduced. In other words it is important that the test packets start to arrive at the destination again after the convergence.

Although care is taken to achieve an as high as possible precision, no experiment is without error. In the next section we will describe the precision of our measurement approach.

5.2.2.f Measurement precision

This section describes the precision of the measurement technique. There are two sources of inaccuracy. The first source is the finite granularity of the measurements and the second is the less than perfect rate at which the test packets are sent.

From Figure 61 we see that the reciprocate of rate (the delay between two test packets) determines the granularity of the measurement. We also see that the measured value can be off by one (both higher and lower) compared to the real value of the convergence period.

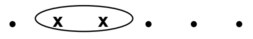
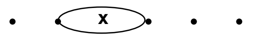
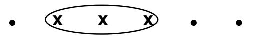
	Experiment	Measured value	Real value
A		2	~2
B		1	~2
C		3	~2

Figure 61: Illustrated here is the imprecision on the measured convergence period introduced by the finite rate of the test packets. The circle represents the convergence period, the dots individual test packets and the x's lost test packets.

In order to keep the error on the measured convergence period as low as possible the rate needs to be as high as possible. However in real life experiments the rate of the test packets is finite due to the limited bandwidth of the network and the limited forwarding capability of the routers. Also the fact that during working conditions no packets may be lost limits the rate too.

Taking into account the fact that the reciprocal of the rate determines the precision of the convergence time, formula (2) needs to be rewritten as:

$$\text{convergence time} = (\text{loss count} \pm 1) * (1 / \text{rate}) \qquad (3)$$

taking the absolute error on the measurement i.e. 1 / rate into account.

The absolute error on the measurements should be as low as possible because it determines the precision of the measurement. However the usefulness of the measurement is determined by the proportion of the absolute error compared to the measured value i.e. the relative error on the measurement. The relative error is commonly expressed in a percentage of the measured value (the percentage error). In Figure 61 the relative error is very high i.e. 50% which is prohibitive. However the figure does not reflect the values in the experiments where the relative error is much lower (ranging from 0.01% up to 5.6%).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	**Lost**
Send	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
Convergence							X	X	X	X	X	X	X	X	X	X	X	X					
Received	1	2	3	4	5	6													19	20	21	22	**14**
Send	.	.						......	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
Convergence							X	X	X	X	X	X	X	X	X	X	X	X					
Received	1	2																	19	20	21	22	**18**
Send	.	.	.	.	.	.	.	.	.	.	.	.	.					.....	.	.	.	.	
Convergence							X	X	X	X	X	X	X	X	X	X	X	X					
Received	1	2	3	4	5	6													19	20	21	22	**14**
Send	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.					.....	.	.	
Convergence							X	X	X	X	X	X	X	X	X	X	X	X					
Received	1	2	3	4	5	6														20	21	22	**9**

Figure 62: This figure shows the influence of delayed test packets on the measured convergence time. The first sequence represents a perfect test packet stream, the convergence period and the measured packet loss. The second, third and fourth sequences show the influence of a delayed test packet on the measured packet loss for different positions of the delayed packet with respect to the convergence period.

This scheme also requires that the rate of the packets is constant. This means that the test packets should be sent at a constant rate but also that they are forwarded with a constant delay over the network. When for example a packet is buffered on a router the instantaneous rate downstream of that node drops. Put in another way, high burstiness or jitter of the test stream introduces an error on the measurements if the burst coincides with the convergence period.

As illustrated above sending packets at a lower rate results in a lower number of missed packets. Similarly sending packets at a higher rate results in more lost packets. When a test packet is delayed by a router then the subsequent packets will be buffered. This means that although the source sends the packets at a fixed rate the packets can also be forwarded at a higher rate.

Now we will determine the probability and the error introduced by an incorrect rate during the convergence period. Unfortunately it is impossible to determine the rate at which the packets are sent during the convergence period because the packets are lost during that period. We can however determine the latency distribution during working conditions.

Table 14: The cumulative latency distribution averaged over 100 experiments. In every experiment 50.000 packets were sent. The large majority of the packets receive a latency between 0.2 and 0.5ms.

Cumulative Latency distribution

<0.2	<0.5	<1	<2.5	<5	<10	<25	**ms**
0	99,32	99,40	99,62	99,89	99,99	100	**%**

Table 14 clearly illustrates that the majority of the packets receive a low delay. 99.4% of the packets receive a delay that is lower than the absolute error of 1ms corresponding with the measurement rate of 1000 packets per second. The problem is the remaining 0.6% of the packets. The reason that some of the packets are significantly delayed compared to others can be found in the fact that Linux does not offer guaranteed scheduling delays (see section 5.2.1.b). These packets can experience a much higher delay, as a matter of fact the highest delay measured is 11.5ms. This translates into an absolute error of 11.5ms. This is acceptable for some experiments because the resulting percentage error is low enough.

However there are other experiments where the absolute error translates into a percentage error in the order of a few ten percents which is not acceptable. For example when the convergence period is 50ms and the absolute error introduced due to a large delay is 10ms then the percentage error is 20%. This percentage error is obviously too high. However when the convergence period is only 50ms the chance that a delayed packet coincides with the convergence period is very small. Moreover when the delayed packet falls entirely inside the convergence period it does not have an influence on the measured convergence time (as illustrated in the third example in Figure 62). So only a delayed packet coinciding with the outmost 20ms of the convergence period (10ms on both sides of the interval) can affect the measured convergence time. When we take into account that the probability to have a delayed packet is low (0.01%) this means that the chance that an experiment is affected is low (0.2% when packets are sent every ms). So we can expect that any errors due to large delays will not influence the average measured convergence time too much.

We are however also interested in the minimum and maximum convergence time. A delayed packet can influence both the minimum and the maximum delay. However since the probability is so small it might not happen at all or if it does it will be easily detected as an anomaly in the series of measured values.

5.3 Investigating convergence times

Now that we have introduced the test network and the measurement scheme we are ready to discuss the convergence time measurements themselves.

Every experiment follows the same pattern. Test traffic starts to flow from source to destination (one packet per ms). Then after a random interval a link failure is introduced. The test traffic continues to flow to allow for convergence. After the convergence statistics are gathered.

In the remainder of this section we will give the test description, the test parameters, the test results and the conclusion for each experiment. When we give the conclusions we try to pinpoint the limitations of the experiments. We start by investigating OSPF convergence.

5.3.1 OSPF convergence

Description of test
In this test we will measure the time required for the network to converge after a single link failure. We will compare two variants of OSPF one where the failure is detected through the Hello protocol and one where the failure detection is immediate.

Immediate failure detection is implemented by disabling the interface on the local router. This will cause the Linux kernel to send a *netlink* message to the OSPF daemon. The OSPF daemon then immediately acts on the failure without relying on the Hello protocol. Note that this netlink message can also be ignored by the OSPF daemon so that it relies on the Hello protocol to detect failures.

Test parameters

Table 15 summarises the experimentation parameters. Every experimentation run consists of 100 individual experiments (Table 15a). During the experiment every 1ms a test packet is sent (Table 15b). As seen before the number of missed packets indicates the convergence time. Standard OSPF parameters are used, i.e. 10s for the hello interval and 40s for the router dead interval (Table 15c-d). The SPF delay is set to 5s (Table 15e). Note that the Hold-Down timer is set to 10s but the experiments are spaced sufficiently far apart that the time between two SPF calculations never exceeds 5s (see Figure 30).

Table 15: Parameters used in the OSPF convergence experiments

	Parameter	**Value**
a	Number of measurements	100
b	Measurement interval	1ms
c	OSPF hello interval	10s
d	OSPF router dead interval	40s
e	OSPF SPF delay	5s

The topology and the network set up shown in Figure 63 have already been described in Figure 60. In this experiment and the following experiments we will simulate a link failure by disabling the interface BC on router B. The convergence time is measured by determining how many test packets, that are sent with a fixed frequency from A towards E, are lost.

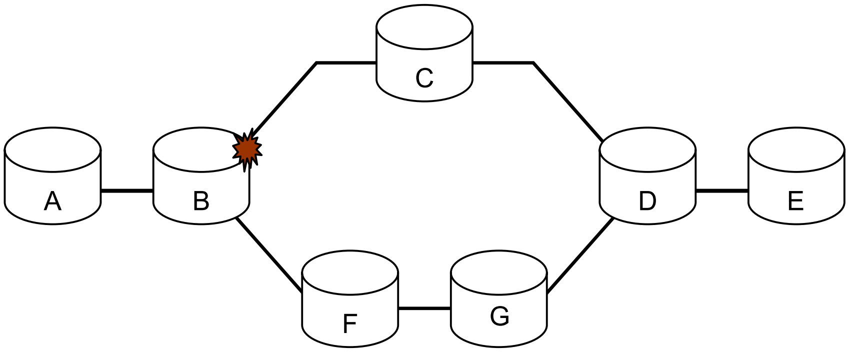
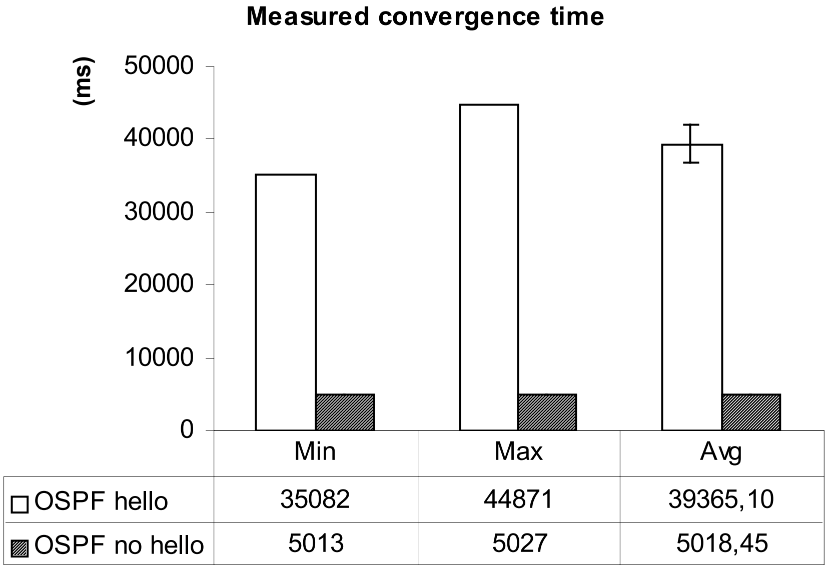


Figure 63: During the OSPF convergence experiments the link BC is brought down. We measure the convergence time by determining how much test packets that are sent between node A and E are lost during convergence.

Test results

Figure 64 shows the results of the two experiments. It is immediately apparent by looking at the graphs that the difference between the two experiments, the failure detection, has a large impact on the convergence time. Note that the standard deviations on the convergence times are depicted on the average bar for both the OSPF protocol with Hello Protocol failure detection and the OSPF protocol with immediate failure detection. However the standard deviation of the latter is so small that it is not visible.



	Min	Max	Avg
□ OSPF hello	35082	44871	39365,10
▨ OSPF no hello	5013	5027	5018,45

Figure 64: The measured convergence times for OSFP with the Hello protocol as failure detection mechanims and OSPF with immediate failure detection.

We will now investigate the experimentally determined convergence times in more detail by comparing them with the theoretical convergence time analysis performed in Chapter 3 and Chapter 4. The second row of Table 16 shows the theoretical convergence time for the OSPF protocol as described in section 3.3.2.c. The third and fourth rows give the theoretical convergence for the two experiments where the parameters are substituted with the values from Table 15.

Table 16: Theoretical convergence times for OSPF

	Fault detection	Notification time	Hold-down time	Recovery operation
	[router dead-hello interval, router dead]s	δ	[SPF delay, hold-down]s	SPF (ε) update RIB (γ)
OSPF+ Hello	[30, 40]s	δ	5s	$\varepsilon+\gamma$
OSPF	κ	δ	5s	$\varepsilon+\gamma$

We start by investigating the fault detection time. The fault detection time for the first experiment is determined by reformulating formula (1) of section 3.2.1.b.

$$T_{detect} = [\text{receive interval– send interval, receive interval}] \qquad (4)$$

If we substitute the Router Dead and the Hello Interval (see Table 15) we get:

$$T_{detectHello} = [30, 40]\text{s} \qquad (5)$$

Note that the actual value of the detection time depends on the timing of the last Hello packet with respect to the failure. If the last packet is sent just before the failure then it will take almost a complete router dead interval before the failure is detected i.e. 40s. However if the last packet is sent almost a hello interval time before the failure it will take up to a hello interval less time to detect the failure i.e. 30s. Since the time that the failure is introduced in the network is randomised we can expect to see more or less the full range of the detection interval ($T_{detectHello}$).

We can determine the detection interval and verify formula (4) by comparing the convergence times of the two experiments. Since the two experiments are identical except for the failure detection scheme the detection time of the Hello protocol can be determined by subtracting the convergence time without failure detection from the convergence time with the hello protocol:

$$T_{detectHello} - \kappa = C_{OSPF+Hello} - C_{OSPF} \qquad (6)$$

We thereby assume that the detection time with immediate failure detection (κ) is very small compared to the detection time with the Hello protocol.

Note that we will denote convergence times by a capital C with the protocol subscripted.

Formula (6) does not really reflect it but convergence times therein are actually intervals which is of course correct since $T_{detectHello}$ is an interval too. To determine $T_{detectHello}$ we will use the average value for the C_{OSPF} and the entire range of the $C_{OSPF+Hello}$ interval. This means that we consider that the C_{OSPF} values are constant which is not entirely true. This assumption introduces an error of about 10ms. Taking this into account leads to the experimentally measured interval for the failure detection with the Hello protocol:

$$T'_{detectHello} = [30064, 39853]\text{ms} \pm 10\text{ms} \qquad (7)$$

Note that we will denote experimental measured values with a single accent.

We see that (7) approximates (5) well. The errors introduced here are due to the fact that we probe the 10s hello interval with 100 individual experiments. We cannot expect that the full range of possible values are exposed this way. However we see that 98% of the theoretical interval is confirmed by the experiments. This should be considered sufficient to experimentally validate (5) given that the parameters are in the second range and that the maximum offset is in the order of 200ms. Another source of inaccuracy is introduced by the absolute error on the measured convergence time and the fact that the convergence times of OSPF with immediate failure detection are not fixed.

We will now focus on the second experiment because its standard deviation is much lower which makes it a lot easier and more precise to determine the length of the other phases of the convergence cycle. Both the failure detection time, the notification time and the convergence operations time are expected to be low. The failure detection time should be low because immediate failure detection is used. In this experiment this means that the OSPF protocol is notified immediately when the link is disabled. The notification time is also low because router B does not depend on the receipt of external LSAs to perform the recovery operations. Finally the recovery operations, the SPF calculation and the update of the RIB are expected to be small too since the network is small (about 10 routes). This means that the SPF delay should dominate the convergence time, from Figure 64 we see that this is the case:

$$C'_{OSPF} = [5013, 5027]\text{ms} \pm 1\text{ms} \qquad (8)$$

This means that the time to detect the failure plus the time to calculate and install the routing tables is somewhere between 13 and 27ms.

Although the SPF delay dominates the convergence time it is not feasible to eliminate the SPF delay since it is an important stability feature of OSPF. We will investigate the OSPF stability in section 5.4.3.

A final observation is that the standard deviation with immediate failure detection is two orders of magnitude lower than when the Hello protocol is used to detect failures (13ms vs. 2628ms respectively). The high standard deviation is caused by the large range over which the detection time can differ between the individual experiments.

Conclusion

The failure detection with the Hello protocol makes up the major part of the OSPF convergence. If immediate failure detection is used then the major part of the convergence time is taken by the SPF delay. This is only true when the network and the number of routes are relatively small otherwise the LSA transmission time, the SPF calculation and the RIP update time become significant too. Also, under these circumstances one SPF may not cover all the LSA updates so the convergence time is delayed with 10s, the Hold-Down time, per additional SPF. Finally, the theoretical convergence times and the experimental convergence times are consistent with each other in this experiment.

5.3.2 MPLS rerouting with RSVP-TE

In this experiment we will look at convergence in MPLS networks. More specifically we will compare soft-state MPLS rerouting with triggered MPLS rerouting.

Description of test

In this test we will measure the time required for the MPLS network to converge after a single link failure on a single LSP. First we set up an LSP and map the test traffic on it. Then we will introduce a link failure on this LSP. The MPLS daemons rely on the OSPF protocol with immediate failure detection to detect this failure and calculate new routing tables. In the first experiment we will use pure soft-state rerouting and in the second experiment we will use triggered rerouting. We will use RSVP-TE as the MPLS daemon since LDP does not support soft-state rerouting.

Test parameters

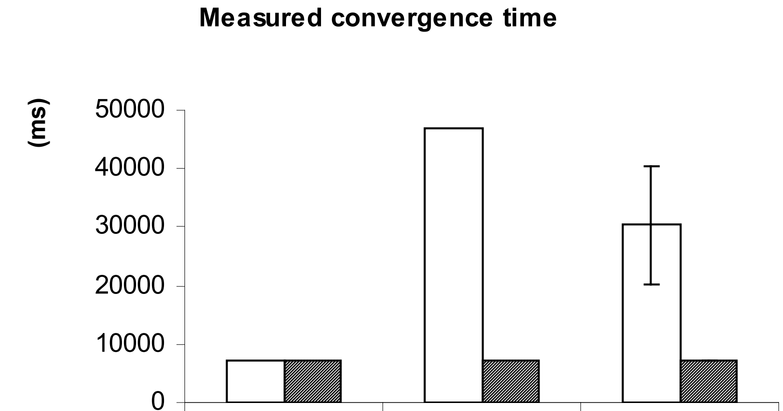
Table 17: Parameters used in the MPLS experiments

	Parameter	**Value**
a	Number of measurements	100
b	Measurement interval	1ms
c	OSPF SPF delay	5s
d	RSVP refresh period (R)	30s
e	RSVP hold-down	2s

We reuse the same experimentation parameters as in the first experiment where applicable. More specifically the same OSPF parameters are used and the same link failure is simulated. A standard RSVP refresh period (R) of 30s is used to determine how often an LSP is refreshed ([0.5R, 1.5R]) [31]. The RSVP hold-down period (2s) determines how long the MPLS daemon waits before rerouting the LSP after the routing tables have changed.

Test results

Figure 65 clearly shows a huge difference in the maximum convergence time between soft-state MPLS rerouting and triggered MPLS rerouting. We will now investigate the convergence times in more detail.



	Min	Max	Avg
□ Softstate RSVP	7031	46803	30348,00
▨ Triggered RSVP	7030	7033	7031,43

Figure 65: The measured convergence times for MPLS soft-state and triggered rerouting.

We start by giving the theoretical convergence times for the two convergence schemes (Table 18).

Table 18: The theoretical convergence times for soft-state and triggered MPLS rerouting.

	Fault detection	Notification time	Hold-down time	Recovery operation
RSVP soft-state	IP convergence	IP convergence	Hold-down + [0,5-1,5]R	LSP setup (ζ)
RSVP soft-state	IP convergence	IP convergence	2s + [15-45]s	LSP setup (ζ)
RSVP triggered	IP convergence	IP convergence	Hold-down	LSP setup (ζ)
RSVP triggered	IP convergence	IP convergence	2s	LSP setup (ζ)

We see that both soft-state and triggered MPLS rerouting depend on the IP convergence protocol. Since the IP convergence is the same in the two experiments it is the trigger for the MPLS convergence that differs significantly. In soft-state rerouting the trigger is given by a request to refresh an existing LSP while in triggered MPLS rerouting the convergence is triggered by the IP convergence. Note that in both cases a hold-down timer of 2s is used.

By subtracting the convergence times we can determine the delay introduced by the RSVP refresh period.

$$C'_{RSVP\_SOFT\_STATE} - C'_{RSVP\_TRGGERED} = [1, 39770]\text{ms} \pm 1\text{ms} \qquad (9)$$

The first observation is that in the best case soft-state rerouting does not introduce an additional delay to the convergence time. In this case the LSP is refreshed exactly at the same time the routing tables are changed. The daemon will then wait 2s (the hold-down time) before rerouting the LSP. This results in exactly the same behaviour as with triggered rerouting. However the worst case behaviour is completely different. From the experiments it is clear that the convergence time can be almost 40s higher. This happens when the LSP has been refreshed just before the routing tables are updated and subsequently a large refresh period is used. Remember that the refresh period is randomised over the interval [0.5, 1.5]R = [15, 45]s. So theoretically the additional convergence of soft-state rerouting compared to triggered rerouting can be up to 45s. In this experiment we only observe an additional delay of 40s. This is caused by the imperfect randomisation of the refresh period in the implementation. We only observed the following refresh periods {20, 23, 25, 26, 30, 38, 40, 42}s over 100 experiments which is a quite crude approximation of the theoretical interval.

The convergence time of triggered MPLS rerouting is much more predictable and in general much lower than with soft-state rerouting. We will investigate the convergence time of triggered MPLS rerouting in the next experiment where we compare the convergence time of triggered MPLS rerouting with OSPF.

Conclusion

Both soft-state and triggered MPLS depend on IP convergence. Soft-state rerouting should be avoided if possible because it can add a significant delay to the convergence time. The maximum delay depends on the length of the refresh period. It is possible to reduce the refresh period but this will increase the signalling overhead and will limit the scalability.

Triggered rerouting has a very predictable convergence time (only 3ms difference between the minimum and the maximum convergence time) and its convergence time is typically much lower.

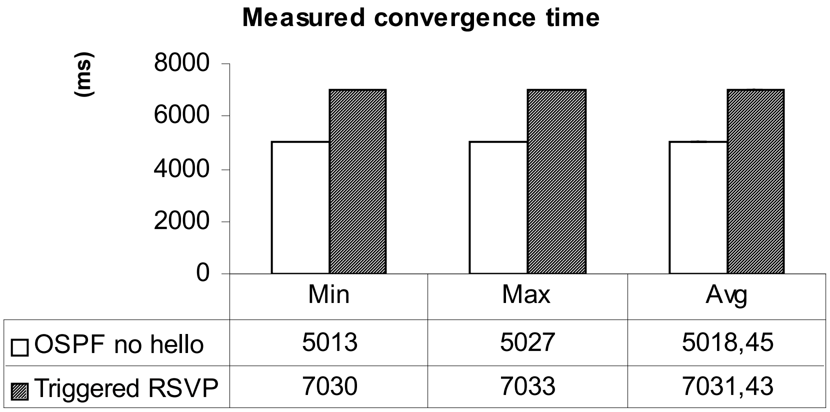
5.3.3 Comparing OSPF with MPLS rerouting

Description of test

In this test we will compare the convergence time of OSPF with triggered MPLS rerouting to gain further insight in the convergence time of triggered MPLS rerouting.

Test parameters

This is not really a new test but rather a comparison of the results from the first and the second experiment so no new test parameters are introduced.



	Min	Max	Avg
□ OSPF no hello	5013	5027	5018,45
▨ Triggered RSVP	7030	7033	7031,43

Figure 66: The experimental convergence times for OSPF and triggered MPLS rerouting.

Test results

As we see in Figure 66, MPLS rerouting is about 2s slower than OSPF rerouting. This is caused by the MPLS hold-down timer of 2s seconds. The timer is used to let the routing tables stabilise before rerouting the LSP. So although the hold-down timer is not beneficial for the convergence time it does increase the stability of the MPLS rerouting. One conclusion might be that the hold-down timer should be reduced to a few milliseconds. However when for example RIP is used the ideal hold-down timer is very hard to determine. Even when OSPF is used in conjunction with MPLS it is difficult to predict the optimal hold-down timer. And finally it is important to note that the hold-down timer is much more important in revertive mode than when converging so an asymmetric hold-down timer might be more appropriate.

Conclusion

The hold-down timer increases the convergence time of triggered MPLS rerouting significantly. In order to reduce the convergence time it is advisable to tune the hold-down timer. However this is not an easy task because it depends on the time it takes the IP routing protocol to stabilise all its routing tables on the path of the LSP which in turn depends on the routing protocol and the load and the size of network. Note that the hold-down timer is of most importance when reverting back to the working path. We will return to the subject of hold-down timers in MPLS when we discuss the stability of MPLS rerouting in section 5.4.4.

5.3.4 Comparing FTCR with Protection Switching

Description of test

In this test we will compare the convergence time of FTCR in failure local FSL selection mode with local protection switching.

Test parameters

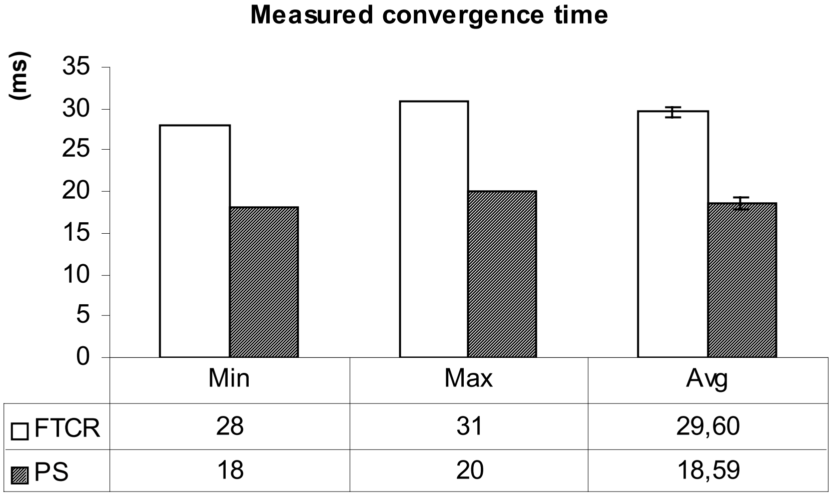
Table 19: The experimentation parameters used in FTCR and protection switching convergence

	Parameter	Value
a	Number of measurements	100
b	Measurement interval	1ms
c	FTCR failure local FSL	
d	FTCR presume link failure	
e	Local protection	

Comparing the local modes of the convergence schemes is fair since in both modes there is no failure notification necessary. Note that we could also compare global protection switching and ingress repair FTCR which typically translates into the same notification time. Since we are not interested in the notification time we choose the relevant local repair modes.

Test results

From Figure 67 we clearly see that the convergence time of FTCR is slower than that of protection switching. This is not surprising since FTCR has to calculate, set up and switch over to the recovery LSP after the failure has been detected while protection switching only has to switch over to an already established recovery LSP (see Table 20).



	Min	Max	Avg
□ FTCR	28	31	29,60
▨ PS	18	20	18,59

Figure 67: Experimental results for the convergence time of FTCR and local protection switching.

Before we analyse the results we would also like to mention that one measurement in the FTCR experimentation run was removed. As we see, the adjusted values range between 28ms and 31ms. However we measured one convergence time of more than 40ms. This value was removed from the series to calculate the average, minimum and maximum convergence times. Note that this individual experiment was affected by a significantly delayed packet as we have described in section 5.2.2.f.

The first observation that we make is that the time to detect the failure and the time to switch over to the pre-established recovery LSP is about 20 ms. We will not try to further analyse the convergence time of protection switching but rather take this value as given since it is highly implementation dependent.

Table 20: Theoretical convergence times for FTCR and protection switching.

	Fault detection	Notification time	Hold-down time	Recovery operation
FTCR	κ	η	0s	Calc. recovery LSP (θ) LSP setup (ζ)
PS	κ	η	0s	Update LIB (ι)

The additional convergence time of FTCR compared to protection switching is the result of the recovery LSP calculation and setup. By investigating the convergence times of FTCR and PS we get an estimation of the LSP calculation (θ) and setup (ζ) time.

$$\zeta + \theta - \iota = C_{FTCR} - C_{PS} \tag{10}$$

$$\zeta' + \theta' - \iota' = C'_{FTCR} - C'_{PS} = [10, 11]\text{ms} \pm 1\text{ms} \tag{11}$$

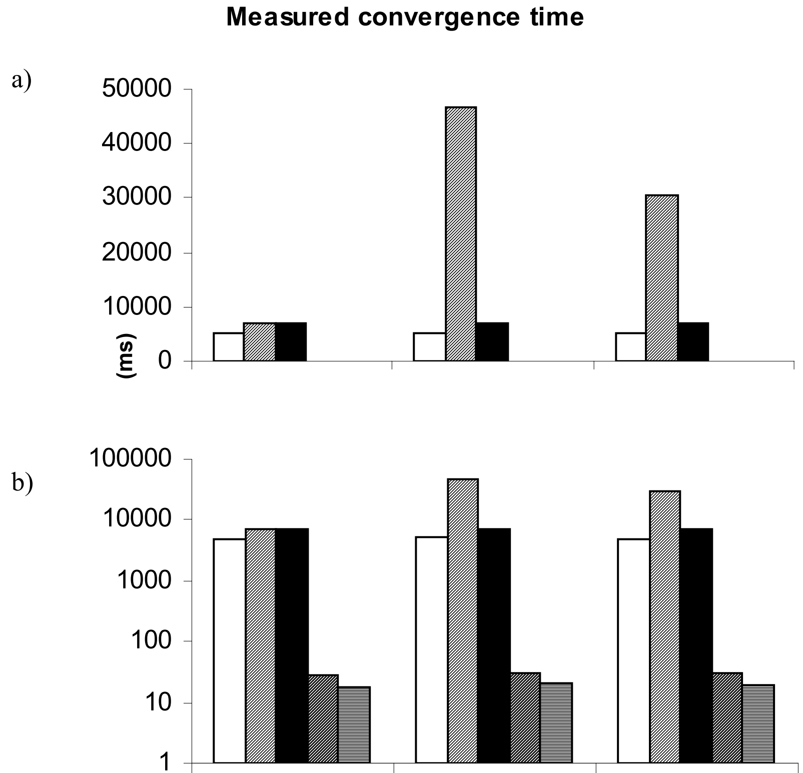
The LIB update time (ι) can be considered small compared to the LSP setup time since the LSP setup time encompasses multiple LIB update times in the different routers along the LSP. Profiling the FSL shows that the calculation takes up less than 1ms. We can therefore conclude that the LSP setup time dominates the difference in convergence time between FTCR and PS in our experiment. However as the network grows larger the SPF calculation can become significant too.

Conclusion

FTCR is slower than PS because the recovery LSPs have to be calculated and set up after the failure has been detected. In our small test network the bulk of this additional time is due to the LSP setup time rather than the SPF calculation time. In larger networks both the SPF time and the LSP set up time will increase thereby increasing the difference in convergence time between PS and FTCR. We will return to this aspect in the last chapter when we discuss how FTCR can be improved.

5.3.5 Conclusions

In this section we look at all the convergence schemes simultaneously (see Figure 68). Looking at these results, the convergence schemes can be classified in three classes. Protection switching and FTCR are the fastest schemes, OSPF and triggered MPLS rerouting are slower but have a quite predictable convergence time and finally pure soft-state MPLS rerouting which is potentially very slow and has a very high standard deviation.



	Min	Max	Avg
□ OSPF no hello	5013	5027	5018,45
▨ Softstate RSVP	7031	46803	30348,00
■ Triggered RSVP	7030	7033	7031,43
▨ FTCR	28	31	29,60
▦ PS	18	20	18,59

Figure 68: Convergence time of all the convergence schemes. a) The graph at the top uses a linear scale while b) the graph at the bottom uses a logarithmic scale.

The three classes are clearly illustrated in Figure 68a where soft-state MPLS rerouting has very high values compared to OSPF and triggered MPLS rerouting and the values from FTCR and PS are so low that they are not even visible with the scale used in this graph. Figure 68b which uses a logarithmic scale does show all the convergence times.

The slowest convergence scheme, soft-state MPLS rerouting, should be avoided if possible. It should only be used when it is impossible for the MPLS daemon to be notified of changes in the routing tables. Even then it might be possible to probe the routing tables which will be much faster than pure soft-state rerouting. So even though soft-state rerouting is an elegant solution to MPLS rerouting it cannot be used when the convergence times are of any importance.

OSPF and MPLS triggered rerouting have a comparable convergence time. MPLS rerouting is slightly slower because a hold-down timer is used to delay the rerouting of the LSPs a bit so that the routing tables can stabilise. Tuning the hold-down timer is recommended and can decrease the convergence time.

Finally, protection switching and FTCR are the fastest schemes. PS is faster than FTCR because the recovery LSPs are pre-established while FTCR calculates and sets up the recovery LSPs as part of the recovery operations. When the network is small then the additional convergence time of FTCR compared to PS is small. However as the network size increases the convergence time of FTCR will increase due to the increased time required to set up and calculate the recovery LSPs while the convergence time for PS will remain more or less constant.

5.4 Investigating stability

5.4.1 Introduction

So far we focussed on the time to converge after a failure. We first investigated the theoretical convergence time and then we investigated the convergence times in the ideal circumstances of a test network. We must however be careful not to draw incorrect conclusions from these experiments because of the ideal circumstances and the limited size of the network in which they were conducted. For example one may come to the premature conclusion that the hold-down timers can be eliminated. This is obviously incorrect because they increase the stability of the convergence schemes. In this section we will investigate the stability features of the convergence schemes and have a critical look at their function and applicability. As we will see the stability features often have a negative effect on the convergence time. So there is typically a trade-off between convergence time and stability. That does not mean that the convergence time cannot be improved while retaining the same stability level.

5.4.2 Failure detection

We start the discussion about the stability of the convergence techniques with the first phase of the convergence cycle: the fault detection. We discuss the OSPF Hello Protocol and hardware-based failure detection.

5.4.2.a The OSPF hello protocol

As we have seen the Hello Interval of the Hello protocol is several times smaller than the router dead interval. This means that more than one hello packet must be lost before a failure event is triggered. Consider for a moment that the hello interval and the router dead interval are equal this would mean that the loss of a single hello packet would lead to a failure event. This obviously leads to much faster failure detection. The drawback is that losing a single hello packet may not be a good measure for detecting a failure since the packet may be lost due to high link load or a transmission error. This leads to a stability/convergence time trade-off. Decreasing the router dead interval decreases the convergence time but it decreases the stability too. The standard values of OSPF i.e. a 10s hello interval and a 40s router dead interval may be too high in contemporary networks with high speed interfaces with low error rates and networks that are over-provisioned even during busy periods.

In Diffserv networks it is also possible to treat the Hello packets with high priority thereby reducing the loss probability which in turn allows for smaller hello and router dead intervals. Yet another way to increase stability of the Hello protocol is to treat every packet received on the interface as an *implicit Hello* [175, 176] (see section 3.2.1.c). This means that the router dead timer is reset whenever a packet is received over an interface. While this decreases the possibility of detecting a false positive failure during high link load it is possible to have false negatives.

Consider that the OSPF control stack of router A crashes. This router will no longer send Hello messages. Router B which has a Hello adjacency with router A will not detect this failure if implicit Hello's are used since any packet received from the interface will reset the router dead timer. The implicit Hellos thus fail to detect the failure of the OSPF protocol stack on router A. This will have very negative effects, for example node B will retransmit LSAs continuously to node A because the LSAs are not acknowledged by A.

In conclusion, while high Router Dead and Hello Interval timers do increase the stability it takes some investigation to optimise the stability/convergence time trade-off for each interface in the network.

5.4.2.b Hardware based failure detection

The stability of this form of failure detection depends on the reliability of the mechanism (note that we introduced reliable failure detection already in section 3.3.4.f). The mechanism is considered reliable when a failure event coming from the mechanism is always the right trigger for a convergence operation. If the failure detection mechanism is not reliable then it may be advisable to take some counter measures. An example might be to introduce a (small) hold-down timer to verify that the failure indication remains valid during the hold-down period.

For example if Packet Over SONET (POS) is used as link-layer the failure detection is very fast. But reacting immediately to the failure indication is not always advisable. If the traffic is protected by the SONET layer a hold-down timer should be used ([177] suggests 60ms). The hold-down timer allows the SONET layer to recover the traffic before the IP or MPLS layer will notice and react to the failure. When the SONET layer cannot recover the fault, the failure indication signal will propagate to the IP or MPLS layer and their convergence schemes can recover the traffic. More multi-layer aspects of MPLS resilience were presented in [6, 178, 179, 180].

5.4.3 OSPF stability

In this section we will investigate which features of the OSPF protocol increase the stability and which have a negative effect on the stability. We will also investigate how this affects the convergence time.

5.4.3.a SPF delay and Hold-Down

We already discussed how the Hello Protocol tries to shield OSPF from detecting false positive failures. Another important stability feature of OSPF are the hold-down timers i.e. the SPF delay and the SPF Hold-Down timer. The purpose of these timers is to reduce the number of SPF calculations. The number of SPF calculations should be limited because a SPF calculation is a disruptive operation which can last up from 1ms in a small network (as in our experiments) up to a few hundreds of milliseconds in large networks [139]. During the SPF calculation the OSPF protocol stack can be unresponsive so that for example Hello packets are not sent or severely delayed. On some router platforms the SPF calculation also interferes with the forwarding.

SPF calculations are triggered by link-state updates. So in order to reduce the number of SPF calculations one should aim to incorporate as many link-state updates as possible. It is thereby important to note that a single failure leads to more than one link-state update. When a link fails the two endpoints of the link send a link-state update, when a node fails all the attached nodes send a link-state update (see section 3.3.2.b). Moreover more than one failure can occur simultaneously leading to even more link-state updates.

Another situation where the OSPF hold-down timers are beneficial is the situation where a link or node fluctuates rapidly between an operational and down status. In this situation the hold-down timers prevent that the routing tables fluctuate too frequently as a result of the fluctuations of the link or node. Especially the SPF Hold-Down period is important because it takes the time since the last SPF calculation into account. Fluctuations like this can be addressed by using an exponential hold-down timer [177]. This means that the hold-down timer is not constant but it increases when SPF calculations are frequent and it decreases to a pre-set minimal when no SPF calculations have occurred in a while. This does reduce the frequency of the SPF calculations gradually during fluctuations. However it also increases the packet loss when the status of the link goes back down.

Even though delaying the SPF calculation is important to catch as much link-state updates as possible the SPF can obviously not be postponed indefinitely. As we have seen the Zebra OSPF implementation delays the SPF calculation somewhere between 5s (SPF Delay) and 10s (SPF Hold-Down) [46]. The above scheme of having two different hold-down timers to the SPF delay seems like a good trade-off between convergence time and stability. During stable periods the SPF calculation is performed quicker (5s) than during periods of instability (10s). It is however easy to illustrate that it is sub-optimal in even common situations. We will illustrate this with an example.

An example

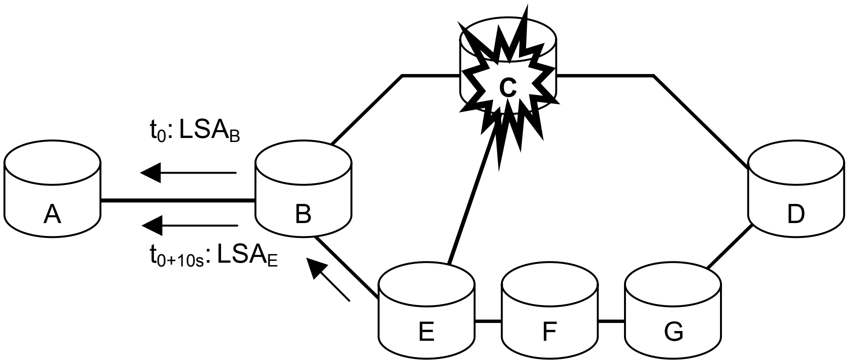


Figure 69: When a node fails, the attached routers will detect that their link attached to that node has failed. However the detection time varies from node to node. Illustrated above is how node E detects the failure 10s after node B. This can lead to temporarily incorrect shortest paths.

Consider that node C fails. Node B, D and node E will eventually detect that the adjacency that they have built with node C fails. However the time at which they detect the failure varies from node to node. With the standard Hello Interval and Router Dead timers the difference can be up to 10s (the detection variation is equal to the Hello Interval, see section 3.3.1.c).

Consider that node B is the first to detect the failure. It will then spread a link-state advertisement at t_0 with the distance from B to C over link BC set to infinity. Note that node B always presumes that the link rather than the node has failed (comparable to FTCR's presume link failure mode). Node failures are handled properly by link-state protocols because the other attached router will also send link-state updates. However as we just mentioned, these updates can be dispersed over time. Consider that this is the case, that for example nodes E and D detect the failure 10s later than node B. When node A receives the link-state advertisement from node B it will wait SPF Delay i.e. 5s before the SPF calculation. However this will not be enough to take the link-state updates from node D and E into account since they occur 10s later. As a result a wrong SPF tree will be calculated (e.g. the shortest path from A to D will be A-B-E-C-D). To make matters worse the next SPF calculation at node A will only be performed after the Hold-Down delay (10s).

In the above example we have seen that LSAs arrive at different times causing routing anomalies and delays in the convergence. In the example this was caused by the fact that the LSAs were sent at different times due to the difference in detection time. Another cause for a large variation of the arrival times of the LSAs is large delays in the transmission of the LSAs in the network. This in turn can be caused by low speed links, networks of large size and congestion in the network. Significant delays in the LSAs can lead to SPF calculations that take the new LSAs into account in one part of the network while in another part of the network the LSAs have not yet been taken into account This can lead to routing anomalies. Again the solution is to increase the hold-down timers but at the cost of slower convergence.

The above illustrates that determining the correct SPF Delay and Hold-Down timers is difficult. There is not only a stability/convergence time trade-off but also the level of disruption a SPF calculation causes, needs to be taken into account. To complicate matters even more the Hello Interval should be taken into account when determining the optimal hold-down timers. Finally the transmission delay of the LSAs must be considered.

5.4.3.b Link-state advertisement storms

As we have seen in the previous section the SPF hold-down timers protect the network against too frequent SPF calculations. We have seen that determining the optimal length of these timers is hard. We will now investigate situations in which OSPF implementations are stressed to the limit or even can *melt-down* [181].

OSPF is not able to withstand a large number of simultaneous or near-simultaneous link-state advertisements (an LSA storm) [175, 176]. An LSA storm can be caused by a failure in the transport network that carries a large number of links. It can also be caused by the simultaneous failure on a number of nodes (e.g. a failure in a hosting complex). Yet another trigger from an LSA storm might be the (near) synchronisation of LSA refresh instants or the network is brought back up after software and or hardware revisions.

Perhaps more important than the reasons for the LSA storms are the consequences. LSA storms impose a high load on the OSPF protocol stack causing packets to be delayed or even dropped. This causes Hello packets to be missed and the Router Dead timer to be triggered. This will lead to more LSAs since link-state advertisements will be sent by the endpoints to declare the link dead. The delay in the processing of the OSPF protocol packets will also cause that some of them will not be acknowledged on time. Both LSAs and database description packets need to be acknowledged. If they are not acknowledged on time they will be retransmitted. The retransmission of these packets causes even more OSPF protocol traffic. If the original LSA storm is large enough then the secondary effects (retransmissions and adjacencies declared dead) can indefinitely sustain the overload in the OSPF protocol stacks. In this case manual intervention might be necessary.

Current solutions proposed to handle LSA storms include prioritising Hello packets and acknowledgements and adding congestion control to OSPF [176, 182].

5.4.3.c Conclusion

We have seen that even though OSPF is a mature routing protocol, instability can still occur. Properly tuned Hello Interval, Router Dead interval, SPF Delay and Hold-Down timers will vastly improve the stability. However increasing these timers typically has a negative effect on the convergence time. Besides, finding the optimal values is difficult and depends on a large number of factors. Recent research proposes new stability enhancing measures to cope with large LSA storms by prioritising certain OSPF protocol packets and by adding congestion control to OSPF.

5.4.4 MPLS rerouting stability

In this next section we will investigate the stability of MPLS rerouting. The stability of MPLS rerouting depends largely on the stability of routing protocol used to determine the paths of the LSPs. After the routing tables have changed a hold-down timer is used to make sure that the routing tables are stable. In the experiments in section 5.3.2 we used a hold-down timer of 2s. As we have seen in the example in the previous section about OSPF stability this might not be enough to ensure stability of the routing tables. So using a hold-down will increase the stability but will not guarantee it in all circumstances.

Adding a hold-down timer is not the only stability enhancing technique that is used in MPLS rerouting, as we have seen in section 2.3.5 both loop detection and loop prevention techniques exist. These techniques prevent that an LSP is used that contains a loop.

MPLS adds more robustness because of the separation of control and forwarding (see section 2.3). Let us revisit the example in the OSPF stability section (see Figure 69). We have seen that incorrect routing tables can occur. In that example node A has calculated an incorrect routing table because it had not received the link-state advertisement from node E yet. As a result, the shortest path from node A to node D will be A-B-E-C-D which is obviously incorrect since node C has failed. Consider now that triggered MPLS rerouting is used. The MPLS daemon on node A will be notified of the new routing tables. After the hold-down period node A will reroute the LSP A-B-C-D along the path A-B-E-C-D. The LSP setup will fail because node C will not process the label request (see Figure 70).

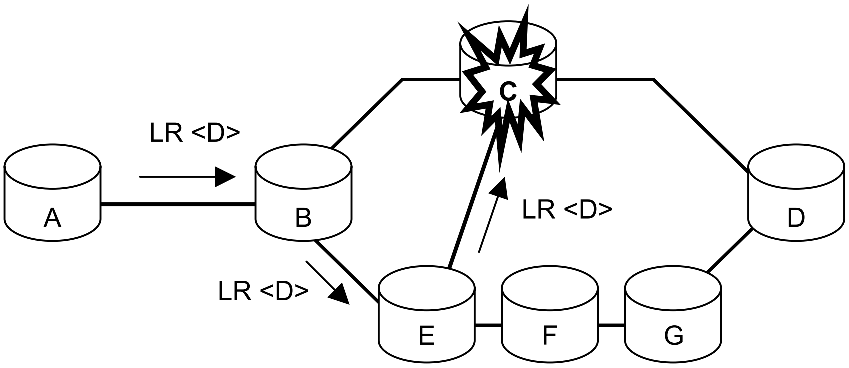


Figure 70: When routing anomalies occur the processing of label requests (LR) can fail. This figure illustrates that the LSP will not be set up because node C is down. We refer to Figure 69 for the circumstances under which this anomaly can occur.

Eventually node A will calculate the correct routing tables (after it has received the LSA from node E and D). It will then trigger node A again and the LSP will be set up over the correct path. The stability improvement comes from the fact that the traffic that was disrupted by the node failure was not switched over to an incorrect LSP.

The above example illustrates that the path of an LSP is validated before traffic is switched over to it. It is clear this improves the stability. However the convergence time does not improve. On the contrary the convergence time in MPLS is slightly higher than in OSPF since the label request messages have to propagate towards the egress and then the label mapping messages have to propagate back from the egress to the ingress. This additional delay compared to OSPF is typically small but might be significant in large networks or networks with high propagation delays.

Finally we look at the differences between CR-LDP and RSVP-TE that are important with respect to stability. CR-LDP uses the reliable communication of TCP as transport protocol. RSVP-TE on the other hand uses raw IP which is not reliable [143]. In both cases retransmissions occur when a signalling packet is lost. CR-LDP relies on TCP for its retransmissions while in RSVP-TE the soft-state mechanism will cause the retransmission. The retransmission with CR-LDP will occur much faster than with RSVP-TE.

In general soft-state protocols are considered more fault tolerant than hard-state protocols. Because RSVP-TE runs over an unreliable, connectionless transport, it lends itself well to a system that must survive hardware failures or software malfunctions by switching over to backup resources (fault tolerance). Any control state that is lost during this *failover* will be recovered by the next state refresh. On the other hand CR-LDP uses reliable delivery of control messages which makes it hard to support failover. Fault tolerance for LDP is currently being standardised [183].

5.4.5 Protection Switching stability

Protection switching uses pre-established recovery LSPs to recover the working LSPs during fault conditions. The fact that the recovery LSPs are pre-established induce both benefits and drawbacks from a stability point-of-view. One benefit is that the computation of the recovery LSP is done in advance which means that PS does not have to take measures to co-ordinate and limit the SPF calculations like in OSPF. The fact that the recovery LSPs are set up in advance means that the potential instability of the routing tables after the failure does not have to be taken into account.

The main weakness of PS with respect to stability is the support for multiple failures. When multiple failures occur there is no guarantee that the recovery LSP is not affected. To cover this case the PSL needs to be informed of both faults on the working paths and faults on the recovery paths. When multiple failures occur more or less simultaneously it is possible that the working traffic needs to be switched over to a number of recovery LSPs before the final recovery LSP is found. Supporting these scenarios requires a large number of recovery LSPs. So full coverage is very difficult to achieve especially in the light of multiple failures. Also care must be taken that the PSL is able to detect that a recovery operation has failed.

5.4.6 FTCR stability

FTCR combines some properties of OSPF rerouting and MPLS rerouting and has some specific properties too. We will start by describing the stability of FTCR compared to OSPF then we look at how FTCR relates to MPLS rerouting in terms of stability.

5.4.6.a Comparing FTCR and OSPF stability

The major convergence speed improvement of FTCR compared to OSPF is attributed to the fact FTCR does not use the SPF Delay and Hold-Down timers. In FTCR, rerouting an affected path is quasi instantaneous. We will now investigate if this affects the stability.

As we have seen OSPF requires hold-down timers in order to reduce the number of SPF calculations and to capture as much LSA as possible for each SPF calculation. This explains why OSPF needs hold-down timers. FTCR does not need these timers because there is always a single node, the FSL, that is responsible for a given failure with respect to an affected LSP. For example when a link fails, the failure will be detected by the two endpoints but only the upstream node will be responsible to recover the LSPs routed downstream over the fault. Similarly when a node fails, only one node is responsible to repair a given LSP. In FTCR the recovery actions are not distributed but rather centralised in the different FSLs. This means that there is no need to co-ordinate the recovery actions. OSPF requires hold-down timers because the recovery operations are distributed. Furthermore, in OSPF all the operations are distributed: the failure detection, the failure notification as well as the recovery operations.

We now investigate the stability of FTCR in more detail. We start by looking at the stability in the different FSL selection modes, then we look at multiple failures followed by a look at the stability when fluctuations occur and finally how FTCR relates to LSA storms.

FSL selection modes

We have seen that the topological location of the FSL depends on the FSL selection mode. In ingress repair mode and potentially in nearest non-looping FSL selection mode a FIS needs to be sent from the failure detecting node towards the FSL. However sending a FIS towards the FSL is not the same as distributing the recovery actions. In this case the recovery operation is not performed by the failure detection node but it is still centralised in a single node i.e. the FSL.

Multiple failures

Multiple failures seem like an exception to the statement that the recovery operations are performed by a single node. When multiple failures occur there can indeed be more than one FSL (in failure-local and nearest non-looping FSL selection mode). However there is still only one FSL responsible for each failure for a given LSP. Under these circumstances the different FSLs do not need to explicitly co-ordinate with each other although they might interact implicitly through (request) rerouting.

For example consider that two failures occur and that the upstream failure occurs first. If the upstream FSL reroutes an LSP over the second failure then the second FSL will simply reroute the LSP (or the LSP request). There is no co-ordination required. The upstream does not need to be informed of the fact that the downstream FSL has rerouted its LSP.

Fluctuations

So far we addressed why FTCR does not need hold-down timers when one or more failures occur. There are two more situations where the hold-down timers are beneficial in OSPF. First of all during reversion it is important to make sure that the preferred working path is operational before reversing the traffic. In revertive mode FTCR relies on IP routing so it does use, implicitly, the hold-down delays.

Secondly when a link or node status fluctuates between up and down status it is important to prevent too frequent route flapping in the network. The hold-down timers of OSPF help to prevent too frequent SPF calculations and thus too frequent routing table updates. We will now investigate how FTCR performs in this situation.

The first remark is that the link failure detection mechanism should try to avoid this unstable behaviour in the first place. But even if this happens FTCR still reacts in a stable way to these events. Let us first repeat how OSPF reacts to this. When the link goes down OSPF will delay the SPF calculation. When the timers expire the new routing tables will be calculated and installed. When the link goes back up the traffic will be reversed back to the preferred working path after the hold-down period. When the link goes back down OSPF will, after the hold-down, switch the traffic back over again etc.

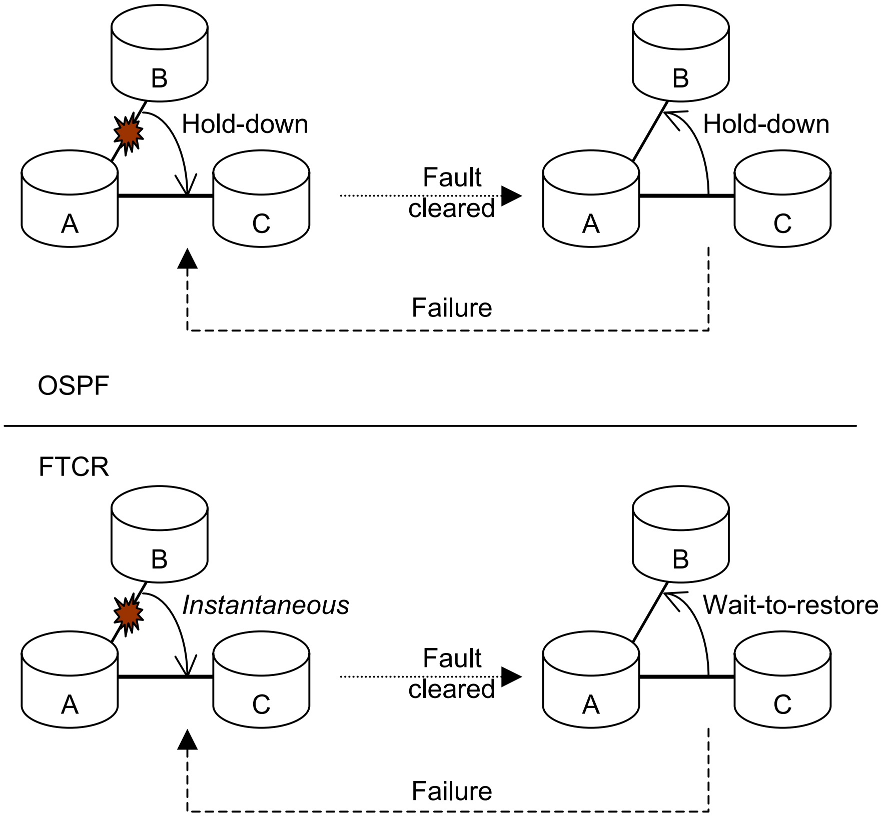


Figure 71: Timing during link up and link down fluctuations in OSPF and FTCR

This is the worst case scenario for OSPF since the link fluctuation is slower than the OSPF delay and the routing tables still fluctuate despite these timers. If the frequency is higher than the hold-down OSPF period then the amount of fluctuations will be limited to maximum once per hold-down period.

Figure 71 illustrates how fluctuations are handled in OSPF and FTCR. In FTCR there is no hold-down timer when recovering the LSPs but there is a wait-to-restore timer during the reversion operations. FTCR waits for OSPF during reversion so the wait-to-restore timer of FTCR is the same as the OSPF hold-down timer.

The frequency of the fluctuations in FTCR is the same as with OSPF. Both OSPF and FTCR recover and reverse back maximum once per period of fluctuation see Figure 72. The important benefit of FTCR over OSPF in this situation is that in FTCR the traffic is recovered immediately after a failure while in OSPF every time the link fails the traffic stays on the affected link for a hold-down period of time.

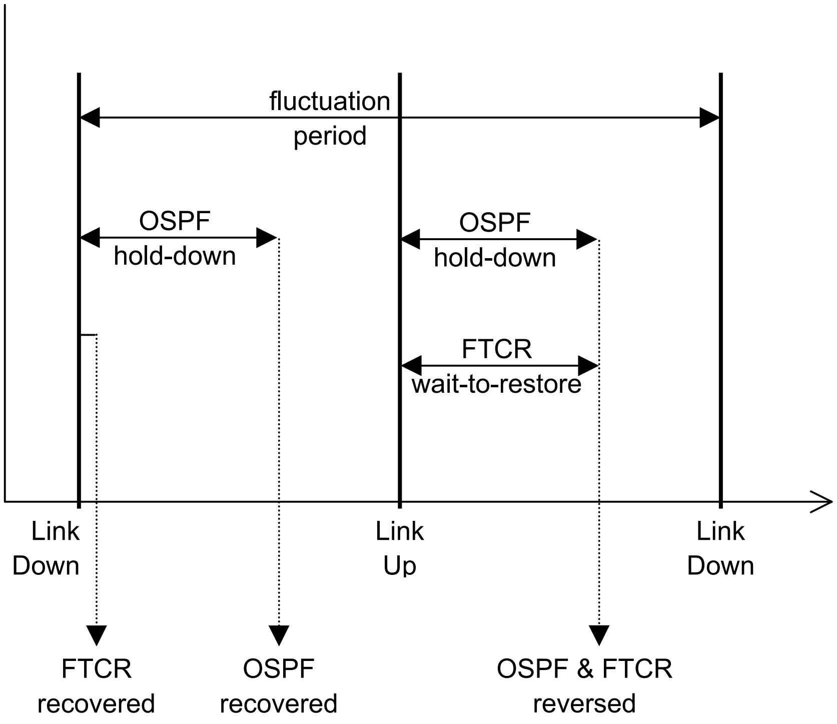


Figure 72: The fluctuation period of FTCR and OSPF are equal. However FTCR reacts faster to a failure thereby reducing the disruption due to lost packets.

Both [139] and [177] propose that OSPF should react fast to bad news (fault) and slow to good news (fault clearance). As we see this behaviour is already part of FTCR but this is not the case for OSPF. For example [184] mentions convergence times of 30s and reversion times of 8s for experiments on the Qwest backbone [185].

An exponential hold-down timer can also be used in OSPF and FTCR to further reduce the fluctuations. In OSPF this decreases the frequency of the fluctuations but also increases the packet loss, in FTCR this reduces the frequency of the fluctuations without increasing the packet loss. We will return to the topic of exponential backoff timers in section 6.3.2.

LSA storms

We have seen that LSA storms can happen in OSPF networks. The easiest way to prevent these (to a certain extent) is to increase the hold-down and retransmission timers. This obviously increases the convergence times of OSPF but when FTCR is used to recover the traffic this does not have a negative effect.

5.4.6.b Comparing FTCR and MPLS rerouting stability

In section 5.4.4 we have seen that triggered MPLS rerouting uses a hold-down timer. The reason for the hold-down timer is to allow the routing tables to stabilise before the recovery LSP is set up. Since FTCR does not use the routing tables to set up the recovery LSPs there is no reason to introduce a hold-down timer like with MPLS rerouting.

An important aspect of FTCR's stability is given by the fact that LSPs are used to reroute the traffic. LSPs are only set up if the path that is calculated by the FSL is actually valid. When we discussed the MPLS rerouting stability we illustrated how the setup of an LSP will fail when the routing tables in the network are invalid. Similarly if the hop specification of the ER-LSP is invalid, the LSP will not be set up either. Note that the FSL will be notified of the failure and that an appropriate action can be taken. Possible actions include the calculating of a different path, retrying to set up the same ER-LSP or to wait for IP convergence and MPLS rerouting the working LSP. Note that this can happen when multiple failure FTCR rerouting is not supported (see section 4.9.5).

Like in MPLS rerouting, loop prevention and loop detection can also increase the stability. Note that loop prevention and loop detection should not be used in conjunction with failure-local repair since loops can occur in that mode (see section 4.4.1 for an example).

5.4.6.c Conclusion

We have seen that eliminating the hold-down time during convergence in FTCR does not lead to instability. The reason for this is two-fold, firstly FTCR does not calculate the recovery paths distributed so implicit co-ordination through timers like in OSPF is not necessary, the second reason is that during reversion there is still a wait-to-restore timer which prevents instability during fluctuations.

We have also seen that during LSP set up the correctness of path is verified (availability of the path and potentially the loop-freeness). This also improves the stability.

When all traffic in the network is protected by FTCR it is possible to increase the OSPF timers without sacrificing the convergence times. This will have a positive influence on the stability of the network.

Now that we have investigated the stability of the convergence schemes we will have a look at the scalability.

5.5 Investigating scalability

In this section we will investigate the scalability of the convergence schemes, more specifically we will investigate scalability with respect to the network size (in terms of links, nodes and connectivity) and with respect to the number of LSPs.

5.5.1 OSPF scalability

The OSPF scalability is limited because the link-state database must be stored and maintained in the network. The link-state database is maintained by flooding link-state advertisements over the network. As the network grows larger the bandwidth used for this flooding will increase and the burden on the OSPF protocol stack to process and forward these packets will increase too. The link-state database needs to be stored in every OSPF router, increasing the network size requires more memory. When two routers bring up an adjacency their link-state database needs to be synchronised. The complexity of this operation again depends on the size of the link-state database. Finally the time required for the SPF calculations also depends on the size of the link-state database. Even in high-end router platforms, the SPF calculation can take a long time in large networks. The complexity of a SPF is, depending on the implementation, O(n.log(n)) or O(n^2) [139]. Note that some implementation are O(n) when the link metrics are limited to 63 [60]. On the other hand, processor speed, memory sizes and link speed increase steadily over the years.

Note that the link-state database keeps track of every route known to OSPF. This includes not only the internal routes but also the external routes (towards other networks). If there is only one default gateway towards the external networks this does not cause a problem because the gateway can be represented by a default route. If however multiple gateways are available and the routes to the individual external networks need to be optimised with respect to these gateways then the external routes (or the most frequently used external routes) need to be injected into the OSPF domain. Care must be taken that not all existing routes are injected since that will introduce too much burden on the OSPF protocol.

The primary scalability feature of OSPF is that the OSPF network can be split up in sub-areas. All the areas are connected to the backbone area (also called the area 0). Each area behaves like an independent network and the shortest paths are only calculated within the individual areas and between areas. This typically leads to less than optimal routing. Some routers, the Area Border Routers (ABR), belong to both an area and the backbone area. The ABR routers maintain a link-state database for each area they belong to. The ABR routers summarise the areas they belong to. These summaries expose the networks in the area but not the internal topology. This form of hierarchical routing increases the maximum size of the OSPF network but it also increases the administrative overhead.

Determining the maximum size of an OSPF network is difficult and depends on the software and the hardware configuration. Concrete numbers in the literature differ but the limit seems to be somewhere between 50 and 1200 routers. For example [53] proposes 70-100 routers per area as a rule of thumb.

5.5.2 MPLS rerouting scalability

MPLS rerouting depends on OSPF so the scalability of OSPF is a first limitation. Another limitation is the number of LSPs that can be supported. With a hard-state signalling protocol like LDP and CR-LDP the number of LSPs that can be supported depends on the size of the LIB and the associated signalling state. Soft-state signalling like RSVP-TE does not scale as well because the state inside the network needs to be refreshed regularly. When the number of LSPs increases the soft-state overhead can consume a significant amount of bandwidth and signalling resources. For example with 10.000 LSPs on a link and the standard refresh period of 30 seconds, this consumes over 600kb per second of link bandwidth [143]. This issue can be addressed by using refresh reduction [54]. Refresh reduction relies on refreshing many LSPs in a single RSVP message. This, together with the ability to indicate that nothing has changed reduces the refresh granularity so that it is closer to a per LSR than to a per LSP level which leads to a significant improvement of the scalability. RSVP-TE scalability tends to be a bit lower than LDP scalability even with refresh reduction although they are in the same order when state reduction is used.

Despite the maximum number of LSPs there is another issue when rerouting LSPs. When a lot of LSPs need to be rerouted this will create a lot of signalling activity. Robust MPLS signalling implementations should be able to handle this although the convergence time may increase as a result of the queuing or even dropping of the signalling messages. RSVP-TE signalling messages are sent over raw IP while LDP sends them over TCP. RSVP-TE messages that are lost will eventually be retransmitted due to the soft-state nature of RSVP but the time before retransmission is much higher than with LDP since it uses TCP.

5.5.3 Protection Switching scalability

The scalability of protection switching mainly depends on the maximum number of LSPs that can be set up simultaneously. This is also true for MPLS rerouting but in PS there are typically more LSPs for a given set of working LSPs. This is because PS is make-before-break scheme where the recovery LSPs are pre-established. Depending on the form of PS and the level of reliability that is required the number of recovery LSPs can be much higher than the number of working LSPs.

The benefit of PS over MPLS rerouting is that a fault that affects many LSPs does not lead to a storm of signalling activity. So the speed of the signalling components in PS is not as critical as in MPLS rerouting. That does not mean that a fault that affects a large number of LSPs cannot stress the PS implementation. When this happens a recovery LSP must be found for every working LSP which might take some time. This obviously depends on the implementation.

5.5.4 FTCR scalability

FTCR requires that a link-state database is maintained in the network. The scalability can therefore be compared to that of OSPF. The scalability of FCTR also depends on the maximum number of LSPs that can be supported and the signalling activity when a large number of LSPs are affected. In that respect the scalability can be compared to that of MPLS rerouting. Note that MPLS rerouting also depends on OSPF so the conclusion is that the FTCR scalability is comparable to that of MPLS rerouting.

5.6 Comparing the backup capacity requirements

In order to have a network with high reliability, spare capacity needs to be present to reroute traffic during fault conditions. In this section we compare the additional cost that is required to reroute traffic after a single failure for each of the convergence schemes covered above.

5.6.1 Simulation model

We assume a network model where all links are bi-directional but the capacity and the link weight can be asymmetric. The demand for a path in the network can be asymmetric too. We assume that link capacities can take any positive value despite discrete link capacities in reality. A discrete capacity model is not necessary to assess the capacity requirements of the different recovery models. Moreover using a discrete capacity model would obscure the comparison. We also assume a linear cost model i.e. the cost of the link is proportional to the link capacity and link length. The total cost of the capacitated network is the sum of the link capacity costs.

The investigated convergence schemes are rerouting, local protection switching and FTCR in failure-local repair mode. There is no reason to discriminate between OSPF rerouting and RSVP-TE soft-state or triggered rerouting because the topological rerouting and hence the capacity requirements are the same. The differences between these three schemes are the technology and the convergence speed but these topics were already covered in the previous sections so there is no reason to discriminate between them in this section.

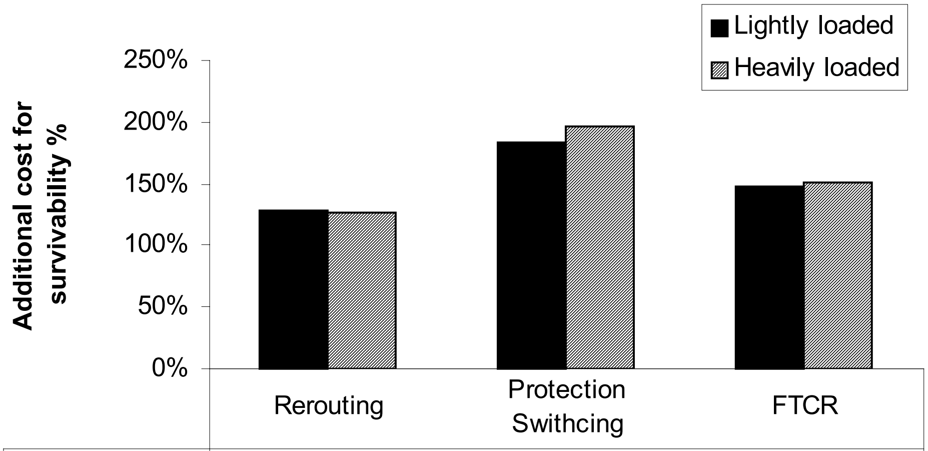
The overall objective of the study is to dimension the capacity of the network so that during normal operation all demands should be fulfilled and that in case of a single link or node failure all affected traffic (except traffic entering or leaving the network via a failing node) can be rerouted by the convergence scheme. The routing of the traffic that is not affected by the failure remains unaltered. We route the traffic during the working conditions according to the shortest path.

We consider both link and node failures and we assume that the type of failure is known. We compare the capacity requirements of the three convergence schemes on a realistic network. The study is based on the e.spire network [55] the topology consists of 44 nodes and 57 links, the link weights are assigned roughly proportional to the estimated distance.

The demand matrices were generated randomly. In order to make larger cities more important each node was assigned a weight. The demand between two nodes depends on the weight of the nodes. Nodes that are part of large city infrastructure have a larger weight leading to a larger demand to and from them. By rounding these random values to integer numbers, a light load will result in sparser demand matrices (i.e., more zero-elements). The demand is routed according to the shortest path over the network. We give the results as the average for ten random demand matrices. More results where published in [4, 5, 6, 55].

5.6.2 Simulation results

Figure 73 shows that the cost of a network with spare capacity is more than doubled compared to a network without any extra spare capacity (the additional cost is more than 100%). Protection switching is the most expensive mechanism and rerouting is the cheapest one with FTCR somewhere in between. This can intuitively be explained as follows. Protection switching has to reroute affected traffic locally, which is obviously worse than rerouting which has global scope and thus potentially spreads out the rerouted traffic over the network. FTCR has upstream the local nature of protection switching but downstream the global nature of rerouting giving it the capacity requirements between the two. Note that if ingress repair FTCR is used then the capacity requirements are similar to those of rerouting because of the global scope.



	Rerouting	Protection Swithcing	FTCR
■ Lightly loaded	128%	184%	148%
▨ Heavily loaded	126%	196%	152%

Figure 73: Additional cost of a reliable network compared to an unprotected network (%) in function of the traffic load.

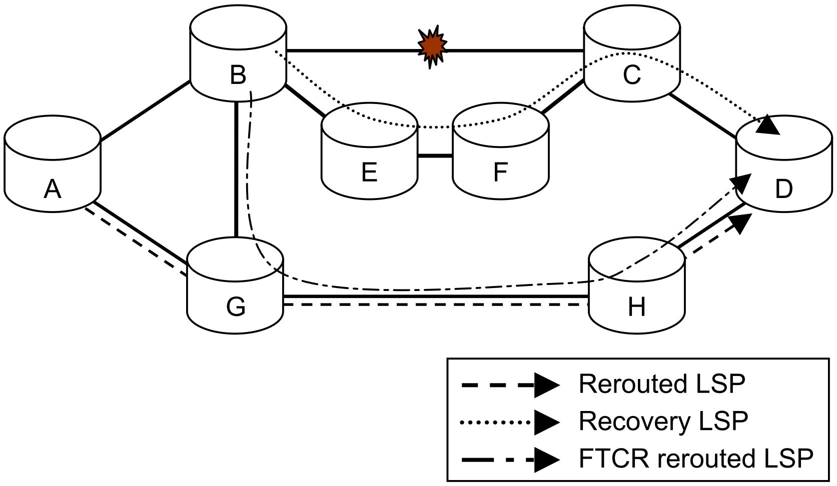
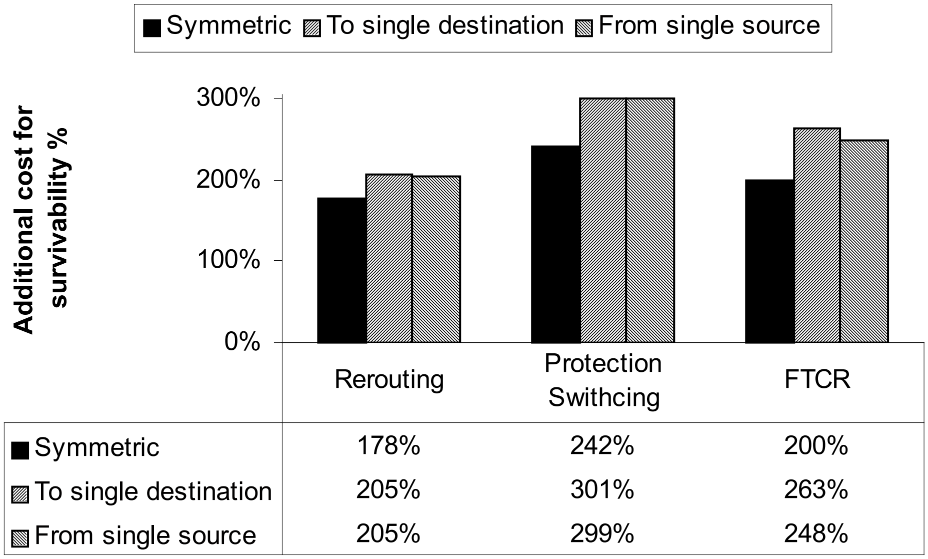


Figure 74: The different paths of the rerouted LSP, the local protection recovery LSP and the FTCR rerouted LSP when a failure happens on link BC of the shortest path from A to D (A-B-C-D).

Figure 74 illustrates how the shortest path LSP from A to E is repaired over different paths depending on the recovery mechanism. Rerouting is able to use the most optimal recovery path from node A. FTCR has a slightly less optimal path from the FSL (node B) towards the destination E. Protection Switching uses a less optimal recovery path because it is routed back to the working path.

Figure 73 also shows the effect of the load (given by the demand matrix) on the extra capacity requirement. We see a slight increase in relative additional cost for survivability. It is important to note that the order of the different convergence schemes remains the same. We should be careful however in drawing general conclusions from this particular case study. Additional simulations [55] show that varying the topology (large and small e.spire network and a large and small Qwest network) does not change these conclusions. Moreover using random topologies and varying the link weights and varying the link density (i.e. the probability that a link exists between two nodes) yields the same results.



	Rerouting	Protection Swithcing	FTCR
■ Symmetric	178%	242%	200%
▨ To single destination	205%	301%	263%
▧ From single source	205%	299%	248%

Figure 75: Additional cost of a reliable network compared to an unprotected network (%) in function of the traffic pattern.

Finally, we investigated the influence of highly asymmetric traffic that terminates or originates in one node (**Error! Reference source not found.**). Still the same conclusions are valid but we observe that (i) the additional cost for survivability is lower for symmetrical traffic and (ii) FTCR is significantly cheaper when traffic is originated from a single source rather than terminates in a single destination. Rerouting and protection switching do not have this property and have almost the same cost for traffic terminating or originating in a single node. This can be explained by FTCR's asymmetric nature while rerouting and protection switching are symmetrical techniques. FTCR has a significantly higher cost when all traffic is terminated in a single node because the FSL reroutes all traffic over a single path leading to a highly concentrated traffic pattern downstream of this node.

In these simulations we considered the extra capacity that is required to reroute traffic during fault conditions. Even protecting against single faults requires significant extra capacity. Protection switching requires the most extra capacity, followed by FTCR and by rerouting that is the most cost effective.

5.7 Conclusions

The chapter started out by describing the experimentation platform and experiments performed therein. The main goal of these experiments is twofold: first of all to prove that FTCR as a proof-of-concept implementation is able to recover traffic during failure conditions and secondly to get a more precise view on the convergence times of the convergence schemes.

As we have seen in the experiments FTCR is indeed able to recover working traffic. Moreover FTCR is faster than MPLS rerouting and OSPF routing. Soft-state rerouting is very slow and should be avoided if possible. Local protection switching proved to be the fastest convergence scheme. MPLS rerouting is slightly slower than OSPF rerouting because a hold-down timer is used to make sure that the routing tables are stable before the recovery LSP is set up. Failure detection with the Hello protocol, especially with the standard timers, is very slow and hardware-based failure detection (here simulated through immediate failure detection) should be used if possible.

Experiments in a test networks of limited size, like those that we conducted, have the intrinsic drawback that the conclusions are only valid within the limits of the test network. The most important limitations are the limited size and the fact that the network links are lightly loaded and have a small delay. Also the number of routes and LSPs is very small in comparison to large operational networks. Therefore we investigated the stability and the scalability of the convergence schemes too.

OSPF deployments are limited in size because the flooding, the size of the link-state database, the duration of the SPF calculations and the size of the routing tables is directly dependant on the number of routes in the network. Care should be taken that the network is not too large and that not too many routes are injected in the routing tables. The primary scalability feature of OSPF is that it is possible to split up the OSPF domain in multiple areas.

Since FTCR does not use a hold-down timer it is important to explain that this does not jeopardise the stability. We have seen that during fluctuations FTCR has the same fluctuation frequency as OSPF. FTCR also has the good property that it reacts fast to failure events and much slower to failure clearance events, unlike OSPF which reacts symmetrically to failure and failure clearance events. The stability of FTCR is also attributed to the fact that a single FSL recovers traffic without co-ordination and by using LSPs. MPLS rerouting is for similar reasons more stable than OSPF. Protection switching is very stable since its recovery actions are limited to switch over actions to pre-established recovery LSPs. It does require that the failure detection is somewhat stable and it is also recommended that a wait-to-restore timer is used before switching back from the recovery LSP to the working LSP.

Since MPLS rerouting depends on IP routing, its scalability depends on the routing protocol. Furthermore the scalability of MPLS rerouting also depends on the number of LSPs that can be supported simultaneously and the speed of the MPLS signalling daemons. FTCR depends for its scalability on the link-state routing protocol and on the scalability of the MPLS signalling daemon. Finally, Protection switching primarily depends on the number of LSP that can be supported simultaneously. The speed of the MPLS signalling daemon is less important since the recovery LSPs are pre-established.

We also investigated the backup requirements. IP rerouting needs the least backup requirements, followed by failure-local FTCR and local protection switching. Rerouting needs the least backup capacity because it has global scope and thus can potentially spread out the rerouted traffic over the network. Protection switching uses most backup capacity because it has to reroute affected traffic locally. Failure-local FTCR has upstream the local nature of protection switching but downstream the global nature of rerouting so its capacity requirements reside between the two. FTCR in ingress repair mode has the same capacity requirements as rerouting.

Figure 76 summarises the properties of the convergence schemes with respect to the criteria covered in this chapter. Protection switching has the fastest convergence times, followed by FTCR which is also very fast especially in small networks. IP routing is significantly slower. MPLS rerouting is even more slower especially when soft-state rerouting is used.

The stability of IP routing can be affected by LSA storms, routing anomalies and symmetric hold-down timers. MPLS rerouting is more stable since it uses LSPs to recover the traffic and because an additional hold-down timer is used. FTCR is even more stable because an asymmetric hold-down timer is used, because only one FSL is responsible for a given failure and because LSPs are used to recover traffic. Finally protection switching is also stable because the recovery operations are very simple.

	IP rerouting	MPLS rerouting	FTCR	PS
Convergence time	–	–(–)	+(+)	++
Stability	–	0	+	+
Scalability	++	+	+	–
Capacity requirements	++	++	+(+)	–

Figure 76: Comparison of the different convergence schemes with respect to the criteria covered in this chapter.

IP routing is the most scalable technique because it is only limited by the size of the link state database. The scalability of MPLS rerouting is more limited because it depends not only on the size of the link-state database but also on the number of LSPs and the signalling activity during recovery. The same reasoning applies to FTCR. The scalability of protection switching is limited due to the fact that protecting against all possible failure scenarios requires an exponential number of LSPs. Its scalability can be considered good if, for example, only single failures need to be protected against.

Finally the backup requirements depend on the scope of the recovery operations. Therefor IP routing has the best backup requirements together with MPLS rerouting. Immediately followed by FTCR in ingress repair mode. Failure-local FTCR requires more backup resources and finally protection switching requires the most resources.

In the final chapter of this work we will look at recent advances in failure convergence in large networks. We will mainly focus on link-state routing and FTCR.

Chapter 6

Improving convergence schemes

6.1 Introduction

In this chapter we will look at techniques that can be used to speed up the time of convergence especially in large networks. An important part of this chapter describes the recent advances in link-state routing. We will explain these techniques and how they speed up link-state routing. In the section following that, we look at how these techniques can be used in combination with FTCR. We will also look at the final limitations for fast convergence in large networks and how FTCR can be used to further speed up the convergence in these networks. This section obviously ends by drawing conclusions. We will start this section with an explanation of the increased importance of the failure detection times.

6.2 Failure detection

Fast protection switching requires fast failure detection because the recovery operations of protection switching are very short. If the failure detection were slow the real advantage of protection switching i.e. fast convergence is diminished because of the slow failure detection. Obviously the fastest failure detection is hardware-based failure detection. A router platform that supports MPLS protection switching typically also supports hardware-based failure detection. However that does not mean that the support for hardware-based failure detection should be limited to protection switching only. For example link-state routing can also improve its convergence times by using hardware-based failure detection. The Hello protocol will then still be used, for example to detect control plane failures and to elect the designated routers. However the Hello Interval and Router Dead timers can somewhat be relaxed since most of the failures can be detected through the hardware-based failure detection (we mentioned this before in section 3.2.1.c)

6.3 Recent advances in link-state routing

In this section we will cover the more recent advances in link-state routing, in the next section we will look at how these techniques relate to FTCR.

We start with a brief history of link-state routing implementations [56]. The first phase, which started around 1990, was really focussed on getting the implementation to work correctly. During the second phase, which started around 1994, there was some interest in speeding up convergence. However convergence speed was more seen as a marketing advantage than as an important field of research. The speed improvements were typically gained by sacrificing stability. From 1995 on, IP networks began to get very large and mission critical. Network outages for prolonged periods started to be very disastrous so stability was the prime concern. Stability was achieved through robust software and hardware implementations and by using large timers. This was obviously detrimental for the convergence speed. More recently there is a renewed interest in improving the convergence times of link-state routing. The driver is that interactive applications like VoIP [186, 187] really demand fast convergence times in order not to disrupt the service during fault conditions. Another driver is that some technologies like MPLS protection switching are actively being promoted by claiming that IP convergence is too slow.

In this section we will cover some techniques that are used to speed up IP convergence with link-state routing. We start by investigating what can be improved on the shortest path calculations and the resulting routing table updates.

6.3.1 SPF optimisations

We have seen that the shortest path calculations can be both disruptive and can take up a significant amount of time. This can be addressed by using an algorithm that scales much better than Dijkstra. The Dijkstra algorithm re-computes all routes every time the topology changes. However one can expect that, especially in large networks where the SPF calculation is critical, that not all the routes are affected by the topology change. This can easily be illustrated with an example (see Figure 77). When for example link CE breaks the routes between node A and B, E, F, G and D do not change. The Dijkstra algorithm does not take this into account and simply recalculates every route in the network.

More recent algorithms [57, 58, 59], Incremental SPF algorithms (I-SPF), reuse the data structures from earlier calculations and only re-compute the affected routes. Their average case complexity is O(log(n)) compared to O(n.log(n)) or O(n^2) for the Dijkstra algorithm. Reference [139] reports speed improvements of up to 10.000 times. However the gain in computation time depends on the location of the failure. When a failure occurs in the centre of a network a lot of shortest paths will be affected and the shortest path computations will still require of lot of processing.

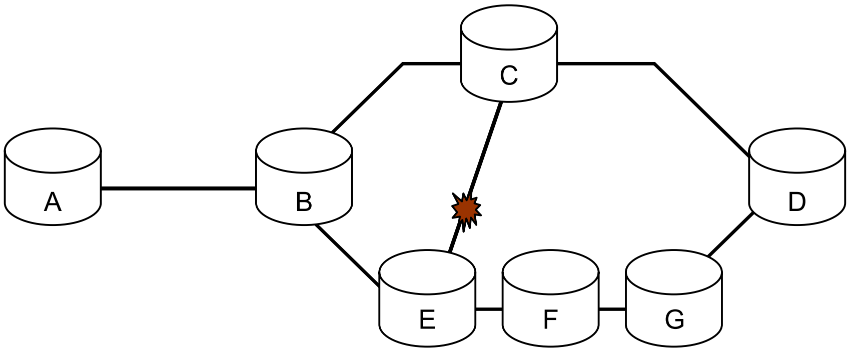


Figure 77: The Dijkstra algorithm recalculates all the routes when the topology changes. This can be very inefficient. For example when link CE breaks none of the routes in node A change and the SPF calculation in node A by the Dijkstra algorithm is superfluous.

Another example of the inefficiency of running Dijkstra at every topology change is when a new leaf node is added to the network. For example consider that a new node, node H, is added to the network by connecting it to node A. The addition of this node will not alter the shortest path routing so it is sufficient to add a routing table entry for node H. This process is called Partial Route Computation (PRC) [177].

6.3.2 Exponential backoff

We already mentioned that exponential timers can help to increase the stability. For example when link fluctuations occur it is important not to calculate the shortest paths too frequently since this will lead to unstable behaviour. Increasing the hold-down timers during frequent calculation will lead to a decrease of the SPF calculation frequency.

Exponential timers can also be applied to decrease the convergence time. By making the hold-down timers adaptive it is possible to decrease the initial hold-down time without jeopardising the stability. With static hold-down timers there is a trade-off between stability and convergence time. With exponential timers the length of the timers increases fast enough so that stability is retained during periods of fluctuations.

Exponential timers are not only applicable to the hold-down before SPF calculations, they can also be applied to PRC and LSA advertisement delays. Exponential backoff timers applied to LSA advertisements are especially good to prevent LSA storms.

6.3.3 Improved convergence times

By using exponential backoff timers, incremental SPF and partial route computation it is possible to achieve sub-second convergence with link-state routing protocols even in large networks [177]. Sub-100 millisecond convergence times in large networks are currently not achievable with link-state routing.

This is because the shortest path computation cost, even with incremental SPF, can be too high. Another reason that sub-100 millisecond convergence times are not achievable is that the flooding delay is too high. In link-state routing the convergence depends on the flooding of the updated link-state advertisements after the failure is detected. When the network is very large the delay introduced by the processing and the transmission of the LSAs can be quite high. The first reason is that the distance that the LSA needs to cross is high. Packets cannot travel faster than the speed of light, in a trans-European or US backbones, the transfer delay only can be in the order of tens of milliseconds. Moreover every LSA needs to be processed by every router to ensure proper flooding (the packets need to be forwarded over the proper interfaces, packets may not live forever) this introduces additional latency. For example in Figure 78, we consider that the shortest path between node A and E crosses the nodes B_1 ... B_n. When link B_nE fails the link-state advertisements from B_n or E need to reach node A before it can recompute the new routing tables. When we consider that the distance or the number of hops between B_1 and B_n (or D_1 and D_n) is very high, the time required for these LSAs to reach A will also be very high.

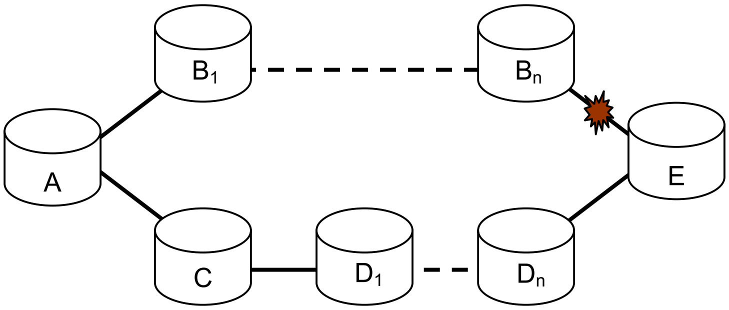


Figure 78: Transmission and process delays of link-state advertisements can be very high in large networks. In this example we consider that there are a large number of nodes (n-2) between B_1 and B_n and D_1 and D_n.

The fact that the shortest paths need to be calculated and that the delay on the LSAs can be very high prohibits sub-100 millisecond convergence times in large networks. Very fast convergence times in large networks is only possible if the recovery actions are based on local actions on pre-computed and pre-established paths. MPLS local protection switching is such a technique, as we have seen there is no failure indication delay and since the paths are pre-computed and pre-established the recovery operations are performed very fast. Local protection switching is believed to offer convergence times in the order of tens of milliseconds [188].

6.4 Improving FTCR

In the previous section we have discussed techniques to speed up the convergence of OSPF. In this section we will investigate if these techniques are also applicable to FTCR. We will also propose other mechanisms to speed up FTCR.

6.4.1 Exponential backoff timers

Using exponential backoff timers in link-state routing protocols makes it possible to set the initial values of the timers to much lower values which typically translates into faster convergence times without jeopardising the stability.

As we have seen, FTCR does not use the SPF hold-down timers of the link-state routing protocol during convergence so FTCR does not benefit from exponential backoff on the SPF and PRC delay timers during convergence.

During reversion FTCR does rely on OSPF so exponential backoff timers can speed up the reversion of FTCR as it does for link-state routing. However reversion is not as time critical as convergence so the gain is considered small. Moreover care must be taken that the wait-to-restore is not too fast in order not to disrupt the traffic that has been recovered.

Exponential backoff timers can also be applied to link-state advertisements. Since FTCR does not use link-state advertisements, FTCR does not directly benefit from exponential backoff applied to them. However when the link-state advertisements are spread faster over the network this can lead to less multiple failure events.

6.4.2 Incremental SPF

Incremental SPF is a technique which speeds up the shortest path calculations. Since FTCR uses the same link-state database and the same shortest path algorithm as link-state routing, incremental SPF is equally useful for FTCR to calculate the shortest path tree in the FSL.

6.4.3 Propagation delays in large networks

As we have seen in section 6.3.3 sub-second convergence times with link-state routing are achievable even in large networks. However faster convergence is difficult to achieve because of the (potentially) large shortest path computation times and the transmission and process delay of the LSAs. In Figure 78 we have seen that receiving the LSAs takes up a lot of time. A similar reasoning is true for FTCR. We will illustrate this with the different FSL selection modes of FTCR.

In ingress repair mode the FIS from B_n needs to reach node A before it can set up the recovery LSP. Sending the FIS from B_n to node A is believed to be slightly faster than flooding the LSA because there is less processing overhead involved in sending a FIS compared to flooding an LSA. The FIS only needs to be forwarded towards the destination while flooding an LSA requires several checks to prevent the LSAs from living forever. However FTCR does have a significant drawback when compared to link-state routing in this respect. When node A receives the FIS it will set the recovery LSP. This means that a label request message is sent from node A to node E traversing all the nodes in between. In turn node E will send a label mapping towards node A. This means that in order to set the recovery LSP the network is almost traversed three times: once for the FIS and two times to set up the LSP.

In failure-local repair node B_n is the FSL. As always in failure-local repair there is no notification time. However node B_n will set up the recovery LSP, which will lead to a set up delay that is in the order of four times the delay between node A and node E. The delay of the recovery LSP will also be significantly higher than on the working LSP, almost two times higher. Note that the topology of Figure 78 triggers such bad behaviour, a more meshed topology renders much better results. For example if a link exists between B_n and D_n then the set up time of the recovery LSP will be very short and the additional latency on the recovery LSP will be negligible.

Nearest non-looping repair does not perform very well either in the topology of Figure 78. When node B_n detects that it cannot set up the recovery LSP it will send a FIS upstream. Depending on the implementation, the FIS will be sent to the previous hop or directly towards node A. In any case, the performance will be at best that of ingress repair. Like with failure-local repair adding the link B_nD_n solves these problems.

We have seen that using LSPs to recover traffic is more stable than updating routing tables distributed and uncoordinated. However when the delays in the network are high the two-way set up procedure of downstream-on-demand label distribution (see section 2.3) of CR-LDP or RSVP-TE introduces a high cost. On the other hand, reference [184] shows that routing loops can exist for periods as long as 1.5s in a network where the LSA propagation delays are about 300ms.

Note that in both link-state routing and FTCR the delays can be improved by giving the control packets preference over data packets. For example by using DiffServ and marking them with EF [96, 176].

Nevertheless, it is clear that sub-100 millisecond convergence times in very large networks are not achievable by both link-state routing and FTCR. The remaining time consuming phases in FTCR convergence are the computation and the setup of the recovery LSPs. They both can be addressed within the framework of FTCR, this will be discussed in the next section.

6.4.4 Pre-computation and pre-establishing recovery LSPs

In this section we will discuss how the convergence times of FTCR can be improved by pre-computing and pre-establishing the recovery LSPs. Obviously when the recovery LSPs are pre-established they need to be pre-computed too. However when the recovery LSPs are pre-computed that does not mean that they should be pre-established. We start by explaining how the recovery LSP can be pre-computed.

Calculating the recovery LSPs with FTCR before a failure happens can be considered as a *calculate-before-break* scheme. The calculations should be performed in the background and should not interfere with any operational work that needs to be performed by the router.

In FTCR with failure-local FSL selection mode, it is quite easy to calculate all possible recovery LSPs for the given set of working LSPs at a certain moment. For every working LSP a recovery LSP is calculated by removing the outgoing interface from the link-state database and by calculating a new downstream path for the LSP. This is possible because there is always only one FSL for a given LSP and a given failure. In ingress repair mode, a LSR needs to calculate a recovery LSP for each possible failure on each LSP that it has set up. This will result in the same number of calculations but the calculations will be concentrated in the ingress of the LSPs (typically the edge nodes). Nearest non-looping repair is more difficult to support because it is difficult for a node to determine for which LSPs and associated failures it can be the nearest non-looping FSL.

When a failure is detected it is easy to look up the correct recovery paths for all affected working LSPs if the pre-computed recovery LSPs are stored per anticipated fault. When a failure happens that does not affect any of the working LSPs of a given router then obviously no recovery actions need to be performed.

However when the link-state advertisements arrive at that router they will be incorporated into the link-state database. As a result the pre-computed recovery LSPs can be invalid. A first approach to resolve this issue is to discard all the pre-computed recovery LSPs. A more refined approach is to check which recovery LSPs are affected and only discard those pre-computed paths that are affected. A similar algorithm to that of incremental SPF can be used.

Finally when a failure occurs for which a given router is not the FSL and immediately later a failure happens for which the router is the FSL, the link-state database and hence the pre-computed recovery LSPs will not reflect the recent changes. This is solved with FTCR multiple failure rerouting as described before (see section 4.6). As a matter of fact this is not specific for FTCR in calculate-before-break mode.

One could take FTCR calculate-before-break one step further and not only calculate the recovery LSPs beforehand but also establish them. This is called FTCR *make-before-break*. When the recovery LSPs are pre-established this will obviously lead to faster convergence times then either FTCR rerouting or FTCR calculate-before-break. However depending on the size of the network and the scalability of MPLS components it may not be possible to set up all possible recovery LSPs. Also it is very important that the recovery LSPs are merged as much as possible and that double booking is prevented (see section 3.3.4.e and section 3.3.4.d respectively). It is recommended that the recovery LSPs are in turn not protected because this would lead to an exponential number of recovery LSPs. Note that it can be signalled that an LSP should not be protected (e.g. by clearing the *local protection desired flag* used by RSVP-TE).

FTCR with make-before-break resembles protection switching fairly well. Especially protection switching where the recovery LSP are calculated on-line (see section 3.3.4.c). Therefore it is important to point out the differences between them.

Protection switching with online path calculation calculates the recovery paths during the working path setup. FTCR starts to calculate the recovery in the background after the working LSPs are set up. In FTCR the working LSP will be set up before the recovery LSPs while in protection switching the working LSPs and the recovery LSPs will be set up intermixed.

A more important difference is the support for multiple failures. When a failure happens in FTCR make-before-break, the traffic is restored by using the pre-established recovery LSP. When another failure occurs on the path of the recovery LSP immediately thereafter, the second failure will be handled by *rerouting* the recovery LSP. The convergence will be slower than recovering a single failure with FTCR make-before-break but it will be faster than MPLS rerouting. With protection switching it is very difficult to support multiple failures since this would lead to an exponential number of recovery LSPs. To support multiple failures the working LSPs need to be protected but also the recovery LSPs themselves. Note that we do not consider two failures that occur on the working LSP while the recovery LSP remains unaffected as a multiple failure event (see section 3.3.1.d).

As we mentioned before it might not be possible or desirable to support all the working LSPs with make-before-break FTCR. Note that in this case protecting all the working LSPs is also not possible or desirable.

6.5 Conclusion

We started by pointing out that hardware-based failure detection is very important for fast convergence. Afterwards we investigated new advancements in link-state routing. Recently there has been a renewed interest in speeding up link-state routing convergence due to the increased importance of interactive services like VoIP and the increasing interest in MPLS protection switching. Speeding up link-state routing can be achieved by making the SPF calculations more efficient by introducing incremental SPF and by avoiding unnecessary SPF calculations when only leaf changes happen in the network through PRC. By using exponential backoff timers on hold-down timers for SPF and PRC and LSA updates it is possible to make the minimum timers much lower which results in faster convergence without affecting the stability. With these measures sub-second convergence times in large networks can be achieved. Lower convergence times are only possible with local recovery operations on pre-calculated paths.

FTCR can also use incremental-SPF to reduce the SPF tree computation which is used to determine the recovery LSPs. Faster LSA propagation times will lead to less multiple failure events because the link-state databases are updated much faster. With these measures the convergence times of FTCR and link-state routing are comparable. However FTCR uses a downstream-on-demand label distribution to set up the recovery LSPs and this can introduce additional latency in large networks compared to link-state routing. On the other hand with link-state routing, routing anomalies can occur which increases the convergence time too.

As we have seen it is only possible to achieve the lowest convergence times in large networks when the recovery actions are based on pre-computed and pre-established recovery paths. FTCR can be extended to support both calculate-before-break and make-before-break. FTCR has then more flexible support for multiple failures compared to protection switching.

Chapter 7

Concluding remarks

We have seen that MPLS has a number of important applications. Traffic engineering, VPNs and protection switching are probably the most prominent of these. With the exception of protection switching these techniques can also be offered in a pure IP network. Despite this fact MPLS has achieved both a significant level of support from equipment vendors and a number of significant deployments.

MPLS offers a path-oriented entity, the LSP, to IP networks. This makes MPLS a favoured technology for some network operators because it brings the well-known "connection-oriented" paradigm to IP networks. At the same time MPLS also causes serious resistance by another group of people, typically network research people with a background in IP. They argue that MPLS destroys the connection-less nature of IP networks. We do not take a standpoint in this discussion but it does make discussing or presenting a MPLS related topic at a conference an interesting experience.

The main subject of this work is network resilience and not MPLS. We discussed several convergence schemes for IP and MPLS. The convergence can be split up in two groups: the first group, the routing schemes, determine and perform the recovery actions at the moment a failure occurs. The other group of convergence schemes uses a make-before-break approach where recovery LSPs are set up before a failure occurs. This group of convergence schemes is called protection switching. A lot of research has been conducted to develop protection switching techniques that take the specifics of MPLS into account. For example the number of recovery LSPs in protection switching can be reduced by merging different recovery LSPs or by using label stacking. This in contrast with MPLS rerouting where little advantage is taken of MPLS specific functionality. FTCR on the other hand does take into account the new functionality that MPLS has to offer.

FTCR is a convergence technique that really takes advantage of the path oriented functionality of MPLS in IP networks. As we have seen MPLS rerouting simply sets up LSPs according to the routing tables in the network. It thereby mimics IP routing by using the LIBs to forward traffic instead of the RIBs that are used in IP forwarding. FTCR really takes advantage of MPLS by setting up the recovery LSPs before IP routing converges.

FTCR is also able to offer a high degree of flexibility because it allows for different FSL selection modes (ingress, nearest non-looping and failure-local repair) and different failure presume modes (presume link failure, presume node failure, presume failure type known). The algorithm that is used to recover ER-LSPs can also be chosen. We have given four different approaches but there are certainly other possibilities too. Perhaps the most important source of flexibility of FTCR is the fact that it can support rerouting where the paths are set up and established on demand but it can also support pre-established and pre-computed recovery LSPs. Therefor FTCR can be seen as a technique that combines the best properties of rerouting and protection switching in one convergence scheme.

One can envisage that these different levels of convergence can be used in networks with service differentiation like DiffServ networks. The more premium services can then be protected by FTCR make-before-break or calculate-before break while the best-effort services can be protected with FTCR rerouting.

Appendix A

Testbed configuration parameters

This small appendix gives the exact hardware and software configuration used to conduct the experiments. While this information will not concern most readers the information is provided so that the experiments can be verified independently.

Hardware

PC routers: AMD K6 550Mhz CPUs with 64Mb, 100Mbit Ethernet adapters
from VIA Technologies (VT3043, Rhine chipset revision 6)

Smartbits 2000 with ML-7710 100/10Mbit Ethernet interfaces

Software

Linux kernel 2.4.19 with MPLS-Linux version 1.172

RSVP-TE daemon 0.70-rc2 modified to support FTCR

OSPF with immediate failure detection Zebra version 0.93b

OSPF failure detection with Hello protocol Zebra version 0.92a

Layer 2 forwarding parameters

Static ARP addresses

Route cache flush delay 0s (/proc/sys/net/ipv4/route/min_delay = 0)

Appendix B

RSVP-TE daemon for DiffServ over MPLS under Linux

B.1 Introduction

The RSVP-TE daemon for DiffServ over MPLS under Linux project supports the important Internet Engineering Task Force (IETF) standards for the set up of MultiProtocol Label Switching [23] tunnels with DiffServ [91] support under Linux by using the ReSource reserVation Protocol [30]. These tunnels support scalable Quality of Service (QoS) in IP networks. While the project might be very specialised for the typical Linux user, it is used by ISPs, network operators and research institutes all over the world. For some of them this project is a "killer application" making them look at Linux for the first time.

The project reuses the code of a few existing projects (some of them abandoned) and created a useable RSVP MPLS daemon for Linux. The project deliberately uses an open, bazaar model with frequent releases to leverage on the ever-growing user base.

This appendix will elaborate on the architecture and used components both in user and kernel space (netfilter, netlink, scheduling and queuing, MPLS).

The appendix concludes by investigating the pros and cons of open sourcing a research project like this that is traditionally developed in house or in a closed group. The comparison is based on a use case: the public demonstration of MPLS technology as proof-of-concept. We compare the closed model used in the Ithaci project with the open source project code that was used in the Tequila demo. The author was responsible for the MPLS signalling software in both demos.

The first section describes the features of the daemon, how the daemon was merged from different projects and the resulting architecture. The section ends with a description of how a packet is forwarded through the forwarding plane.

The RSVP-TE daemon architecture

First of all the daemon supports the set up of Label Switched Paths (LSPs) in the network according to the IP routing tables or by explicitly specifying the hops to be traversed (shortest path LSPs and Explicitly Routed-LSPs). There is an RSVP API (RAPI), an API to aid developers to build custom applications that interact with the RSVP daemon. There is support for DiffServ allowing to differentiate the forwarding behaviour based on the value in the EXP field of the MPLS header. There is also support for IntServ where resources are explicitly allocated on a per-LSP level. Traffic can be very flexibly mapped on the LSPs based on the destination address, protocol, destination ports and port ranges of the IP packets. There is also the ability to trace an LSP, comparable to IP's traceroute. It checks the route taken by an LSP by probing the routers along the path. Resilience is also supported with LSP rerouting and LSP protection switching.

Merged components

The daemon is based upon the Nistswitch version 2.0 daemon for Free BSD by USC (Figure 79: RSVP-MPLS BSD [164]) and a port of an IntServ RSVP daemon to Linux by Alexey Kuznetsov (Figure 79: RSVP Linux [163]). Both daemons are based on the same code base (ISI RSVP implementation, Figure 79: RSVP BSD [162]) but they forked a while ago. This effort combines the daemons again so that the MPLS support found in the Nistswitch version is now available on Linux. Moreover support for DiffServ over MPLS (DS/MPLS) is also added (Figure 79: DiffServ extension).

The MPLS Linux kernel code is based upon mpls-linux by James R. Leu (Figure 79: MPLS Linux). Our release originally added kernel support for DiffServ over MPLS support, the use of multiple routing tables and LSP byte and packet counters. However the more recent version 1.1 branch adds this functionality so we are using an unpatched version now.

Figure 79 illustrates the merge process. We started by extracting the MPLS extensions from the Nistswitch version and applying these extensions to Alexey's Linux RSVP daemon. We then added support to daemon for James Lieu's Linux MPLS patches. Finally we added our own support for DiffServ over MPLS and flexible traffic mappings.

Other small patches are required to iptables (DSCP based matching) and tc (support for the MPLS protocol).

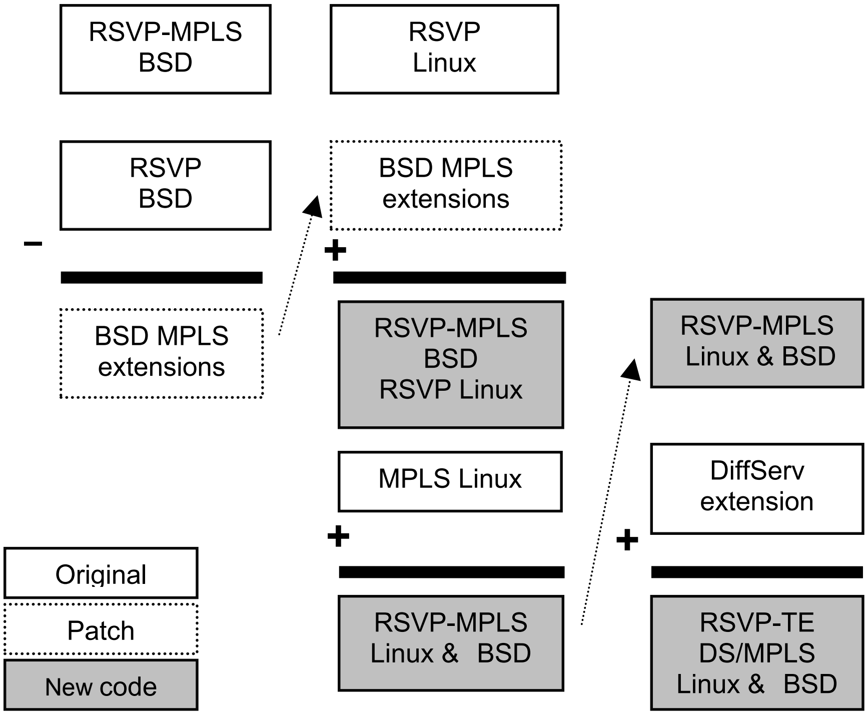


Figure 79: Software Merge Map of the RSVP-TE daemon. The BSD MPLS extension patch is the result of determining the difference of the source code of the original BSD RSVP daemons (-) this patch can be applied to Linux RSVP daemon because it is based on the same source code base. Adding MPLS and DiffServ functionality results in a RSVP-TE daemon for Linux (and BSD).

The overall architecture

The overall architecture (Figure 80) depends on a number of components both in user space and kernel space. The important parts of the kernel that are used are netfiler to classify the packets, QoS and fair queuing to support differentiating between flows and of course MPLS support.

The prime user space component is the RSVP daemon that is responsible for the RSVP signalling and the maintenance of the MPLS state. The daemon is responsible for the allocating and installation of the MPLS labels during LSP set up and freeing and removing labels on LSP tear down.

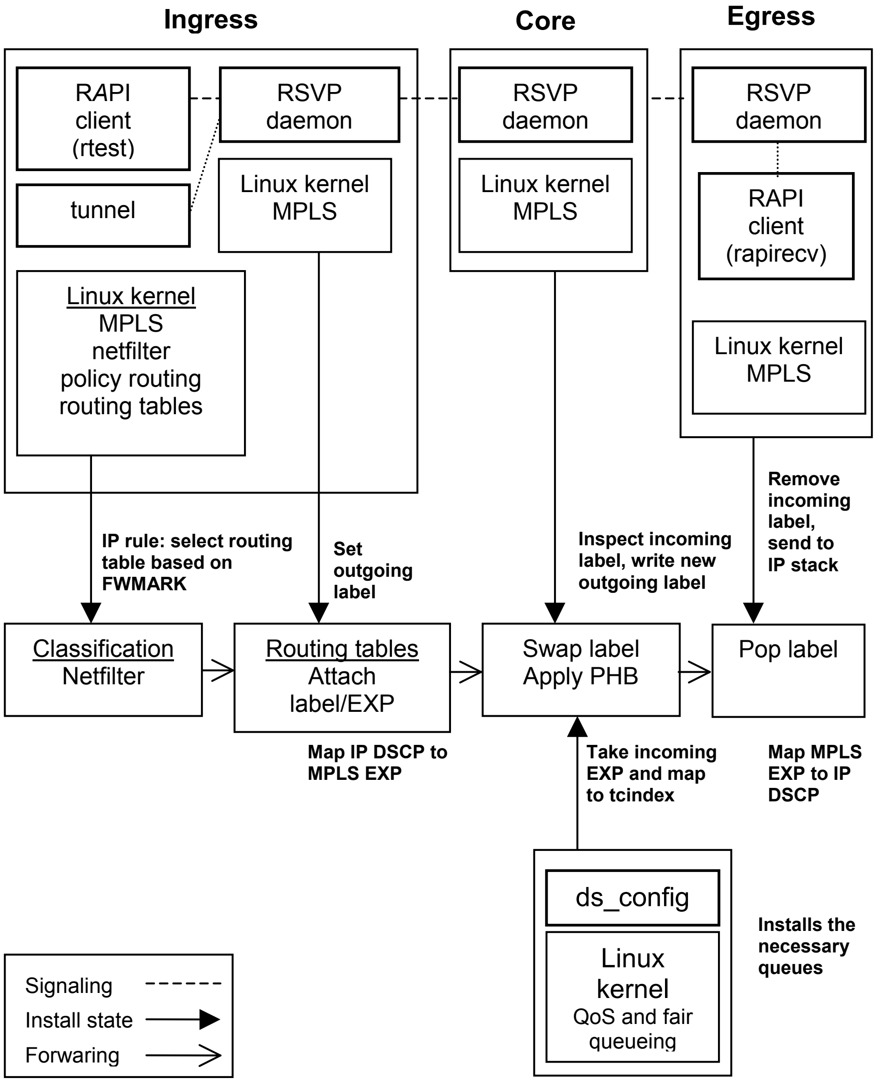

Ingress
Core
Egress
RAPI client (rtest)
RSVP daemon
tunnel
Linux kernel MPLS
Linux kernel MPLS netfilter policy routing routing tables
RSVP daemon
Linux kernel MPLS
RSVP daemon
RAPI client (rapirecv)
Linux kernel MPLS
IP rule: select routing table based on FWMARK
Set outgoing label
Inspect incoming label, write new outgoing label
Remove incoming label, send to IP stack
Classification Netfilter
Routing tables Attach label/EXP
Swap label Apply PHB
Pop label
Map IP DSCP to MPLS EXP
Take incoming EXP and map to tcindex
Map MPLS EXP to IP DSCP
ds_config
Linux kernel QoS and fair queueing
Installs the necessary queues
Signaling
Install state
Forwaring

Figure 80: DiffServ over MPLS using RSVP-TE under Linux overall architecture

Two components use the RAPI: “rtest” and “rapirecv”. rtest is an application that takes LSP requests and issues them to the daemon. rapirecv is an application that receives label requests at the egress and dictates the daemon to send a response back to the ingress. rtest2 and rapirecv_auto (not shown in the figure) are extended version of rtest and rapirecv respectively that support the automatic set up of a (large) number of LSPs.

“Tunnel” is an application that maps traffic on an existing LSP. Mapping can be based on destination address, protocol, destination ports and port ranges (TCP and UDP). So basically anything that netfilter supports. Tunnel sends status packets requests (mstat packets) to the RSVP daemon in order to receive information about the installed state in the daemon (existing sessions, labels, reservations etc.).

The forwarding path explained

In the ingress the packets are classified with netfilter. Packets are filtered on the OUTPUT and PREROUTING chain of the mangle table. The mangle table is needed because the fwmark needs to be set. The OUTPUT and PREROUTING chains are used in order to filter on both locally generated and incoming traffic. Based on the value of the fwmark a routing table is selected (using policy routing). In the resulting routing table there is a MPLS tunnel interface acting as the default gateway. The tunnel interface encapsulates the packets on the LSP by attaching the correct outgoing label. The incoming DSCP is mapped to the EXP field in the MPLS header. A limitation of this architecture is that netfilter can only write (mark) on the mangle table and that only a single mark operation is possible. So while we can map traffic on the LSP also setting the DSCP at the same node is impossible. The solution is to set the DSCP before the traffic enters the ingress LSP.

In the core node the MPLS stack inspects the incoming label and sets the new outgoing label and next hop. At the DiffServ level the current EXP value of the packet is inspected. There is a mapping from this EXP value to a tcindex. The tcindex in turn determines the correct outgoing queue so the correct forwarding behavior (PHB) can be applied.

Finally in the egress the incoming EXP field is mapped to the DSCP field and then the MPLS header is stripped off and the packet is sent to the IP layer.

B.2 The open source community

This section covers some less technical issues related to the consequences (both positive and negative) of publishing a work as an open source project.

Motivation

The decision to open-source the project was motivated by a number of factors (given in a random order). A first reason was to prevent others from having to integrate the daemons all over again or to write RSVP-TE code from scratch. And while you prevent others from doing that you can find developers and users willing to debug and develop the system. Starting an open-source project was also seen as nice way to create some publicity for our department “INTEC”, the European project “Tequila” and we as developers. Another reason was that dissemination of information should be an important task of research institutes. And finally we wanted to give back to the community and strengthen the networking offering of Linux.

Now let us have a look at how we can reflect on these factors a year after the start of the project. I believe that the project has indeed prevented others from building from scratch or integrating a new daemon. The project has attracted a number of external developers and quite a few users. These users have helped to track down numerous bugs and developers have contributed important code parts and a signification number of bug fixes. The project also led to a number of new opportunities although it is difficult to assert which opportunities were caused by the project and which were not.

A look at the pros and cons

I will sum up some of the advantages of open-sourcing a project as opposed to working quietly on your own. I will start with the disadvantages. First of all maintaining a web site and mailing list is a non-trivial task. In my particular case I am more or less forced to maintain a public and a private tree which obviously increases the overhead. Also certain types of questions on the mailing list tend to involve quite a bit of work. For instance bug reports often involve looking through debug backtraces, daemon log files and kernel state. Probably the most annoying things are people asking questions that already have been addressed in a mailing list thread or in the FAQ.

On the positive side, one of most often mentioned benefits is that extra users lead to finding bugs more quickly. This is undoubtedly true. Some bugs are only triggered by specific configurations. When somebody stumbles on an error before you, you can fix the bug when you want to, often before it is on your critical path. Complex software systems also have the property that fixing a certain problem can also fix another, on first sight unrelated, problem. This can lead to the situation where you fix a bug for a user and as a side-effect fix a bug you have been chasing down unsuccessfully till now.

Programming in an open environment tends to influence your coding style too. You rely less on hacks because you know that a hack may cause problems on other configurations and if that happens your users will report that back to you. An open-source project also requires that you keep at least a minimal set of documentation (installation instructions, change log and FAQ).

Finally there are the external developers. I have been lucky enough to have received a good deal of bug fixes but also important functionality enhancements from external developers.

Case study

We will now investigate the pros and cons of open sourcing a research project like this that is traditionally developed in house or in a closed group. The comparison is based on a use case of the public demonstration of MPLS technology as proof-of-concept. Both of the demonstrations were part of the experimentation activity of European Commission sponsored projects. We compare the closed model used in the ACTS Ithaci project [20] with the open source project code that was used in the IST Tequila project's demo [107]. The author was responsible for the MPLS signalling software in both demos.

The European projects like Ithaci and Tequila are run by a consortium that typically consists of research institutes, private companies and universities. Within a limited time period they try to tackle a specific problem space. In the case of Ithaci this was IP switching and MPLS with a special focus on multicast. In the case of Tequila this is Traffic Engineering and QoS in large scale networks. The projects are typically split up in a number of work packages. The work conducted in these packages can vary but both projects took a similar approach. The first package handled the administration and management tasks, the second package the theoretical research, the third package involves the development of the experimentation platform and the last package the integration and experimentation itself. The European Commission representatives and the auditors usually encourage the projects to conduct a public demonstration of the developed system.

The focus of the Ithaci demonstration was a testbed where unicast and multicast MPLS was run simultaneously on the same routers. During the Ithaci project we used a closed source approach even between the partners of the consortium. The MPLS stack was written by one partner and made available to us. The code was a small kernel patch and binary module. We were responsible for the unicast LDP (Label Distribution Protocol) signaling daemon. We tested our daemon in our own network prior to the integration meeting. During the integration meeting we took the components from every partner and installed and integrated them in to a large network. The integration was the preparation step before the actual demonstration. During the integration of the demo numerous bugs were discovered and we basically needed a whole week to straight things out. This is not surprising and we were well prepared and the demo was very successful. It however does contrast with the approach taken in the Tequila project.

For the Tequila demo we were, among other things, responsible for the MPLS signalling daemon. Because the daemon was open-source, it had already received diverse and thorough testing. Not only the daemon itself but also the installation instructions (which are not trivial and involve patching the kernel, tc and iptables). Some partners already got to test the daemon at their premises before the actual integration meeting. This did not create any additional overhead for us because it was sufficient to point our projects' URL. Integrating the RSVP daemon in the final experimentation platform only revealed two bugs. Both of them were related to the embedding of the applications rtest and tunnel software in a Genertic Adaptation Layer (GAL). (The GAL is used to make abstraction of the router used, the GAL supports Linux routers, Cisco and an experimental router based on a fast hardware based translator (IFT) and Linux [154].)

It is an oversimplification to state that the significant lower number of bugs discovered in the signalling daemon during integration meeting of the Tequila project is solely caused by the public availability of its source code. However distributed testing of code will lead to bugs being found earlier and to more bugs being exposed.

Appendix C

An experiment automation framework

C.1 Introduction

This appendix describes the experimentation framework that is used to conduct the experiments described in section 5.3. As we have explained in Chapter 5, it is quasi impossible to manually orchestrate the more than 600 individual measurements. Therefore we developed a measurement automation framework. Note that this appendix is not a critical part of this work but it is provided as an aid in verifying the experimentation results obtained therein. Together with Appendix A and the parameters given in section 5.3 we believe that our results are reproducible.

The framework is based on the *expect* tool. 'Expect' is a tool to automate applications that are usually performed interactively. From the expect web site [165]:

> *Expect is a tool for automating interactive applications such as telnet, ftp, passwd, fsck, rlogin, tip, etc. Expect really makes this stuff trivial. Expect is also useful for testing these same applications. ... Expect can make easy all sorts of tasks that are prohibitively difficult with anything else. You will find that Expect is an absolutely invaluable tool - using it, you will be able to automate tasks that you've never even thought of before - and you'll be able to do this automation quickly and easily.*

We will now illustrate how Expect is used in combination with shell scripts to automate the experiments. We illustrate by using the FTCR convergence time experiment as an example (see section 5.3.4).

C.2 An example

To explain the example we will first give the pseudo code. We will then give the script which is used to run the experiment. Afterwards we will explain the Expect primitives that are used in the example.

Pseudo code

Within a FTCR experiment we set up an LSP and map the test traffic to the LSP. Then the Smartbits is instructed to start sending the test traffic and to collect the statistics. Then a random time is waited before the link failure is simulated. Subsequently the link is put back up but only after sufficient time has elapsed to let FTCR recover the LSP and the Smartbits to stop sending traffic. Finally the routing daemons are restarted.

```
do numtest times
     Set up LSP
     Map test traffic on LSP
     Start sending and measuring test traffic (50s)
     Wait [30,40]s
     Tear down link
     Wait 30s
     Link up
     unmap traffic
     Restart routing daemons
     wait [100, 104]s
done
```

Note that the measurement stops before the link is brought back up (and hence before the test traffic is unmapped from the LSP). Also notice that the time between the start of the measurements and the time the link is brought down is randomised. This randomisation is less important in the FTCR experiment but important in the OSPF with Hello failure detection and the soft-state MPLS rerouting experiments in order to get the full range of possible convergence times.

Bash Script code

This section illustrates how the pseudo code is translated in a bash script that uses some expect primitives. The expect primitives can easily be recognised because the start with '`./rsvp_expect`' followed by the command. For example:

```
     ./rsvp_expect r rsvpd
```

The r(un) command will first kill any instances of the argument (`rsvpd`) and then run the argument on every machine of the test network. The above code line will restart the rsvpd daemon on every machine of the network.

The ‘`waitrand x y`’ command waits for a random time between x seconds and maximum x+y seconds with a 1ms granularity.

Note that in bash comments start with ‘#’ and output statements with ‘echo’.

```
C=0
while expr $C '<' $1
do
     echo Run === $C ===
     echo [1] restart rsvpd and setup LSP
     # stop the daemon (rsvpd) on all machines,
     # start the daemon on all machines
     ./rsvp_expect r rsvpd
     # prepare to receive PATH messages at the egress LER
     ./rsvp_expect a
     sleep 1
     # send label request from ingress LER to egress LER
     ./rsvp_expect i
     sleep 1
     echo [2] map traffic
     # create a mapping for the test traffic
     A=`./rsvp_expect m | grep LSPID`
     # check the output of the map instruction
     A=`echo $A | cut -d' ' -f1`
     # when the output contains “Wrong” something went wrong
     if expr $A : Wrong
     then
          echo "LSP not set up!"
          # retry LSP set up
          continue
     fi
     echo [2] Start Smartbits
     # start sending traffic for 50s, packet every 1ms
     # log to ftcr$C ($C is the number of the experiment)
     ./smartapp/smartapp 50 1 ftcr$C &
     echo [3] Wait for Smartbits
     sleep 20
     # wait between [10, 10+10]s
     waitrand 10 10
     echo [4] tear down link
     ./rsvp_expect d
     sleep 30
     echo [5] link up
     # set link back up
     ./rsvp_expect u
     echo [6] unmap traffic
     # unmap traffic
     ./rsvp_expect I
     # stop sending PATH messages
     ./rsvp_expect s
     # stop receiving and responding to PATH messages
     ./rsvp_expect A
     echo [7] Wait for rerouting
     # restart the routing daemons
     ./rsvp_expect rospf
     waitrand 100 4
     C=`expr $C + 1`
     # increment C
done
```

Expect primitives

r **program**	kill old instances of **program** and run **program**
d	Put down interface (simulate link failure)
m	Map test traffic on LSP
I	Unmap traffic from test LSP
a	Start rapirecv_auto (see Appendix B)
A	Stop rapirecv_auto
i	Set up test LSP
s	Tear down test LSP

rospf	(set interface back up and) Restart OSPF

Smartapp

Smartapp is an application controlling the Smartbits measurement component. There are three arguments to its invocation. The first one sets the time in seconds of the experiments, the seconds argument sets the inter-packet delay and the last argument sets the log file to which the results are logged. The results contain the packets sent at the sending side and the packets lost at the receiving side. It also contains the delay distribution of the packets.

References

All online references have been checked for their availability in April 2003. We consider that the IETF RFCs and the ATM forum documents have a stable URL so they have not been specifically rechecked during that month. Also the URL to the expired IETF drafts have not been rechecked. However all the references to them use the same draft archive [189] which has provided very complete and reliable over the recent years. All IETF drafts that have not expired have been updated to their most recent version during the same month.

[1] Prof. Dr. Guido Geerts, Drs. Ton Den Boon, “Van Dale groot woordenboek der Nederlandse taal”, dertiende herziene uitgave, 1999 Utrecht/Antwerpen.

[2] P. Van Heuven, S. Van den Berghe, F. De Turck, P. Demeester, “Wiley encyclopedia of technology (MPLS section)”, Wiley and Sons, ISBN 0-471-36972-1, December 2002.

[3] P. Van Heuven, S. De Maesschalck, D. Colle, S. Van den Berghe, M. Pickavet, P. Demeester, “Recovery in IP based networks using MPLS”, IEEE workshop on IP-oriented Operations & Management IPOM'2000, Pages 70–78, September 2000.

[4] D. Colle, P. Van Heuven, C. Develder, S. Van den Berghe, I. Lievens, M. Pickavet, P. Demeester, “MPLS recovery mechanisms for IP-over-WDM networks”, Photonic Network Communications, Kluwer Academic Publishers, Vol. 3, Nr. 1/2, Pages 23–40, January 2001.

[5] D. Colle, S. De Maesschalck, C. Develder, P. Van Heuven, A. Groebbens, J. Cheyns, I. Lievens, M. Pickavet, P. Lagasse, P. Demeester, “Data centric Optical Networks and Their Survivability”, IEEE journal of selected areas in communications, Vol. 20, Nr.1, January 2002, Pages 6–21.

[6] D. Colle, A. Groebbens, P. Van Heuven, S. De Maesschalck, M. Pickavet, P. Demeester, “Porting MPLS-recovery techniques to the MPlS paradigm”, (Invited paper) Optical Networks Magazine, Special Issue on

Protection and Survivability, Vol. 2, Nr. 4, Pages 29–47, July/August 2001.

[7] P. Van Heuven, J. Coppens, S. Van den Berghe, P. Demeester, "RSVP-TE daemon for DiffServ over MPLS under Linux", Linux Kongress, Pages 141–155, 5–6 September 2002, Cologne, Germany.

[8] P. Van Heuven, J. Coppens, S. Van den Berghe, D. Colle, P. Demeester, "Quantitative and Theoretical Analysis of Recovery Convergence in IP networks", SoftCom 2002, p 209-213, 8–11 October 2002, Split, Kroatia.

[9] J. Postel, "Internet Protocol", IETF RFC 791, http://www.ietf.org/rfc/rfc791.txt, September 1981.

[10] T. Socolofsky, C. Kale, "A TCP/IP Tutorial", IETF RFC 1180, http://www.ietf.org/rfc/rfc1180.txt, January 1991.

[11] F. Baker, "Requirements for IP Version 4 Routers", IETF RFC 1812, http://www.ietf.org/rfc/rfc1812.txt, June 1995.

[12] C. Huitema, "Routing in the Internet", New Jersey: Prentice Hall, 1995.

[13] A. Viswanathan et al., "Evolution of Multiprotocol Label Switching", IEEE communications magazine, May 1988.

[14] S. Nakazawa, H. Tamura, K. Kawahara, Y. Oie, "Performance analysis of IP datagram transmission delay in MPLS: impact of both the number and the bandwidth of LSPs of layer 2", IEEE International Conference on Communications, ICC 2001, Volume: 4, Pages 1006-1010, 2001.

[15] G. Armitage, MPLS: The magic behind the myths, IEEE communications magazine, vol. 38, no. 1, Pages 124–131, 2000.

[16] J. Boyle et al., "Applicability Statement for Traffic Engineering with MPLS", IETF RFC 3346, http://www.ietf.org/rfc/rfc3346.txt, August 2002.

[17] G. Swallow, "MPLS Advantages for Traffic Engineering", IEEE Communications Magazine, December 1999.

[18] J. Guichard and I. Pepelnjak, MPLS and VPN Architectures: A Practical Guide to Understanding, Designing and Deploying MPLS and MPLS-Enabled VPNs, Indianapolis: Cisco Press, 2000.

[19] H. Lee, J. H. Wang, B. Kang, K. Jun, “End-to-end QoS architecture for VPNs: MPLS VPN deployment in a backbone network”, Proceedings of International Workshops on Parallel Processing, Pages 479–483, 2000.

[20] I. Andrikopoulos, G. Pavlou, P. Georgatsos, N. Karatzas, K. Kavidopoulos, J. Rothig, S. Schaller, D. Ooms, P. Van Heuven, “Experiments and enhancement for IP and ATM integration: The IthACI project”, IEEE Communications Magazine, Vol. 39, Nr. 5, Pages 146–155, 2001.

[21] P. Dumortier, “Towards a New IP over ATM Routing Paradigm”, Proceedings of the ISS ’97 World Telecommunications Congress, Toronto, September 1997, also published as selected paper in IEEE Communications Magazine, Pages 82–86, Januari 1998.

[22] N. Ghani, S. Dixit, T.S. Wang, “WDM optical networks: A reality check on IP-over-WDM integration”, IEEE communications magazine, Pages 72–84, March 2000.

[23] E. Rosen, A. Viswanathan, R. Callon, “Multiprotocol Label Switching Architecture”, IETF RFC 3031, http://www.ietf.org/rfc/rfc3031.txt, January 2001.

[24] T. D. Nadeau, “Multiprotocol Label Switching (MPLS) Forward Equivalency Class-To-Next Hop Label Forwarding Entry Management Information Base”, IETF draft, http://www.ietf.org/internet-drafts/draft-ietf-mpls-ftn-mib-05.txt, November 2002.

[25] L. Andersson, P. Doolan, N. Feldman, A. Fredette, B. Thomas, “LDP Specification”, IETF RFC 3036, http://www.ietf.org/rfc/rfc3036.txt, January 2001.

[26] C. Boscher et al., “LDP state machine”, IETF RFC 3215, http://www.ietf.org/rfc/rfc3215.txt, January 2002.

[27] B. Thomas, E. Gray, ”LDP Applicability”, IETF RFC 3037, http://www.ietf.org/rfc/rfc3037.txt, January 2002.

[28] B. Jamoussi et al., “Constraint-Based LSP Setup using LDP”, IETF RFC 3212, http://www.ietf.org/rfc/rfc3212.txt, January 2002.

[29] J. Ash, “Applicability Statement for CR-LDP”, IETF RFC 3213, http://www.ietf.org/rfc/rfc3213.txt, January 2002.

[30] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow, “RSVP-TE: Extensions to RSVP for LSP Tunnels”, IETF RFC 3209, http://www.ietf.org/rfc/rfc3209.txt, December 2001.

[31] R. Braden et al., “Resource ReSerVation Protocol (RSVP)- Version 1 Functional Specification”, IETF RFC 2205, http://www.ietf.org/rfc/rfc2205.txt, September 1997.

[32] J. Wroclawski, "The Use of RSVP with IETF Integrated Services", IETF RFC 2210, http://www.ietf.org/rfc/rfc2210.txt, September 1997.

[33] D. Awduche et al., “Applicability Statement for Extensions to RSVP for LSP-Tunnels”, IETF RFC 3210, http://www.ietf.org/rfc/rfc3210, December 2001.

[34] Y. Ohba et al., “MPLS Loop Prevention Mechanism”, IETF RFC 3063, http://www.ietf.org/rfc/rfc3063.txt, February 2001.

[35] B. Davie et al.,“MPLS using LDP and ATM VC Switching”, IETF RFC 3035, http://www.ietf.org/rfc/rfc3035.txt, January 2001.

[36] J. Moy, “OSPF Version 2”, IETF RFC 2328, http://www.ietf.org/rfc/rfc2328.txt, April 1998.

[37] “Information technology, Telecommunications and information exchange between systems, intermediate system-to- intermediate system routing information exchange protocol for use in conjunction with ISO 8473”, ISO 10589, 1990.

[38] R. Callon, “Use of OSI IS-IS for Routing in TCP/IP and Dual Environments”, Digital Equipment Corporation, IETF RFC 1195, http://www.ietf.org/rfc/rfc1195.txt, December 1990.

[39] J. Halpern, “RIPv1 Applicability Statement for Historic Status”, IETF RFC 1923, http://www.ietf.org/rfc/rfc1923.txt, March 1996.

[40] G. Malkin, “RIP Version 2”, IETF RFC 2453, http://www.ietf.org/rfc/rfc2453.txt, November 1998.

[41] G. Malkin, “RIP Version 2 Protocol Analysis”, IETF RFC 1721, http://www.ietf.org/rfc/rfc1721.txt, November 1994.

[42] T. Rekhter, Y. Li, “A Border Gateway Protocol 4”, IETF RFC 1771, http://www.ietf.org/rfc/rfc1771.txt, March 1995.

[43] V. Sharma, Fiffi Hellstrand, “Framework for MPLS-based Recovery”, IETF RFC 3469, http://www.ietf.org/rfc/rfc3469.txt, February 2003.

[44] J. Coppens, “Evaluatie van herstel in IP en MPLS netwerken” (dutch), master thesis, June 2001.

[45] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, “Network flows: theory, algorithms and applications”, Prentice Hall, 1993.

[46] Kunihiro, “GNU Zebra”, [Online], http://www.zebra.org, web site, April 2003.

[47] D. Haskin, R. Krishnan, “A Method for Setting an Alternative Label Switched Paths to Handle Fast Reroute”, exprired IETF Draft, http://www.watersprings.org/pub/id/draft-haskin-mpls-fast-reroute-05.txt, November 2000.

[48] “What is Linux?”, [Web site], http://www.linux.org/info/index.html, April 2003.

[49] “Introduction to Linux and Linux.com”, [Web site], http://linux.com/article.pl?sid=02/03/09/1727250, April 2003.

[50] “GNU's Not Unix!”, [Web site], http://www.gnu.org, April 2003.

[51] Free Software Foundations, “GNU General Public License”, [Online], http://www.gnu.org/copyleft/gpl.html, April 2003.

[52] Spirent Communications, “Industry Standard Network Performance Analysis System SmartBits® 2000”, [Online], http://www.spirentcom.com/documents/47.pdf, April 2003.

[53] J. P. Vasseur, “Area size” question on OSPF mailing list, [Online], http://www.cs-ipv6.lancs.ac.uk/ipv6/mail-archive/ospf/1997-11/0021.html, April 2003.

[54] L. Berger, et al., “RSVP Refresh Overhead Reduction Extensions”, IETF RFC 2961, http://www.ietf.org/rfc/rfc2961.txt, April 2001.

[55] D. Colle, “Design and Evolution of Data-centric Optical Networks”, PhD Thesis, Ghent University, INTEC-IBCN, 2001–2002.

[56] D. Katz , “Why are we scared of SPF? IGP Scaling and Stability”, Nanog 25 meeting, June 2002, [Online], http://www.nanog.org/mtg-0206/katz.html, April 2003.

[57] P. Franciosa, D. Frigioni, R. Giaccio, “Semi-dynamic shortest paths and breath-first search in digraph”, In Proceedinds of the 14th Annual Symposium on Theoretical Aspects of Computer Science, Pages 113–124, March 1997.

[58] D. Frigioni, A. Marchetti-Spaccamela, U. Nanni, “Incremental algorithms for single-source shortest path trees”, In Proceedings of Foundations of Software Technology and Theoretical Computer Science, Pages 113–124, December 1994.

[59] Paolo Narvaez, Kai-Yeung Siu, Hong-Yi Tzeng, “New dynamic SPT algorithm based on a ball-and string model”, In Proceedings of the IEEE Infocom, 1999.

[60] C. Filsfils, “IGP Fast Convergence, ISIS Case Study”, Ripe meeting 41, January 2002, [Online], http://www.ripe.net/ripe/meetings/archive/ripe-41/presentations/eof-isis/index.html, April 2003.

[61] “Merriam-Webster Online, The language center”, [Online], http://www.m-w.com, April 2003.

[62] S. De Maesschalck, D. Colle, A. Groebbens, C. Develder, I. Lievens, P. Lagasse, M. Pickavet, P. Demeester, F. Saluta, M. Quagliotti, “Intelligent optical networking for multilayer survivability”, IEEE Communications Magazine, Vol. 40, Nr. 1, January 2002, Pages 42–49.

[63] A. Groebbens, D. Colle, S. De Maesschalck, M. Pickavet, P. Demeester, “Spare capacity cuts in multi-protocol lambda switching networks using Backup Trees”, Proceedings of the Sixth Informs Telecom Conference, Boca Raton, Florida, USA, Pages 51-52, March 10-13 2002.

[64] U. Black, “ATM: Foundation for Broadband Networks”, Prentice Hall Series, 1995.

[65] M. De Prycker, “Asynchronous Transfer Mode - Solution for Broadband ISDN”, 3rd Ed., Prentice Hall, 1995.

[66] R. Handel, M.N. Huber, S. Schroder, “ATM Networks - Concepts, Protocol`s, Applications”, Addison-Wesley, 1994.

[67] S. Keshav, “An Engineering Approach to Computer Networking - ATM Networks, the Internet and the Telephone Network”, Addison-Wesley, 1997.

[68] B. Davie, P. Doolan, Y. Rekhter, “Switching in IP Networks”, Morgan Kaufmann, 1998.

[69] Y. Rekhter, “Tag Switching Architecture – Overview”, IETF RFC 2105, http://www.ietf.org/rfc/rfc2105.txt, February 1997.

[70] N. Feldman, “ARIS Specification”, expired IETF draft, http://www.watersprings.org/pub/id/draft-feldman-aris-spec-00.txt, March 1997.

[71] Y. Katsube et. at., “Toshiba's Router Architecture Extensions for ATM : Overview”, http://www.ietf.org/rfc/rfc2098.txt, February 1997

[72] P. Newman et al., “Ipsilon Flow Management Protocol Specification for IPv4”, IETF RFC 1953, May 1996.

[73] A.Acharya et al., “IPSOFACTO: IP Switching Over Fast ATM Cell Transport”, expired IETF draft, http://www.watersprings.org/pub/id/draft-acharya-ipsw-fast-cell-00.txt, July 1997.

[74] G. Swallow, L. Andersson, “The IETF MPLS working group charter”, http://www.ietf.org/html.charters/mpls-charter.html, last modified September 2002.

[75] ATM forum technical committee, “Multi-Protocol over ATM version 1.0”, ftp://ftp.atmforum.com/pub/approved-specs/af-mpoa-0087.000.pdf, July 1997.

[76] R. Jain, “Multiprotocol over ATM”, Tutorial, http://www.cis.ohio-state.edu/~jain/atm/atm_mpoa.htm

[77] A, Mallis, “Internetworking Over NBMA IETF working group charter”, http://www.ietf.org/html.charters/OLD/ion-charter.html, June 2000.

[78] T. Li, “MPLS and the Evolving Internet Architecture”, IEEE communications magazine, Pages 38–41, December 1999.

[79] “Data Networks and Open Systems Communications: Open Systems Interconnection – model and notation”, ITU-T Rec. G.805, March 2000.

[80] E. Rosen et al., “MPLS Label Stack Encoding”, IETF RFC 3032 http://www.ietf.org/rfc/rfc3032.txt, January 2001.

[81] “The frame Relay Forum, [Website], http://www.frforum.com/, April 2003.

[82] A. Conta et al., “Use of Label Switching on Frame Relay Networks Specification”, IETF RFC 3034, http://www.ietf.org/rfc/rfc3034.txt, January 2001.

[83] K. Ramakrishnan, S. Floyd, “A Proposal to add Explicit Congestion Notification (ECN) to IP”, IETF RFC 2481, http://www.ietf.org/ietf/rfc2481, January 1999.

[84] ATM forum technical committee, “ATM User Network Interface (UNI) Signalling Specification version 4.1”, ftp://ftp.atmforum.com/pub/approved-specs/af-sig-0061.001.pdf, April 2002.

[85] ATM forum technical committee, “Private Network-Network Interface Specification v.1.1”, ftp://ftp.atmforum.com/pub/approved-specs/af-pnni-0055.002.pdf, April 2002.

[86] P. Agarwal, “Time-to-live (TTL) Processing in MPLS Networks (Updates RFC 3032)”, IETF RFC 3443, http://www.ietf.org/rfc/rfc3443.txt, January 2003.

[87] Y. Rekhter et al., “Carrying Label Information in BGP-4”, IETF RFC 3107, www.ietf.org/rfc/rfc3107.txt, May 2001.

[88] J. Wroclawski et al., “Specification of the Controlled-Load Network Element Service”, IETF RFC 2211, http://www.ietf.org/rfc/rfc2211.txt, September 1997.

[89] S. Shenker, C. Partridge, R. Guerin, “Specification of Guaranteed Quality of Service”, IETF RFC 2212, http://www.ietf.org/rfc2212.txt, September 1997.

[90] V. Jacobson, “An Architecture for Differentiated Services”, Talk at IRTF End-to-end Working Group, July 1997, [Online], ftp://ftp.ee.lbl.gov/talks/vj-e2e-jul97.pdf, April 2003.

[91] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, “An Architecture for Differentiated Services”, IETF RFC 2475, http://www.ietf.org/rfc/rfc2475.txt, December 1998.

[92] K. Nichols, V. Jacobson, L. Zhang, “A Two-bit Differentiated Services Architecture for the Internet”, IETF RFC 2638, http://www.ietf.org/rfc/rfc2638.txt, July 1999.

[93] D. Grossman, “New Terminology and Clarifications for Diffserv”, IETF RFC 3260, http://www.ietf.org/rfc3260.txt, April 2002

[94] K. Nichols, S. Blake, F. Baker, D. Black, “Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers”, IETF RFC 2474, http://www.ietf.org/rfc/rfc2474.txt, December 1998.

[95] S. Brim et al.,“Per Hop Behavior Identification Codes”, IETF RFC 2836, http://www.ietf.org/rfc/rfc2836.txt, May 2000.

[96] V. Jacobson, K. Nichols, K. Poduri, “An Expedited Forwarding PHB”, IETF RFC 2598, http://www.ietf.org/rfc/rfc2598.txt, June 1999.

[97] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, “Assured Forwarding PHB Group”, IETF RFC 2597, http://www.ietf.org/rfc/rfc2597.txt, June 1999.

[98] F. Le Faucheur, et al.,”Multi-Protocol Label Switching (MPLS) Support of Differentiated Services”, IETF RFC 3270, http://www.ietf.org/rfc/rfc3270.txt, May 2002.

[99] D. Awduche, “MPLS and Traffic Engineering in IP Networks”, IEEE Communications Magazine, December 1999.

[100] B. Fortz, M. Thorup, “Internet Traffic Engineering by Optimizing OSPF Weights”, IEEE INFOCOM, 2000.

[101] J. Harmatos, “A heuristic algorithm for solving the static weight optimisation problem in OSPF networks”, Global Telecommunications Conference, IEEE GLOBECOM, Volume 3, Pages 1605–1609, 2001.

[102] B. Fortz, M. Thorup, “Optimizing OSPF/IS-IS weights in a changing world”, Selected Areas in Communications, IEEE Journal on, Volume 20 Issue: 4, Pages 756 –767, May 2002.

[103] A. Iwata, Traffic Engineering Extensions to OSPF Summary LSA, expired IETF draft, http://www.watersprings.org/pub/id/draft-fujita-ospf-te-summary-00.txt, March 2000.

[104] D. Katz et al., “Traffic Engineering Extensions to OSPF”, IETF draft, http://www.ietf.org/internet-drafts/draft-katz-yeung-ospf-traffic-09.txt, October 2002.

[105] C. Villamizar, “OMP tutorial”, [Website], http://www.fictitious.org/omp/tutorial.html, April 2003.

[106] C. Villamizar, “OSPF Optimized Multipath (OSPF-OMP)”, expired IETF draft, http://www.watersprings.org/pub/id/draft-ietf-ospf-omp-02.txt, February 1999.

[107] Tequila consortium, “Tequila: Traffic Engineering for Quality of Service in Internets at large scale”, [web site], http://www.ist-tequila.org, April 2003.

[108] P. Van Heuven (ed.), “IP-QoS State of the Art Overview, Requirements and Assumptions”, TEQUILA Internal Report 1.1, August 2000.

[109] D. Goderis (ed.), D1.1: Functional Architecture Definition and Top Level Design, CEC no. 101/Alcatel/b1, July 2001.

[110] P. Trimintzios (ed.), D1.2: Protocol and Algorithm Specification, TEQUILA Consortium Deliverable, CEC no. 102/UniS/b1, January 2001.

[111] P. Van Heuven (ed.), D1.3: Intermediate-Results based Protocol and Algorithm Specification, TEQUILA Consortium Deliverable, CEC no. 103/IMEC/b1, October 2001.

[112] P. Van Heuven (ed.), D1.4: Final Protocol and Algorithm Specification, TEQUILA Consortium Deliverable, CEC no. 104/IMEC/b1, April 2002.

[113] G. Apostopoulos et al., “QoS Routing Mechanisms and OSPF Extensions”, IETF RFC 2676, http://www.ietf.org/rfc/rfc2676.txt, August 1999.

[114] E. Rosen et al., “BGP/MPLS VPNs”, IETF RFC 2547, http://www.ietf.org/rfc/rfc2547.txt, March 1999.

[115] E. Rosen et al., “BGP/MPLS VPNs”, IETF draft, http://www.ietf.org/internet-drafts/draft-ietf-ppvpn-rfc2547bis-03.txt, October 2002.

[116] E. Mannie, “Generalized Multi-Protocol Label Switching (GMPLS) Architecture”, IETF draft, http://www.ietf.org/internet-drafts/draft-ietf-ccamp-gmpls-architecture-05.txt, March 2003.

[117] Eric Mannie, D. Papadimitriou, “Generalized Multi-Protocol Label Switching Extensions for SONET and SDH Control”, IETF draft,

http://www.ietf.org/internet-drafts/draft-ietf-ccamp-gmpls-sonet-sdh-08.txt, February 2003.

[118] D. Papadimitriou, “Generalized MPLS Signalling Extensions for G.709 Optical Transport Networks Control”, IETF draft, http://www.ietf.org/internet-drafts/draft-ietf-ccamp-gmpls-g709-03.txt, November 2002.

[119] L. Berger, “Generalized MPLS Signaling - RSVP-TE Extensions“, IETF RFC 3473, http://www.ietf.org/rfc/rfc3473.txt, January 2003.

[120] P. Ashwood-Smith, L. Berger, “Generalized MPLS Signaling - CR-LDP Extensions”, IETF RFC 3472, http://www.ietf.org/rfc/rfc3472.txt, August 2002.

[121] K. Kompella et al., “Link Bundling in MPLS Traffic Engineering”, IETF draft, http://www.ietf.org/internet-drafts/draft-ietf-mpls-bundle-04.txt, July 2002.

[122] Vishal Sharma, Fiffi Hellstrand, “Framework for MPLS-based Recovery”, IETF RFC 3469, http://www.ietf.org/rfc/rfc3469.txt, February 2003.

[123] C. Develder, D. Colle, P. Van Heuven, S. Van den Berghe, M. Pickavet, P. Demeester, “Influence of recovery time on TCP behaviour”, MPLS world congress 2001, Paris, France, February 6–9, 2001.

[124] J.Moy, OSPF protocol analysis, IETF RFC 1245, ftp://ftp.ietf.org/rfc/rfc1245.txt, July 1991.

[125] J.Moy, Experience with the OSPF protocol, IETF RFC 1246, ftp://ftp.ietf.org/rfc/rfc1246.txt, July 1991.

[126] J. Moy, “OSPF: Anatomy of an Internet Routing Protocol”, Addison-Wesley, 1998.

[127] “The ATM forum glossary”, http://www.atmforum.com/aboutatm/glossary.html, November 2002.

[128] P. Willis, “What are the requirements for MPLS OAM?”, http://www.ietf.org/proceedings/01dec/slides/mplsoam-4/sld016.htm, December 2002.

[129] H. Ohta, “Assignment of the 'OAM Alert Label' for Multiprotocol Label Switching Architecture (MPLS) Operation and Maintenance (OAM) Functions”, IETF RFC 3429, http://www.ietf.org/rfc/rfc3429.txt, November 2002.

[130] D. Allan, “A Framework for MPLS User Plane OAM”, IETF draft, http://www.ietf.org/internet-drafts/draft-allan-mpls-oam-frmwk-04.txt, February 2003.

[131] K. Owens et al., “A Path protection/Restoration Mechanism for MPLS networks”, http://www.watersprings.org/pub/id/draft-chang-mpls-path-protection-03.txt, 2003.

[132] K. Owens et al., "Extensions to RSVP-TE for MPLS Path Protection", http://www.watersprings.org/pub/id/draft-chang-mpls-rsvpte-path-protection-ext-02.txt, July 2001.

[133] P. Van Heuven, S. Van den Berghe, J. Coppens, P. Demeester, “RSVP-TE daemon for DiffServ over MPLS under Linux”, open source project, [Website], http://dsmpls.atlantis.rug.ac.be, April 2003.

[134] C. Labovitz et al., “Delayed Internet Routing Convergence”, IEEE/ACM transactions on networking, Vol. 9, No. 3, June 2001.

[135] C. Hedrick, “Routing Information Protocol”, IETF RFC 1058, http://www.ietf.org/rfc/rfc1058.txt, June 1988.

[136] S. Sherry, G. Meyer, “Protocol Analysis for Triggered RIP”, IETF RFC 2092, http://www.ietf.org/rfc2092.txt, January 1997.

[137] R. Perlman, “A comparison between two routing protocols: OSPF and IS-IS”, IEEE Network , Volume: 5 Issue: 5 , Page(s): 18 –24, Sept. 1991.

[138] O. Sharon, “Dissemination of routing information in broadcast networks: OSPF versus IS-IS”, IEEE Network , Volume: 15 Issue: 1 , Pages: 56 – 65, Jan.-Feb. 2001.

[139] C. Alaettinoglu, V. Jacobson, H. Yu, “Towards mili-second IGP convergence”, expired IETF draft, http://www.watersprings.org/pub/id/draft-alaettinoglu-isis-convergence-00.txt, November 2000.

[140] Cisco Systems, “OSPF Design Guide”, http://www.cisco.com/warp/public/104/2.html, December 2002.

[141] Juniper Networks, “JUNOS 5.0 Internet Software Configuration Guide: Routing and Routing Protocols, Hello-Interval”, http://www.juniper.net/techpubs/software/junos50/swconfig50-routing/html/ospf-summary13.html#1014246, February 2003.

[142] Juniper Networks, “JUNOS 5.0 Internet Software Configuration Guide:Routing and Routing Protocols, Dead-Interval”, http://www.juniper.net/techpubs/software/junos50/swconfig50-routing/html/ospf-summary6.html#1014092, February 2003.

[143] P. Brittain, A. Farrel, “MPLS Traffic Engineering: A Choice Of Signalling Protocols”, January 2000.

[144] V. Paxson, “End-to-End Routing Behavior in the Internet”, IEEE/ACM Trans on Networking, vol 5,no 5, October 1997.

[145] K. Owens et al.,“A path protection/restoration mechanism for MPLS networks”, expired IETF draft, http://www.watersprings.org/pub/id/draft-chang-mpls-path-protection-03.txt, July 2001.

[146] P. Pan, “Fast Reroute Extensions to RSVP-TE for LSP Tunnels”, http://www.ietf.org/internet-drafts/draft-ietf-mpls-rsvp-lsp-fastreroute-02.txt, February 2003.

[147] D. Stamatelakis, W.D. Grover, “IP layer restoration and network planning based on virtual protection cycles “, Selected Areas in Communications, IEEE Journal on , Volume: 18 Issue: 10 , Page(s): 1938 –1949, Oct. 2000.

[148] JTC 1/SC 7, “Information processing -- Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts”, ISO 5807:1985, May 1999.

[149] R. L. Oakman, “The Computer Triangle”, section 5.2, “Flowchart symbols”, [Online], http://www.wiley.com/college/busin/icmis/oakman/outline/chap05/slides/symbols.htm, April 2003.

[150] A. Shaikh, M. Goyal, A. Greenberg, R. Rajan, K. K. Ramakrishnan, “An OPSF Topology Server: Design and Experience”, IEEE journal on selected areas in communications, vol. 20, no.4, May 2002.

[151] R. Siamwalla, R. Sharma, S. Keshav, “Discovering internet topology”, July 1998, [Online], http://www.cs.cornell.edu/skeshav/papers/discovery.pdf, April 2003.

[152] R. Govindan, H. Tangmunarunkit, “Heuristics for Internet map discovery”, Proc. IEEE INFOCOM, Pages1371-1386, Mar. 2000.

[153] E. Van Breusegem, “Ontdekking van de toestand en de topologie van een netwerk” (dutch), master thesis, June 2001.

[154] C. Duret, F. Rischette, J. Lattmann, V. Laspreses, P. Van Heuven, S. Van den Berghe and P. Demeester; “High Router Flexibility and Performance by Combining Dedicated Lookup Hardware (IFT), Off-the-Shelf Switches and Linux”, Networking 2002, Pisa, Italy, 19-24 May 2002. [Online], http://www.ist-tequila.org/publications/routers-networking2002.pdf, April 2003.

[155] James Lieu (maintainer), “MPLS for Linux”, [Web site], http://sourceforge.net/projects/mpls-linux, April 2003.

[156] “OpenBSD: Free, Functional and Secure”, [Web site], http://www.openbsd.com, April 2003.

[157] “FreeBSD: The power to serve”, [Web site], http://www.freebsd.org, April 2003.

[158] “NetBSD: of course it runs NetBSD”, [Web site], http://www.netbsd.org, April 2003.

[159] “Audio-latency test results”, [Web site], http://kpreempt.sourceforge.net/benno/linux-2.4.6/3x256.html, April 2003.

[160] “Linux scheduling latency”, [Web site], http://www.zip.com.au/~akpm/linux/schedlat.html, April 2003.

[161] FSMLabs, “FSMLabs RTLinux”, [Web site], http://www.fsmlabs.com/community/, April 2003.

[162] Information Science Institute, “RSVP ReSerVation Protocol”, [Web site], http://www.isi.edu/div7/rsvp, April 2003.

[163] A. Kuznetsov, “RSVP download repository” [Online], ftp://ftp.inr.ac.ru/ip-routing/rsvp, April 2003.

[164] NIST Internetworking Technology Group (ITG), "The NIST Switch home page",[Web site], http://snad.ncsl.nist.gov/itg/nistswitch, April 2003.

[165] D. Libes, "The expect home page", [Web site], http://expect.nist.gov, April 2003.

[166] M. Muuss, "The Story of the PING Program", [Online], http://ftp.arl.mil/~mike/ping.html, April 2003.

[167] J. Postel, "Internet Control Message Protocol DARPA Internet Program Protocol Specification", IETF RFC, http://www.ietf.org/rfc/rfc792.txt, September 1981.

[168] V. Paxson et al.,"Framework for IP Performance Metrics", IETF RFC, http://www.ietf.org/rfc/rfc2330.txt, May 1998.

[169] US Naval Observatory, "Global Positioning System Overview", November 2002, [Online], http://tycho.usno.navy.mil/gps.html, April 2003.

[170] D. L. Mills, "Network Time Protocol (Version 3) Specification, Implementation and Analysis", IETF RFC, http://www.ietf.org/rfc/rfc1305.txt, March 1992.

[171] Spirent Communications, "GX-1405B, GX-1405Bs, SX-7210, SX-7410B 10/100/1000 Mbps Ethernet SmartCards", [Online], http://www.spirentcom.com/documents/605.pdf, April 2003.

[172] Spirent Communications, "SmartBits 2000 Chassis", [Online], http://www.spirentcom.com/analysis/product_product.cfm?PL=33&PS=16&PR=139&P=240, April 2003.

[173] Spirent Communications, "Software Developer s Kit Software Developer's Kit SmartLib", [Online], http://www.spirentcom.com/documents/603.pdf, April 2003.

[174] Spirent Communications, "Smartlib command reference, Programming Library Version 3.20", May 2002.

[175] G. L. Choudhury, A. S. Maunder, V. D. Sapozhnikova, , "Faster link-state IGP convergence and improved network scalability and stability", Local Computer Networks, LCN 2001, 26th Annual IEEE Conference on, Proceedings Page(s): 149 –158, 2001.

[176] G. L. Choudury et al., “Explicit Marking and Prioritized Treatment of Specific OSPF Packets for Faster Convergence and Improved Network Scalability and Stability”, IETF draft, http://www.ietf.org/internet-drafts/draft-ietf-ospf-scalability-02.txt, November 2002.

[177] C. Filsfils, “Deploying Tight-SLA services on an IP Backbone: ISIS Fast Convergence and Differentiated Services Design”, Nanog 25 meeting, June 2002, [Online], http://www.nanog.org/mtg-0206/filsfils.html, April 2003.

[178] P. Van Heuven, P. Demeester, “Defining the rationale for moving restoration and protection up to the MPLS layer”, MPLS Forum, June 12-13, 2000, Dublin, Ireland.

[179] P. Van Heuven, P. Demeester, “Defining the rationale for moving restoration and protedtion up to the MPLS layer”, The Second Vision in Business summit on MPLS, June 14-16, 2000, Dublin, Ireland.

[180] D. Colle, C. Develder, P. Van Heuven, S. Demaesschalck, A. Groebbens, M. Pickavet, P. Demeester, “Resilience in IP-over-WDM networks”, (Invited paper) Proceedings of the 5th Working Conference on Optical Network Design and Modelling (ONDM 2001), 5-7 February 2001, Vienna, Austria.

[181] D. Pappalardo, “AT&T, costumers grapple with ATM net outage”, Network World, February 26, 2001 [Online], http://www.nwfusion.com/news/2001/0226attatm.html, April 2003.

[182] J. Ash, et.al., “Proposed Mechanisms for Congestion Control Failure Recovery in OSPF & ISIS Networks”, expired IETF draft, http://www.watersprings.org/pub/id/draft-ash-ospf-isis-congestion-control-02.txt, June 2002.

[183] A. Farrel et al., “Fault Tolerance for the Label Distribution Protocol (LDP)”, Internet draft, http://www.ietf.org/internet-drafts/draft-ietf-mpls-ldp-ft-06.txt, September 2002.

[184] C. Alaettinoglu, S. Casner, “Detailed Analysis of ISIS Routing Protocol on the Qwest Backbone: A recipe for subsecond ISIS convergence”, Presentation, February 2002, [Online], http://www.packetdesign.com/news/presentations/2002/qwest.pdf, April 2003.

[185] Qwest Communication International Inc., “Qwest North American Broadband Network”, [Online], http://www.qwest.com/about/qwest/network/northamerican.html, April 2003.

[186] M. Handley et al., “SIP: Session Initiation Protocol”, IETF RFC, http://www.ietf.org/rfc/rfc2543.txt, March 1999.

[187] International Telecommunication Union, “Recommendation H.323”, November 2000.

[188] S. Rao, S. Ahmed, R. Southern, “Fast Reroute, A high availability addition to MPLS”, Nanog 26 meeting, [Online], http://www.nanog.org/mtg-0210/ppt/shankar.ppt, April 2003.

[189] Watersprings.org, “Internet-Draft Archive”, [Online], http://www.watersprings.org/pub/id/index-all.html, April 2003.