Universidade de Aveiro
2012

Departamento de Electrónica, Telecomunicações e Informática

João Pedro Pereira Oliveira Lopes

Switches Auto-Configuráveis

Universidade de Aveiro
2012

Departamento de Electrónica, Telecomunicações e Informática

João Pedro Pereira Oliveira Lopes

Switches Auto-Configuráveis

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e de Telecomunicações, realizada sob a orientação científica do Professor Doutor André Ventura da Cruz Marnoto Zúquete, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro, e da Professora Doutora Susana Isabel Barreto de Miranda Sargento, Professora Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

À Paula, à Inês e ao José Pedro. Por todo o tempo que vos roubei.

o júri / the jury

presidente / president

Professor Doutor Paulo Bacelar Reis Pedreiras
Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Professor Doutor Rui Manuel Rodrigues Rocha
Professor Associado do Departamento de Engenharia Electrotécnica e de Computadores do Instituto Superior Técnico de Lisboa

Professor Doutor André Ventura da Cruz Marnoto Zúquete
Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro (orientador)

Professora Doutora Susana Isabel Barreto de Miranda Sargento
Professora Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro (coorientador)

agradecimentos / acknowledgements

Os meus agradecimentos abrangem três vectores distintos: académico, profissional e pessoal.

A nível académico, quero agradecer aos dois professores que me acompanharam nesta jornada. Tive a felicidade de conhecer e empreender este projecto com duas pessoas extremamente competentes, carismáticas e motivadoras. Muito obrigado pela orientação ao longo do trabalho e pelo profissionalismo com que o fizeram.

A nível profissional, à Vodafone Portugal, pelas condições que me proporcionou para desempenhar este meu trabalho. De uma forma especial, o meu profundo agradecimento à manager da equipa NSU IP Operations PT, Engenheira Maria João Correia, pelo seu apoio, interesse e incentivo constante.

A nível pessoal, aos meus familiares mais próximos, principalmente aos meus pais. Todo o esforço de uma vida em prol da educação e felicidade dos filhos é com certeza o maior exemplo que desejo seguir e concretizar. Espero que se sintam recompensados e concretizados nesse vosso esforço.

Aos meus filhos Inês e José Pedro. São simplesmente a minha vida e as crianças mais bonitas e extraordinárias que conheço. Deixam-me orgulhoso e efectivamente realizado por ser pai.

À minha esposa Paula. Incentivo, dedicação, motivação, apoio, compreensão, superação, paciência, carinho, ajuda, espírito de sacrifício e preocupação com o meu sucesso e bem-estar, são apenas algumas das palavras com que posso descrever esta pessoa fantástica que tenho ao meu lado. Muito obrigado por seres tu. Sabes bem que não há páginas nem palavras que cheguem para te agradecer.

palavras-chave

Spanning Tree Protocol, TRILL, SPB, SCS, Redes Layer 2, Redes Ethernet

resumo

O protocolo Spanning Tree é antigo e completamente desajustado às necessidades das redes Ethernet de hoje em dia. Os seus sucessores, Rapid Spanning Tree Protocol e Multiple Spanning Trees apresentam igualmente problemas de estabilidade e tempos de convergência inadequados.

Recentemente foram propostos para standardização, dois novos protocolos que visam a substituição desses protocolos baseados em árvores abrangentes (spanning trees): o TRILL, por parte do IETF, e o SPB, por parte do IEEE. Muito embora sejam idênticos em diversas características, apresentam paradigmas de encaminhamento bastante distintos. O TRILL actua como um protocolo de Layer 3, enquanto que o SPB comporta-se como um típico protocolo de Layer 2. Tanto o TRILL como o SPB adequam-se a grandes redes Ethernet, requerem muito processamento das máquinas e implicam avultados investimentos em novos equipamentos.

O protocolo Self-Configurable Switches (SCS) visa substituir os protocolos de spanning tree, alterando radicalmente a filosofia de encaminhamento e controlo de tramas nas redes Ethernet, mas mantendo-se adequado ao segmento dos equipamentos e redes que actualmente correm esses protocolos de spanning tree, minimizando assim potenciais investimentos exagerados.

Esta tese apresenta um resumo das funcionalidades e problemas dos actuais protocolos de spanning tree, as principais características dos novos protocolos propostos a standardização e os processos e mecanismos do novo protocolo Self-Configurable Switches (SCS). No final, são apresentados resultados de testes comparativos de funcionalidade, aplicabilidade e performance do protocolo SCS versus os protocolos de spanning tree, que atestam o desempenho superior do protocolo SCS.

keywords

Spanning Tree Protocol, TRILL, SPB, SCS, Layer 2 Networks, Ethernet Networks

abstract

The Spanning Tree Protocol is old and completely misadjusted towards current Ethernet networks requirements. Its successors, Rapid Spanning Tree Protocol and Multiple Spanning Trees reveal also stability problems and poor convergence times.

Recently two new protocols aiming spanning tree protocols substitution were proposed to standard: IETF's TRILL and IEEE's SPB. Although similar in many aspects, their forwarding paradigms are quite different. TRILL acts like a Layer 3 protocol, whereas SPB keeps the typical Layer 2 protocol behaviour. Both TRILL and SPB target backbone or core Ethernet networks, requiring great processing power from bridges and huge investment in new gear.

Self-Configurable Switches protocol (SCS) objective is to replace the spanning tree protocols, changing radically the frame forwarding and control philosophy over Ethernet networks, but keep being suitable to the range of equipment and networks that tipically run those spanning tree protocols, minimizing potential large investments.

This thesis presents an overview of all features and problems of spanning tree protocols, the main characteristics of the new proposal standards and the processes and mechanisms of the new Self-Configurable Switches protocol. At the end, test results are presented regarding features, feasability and performance of SCS protocol versus the spanning tree protocols, which attest SCS protocol superior performance.

Contents

List of Figures

List of Tables

Acronyms

BEB - Backbone Edge Bridges
BMAC - Backbone MAC Address
BVID - Backbone VLAN ID
CAPEX - Capital Expenditure
CMAC - Client MAC addresses
IEEE - Institute of Electrical and Electronics Engineers
IETF - Internet Engineering Task Force
I-SID - Service Instance Identifier
ISIS, IS-IS – Intermediate System to Intermediate System
LSP - Link State Packets
MST – Multiple Spanning Trees
MSTP - Multiple Spanning Trees Protocol
MSTPI - Multiple Spanning Trees Protocol Instance
Ns3 - Network simulator, version 3
OPEX – Operational Expenditure
OSPF – Open Shortest Path First
RBridge - Routing Bridge
RSTP – Rapid Spanning Tree Protocol
SCS – Self-configurable Switches
SCSID - SCS Bridge ID
SPB – Short Path Bridging (802.1aq)
STP – Spanning Tree Protocol
TRILL – Transparent Interconnection of Lots of Links

Introduction

The introductory chapter of this thesis presents the main motivation and objectives for this project and also the document's outline.

Motivation

Beth Israel Deaconess Medical Centre, November 13th, 2002.

The prestigious Beth Israel Deaconess Medical Centre network crashed repeatedly over four days, forcing the hospital to revert back processes thirty years, into 1970's paper systems. The hospital had backup generators, backup for backup generators, clustered servers, disk mirroring and redundant connections. However, the bridging network was overlooked and *Spanning Tree Protocol* was running…

After four dramatic, non-sleeping days, the problem was, basically, the violation of Spanning Tree Protocol hop count limit.

Beth Israel Deaconess Medical Centre spent near three million dollars to redesign and replace its entire network.

This issue reached IEEE 802 LAN/MAN Standards Committee. Tony Jeffree answers quite interestingly to Robert D. Love concern about this issue: "*This is not a 'problem with Spanning Tree', but a problem with *configuring* (or more to the point, mis-configuring) Spanning Tree.*" [2][3].

Ten years have passed. However, Spanning Tree protocol still runs over millions of Ethernet networks, hopefully, at least, in its latest versions *Rapid Spanning Tree* (RSTP) or *Multiple Instances Spanning Tree* (MSTP). Spanning Tree Protocol was a beautiful solution to a difficult problem. Great work performed by Radia Perlman. Nevertheless, the *spanning tree* concept is no longer suitable for today's network requirements, and neither RSTP nor MSTP are appropriate upgrades for such availability, resilience and performance demands.

Personally, I have already faced a dozen of Spanning Tree issues during my short 12-year networking career, all with major impact in production services and hard to troubleshoot. It is challenging to proactively detect network loops and/or Spanning Tree problems; as soon as you detect them, it's already too late. Although the trigger might change, the cause is always the same: a network loop, somewhere.

Problems like the one faced by Beth Israel Deaconess Medical Centre are pretty common. However, not everyone has the candour and humbleness of John Halamka, CIO of Beth Israel Deaconess Medical Centre, who shared the details of one of the worst health-care IT crises in history. "*I made a mistake, and the way I can fix that is to tell everybody what happened so they can avoid this*", he said. Nevertheless, it is understandable that, most of the times, Spanning Tree Protocol induced outages are *hidden* from the public as business credibility and operationally relies directly on network performance and services availability.

Beth Israel Deaconess Medical Centre incident had such magnitude that medical procedures, disaster recovery plans and networking protocols were revised worldwide and new processes were studied and developed. At network level, many people started working on Spanning Tree enhancements; others, on its replacement. Even its *mother*, for years, is driving her genius to give birth to a new paradigm for bridges, contributing passionately with TRILL IETF working group [4].

Recently, two protocols were proposed to replace Spanning Tree: IETF's *Transparent Interconnect of Lots of Links* (TRILL) and IEEE's *802.1aq Shortest Path Bridging* (SPB).

TRILL and SPB are very similar on many factors and are clearly tailored for carrier/backbone networks. Both use the *Intermediate System-to-Intermediate System* (IS-IS) link-state protocol to discover and advertise network topology and compute shortest paths. However they have completely different forwarding paradigms: TRILL acts like a Layer 3 protocol, rewriting the Layer 2 address hop-by-hop, whereas SPB behaves like a regular Layer 2 protocol, not changing the frame until it gets to its destination. Those two new protocols are quite complex and processor intensive, pulling up resources demands for bridges. Moreover, ISIS expertise is needed to maintain and troubleshoot such networks. Those two will require, for sure, *wealthy wallets*...

There is obviously a huge gap between Spanning Tree and new TRILL and SPB philosophies. That's our window of opportunity: *Self-Configurable Switches* replace Spanning Tree in a simple and easy way, upgrading network service with resiliency, load balancing, efficiency, stability and performance, maintaining Ethernet bridges *as is*: bridges.

Objectives

Self-Configurable Switches protocol targets Spanning Tree Protocol replacement.

Spanning Tree Protocol should be replaced by a simple protocol requiring zero-configuration, providing self-configuring capabilities, delivering optimized unicast and broadcast forwarding, presenting minimized transitory convergence periods, allowing traffic load sharing between redundant paths and exhibiting real *plug-and-play* methodology, not Spanning Tree's *plug-and-pray*.

The recent proposed standards TRILL and SPB have completely different scopes. Although they accomplish some of the aforementioned requisites, they do not deliver a smooth transition from Spanning Tree Protocol, as their analysis throughout this thesis will expose.

SCS algorithm has considerably different paradigm than Spanning Tree. Spanning Tree breaks all the redundant paths, creating a unique loop-free topology. By the contrary, SCS assumes the network as looped and maximizes the redundancy provided and the investment made, complying also with the above specifications for Spanning Tree Protocol replacement, leaving bridges core function and bridging fundamentals intact.

"*If you know your enemies and know yourself, you will not be imperiled in a hundred battles*"

- The Art of War, Sun Tzu.

In order to replace Spanning Tree Protocol, its algorithm, weaknesses, misbehaviours and inadequacy over Ethernet networks need to be studied, analysed and attested experimentally. Accordingly, three chapters are fully dedicated to the Spanning Tree Protocols study, where all their characteristics, processes and multiple problems are exposed. Moreover, eight different typical network scenarios assess experimentally Spanning Tree Protocol's behaviour and performance.

The same way, *Self-Configurable Switches* protocol (SCS) is explained and all its processes and mechanisms are detailed. Challenged against Spanning Tree Protocol over the same lab setups, SCS response couldn't be more assertive.

Outline

The chapters are organized into five different parts, covering each one a different set of the thesis materials.

Part I covers Spanning Tree Protocols overview and analysis. Starting by the core reason for Spanning Tree existence on the *Ethernet Networks* chapter, the remaining chapters describe the functionality and associated problems of the Spanning Tree Protocol and its variants, Rapid Spanning Tree Protocol and Multiple Spanning Trees Protocol.

Part II presents the fresh standard proposals for Spanning Tree Protocols substitution, IETF's TRILL and IEEE's 802.1aq SPB.

Part III introduces the *Self-Configurable Switches Protocol* strategy and details all its processes, mechanisms, features and deployment plan.

Part IV exposes the way *Self-Configurable Switches Protocol* was simulated and the models used.

Part V starts with the SCS protocol overhead analysis, in terms of processor and link bandwidth requirements. Then, it presents the ultimate facts about *Self-Configurable Switches Protocol* feasibility to succeed Spanning Tree Protocols. It's a roll of cases where SCS overcomes the problems Spanning Tree Protocols face, one by one, either in theory, as in practical experimentations.

Finally, the *Conclusions* chapter presents the final conclusion regarding all the concepts, analysis and work performed throughout this thesis, besides some considerations about the future work, which would enhance current SCS features.

Part I – Spanning Tree Protocols

Part I covers Spanning Tree Protocols overview and analysis. Starting by the core reason for Spanning Tree existence on the *Ethernet Networks* chapter, the remaining chapters describe the functionality and associated problems of the Spanning Tree Protocol and its variants, Rapid Spanning Tree Protocol and Multiple Spanning Trees Protocol.

1 Ethernet Networks

This chapter presents a high level overview of Ethernet networks and the intrinsic need for some mechanism to *control* flooded Ethernet traffic over some network topologies. A basic comparison with Layer 3 protocols will be referenced, as well as the need for Ethernet bridges and the spanning tree protocol.

1.1 Ethernet Networks Overview

Ethernet concept was somehow controversial in the beginning; but even this chapter title can be!

Many still blame Metcalfe[1] for potentiate such a *not-so-good idea*. Well, the reality is that after 40-years from its start, Ethernet represents a several hundred billion dollar industry, continuously growing, developing and delivering super-fast communication channels. It's amazing its growth and curious the reason why our personal computers use technology invented before the *personal computer* itself. Even more curious is the fact that, in the *core* functionality, the concept of a 90's Ethernet adapter differs little from today's 1Gbps Ethernet adapter that we have in our computers [14]. Pretty amazing!

It is very common to talk about *Ethernet networks* features, weakness and potentialities. However, Ethernet is not a network; it's a *link*! Probably, the correct designation would be something like *Ethernet Links Network*. But then, this designation would be also controversial…

Well, taking this into consideration, this document will, though, continue to line up with the commercial designation of *Ethernet Networks*. It's easier, more comprehensible and aligns with Ethernet aura.

Let's start by the basics.

ISO OSI Reference Model

International Organization for Standardization (ISO) Open Systems Interconnection (OSI) Reference Model [11][12] describes seven abstraction layers, useful mainly to standardize the categorization of the communication systems functions or processes. Nevertheless, actual network protocols are far more complex than the abstraction performed by the OSI Model.

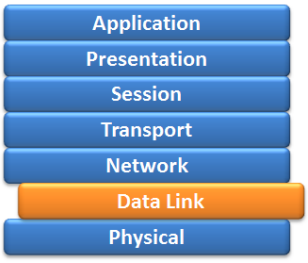


Figure 1-1 - OSI Reference Model

[1] Robert Metcalfe invented the Ethernet back in the 70's. Currently, is a Professor in the University of Texas at Austin's Cockrell School of Engineering.

The OSI model has many controversial interpretations, mainly for the upper layers. For the bottom four layers, the descriptions are mostly consensual. But as the name implies, the OSI Model is for *reference*.

For this document purpose, it is relevant the OSI Reference Model Layer 2. In basic terms, a Layer 2 protocol is mainly responsible for allowing neighbour nodes on a link to exchange data frames, maintaining the data link between two devices to enable communications. That's why it is referred as the *Data Link Layer*.

Ethernet: The need for Bridges

The concept of Local Area Networks (LANs) has evolved, especially over Ethernet. From now on, when referring LAN, it should be interpreted as LAN over Ethernet technology (802.3).

Based on CSMA/CD (Carrier Sense Multiple Access / Collision Detection), Ethernet has an intriguing addressing scheme: to connect only a small, typically near number of nodes, the standard defined a 48-bit address [9]. Comparing with IPv4 32-bit addressing scheme, which addresses ALL the IPv4 Internet, it seems odd!

The 6-byte address was initially thought to avoid configuring statically address when connecting systems into a network. Accordingly, hardware manufacturers purchased blocks of Ethernet addresses and hardcoded unique addresses into every device: the MAC address.

Using that ***flat addressing*** scheme, an Ethernet device could then be connected to any Ethernet network, without worrying about address collision and device addressing. A device would work, independently of its physical location, even geographical.

With the continuous need for connectivity, Ethernet networks became bigger and wider, emerging the need to merge different LANs together and for connecting systems beyond Ethernet electrical restrictions. That time, the concept of ***bridges*** was developed (transparent bridges), mainly to forward Layer 2 traffic between devices on different links.

Ethernet: Bridging what was supposed to be local

Like said before, Ethernet is a link. Consequently, Ethernet connectivity was supposed to be confined to a single link.

Bridges extended the *link* concept of Ethernet into *network*, where Layer 2 traffic is forwarded between devices. However, Ethernet was (is) not suitable to be forwarded… Nevertheless it's huge 48-bit addressing scheme, it lacks TTL or Hop Count to detect and discard looping packets, neither its address flatness allow referencing the device in the network, nor provide mechanisms to discover other devices and ways to reach them.

Forwarding Mechanism

The way bridges work is very simple: they promiscuously listen the incoming traffic, remember the source address of that traffic, and forward traffic based on that learning method. If the destination is unknown, the bridge *floods* the traffic throughout all ports, except the incoming one.

For non-redundant paths, this works just fine. However, in the case of multiple possible paths between any pair of devices, this was a huge issue.

As Ethernet does not have TTL/hop count, a loop will have exponential proliferation and the effect is devastating; it is mandatory to have a loop-free topology. Beth Israel Deaconess Medical Centre disaster [1] is an excellent example of how devastating and service impacting a loop can be.

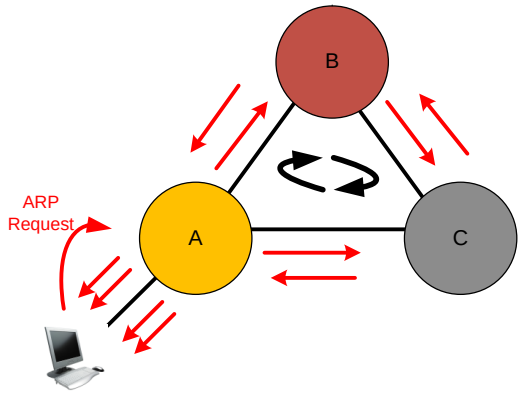


Figure 1-2 - Bridging Loop

If we have a looped topology, issuing a single ARP request would meltdown the network. Figure 1-2 illustrates the continuous ARP Request flooding, until resources exhaustion.

Intrinsic to the transparent bridging concept is another major threat to Layer 2 networks. Probably you already saw a Layer 3 loop. Have you faced any problem? Probably not, due to two main reasons: the TTL already discussed and the routing mechanism. In a Layer 3 loop, it is the SAME packet that is looped; in a Layer 2 loop, the *flood* frame is REPLICATED towards all ports (except the incoming one); this increases the traffic drastically and it is the main concern on Layer 2 redundant topologies.

"…bridges find a spanning tree."

I think that I shall never see
a graph more lovely than a tree.
A tree whose crucial property
is loop-free connectivity.
A tree that must be sure to span
so packet can reach every LAN.
First, the root must be selected.
By ID, it is elected.
Least-cost paths from root are traced.
In the tree, these paths are placed.
A mesh is made by folks like me,
then bridges find a spanning tree.
Radia Perlman: Algorhyme

It was in a form of poem, that a solution for Figure 1-2 issue arrived. *The Mother of the Internet*, Radia Perlman, developed the original Spanning Tree Protocol to enhance the bridging concept. This way, the network could have any kind of physical distribution that bridges would process it and create a loop-free topology. Layer 2 traffic could then be forwarded, without being looped forever. All bridges needed to do were decide which ports should be forwarding and which should be blocking, in order to create the loop-free connectivity.

Figure 1-3 pictures the result of a spanning tree algorithm, with bridge B blocking its port towards bridge C, in order to open the loop and create the desired loop-free topology.

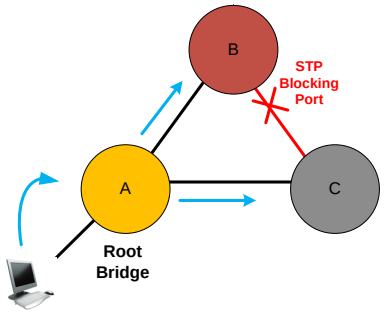


Figure 1-3 - Loop-free Topology

All characteristics and issues about the standard spanning tree protocols will be detailed in future chapters.

Replacing Bridges by IP Routers

So, forwarding Layer 2 traffic over Ethernet links via bridges is not a good idea. How about replacing all bridges with IP routers? IP traffic has TTL and its addressing is not flat…

Bridges forward traffic directed to a device (MAC address). IP routers forwards traffic directed to links.

Each link IP has its own addressing scheme. A device can be connected to several links, requiring only one different IP address from each different address block. If the

device moves into a different IP link, readdressing must occur in order to comply with the new IP link addressing range. Conversely, Ethernet devices cannot connect to several links and if they move, the MAC address will remain the same independently the device had travelled half-way around the globe to be connected on a new Ethernet link five-thousand kilometres away from the original one.

Therefore, we have wasted 48-bit in a local addressing scheme and just 32-bit in a global one. The 48-bit one is flat; the 32-bit is not enough for today demands. Hence, it is still popular to create large bridged networks, as a bridge set of links looks to IP like a single network.

Take into consideration that this *link routing* property is exclusive of IP and it's not common to all Layer 3 protocols. For example, OSI CLNP uses 20-byte address scheme and routes traffic to *areas* with lots of links, instead of single links like IP does. Within the area, the end devices announce themselves to routers and the routing is performed directly to the end device. Seems better than IP and it is indeed, but because of several *movements*, IP was the protocol of choice in the past, until today.

2 Spanning Tree Protocol

Transparent bridging, by definition, does not provide any mechanism for aging out frames, like the TTL field in IP packets, allowing in some circumstances specific type of frames to be bridged forever in a looped Layer 2 topology. Therefore, frames with unknown or undefined specific destination address, like broadcast, multicast or unicast flooded frames, will be looped forever, continuously consuming links bandwidth and bridges resources until exhaustion and, consequently, meltdown the network in a few seconds.

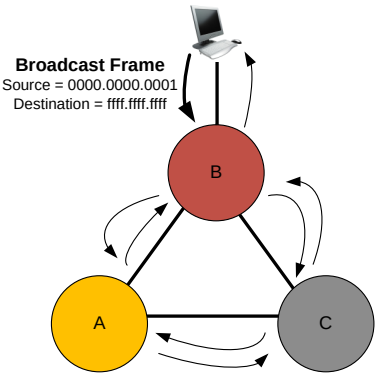


Figure 2-1 - Looped layer 2 topology

Figure 2-1 illustrates this continuous flooding and the real need for some mechanism to prevent loops in Layer 2 topologies, while still providing some degree of redundancy.

Back in the eighties, the Spanning Tree Protocol – STP – was the answer.

Spanning Tree Protocol was originally developed by DEC in 1983 to address the need for loop free Layer 2 topologies. Based on DEC STP implementation, IEEE specified the 802.1D standard, initially developed by Radia Perlman.

2.1 Loop Free Topology

Spanning tree objective is to build a loop free Layer 2 topology, sourced at a *main* bridge designated by ***Root Bridge***, and spanning to all other bridges in the network. The operation is quite simple, and STP only has to determine if the bridge port should be forwarding or blocking frames towards it.

In 802.1D IEEE standard, only a single common loop free logical topology is built. However, some vendors have implemented some variants of the standard and, for example, Cisco Systems allows the existence of one spanning tree per VLAN, meaning that a port might be blocking for some specific VLANs and forwarding for another. Those variants aim, mainly, the possibility of load balance and segregate traffic, which Spanning Tree Protocol didn't have natively.

In Spanning Tree Protocol, the *Root Bridge* is the reference point of the loop free topology. Once the Root Bridge is elected, each bridge will determine the best path towards it and any other paths to the root are blocked, in order to prevent loops. In a converged topology, Spanning Tree ports are either in *forwarding* or *blocking* state.

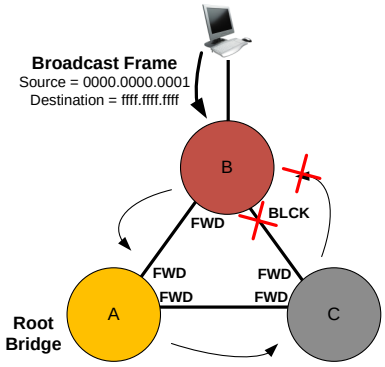


Figure 2-2 - STP converged topology

Figure 2-2 shows the converged state of the topology where link B-C @ B side was selected by Spanning Tree to be in the *Blocking* state. With that port *blocking*, the STP opened the loop that existed in the Layer 2 network. That way, the broadcast sent by the PC, travels through all the Layer 2 topology but never returns to the original bridge B again, breaking the loop.

The normal port state transition for a port is *Blocking* → *Listening* → *Learning* → *Forwarding*. It is important to note that a new port never goes via the *Blocking* state; it goes directly to *Listening* state. For example, when you attach a PC to a network, that bridge port starts at *Listening* state; the *Blocking* state is restrict for ports that might allow the occurrence of a Layer 2 loop and Spanning Tree forced it to that *special* state.

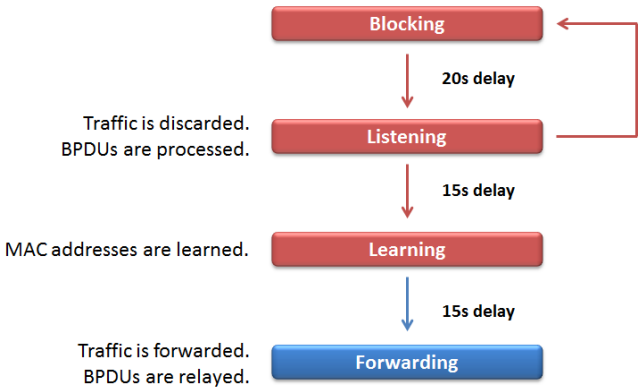


Figure 2-3 – STP port states

The *Listening* state is one of the most important states in the process of building a topology, as it is in this state that the bridges decides *what* to do with the port, as it receives and processes BPDUs (Bridge Protocol Data Units). This state allows Root Bridges to be elected and ports to be defined as *root*, *designated* or *non-designated* ports.

The *Learning* state is just to limit the flooding of traffic after transition to *Forwarding*, due to unknown unicast destination MAC addresses.

The *forwarding* state, as the name implies, is the state where traffic is effectively forwarded through that port.

Figure 2-3 shows the possible STP port states and transitions.

BPDU

Bridges exchange Bridge Protocol Data Units (BPDU).
The two types of BPDUs are *Configuration* and *Topology Change Notification* (TCN).
Figure 2-4 illustrates the 802.1D BPDU format.

Configuration BPDUs are sent every 2 seconds from the root bridge, by default, to:

- Be used during an election.
- Allow bridges to maintain connectivity with other bridges.
- Inform all bridges about root bridge's timers.

TCN BPDUs are sent by a bridge towards the root when:

- There is a link failure.
- The bridges receives a TCN from a neighbor.
- A port starts forwarding and there is already a designated port.

Figure 2-4 - 802.1D BPDU format

Root Bridge Election

As said before, Spanning Tree Protocol defines a reference point for the loop free topology, named ***Root Bridge***. In order to determine the Root Bridge, Spanning Tree selects the *best* bridge in the campus, after an election between all the bridges. It's fairly democratic!

Each bridge is uniquely identified by a Bridge ID, which consists of a 2-byte Bridge Priority plus the Bridge MAC address.

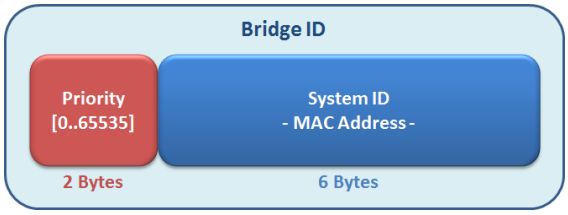


Figure 2-5 – STP Bridge ID

Bridge Priority can be customized to any value between 0 and 65535. Lower priority value means better Bridge ID value.

This Bridge ID value is used to elect the Root Bridge, which will be the one with the *lowest Bridge ID*. The election process starts by each bridge assuming it will be the root, beginning to send Configuration BPDUs. When a bridge receives a Configuration BPDU, it compares the Bridge ID received with the one it has stored as being the root and if it is lower, it immediately considers the new bridge as the root and propagates that new info to its neighbours.

In the following Figure 2-6, bridge A will be elected the Root Bridge, as its priority value 16384 is the smallest one within the campus, resulting in the lowest Bridge ID.

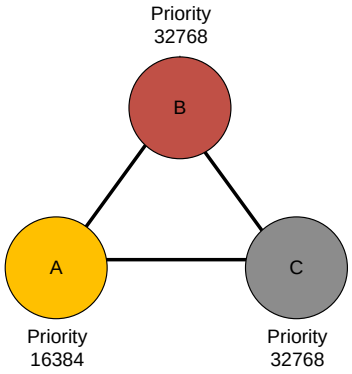


Figure 2-6 – Bridge A is elected Root Bridge due to its lower priority value

The loop free topology is built taking into consideration the *lowest cost path* to the Root Bridge. The concept of *cost* is a measure of preference of a link and it is inversely proportional to the link bandwidth. Lower the cost, more preferable is the link. This value can be customized, in order to influence the root path choice.

Besides Root Bridge, the election process also defines the root ports, designated ports and non-designated ports.

The following sequence of events lists the steps that may be taken to define the root bridge and all those bridge port roles:

1. Lowest root Bridge ID
2. Lowest root path cost to Root Bridge
3. Lowest upstream Bridge ID
4. Lowest upstream Port Priority
5. Lowest upstream Port ID

Spanning Tree Timers

There are three important timers on STP:

- **Hello Interval** - Time between each BPDU sent on a port. By default, 2 seconds.
- **Forward Delay** - Time spent in Listening and Learning states. By default, 15 seconds.
- **Max Age** - The maximum time a BPDU is considered valid. By default, 20 seconds.

Those timers are set by the Root Bridge and inserted in the Configuration BPDUs, in order for all other bridges to sync their timers with those set by the Root Bridge. Previous Figure 2-4 illustrated in orange the timer fields on the 802.1D BPDU frame.

Besides those three timers, there is another one with an important role in STP convergence, the ***Message Age***. The **Message Age** timer is not fixed; it's the length of time that passed since the Root Bridge initially originated the BPDU. In fact, it measures the distance to the root from a bridge. The Root Bridge sends BPDUs with message age at 0 and each bridge increases the message age by 1.

When a bridge receives a BPDU, it stores the information contained in the BPDU if the BPDU is equal or better than the info already recorded on the port. In that instant, the age timer begins to run starting at the **message age** value received in that BPDU. If the age timer reaches the **Max Age** before receiving another BPDU, the information is aged out for that port. As the distance from the Root Bridge increases, it decreases the timeout interval for the BPDU information: a bridge six-hop away will timeout in 14 seconds ($MaxAge - MessageAge$).

2.2 Topology Changes

In general, the default aging time for forwarding databases entries are, in most platforms, 300 seconds - 5 minutes. Only after a quiet period of 5 minutes the forwarding database entry will age out. This could bring enormous problems for a Layer 2 network, because if one link in the forwarding path fails, there would be a black hole of 300 seconds in communication, until the forwarding databases timeout.

To cope with this situation, Spanning Tree Protocol uses the *Topology Change Mechanism*. As soon as a bridge detects a change in the topology, all other bridges are notified of that change and reduce the aging time of their forwarding databases to Forward Delay (15 seconds by default) for a period of time of $MaxAge + ForwardDelay$ (35 seconds by default).

Spanning Tree considers a topology change when the following events occur, independently of the *type* of the port:

- A forwarding port goes down.
- When a port transitions into the forwarding state.

This means that even when a user connects or disconnects its portable laptop to the network, the bridge considers it a topology change and will notify the whole network about it. Although, it is important to note that this topology change does not start a Spanning Tree calculation; it's only a notification mechanism to inform all other bridges that some port started or stopped forwarding frames and only has impact on the aging time of the bridge's forwarding databases.

When a bridge detects a topology change, starts a notification process intended to inform all other bridges of this change. This Topology Change Notification process involves two steps:

1. The bridge notifies the Root Bridge of the spanning tree via TCN BPDUs. The Topology Change Notification BPDU is a special BPDU that a bridge sends via its root port, towards the Root Bridge. That TCN is acknowledged back by all *upstream* bridges and relayed via upstream bridges root ports. The process continues until the TCN reaches the Root Bridge. This step is pictured in Figure 2-7.
2. The Root Bridge *broadcasts* the information into the whole network. When the Root Bridge receives the TCN, it starts sending it's Configuration BPDUs with the TC (Topology Change) bit set, during $MaxAge + ForwardDelay$ (35 sec by default). This way, all bridges get knowledge of a topology change that happened *somewhere* in the network and change their aging time to $ForwardDelay$ (15 sec by default). Figure 2-8 illustrates this step.

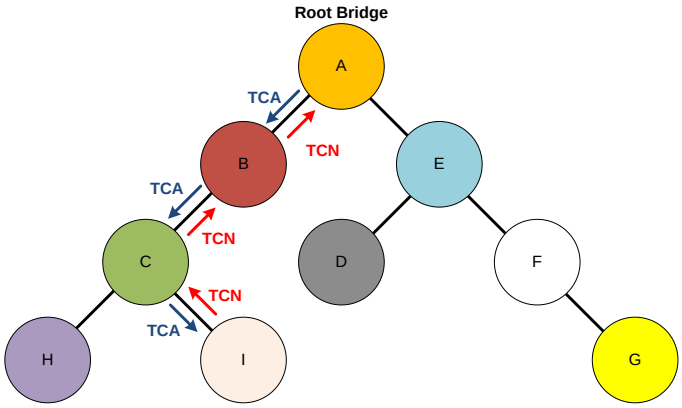


Figure 2-7 – Bridge I notifies the Root Bridge A about a topology change

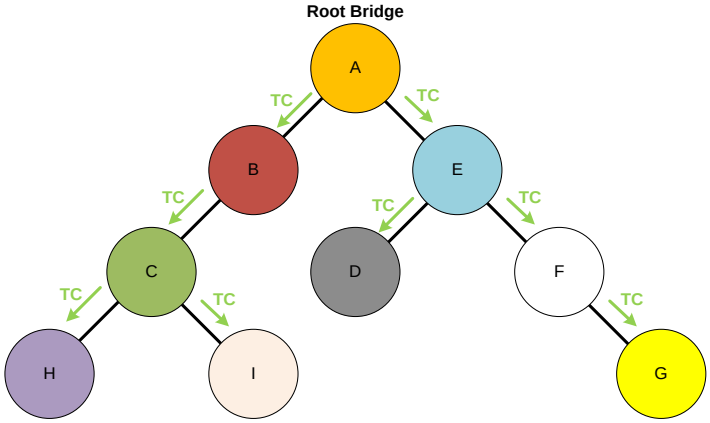


Figure 2-8 – The Root Bridge A announces the topology change to all bridges

Using this two-step process, the entire network get knowledge of a topology change that occurred and all bridges reduce their forwarding databases aging time to 15 seconds, by default.

2.3 Convergence Performance

In the previous section the topology change mechanism was analysed. As it was said, a simple laptop plugged in or unplugged from the network would trigger a topology change notification that would cause aging time adjustments, not a spanning tree calculation.

However, there are events that would start also spanning tree calculations, either by link failures or link insertions into the topology.

Link Failures

There are two types of failures: direct and indirect.

Direct fails are those related with physical links failures and the convergence time is, by default, 30 seconds, as it represents two times the forward delay timer, summing up the time spent in the listening and learning states.

Indirect failures are those not related with physical issues. Indirect failures are detected by the expiration of Max Age timer, which is by default 20 seconds. So, the convergence time for an indirect failure is 50 seconds by default: Max Age 20 seconds plus the port transition from listening state into forwarding, meaning, plus 30 seconds.

Spanning Tree convergence times, 30 seconds for direct failures and 50 seconds for indirect failures, are completely inadequate to modern switching networks. These kinds of limitations and others will be analysed later more in-depth, in the *Spanning Tree Protocol Problems* section.

Link Insertion

The insertion of a new link in a network, or even the recovery of a link that went down, can be extremely problematic in STP.

The *Spanning Tree Protocol Problems* section investigates this issue in detail, but in sum, the insertion of a link between two bridges in a Layer 2 topology can create a 30-second black hole in the network, using the default timers.

2.4 Spanning Tree Protocol Problems

The Spanning Tree Protocol targets the assurance of a loop free Layer 2 topology via BPDU exchange between bridges. A Root Bridge is elected and port roles are defined for every bridge port on every bridge of the campus.

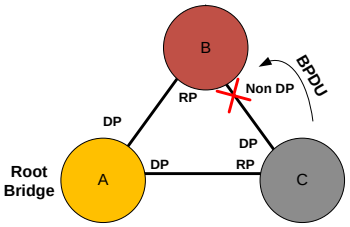
Spanning Tree Protocol, by nature, it's a ***loop preventive*** protocol; not ***loop detective***. This means that STP prevents undesirable loops in the network, by managing bridges links, but it does not have inherently effective loop detection mechanisms. In summary, if a loop occurs in the network, in the majority of the occurrences, it's not STP the one that will *break* it. Moreover, in some circumstances, are STP failures in loop prevention the responsible for loops to occur, and since it does not have loop detection mechanism, the loop persist in the network.

If, by some means, the STP fails, the result may be catastrophic as the Layer 2 network is no longer loop free, with the well-known consequences of this for the

network connectivity. Additionally, troubleshooting these events can be extremely difficult, increasing exponentially with the complexity of the Layer 2 network topology.

Almost all of the STP issues are related with some bridge port that should be blocking but, instead, is forwarding traffic. Most of the times, this is due to loss of BPDUs, causing the port transition from STP Blocking to STP Forwarding state. This can easily lead to bridging loops.

It is important to note that a bridge port remains in the STP blocking state if it continues receiving *superior* BPDUs from the neighbour bridge via that port. If it stops receiving those BPDUs, it will start the process for transition to the STP Forwarding state, i.e., after Max Age + 2x Forward delay, by default in 50 seconds, the port will be forwarding traffic, causing a bridge loop.



Bridge C has *superior* BPDU than bridge B. Bridge C continues to send BPDUs advertising its superiority over Bridge B. As long as bridge B continues to receive these *superior* BPDUs on port to C, the port remains in the STP Blocking mode.

Figure 2-9 – STP loop free topology

If we consider STP's BPDU exchange as a *keepalive* mechanism, and it is indeed as keepalives (BPDUs) are used to maintain a protocol state, this STP behaviour is really disappointing and effectively *enables* the creation of Layer 2 loops. I tend to designate it as ***STP loop enabler feature***: in the event of a failure on the *keepalive* mechanism, traffic forwarding by a bridge port is allowed, creating Layer 2 loops. This completely the opposite of the purpose of a *keepalive* mechanism! If a bridge stops receiving *keepalives* on a port, that should mean a failure and that port must be considered *not trusted*, and consequently *not good to forward traffic*. Nevertheless, STP behaviour is completely the opposite, where a failure in the keepalive mechanism triggers a bridge port transition to the STP Forwarding state and, probably, the network to meltdown.

There are some well-known issues that cause loss of BPDUs and consequently induce STP failures and Layer 2 loops. This chapter will focus on the most common of them, split in two major categories: *configuration-driven* and *system-driven* STP issues.

Besides loss of BPDUs issues, there are other situations where STP behaviour is not adequate. Those STP issues are not operational; they are design and performance issues. As it will be easier to see, by design, STP is not suitable for all types of Layer 2 topologies, due to several limitations. Those situations were labelled *protocol-driven* issues.

Two network standards arose in order to overcome some of those many weaknesses of STP: Rapid Spanning Tree Protocol (RSTP) and Multiple Spanning Trees Protocol (MST). Those enhanced Spanning Tree protocols will be analysed in chapters 3 and 4, respectively.

Configuration-driven STP Issues

There are common STP issues driven directly by human configuration activities.

The following scenarios describe some situations where misconfigurations lead to STP loops:

- Tuning the STP parameters to achieve better performance
- New Bridges in the Layer 2 production network
- Duplex Mismatches
- Temporary loops

The human factor is one of the major's contributors for network instability. Misinterpreted or defectively implemented configuration procedures and abusive network resources use and allocation can be considered the top contributors for Layer 2 network instabilities.

Following is the individual analysis of all the above situations and their contribution to the advent of Layer 2 loops.

Tuning the STP parameters to achieve better performance

STP convergence and stability is supported on three important timers: Hello Interval, Max Age and Forward Delay, as stated before. Furthermore, there is another very important concept in STP networks that is correlated with STP timers and which is crucial for STP convergence and stability: ***bridge network diameter***.

The *bridge network diameter* restricts the distance from each other bridges can be. This is mainly due to the age field in the BPDUs.

The IEEE recommendation is to consider a maximum diameter of seven bridges, for the conservative default STP timers, meaning that two bridges cannot be away from each other, more than seven hops. If the Root Bridge is too far away from some other bridge, its BPDUs may not reach that bridge and STP convergence and stability would be affected.

Beth Israel Deaconess Medical Centre disaster described on the *Motivation* section of the *Introduction* is a phenomenal example of potential consequences when the maximum diameter recommendation is disregarded.

It is very important to understand that changing STP timers causes the network diameter to change, and more aggressive STP timers mean shorter diameters.

CPU spikes, application slow-ups, instabilities, convergence issues and network meltdown are some of the effects that could arise when misconfiguring the STP timers. Trying to achieve faster convergence via STP timers tuning is dangerous and dependent on the network topology. A simple link failure could change the network topology and some extra unexpected hops may be introduced between bridges, and associated with the non-optimal paths STP may have, the initial projected distances between bridges can easily became deprecated and stability and convergence of the network may be in risk. Besides, as hello, forward and max age timers are carried out in BPDUs, only those timers configured on Root Bridge are relevant; if the Root Bridge is lost, the timers configured in the new Root Bridge will be taken into consideration and the previous tuning made is lost.

New Bridges in the Layer 2 production network

There are two possible, distinct situations about bridges insertion on network that could affect the STP Topology: abusive insertion or planned but careless insertion. Besides the purpose of the insertion, the effects are the same: instability and network connectivity losses.

The planned but careless insertion is a threat, but well trained network administrators have knowledge of the possible consequences and the risk of bridge insertion can be minimized with careful planning and operational procedures.

Regarding abusive insertion, this presents a greater threat to networks. Most of the bridge interfaces are to be used as *host* interfaces, meaning that they are purposed to provide network connectivity to end-user equipment, like personal computers, printers, IP phones, etc. Sometimes, end-users like to expand by their own the network port capacity available to them, inserting bridges on the network, using the initial *host* interface to provide main network connectivity. What they probably don't know is that these elements will be active speakers in the production network.

In both cases, abusive and careless, new bridge insertion into a production network could lead to network topology change and, in worst cases, new Root Bridge election. As we can see in Figure 2-10, the STP initial topology in red on picture 1 will change drastically into the one of picture 2, just because of the unauthorized/careless insertion of bridge Bx in the network.

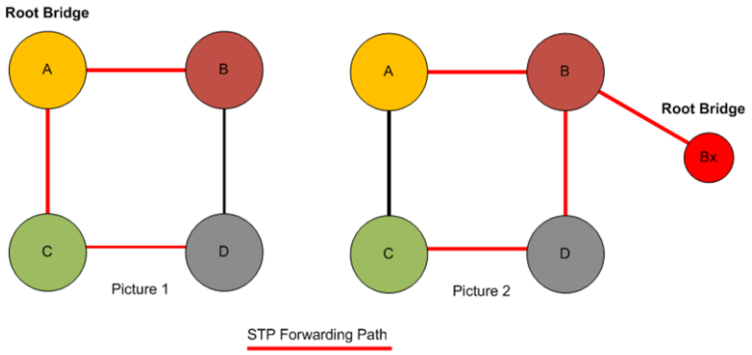


Figure 2-10 – STP topology change because of Bridge Bx

Duplex Mismatches

On a CSMA/CD network, when using the full duplex mode, the CS (carrier sense) mechanism is disabled, meaning that a port in full duplex mode no longer negotiates the "*access*" to the medium. On the other hand, a half-duplex port uses the CS mechanism to assess the link access and if it detects a collision, runs the *backoff algorithm* before attempting another transmission of the frame.

If you fall in a scenario where you have a full duplex port *talking* with a half-duplex one, the full duplex port may induce link access starvation into the half-duplex one. This causes the bridge which has the half duplex port not being able to send traffic to the other bridge, including BPDUs.

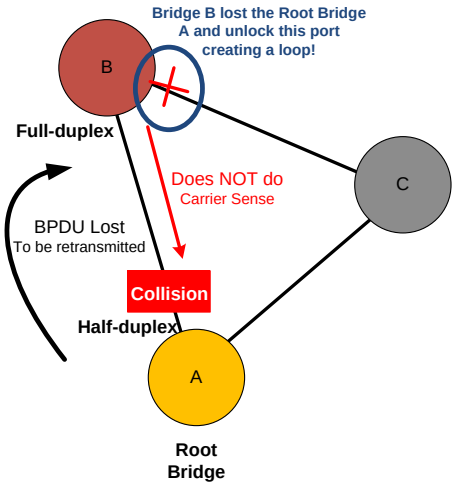


Figure 2-11 – Duplex mismatch issue

Note that from Figure 2-11, for example, after bridge B unlocking the port to C, a broadcast that arrives from a bridge C host, is propagated to bridge A and bridge B. Bridge A cannot send it to bridge B as cannot access the medium, but as bridge B communication to bridge A is working, the broadcast arrives again to bridge A via bridge B. Bridge A then forwards again the broadcast into bridge C and a loop is formed: the broadcast is propagated forever from bridge C → bridge B → bridge A → bridge C.

The *Duplex Mismatches* is a particular case of a more general situation: unidirectional links. The unidirectional *state* of a link is a great menace to Layer 2 topologies as it easily induces Layer 2 loops. This general situation will be analysed in the section *System-driven STP Issues*.

Temporary loops

Temporary loops will be always possible to occur, in any network running any protocol. However, a protocol must be prepared to deal with them. Clearly, it's not STP case. With Cisco *Portfast* enabled, for example, it is pretty simple to create a temporary loop. Moreover, *portfast* demonstrates the weakness of STP state machine in terms of convergence times.

Portfast is a Cisco proprietary feature that allows STP to directly transition a port into the Forwarding state, bypassing the Listening and Learning STP states. Although proprietary, it was somehow included natively in the Rapid Spanning Tree protocol standard by *edge-ports* designation.

Portfast can be so dangerous to STP that Cisco warns you every time you enter the *portfast* command on Cisco's CLI:

```
   %Warning: portfast should only be enabled on ports connected to a
single host. Connecting hubs, concentrators, switches, bridges,
etc... to this interface when portfast is enabled, can cause
temporary bridging loops.
Use with CAUTION
```

Figure 2-12 - Portfast: Cisco CLI warning

If it so risky, why should someone use it? Furthermore, what's the reason for the STP successors *Rapid Spanning Tree* and *Multiple Instances Spanning Tree* embed *portfast* natively?

The answer is in the STP timers: by default, as we seen before, *Max Age* is 20 seconds and *Forward Delay* 15. A port to transition to STP Forwarding state takes 2x Forward Delay, one for the Listening and another for the Learning states, i.e, by default, 30 seconds.

30 seconds in modern computer networks it's unbearable! With *portfast*, in a few milliseconds the port is forwarding traffic.

With *portfast* enabled, as a port is immediately inserted in STP Forwarding state, it can introduce transient loops. Figure 2-13 illustrates the loop created by the *portfast* feature.

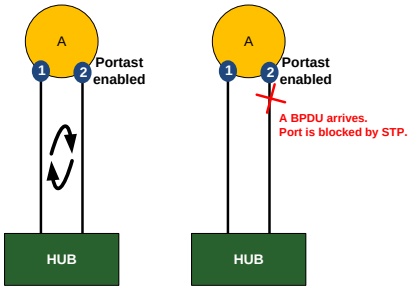


Figure 2-13 – Portfast temporary loop, after connecting port 2 to the hub

As soon as the "new" port 2 receives a BPDU, it transitions to STP Blocking state and, if that's the case, the loop is broken. However, sometimes this could not be the case.

The BPDU cadence is 2 seconds, by default. In the worst scenario, there will be a 2-seconds loop and the effect of that loop depends on the looped traffic; if the traffic is intensive, that could affect the bridges resources and BPDU transmission may be in check, meaning that the BPDU that should have solved the transient loop may not be transmitted, and the loop persists.

In fact, *portfast* reduces drastically the time a port should wait until it is allowed to forward traffic into the network and sometimes it is crucial to have it configured, in order to the services running on host machines to work properly, like DHCP requests. However, it could be extremely dangerous in some situations.

System-driven STP Issues

Besides issues driven directly by human configuration activities, there are some situations where system events could trigger loss of BPDUs and Layer 2 loops.

The following scenarios describe some situations where system events lead to STP loops:

- Software / Hardware errors
- Resource exhaustion
- Unidirectional links

We will analyse all these situations next.

Software/Hardware errors

Every software and hardware device has errors and the effects of them cannot be adequately described. Sometimes they are random, increasing the difficulty of prediction and identification.

For STP, the most dangerous situations are those that may affect BPDU transmission, meaning a port that should be blocked will be forwarding.

These kinds of errors are not specific of STP but the above mentioned *STP loop enabler feature* helps to leap the effects of those errors into even more drastic situations.

A Tx link failure should bring a neighborship down, not the entire network!

Resource exhaustion

For modern bridges, STP is not processor-intensive and it should be prioritized over other processes. However, if for any reason the bridge's CPU is overloaded, there could be not enough resources for the transmission of BPDUs.

Unidirectional links

This situation is similar to the *Duplex Mismatches* aforementioned but, in this case, it is triggered by some system events.

The unidirectional *state* of a link becomes a great menace to Layer 2 topologies because it easily induces loops.

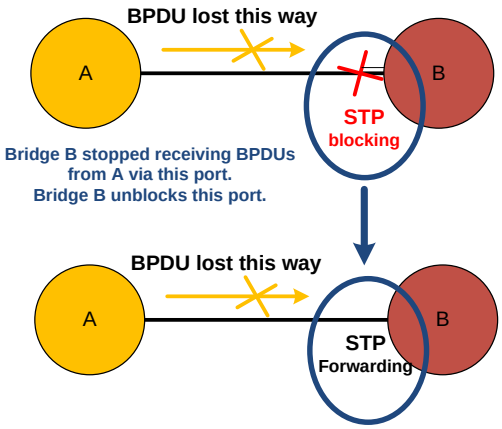


Figure 2-14 – Unidirectional link from bridge B to A

Taking into consideration that, if bridge B stops receiving BPDUs from bridge A, it will transition the port into STP Forwarding state and starts forwarding traffic, in certain topologies a loop will occur, like Figure 2-15 demonstrates.

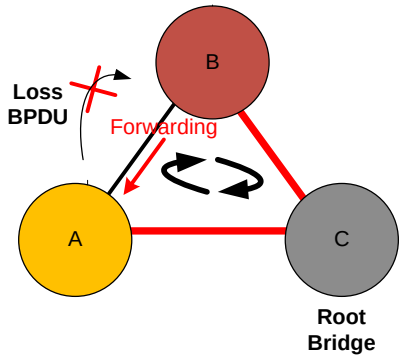


Figure 2-15 – Unidirectional link causes loop

There are several situations that could result in unidirectional links, like transceiver problems, duplex mismatch (already analysed before) and fibre optical patch failures (Tx @ bridge A in the above example).

Natively, STP does not have a mechanism to deal with these situations and a simple link failure or hardware defect could bring down an entire Layer 2 domain.

Protocol-driven STP Issues

STP is not suitable for all type of topologies for a vast list of limitations. The following, presents the most important STP characteristics that contribute for protocol-driven issues:

- Conservative STP timers
- Network diameter
- Least cost path
- Topology change mechanism
- Convergence black hole

If those characteristics are not considered when designing the Layer 2 network, unpredictable behaviour, waste of resources and potential instabilities may occur.

Conservative STP timers

The conservative timers do not allow a Layer 2 network to have fast convergence. In the presence of topology changes, network is limited to Forward Delay and Max Age timers, meaning, convergence times close to 50 or 30 seconds.

In modern networks, these times are unacceptable.

Tuning the STP timers, as explained before, has several drawbacks and must be carefully planned.

Network diameter

The IEEE recommendation is to consider a maximum diameter of seven bridges, for the conservative default STP timers.

For large Layer 2 domains, this might be a major limitation.

The following figure illustrates a carrier Ethernet topology, where STP is unsuitable:

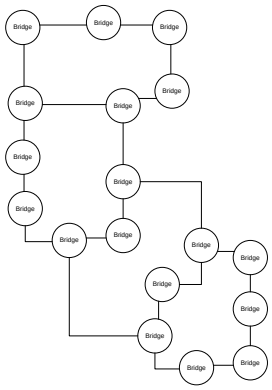


Figure 2-16 – Carrier Ethernet network topology

Least cost path

The Spanning Tree Protocol calculates a single *spanning tree* where messages from any bridge to the Root Bridge traverse a least cost path, among all paths from that bridge to the root.

The STP calculations are performed taking into consideration the communication towards the Root Bridge.

How about the communication *between* non-Root Bridges? They aren't for sure least cost paths. Figure 2-17 illustrates this problem.

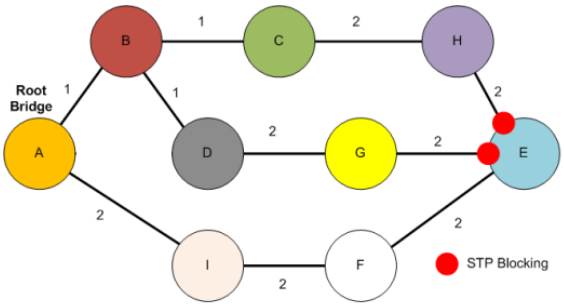


Figure 2-17 – STP sub-optimal paths

These sub-optimal paths cause unexpected over utilization of network resources, like link's bandwidth and bridge's switching capacity. For example, in the above example, traffic from E to G unnecessarily crosses five bridges and six links. Moreover, all communication between bridge E and any of the other 8 bridges of the network will cross the single link E-F.

So, least cost paths towards Root Bridges may be in fact high cost paths for network resources, as they tend to be sub-optimal paths.

That's why a good Layer 2 design must carefully study the placement of the Root Bridge.

Besides potential *high cost paths* for network resources, STP disallow traffic ***load balancing***. In the above example, bridge E has two equal cost paths towards bridge B: E–H–C–B and E-G-D-B, but STP forces bridge E to use a single higher cost path, instead of potentially using those two equally least cost paths via bridges H and G.

Topology Change Mechanism

In the above *Topology Changes* section, the notification process of changes was analysed. We saw that in the advent of a TCN, every bridge reduces the forwarding database's aging time to Forward Delay (15 seconds by default) for a period of time of $MaxAge + ForwardDelay$ (35 seconds by default).

In very large networks, we can have the network permanently in topology change status, as any bridge port change could trigger a TCN, even a host port. This would mean that the aging time will be always in 15 sec and there may be massive flooding in the network.

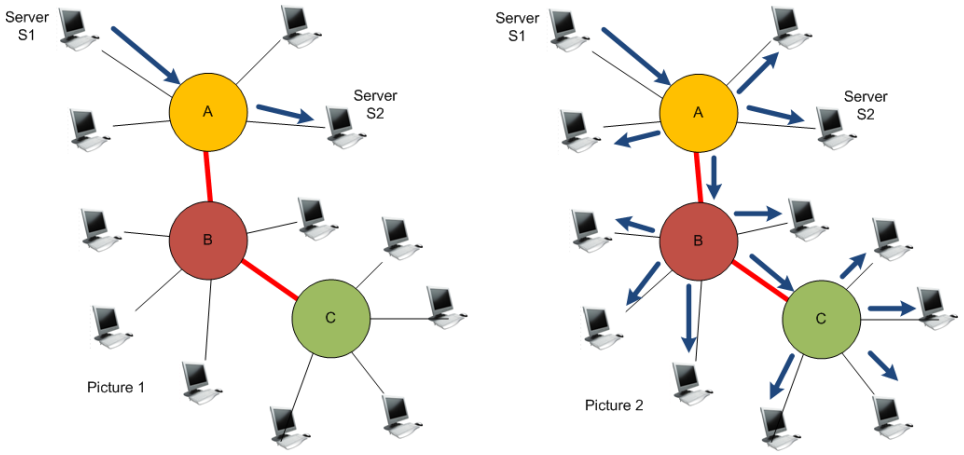


Figure 2-18 – Topology Change state issue

In some traffic cases, that's a huge problem.

Suppose server S1 on Figure 2-18 is sending unicast traffic to server S2, and that traffic is *heavy*. Due to the nature of that traffic, server S2 is only *receiving* traffic; it almost never sends traffic towards S1. In picture 1 of Figure 2-18, S1 unicast traffic is sent over S2 port only. However, if a topology change occurs in the network, bridge A will reduce the aging timer to 15 seconds, by default, and by so, it will eventually age out entry for S2. Picture 2 illustrates the effect: until S2 sends a frame again, S1 *heavy* traffic is flooded throughout the network consuming unnecessarily links bandwidth, bridges and other systems resources.

Convergence Black Hole

In STP, the insertion of a link between two bridges in a Layer 2 topology can create a 2x Forward delay black hole in the network, due to… fast convergence of a bridge! It is indeed curious the fact that the fast convergence below 1 second on one bridge, creates a big impact on network connectivity.

Figure 2-19 illustrates this problem.

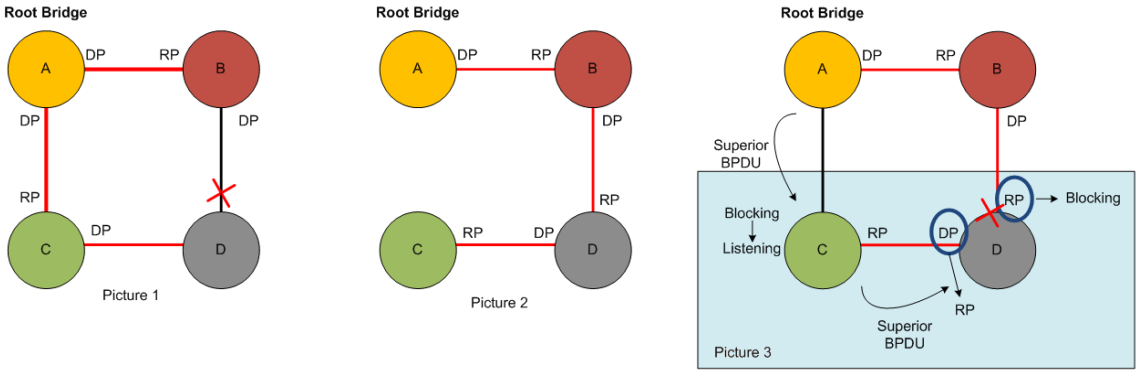


Figure 2-19 – Fast convergence creates black hole

In Figure 2-19, the original topology has bridge A as Root Bridge and port D-B in blocking state. If link C-A fails the topology converges into the one presented in picture

2. As soon as the link C-A is restored, after a few seconds the network faces a black hole of 30 seconds!

The reason lays on the fact that after a few seconds, bridge C places the port C-A into Listening state and processes the superior BPDUs it receives from the Root Bridge A. Those BPDUs are relayed into bridge D that puts immediately (fast convergence @ bridge D) port D-B into Blocking state; that change isolates bridges C and D from the network, during 15 seconds for passing from Listening to Learning state, plus more 15 seconds for changing from Learning to Forwarding state at bridge C. Until bridge C transitions port C-A to Forwarding state, bridges C and D remain isolated from the rest of the network.

RSTP overcomes this issue, because it reengineered the way it handles topology changes and notifications. RSTP uses the Proposal-Agreement process, as we will see in the next chapter.

3 Rapid Spanning Tree Protocol

Rapid Spanning Tree Protocol (RSTP) supersedes the Spanning Tree Protocol standard 802.1D (STP) and introduces several changes to the way spanning tree works, mainly to optimize convergence times and increase stability. It can be seen as an *evolution* of the STP standard more than a *revolution*.

The new IEEE 802.1D-2004 standard [7] incorporates the standard Rapid Spanning Tree Protocol (IEEE 802.1w) and obsoletes original standard IEEE 802.1D Spanning Tree Protocol.

RSTP uses the same criteria as STP to calculate the final topology for the spanning tree and, for so, it can interoperate with legacy STP bridges on a per-port basis. This effectively drops the benefits it introduces, as described in *RSTP Protocol Degradation* section.

RSTP changed slightly the BPDU format and the way it handles them, providing significantly faster spanning tree convergence after a topology change, near 6 seconds for indirect failures (3x hello interval) by default, and few milliseconds for direct failures. However, in some specific situations RSTP exhibits the classic ***count to infinity*** behaviour and those convergence times are far away from the ones advertised. Moreover, in some network topologies, ***port role negotiation*** can also contribute to several seconds' convergence times. Those two situations will be analysed in the section *Rapid Spanning Tree Problems*.

Classic STP timers, such as Forward delay and Max Age, are only used as a backup and should not be necessary if ***point-to-point*** links and ***edge ports*** are properly identified and set. *Edge ports* are ports directly connected to end stations and *point-to-point links* are links between bridges operating in full-duplex mode.

Furthermore, RSTP delivers natively some features very close to some proprietary mechanisms that were developed to optimize the original STP, like Cisco *PortFast*, *UplinkFast*, *BackboneFast* and *BPDU Guard*.

In order to speed up convergence and detach the concepts of *state* and *role* of a bridge port, RSTP, besides designated and root ports, uses two new port *roles*, ***Alternate*** and ***Backup*** ports:

- ***Alternate port*** is a port in blocking state which receives superior BPDUs from another bridge.
- ***Backup Port*** is a port in blocking state which receives superior BPDUs from the same bridge.

3.1 Loop Free Topology

Recalling the STP process, every non-root bridge accepts and stores the *best* root bridge information. Using this info, the bridge elects one root port towards the Root Bridge and blocks all other alternate paths to the root. That best information received is then relayed *downstream* to all other bridges. If a bridge learns *superior* information about a Root Bridge, on any of its ports, the current information stored is erased and the new one will be immediately accepted and relayed.

In summary, STP bridges calculate a single best path towards the Root Bridge and store the best, most recent BPDUs received by every port (even on blocked ports), relaying that best information *downstream* to other bridges; *inferior* BPDUs are ignored. Bridges store BPDUs information during Max Age seconds and any transitions from Blocking to Forwarding take 2x Forward Delay, in order to flush MAC Addresses tables and synchronize topology change in all bridges.

Rapid Spanning Tree changes the behaviour over BPDUs processing:

- Every bridge sends its own BPDUs, every Hello Interval. In STP only the Root Bridge send BPDUs and non-root bridges simply *relay* them.
- RSTP does not only stores the *best* information received on BPDUs, but also takes into consideration *alternate paths* to the Root Bridge, which it will use when the primary path fails.
- Any change in Root Bridge information, on any port, must be synchronized with all *downstream* bridges, using the *proposal/agreement* mechanism. Even on blocking ports. This process is named ***RSTP Sync Process*** and guarantees a loop-free topology.

RSTP Sync Process

Any change in Root Bridge information must be negotiated between bridges, via the *proposal/agreement* mechanism of the RSTP Sync Process.

Point-to-point links are mandatory to RSTP sync process on designated ports, in order for the proposal/agreement process to be taken only by two parties; synchronizing several bridges on a shared segment would be excessively complex. RSTP can only achieve rapid transition to the forwarding state on edge ports and on point-to-point links.

Follows the high level process of RSTP sync:

1. When a bridge receives a *superior* BPDU via any port or its root port changes, the bridge will synchronize with its downstream bridges:
    - Immediately blocks all *non-edge designated ports*, i.e., the ports which connect to *downstream* bridges. Blocking the ports is crucial to ensure a loop-free topology during this process.
    - Send *proposal* via those *downstream* ports.
2. The *downstream* bridge compares the new BPDU information with the one locally stored.
    - If new information is *superior*, the bridge elects that port as root port, blocks all other ports and sends *agreement* back to upstream bridge.
    - If new information is *inferior*, the bridge rejects the proposal and sends back its own proposal with the *superior* information it has.
        - The bridges that had its proposal rejected, when receives back a new proposal from the *downstream* bridge, it compares that info with the one it has and, blocks the port, stopping the sync process (new info is not *superior*), or shifts the sync process to the *opposite direction*.

3. When the upstream bridge receives an *agreement*, unblocks that *downstream* port and starts forwarding. The sync process has progressed into another *wave*, i.e., the blocked ports have now *moved down one level*. This process continues *down* the topology, until all bridges have negotiated their ports.
4. The sync process stops when the leaf bridges have no longer *downstream* bridges to negotiate the ports with and/or the proposals reach bridges which reject them, meaning they have *superior* BPDUs stored than the one proposed.

If a bridge receives an *inferior* BPDU via a blocked port, it immediately sends a *proposal* with its cached info about the Root Bridge. RSTP sync process is supposed to have always the Root Bridge information valid, and that's why the bridge immediately answers back with a proposal with its information, without any check performed. Later on, the *Rapid Spanning Tree Problems* section will present some cases where this *blind trust* leads to convergence issues.

It is important to note that STP bridges will never participate in the sync process and if a RSTP bridge has a *downstream* STP bridge, the STP convergence process and timings, rule.

Rapid Spanning Tree Timers

For backward compatibility with 802.1D STP, Rapid Spanning Tree also interprets and uses STP timers, Forward Delay and Max Age, when necessary.

However, for native RSTP operations, only the following timers are involved:

- **Hello interval** - Time between each BPDU sent on a port. By default, 2 seconds, like in 802.1D STP.
- **Max Age** - The maximum time a BPDU is considered valid. By default, 20 seconds, like in 802.1D STP.
- **TCWhile** – Time interval to keep sending TCN towards the Root Bridge. 2x Hello interval, by default.

3.2 Topology Changes

In STP, topology changes were targeted to the Root Bridge. As soon as the Root Bridge received the TCN, it flagged the topology change via the TC bit on its BPDUs and flooded that information throughout the network. Upon receiving the BPDUs with TC bit set, the non-root bridges will reduce their forwarding databases aging time into *Forward Delay* seconds.

RSTP handles topology changes using a slight different approach.

First of all, only ports going into forwarding state causes topology change events. At first sight this seems a little odd, but it makes sense in the way RSTP process it: a loop cannot be formed if a port goes down and two situations might occur at the *downstream* bridge:

1. The *downstream* bridge has alternate path to the root. In this case the *downstream* bridge unblocks that alternate path and starts its own topology change mechanism. That's the reason why RSTP advertises that only transitions to forwarding are the only ones that trigger topology changes; the onus is passed to *downstream* bridges.
2. The *downstream* bridge does not have alternate path to the root: nothing can be done.

Next, edge ports do not generate topology changes, as it is assumed no other bridges connect to edge ports.

As it aims fast convergence, reducing the forwarding databases aging time like STP is not enough. So, RSTP instructs the bridge to ***flush*** all the entries of the forwarding database. In fact this accelerates the convergence, but introduces more unicast flooding into the network. Personally, I understand the unicast flooding concern but I strongly support this process. Besides, Self-Configurable Switches Protocol also performs it.

Similarly to STP, when a bridge detects a topology change, it sends TCN towards the Root Bridge, but now during *TCWhile* seconds, 2x *HelloTime* seconds by default. However, conversely to STP, when a bridge receives a TCN via its designated port, flushes all MAC Addresses, except the ones associated with the *inbound port*, and starts its own flooding process, sending TCN via root port (*upstream*) and designated ports (*downstream*).

Figure 3-1 shows the RSTP flooding process.

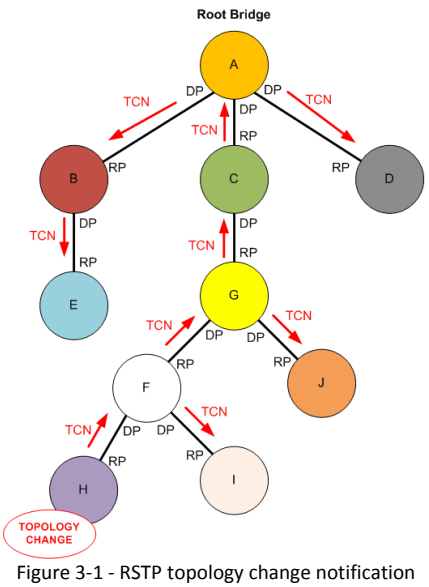


Figure 3-1 - RSTP topology change notification

3.3 Convergence Performance

RSTP's fast convergence in the order of milliseconds is widely advertised. However, indeed, it depends on several factors and it is not trivial to calculate. Simple factors like the *type* of ports, failure location, network topology and bridge role, greatly contribute

to the convergence performance. As the convergence mechanism propagates as a *wave* towards the leaf nodes of the topology, the closer failure is to the Root Bridge, the longer it will take to converge. Besides, RSTP is permeable to "*race conditions*" and this raises serious problems in redundant topologies, especially those *very* redundant, but even for simpler topologies as we will see in short.

RSTP convergence, like STP's, is based on info received from peers. When critical nodes fail, like the Root Bridges, for example, this could lead to old information still being relayed in the network and RSTP may take long seconds to converge, suffering from the classic effect known as ***count to infinity***, especially on large topologies with rich set of redundant links.

Count to infinity is a well-known problem of distance vectors protocols like RIP. STP and RSTP may be also considered distance vector protocols.

Best practices advise the better protection against RSTP limitations as maintaining the redundancy as minimum as possible, small network topologies and protect Root Bridges by all means in order to avoid its loss. Well, summing up advised minimum redundancy and small topologies with inherent unsupported load balancing, probable sub optimal paths choice and topology-dependant convergence time, I think we get a… pretty good protocol for home networks!

3.4 Rapid Spanning Tree Problems

In this section, *count to infinity* behaviour will be analysed, as it is one huge handicap of RSTP. Additionally, *RSTP protocol degradation*, in the presence of legacy STP bridges, and *port role negotiation*, used to guarantee a loop-free topology, are exposed as possible contributors to slower convergence times and poorer RSTP performance.

Count to infinity

When the Root Bridge fails in a looped topology, RSTP frequently exhibits the classic *count to infinity* behaviour, as *stale* BPDUs for the failed Root Bridge might persist in the network, flowing around the topology. *Stale* BPDUs are still relayed by other bridges, under certain circumstances, until BPDUs Message Age reaches Max Age, i.e, after Max Age hops. This way, forwarding loops are formed during tens of seconds, even in small networks.

Please recall that Message Age:

- Measures the distance to the root from a bridge.
- The Root Bridge sends BPDUs with Message Age at 0.
- Each bridge increases the Message Age by 1.
- Max Age is the amount of time a BPDU is considered valid.

RSTP exhibits *the count to infinity* behaviour mainly due to:

1. The network has physical loops, which RSTP should have open, and both *fresh* and *stale* BPDUs will race against each other. In RSTP, multiple races naturally occur between the RSTP state machines and some of them can be harmful [5].
2. The *fresh* information will be continuously eliminated from the network, as the information from *stale* BPDUs is preferable. *Stale* BPDUs will continuously propagating around the network until aging out.
3. The *synchronization* process is not performed as the proposal received by the blocking port carries worse information, allowing the formation of a loop (Figure 3-3).

This behaviour can be seen in a simple five bridge topology, where the Root Bridge is lost and the remaining topology has a physical loop. Figure 3-2, Figure 3-3 and Figure 3-4 illustrate the *count to infinity* behaviour, where a network loop is created as the old information is looping around *poisoning* the network, while the new information chases it.

In the basic network topology pictured in the following figures, if Root Bridge A gets isolated from the network, bridge B will advertise itself as the new root, but under certain *time* circumstances (bridge E receives first the new BPDUs from D, for example), bridge A's BPDUs will persist flowing in and a loop occurs. Only after Max Age hops bridge A's stale BPDUs will age out and the network converges, if not already down.

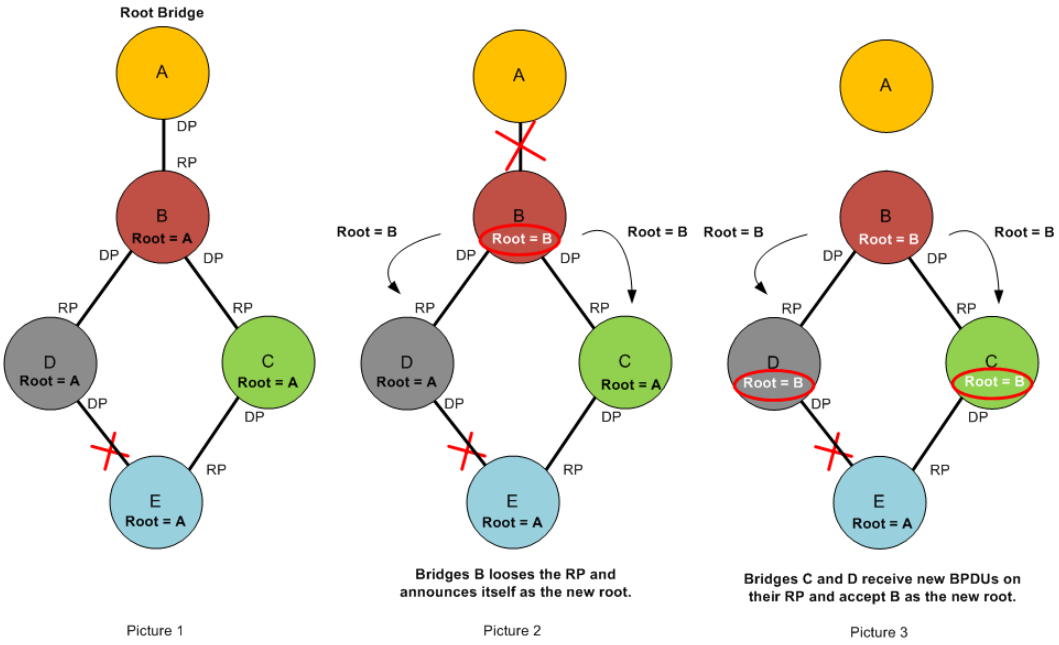


Figure 3-2 – *Count to infinity*: new root advertisement

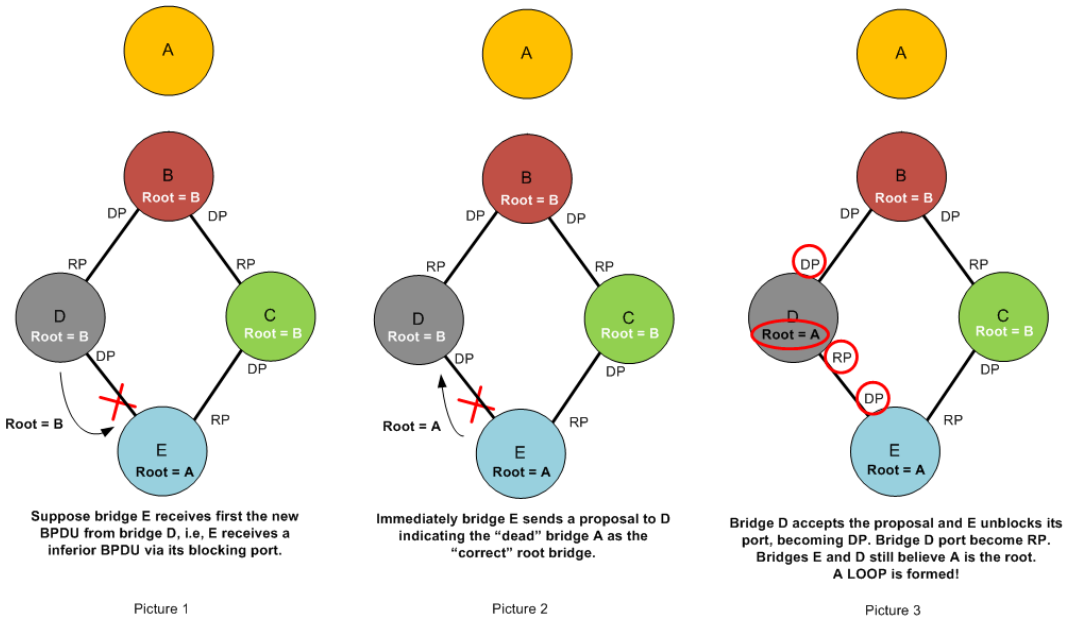


Figure 3-3 – Count to infinity: creating the loop

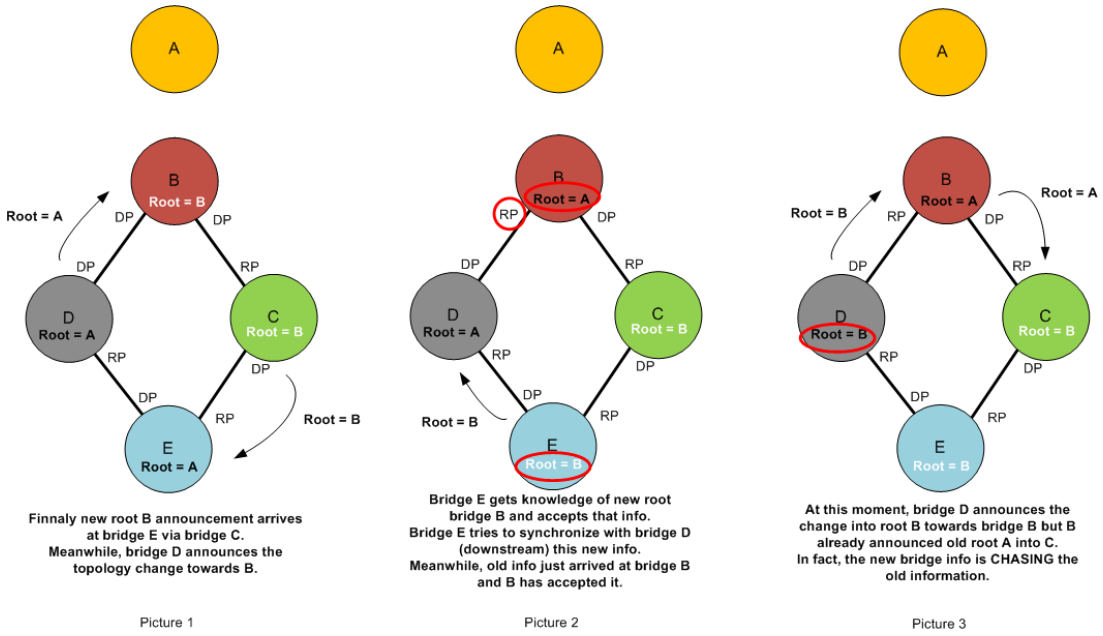


Figure 3-4 – Count to infinity: new information chases old information

Elmeleegy *et al.* [5] presents a very comprehensive explanation of this *count to infinity* behaviour.

Port role negotiation

RSTP negotiates each port role transition, in order to guarantee a loop-free topology.

In a ring topology, for example, with a certain number of bridges in the ring, if one of the two Root Bridge's ports fails, the negotiation overhead delays considerably the network convergence time to several seconds. Some studies advance that after 10 bridges, the convergence time goes above 3 seconds [6].

In a presence of a Root Bridge's link failure, all the bridge ports of half ring will have to change their role from *root port* into *designated port* and from *designated port* into *root port*. Figure 3-5 illustrates this situation.

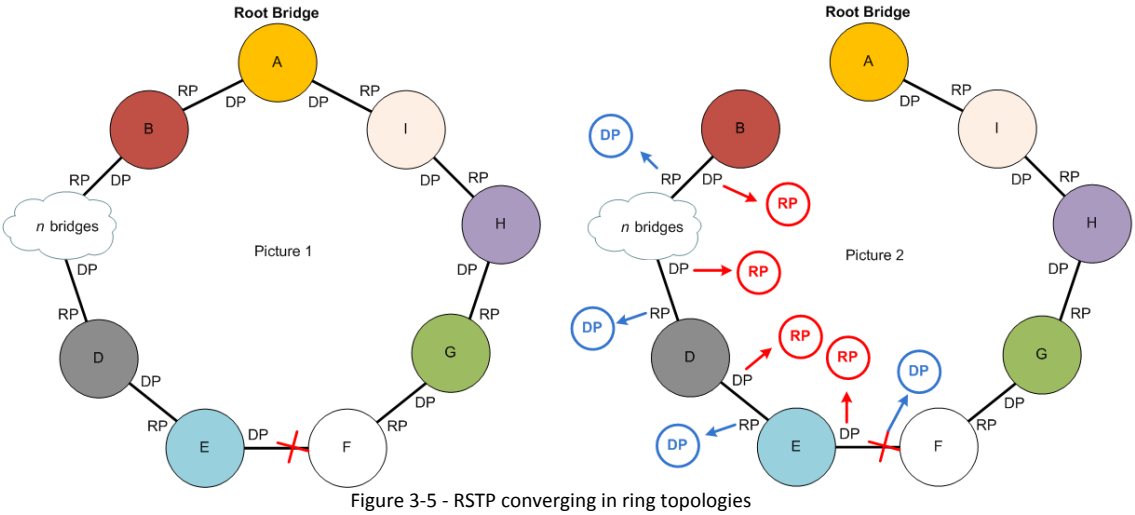


Figure 3-5 - RSTP converging in ring topologies

Usually, the negotiation between two bridges is quite fast, but during periods of convergence when several bridges send simultaneously BPDUs updating root patch costs, RSTP restricts BPDU send rate to one per port per second. This BPDU transmission rate can add seconds to convergence total time [6].

RSTP Protocol Degradation

RSTP advertises fast convergence, within milliseconds.

In the previous analysis, there were revealed two handicapping issues which potentially shift the protocol out of large, service-critical networks.

This section exposes one other issue that contributes even more for reducing the potential benefits of using RSTP on a network: RSTP allows STP bridges to operate in a RSTP network.

RSTP protocol degradation occurs anytime a bridge port detect legacy STP 802.1D bridges on the network and all RTSP interfaces are reverted back to 801.D STP specifications.

RSTP needs to revert back to STP mode in order to avoid having bridges in the network that doesn't have the correct network information, as STP bridges do not understand RSTP BPDUs.

As soon as a RSTP bridge detects a STP BPDU via one of its ports, it relies on *time* to *heal* this condition, hopping STP BPDUs to be relative to transient configuration. In fact, RSTP waits 2x Hello Interval to see if STP BPDUs will cease; if they do not, RSTP will have to *revert back to STP* mode and *all bridges* will have to communicate using STP specifications. That way, the RSTP network loses all the advertised potential enhancements over STP.

Sometimes the *all bridges* statement is somehow misunderstood. Although 802.1D-2004 standard states that protocol operation on other ports is unchanged, effectively, the RSTP bridge starts using STP BPDUs on all bridge ports ([7] clause 17.4).

Although *Part V - Protocols Assessment* will present detailed analysis on RSTP protocol degradation consequences, Figure 3-6 illustrates a simple scenario where a link failure in a STP bridge creates a problem in the RSTP network.

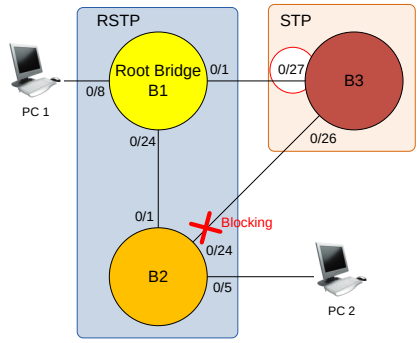


Figure 3-6 - RSTP protocol degradation example

Bridges B1 and B2 are running RSTP and B3 STP. Spanning tree has blocked interface 0/24 @ B2, to break the loop. Pc1 is communicating with Pc2.

If the link B3-B1 fails, the STP bridge B3 needs to change its root port from 0/27 to 0/26 and RSTP bridge B2 to unblock interface 0/24. Figure 3-7 shows the Spanning Tree information of bridge B2, before and after de link failure.

```
Interface        Role Sts Cost      Prio.Nbr Type              Interface        Role Sts Cost      Prio.Nbr Type
---------------- ---- --- --------- -------- ----------------  ---------------- ---- --- --------- -------- ----------------
Fa0/1            Root FWD 20        128.3    P2p               Fa0/1            Root FWD 20        128.3    P2p
Fa0/5            Desg FWD 19        128.7    P2p               Fa0/5            Desg FWD 19        128.7    P2p
Fa0/24           Altn BLK 19        128.26   P2p Peer(STP)     Fa0/24           Desg FWD 19        128.26   P2p Peer(STP)
```

Figure 3-7 - Bridge B2 spanning tree information

The rapid convergence of RSTP is damaged by a failure in a STP bridge, reverting back to STP values. The following ICMP communication output between Pc1 and Pc2, presents the 30-seconds downtime characteristic of STP networks.

```
64 bytes from 10.10.10.1: icmp_seq=56 ttl=128 time=1.32 ms
64 bytes from 10.10.10.1: icmp_seq=57 ttl=128 time=1.14 ms
64 bytes from 10.10.10.1: icmp_seq=58 ttl=128 time=1.46 ms
64 bytes from 10.10.10.1: icmp_seq=59 ttl=128 time=0.781 ms
64 bytes from 10.10.10.1: icmp_seq=60 ttl=128 time=1.10 ms
64 bytes from 10.10.10.1: icmp_seq=61 ttl=128 time=1.17 ms
64 bytes from 10.10.10.1: icmp_seq=92 ttl=128 time=1.35 ms
64 bytes from 10.10.10.1: icmp_seq=93 ttl=128 time=1.67 ms
64 bytes from 10.10.10.1: icmp_seq=94 ttl=128 time=1.49 ms
64 bytes from 10.10.10.1: icmp_seq=95 ttl=128 time=1.56 ms
64 bytes from 10.10.10.1: icmp_seq=96 ttl=128 time=1.63 ms
```

Figure 3-8 - Convergence times stuck to STP behaviour

Moreover, if a RSTP bridge begins running in *STP compatible mode* some vendors do not implement automatic mechanisms to revert back into RSTP; manual intervention may be necessary.

4 Multiple Spanning Trees Protocol

The IEEE standard 802.1s, Multiple Spanning Trees [8], extends the specifications on 1998 edition of IEEE standard 802.1Q, Virtual Bridged Local Area Networks, to allow groups of different VLANs to be mapped into different spanning trees, while maintaining interoperability with prior spanning tree protocols standards 802.1D STP and 802.1w RSTP.

IEEE 802.1s-2002 Multiple Spanning Trees was included in the 2005 revision of IEEE standard 802.1Q, 802.1Q-2005 [9]. Currently, 802.1Q standard is specified on 802.1Q-2011 [10].

Multiple Spanning Trees targets the multi-VLAN spanning tree paradigm.

Prior standards 802.1D Spanning Tree Protocol and 802.1w Rapid Spanning Tree Protocol, defined the use of a single spanning tree instance, named ***Common Spanning Tree*** (CST), i.e., there was only one Layer 2 network topology and ALL VLANs mapped into that unique topology.

802.1s Multiple Spanning Trees allows customized mapping of a configurable number of VLANs into groups and the coexistence of multiple, independent, spanning tree instances to run simultaneously for those groups, with the flexibility of one spanning tree instance to be completely independent of others.

All VLANs mapped to the same group, belong to the same spanning tree instance and share the same spanning tree logical topology. That way, traffic load sharing is achieved using multiple instances and different non-overlapping network topologies.

Along with several other vendors, Cisco Systems developed their own enhanced versions of spanning tree to deal with the multi-VLAN spanning tree paradigm, like PVST+, meaning *Per VLAN Spanning Tree*. In PVST+, EVERY VLAN runs over a different spanning tree instance, which means that if we had 2500 VLANs, we would get 2500 spanning tree instances, probably overlapped over just 2 or 3 different Layer 2 logical topologies. This represents a huge burden to the bridge resources, although allowed traffic load sharing on Layer 2 networks, which the standards did not. However, with MSTP, the 2500 VLANs could be distributed through 2 or 3 MSTP instances and, for so, only the targeted 2 or 3 different logical topologies would exist with minor impact on bridge resources. That's MSTP great power.

Cisco RPVST+ (*Rapid Per VLAN Spanning Tree*), also known as PVRST+ (*Per VLAN Rapid Spanning Tree*), is the same as PVST+ but supported on RSTP instead of legacy STP.

Figure 4-1 and Figure 4-2 picture potential converged topologies for a nine-bridge network, with 500 VLANs.

In Figure 4-1, picture 1 illustrates the physical connectivity scheme and picture 2 the converged topology, either in STP and RSTP as, by the standard, they use only a CST for all 500 VLANs.

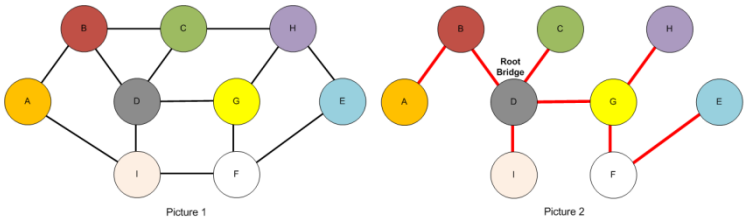


Figure 4-1 - STP / RSTP Converged Topology

Figure 4-2 shows the different network topologies MSTP allows. For example, if VLANs 1-250 were mapped to MSTP Instance 1 and VLANs 251-500 mapped to MSTP Instance 2, the network converged topologies could be completely different, like Picture 2, for MSTP Instance 1, and Picture 3, for MSTP Instance 2, demonstrate.

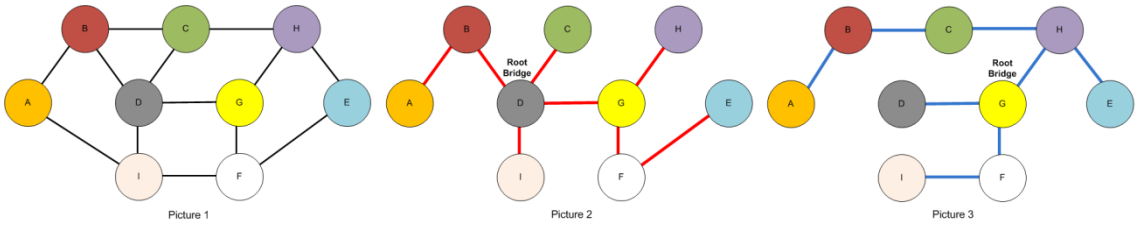


Figure 4-2 - MSTP Instances Converged Topology

The above figures show an interesting characteristic of MSTP. If you compare both Pictures 2, you will see that they are the exactly the same, and that's not unintentional; it illustrates that MSTP heavily relies on RSTP, as it runs RSTP *inside* each MSTP instance.

MSTP is suitable for large Layer 2 networks and its manageability is fully supported on the ***MSTP Region*** concept, analysed next.

MSTP Regions

An MSTP Region can be defined as a group of bridges under a common administration.

To determine common administration boundaries, MSTP Regions have three very important attributes, which must be manually configured:

- **Region Name** – a name given to the region
- **Revision Number** – an administrator tag for configuration revision number
- **VLAN Association Table** – the VLAN-into-instance mapping

Two bridges are considered in the same MSTP Region if ALL those three attributes match. In the case of one (or more) of them mismatches, the bridges belong to *different* MSTP Regions.

Bridges need to know the VLAN mapping that is being performed on the MSTP Region. The fully *VLAN Association Table* is not exchanged; instead, the MSTP BPDUs carry just a *VLAN Association Table* MD5 hash, which is used to determine if VLAN Association Tables are identical. The other two attributes are administrator *tags* to identify the region, *Region Name*, and track configuration changes, *Revision Number*.

It is mandatory to match all three attributes among all MSTP Region bridges, as they identify the region and define the VLAN-into-instance mapping the bridge should perform. If any of them does not match, it's either not associated with the same region ID and/or does not perform the same mapping and a *MSTP Region Boundary* exists.

For MSTP, it is extremely important to get familiarity with the concept of ***MSTP Region Boundary***.

MSTP Region Boundary

Primarily, different MSTP Regions do not *speak* MSTP. The spanning tree protocol which runs on MSTP Region Boundaries is the *best* of the legacy ones supported by both ends of the boundary: RSTP or STP.

A bridge port is considered at the edge of a MSTP Region if:

- Other bridge on its segment is in a different MSTP Region
- It receives legacy BPDUs, either RTSP or STP.

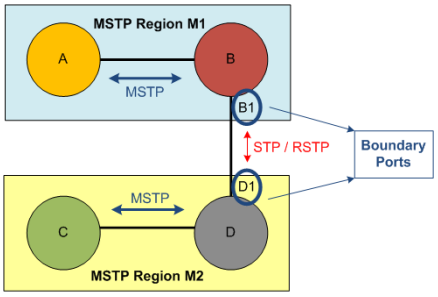


Figure 4-3 – Different MST Regions

On Figure 4-3, bridges A and B belong to the same MSTP Region M1, whereas bridges C and D belong to MSTP Region M2. Even if the other attributes matched, as *Region Name* does not match, the MSTP Regions would be different. Consequently, bridge ports B1 and D1 are ***boundary ports***, as they are at the boundary of their MSTP Regions.

The same figure illustrates that between different MSTP Regions, runs STP or RSTP; not MSTP.

MSTP Instances

Multiple Spanning Trees Protocol is all about creating *instances*.

A MSTP ***instance*** is a group of a customized number of VLANs, which runs an independent spanning tree instance.

The standard 802.1s defines that a bridge must handle:

- One **IST** - Internal Spanning Tree [mandatory]
- One or more independent **MSTPI** – MSTP instances

The ***IST*** is simply the CST extension inside the MSTP Region. The IST is responsible for exchanging BPDUs with the *external* CST. Regarding the previous MSTP Boundary

definition, when a MSTP Region needs to communicate outside the region, it uses the IST to *connect* into the CST that exists beyond the region.

The ***MSTPIs*** are the configurable MSTP Instances that reside only inside the MSTP Region and run RSTP natively. Unlike IST, the MSTPI never interact with the *outside world*.

As an example, suppose a campus network in production with 500 VLANs. A new campus backbone was built and decided to run MSTP, mapping VLAN range 1-250 to MSTPI 1 and 251-500 to MSTPI 2. Bridge A and B are backbone bridges and were configured with Region Name *CampusBB*, Revision Number 1 and the VLAN mapping described. Therefore, bridge A and B belong to the same MSTP Region.

To interconnect the new campus backbone to the production network, it was used the schema of Picture 1 on Figure 4-4. Consequently, at that stage, the legacy production network would be attached directly to the new campus backbone via a single link. That way, B1 port would become a boundary port and would be communicating with the legacy network CST via the MSTP Region IST.

From Picture 1, bridge A and B run separate spanning instances for MSTI 1 and MSTI 2, but to the *outside world*, the MSTP Region appears as a single ***virtual bridge*** running the legacy spanning tree protocol, as illustrated on Picture 2.

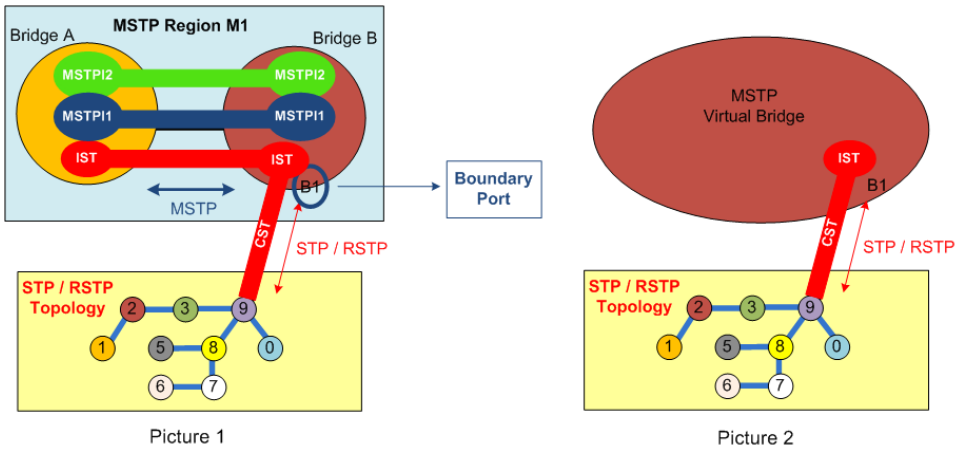


Figure 4-4 - MSTP IST / CST Interaction

IEEE 802.1s specification [8] fully describes the mechanism that makes an entire MSTP region to appear as a single virtual bridge to the *outside world*. This MSTP property is very important although somehow complex; however, visualizing the basic concept of MSTP Region virtualization towards the CST, simplifies understanding and troubleshooting a mixed topology.

Figure 4-5 presents a good example where the virtualization concept is necessary to understand the topology. In Picture 1, it seems a little odd how RSTP blocked port A->E, being A the Root Bridge, and port C->G instead of, for example, port G->F. Picture 2 makes it simple to understand, extrapolating the MST region into the virtual switch concept.

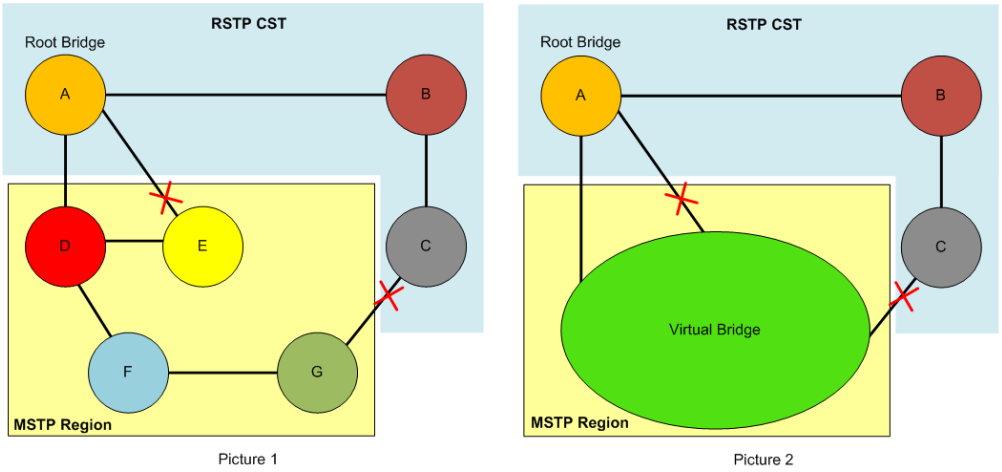


Figure 4-5 - MSTP Virtual Bridge

MSTP BPDU

As said before, IST is a special MSTP instance that runs all over within a MSTP Region and interact with the outside world's CST via MSTP boundary ports. MSTPIs are configurable RSTP instances internal to a MSTP Region, which do not communicate outside the MSTP Region.

IST is the only MSTP instance that sends and receives BPDUs to/from a boundary port. Moreover, MSTPI do not send any individual BPDUs at all.

MST BPDUs are just 1 BPDU for all MSTP instances (IST + MSTPIs), containing the IST information, the MSTP Region Information and plus one *MRecord* for each MSTPI.

Bridges exchange MSTP BPDUs periodically inside the region, which have two particular characteristics:

- They appear as normal RSTP BPDUs for the IST.
- They contain additional information for each MSTPI – ***MRecords***.

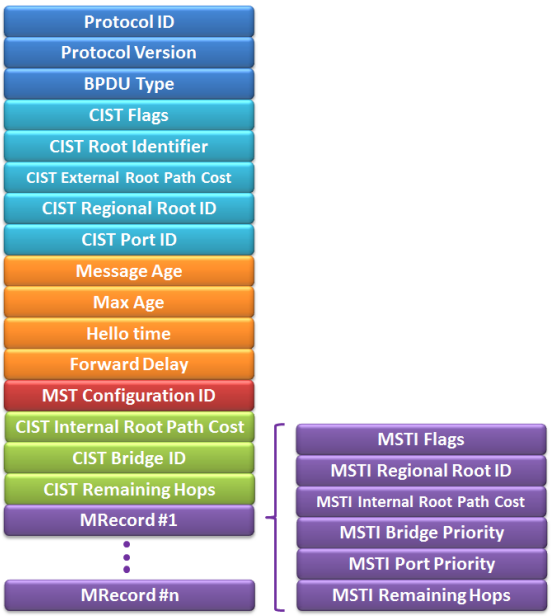


Figure 4-6 - MSTP BPDU Format

The ***MRecords*** contain all the necessary information to calculate the final topology of that particular instance, and do not rely on any timer. The MSTPI depend on the IST to transmit their information and so, there is only need for the IST to rely on timers (Hello Interval, Forward Delay, and Max Age).

At MSTP Region boundaries, only IST BPDUs are sent out; *MRecords* do not leave the MSTP Region.

4.1 Loop Free Topology

MSTP is very complex and this section only overviews some interesting particularities of the protocol. For a detailed description of MSTP operation, please consult the IEEE standard specification [9].

In certain circumstances, MSTP might elect *several* root bridges:

- **CIST Root Bridge** - the bridge that has the lowest Bridge ID among ALL regions.
- **CIST Regional Root Bridge** - Root Bridge of the IST for the given region and also a boundary bridge elected for every region based on the shortest external path cost to reach the CIST Root.
- **MSTPI Root Bridge** – Root Bridge for the particular MSTPI.

As described in previous sections, MSTP BPDUs are carried over the IST, they contain regular RSTP information for the IST itself, as well as additional informational about MSTP, like configuration name, revision number and the VLANs-to-Instance hash, and additional MRecords, one for each MSTP instance. So, as IST and MSTPIs are different spanning tree instances, each will have to elect a root bridge, based on same criteria as STP/RSTP. Moreover, within a region the MSTPIs have no perception of *other* Root Bridges whatsoever, as each instance is completely independent; the sole dependency is over IST, which propagates MSTIs MRecords.

Like STP/RSTP, every bridge sends MSTP BPDUs every Hello Interval and using RSTP convergence mechanisms, separate spanning tree calculations will be performed for the IST and each MSTPI.

Spanning Tree Timers

The MSTPIs do not use any timer, besides Hello Interval; they rely on spanning tree timers defined for the IST.

IST uses the regular spanning tree timers, Max Age and Forward Delay. Those timers are vital when MSTP interact via IST to the CST.

However, MSTP has a specific mechanism to age out information on IST, based on hop count. The field ***CIST Remaining Hops*** in Figure 4-6 presents the TTL-style used by MSTP. The IST Root Bridge sends BPDUs with *CIST Remaining Hops* equal to a configurable value (***MaxHops***) and every other non-root bridge decrement the counter on reception of the BPDU. Reaching zero, the BPDU is ignored and, consequently, that bridge may start declaring itself as the new IST Root Bridge.

CIST Remaining Hops is crucial to define the *length* of the MSTP topology.

4.2 Topology Change

In a multi-region scenario, MRecords do not propagate outside a MSTP Region. Hence, any change affecting one MSTPI in one MSTP Region will not affect MSTPIs on other regions; they are internal to the region. However, CST recalculations affect every region and all MSTPIs.

Topology changes is MSTP are treated as regular RSTP ones. Nevertheless, as MSTP might run several spanning tree instances simultaneously, the effect of a topology change will depend on the topologies created. For example, a link failure might represent a big issue for a certain MSTPI, but for other could be no service-impacting as that link could be blocking on the topology of that particular instance. The same applies for IST.

4.3 Convergence Performance

MSTP relies on RSTP. Therefore, the advertised RSTP's fast convergence in the order of milliseconds still applies to MSTP. However, all those issues analysed for RSTP also exist in MSTP, even the ***count to infinity*** behaviour.

Though, MSTP run separate RSTP instances for each MSTPI and that leverages significantly the performance.

4.4 MSTP Problems

Count to infinity behaviour still applies to MSTP and, especially in larger topologies, transient bridging loops might occur, as analysed in the *Rapid Spanning Tree Protocol* chapter.

Two main drawbacks specific of MSTP are analysed next: ***protocol complexity***, when compared with STP and RSTP, and the challenging ***interaction with legacy bridges***.

Being MSTP a complex protocol, it is most of the times misunderstood. A lot of misconfigurations can be performed with devastating consequences, even in the simpler networks topologies as ***Misinterpreting IST*** section will present.

Protocol Complexity

MSTP is much more complex than legacy spanning tree protocols and requires additional training. Simple single-region setups are somehow intuitive to configure, however, harder to understand and even more difficult to troubleshoot.

A single misconfiguration of one of the MSTP parameters, like the revision number, might split the MSTP Region and create region boundaries, with all the effects inherent to that change.

Multi-region topologies and scenarios running MSTP with legacy STP/RSTP require design expertise and more to troubleshoot, taking into account the number of Root Bridges and potential topologies within each MSTP Region, besides the complexity interconnecting MSTP regions and IST integration in CST for STP/RSTP.

Interaction with legacy bridges

Interaction with legacy bridges can be a challenge.

Previous *MSTP Region Boundary* and *MSTP Instances* sections addressed the concepts behind connecting legacy bridges with MSTP Regions. As soon as a MSTP bridge senses a legacy bridge, it creates a boundary port. That way, the legacy bridge connects to the MSTP Virtual Switch via the CST. As CST topology changes affect every bridge on every MSTI and IST and on all regions, this legacy bridge can induce unwanted instability to all MSTP network and greatly increases network topology complexity.

Therefore, it is recommended not to map any VLAN to IST and avoid connecting MSTP regions to legacy spanning tree domains.

Misinterpreting IST

There are numerous MSTP misconfiguration examples and analysis published, as MSTP tend to be complex and misunderstood. In this section it will be presented an extremely simple case of two bridges connecting to each other, where a misconfiguration leads to lack of connectivity.

Suppose a two-bridge network with two VLANs, one mapped on the IST and the other on a MSTPI:

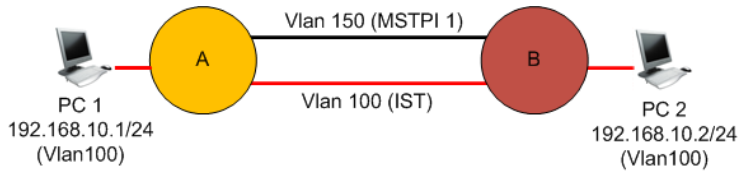


Figure 4-7 - IST Misconfiguration

Even with this basic case, PC1 is not able to communicate with PC2. Why?

Connectivity issues are related with misinterpretations of the IST concept, most of the times.

Even in the Figure 4-7 simple topology, IST is playing an important role and if its main characteristics are disregarded, connectivity issues would arise. Recall that IST is spread all over the MSTP Region by all bridge ports, the MSTP BPDUs are transmitted only over IST and every MSTP BPDU carry the IST information plus one or many MRecords.

The MSTP BPDUs which carry the MSTPI 1 information over the top link also transport the IST information and, therefore, the IST topology for Figure 4-7 is effectively:

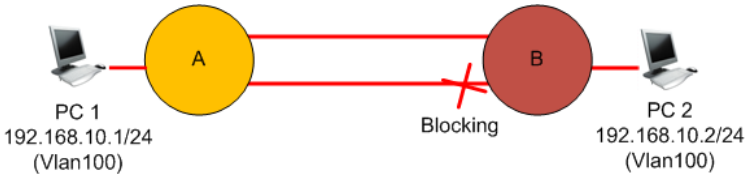


Figure 4-8 - IST Converged Topology

Even though VLAN 150 is mapped on MSTPI 1 and flows only over the top link, the IST is also at that link and, for that topology, that top link is preferable over the bottom one and, therefore, the bottom one is blocked by spanning tree.

As VLAN 100 is flowing solely through the bottom link, it causes the connectivity issues.

The recommended solution for this particular case would be mapping all VLANs out of IST, over particular MSTPIs.

PART II – Emerging Standards

Part II presents the fresh standard proposals for Spanning Tree Protocols substitution, IETF's TRILL and IEEE's 802.1aq SPB.

5 TRILL Protocol

TRILL stands for *Transparent Interconnection of Lots of Links* and is an IETF standard [15]. It is interesting to see TRILL emerging as an Ethernet standard from IEFT, instead of IEEE. Actually, TRILL was first proposed to IEEE back in 2006, but the 802.1 group did not pursue it. IEEE decided to fight back with 802.1aq… or it is the opposite, being TRILL a faster response for IEEE leisureliness? Well, politics apart, it is quite fascinating to see that IETF TRILL and IEEE 802.1aq are indeed conceptually identical.

IEEE 802.1aq will be described in the next chapter.

Devotees state TRILL bridges still looks like a bridge to the outside world, although using Layer 3 routing and encapsulation techniques.

I'm not a TRILL devotee, as TRILL is not trilling! TRILL dramatically changes the bridging concept and "*routerializes*" a bridge.

Some big companies are believed to be behind TRILL development, like Sun Microsystems, Cisco Systems, Brocade, Juniper and Nuova (currently Cisco Nexus). On the other hand, there are other big companies that opposes to TRILL (or to those big companies?).

Naturally, there are lots of opinion articles over Internet about pros and cons of TRILL. Such discussions are completely out-of-scope of this document; this chapter's objective is to inform about TRILL development and its main characteristics.

5.1 Link State Protocols Short Description

TRILL uses a link state protocol: *Intermediate System-to-Intermediate System* (IS-IS or ISIS). Very shortly, link state protocols are routing protocols that establish neighbour relationships, in which each router determines who its neighbours are and sends them *Link State Packets* (LSP), containing its identity and path costs towards all those neighbours.

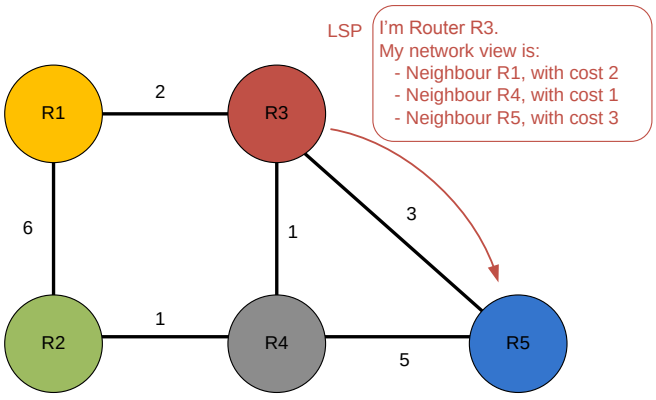


Figure 5-1 - Link state protocol LSP example

Figure 5-1 shows an example of the type of information some LSPs might contain.

Figure 5-2 shows a real OSPF packet capture, where the type of LSP, link state ID and the network view are highlighted.

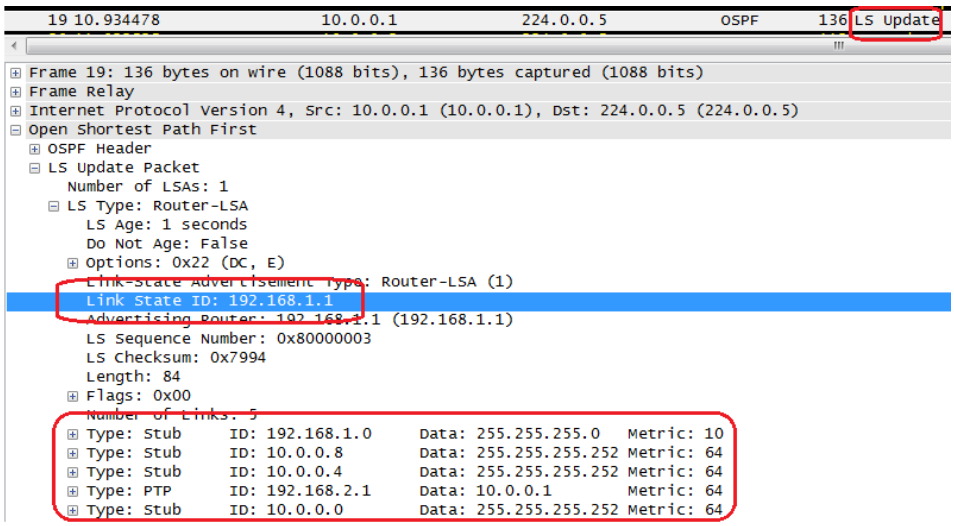


Figure 5-2 - OSPF LSP sample packet capture

This LSP exchange allows each router to create a *Link State Database*, equal on all routers of the network, containing all routers, all paths between them and their costs. Running a *Shortest Path First* algorithm over the Link State Database, link state protocols compute best paths.

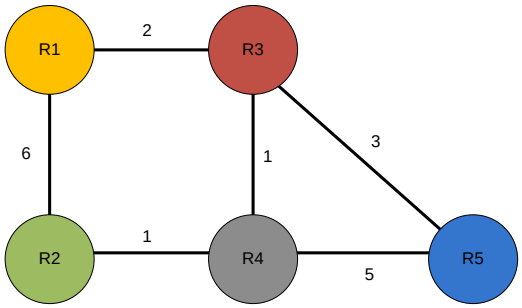


Figure 5-3 - Routing network example

Figure 5-3 shows a sample network topology, built over five routers running a link state protocol. That specific topology would generate the following Link State Database of Table 1, equal on all five routers:

Link State Database				
R1	**R2**	**R3**	**R4**	**R5**
$R_3/2$	$R_1/6$	$R_1/2$	$R_2/1$	$R_3/3$
$R_2/6$	$R_4/1$	$R_4/1$	$R_3/1$	$R_4/5$
		$R_5/3$	$R_5/5$	

Table 1 - Link State Database example

The most common link state routing protocols are OSPF [17] and ISIS [18].

OSPF runs on top of IP layer and requires all routers to have Layer 3 addresses, i.e., IP addresses. Conversely, ISIS runs over Layer 2 directly (CLNS), so no need for IP to establish neighbour relationships.

ISIS allows additional fields to be easily added, encoded with *Type-Length-Value* (TLV), which OSPF does not.

Those are the main reasons why TRILL (and IEEE 802.1aq) bridges run ISIS.

5.2 TRILL Overview

TRILL is a Layer "*2.5*" protocol.

It is below Layer 3 because it relates with a single link, and it is above Layer 2 because it terminates the traditional Ethernet networks, like Layer 3 devices would do. TRILL uses Layer 3 routing techniques to create large groups of links, which appear to IP devices like single links.

IETF TRILL working group exists since 2005 and in 2010 the IESG approved the standardization of the ***RBridge*** concept, currently in RFC. IESG is the entity responsible for the Internet standards process.

TRILL has a flat 16-bit addressing scheme. This is an advantage over 48-bit Ethernet addressing, as a simple table lookup can be used to find the entry in the output port, whereas Ethernet needs CAM table.

TRILL uses ISIS protocol to get a snapshot of the network and calculate the best paths. Those best paths are not calculated using MAC address info, but rather on the RBridges ID.

RBridges store mappings for destination MAC address along with RBridges ID, like the following Figure 5-4 illustrates:

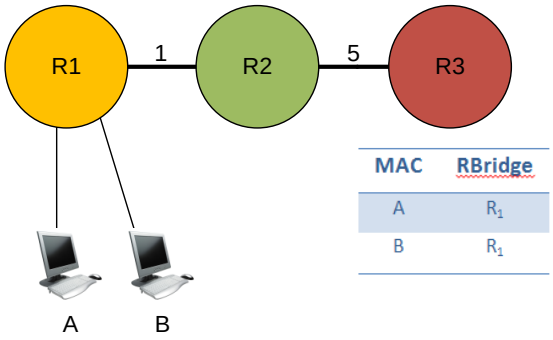


Figure 5-4 - MAC address mapping into RBridges

Topologies changes do not affect the MAC table. Thus, it is only necessary to recalculate the ISIS "part" and the consequent TRILL *routing table*. No flooding will occur upon TRILL failures, when alternative paths exist!

Routing choice is made hop-by-hop, each RBridge making its own decision how best to get to the destination RBridge. However, the choice of the egress RBridge is determined by the ingress RBridge. In previous Figure 5-4, the ingress R3 chooses the egress R1, when forwarding traffic towards A, for example.

To mitigate temporary loop issues, RBridges target traffic to the specific next hop Rbridge, when forwarding unicast frames, avoiding traffic replication during a temporary loop.

Additionally, TRILL uses a hop count to protect against looped traffic.

As *TRILL Header* section will describe, Ethernet frames are encapsulated with two additional headers.

RBridges

RBridges, or Routing Bridges, are devices that implement TRILL and run a link state protocol, ISIS. RBridges have the link state protocol view of the network, as they have knowledge of all RBridges in the network and all the links between them.

RBridges are specified on IETF RFC 6325 [15].

RBridges somehow *merge* both bridge and router concepts:

- Terminate Spanning Tree Protocol, like routers.
- Allow plug-and-play, like bridges.
- Use optimal paths, have fast convergence and no meltdowns, like routers.
- Create broadcast domains, like bridges.

TRILL Header

The 64-bit TRILL header is used to encapsulate Ethernet frames. The most important fields are:

- Ingress RBridge nickname (16 bit)
- Egress RBridge nickname (16-bit)
- Hop count (6-bit)
- Multi-destination flag bit (1-bit)

Ingress and egress nicknames for Rbridges are dynamic or manual abbreviations of the 48-bit ISIS system ID of the Rbridge, derived from any of the RBridge's unique MAC addresses. Those nicknames are unique within the TRILL network.

Ethernet frames over TRILL networks get three headers, like Figure 5-5 shows:

- The original Ethernet frame header (*inner header*).
- Trill header – like a regular IP header, remains unchanged until reaches the egress RBridge.
- Hop-by-hop header (*outer header*) – contains ingress RBridge as source, egress RBridge as destination and the VLAN information. This header is stripped-off hop by hop.

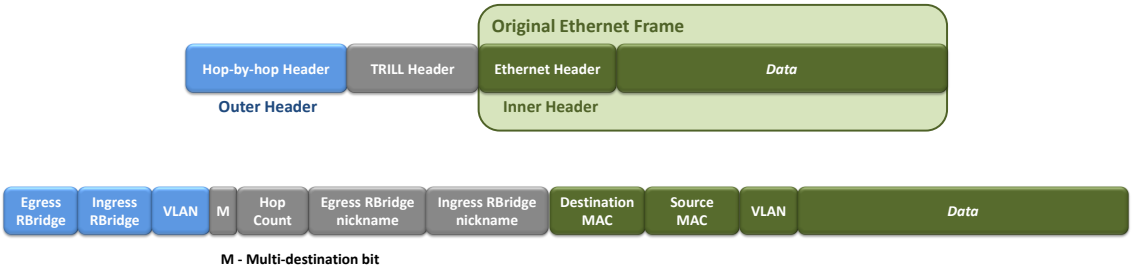


Figure 5-5 - Ethernet frames encapsulation

The following Figure 5-6 illustrates an example of machine S communicating with D, via a TRILL network. To ease header identification, Figure 5-5 colour scheme is maintained.

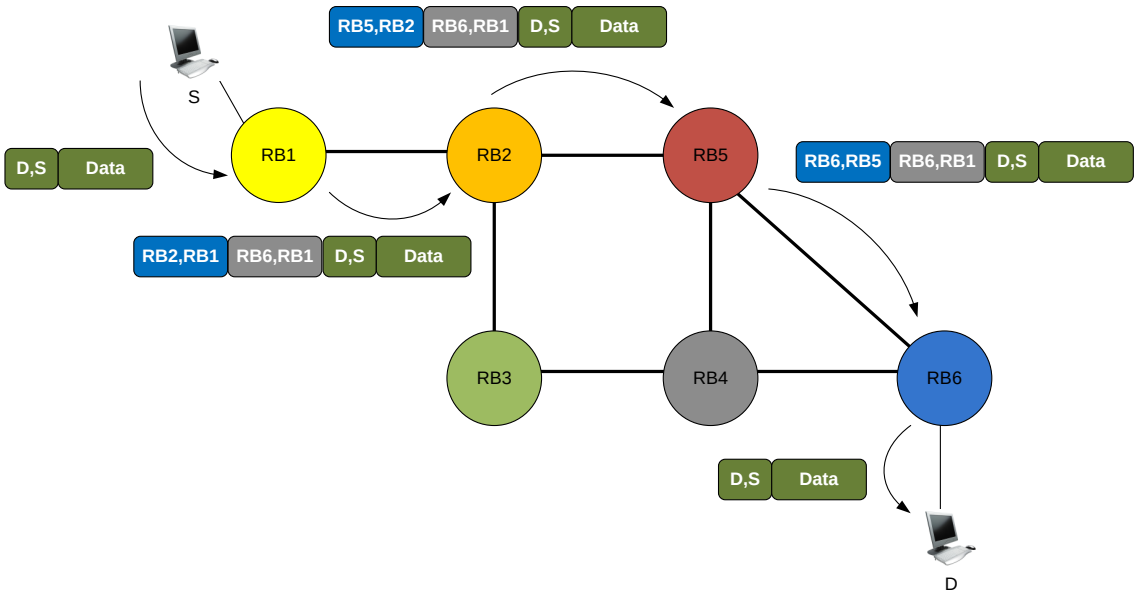


Figure 5-6 - Ethernet frame encapsulation example

Upon receiving machine S traffic, RBridge RB1 encapsulates it with TRILL header, setting the RBridge holding destination host D, RB6, as the egress RBridge. Next, RB1 inserts the hop-by-hop header, setting RB2 as egress RBridge and forwards the TRILL frame towards that RB2.

As Figure 5-6 shows, upon insertion of the TRILL header by ingress RBridge RB1, it remains unchanged until reaching RBridge RB6. Only the hop-by-hop header is consecutively inserted and removed, on a hop-by-hop basic; the remaining headers are not changed.

It is created a kind of a tunnel between those two RBridges RB1 and RB6. Then the hop-by-hop header is encapsulated/decapsulated as needed, to forward TRILL traffic through the network.

TRILL works like *Layer 2 routing*: at each hop it is added a layer 2 header to get to the next bridge, the hop-by-hop header.

Multicast / Broadcast Support

TRILL support for multicast and broadcast is based on RBridges calculation of *distribution trees* to deliver multi-destination traffic. The number of distribution trees is configurable.

One advantage of TRILL is that it does not require an additional protocol to calculate the distribution trees, as the link state database already provides a snapshot of the entire network. Moreover, the hop count can minimize temporary loops effects, as bridges can calculate the accurate hop count towards the egress RBridge.

The multi-destination traffic forwarding is a bit complex in TRILL and is still under development. However, TRILL defines distribution trees and a Tree Root is elected, being the RBridge with highest priority.

The Tree Root RBridge uses LSPs to instruct other RBridges about how many and which trees should be created.

For each tree, the Root RBridge acquires one distinct TRILL nickname to be used by others in the egress RBridge nickname of the TRILL header. The egress RBridge nickname specifies the distribution tree to be used.

When a multi-destination frame is encapsulated by a RBridge, it sets the multi-destination flag bit and the egress RBridge nickname with the specific Root RBridge nickname.

As RBridges announce via LSP which VLANs they are attached to, the multi-destination frames are only sent to those RBridges that have interfaces on the VLAN related with the multi-destination frame.

Learning End Nodes Locations

For local non TRILL received frames, the Mac addresses are learnt *normally*, by looking at the source MAC address of Ethernet frames.

From TRILL frames received, RBridge learns also the source MAC address, looking at the Ethernet header (inner header) of frames.

For unknown destination traffic, distribution trees are calculated.

Like stated before, RBridges calculate trees for delivering multi-destination traffic.

If a RBridge doesn't know the end node destination, it sets the multi-destination flag bit indicating it should be transmitted through the multi-destination tree calculated, towards all other RBridges.

The default mechanism to learn end nodes locations is quite similar to Transparent Bridging concept: the egress RBridge decapsulates the TRILL packet and learns the correspondence between the ingress RBridge and the source MAC address.

There are some dynamic alternatives, like ESADI [16], *End Station Address Distribution Information*, where RBridge announces some or all end nodes' MAC addresses attached to it.

However, strangely, it seems that MAC addresses learning is TRILL's least explored area.

VLAN Support

TRILL supports VLANs using two VLAN tags:

- ***Inner header VLAN tag*** – VLAN information regarding the original host communication.
- ***Outer header VLAN tag*** – for inter-RBridge communication, traversing an Ethernet network.

Both VLAN tags were already pictured in previous Figure 5-5:

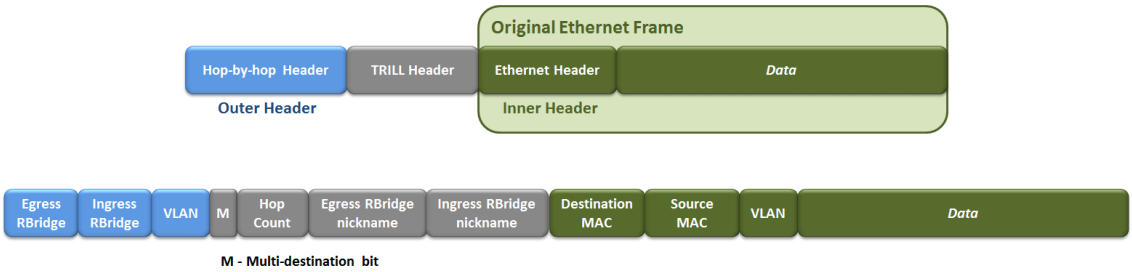


Figure 5-5 - Ethernet frames encapsulation

5.3 TRILL Protocol Considerations

TRILL inherits the equal-cost multipath capability and optimal path calculations provided by ISIS. Therefore, load balancing for unicast traffic and optimal path traffic forwarding is supported by TRILL.

TRILL is marketed requiring minimal configuration, supporting broadcast and multicast traffic and providing load sharing among multiple equal paths. Besides, it mitigates routing loops using TTL fields and is as much secure as existing bridged solutions are.

Although it is still under development, TRILL already presents some disadvantages:

- Complex.
- Processor intensive.
- Multi-destination trees election and maintenance.
- Overhead introduced by two new headers.

ISIS protocol is not simple to implement and bridges must run it, as TRILL is based on it. Besides not being necessary to master ISIS to configure TRILL, it is important to have some degree of knowledge on ISIS to troubleshoot potential TRILL issues.

The multi-destination trees concept is *grey* and it does not seem a great technological leap, regarding other Layer 2 protocols.

There might be also capacity issues regarding MAC address table size and growth. As the number of endpoints grows in a network, TRILL MAC address tables will grow linearly and it is not possible to use address aggregation, as layer 2 addresses are flat.

TRILL is a quite recent protocol, hard to keep track of, as dozens of RFCs and internet drafts are published and many information is continuously superseding previous considerations. Just to have an idea of its information freshness and

magnitude, the following Figure 5-7, Figure 5-8 and Figure 5-9 show the RFCs published, as well as both the old and the currently active internet drafts for TRILL.

Some documents are pretty recent, even 15-days old!

TRILL IETF working group is still active, at the moment of this document elaboration, and its closure is schedule for July 2012.

RFC 5556 (draft-ietf-trill-prob)	Transparent Interconnection of Lots of Links (TRILL): Problem and Applicability Statement	2009-05	RFC 5556 (Informational) Errata
RFC 6325 (draft-ietf-trill-rbridge-protocol)	Routing Bridges (RBridges): Base Protocol Specification	2011-07	RFC 6325 (Proposed Standard) Updated by RFC 6327, RFC 6439 Errata
RFC 6326 (draft-ietf-isis-trill)	Transparent Interconnection of Lots of Links (TRILL) Use of IS-IS	2011-07	RFC 6326 (Proposed Standard) Errata
RFC 6327 (draft-ietf-trill-adj)	Routing Bridges (RBridges): Adjacency	2011-07	RFC 6327 (Proposed Standard)
RFC 6361 (draft-ietf-pppext-trill-protocol)	PPP Transparent Interconnection of Lots of Links (TRILL) Protocol Control Protocol	2011-08	RFC 6361 (Proposed Standard)
RFC 6439 (draft-ietf-trill-rbridge-af)	Routing Bridges (RBridges): Appointed Forwarders	2011-11	RFC 6439 (Proposed Standard)

Figure 5-7 - TRILL RFCs

draft-bond-trill-rbridge-oam-01	RBridges: Operations, Administration, and Maintenance (OAM) Support	2011-03-11	Replaced by draft-ietf-trill-rbridge-oam
draft-bryant-perlman-trill-pwe-encap-00	TRILL using Pseudo-Wire Emulation (PWE) Encapsulation	2005-10-19	Expired
draft-dunbar-trill-directory-assisted-encap-01	Directory Assisted TRILL Encapsulation	2011-10-26	Expired
draft-dunbar-trill-server-assisted-edge-00	Directory Server Assisted TRILL edge	2011-03-07	Expired
draft-eastlake-isis-trill-01	TRILL Use of IS-IS	2010-06-28	Replaced by draft-ietf-isis-trill
draft-eastlake-trill-802-protocols-00	Interaction of RBridges and 802 Protocols	2007-02-27	Expired
draft-eastlake-trill-rbridge-bfd-00	RBridges: OAM and BFD Support for TRILL	2010-10-18	Replaced by draft-ietf-trill-rbridge-channel
draft-eastlake-trill-rbridge-channel-00	RBridges: OAM Channel Support in TRILL	2011-03-07	Replaced by draft-ietf-trill-rbridge-channel
draft-eastlake-trill-rbridge-clear-correct-03	TRILL: Clarifications, Corrections, and Updates	2012-01-08	Replaced by draft-ietf-trill-clear-correct
draft-eastlake-trill-rbridge-fine-labeling-02	RBridges: Fine-Grained Labeling	2011-10-28	Replaced by draft-ietf-trill-fine-labeling
draft-eastlake-trill-rbridge-isis-02	RBridges: Use of IS-IS	2008-11-03	Replaced by draft-eastlake-isis-trill
draft-eastlake-trill-rbridge-more-options-03	RBridges: More Proposed TRILL Header Options	2011-10-17	Expired
draft-eastlake-trill-rbridge-notes-01	Rbridge Notes	2008-12-10	Expired
draft-eastlake-trill-rbridge-options-04	Rbridges: TRILL Header Options	2009-09-30	Replaced by draft-ietf-trill-rbridge-options
draft-gai-perlman-trill-encap-00	An encapsulation for TRILL	2006-10-11	Expired
draft-gibanez-trill-abridge-01	ABridges as RBridges: Transparent Routing with Simplified Multiple Spanning Trees.	2006-06-16	Expired
draft-gray-trill-rbridge-arch-01	The Architecture of an RBridge Solution to TRILL	2006-06-27	Replaced by draft-ietf-trill-rbridge-arch
draft-gray-trill-routing-reqs-01	TRILL Routing Requirements in Support of RBridges	2006-06-27	Replaced by draft-ietf-trill-routing-reqs
draft-hares-trill-multicast-00	Multicast Link State For Rbidges	2005-10-20	Expired
draft-hu-vrrp-trill-00	Extending the Virtual Router Redundancy Protocol for TRILL campus	2011-02-24	Expired
draft-ietf-trill-rbridge-arch-05	The Architecture of an RBridge Solution to TRILL	2008-02-25	Expired WG Document
draft-ietf-trill-routing-reqs-02	TRILL Routing Requirements in Support of RBridges	2007-02-01	Expired (IESG: Dead) WG Document
draft-liyz-trill-reqs-mlt2uni-opt-00	Requirements for multicast to unicast optimization in Trill	2010-07-04	Expired
draft-manral-trill-bfd-encaps-01	Rbridges: Bidirectional Forwarding Detection (BFD) support for TRILL	2011-03-14	Replaced by draft-ietf-trill-rbridge-bfd
draft-mme-trill-fcoe-01	FCoE over TRILL	2011-09-11	Expired RFC Editor State: ISR-AUTH
draft-perlman-trill-rbridge-af-00	RBridges: Appointed Forwarders	2011-03-29	Replaced by draft-ietf-trill-rbridge-af
draft-perlman-trill-rbridge-multilevel-03	RBridges: Multilevel TRILL	2011-10-31	Expired
draft-perlman-trill-rbridge-protocol-00	Rbridges: Base Protocol Specification	2006-02-15	Replaced by draft-ietf-trill-rbridge-protocol
draft-perlman-trill-rbridge-vlan-mapping-00	RBridge VLAN Mapping	2008-10-27	Replaced by draft-ietf-trill-rbridge-vlan-mapping
draft-tissa-trill-cmt-01	Coordinated Multicast Trees (CMT)for TRILL	2012-04-13	Replaced by draft-ietf-trill-cmt
draft-touch-trill-prob-00	Transparent Interconnection of Lots of Links (TRILL): Problem and Applicability Statement	2005-11-21	Replaced by draft-ietf-trill-prob
draft-touch-trill-rbridge-arch-01	The Architecture of an RBridge Solution to TRILL	2006-03-08	Replaced by draft-ietf-trill-rbridge-arch
draft-zebrose-trill-rbridge-mib-00	RBridges: TRILL Base MIB	2009-10-19	Replaced by draft-ietf-trill-rbridge-mib
draft-zhang-trill-vlan-extension-01	To Address the Space Limitation of Inner VLAN	2011-06-30	Expired

Figure 5-8 - TRILL old internet drafts

draft-aldrin-trill-data-center-interconnect-00	TRILL Data Center Interconnect	2012-03-05	I-D Exists
draft-balaji-trill-over-ip-multi-level-05	Connecting Disparate Data Center/PBB/Campus TRILL sites using BGP	2012-03-25	I-D Exists
draft-balaji-trill-te-multi-site-interconnect-00	Interconnecting multiple TRILL sites deploying Traffic Engineering	2012-03-25	I-D Exists
draft-dunbar-trill-directory-assisted-edge-05	Directory Assisted RBridge Edge	2012-03-11	I-D Exists
draft-dunbar-trill-scheme-for-directory-assist-00	Mechanisms for Directory Assisting RBridge	2012-03-06	I-D Exists
draft-eastlake-isis-rfc6326bis-07	Transparent Interconnection of Lots of Links (TRILL) Use of IS-IS	2012-03-10	I-D Exists
draft-eastlake-trill-lldp-00	Transparent Interconnection of Lots of Links (TRILL) Support of the Link Layer Discover Protocol (LLDP)	2012-03-05	I-D Exists
draft-eastlake-trill-rbridge-dcb-03	RBridges: Support of IEEE 802.1Qbb, 802.1Qaz, and 802.1Qau	2012-03-02	I-D Exists
draft-eastlake-trill-rbridge-multi-topo-02	Multiple Topology TRILL	2012-01-10	I-D Exists
draft-hu-trill-pseudonode-nickname-02	RBridge: Pseudo-Nickname	2012-05-15	I-D Exists
draft-hu-trill-rbridge-esadi-04	TRILL: The ESADI Protocol	2012-04-11	I-D Exists
draft-hu-trill-rbridge-vrrp-02	Extending the Virtual Router Redundancy Protocol for TRILL campus	2011-12-29	I-D Exists
draft-hu-trill-traffic-engineering-00	TRILL: Traffic Engineering	2012-01-11	I-D Exists
draft-ietf-trill-clear-correct-03	TRILL: Clarifications, Corrections, and Updates	2012-05-08	AD Evaluation (for 5 days) Submitted to IESG for Publication
draft-ietf-trill-cmt-00	Coordinated Multicast Trees (CMT)for TRILL	2012-04-17	I-D Exists WG Document
draft-ietf-trill-fine-labeling-00	TRILL: Fine-Grained Labeling	2011-12-08 expires soon	I-D Exists WG Document
draft-ietf-trill-rbridge-bfd-05	TRILL: Bidirectional Forwarding Detection (BFD) Support	2012-04-10	In Last Call (ends 2012-06-06) (for 7 days) IESG Telechat: 2012-06-07 Submitted to IESG for Publication
draft-ietf-trill-rbridge-channel-06	TRILL: RBridge Channel Support	2012-05-15	AD Evaluation (for 5 days) Submitted to IESG for Publication
draft-ietf-trill-rbridge-extension-04	TRILL: Header Extension	2012-05-02	In Last Call (ends 2012-06-06) (for 7 days) IESG Telechat: 2012-06-07 Submitted to IESG for Publication
draft-ietf-trill-rbridge-mib-07	Definitions of Managed Objects for RBridges (Routing Bridges)	2012-04-05	IESG Evaluation::AD Followup (for 125 days) Submitted to IESG for Publication
draft-ietf-trill-rbridge-oam-02	Routing Bridges (RBridges): Operations, Administration, and Maintenance (OAM) Support	2012-03-12	I-D Exists WG Document
draft-ietf-trill-rbridge-options-06	RBridges: Further TRILL Header Extensions	2011-12-02 expires soon	I-D Exists WG Document
draft-ietf-trill-rbridge-vlan-mapping-07	RBridges: Campus VLAN and Priority Regions	2012-04-27	I-D Exists WG Document
draft-manral-isis-trill-multi-topo-03	Multiple Topology Routing Extensions for Transparent Interconnection of Lots of Links (TRILL)	2011-12-30	I-D Exists
draft-mrw-trill-over-ip-01	Transparent Interconnection of Lots of Links (TRILL) over IP	2012-03-12	I-D Exists
draft-perlman-trill-rbridge-data-encoding-01	RBridges: TRILL Link Data Optimizations	2012-01-14	I-D Exists
draft-rohit-trill-proactive-oam-00	Pro-active connectivity monitoring for TRILL	2012-02-24	I-D Exists
draft-tissa-trill-mt-encode-00	Multi Topology Encoding within TRILL data frames	2012-03-26	I-D Exists
draft-tissa-trill-multilevel-00	Default Nickname Based Approach for Multilevel TRILL	2012-02-21	I-D Exists
draft-tissa-trill-oam-03	ICMP based OAM Solution for TRILL	2012-01-06	I-D Exists
draft-tissa-trill-oam-req-01	Requirements for Operations, Administration and Maintenance (OAM) in TRILL	2012-05-04	I-D Exists
draft-xl-trill-over-wan-00	Extending TRILL over WAN	2011-12-06 expires soon	I-D Exists
draft-yizhou-trill-multi-destination-ping-02	OAM tool for RBridges: Multi-destination Ping	2012-05-03	I-D Exists
draft-yizhou-trill-tree-selection-00	VLAN based Tree Selection for Multi-destination Frames	2012-03-05	I-D Exists
draft-yong-trill-trill-o-mpls-01	Transparent Interconnection of Lots of Links (TRILL) over MPLS Pseudo Wires	2012-03-12	I-D Exists
draft-zhang-trill-aggregation-02	RBridge Aggregation	2012-05-02	I-D Exists
draft-zhang-trill-dvlan-auto-00	Auto-Configuration of Designated VLANs	2012-01-10	I-D Exists
draft-zhang-trill-mtu-negotiation-02	TRILL IS-IS MTU Negotiation	2011-12-13	I-D Exists
draft-zhang-trill-multi-topo-rpfc-00	Reverse Path Forwarding Check under Multiple Topology TRILL	2012-01-30	I-D Exists
draft-zhang-trill-vlan-assign-03	Adaptive VLAN Assignment for Data Center RBridges	2012-03-06	I-D Exists

Figure 5-9 - TRILL active internet drafts

6 Shortest Path Bridging Protocol

802.1aq *Shortest Path Bridging* (SPB) is the IEEE's proposal to replace the older spanning Tree Protocols, IEEE 802.1D STP, IEEE 802.1w RSTP and IEEE 802.1s MSTP. Its approach is considered both an advance over *Multiple Spanning Trees Protocol* (MSTP) and the natural evolution of 802.1ah [28]. 802.1ah was developed by Nortel, under the *Provider Backbone Bridging* name and standardized by IEEE back in 2008.

On May 2012 the IEEE approved the new 802.1aq standard [26].

SPB is clearly tailored for carrier/backbone networks. It uses the *Intermediate System-to-Intermediate System* (IS-IS) link-state protocol to discover and advertise network topology and compute shortest path from all bridges in the SPB network.

SPB combines the power of ISIS link state protocol with Ethernet, delivering more scalability and performance to Layer 2 networks, preserving the plug-and-play nature of Ethernet and reducing services provisioning complexity.

SPB is a *service-aware* Layer 2 link state routing protocol. It has two modes: ***SPBV*** (*Shortest Path Bridging VLAN)* and ***SPBM*** (*Shortest Path Bridging Mac-in-Mac*). SPBV uses IEEE 802.1ad Q-in-Q encapsulation, whereas SPBM uses IEEE 802.1ah Mac-in-Mac encapsulation.

Chronologically, SPBV came first to address MSTP scalability and convergence limitations. SPBV targets scenarios where only the VLAN tag is used to separate traffic service instances, whereas SPBM is intended to provide complete isolation between client traffic service instances. SPBM aims larger networks.

Both SPBV and SPBM provide interoperability with Spanning Tree.

Supporting SPB technology development and implementation are some big networking companies like Avaya, Alcatel, Hauwei and Cisco Systems. Curious is the fact that Cisco Systems supports both SPB and TRILL movements.

SPB derives from technology deployed for several years, and that is pointed as an immediate advantage over TRILL, which is completely new technology.

6.1 SPB Overview

Both SPB variants use ISIS to compute shortest path trees between nodes:

- SPBV uses a Shortest Path VLAN ID (SPVID).
- SPBM uses a combination of Backbone MAC (BMAC) and Backbone VLAN ID (BVID).

SPBM version of SPB is the one more publicized, documented and targeted to battle TRILL. Therefore, the remaining sections of this chapter will focus on SPBM mode only.

I-SID

I-SID stands for *Service Instance Identifier*.

SPB breaks with the traditional 802.1Q 12-bit VLAN ID, using a 24-bit IEEE I-SID. This leverages the approximate 4-thousand VLANs into 16-million unique services.

A single VLAN or multiple VLANs are mapped to an I-SID at the network edge, abstracting the service from the network. Then, SPB (ISIS) creates shortest paths for those I-SIDs.

SPBs I-SID is a service ID provisioned at the network edge. This way, SPB reduces the network core to just an Ethernet based link state protocol.

Relies on ISIS

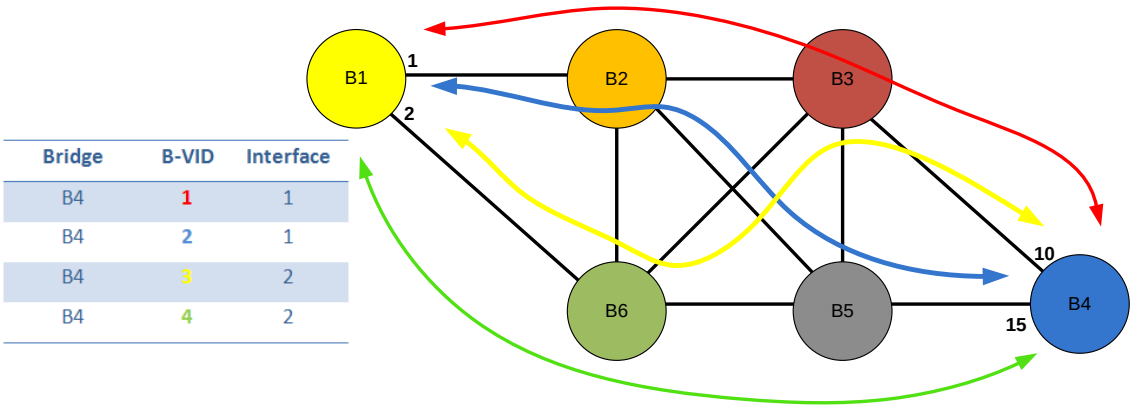
SPB uses ISIS link state protocol to advertise both topology and logical network membership [25]. ISIS builds optimized paths and allows load balancing between equal cost paths.

Following the link state protocol behaviour, SPB performs computations individually, but based on a common network view (link state database).

ISIS forms adjacencies between neighbour SPB bridges and advertises bridge's backbone MAC addresses (BMAC) and user services configured. Link state databases are formed, equal on all SPB network bridges, and consequently best paths towards all SPB bridges are computed.

SPB take always into consideration the service provisioned, using I-SIDs to map VLANs into services.

SPB ***forwarding databases*** (FDB) contains all possible paths from one bridge towards all others. For example, Figure 6-1 illustrates Bridge B1 possible forwarding database entries towards bridge B4.



Bridge	B-VID	Interface
B4	1	1
B4	2	1
B4	3	2
B4	4	2

Figure 6-1 - SPBM Unicast Forwarding Database

Note the bi-directionality of the paths and the uniquely assigned B-VIDs for all four possible paths.

For bridge B4, the FDB for B1 would be the reciprocal, like Table 2 shows.

Bridge	B-VID	Interface
B1	1	10
B1	2	10
B1	3	15
B1	4	15

Table 2 - Bridge B4 FDB

Therefore, ISIS computes shortest path trees based on link metrics and does not block any link like STP does, creating unicast, symmetric paths between every two bridges.

In SPB, ISIS advertises service memberships. When a new service is provisioned, service membership is advertised throughout the network via ISIS. For example, suppose nodes A@B1, B@B2 and C@B3 have a common service running and need to communicate over VLAN 150. After the provision of VLAN 150 with I-SID 200 in bridges B1, B2 and B3, bridge's membership on I-SID 200 is advertised to all other SPB bridges via ISIS.

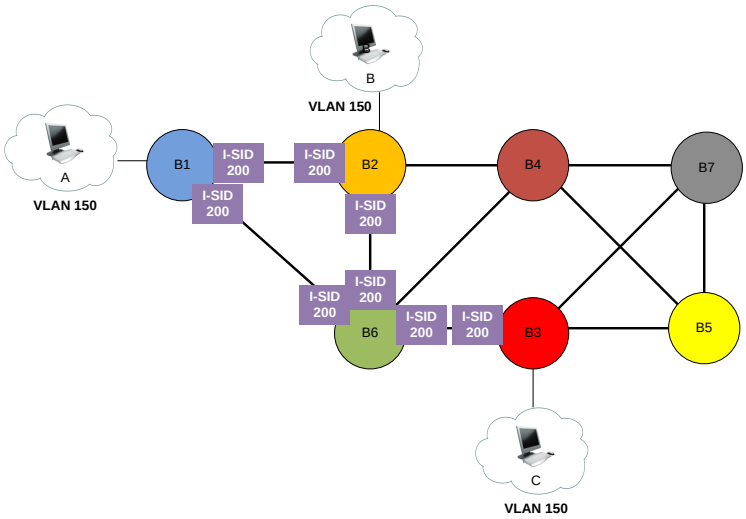


Figure 6-2 – VLAN 150 / I-SID 200 mapping

So, for I-SID 200, SPB calculates three shortest path trees. Figure 6-3 (in blue), Figure 6-4 (in orange) and Figure 6-5 (in red) shows those three shortest path trees.

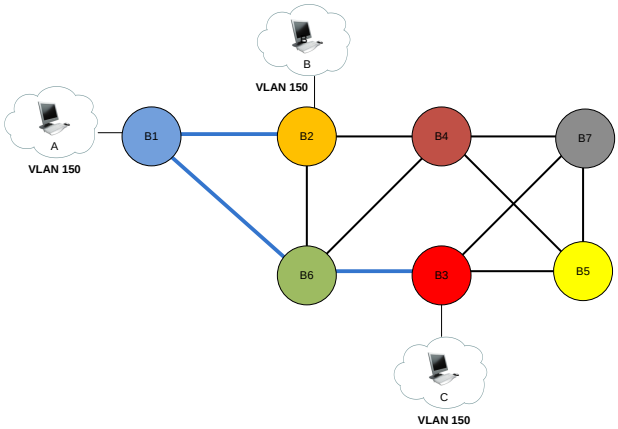


Figure 6-3 - Shortest path tree for source A

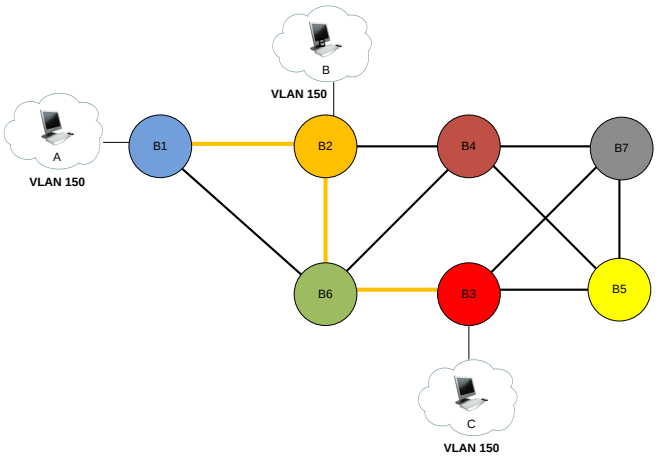


Figure 6-4 - Shortest path tree for source B

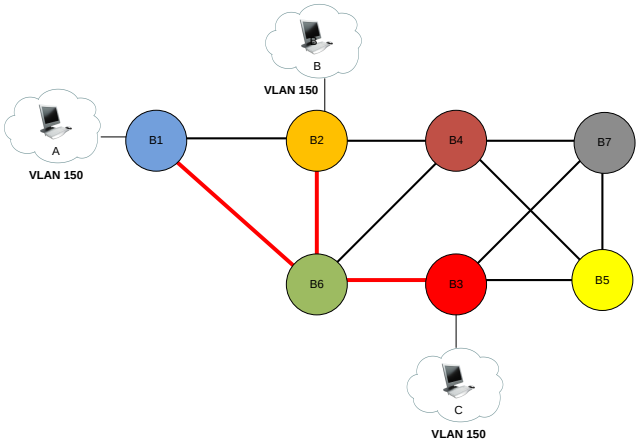


Figure 6-5 - Shortest path tree for source C

Any path between two nodes is always bidirectional and the shortest; TRILL's may not be bidirectional.

Traffic arriving on bridge B1 over VLAN 150 is forwarded via the shortest path of Figure 6-3, traffic arriving on bridge B2 and VLAN 150 via the one of Figure 6-4 and on VLAN 150 of bridge B3 the path of Figure 6-5 is used.

Backbone Edge Bridges (BEBs) handle the boundary between the domains SPB and 802.1Q, mapping VLANs into I-SIDs. In our example, B1, B2 and B3 are BEBs.

If machine A tries to communicate with C and bridge B1 does not know C MAC address, it floods the traffic towards B2 and B3, the other members of I-SID 200 (VLAN 150). If B1 knew C MAC address, it would only forward the traffic towards B3, because each BEB maintain a mapping between client MAC addresses (CMACs) and Backbone MAC Addresses (BMACs). In our case, B1 would have machines B and C, which are on VLAN 150 connected to B2 and B3, with BMAC of B2 and B3, respectively.

Handling unicast traffic for known destinations is pretty similar between 802.1aq and TRILL, at control plane level: ISIS builds the topology and SPB/TRILL creates the shortest paths and assigns traffic to those paths. At forwarding data plane, they are

very different, as TRILL rewrites frames in a hop-by-hop basis, whereas 802.1aq does not change the frame throughout all SPB network, as next section will present.

However, handling multicast, broadcast and unknown destination unicast traffic is quite different. SPB uses the same paths as known unicast, but it floods the traffic to I-SID members. TRILL, in turn, uses distributions trees, like described in the previous TRILL chapter.

Ethernet Frames Encapsulation

SPB supports two types of encapsulation: 802.1ah (MAC-in-MAC or Provider Backbone Bridges) and 802.1ad (Q-in-Q or Provider Bridging).

802.1ah is the one *taken seriously* in SPB, and it's used in SBPM mode.

When traffic arrives at SPB bridges, the original frames are encapsulated with:

- Backbone addresses:
  - B-DA – Backbone destination address.
  - B-SA – Backbone source address.
  - B-VID – Backbone VLAN ID.
- I-SID – Service instance ID

Backbone addresses B-DA and B-SA are addresses used by SPB bridges to communicate within the SPB domain. This way, SPB improves network stability by *hiding* client MAC addresses from backbone SPB network.

Figure 6-6 illustrates the encapsulation fields and Figure 6-7 shows how SPB hides clients MAC addresses in the backbone, when machine A is communicating with machine B, over VLAN 150, mapped to I-SID 200.

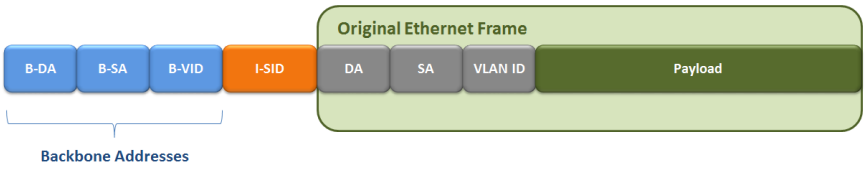


Figure 6-6 - 802.1ah frame format

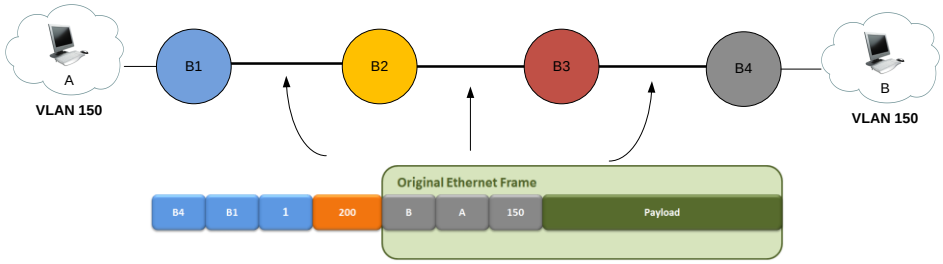


Figure 6-7 - Client MAC addresses are hidden in the core

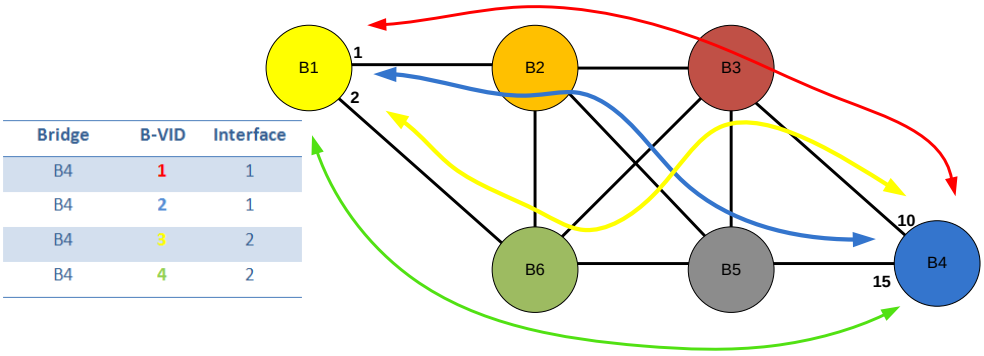
Note from Figure 6-7 the frame is not rewritten by the bridges; it leaves the headers unchanged throughout the SPB network.

Load Balancing

SPB, empowered by ISIS, allows all paths to be active with multiple equal cost paths. This allows much larger layer 2 topologies with faster convergence times, and improves bandwidth and redundancy between all devices, allowing traffic to be load shared across all paths.

All physical connectivity is used, because SPB loop avoidance uses a global view of network topology, the ISIS link state database.

Previous Figure 6-1 illustrates the four possible paths between two bridges.



Bridge	B-VID	Interface
B4	1	1
B4	2	1
B4	3	2
B4	4	2

Figure 6-1 - SPBM Unicast Forwarding Database

Multicast / Broadcast Support

Multicast traffic is fully supported by SPB. The routing is on symmetric shortest paths and several equal cost shortest paths can be used.

When a BEB doesn't know the unicast destination MAC address, it uses a B-DA multicast address to speak to all other members of that I-SID. The same flooding occurs for multicast and broadcast traffic.

Failure Recovery

ISIS, *per-se*, normally advertises link failures and, consequently, new Forwarding Databases are computed. Since ISIS does not advertise Ethernet addresses, the network core bridges remain unaffected by leaf links failure.

SPB delivers, like TRILL, fast convergence as ISIS link state database provides global view of network topology. However, SPB provides a much faster failure detection mechanism than TRILL's ISIS hello message loss detection, via IEEE 802.1ag CFM messages support [27].

Another great advantage of SPB is that unicast and broadcast/multicast connectivity is restored in parallel, with no need to employ other procedures to restore distribution trees, like TRILL needs.

6.2 SPB Protocol Considerations

Even though two years late, since the announced milestone, SPB is now an IEEE standard.

Shortest Path Bridges protocol is very similar with TRILL on many factors. Hence, its ISIS inheritance allows SPB to provide also fast convergence, equal-cost load balancing and shortest paths calculations. However, they have completely different forwarding paradigms: TRILL acts like a Layer 3 protocol, rewriting the Layer 2 address hop-by-hop, whereas SPB behaves like a regular Layer 2 protocol, not changing the frame until it gets to its destination.

Like TRILL, SPB is complex and processor intensive. ISIS expertise is needed to maintain and troubleshoot SPB networks. Also, SPB forwarding databases size will pull up resources demands for bridges.

Briefly comparing SPB and TRILL, taking into consideration the theoretical concepts, I would go for SPB as the service approach is much more interesting and the flooding traffic method quite superior.

It is interesting to see the two concurrent standards, very similar in many aspects, to be standardized close to each other. The battle continues and many Internet forums brandish hot discussions.

Nowadays, however, team managers are struggling against tight budgets and CAPEX reducing demands by top management, and those two new protocols will require significant investments on network gear. Additionally, also OPEX would increase as operational and engineering personnel would need to attend training courses or lectures, for example.

PART III – SCS New Approach

Part III introduces the *Self-Configurable Switches Protocol* strategy and details all its processes, mechanisms, features and deployment plan.

- Chapter 7 – Self-Configurable Switches Protocol
- Chapter 8 – SCS Processes

7 Self-Configurable Switches Protocol

Replacing the spanning tree protocol is not an easy task. Spanning tree algorithm is already widely deployed, studied, enhanced and deeply understood by the networking professionals. It's running for decades! Moreover, any new protocol is always received by the technical community with deep reservations and scepticism, independently of its potential and advantages. Additionally, the learning curve is always a barrier for new protocols implementation.

However, there is the strong conviction that the ideas behind the SCS protocol are feasible and would greatly improve performance, simplicity, self-assessment and configuration of today's bridges, without revolutionizing their simple core function: forwarding frames in a transparent, wire speed way.

Self-Configurable Switches Protocol (SCS) is suitable for any Ethernet network topology, resilient to topology changes, intelligently manages the load balancing between equal cost links, efficiently forwards traffic even around looped layouts and, as it is self-configurable, protects the network of one of the most critical threat networks have to deal in daily basis, the human factor.

As the name implies, SCS empowers an Ethernet bridge with self-configuring capabilities, mainly to deal with the multicast, broadcast and unicast traffic that must be flooded throughout all bridge ports, as well as with the topologies changes that might occur in a network.

The performance is taken seriously, as also stability and resilience to network events, are.

The first major characteristic of SCS is that it is not STP backwards compatible, meaning no interoperability between SCS and any STP flavour will exist. There are some good protocols with great ideas that decrease significantly their power just because they allow *inferior* STP bridges connect to their domain, downgrading their features and performance to suit the *inferior* STP bridge's needs. SCS won't do it and will manage STP bridges as *external unreliable looped networks*, as future chapters will describe.

SCS is fully committed to provide legitimate neighbours connections, implementing a simple scheme that protects the SCS network about reckless insertion of new SCS bridges into the network, which in STP case can represent a huge menace to network stability.

This chapter will overview SCS protocol features, characteristics and potentialities.

7.1 Self-Configurable Switches Overview

SCS protocol targets the mitigation of most the issues Spanning Tree raises. Network resources misuse, suboptimal paths, stability easily permeable to external events, fragile over topology changes and poor converge times, are some of the main problems that networks running Spanning Tree must face. SCS, by the contrary, is resilient, fast, robust and optimizes resources usage.

SCS algorithm is considerably different than Spanning Tree, even in its motto. Spanning tree aims to create an unique loop-free topology within the network, breaking all the redundant paths. By the contrary, SCS assumes the network as looped and works over it using as much as equal cost paths available, in order to maximize the redundancy provided and the investment made.

The main characteristics of SCS protocol, which undoubtedly allow it to supersede the Spanning Tree Protocol in all vectors, are described in the following sections of this chapter.

Two interface types

SCS distinguishes the interfaces by the type of network device attached to the bridge. Interfaces connecting SCS bridges are automatically classified as ***bridge interfaces***. Two SCS bridges are always connected via *bridge interfaces*. Any other type of network device that connects to a SCS bridge uses ***host* interfaces**, even STP bridges. The following *STP incompatible* section describes how SCS deals with STP bridges connected to the SCS domain.

This interface classification is very important to deal with the traffic that needs to be flooded throughout the network, as SCS methods are quite different for *host* and *bridge* interfaces, as it will be described in future sections.

STP incompatible

SCS is not compatible with any flavour of Spanning Tree Protocol.

STP has many flaws and completely different timers and processes among STP protocol versions. Thus, it would not be either feasible or logical to allow STP to participate in SCS networks.

It is mandatory by design that SCS disrupts with STP premises. This does not mean that a STP bridge is not allowed to connect into a SCS network; it means that a STP bridge will be treated *specially*. Likewise, an entire STP network could be connected to a SCS network. However, that could be dangerous for the STP domain and SCS would need to protect itself from the STP instabilities!

Recall that STP builds a loop-free topology, blocking all redundant paths; SCS allows and favours redundant paths. Thus, SCS can be pictured into STP networks as a giant hub; STP can be pictured into SCS as a single host. Figure 7-1 and Figure 7-2 shows those representations.

Interconnecting STP and SCS domains can, thus, endanger STP stability and its loop-free topology.

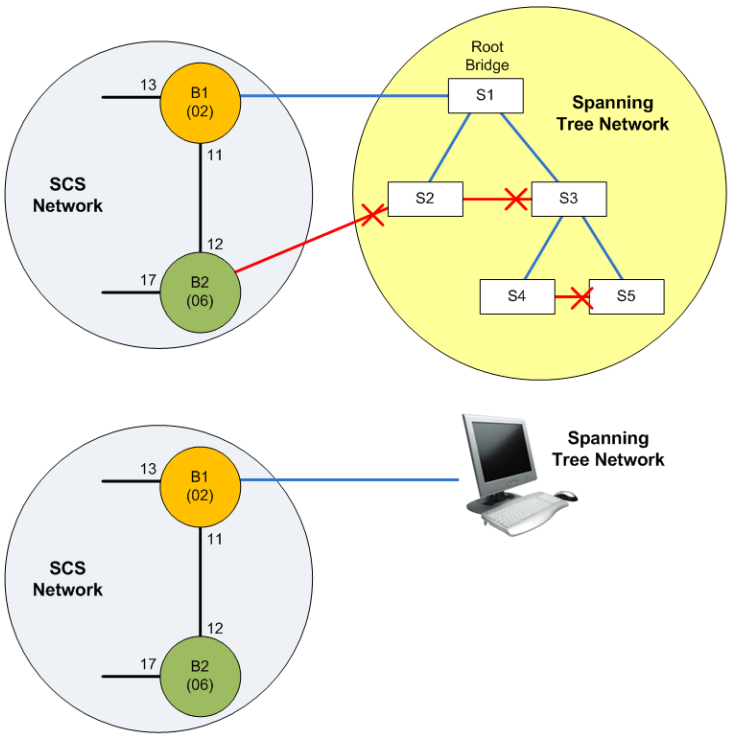


Figure 7-1 - Spanning Tree network is seen by SCS as a single host

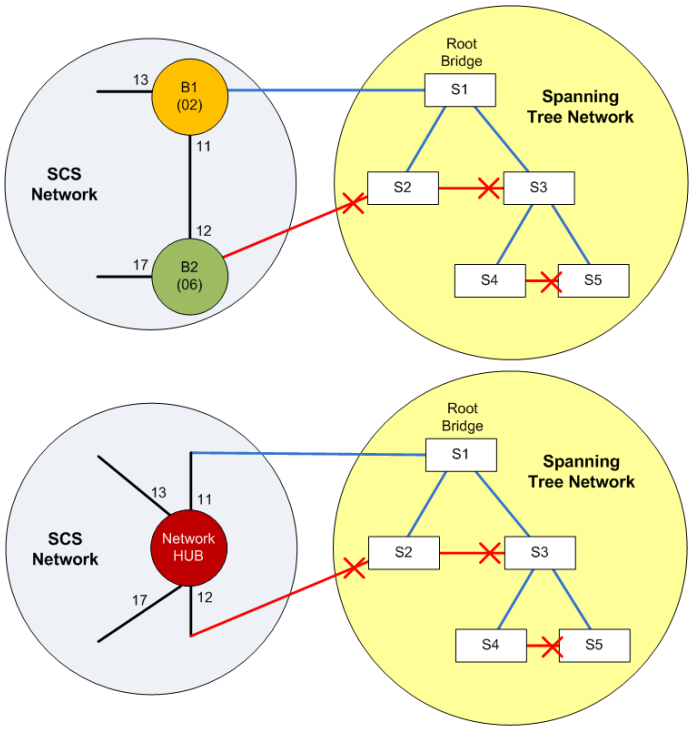


Figure 7-2 - SCS network is seen by STP as a hub

Interfaces connecting to STP bridges are SCS *"special" host interfaces*. As soon as a SCS bridge detects STP BPDUs via one of its ports, it starts protecting itself from the

STP network, reducing by half the ***HelloTimer*** interval through that interface, to 0.5 seconds by default. Moreover, the SCS bridge prevents that interface changing from *host interface* to *bridge interface*, i.e., if it detects another SCS bridge via that path, it shuts down the interface.

HelloTimer interval will be described in future chapters but it can be shortly defined as the *keepalive* interval for control messages (*SCS Hellos*) between adjacent SCS bridges, which allow bridges to establish neighborship.

These two mechanisms speed up *Neighbour Auto-discovery Mechanism* by a factor of 2, meaning bridges will detect other SCS bridges 2 times faster and immediately isolate that path from SCS network, if another SCS bridge is detected via STP networks. *Neighbour Auto-discovery Mechanism* is explained in the next chapter.

When interconnecting SCS and STP networks, two different situations might occur:

1. Single boundary
2. Multiple boundaries

In the first situation, the whole STP network is considered a *host* and the SCS interface is classified as *host interface*. In this case, no loop exists between both domains and flooding is never fed back to the original network.

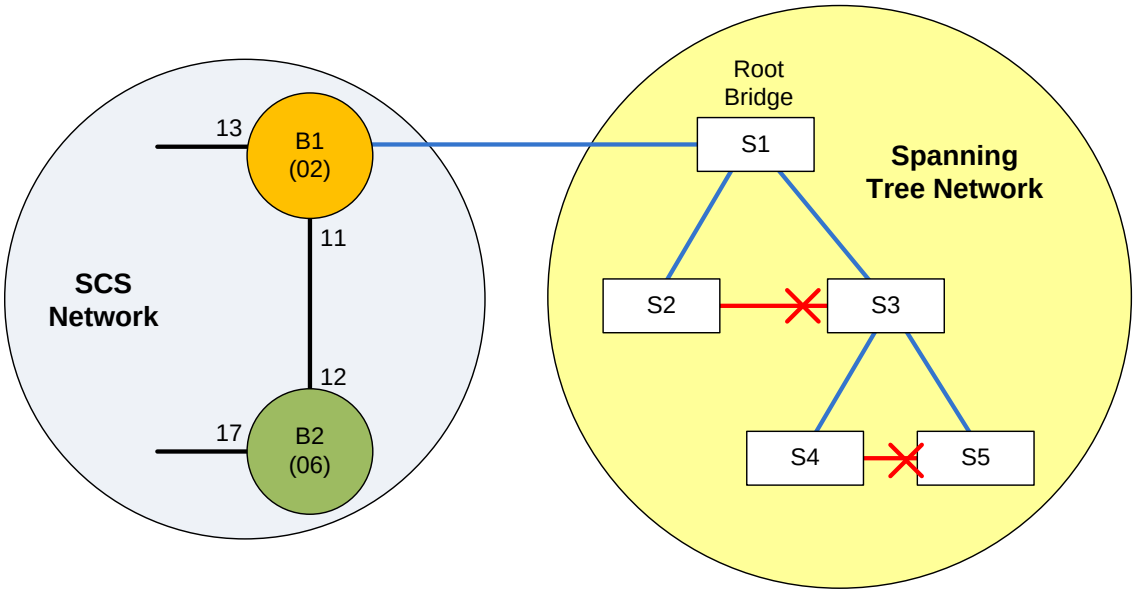


Figure 7-3 - Single boundary between SCS and STP

It this case, B1 SCS Hello's never come back to B1 neither reaches other SCS bridge. That way, SCS and STP networks are communicating and safe.

The second situation requires protection by SCS. The following figure illustrates one possible multiple boundaries issue:

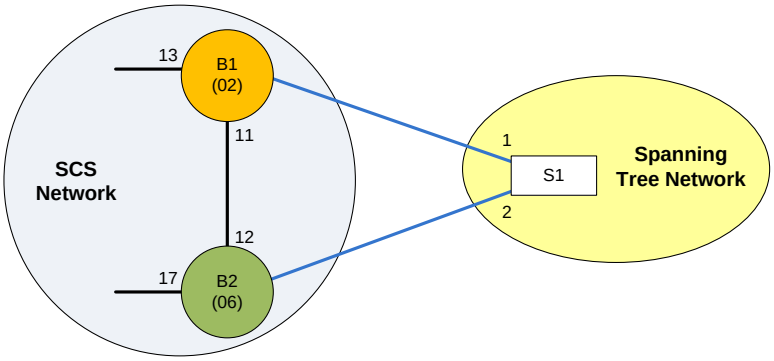


Figure 7-4 - Multiple boundaries scenario

In this case, STP will converge and one of the two S1 interfaces will be STP blocking.

Spanning Tree standards state that bridges send BPDUs to a well-known multicast MAC address: *01:80:C2:00:00:00*. Figure 7-5 shows a sample packet capture of a STP BPDU.

```
⊞ Frame 23: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
⊟ IEEE 802.3 Ethernet
  ⊟ Destination: Spanning-tree-(for-bridges)_00 (01:80:c2:00:00:00)
      Address: Spanning-tree-(for-bridges)_00 (01:80:c2:00:00:00)
      .... ...1 .... .... .... .... = IG bit: Group address (multicast/broadcast)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
  ⊟ Source: Cisco_68:05:e6 (00:15:2b:68:05:e6)
      Address: Cisco_68:05:e6 (00:15:2b:68:05:e6)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    Length: 39
    Trailer: 00000000000000
⊞ Logical-Link Control
⊞ Spanning Tree Protocol
```

Figure 7-5 - Spanning Tree packet capture

Suppose S1's interface 1 superior regarding interface 2, in terms of STP. When B1 receives the multicasted STP BPDU from S1's interface 1, it will forward the BPDU to B2, using a specific SCS packet designated ***SCS Unicast Flood*** packet. SCS Unicast Flood packets are unicast packets, which contains the *broadcast* packets to be transmitted. SCS UF packets are covered later.

B2 receives the SCS UF packet from B1 and forwards the original BPDU into S1's interface 2. S1 detects its own BPDU and blocks interface 2. This would break the loop created upon connecting both networks.

However, SCS won't trust STP and B1 as soon as it detects STP BPDUs starts sending SCS Hellos, each 500 ms. As one of S1 interfaces 1 or 2 is not forwarding, those SCS Hellos are discarded.

The same applies to the opposite direction, i.e., from S1 interface 2 towards interface 1. To keep the example simple, we will consider only the first direction.

In normal circumstances, the STP network will behave correctly and no loop would occur. Thus, SCS Hellos from B1 would never reach B2. However, SCS is forced to protect itself from potential STP stability issues. If by some means the STP network builds a loop topology, like Figure 7-6 shows, the SCS Hellos from B1 would quickly reach B2 and B2 would immediately shut down interface 0b, breaking the loop. The opposite would also happen with B1 shutting down 0a, that way, SCS protected itself from STP problematic behaviour, isolating STP misbehaved network from SCS domain.

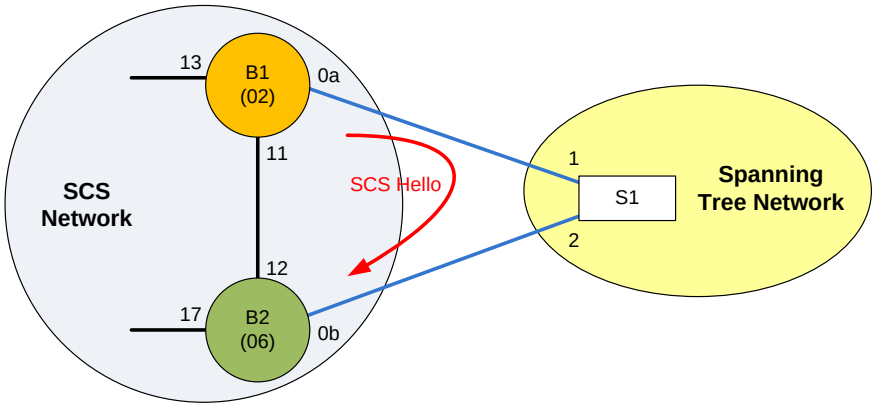


Figure 7-6 - STP loop example

In worst case scenario, SCS network protects against a STP loop two times faster than RSTP/MST and four times than STP, if RSTP/MST/STP could ever overcome the looped situation and regain the loop-free topology.

Figure 7-1 has already shown an entire STP network being recognized as a single host. This picture also represents another case of multiple boundaries between SCS and STP.

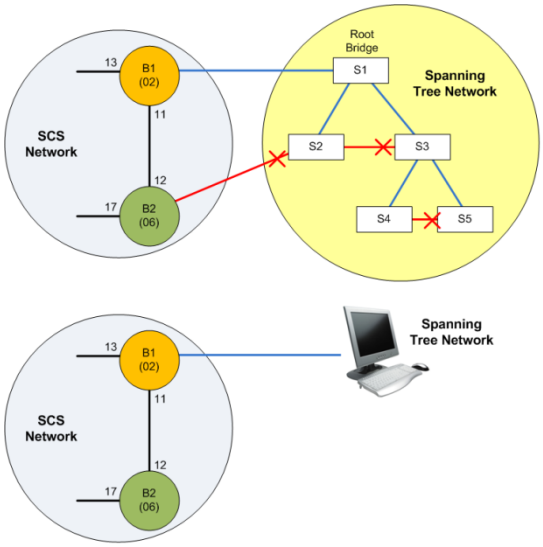


Figure 7-1 - Spanning Tree network is seen by SCS as a single host

Although STP would block, for example, S2 bridge interface towards SCS bridge B2 by receiving superior BPDUs from bridge S1, it is not reasonable to rely on an inferior external protocol to protect SCS domain against uncontrolled looped traffic. Consequently, SCS protects itself by speeding up the SCS Hello transmission to quickly detect a STP loop. As soon as a SCS bridge detects a SCS Hello via such an interface, the interface is immediately shut down. In the worst case, it would take 500ms to detect the STP loop.

SCS allows the network administrator to completely disable the creation of SCS – STP boundaries. In such a case, as soon as a SCS bridge detects a STP BPDU, the interface is shut down by SCS.

The reason for such segregation relies on performance issues introduced by SCS-STP boundaries in SCS network. In fact, it raises performance issues on both networks, but it is only our concern the SCS ones.

STP BPDUs are multicasted by STP bridges, as stated before. Original STP sends BPDUs each 2-second, as RTSP/MSTP each 1-second, by default. Thus, SCS bridges will encapsulate BPDU traffic and *flood it* over the SCS network each 1/2-seconds for… nothing useful to SCS; just to allow STP to block redundant paths via SCS network. That way, an administrator can clean BPDU trash traffic out of SCS network, by disallowing SCS – STP boundaries. That's the pinnacle of SCS's STP incompatibility.

Almost-zero configuration

The only parameter that is mandatory to configure in a SCS bridge is the ***neighborship key***. This key is shared by all bridges and identifies the ones that are allowed to establish neighbour relationships between them, creating the SCS domain. This key usage will be detailed later in the *Safe neighborship se*ction.

All the processes and mechanisms inherent to SCS are self-configurable and self-assessed by the bridge itself.

Bridges remain bridges but...

SCS doesn't change the way current bridges transparently forward traffic to known destinations. However, the way it deals with unknown destination traffic is radically different on *bridge interfaces*; for host ports, bridges behaviour remains unchanged.

Regular bridges follow accordingly:

1. Any known destination unicast traffic is forwarded through the correct port.
2. Any unknown destination traffic is flooded throughout all ports, except the incoming one.

What SCS does differently is the way *flooded* traffic is forwarded, breaking up the second rule, which relates with broadcast, multicast and unknown destination unicast traffic. SCS changes this second rule to:

2a. Any unknown destination traffic is flooded throughout all ***host*** ports, except the incoming one.

2b. Create ***SCS Unicast Flood*** packets containing the unknown destination traffic and send them towards all neighbour SCS bridges (*bridge* interfaces).

SCS Unicast Flood packets are covered on *Flood Control Process* section of the *SCS Processes* chapter.

This SCS behaviour removes all *flooded* traffic from inter-bridge links. This is how SCS manages traffic flooding within looped topologies: it unicasts all the unknown destination traffic towards all neighbour bridges and ask some of its neighbours to forward that traffic onwards.

Topological view of a flat network

SCS Bridges do not participate in any kind of Root Bridge election, neither build hierarchical topologies based on some bridge being *superior* over others.

SCS bridges create neighborship relations with directly attached bridges, one for each link between them. Each bridge is then responsible for advertising to its neighbours all the paths it knows towards all other SCS bridges in the campus. With that information exchanged, they construct their own view of the network, creating a topology database.

Each bridge will manage five different, yet simple, tables:

- ***FwdT*** – Forwarding Table – like current bridges CAM Table.
- ***NB Table*** – Neighbour Table – contains information about adjacent neighbours.
- ***TP Table*** – Topology Table – identifies all bridges in the network and the best path from the bridge towards them.
- ***FT*** – Flooding Table – holds entries about how to reach certain remote bridge, via an adjacent neighbour.
- ***DT*** – Delegation Table – specifies the neighbours and destinations, for which the bridge is delegate of.

All these tables and the associated methods and procedures will be described in detail on *SCS Processes* chapter.

Internal loop protection

As said above, SCS bridges create neighbour relationships with adjacent bridges. There is a keepalive mechanism to control such relationships, via *SCS Hello* messages exchange.

If by some means a bridge receives its own keepalives, that incoming interface is automatically shut down by SCS and an error is reported.

SCS Frames Format

SCS protocol messages are encapsulated in the payload of Ethernet frames, using a new EtherType to identify SCS traffic: ***0x0834***. Figure 7-7 illustrates the SCS Ethernet frame format.

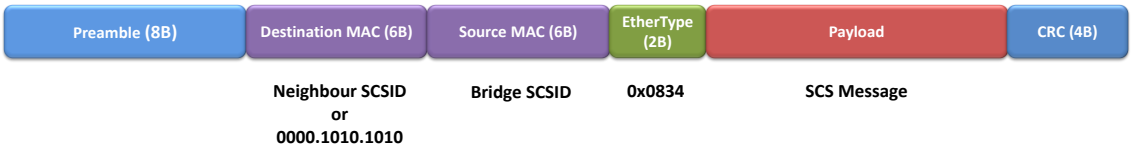


Figure 7-7 - SCS frames format

SCS frames are regular Ethernet frames that carry SCS information in the *payload*.

The *EtherType* is 0x0834 to distinguish the SCS protocol and the *Source MAC* is the ***SCSID***, i.e., the 48-bit MAC address of the SCS bridge.

The *Destination MAC* must be the SCS neighbour bridge SCSID. However, for *SCS Hello* frames (the keepalive frames to keep neighbour relationships up), as the bridge doesn't know yet the neighbour bridge's SCSID, SCS uses the 48-bit MAC address 0000.1010.1010; this MAC address is defined as ***SCS Ethernet Destination MAC***[2].

Optimized frame forwarding

SCS allows traffic flow optimization, comparing to Spanning Tree.

Spanning Tree blocks all redundant paths and traffic always flow through the same, unique, loop-free topology calculated by the Spanning Tree Algorithm.

SCS favours redundant paths potentiating traffic load balancing. It is important to note that SCS load balancing it is not a typical routing protocol load balancing scheme, where traffic can flow via equal cost redundant paths (some even allow unequal load balance). SCS load balance only fits on that characterization for parallel links between the same two bridges.

SCS allow load balance of traffic in terms of link occupation, i.e., as SCS do not block any link, they are all available to be used. That way, for certain destinations, some links will be used, whereas for other, different links will be selected.

This brings extra advantages for network resource utilization, as traffic flow in a stable network is deterministic, like in STP, but now via all available links on the network.

Another major upgrade SCS introduces is in the FdwT. FdwT will no longer contain just a single entry for a source. In case of parallel links to the same bridge, the SCS bridge will replicate all entries known through a link to all other parallel links to the same neighbour, allowing traffic load balancing over those parallel links, without the need for EtherChannel or any other kind of link aggregation protocol.

The *Forwarding Process* section on the next chapter will detail this frame forwarding optimization.

Safe neighborship

SCS delivers natively *neighbour bridges authentication*.

The *neighbour bridges authentication* is a simple scheme used by SCS just to protect the SCS network about reckless insertion of new SCS bridges into the network. The simple clear-text 6-bit key is not suitable for security purposes, so it does not target secure communications neither bridge protection; it aims SCS network protection against careless insertion of bridges.

The *Spanning Tree Protocol* chapter presented several problems about STP networks. The section *New Bridges in the Layer 2 production network* demonstrated the weakness of STP regarding unofficial insertion of STP bridges into the production network and the connectivity problems that might arise from those insertions.

SCS uses the *neighbour bridges authentication* mechanism to protect against such insertions. With a simple 6-bit key, an official SCS bridge running on a production

[2] SCS Ethernet Destination MAC should be a 48-bit Mac address matching the OUI (Organizationally Unique Identifier) reserved for SCS. For this paper purposes only, the MAC 0000.1010.1010 is used as an example and it is not in any form related with SYTEK INC, the legal owner of 00-00-10 OUI.

network effectively recognizes with which new bridges it should establish new neighbour relationships. If this key mismatches, the SCS bridge will not allow the other bridge to inject and receive traffic to/from the production network.

Flooding propagation protection

Ethernet bridging lacks some mechanism to limit the flooding propagation in the advent of a loop. Moreover, all spanning tree protocols, STP, RSTP and MSTP, being control protocols, should have included such a mechanism. Consequently, a loop in an Ethernet network causes flooded traffic to be indefinitely forwarded between bridges and raises serious connectivity problems.

SCS protects against such events is two distinct ways:

1. All flooding traffic received is converted in *special unicast traffic* exchanged between bridges: SCS UF packets. That way, all flooded traffic is limited to a local bridge.
2. TTL counter is enforced in SCS UF packets, to limit eventual traffic replication over a network loop.

Inverted Flooding

Like stated in earlier chapters, Ethernet bridges promiscuously listen the incoming traffic, remember the source address of that traffic, and forward traffic based on that learning method. If the destination is unknown, the bridge *floods* the traffic throughout all ports, except the incoming one, even for unicast traffic: this is known as *unknown unicast flooding*.

This may introduce several issues, especially during topology changes.

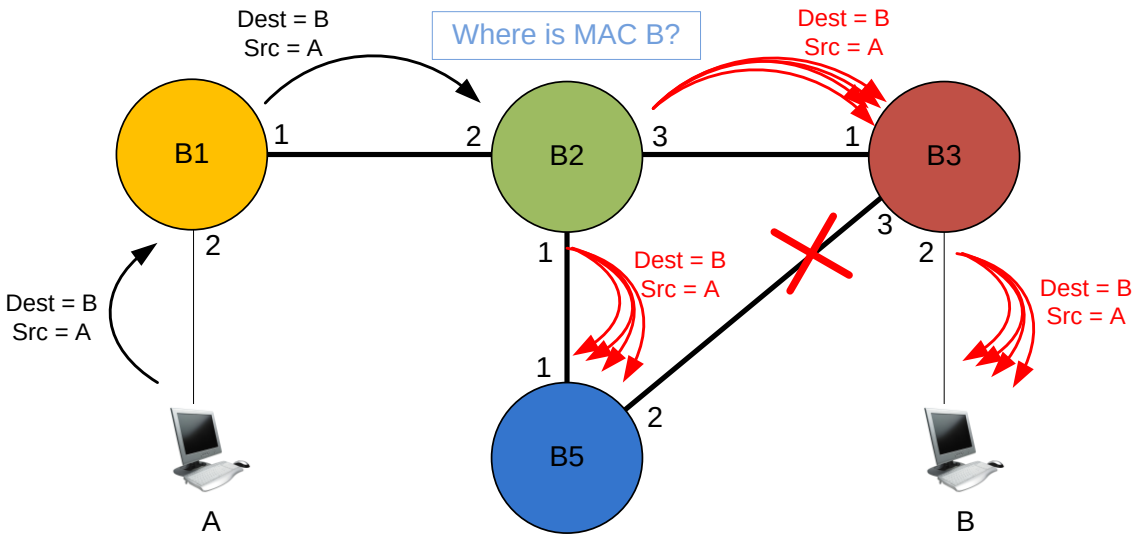


Figure 7-8 - Ethernet bridges unknown unicast flooding

For example, in Figure 7-8, if link $B3_3$-$B5_2$ fails, even after the topology shifted to a final state, all traffic towards host B would be unicast flooded throughout all network,

until host B sends a frame and bridge B2 learns host B MAC address via its port 3. This behaviour is pictured in red in the above figure.

Conversely, under the same circumstances, SCS bridges proactively inform the entire network of possible forwarding path changes, by advertising their host MAC addresses.

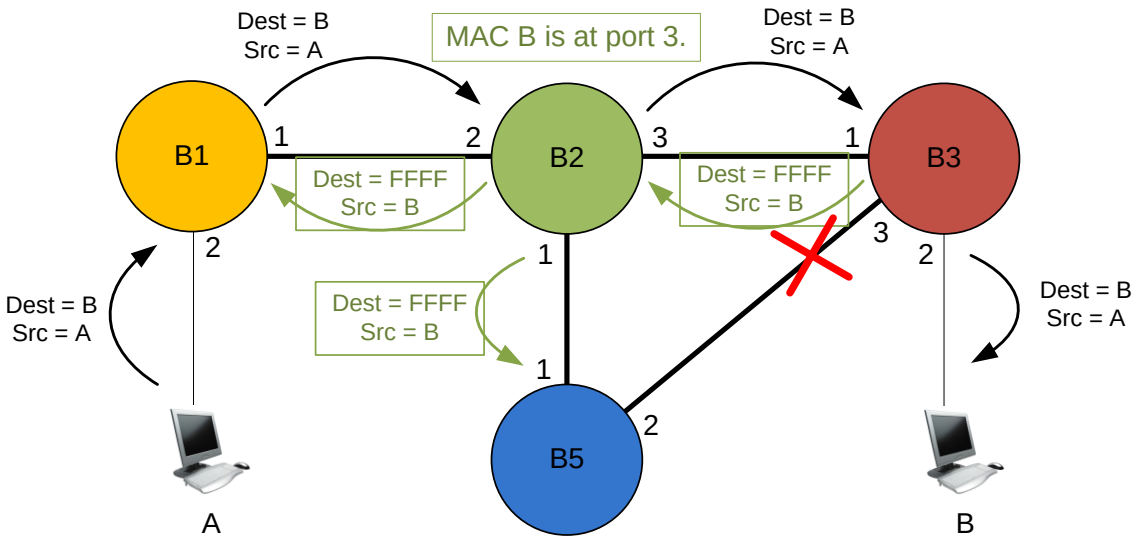


Figure 7-9 - SCS inverted unknown unicast flooding

Like Figure 7-9 pictures, as soon as bridge B3 detects neighbour relationship with B5 is broken, it impersonates its hosts and proactively informs the remaining neighbours about the hosts connected to it, sending dummy broadcasts frames spoofing its host MAC addresses. Therefore, all neighbour bridges will quickly update their FdwT tables (forwarding tables) accordingly, network flooding is reduced significantly and overall network convergence is incredibly much faster.

The process will be similar for bridge B5, the other neighborship peer.

As SCS bridges is concerned, those dummy broadcasts will be considered normal Ethernet broadcast packets and, as such, will be encapsulated as SCS Unicast Flood packets and sent over the SCS network, accordingly.

The processing of SCS UF packets performed by bridges will be analysed in chapter 8.4 - Flooding Process (data/control plane).

These are the main SCS protocol features and characteristics. Next chapter will explain in detail the complex logic behind SCS protocol, dissecting its four internal processes, which control both data and control plane of SCS bridges.

8 SCS Processes

SCS Protocol has four distinct internal processes, to control both data and control plane:

- Forwarding process (data plane)
- Neighborship process (control plane)
- Topology process (control plane)
- Flooding process (data + control plane)

The flooding process has both data and control components.

The data plane component of the Flooding process, defines the way a SCS bridge forwards traffic that need to be flooded throughout the network. The control plane component defines the flooding control mechanism to avoid unnecessary data replication and looped frames within the network.

Next, all four processes are exposed in detail.

8.1 Forwarding Process (data plane)

The Forwarding Process defines the way how frames are forwarded in a SCS network. As said previously, SCS does not change the overall transparent bridging characteristic of a regular bridge. So, the bridge forwards traffic as before.

A ***Forwarding Table*** (**FwdT**) is built, similarly to current bridge's CAM Table. It contains the following fields:

- 48-bit MAC Address
- VLAN[3]
- Entry type (static / dynamic)
- Mac Learning (on/off)
- Aging timer – by default on the majority of the platforms, 300 seconds.
- Interface

However, there is a major difference between SCS FwdT Table and today bridge's CAM Table: SCS FwdT Table is optimized to allow several entries to the same MAC Address. This change will allow the SCS Bridge to load balance traffic between parallel links towards the same neighbour.

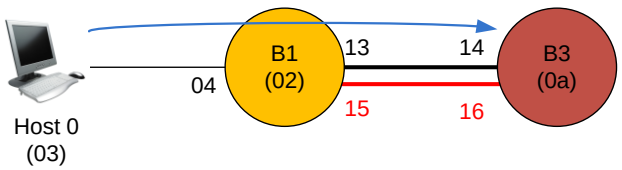


Figure 8-1 - Forwarding Table multiple entries

[3] SCS v1.0 bridges do not take into consideration VLANs for any decision; the topology database built is unique, independently of the VLAN information, likewise the CST of STP. However, it is thought for future SCS versions a domain-like behaviour, where the neighbour key would be used to create the domain boundaries and bridges segregation.

For example, in Figure 8-1, the bridges would have the following FwdT Tables:

Bridge B1 FwdT Table	
MAC Address	Interface
00:00:00:00:00:03	00:00:00:00:00:04

Bridge B3 FwdT Table	
MAC Address	Interface
00:00:00:00:00:03	00:00:00:00:00:14
00:00:00:00:00:03	00:00:00:00:00:16

Table 3 - Bridge B1 and B3 Forwarding Tables

This optimization is achieved by replicating in B3 all FwdT entries relative to interfaces 00:14 and 00:16, as those interfaces connect the same adjacent bridge B1.

Feeding the Forwarding Table

Feeding the FwdT is like feeding today's CAM table, it uses *Source MAC Address learning*, if set to ON in that interface: when an Ethernet frame arrives, the bridge *learns* the Source MAC Address of the frame, via that interface. Nevertheless, after inserting the new record on the FwdT table, if the bridge *knows* there is another SCS bridge over that interface, it will replicate that entry over all other interfaces that lead to that same bridge. This will allow traffic load sharing between parallel links.

Bridge B3 FwdT Table in Table 3 shows that replication: although the learning was performed via interface 00:14, as B3 *sees* Host0 also via 00:16, and thus, it replicates the 00:03 entry over 00:16 also.

Querying the Forwarding Table

Queries to the FwdT are done whenever the bridge needs to forward a frame. If an entry is found, the frame is forwarded via a particular interface, whereas if it is not found, the frame is flooded.

Frame flooding in SCS is quite different from current bridge's behaviour, as it will be analysed in the *Flooding Process*; as the Forwarding Process is concerned, the frames are flooded.

In the presence of parallel links between bridges, replicated entries will exist on bridge's FwdT. In that case, FwdT queries would result in multiple records, which will be processed by a load balancer in order to allow traffic distribution among those several parallel links.

In SCS v1.0, the load balance engine is purely a random choice between the multiple records.

Cleaning the Forwarding Table

There are specific situations where it is imperative to update the FwdT, in order to achieve fast convergence. Updating the FwdT means necessarily the removal of some specific entries or even the entire FwdT.

As described throughout the next SCS Processes analysis, there are specific network events which trigger FwdT cleaning for all entries matching a specific interface or the full table. In those situations, there will be inevitably traffic flood, but that's a well-known trade-off: to achieve faster convergence times, more traffic would have to be flooded in order to quickly update all bridges FwdT tables, avoiding traffic duplication and loops.

8.2 Neighborship Process (control plane)

The Neighborship Process is a control plane mechanism and it is the basis of the SCS Protocol.

Three mechanisms control the entire course of the Neighborship Process:

- **Neighbour Auto-discovery Mechanism** – Discover and establish neighbour relationships between adjacent SCS bridges.
- **Relationship Mechanism** - Maintain neighbour relationship between adjacent SCS bridges.
- **Purging Mechanism** – Neighbour relationship removal.

Those three mechanisms will be described below, but before the neighborship states must be defined.

Neighborship States

Each neighbour relationship between adjacent SCS Bridges has a specific state. SCS Neighborship Process state machine defines three possible states:

- **UP** – Processes and forwards traffic from/to that neighbour.
- **Down** – Does not accept neither forward frames from/to that neighbour.
- **DelayUP** – Transition state between DOWN and UP states. In this state, the bridge waits for the arrival of a specific number of SCS Hello packets (*DelayUpValue*) within a particular time window (*DelayUpTimer*), before transition from DOWN to UP state.

Figure 8-2 presents the state machine transactions for the Neighborship Process.

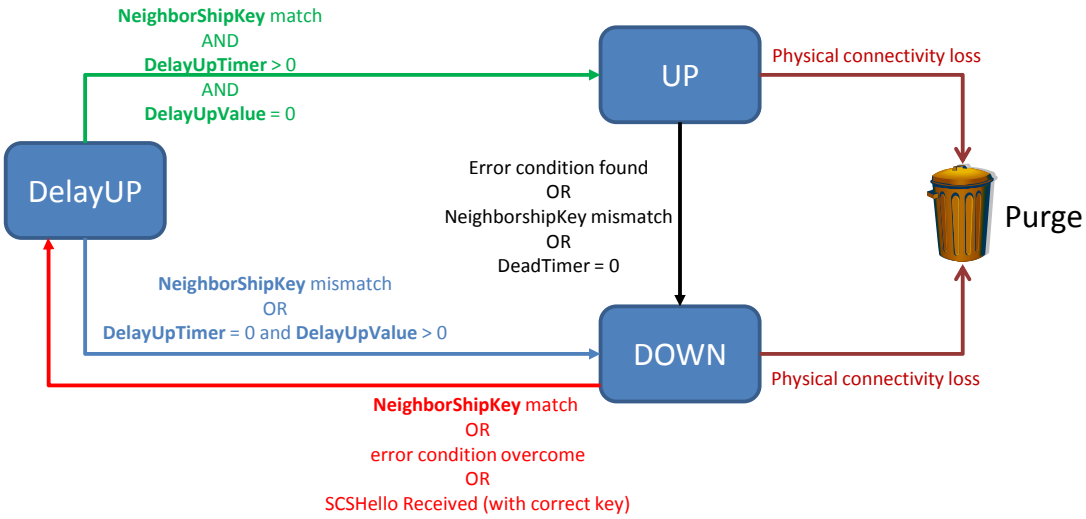


Figure 8-2 - Neighborship process state machine

Neighbour Auto-discovery Mechanism

This mechanism deals with communication between adjacent bridges within the ***DelayUp*** state and it targets neighbour relationship establishment.

Each SCS Bridge will start discovering bridges on the other end of the link, sending ***SCS Hello*** control frames, as soon as a link comes up. This is known as ***SCS neighbour auto-discovery mechanism***.

First of all, the *on the other end of the link* statement is not innocent: by design, SCS does not allow to have more than two bridges on the same Ethernet segment. Besides being rare in modern networks, connecting multiple bridges on the same Ethernet segment does not favours performance neither protocol's simplicity. If a SCS bridge detects SCS Hellos from different neighbours via the same interface, it shuts down the interface.

Figure 8-3 illustrates the *SCS neighbour auto-discovery mechanism*.

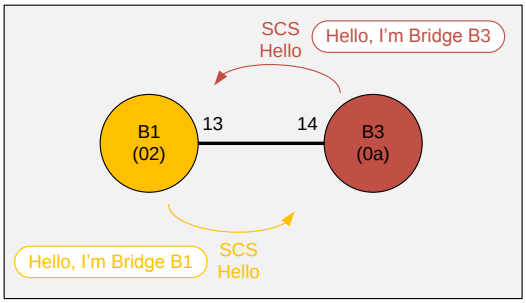


Figure 8-3 - SCS neighbour auto-discovery mechanism

SCS Hello control frames are sent throughout all bridge's interfaces and transported in the payload of an Ethernet frame, as above described in *SCS Frames Format* section of *Self-Configurable Switches Overview* chapter.

Every SCS bridge will accept and process the frame, whereas any other network element will discard the frame as it doesn't recognize the new EtherType 0x0834 neither is listening to the *SCS Ethernet Destination MAC*.

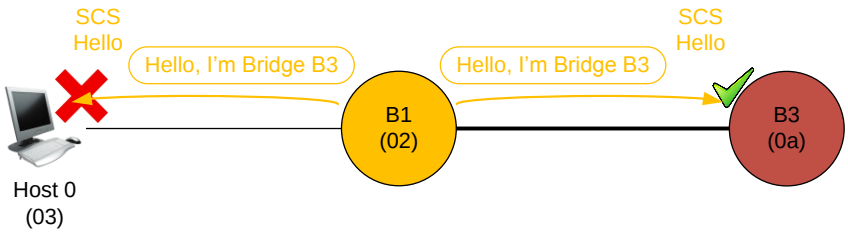


Figure 8-4 - SCS Hello acceptance

SCS Ethernet Destination MAC is a 48-bit MAC address exclusive for SCS bridges, as stated in the previous chapter.

The ***SCS Hello*** message exchanged between neighbour bridges is very simple and contains just a flag, the bridge's MAC Address and a neighborship key, as the next figure illustrates:

Flag	Neighborship Key	SCSID
01	6-bit	48-bit MAC Address

Figure 8-5 - SCS Hello message format

The ***Flag*** is a 2-bit value which identifies the *type* of SCS messages. In this particular case, "***01***" classifies as SCS Hello Messages. Other values could be "10", for SCS Updates, and "11" for UF Packets. Those two last packet types will be described later on.

Neighborship Key, as stated in the previous chapter, is a 6-bit key shared among all SCS bridges of the network, avoiding unauthorized SCS bridges insertion in the production network.

SCSID means SCS Bridge ID and is actually the 48-bit bridge's MAC Address. This ID must be unique and, for SCS, it should be the lowest MAC Address the bridge has, at SCS Protocol startup. As this MAC Address is just used for identification purposes, SCS sticks to it and only rebooting the bridge allows changing this value.

SCS Hello messages exchange, allows bridges to establish neighbour relationship between them. SCS Hellos are sent every ***HelloTimer*** interval, which is one second by default. Upon receiving the hellos, each SCS bridge will feed the ***NB Table*** (Neighbour Table), accordingly.

Feeding the NB Table

The Neighbour Table (***NB Table***) is maintained to keep track of all neighbour bridges.

Table 4 describes NB Table record headers.

NB Table						
SCSID	Interface	State	DelayUp Value	DelayUp Timer	Dead Timer	Hello Timer

Table 4 - NB Table

Each NB Table record has three timers:

- ***HelloTimer*** – By default is 1 seconds. It's the *SCS Hello* messages transmission interval.
- ***DeadTimer*** – It's a counting down timer, equal to 3x*HelloTimer*. Every time a bridge receives a SCS Hello, this timer is reset to its default. This timer expiration represents a neighborship state transition. By default, it's 3 seconds.
- ***DelayUpTimer*** – This timer, associated with ***DelayUpValue*****,** allows protection against faulty communication channels. It protects SCS from link flaps that wouldn't allow *DeadTimer* expiration and would introduce network errors.

Besides those timers, the NB Table includes the ***SCSID*** from the neighbour bridge and the ***Interface*** from which it received the neighbour's SCS Hello.

DelayUp Feature

The ***DelayUpValue*** empowers SCS with a delay buffer, before allowing traffic forwarding through a new link. Please note that this mechanism is completely different from STP's listening/learning states, as STP runs those states for every port. Conversely, SCS runs DelayUp just over *bridge* interfaces, as only SCS Hello messages reception from a neighbour triggers NB Table updates.

The importance of this delay buffer is presented in the next figure.

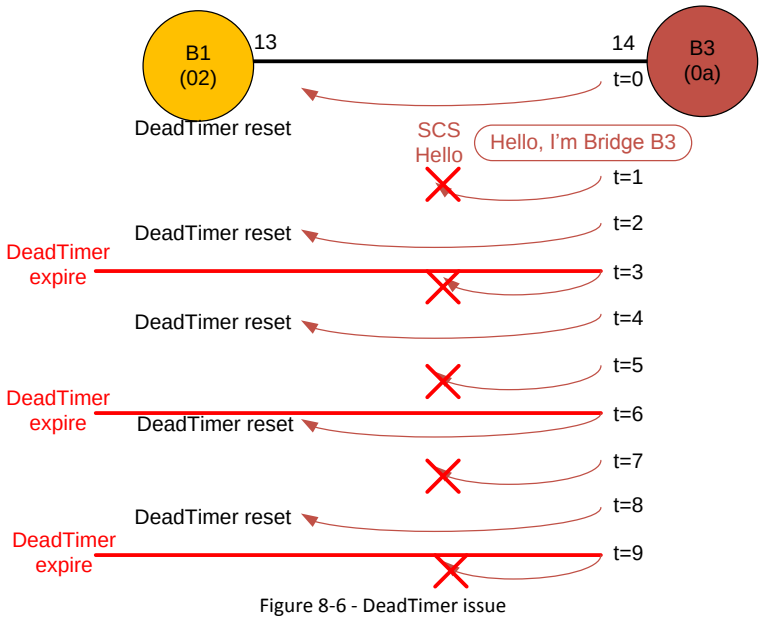


Figure 8-6 - DeadTimer issue

As Figure 8-6 presents, if there were a faulty link with a cadence of one link failure each two seconds, for example, the *DeadTimer* would never expire and it would never cause a NB state transition. This could impact network connectivity.

The ***DelayUp feature*** allows fine tuning the inter-bridges communication. In a certain time interval, it is only allowed to miss one SCS Hello, and this interval can be administratively defined, as some links could be considered more trustful, compared to others. Figure 8-7 depicts ***DelayUpTimer*** formula**.**

$$DelayUpTimer = (DelayUpValue + 1) \times HelloTimer$$

Figure 8-7 - DelayUpTimer Formula

By default, *DelayUpValue* is equal to three (SCS Hellos).

As we can see from the yellow box on Figure 8-8, the previous scenario issue is overcome by the use of this timer. Each four seconds sliding interval, SCS will verify if at least three SCS Hellos have arrived. If not, it triggers a NB state transition, as necessary to put that faulty link out-of-service.

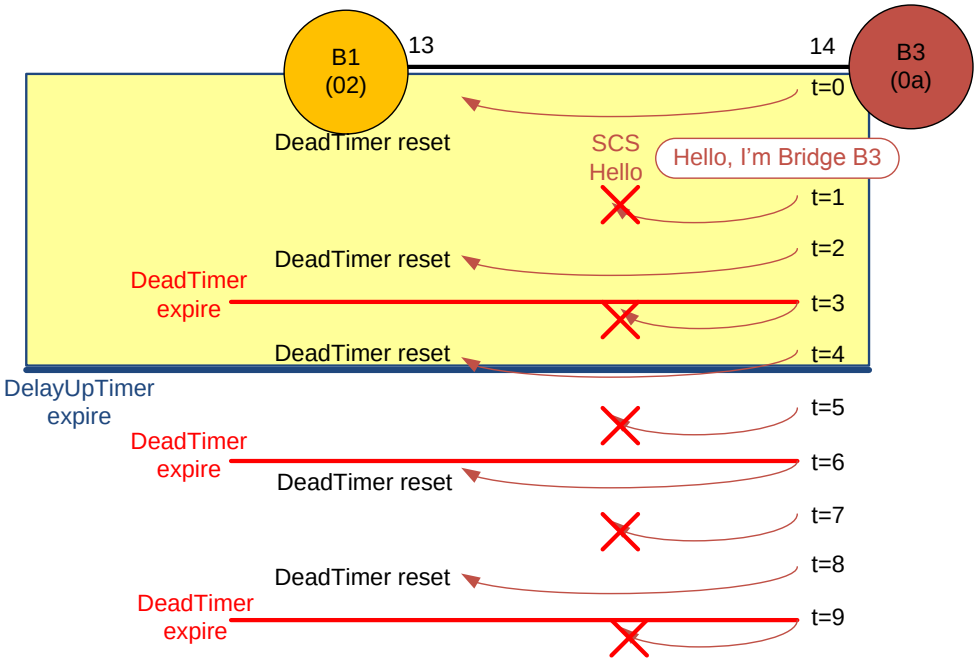


Figure 8-8 - DelayUptimer Expiration

Please note that each timer has its own purpose: *DeadTimer* aims links unavailability, whereas *DelayUpTimer* targets faulty links.

Traffic Load Sharing

SCS allows traffic load sharing over multiple connections between two adjacent bridges.

If SCS detects the same neighbour over different interfaces, it starts the replication of FwdT entries between those interfaces and the round-robin engine to equal balance flood traffic over those parallel links.

Neighbour Auto-discovery Mechanism Example

The following example demonstrates the *Neighbour Auto-discovery Mechanism* functionality.

Suppose the scenario illustrated on Figure 8-9. Link B1-B2 is “shut down”.

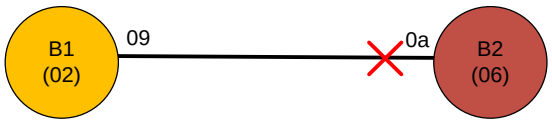


Figure 8-9 - NB auto-discovery mechanim example

When link B1-B2 is turned on, both B1 and B2 will send SCS Hellos throughout that new link.

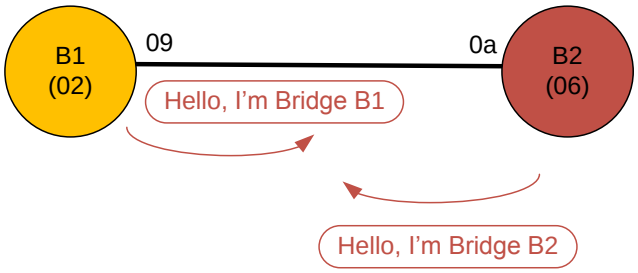
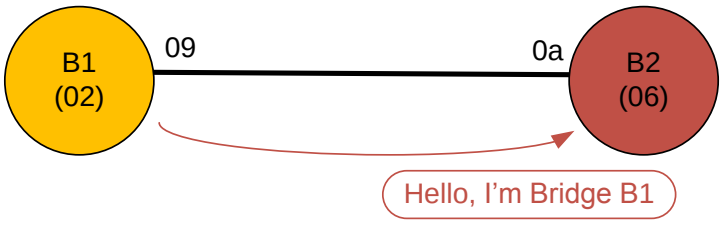


Figure 8-10 - Bridges exchanging SCS Hellos

To keep the example simple, as SCS exchange and processing is similar in both bridges, it will be analysed only B2 side. Also, the *HelloTimer* column of the NB Table is not referenced in the pictures, as it is not relevant to analyse in this context.

As soon as B2 receives B1’s SCS Hellos, it creates a new NB Table record, in delayUP state:



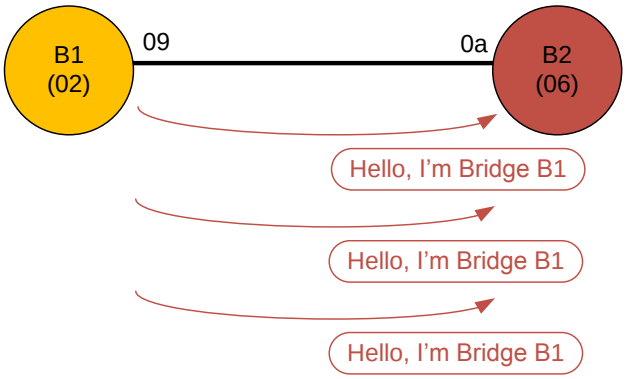
NB Table

SCSID	Interface	State	DelayUpValue	DelayUpTimer	DeadTimer
0000.0000.0002	0000.0000.000a	delayUP	3	4	3

Figure 8-11 - delayUP state

In fact, the NB table record start at DOWN state, but DOWN state is immediately transitioned towards delayUp state. In this state, DelayUpTimer and DeadTimer timers start counting down, and also the DelayUpValue counter.

After receiving the next B1’s SCS Hello, B2 will update the NB table accordingly, i.e., decrease the DelayUpValue counter and reset DeadTimer. If the link is working correctly, after 3 seconds, the DelayUpValue would reach 0 and DelayUpTimer has not reach 0 yet. So, the link is considered trustworthy and UP state is reached:



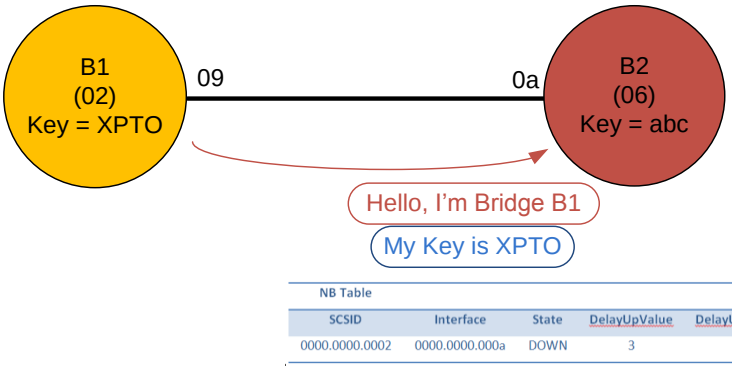
NB Table

SCSID	Interface	State	DelayUpValue	DelayUpTimer	DeadTimer
0000.0000.0002	0000.0000.000a	UP	0	1	3

Figure 8-12 - UP state

Transitions towards UP state will trigger the Topology Process mechanisms, described in next sections.

In this simple scenario, several issues might arise that could lead the neighborship to a different state than UP. In the previous *DelayUp Feature* section one such example was described. Another issue that could arise is *Neighborship keys* mismatch. In such circumstances, the interface would be logically shutdown by SCS, i.e., the NB state would be considered DOWN. Figure 8-13 illustrates a SCS Hello containing the *XPTO* key arriving to bridge B2, whereas B2 was configured with *abc* key. Consequently, B2 would discard ALL traffic received by that interface.



NB Table

SCSID	Interface	State	DelayUpValue	DelayUpTimer	DeadTimer
0000.0000.0002	0000.0000.000a	DOWN	3	4	3

Figure 8-13 - Neighborship key mismatch

After enabling the establishment of neighbour relationships, the Neighborship Process is also responsible for maintaining the neighborship. For such control functions, the ***Relationship Mechanism*** exists.

Relationship Mechanism

After the completion of the *Neighbor Auto-discovery Mechanism*, NB Table records are in one of two possible states: UP or DOWN; delayUP state is just a transient state between DOWN and UP.

Both DOWN and UP states are considered an adjacency and need to be maintained. Therefore, SCS uses the *Relationship Mechanism* to keep up DOWN and UP states. This mechanism guarantees the maintenance of those neighbour relationships via the periodic transmission of *SCS Hello* messages (or lack of it).

Relationship Mechanism: DOWN state

The state is maintained DOWN, in the following conditions:

1. If the *SCS Hello's Neighborship Key* remains different between adjacent bridges.
2. Some error condition persists.
3. *DeadTimer* expired and no *SCS Hello* is received.

So, you may be wondering why it is important to keep track of such a DOWN adjacency… The main reason is loop prevention. If any of the above conditions occur, something is not correct between those two adjacent bridges. So, it is preferable to keep neighborship down and stop forwarding traffic via that particular link, than injecting errors or bad decisions on the SCS network that could lead to network meltdown. Consequently, for the first condition, it's an administration error or network resources abuse; it is a good practice to logically shut down that link and feedback the network administrator with some log error message indicating Neighborship Key mismatch. The second condition speaks for itself. This is a generic bucket for all SCS error conditions that may be detected in the future; currently, none SCS event maps to this condition. Regarding the third condition, *DeadTimer* expiration, this is clearly a problem with a particular SCS neighbour relationship. This relates with an active NB table record, which lacked the SCS Hello reception. Several physical events map into this condition, like unidirectional links and faulty cables, for example. In order to prevent potential loops or erroneous network decisions, no traffic should flow through that bridge.

Relationship Mechanism: Leaving DOWN state

Once in DOWN state, the following events must occur in order to a NB record leave that state:

- If the switch receives a *SCS Hello* with the correct *Neighborship Key*, the *DeadTimer* would be reset to its default and the NB state will transit to *delayUP*. This would overcome previous condition number one and three.
- Naturally, if the error condition is overcome, whatever it may be, the second condition no longer exists and the NB state will transit to *delayUP*.

Administratively removing the neighborship, that might be accomplished by shutting down and bringing up again the interface, cannot be considered a valid process for a particular NB entry state to leave the DOWN state, as after shutting down the interface, the NB table entry would be immediately removed; so, no entry would exist when the neighborship process is starting over.

When leaving DOWN state, the Neighborship Process must inform the other SCS Processes to refresh their information accordingly and take the appropriate actions to reflect such change.

Relationship Mechanism: UP state

Every time a *SCS Hello* is received, the *DeadTimer* is reset to 3x*HelloTimer*.

The state is maintained UP, in the following conditions:

- If a bridge receives periodic *SCS Hellos* with correct *Neighborship Key*.
- *DeadTimer* does not timeout.

SCS bridges transit from UP to DOWN states if some of the following conditions occur:

- If receives *SCS Hellos* containing a *Neighborship Key* mismatching its own.
- If stops receiving *SCS Hellos* and *DeadTimer* expires.
- Some error condition is found.

Basically, the above are just the conditions necessary to maintain the state as DOWN, like seen before.

When leaving the UP state, the Neighborship Process must flag the other SCS Processes to behave accordingly to the new information provided.

Purging Mechanism

In certain conditions, a particular neighbour relationship must be purged. Those conditions are intrinsically associated physical connectivity losses.

The purging mechanism, besides starting the hosts MAC address spoofing by the bridge (inverted flooding feature, as explained in the SCS Overview chapter), it also triggers the Neighborship Process communication with the other SCS processes, in order to the latter reflect the changes introduced by the Purging Mechanism.

8.3 Topology Process (control plane)

The Topology Process is another fundamental control plane engine. Basically, it provides a SCS bridge with the knowledge of all other SCS bridges running on campus, besides its own adjacent neighbours, and the *cost* of reaching all them. The core function of the Topology Process relies on the ***Topology Change Mechanism***.

Topology Change Mechanism uses an internal table, named ***Topology Table (TP Table)***. Table 5 shows the simple structure of the TP Table.

TP Table		
Neighbour SCSID	**Interface**	**Metric**

Table 5 - Topology Table structure

TP Table fields are self-explanatory: ***Neighbour SCSID*** is the MAC Address of the remote bridge, ***Interface*** is the inbound interface from which the bridge receives *Topology Updates* and ***Metric*** is the path cost.

TP table will be fed with all paths towards all other bridges. However, SCS Topology Process optimizes TP Table entries by allowing just the best paths to be inserted into the TP table. For example, in the network topology of Figure 8-14, bridge B1 would have the TP table presented on Table 6, although there would also be another path to B2 (00:00:00:00:00:06) with metric 3: B1-B3-B4-B2.

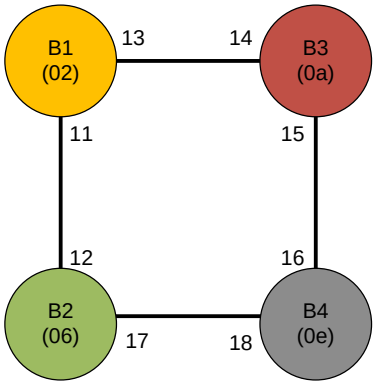


Figure 8-14 - Square topology example

B1 TP Table		
Neighbour SCSID	**Interface**	**Metric**
00:00:00:00:00:06	00:00:00:00:00:11	1
00:00:00:00:00:0a	00:00:00:00:00:13	1
00:00:00:00:00:0e	00:00:00:00:00:13	2
00:00:00:00:00:0e	00:00:00:00:00:11	2

Table 6 – B1's TP table for the square topology example

The ***metric*** attribute for SCS is interpreted as the *path cost* towards a bridge. The formula in Figure 8-15 presents the *metric* calculation for a specific link connecting two bridges.

$$Metric = \frac{BW_{ref}}{Link\ Speed_{bps}}$$

Figure 8-15 - Metric formula

The BW_{ref} parameter is considered equal to 40Gbps, i.e., 40×10^9 bps. $Link\ Speed_{bps}$ is the link bandwidth in Mbps, i.e., 10^6 bps.

Table 7 presents the metric values for the most common Ethernet link speeds.

Link Speed	*Metric* value
10 Mbps	4000
100 Mbps	400
1 Gbps	40
10 Gbps	4
40 Gbps	1

Table 7 - Metric values for common link speeds

To simplify the protocol study, all links in the examples and ns3 simulations are considered to be 40 Gbps. This way, the ***metric*** throughout this document will be just the *hop count* between bridges.

Updating the TP Table: SCS Updates

Topology tables are updated via dedicated messages exchanged between neighbour bridges: the ***SCS Update*** packets.

SCS Update control frames are triggered whenever a bridge needs to advertise a change towards all other bridges in the network. For example, after establishing a new neighbour relationship with a directly connected bridge, all other adjacent neighbours are signalled about the change. Next, the adjacent neighbour bridges will create their own *SCS Updates* and propagate them towards their local neighbours. That way, all bridges get knowledge about the new neighborship and calculate the cost towards it.

Figure 8-16 illustrates this example.

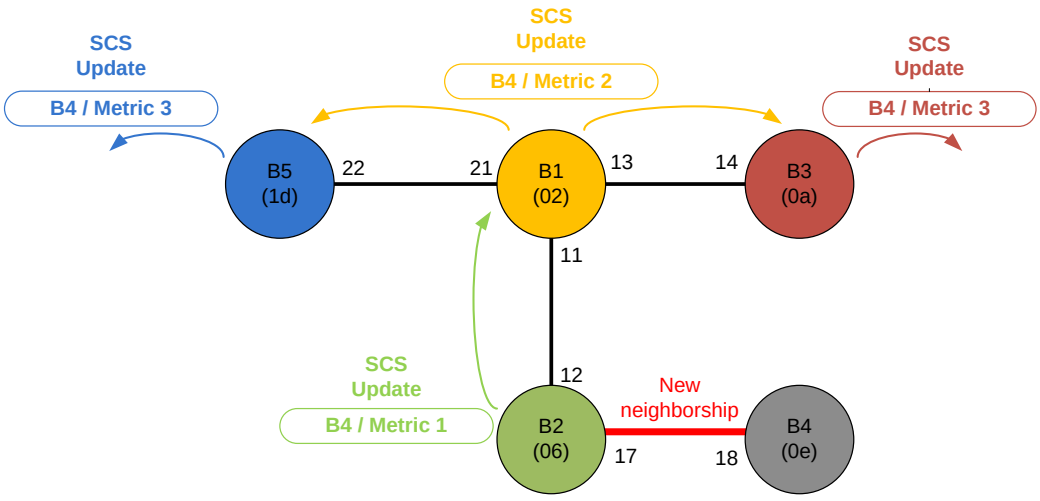


Figure 8-16 - SCS Update for a new neighborship

Like SCS Hello messages, SCS Updates are transported in the payload of Ethernet frames, following the same scheme as the one described in Figure 7-7 of *SCS Frames Format* section.

Upon reception of a SCS Update packet, the bridge does not forward it, neither relay it; the bridge processes the information received and, if necessary, answers back or inform its neighbours using its own SCS Update packets.

SCS Update frame format is illustrated in Figure 8-17.

Flag	Neighborship Key	Neighbour ID	Origin ID	Metric	clearFlag
10	6-bit	48-bit MAC Address	48-bit MAC Address	16-bit	8-bit

Figure 8-17 - SCS Update message format

The ***Flag*** is a 2-bit value which identifies the type of SCS messages, in this particular case, "***10***" specifies *SCS Update Messages*.

Neighborship Key, as discussed before, is the 6-bit shared secret key that avoids unauthorized SCS bridges insertion in the production network.

Neighbour ID means *SCS Neighbour Bridge ID* and is actually the 48-bit MAC Address of the bridge which the SCS Update concerns. This Neighbour ID can represent both adjacent and remote bridges.

Origin ID is the 48-bit MAC Address of the bridge that initially originated the SCS Update. This field signals the network about the origin of the change.

Metric is the path cost towards the Neighbour ID, as stated before.

clearFlag is a control flag used by the Topology Change Mechanism to signal neighbour bridges about the action required to be taken upon the reception of a *SCS Update*. This flag can have the following values and meanings:

- **0** – install topology record
- **1** – clear topology record
- **2** – query for topology record
- **3** – install delegation record
- **E** – remove delegation record

The bottom two, '**3**' and '**E**', will be described in the ***control plane component*** of the ***Flooding Process***.

SCS Updates: clearFlag 0

The Topology Change Mechanism defines the following algorithm to process the information received over a SCS Update with clearFlag equalling zero, i.e., when the SCS Update is signalling the bridge to update its TP Table regarding *Neighbour ID* bridge:

1. Increase metric accordingly.
2. Drop SCS Update if Neighbour ID or Origin ID is equal to its own SCSID.
3. The bridge queries its own TP Table and
    a. Install a new entry if no information about Neighbour ID is present.
    b. Update an existent entry if the receiving metric is better (lower cost).

   c. Install a new entry if the receiving metric is equal to an existent entry, but it is sourced from a different neighbour bridge (redundant equal cost path) or arrives from a different interface (redundant parallel path).
   d. Drop the SCS Update if the receiving metric is superior.
4. In case of 3a, 3b or 3c, the bridge must do all the following:
   a. Inform all other neighbours about its new TP table record, creating SCS Update packets, maintaining the Origin ID.
   b. Flush its FwdT Table.
   c. Update its Flood Table (FT). FT table is described in the *Flooding Process*.
   d. Start inverted flooding process.

SCS Updates: clearFlag 1

If clearFlag equals one, the bridge is being signalled to clear a TP table entry. In such a case, the bridge queries its own TP table and:

1. Drop SCS Update if Neighbour ID or Origin ID is equal to its own SCSID.
2. If a Neighbour ID entry exists via that interface with such a metric, the bridge removes the TP table entry and:
   a. Signals its neighbours about this removal, triggering SCS Updates to them.
   b. As the bridge has lost a path to a bridge, the bridge will try to discover an alternate path towards it. Consequently, if the bridge does not have an alternative path in the TP table, it will ask all its neighbours for a new path. This request is signalled via a SCS Update packet with clearFlag equal to 2.
3. If a Neighbour ID entry exists but via other interface and the metric is equal or better than the one received, the bridge drops the SCS Update but sends back a new SCS Update with clearFlag set to 0 to help the other bridge to get a quicker alternate path towards the bridge it has lost.

This case flushes for all the FwdT entries relative to that particular interface and the FT is queried to see if it needs to be updated.

SCS Updates: clearFlag 2

After removing a TP table entry, if the bridge lacks a path towards a bridge, it will ask all its neighbours for a new path. This request is signalled via a SCS Update packet with clearFlag equal to 2.

Upon receiving the SCS Update packet with flag set to 2, the neighbour bridges acknowledge that they are being queried for a specific path towards another bridge in the campus. Each neighbour bridge will then behave accordingly:

- If it does not have a path for that destination, silently drops the SCS Update packet.
- If it does have a path for that destination but that path is via the interface from which it received the SCS Update, it removes its TP Table entry and triggers SCS Updates with clearFlag set to 1 to all its neighbours.

- If the bridge has a path via another interface, it answers back with SCS Update packet with clearFlag set to 0.

Topology Change Mechanism Example

Consider the following topology:

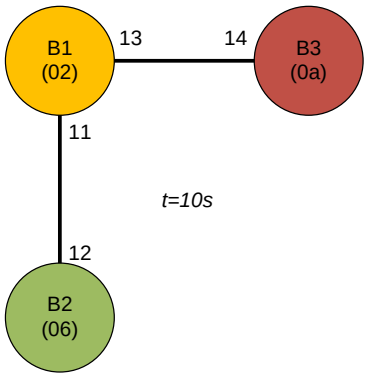


Figure 8-18 - Topology change mechanism example

At time instant *t=10s*, the network has bridges B1, B2 and B3 connected like Figure 8-18 illustrates. Consequently, their TP tables are:

B1 TP Table		
Neighbour SCSID	Interface	Metric
00:06	00:11	1
00:0a	00:13	1

B2 TP Table		
Neighbour SCSID	Interface	Metric
00:02	00:12	1
00:0a	00:12	2

B3 TP Table		
Neighbour SCSID	Interface	Metric
00:02	00:14	1
00:06	00:14	2

Table 8 - Topology change mechanism example: TP Tables

Suppose a new bridge B4 is connected to bridge B2, like Figure 8-19 pictures, at *t=50s*.

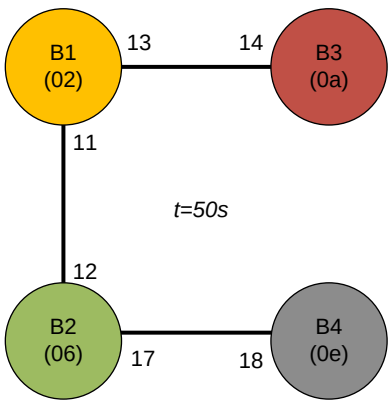


Figure 8-19 - Topology change mechanism example: B4 insertion

After B2-B4 neighborship is established, the following ns-3 SCS protocol simulation outputs occur, regarding the Topology Change process, at B2 and B4 bridges (B1 and B3 output is omitted):

<table>
<tr><th>Event</th><th>NS3 Simulation Output</th></tr>
<tr><td>B2 starts the Topology Process.</td><td>t=53.0021: Bridge B2 changed NB Table State to UP for neighbour B4 and started the TOPOLOGY PROCESS for this neighbour via interface 00:00:00:00:00:17</td></tr>
<tr><td>B2 updates its TP table.</td><td>B2 TP Table
<table><tr><th>Neighbor SCSID</th><th>Interface</th><th>Metric</th></tr><tr><td>00:02</td><td>00:12</td><td>1</td></tr><tr><td>00:0a</td><td>00:12</td><td>2</td></tr><tr><td>00:0e</td><td>00:17</td><td>1</td></tr></table></td></tr>
<tr><td>B2 informs all adjacent neighbours about B4 and B4 of all B2 TP entries.</td><td>t=53.0021: Bridge B2 sent an SCS TP Update Packet to destination B1 about entry related to B4 with clear flag=0
t=53.0021: Bridge B2 sent an SCS TP Update Packet to destination B4 about entry related to B1 with clear flag=0
t=53.0021: Bridge B2 sent an SCS TP Update Packet to destination B4 about entry related to B3 with clear flag=0</td></tr>
<tr><td>B4 starts the Topology Process.</td><td>t=53.0044: Bridge B4 changed NB Table State to UP for neighbour B2 and started the TOPOLOGY PROCESS for this neighbour via interface 00:00:00:00:00:18</td></tr>
<tr><td>B4 updates its TP table.</td><td>B4 TP Table
<table><tr><th>Neighbor SCSID</th><th>Interface</th><th>Metric</th></tr><tr><td>00:06</td><td>00:18</td><td>1</td></tr></table></td></tr>
<tr><td>B4 receives SCS Update regarding B1 and updates the TP table.</td><td>t=53.0087: Bridge B4 is processing the SCS Update received from B2 relative to B1 with Clear Flag = 0 and metric=2.
t=53.0087: Bridge B4 received a SCS Update packet via interface 00:00:00:00:00:18 for a new entry to Bridge B1 and created a TP entry.

B4 TP Table
<table><tr><th>Neighbor SCSID</th><th>Interface</th><th>Metric</th></tr><tr><td>00:06</td><td>00:18</td><td>1</td></tr><tr><td>00:02</td><td>00:18</td><td>2</td></tr></table>
t=53.0087: Bridge B4 flushed FwdT.</td></tr>
<tr><td>B4 receives SCS Update regarding B3 and updates the TP table.</td><td>t=53.0176: Bridge B4 is processing the SCS Update received from B2 Relative to B3 with Clear Flag = 0 and metric=3.
t=53.0176: Bridge B4 received a SCS Update packet via interface 00:00:00:00:00:18 for a new entry to Bridge B3 and created a TP entry.

B4 TP Table
<table><tr><th>Neighbor SCSID</th><th>Interface</th><th>Metric</th></tr><tr><td>00:06</td><td>00:18</td><td>1</td></tr><tr><td>00:02</td><td>00:18</td><td>2</td></tr><tr><td>00:0a</td><td>00:18</td><td>3</td></tr></table>
t=53.0176: Bridge B4 flushed FwdT.</td></tr>
</table>

Table 9 – Topology change example: ns3 simulation events and outputs

Table 10 shows the TP tables for all bridges after the topology convergence.

B1 TP Table		
Neighbor SCSID	**Interface**	**Metric**
00:00:00:00:00:06	00:00:00:00:00:11	1
00:00:00:00:00:0a	00:00:00:00:00:13	1
00:00:00:00:00:0e	00:00:00:00:00:11	2

B2 TP Table		
Neighbor SCSID	**Interface**	**Metric**
00:00:00:00:00:02	00:00:00:00:00:12	1
00:00:00:00:00:0a	00:00:00:00:00:12	2
00:00:00:00:00:0e	00:00:00:00:00:17	1

B3 TP Table		
Neighbor SCSID	**Interface**	**Metric**
00:00:00:00:00:02	00:00:00:00:00:14	1
00:00:00:00:00:06	00:00:00:00:00:14	2
00:00:00:00:00:0e	00:00:00:00:00:14	3

B4 TP Table		
Neighbor SCSID	**Interface**	**Metric**
00:00:00:00:00:02	00:00:00:00:00:18	2
00:00:00:00:00:06	00:00:00:00:00:18	1
00:00:00:00:00:0a	00:00:00:00:00:18	3

Table 10 - Topology change mechanism example: TP Tables after B4 insertion

8.4 Flooding Process (data/control plane)

The ***Flooding Process***, *data plane component*, states how *flooded* frames must be forwarded. The *control plane component*, defines how bridges process *flooded* frames, both local to the bridge and received from neighbour bridges.

Data plane component

The data plane component of the Flooding Process, defines how to forward frames with unknown destination address, specifically broadcast, multicast and unicast frames for which the bridge doesn't have an entry on FwdT and doesn't know where to forward it. The standard Transparent Bridging concept states that those frames should be flooded throughout all ports, except the one that received the frame.

SCS Protocol differentiates the flooding method by the *type* of the port. Therefore, traffic *flooding* is different for *host* and *bridge* interfaces.

Upon receiving a *flooded frame*:

- If the incoming interface is a ***host*** port, the bridge will:
  - Forward the frame throughout all *host* ports, except the incoming one.
  - For all *bridge* interfaces, which connect adjacent SCS neighbours, the bridge must create ***SCS Unicast Flood packets*** (SCS UF) containing the unknown destination traffic and unicast them via the *bridge* port.
- If the incoming interface is a ***bridge*** port, the bridge is actually receiving a SCS UF packet. This particular scenario will be described in *Loop avoidance and forwarding mechanisms* section. However, generically, the bridge will:
  - Forward the Ethernet frame encapsulated in the SCS UF packet throughout all *host* ports.
  - If it is delegate for such traffic, create ***SCS Unicast Flood packets*** containing the unknown destination traffic and unicast them via the *bridge* ports for which it is delegate.

In case of equal load-balancing links between bridges, the flooded frame is only sent by one of them, in order to avoid packet duplication. By design, SCS uses a round-robin engine that chooses one of the parallel redundant paths in the topology table to forward the frame.

SCS Unicast Flood packets and *Delegation* are very important concepts which are covered next on the analysis of the control plane *Flood Control Process* section.

Control plane component

The control plane component defines all the procedures that allow bridges to know how to process received *flooded* traffic, both local via host ports and remote via bridge interfaces.

This Flooding Process component is extremely important as it defines the way SCS protocol breaks loops. Literally, it controls the frame flooding process, in order to avoid network loops.

The Spanning Tree protocol eliminates loops from the network by logically disabling all redundant paths between bridges. This not only limits the flooded traffic forwarding, but also the known unicast possible paths! As some of the main advantages of SCS are the use of all available network resources and traffic load balancing allowance, it does not break loops disabling redundant paths; SCS intelligently manages all the links and chooses the best paths from one bridge to any other bridge in the campus. To accomplish this, SCS relies on a mechanism called ***SCS Delegation***, to deal with traffic that needs to be flooded.

SCS Delegation Mechanism

When a SCS bridge updates its TP table with a path for a non-adjacent bridge, it must calculate the better way to reach all probable hosts behind that bridge, via an unique optimized path. So, the SCS bridge will do two things:

1. Update its Flood Table (FT).
2. Instruct an adjacent neighbour to be its delegate to reach a particular bridge. That neighbour updates its Delegation Table (DT).

SCS Delegation Mechanism uses two tables, FT and DT. An interesting singularity of SCS Delegation is that it plays with two tables, but one is installed on a remote bridge.

Table 11 shows the simple structure of the Flood Table (FT).

FT Table		
Remote Bridge SCSID	**outInterface**	**Delegate SCSID**

Table 11 – Flood Table format

Table 12 shows the Delegation Table (DT) format.

DT Table		
Interface	**Source SCSID**	**Destination SCSID**

Table 12 - Delegation Table Format

To better understand the delegation concept, consider the following basic topology of Figure 8-20.

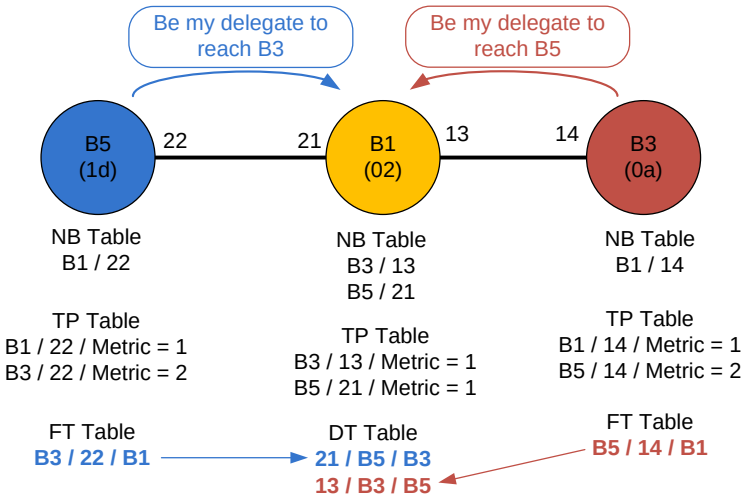


Figure 8-20 - Delegation example

Bridge B5 knows that B3 is not an adjacent neighbour, as it is not present in the NB table. Querying its TP table, acknowledges B3 is reached via interface 22. Correlating NB and TP tables, B5 concludes it reaches B3 via B1. Consequently, B5 asks B1 to be its delegate to reach B3. The same for B3, reaching B5. That way, B1 DT table is updated by adjacent neighbour's requests: B5 to B3 and B3 to B5.

B1 was delegated by B5 to flood B5's flooded traffic into B3. B5 does not worry about how B1 will reach B3; that's now B1 responsibility. However, B5 knows B1 cost to reach B3 is 1. As soon as B1 receives a flood frame from B5, it will forward the frame throughout all host ports and a unicast SCS UF packet towards B3.

Delegations can be progressive, meaning a bridge may be delegate towards some destination for one bridge, but it delegates in some other bridge the responsibility to reach that destination. That's another great contribution of FT. The following example simplifies the understanding of this progression scheme, which seems complex but it is indeed very simplistic:

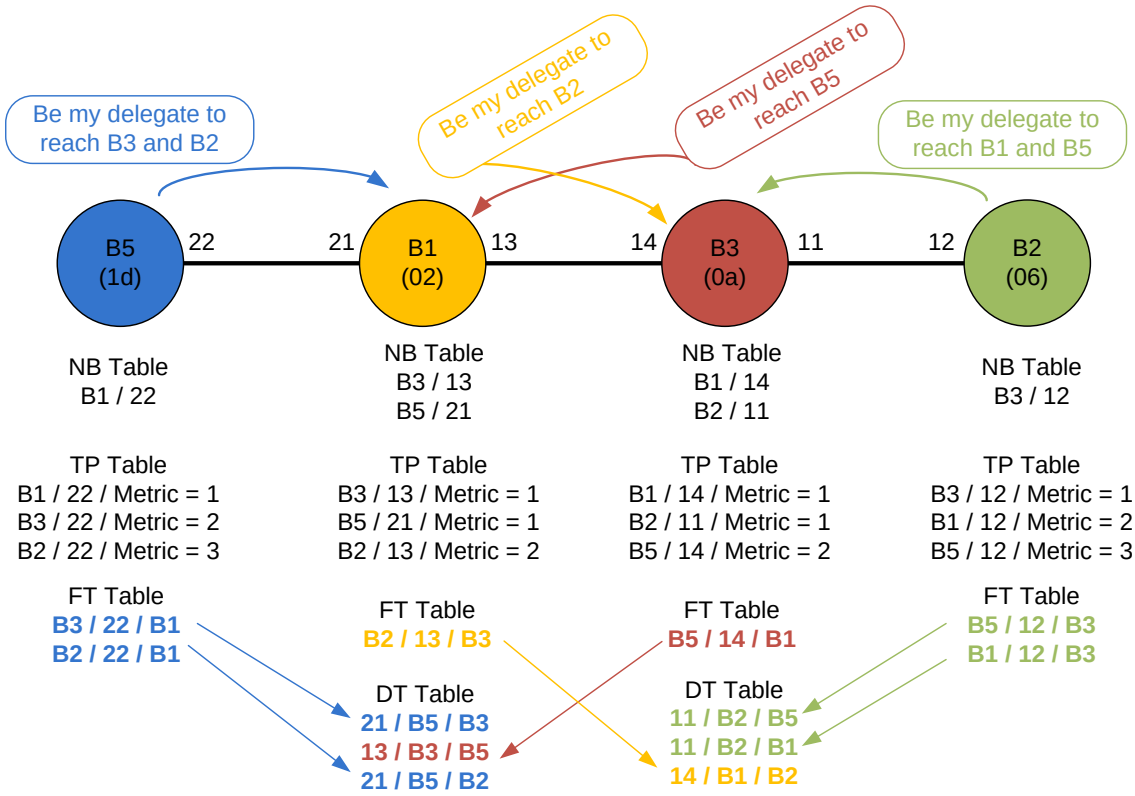


Figure 8-21 - Progressive delegation

As Figure 8-21 exemplifies, if B5 receives a broadcast, it sends to B1, its adjacent neighbour. Next, B1 as delegate for B5 towards B3 and B2, is responsible for propagating that broadcast to B3 and B2. B1 realises B3 is a local neighbour and to reach B2 is also via B3 (FT = B2/13/B3). That way, B1 sends the "*broadcast*" (SCS UF packet) to B3 only. B3, in turn, receiving B1 traffic, queries its DT and acknowledges it is delegate for B1 towards B2. Consequently, B3 sends the "*broadcast*" into B2.

Figure 8-21 shows the progressive delegation of B5 in B1 and B1 in B3.

Delegation requests are signalled via SCS Update messages, using clearFlag '3' to install a delegation and 'E' to remove.

SCS Unicast Flood packets

As said previously in the data plane component, SCS bridges flood frames via all its *host* ports. Regarding *bridge* ports, SCS bridges encapsulate the local received flood frames in ***SCS Unicast Flood packets*** and unicast them towards all adjacent neighbour SCS bridges.

In contrast with SCS Hellos and SCS Updates, SCS UF packets are not messages carried on the payload of new Ethernet frames.

SCS Hellos and SCS Updates are packets created by SCS bridges for protocol control purposes. Conversely, ***SCS Unicast Flood (SCS UF) packets*** are Ethernet frames carrying real traffic that was changed by SCS bridges to control frames that need to be flooded.

SCS UF Headers have the following structure:

Flag	Neighborship Key	Origin SCSID	Protocol	TTL
11	6-bit	48-bit MAC Address	16-bit	8-bit

Table 13 - SCS Unicast Flood header structure

The ***Flag*** field, as before, is a 2-bit value which identifies the type of SCS messages. For SCS Unicast Flood packets, it is set to "***11***".

Neighborship Key, as discussed before, is the 6-bit shared secret key that avoids unauthorized SCS bridges insertion in the production network.

Origin SCSID identifies the 48-bit MAC address of the SCS bridge that originated the *flood* frame. That bridge sets this field with its 48-bit MAC address and all delegate bridges do not change it; this value arrives to the most remote bridge unchanged.

The ***Protocol*** field is very important, in order to rebuild the original Ethernet frame, after removing the SCS UF "*treatment*", as this original info is rewritten by SCS.

The ***TTL*** is just a counter to limit the lifespan of SCS UF packets, in order to prevent the info to be looped indefinitely in the network. The origin SCS bridge sets TTL to a value equal to the greatest metric available at its TP table, defining the *broadcast range* for the information. At the reception of a SCS UF packet, the TTL is decremented accordingly to the cost (metric) towards the neighbour bridge that sent the SCS UF packet (not the origin bridge) and used if that bridge is delegated for that traffic. If TTL reaches a value less or equal zero, the SCS UF packet is discarded.

Figure 8-22 presents an example of a SCS UF packet creation. Note the EtherType rewrite and the SCS UF 10-byte Header insertion.

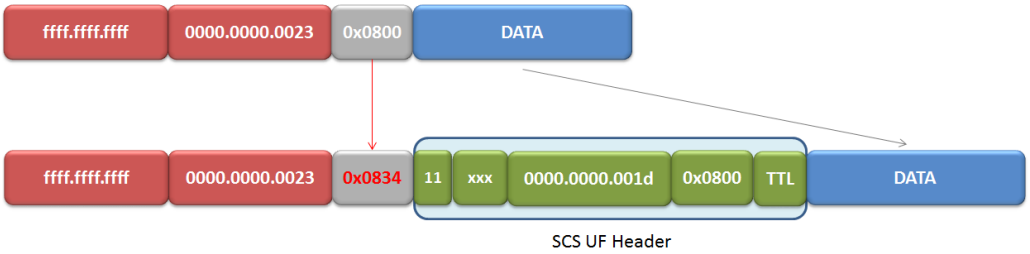


Figure 8-22 - UF packet example

Although the destination address is still a broadcast, the receiving SCS bridge knows that is a SCS UF packet via the EtherType field and the SCS flag set to '11'.

This technique still allows bridges to update their FwdT tables, learning the source MAC addresses from the Ethernet frames. Besides, for unknown destination unicast traffic, this can be a major performance improvement, as flooding may not occur if a bridge already knows the unicast destination location.

Suppose that in Figure 8-23, N1 is sending one broadcast to the network. When bridge B5 receives the broadcast from that host interface, it floods throughout all other host interfaces the same, unchanged broadcast frame. That way, the broadcast

reaches N0 also connected at B5. Next, B5 will create a SCS UF packet containing the original broadcast and forwards it to all its neighbour bridges, in this case, B1.

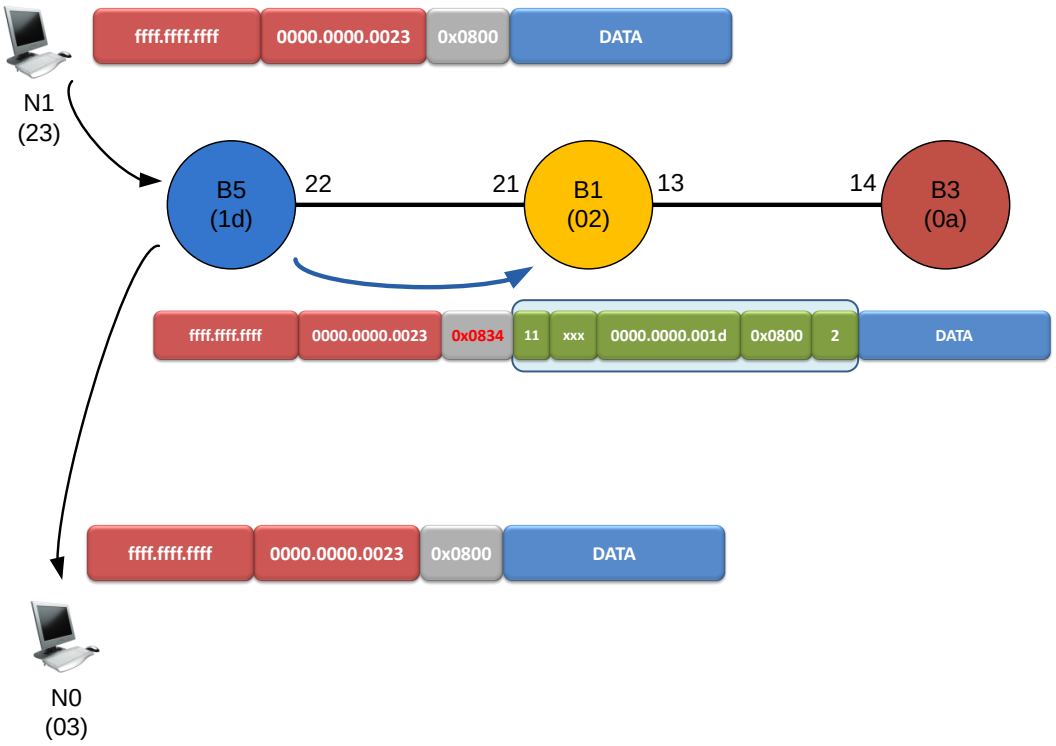


Figure 8-23 - Broadcast flooding example

When the SCS UF packet reaches B1, B1 queries its DT table and verifies that it is delegate for B5's SCS UF packets towards B3. Consequently, B1 decreases TTL by $Metric_{B1-to-B5}$ and replicates the SCS UF packet towards B3. As Figure 8-24 shows, the SCS UF packet is the same, TTL value apart.

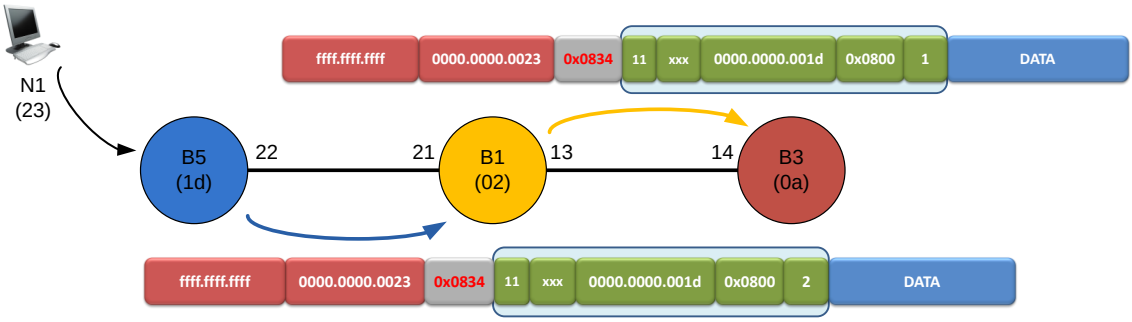


Figure 8-24 - B1 forwards to B3 the received SCS UF packet from B5

Then, B1 reconstruct the original broadcast frame from the SCS UF packet and floods it through all its host interfaces, like Figure 8-25 illustrates.

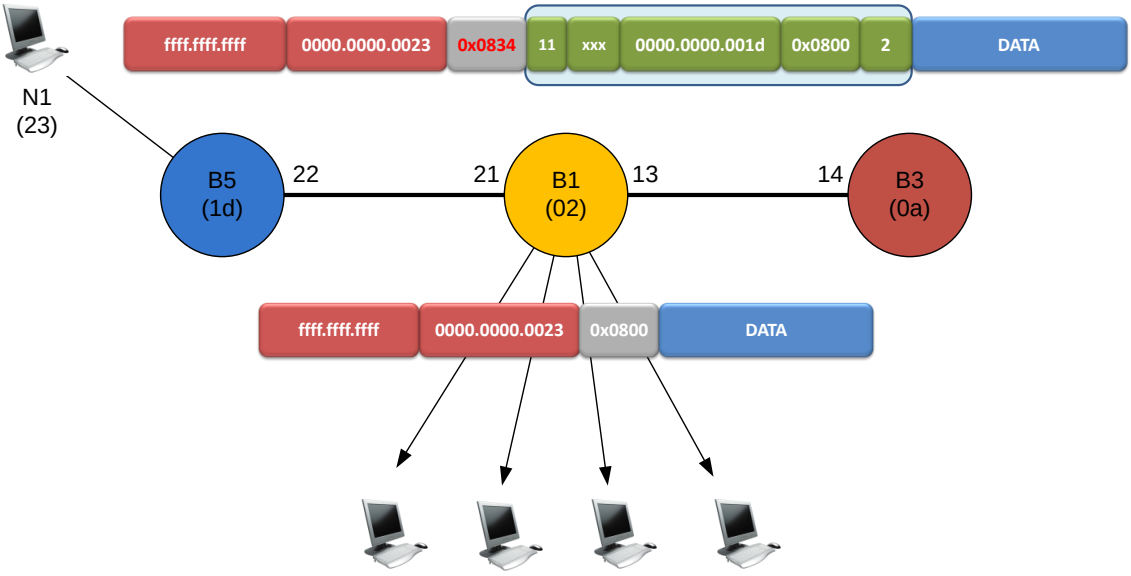


Figure 8-25 - B1 floods broadcast to host interfaces

Meanwhile, B3 received the SCS UF packet from B1 and after extracting the original broadcast frame from the SCS UF packet, it also flooded it to all its host interfaces:

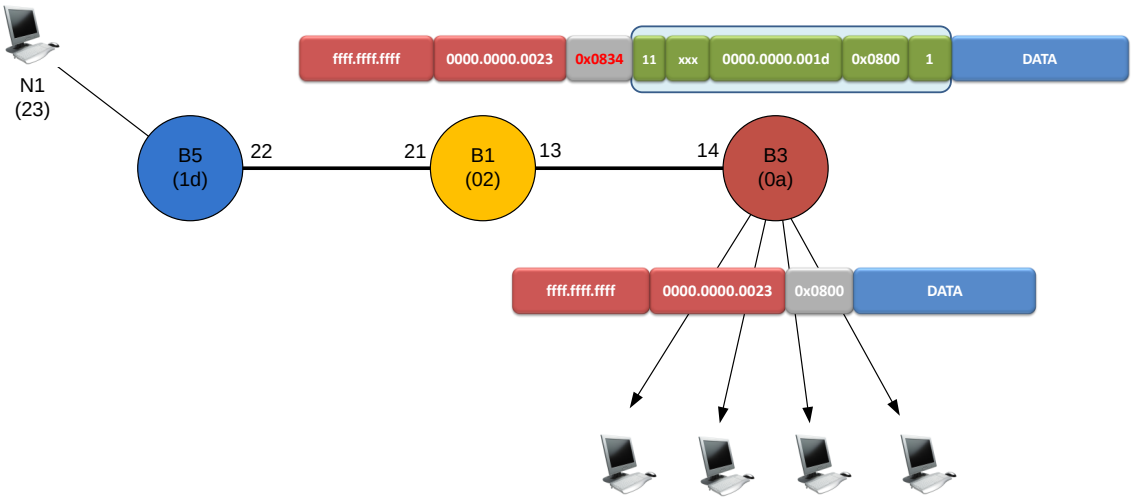


Figure 8-26 - B3 floods broadcast to host interfaces

One major improvement can be achieved over traditional STP networks with unknown destination unicast traffic. The original STP blocks all redundant links, so no MAC addresses are learnt and the unknown unicast must travel all the way to the STP Root Bridge, in some circumstances. In such a scenario, all traversed STP bridges would flood the traffic throughout all ports. Picture 1 of Figure 8-27 illustrates such flooding.

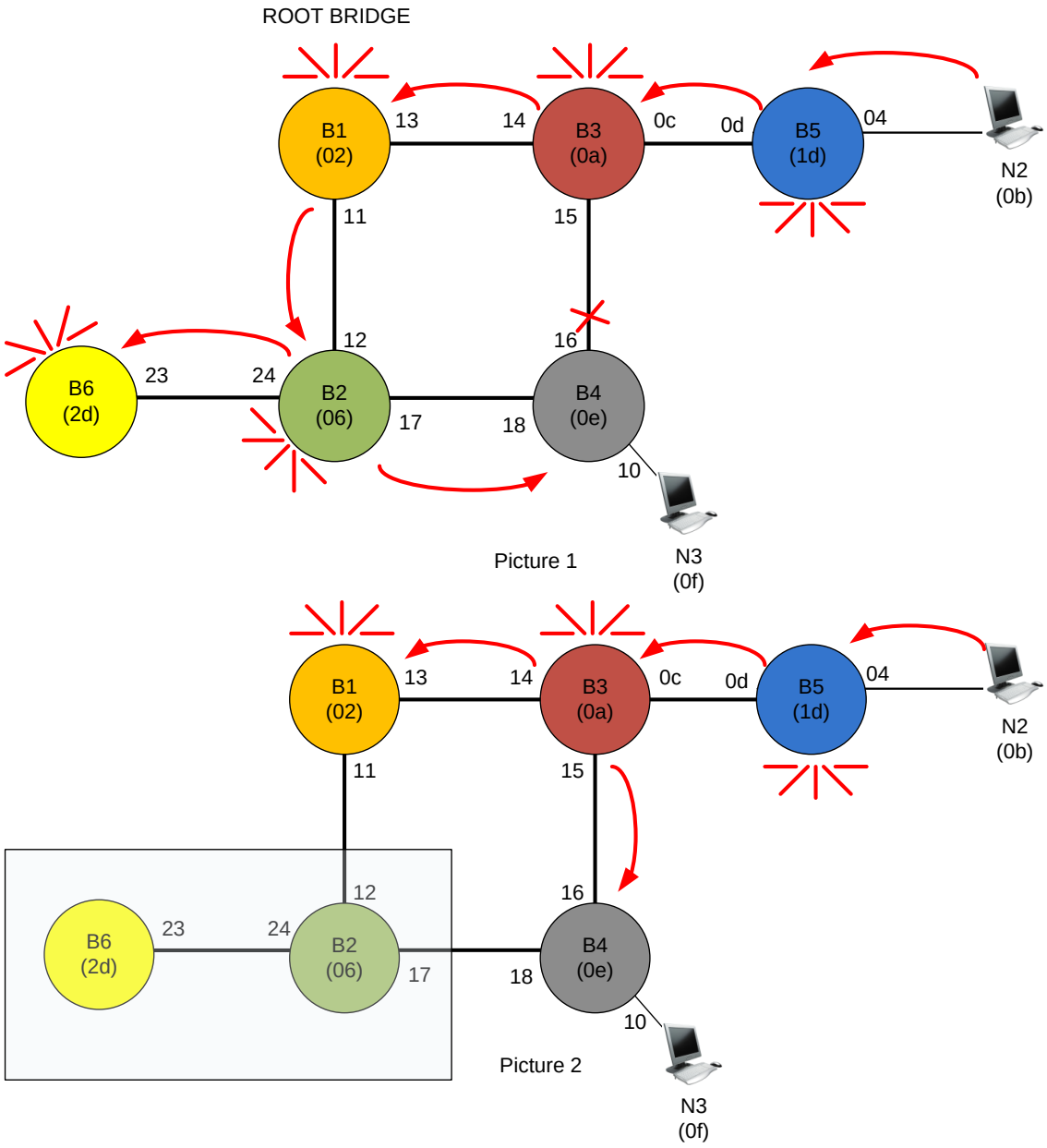


Figure 8-27 - STP vs SCS unknown unicast flooding

With SCS, major flooding reduction would occur. If bridge B3 is delegate of B5 towards B1, B2, B4 and B6, and B4 is delegate of B3 towards B2 and B6, bridges B2 and B6 would never receive the unnecessary traffic to be flooded to all its interfaces. That way, no flooding would occur in B1-B2 link, all B2 interfaces, B2-B6 link and all B6 interfaces. That's great bandwidth savings and resources optimization achieved by SCS.

Loop avoidance and forwarding mechanisms

We have seen previously the SCS Delegation Mechanism and the SCS Update packets creation and exchange. This sub-chapter ends the *Part III - SCS New Approach* section of the thesis, by presenting some vital loop avoidance and forwarding mechanisms that exist in the Flooding Process.

That way, upon SCS UF packets reception:

1. The receiving bridge ensures its SCSID differs from the SCS UF packet Origin SCSID. If not, the SCS UF packet is dropped.
2. If the SCS UF packet contains an Origin SCSID that is in bridge's NB Table and the SCS UF packet is received from other neighbour, the bridge drops the SCS UF packet because it has already received it directly from the adjacent neighbour with Origin SCSID. The flooding ends for this bridge.
3. If the Ethernet destination MAC address is unicast and the bridge has in its FwdT a record for that destination:
    a. If FwdT interface is a *host* interface, the bridge unicasts the original frame to that interface, stopping the flooding.
    b. If FwdT interface is a *bridge* interface and the bridge is delegated for the adjacent bridge that sent that frame to it, the bridge sends the SCS UF through that interface. If the bridge is not delegate, it does not send the packet via that interface, because *some other bridge* has that responsibility. This behaviour avoids packet duplication. The flooding stops for this bridge.
4. The bridge extracts the original packet from the SCS UF packet and floods it throughout all its *host* interfaces.
5. The bridge accepts SCS UF packets originated at the bridge with Origin SCSID, only from its adjacent bridge which is used to reach that Origin SCSID bridge. For example, SCS UF packets originated at bridge By are only accepted if arriving from Bridge Bx, if the bridge FT table has a record By/Bx. Otherwise, the SCS UF packet is dropped.
6. The bridge performs a reverse path check, querying the DT for a specific entry Source SCSID / Origin SCSID. If exists, the SCS UF packet is no longer forwarded. This event signals the bridge to break a loop, as it received a SCS UF packet from a bridge for which it is delegated to reach the origin bridge. For example, Bx has DT= By/Bz. This means that Bx is used by By to reach Bz. If bridge Bx receives from By a SCS UF packet with Origin SCSID Bz, that's not possible, unless a loop exists and Bx has to open it.
7. The bridges queries it's DT for the adjacent bridge SCSID, which sent the packet. Please note that it does not query for the Origin SCSID; queries for the SCSID of the adjacent neighbour connected to the SCS UF packet input interface. If the bridge is not delegate for that bridge, it does not create more SCS UF packets.
8. If 5, 6 and 7 does not occur, the bridge creates SCS UF packets and sends them accordingly to DT.

PART IV - Simulations

Part IV presents the methodology and the device models used to simulate the *Self-Configurable Switches Protocol*.

9 SCS Protocol Simulation

SCS is a new protocol. Before moving on to physical implementations, SCS must be assessed on network simulators.

Ns-3 network simulator was chosen to test SCS algorithm, potential and performance over different network topologies.

This chapter describes how SCS was coded in ns-3 and the changes performed on existent ns-3 modules, which enable SCS to be assessed.

9.1 NS-3 Simulator

Ns-3 means *network simulator, version 3*. Being version 3, it is natural to have a version 2. However, ns-3 abandoned backward-compatibility with ns-2 and was written from scratch using C++. Its first release was made in June 2008 and is updated quarterly.

Ns-3 version 3.13 was used to simulate SCS protocol. Latest version is 3.14, released in June 2012.

Ns-3 is free software under GNU GPLv2 licence [32], targeted for research and educational purposes. It is a discrete-event network simulator, in the sense of simulated events being chronological sequenced, occurring at specific time instants.

Ns-3 is time-consuming to learn and use. All configurations, debugs, executions and outputs are performed at the command line, as no IDE is maintained by ns-3 project. Personally, this was deterrent to natural work progress, as my background in ns-3, or any other ns version, was null. However, basic models of ns-3 are documented and as soon as ns-3 *philosophy* is interiorized, it's all about C++ programming, which, for a *pure C guy*, it was challenging to track so many *abstractions*!

Overall, ns-3 simulation models are quite realistic, which allow ns-3 to be used as a real time network emulator. I'm strongly convinced that ns-3 provides results proximate to real world. That way, I believe it was a good choice to assess and evaluate the SCS protocol.

Ns-3 information can be accessed at its main web site http://www.nsnam.org/.

The available documentation for version 3.13 is provided under the documentation tab http://www.nsnam.org/docs/release/3.13/doxygen/index.html.

9.2 NS-3 Abstractions

This section presents some of the basic ns-3 terms, which were relevant for SCS coding.

- ***Node*** - Basic computing device that connects to a network. This ns-3 abstraction is represented by the class *Node*. For our simulations, SCS bridges are *special* nodes.
- ***Channel*** - The communication media. Represented by the C++ class *Channel*. SCS simulation uses *CsmaChannel*, to model Ethernet CSMA communication.
- ***NetDevice*** – The interface for a node. A *net device* is installed in a *node*, allowing the node to communicate with other *nodes* via *channels*. In C++ is represented by the class *NetDevice*.
- ***BridgeNetDevice*** – It's a virtual net device that bridges multiple LAN segments, aggregating real net devices to implement data plane forwarding of 802.1D.

9.3 SCS Protocol Model

The *BridgeNetDevice* abstraction implements the data plane of a bridge. Adding a *BridgeNetDevice* to a *Node* you get a bridge that forwards traffic between its multiple segments and feeds a forwarding table by learning the source addresses of frames.

802.1D control plane, the Spanning Tree Protocol, is not implemented.

This is the starting point for modelling the SCS protocol: a simple bridge without control plane, whatsoever.

Regarding the "learning" process associated with *BridgeNetDevices*, without underestimating in any way the excellent work performed previously, this process is not suitable for SCS model, as the forwarding table is kept at the *BridgeNetDevice* level, which would reduce SCS protocol's intelligence on decisions it may have to take. That way, SCS model uses a completely new, centralized forwarding table, coded at the *Node* level.

The *Node* class was completely re-coded to accomplish SCS requirements and objectives.

The SCS bridge abstraction is accomplished via *BridgeNetDevices* attached to a "*redesigned" Node* enhanced by the SCS protocol, which works at both control and data plane. All the SCS intelligence, methods, tables, headers, frames and algorithms were developed from scratch at the Node class.

This project started from a simple bridge model that forwards received frames to the known destination interface, or broadcasts in case it doesn't know the destination port, and revolutionized it with complex logic towards the SCS bridge model.

Figure 9-1 and Figure 9-2 illustrate the model's complexity and scope, for both the native ns-3 bridge and the developed SCS bridge model, respectively.

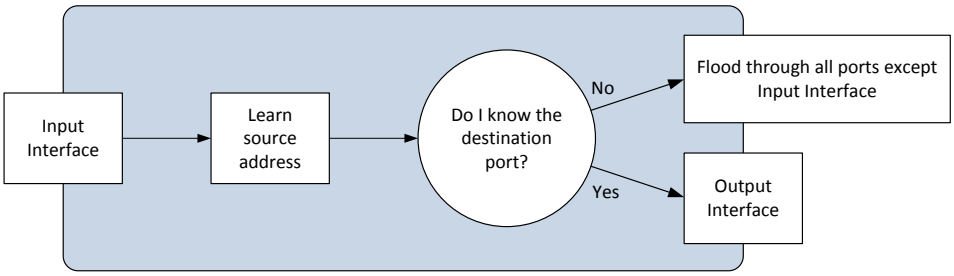


Figure 9-1 - Ns3 bridge model

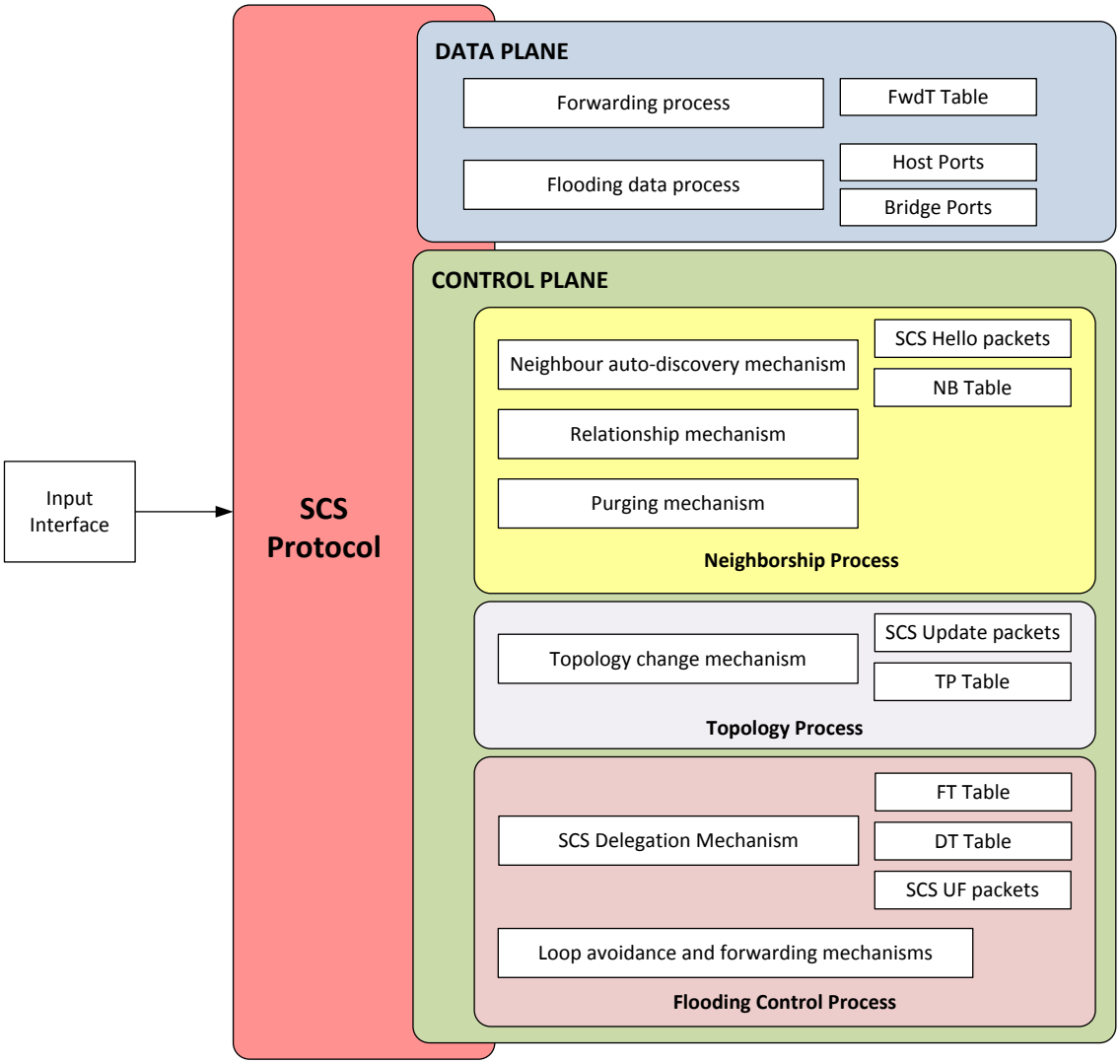


Figure 9-2 - SCS bridge model

9.4 Simulation Output

SCS was coded in ns-3 taking into consideration analysis and debug purposes. That way, a lot of information is sent to the ns-3 console whenever a simulation script is run, in order to allow the assessor to evaluate the SCS state machine progress and all the control messages exchanged between the SCS bridges.

Figure 9-3 shows an excerpt of the output for a particular test performed, where the memory and MAC addresses of five nodes (simulating PCs) are presented. Also, all the interfaces of SCS Bridge 1 are listed, both memory and MAC addresses.

```
N1:0x8b40728  MAC= 00:00:00:00:00:03
N2:0x8b409a0  MAC= 00:00:00:00:00:07
N3:0x8b40b60  MAC= 00:00:00:00:00:0b
N4:0x8b40cb0  MAC= 00:00:00:00:00:0f
N5:0x8b40e08  MAC= 00:00:00:00:00:13
------------------------------------------------------------
Bridge 1: 0x8b40f98
Bridge 1 - Interface 0: 0x8b421d8  MAC=00:00:00:00:00:02
Bridge 1 - Interface 1: 0x8b42b28  MAC=00:00:00:00:00:04
Bridge 1 - Interface 2: 0x8b47a28  MAC=00:00:00:00:00:15
Bridge 1 - Interface 3: 0x8b483a0  MAC=00:00:00:00:00:17
```

Figure 9-3 - SCS ns-3 simulation output example

Figure 9-4 illustrates another example of information provided by the SCS code to the user. In this example, SCS Bridge 0000.0000.000e is exchanging SCS Hellos with its neighbour 0000.0000.0012 and the neighbour relationship is in *delayUP* state.

```
t=103.002 : Bridge 00:00:00:00:00:0e NB Table State = delayUP for neighbor 00:00:00:00:00:12 via interface 00:00:00:00:00:1d
t=103.002 : Bridge 02-06-00:00:00:00:00:0e changed NB Table State to UP for neighbor 00:00:00:00:00:12
        and started the TOPOLOGY PROCESS for this neighbor via interface 02-06-00:00:00:00:00:1d

t=103.002 : Bridge 02-06-00:00:00:00:00:0e sent an SCS TP Update Packet to destination 00:00:00:00:00:06
        about entry related to: 00:00:00:00:00:12 with clear flag=0
t=103.002 : Bridge 02-06-00:00:00:00:00:0e sent an SCS TP Update Packet to destination 00:00:00:00:00:0a
        about entry related to: 00:00:00:00:00:12 with clear flag=0

t=103.002 : Bridge 02-06-00:00:00:00:00:0e sent an SCS TP Update Packet to destination 00:00:00:00:00:12
        about entry related to: 00:00:00:00:00:06 with clear flag=0
t=103.002 : Bridge 02-06-00:00:00:00:00:0e sent an SCS TP Update Packet to destination 00:00:00:00:00:12
        about entry related to: 00:00:00:00:00:02 with clear flag=0
t=103.002 : Bridge 02-06-00:00:00:00:00:0e sent an SCS TP Update Packet to destination 00:00:00:00:00:12
        about entry related to: 00:00:00:00:00:0a with clear flag=0
t=103.002 : Bridge 02-06-00:00:00:00:00:0e sent an SCS TP Update Packet to destination 00:00:00:00:00:12
        about entry related to: 00:00:00:00:00:02 with clear flag=0


Node TP Table ------------
     for Bridge 02-06-00:00:00:00:00:0e @t=103.002 :
-----------------------------------------
Bridge SCSID=00:00:00:00:00:0e / Neighbor SCSID=00:00:00:00:00:06 / Inbound Interface=02-06-00:00:00:00:00:1c / Metric=1
Bridge SCSID=00:00:00:00:00:0e / Neighbor SCSID=00:00:00:00:00:02 / Inbound Interface=02-06-00:00:00:00:00:1c / Metric=2
Bridge SCSID=00:00:00:00:00:0e / Neighbor SCSID=00:00:00:00:00:0a / Inbound Interface=02-06-00:00:00:00:00:1a / Metric=1
Bridge SCSID=00:00:00:00:00:0e / Neighbor SCSID=00:00:00:00:00:02 / Inbound Interface=02-06-00:00:00:00:00:1a / Metric=2
Bridge SCSID=00:00:00:00:00:0e / Neighbor SCSID=00:00:00:00:00:12 / Inbound Interface=02-06-00:00:00:00:00:1d / Metric=1
```

Figure 9-4 - SCS messages output example

However, at the middle of time instant t=103.002 seconds, that neighborship became UP and the Topology Process was started, with bridge 000e updating its TP table with neighbour 0012 information and advertising that change to all other neighbours, besides pushing its all TP table to the new neighbour.

Figure 9-5 presents the Delegation Tables (DT) information of five bridges, dumped into user's console at instant t=80 of the simulation.

```
------------------------------------------------------------------------
- DUMP BRIDGES DELEGATION TABLES ---------------------------------------
------------------------------------------------------------------------


Bridge Delegation Table ------------
     for Bridge 02-06-00:00:00:00:00:02 @t=80 :
-------------------------------------------
Interface = 02-06-00:00:00:00:00:15 / Source Bridge =00:00:00:00:00:06 / Flood to Bridge = 00:00:00:00:00:0a
Interface = 02-06-00:00:00:00:00:17 / Source Bridge =00:00:00:00:00:0a / Flood to Bridge = 00:00:00:00:00:06
-----------------------------------------------------


Bridge Delegation Table ------------
     for Bridge 02-06-00:00:00:00:00:06 @t=80 :
-------------------------------------------
Interface = 02-06-00:00:00:00:00:16 / Source Bridge =00:00:00:00:00:02 / Flood to Bridge = 00:00:00:00:00:0e
Interface = 02-06-00:00:00:00:00:1b / Source Bridge =00:00:00:00:00:0e / Flood to Bridge = 00:00:00:00:00:02
-----------------------------------------------------


Bridge Delegation Table ------------
     for Bridge 02-06-00:00:00:00:00:0a @t=80 :
-------------------------------------------
-----------------------------------------------------


Bridge Delegation Table ------------
     for Bridge 02-06-00:00:00:00:00:0e @t=80 :
-------------------------------------------
Interface = 02-06-00:00:00:00:00:1a / Source Bridge =00:00:00:00:00:0a / Flood to Bridge = 00:00:00:00:00:12
Interface = 02-06-00:00:00:00:00:1c / Source Bridge =00:00:00:00:00:06 / Flood to Bridge = 00:00:00:00:00:12
Interface = 02-06-00:00:00:00:00:1d / Source Bridge =00:00:00:00:00:12 / Flood to Bridge = 00:00:00:00:00:06
-----------------------------------------------------


Bridge Delegation Table ------------
     for Bridge 02-06-00:00:00:00:00:12 @t=80 :
-------------------------------------------
-----------------------------------------------------
```

Figure 9-5 - DTs output example

Besides control information, ns-3 allows packet captures at *netdevice* level, generating *pcap* packet trace files, which can be read using *tcpdump* or *wireshark*, for example. Figure 9-6 pictures a *pcap* file created from a SCS ns-3 simulation, being analysed on wireshark.

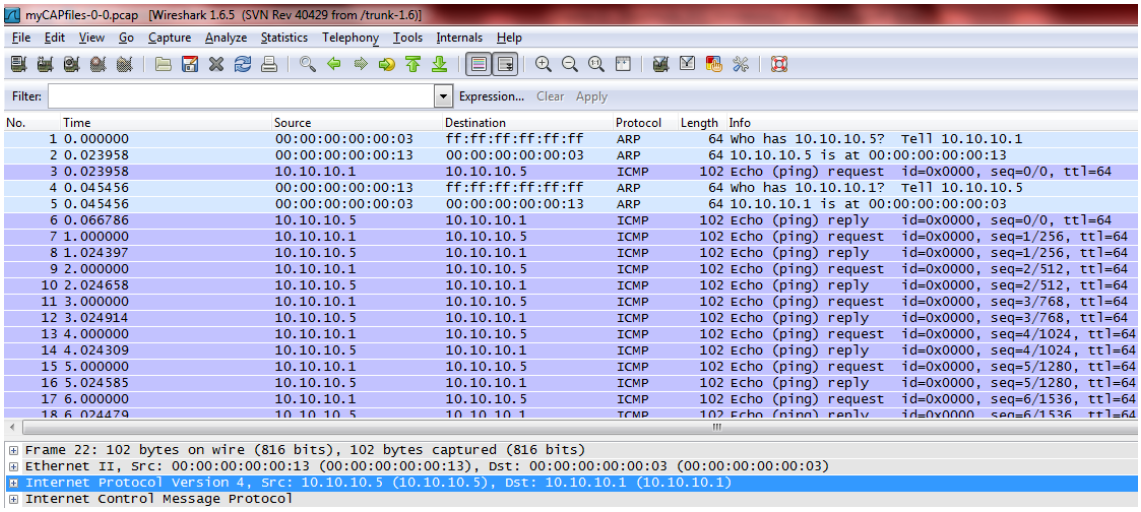


Figure 9-6 - Reading pcap file on Wireshark

PART V – Protocols Assessment

Part V starts with the SCS protocol overhead analysis, in terms of processor and link bandwidth requirements. Then, it presents the ultimate facts about *Self-Configurable Switches Protocol* feasibility to succeed Spanning Tree Protocols. It's a roll of cases where SCS overcomes the problems Spanning Tree Protocols face, one by one, either in theory, as in practical experimentations.

10 SCS Protocol Overhead

The next chapters 11 and 12 will be focused on SCS protocol comparison over STP, proving the SCS superior response over all service impacting issues STP protocols face.

This chapter, however, presents an analysis of SCS focused on the protocol overhead for a simulated complex network topology assembly, step-by-step. Then, two broken parts of the complex topology will be merged to evaluate SCS behaviour in terms of overhead and connectivity response. At the end of the chapter, the overhead produced via random link failures will be presented and discussed.

Figure 10-1 pictures the *somehow* complex topology over which SCS will be assessed.

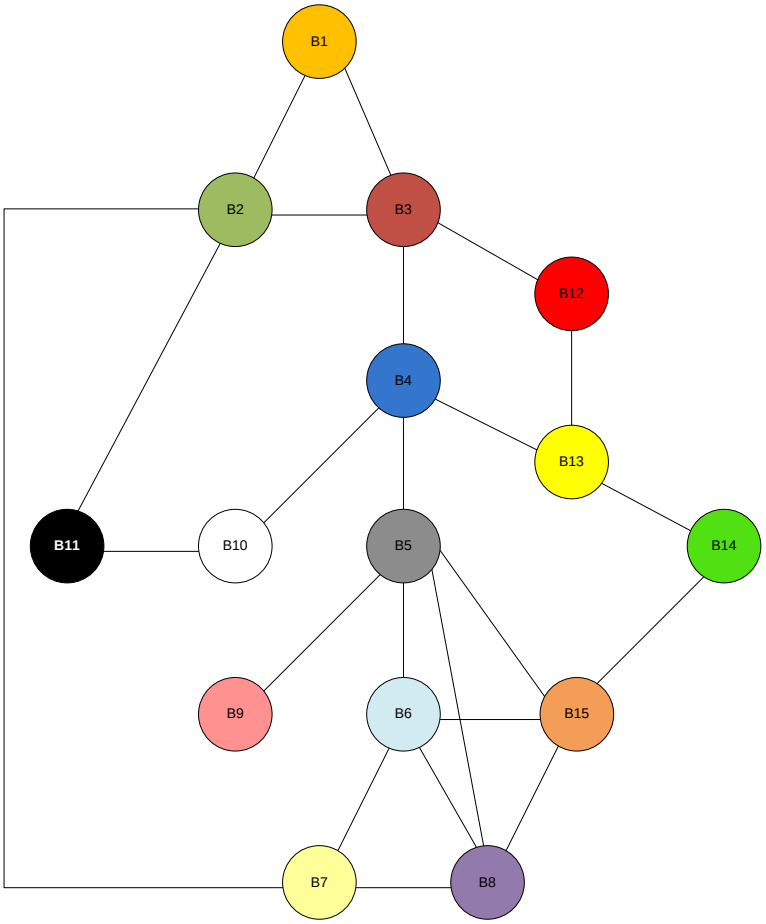


Figure 10-1 – Sample complex topology

The above network topology was structured to comprise several common topologies. For example, bridges B5, B6, B8 and B15 form a *full mesh* topology and bridges B1, B2, B3, B4, B12, B13 and B14 form a multi-equal cost path from B14 towards B1 and B2 (Figure 10-2). Bridges B1, B2, B7, B8, B15, B14, B13, B12 and B3 form a *ring* topology (Figure 10-3), like B2, B3, B4, B10 and B11. The same way, the above topology allows observing SCS behaviour when two different network topologies are merged (Figure 10-4), which will be further analysed in this chapter.

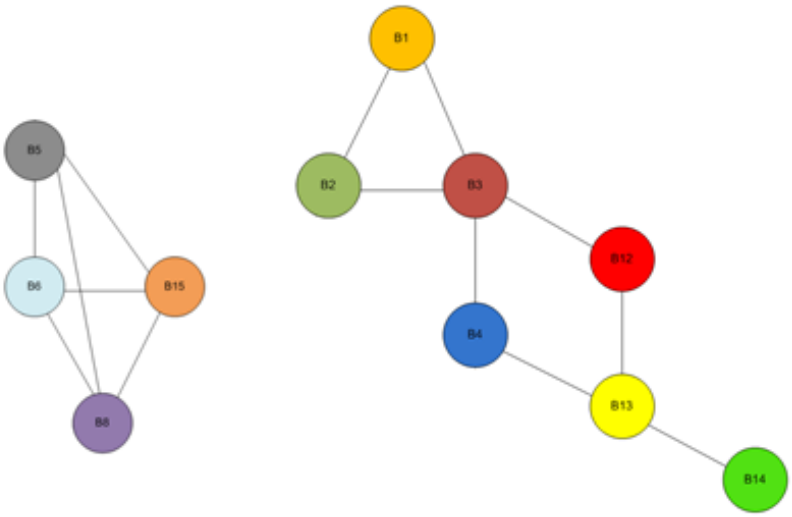


Figure 10-2 - Full-mesh and multi-path topologies

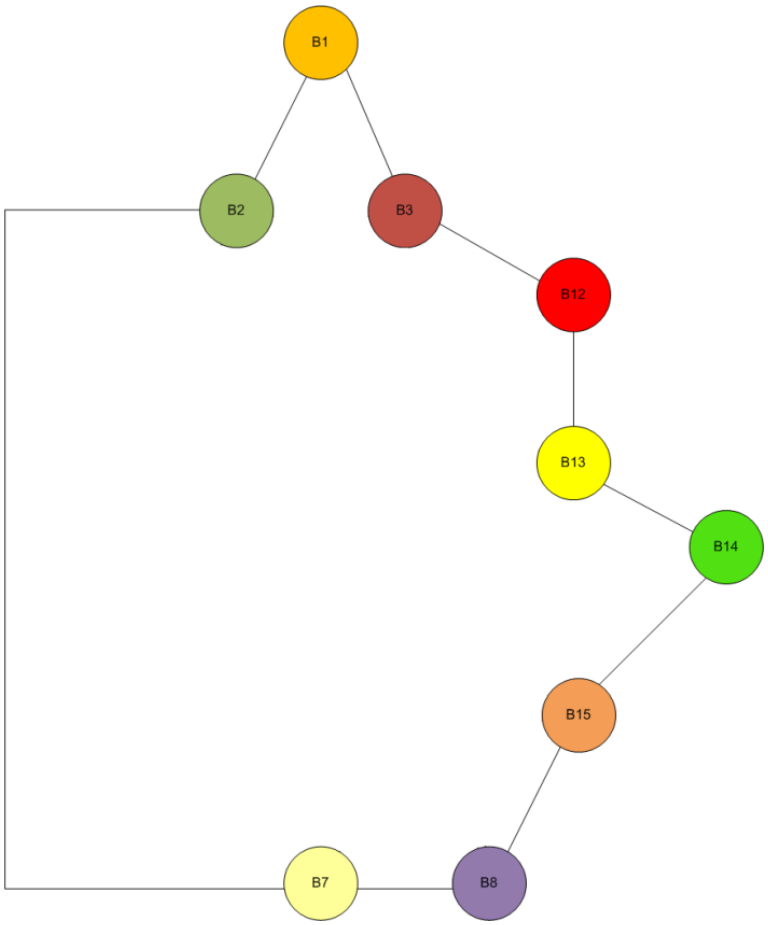


Figure 10-3 - Ring topology

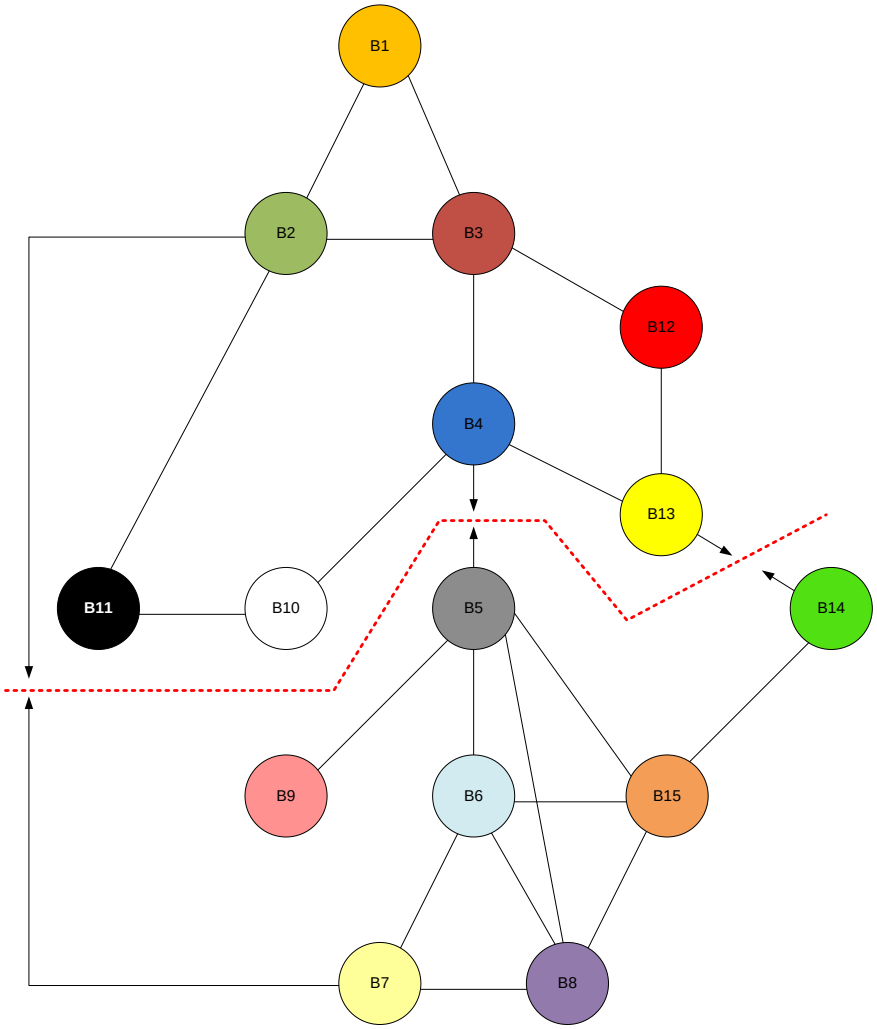


Figure 10-4 - Topologies merging

10.1 Topology Assembly Overhead Analysis

The order by which the assembly was made was:

1. Red path
2. Green path
3. Blue path
4. All the remaining

Those paths can be observed in Figure 10-5.

This way, we can present analysis for SCS overhead in terms of the number of SCS messages exchanged, for each path assembly (individual contribution) and for the overall overhead distribution (cumulative contribution).

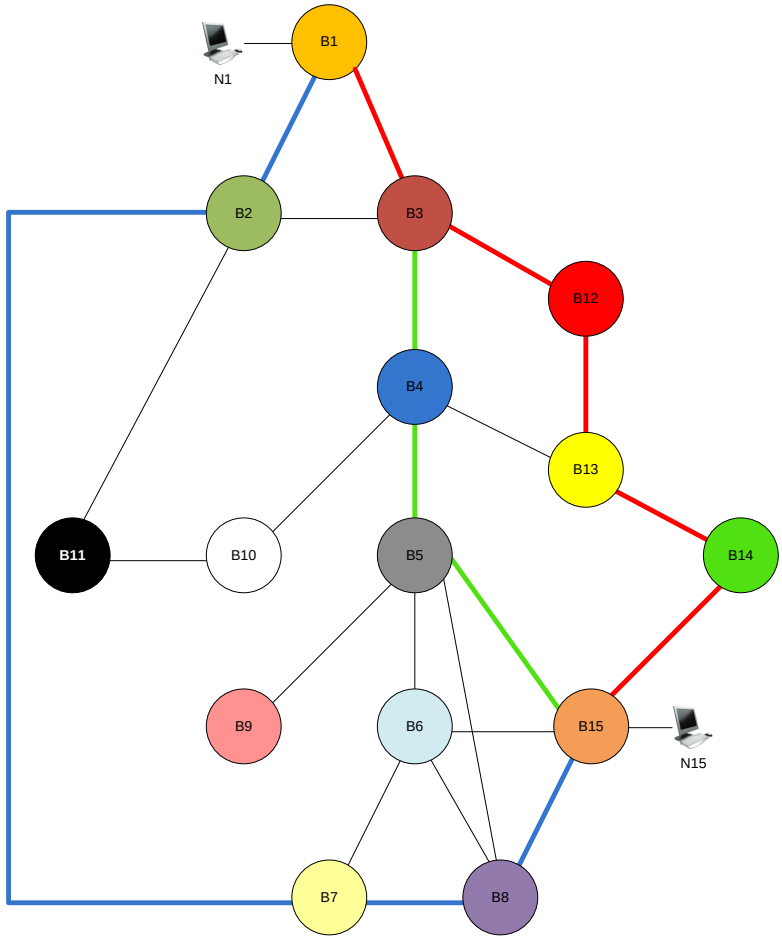


Figure 10-5 - Topology assembly analysis

As the following graphs present, as soon as the network becomes more complex, the required SCS message exchange increases, especially when redundant paths are included.

Figure 10-6, Figure 10-7 and Figure 10-8 present the total raw number of messages exchanged between bridges, suggesting the neighbour to change its topology table, delegation creation and delegation deletion, respectively. As it can be seen, to create the redundant, yet simple paths red, green and blue, summing up twelve links between bridges, it was necessary a small amount of messages, when compared with the remaining eleven links (represented on the graphs by the *other* bar). As more redundancy is added to the network, the SCS protocol will have to process a significantly more complex link catalogue and more messages will have to be necessarily exchanged.

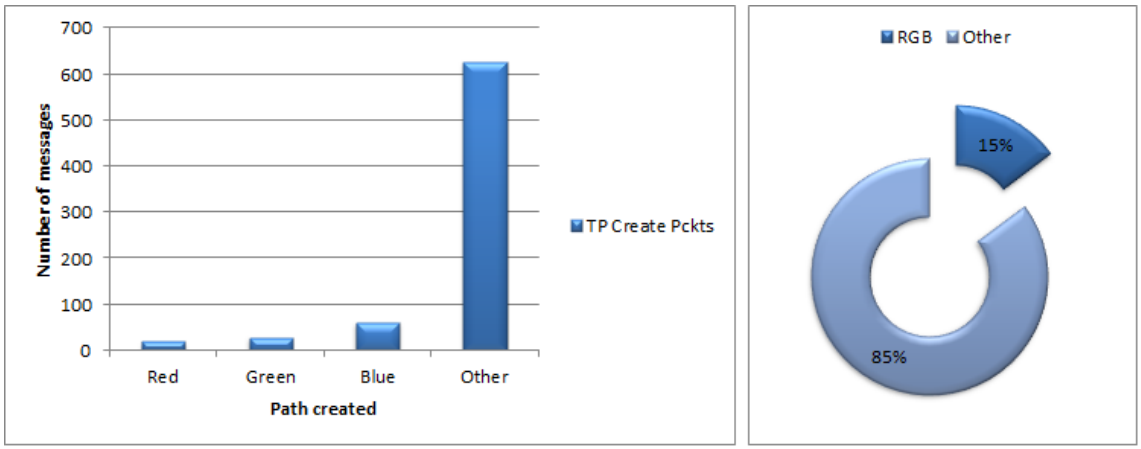


Figure 10-6 - Create topology notification

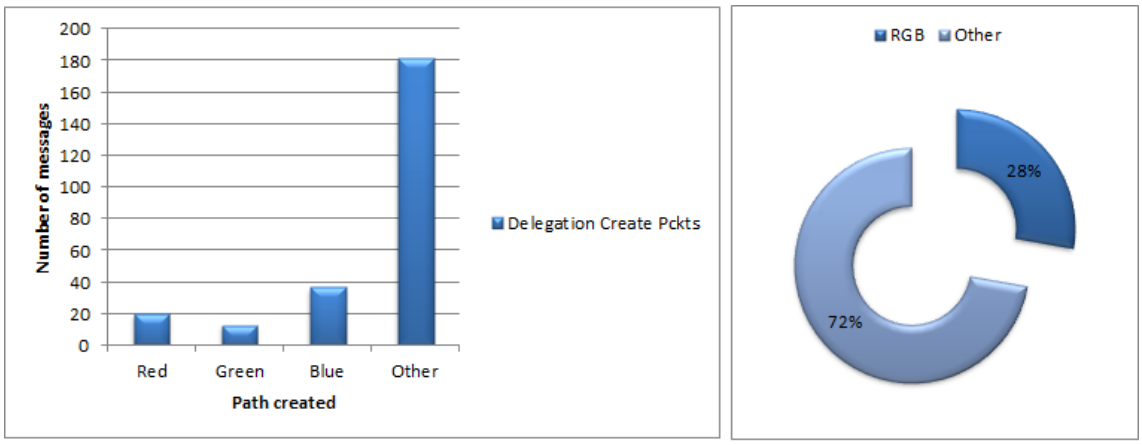


Figure 10-7 - Create delegation notification

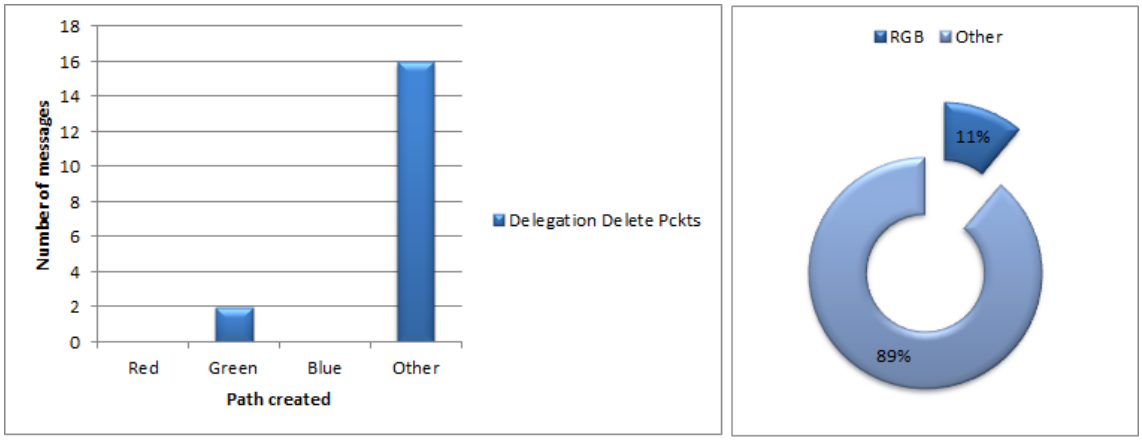


Figure 10-8 - Delete delegation notification

The following graph in Figure 10-9 presents the measure of the number of topology create packets exchanged per second, providing a more precise snapshot of the protocol behaviour as the number of links and network redundancy increases.

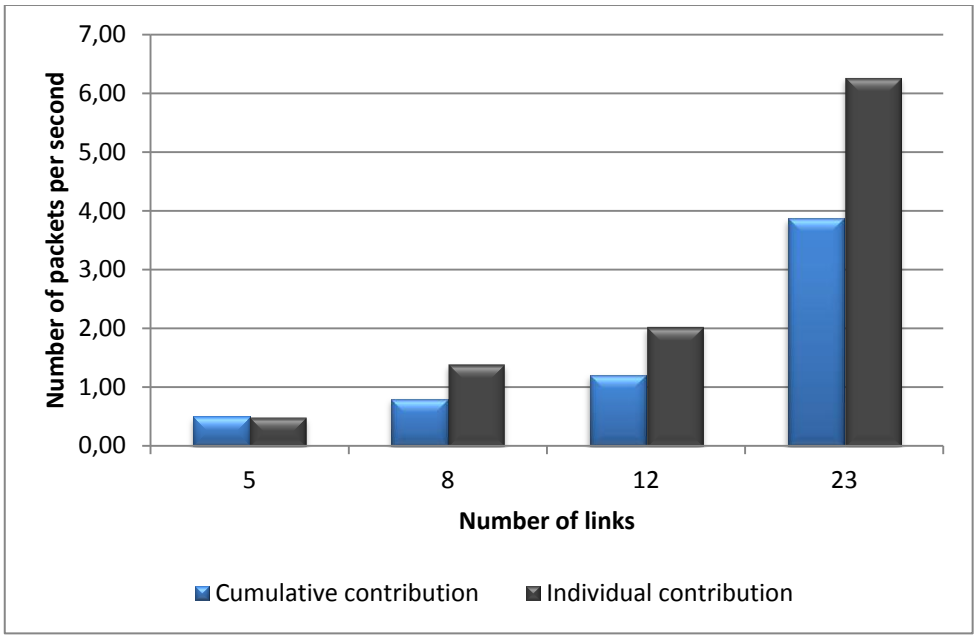


Figure 10-9 - Cumulative and individual contributions

SCS message exchange volume is clearly dependent upon the existent network redundancy. However, even for complex topologies like the one presented in the sample topology of Figure 10-1, the volume of messages exchanged per second is not so significant. Like the graph on Figure 10-9 presents, to assembly those 23 links, SCS needs to exchange near 4 SCS packets per second, in average. Naturally, as more links are added to the network, more messages will be needed to be exchanged and the individual contribution bars will pull up the cumulative mean.

SCS Bandwidth Consumption

Regarding bandwidth consumption, although the number of messages could reach near 800, upon all topology assembly, the overall bandwidth overhead is insignificant, as those control messages are below 100 bytes, each.

SCS Hello packets are, in turn, periodic, 1 pps by default. This keepalive mechanism overhead is also irrelevant for modern bridges, both in bandwidth and processing power requirements.

10.2 Topology Merge Overhead

Topologies merging can be problematic for some protocols. SCS behaviour in such circumstances will be evaluated and, for that, the two topologies pictured in Figure 10-4 will be merged.

Upon merging, the topology tables will need to be completely recalculated. Several simultaneous re-computations and many messages will need to be exchanged to achieve a stable and converged topology.

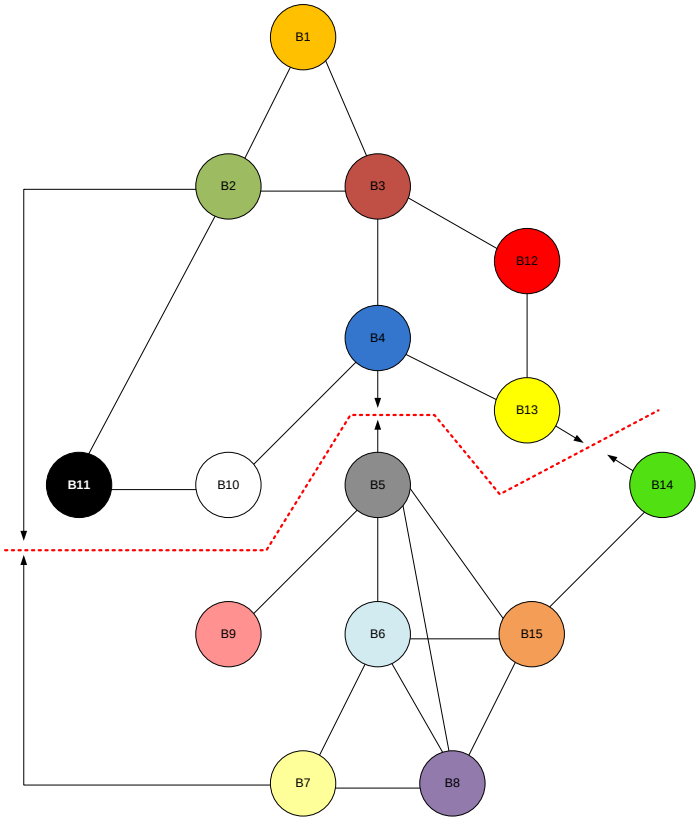


Figure 10-4 - Topologies merging

As protocol overhead is concerned, the following table summarizes the total number of messages exchanged due to the merging process, observed the simulation.

Topology Packets	Create Delegation Packets	Remove Delegation Packets
610	167	4

Table 14 - Topology merging simulation results

Note the number of delegation and topology packets exchanged, which reflect the major recalculations SCS needed to perform. However, as soon as the networks are merged and converged into a single topology, connectivity from n1@bridge1 towards n15@bridge15 is possible; with SCS, it's immediately after the delayUP interval.

Taking into consideration that the simulated merging interval was t=150s, Figure 10-10 presents a packet capture on n15.

```
154.041423 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 0, length 64
154.045820 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 1, length 64
154.066877 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 0, length 64
154.067061 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 1, length 64
155.015155 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 2, length 64
155.015155 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 2, length 64
156.015746 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 3, length 64
156.015746 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 3, length 64
157.015688 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 4, length 64
157.015688 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 4, length 64
158.015267 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 5, length 64
158.015267 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 5, length 64
159.015169 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 6, length 64
159.015169 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 6, length 64
160.016003 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 7, length 64
160.016003 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 7, length 64
161.016548 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 8, length 64
161.016548 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 8, length 64
162.015226 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 9, length 64
162.015226 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 9, length 64
```

Figure 10-10 - Connectivity from n1 to n15

10.3 Random Failures Overhead

Is this section, random link failures will be generated and the consequent SCS protocol overhead, in terms of volume of messages exchanged, is presented and discussed.

For simulation purposes, the time instant and link number were generated randomly. The following Figure 10-11 and Table 15 present the links affected and the SCS message volume observed in each test.

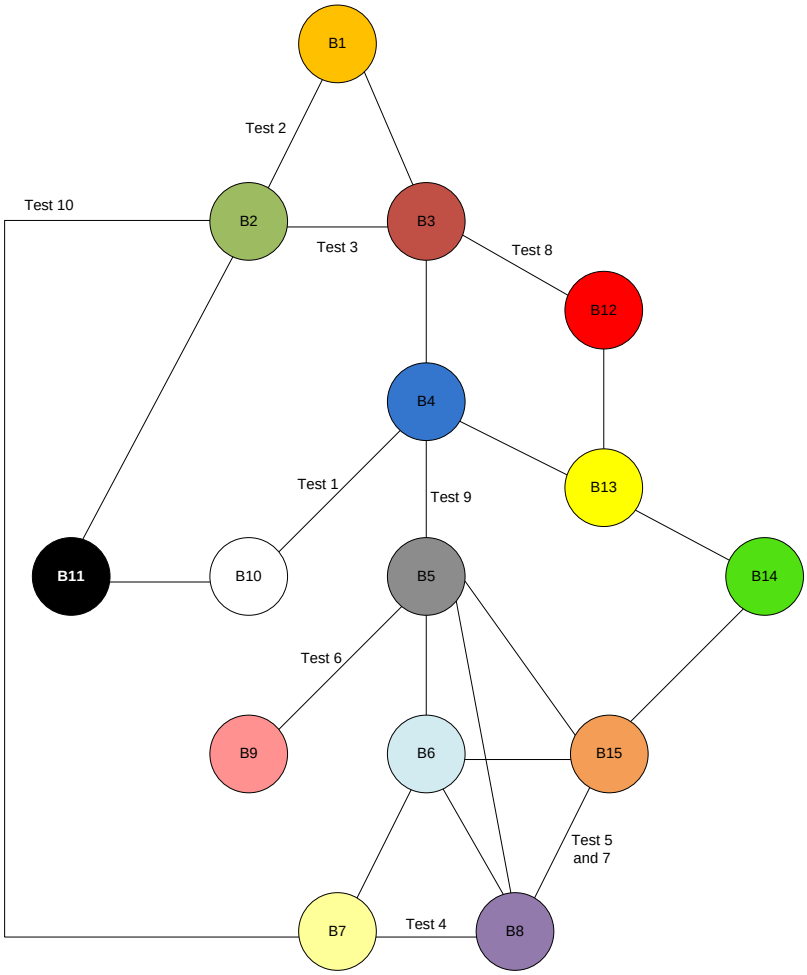


Figure 10-11 - Links affected by failures

Test	Link Failed	Failure t	SCS TP Create	SCS TP Delete	SCS Del. Create
1	8	287.96	301	81	37
2	1	494.68	155	37	10
3	3	370.52	492	117	39
4	21	312.77	492	116	16
5	24	299.33	310	65	8
6	14	501.34	0	53	4
7	24	988.41	313	65	10
8	7	644.90	341	96	36
9	10	815.27	831	229	82
10	4	437.44	606	177	43

Table 15 - SCS messages volume, per test

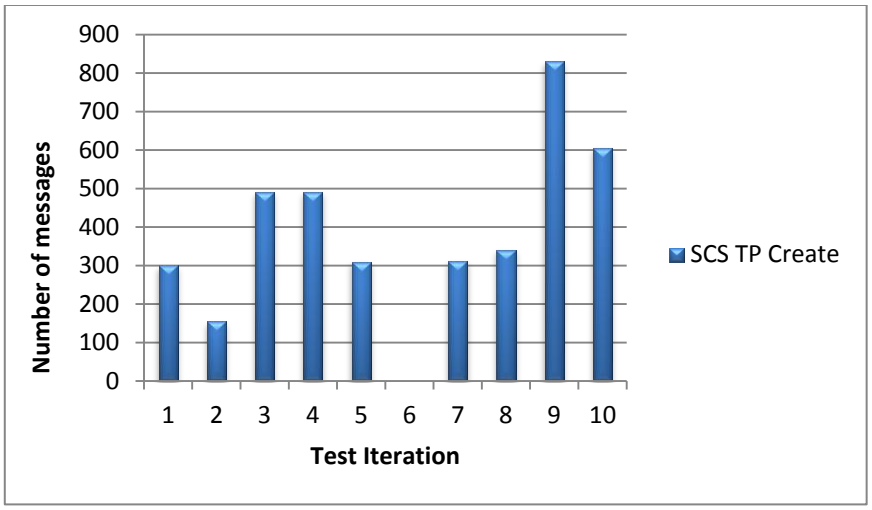


Figure 10-12 - Number of topology create messages

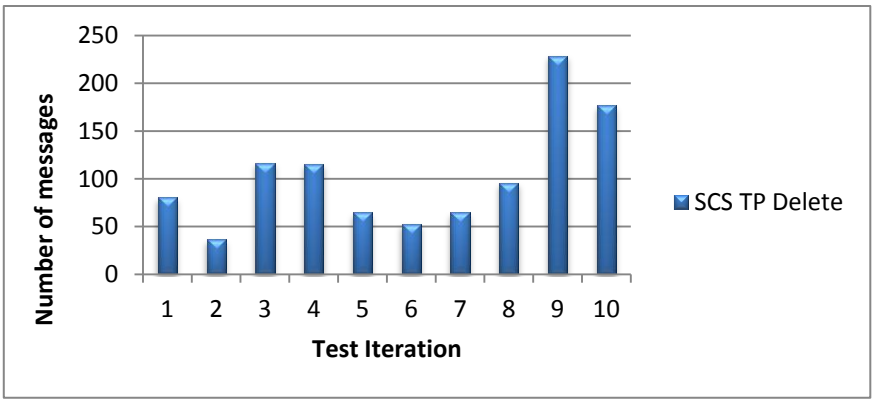


Figure 10-13 - Number of topology delete messages

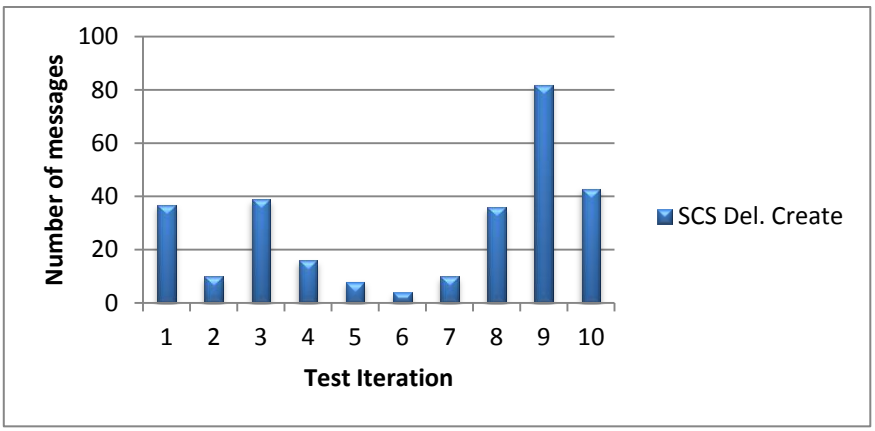


Figure 10-14 - Number of delegation create messages

As Table 15 data and the above three graphics present, the volume of messages exchanged is naturally dependant on the *importance* of the link to the converged topology. For example, test #6 produced the most insignificant volume, as it is related to bridge9 that became isolated. In this case, only topology delete and delegation delete messages were triggered. Conversely, tests #9 and #10 produced the most significant volume of messages. Referring back to Figure 10-11, we state that the

affected links are the ones *merging* the topologies (plus the link B14-B15), like the previous chapter presented. Such failures require necessarily more SCS computations.

This chapter presented the irrelevant link bandwidth overhead of SCS, but the clear dependence of SCS messages volume over network redundancy and link relevance, nevertheless maintaining somehow low level values, taking into consideration the network complexity and size.

11 STP Primacy over SCS

I'm strongly convinced of SCS primacy over STP in almost all aspects. All simulations and lab tests confirm our theory. However, STP has the experience of a lifetime and innumerable research was performed to improve Spanning Tree features and behaviour. So, it is natural to agree with some ideas and behaviours of STP.

Although many advances over STP convergence times were achieved during the past two decades, the STP algorithm is not oriented to provide redundancy, optimal paths and load sharing, like SCS is. It is not possible for STP to provide such features without breaking up with the way Spanning Tree operates. This would mean Spanning Tree Protocol reneging on its origins, the Spanning Trees calculations!

Nevertheless, the concept behind MSTP protocol, specifically the creation of MSTP regions running disjoint Spanning Tree instances, is quite keen. This particularity of MSTP is not supported on SCS, yet.

MSTP regions provide traffic segregation. Is it planned for SCS the support for *SCS Domains*, as it will be described on the *Future Work* section of the *Conclusions* chapter. *SCS Domains* will allow traffic distribution over redundant topologies, which MSTP does not, creating multiple SCS domains, redundant of each other. So, besides resiliency, redundancy, fast convergence, optimal paths and load balancing, SCS will also provide traffic segregation.

12 Revisiting STP, RSTP and MSTP Problems

Throughout Part I chapters 2, 3 and 4, a wide variety of problems were presented and analysed for STP, RSTP and MSTP protocols.

In this chapter, all those issues are revisited one-by-one, revealing the great improvements achieved by replacing Spanning Tree protocols by the Self-Configurable Switches protocol. To corroborate the theory exposed, the output of some experiments, performed on NS-3 simulator for SCS protocol and physical lab setups for Spanning Tree protocols, are presented.

This chapter does not describe in detail those Spanning Tree Protocol problems, once they were thoroughly exposed and explained over the previous STP, RSTP and MSTP chapters.

12.1 Test Beds

The experiments performed to substantiate SCS superior performance over STP were based in two distinct network topologies, presented in Figure 12-1 and Figure 12-2, and eight lab setups. In each scenario, ICMP traffic was used to assess connectivity, with 1 second cadence between each ICMP packet.

SCS was tested using ns3-simulator, whereas STP in real lab setups, for the same topologies.

The hardware used in STP labs was:

- 1x Toshiba Tecra M2, running CentOS 5.
- 1x Dell Latitude E6400, running Windows 7.
- 15x switches D-Link 3010G.

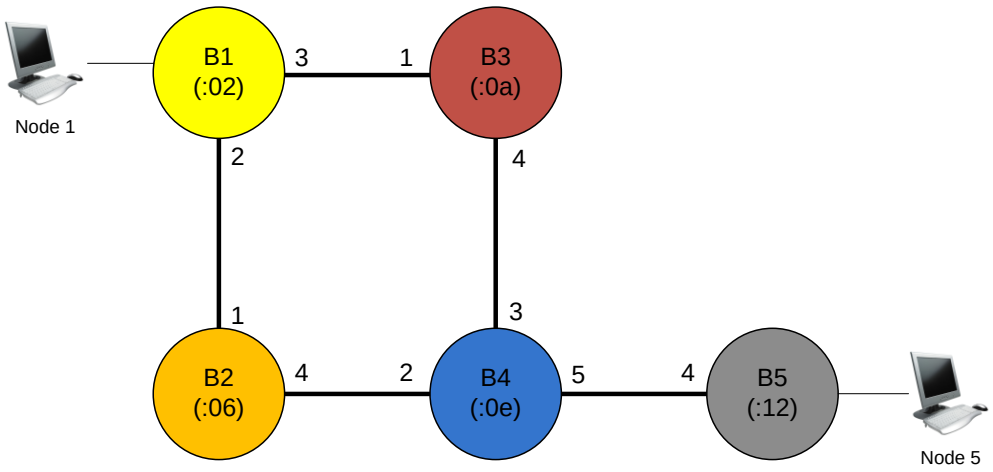


Figure 12-1 - Test bed #1

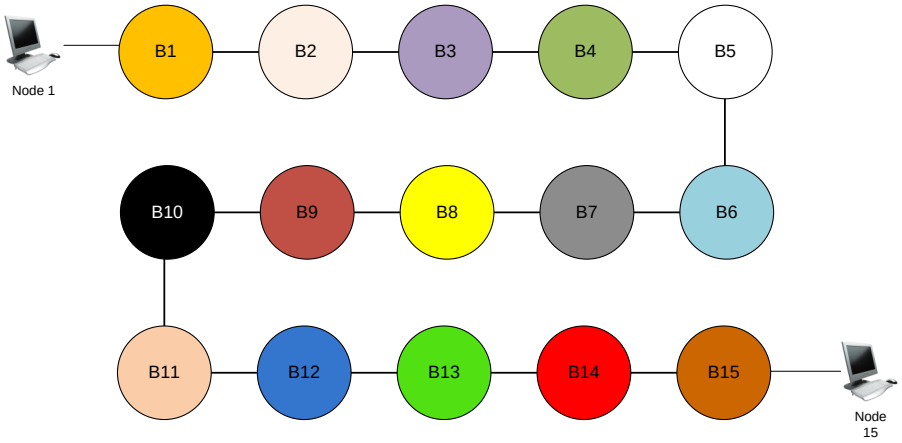


Figure 12-2 – Test bed #2

12.2 802.1D STP Problems

Previously in the *Spanning Tree Protocol* chapter, several problems regarding the 802.1D STP protocol were presented. Those problems were grouped into:

- Configuration-driven STP issues – issues induced by human misconfigurations.
  - Tuning the STP parameters to achieve better performance
  - New Bridges in the Layer 2 production network
  - Bridge ports duplex mismatch
  - Temporary loops
- System-driven STP issues – protocol misbehaviour over system failures.
  - Software / Hardware errors
  - Resource exhaustion
  - Unidirectional links
- Protocol-driven STP issues – protocol inherent deficiencies.
  - Conservative STP timers
  - Network diameter
  - Least cost path
  - Topology change mechanism
  - Convergence black hole

Next, the same issues will be presented but now explaining how SCS avoids and/or overcomes those same problems.

Configuration-driven STP Issues

In this section, human misconfigurations that lead to STP loops are revisited, namely:

- Tuning the STP parameters to achieve better performance
- New Bridges in the Layer 2 production network
- Bridge ports duplex mismatch
- Temporary loops

Tuning the STP parameters to achieve better performance

STP convergence and stability is supported on very low performance timers. Therefore, many network administrators tend to tune STP parameters to achieve better performance. However, instabilities, convergence issues and network meltdown are all they get.

Trying to achieve faster convergence tuning STP timers is very dangerous and completely dependent of the network topology; it requires extreme planning and expertise.

SCS timers are already optimized for stability and fast convergence. Besides, SCS provides redundant paths allowing some great percentage of traffic to remain unaffected in a link failure scenario, for example. Whereas, in STP, some link failures force STP topology recalculation with massive impact on network traffic.

New Bridges in the Layer 2 production network

Careless insertion of a new STP bridge into a STP production network might cause great damage to network availability. STP does not protect, at all, the network against such insertions and massive outages can happen. In most, what network administrators might do is reducing the targeted root bridge priority value to a low value, to prevent such occurrences.

SCS, in turn, enforces network protection against reckless bridge insertion via the simple *Neighborship Key*, as already explained before.

Bridge ports duplex mismatch

The *bridge ports duplex mismatch* is a particular case of the *unidirectional links* issue and will be analysed in that section of the *System-driven STP Issues* group.

Temporary loops

In worst case scenarios, there will be a 2-seconds temporary loop, whose effect depends on the looped traffic, which could even lead to permanent loops. Those temporary loops in STP might prevent a bridge from sending BPDUs, for example due to high CPU, which in turn may cause a port transition from blocking to forwarding, worsening the network loop scenario in a supposed loop-free topology.

In SCS, even in the occurrence of potential temporary loops that might prevent a bridge from sending SCS Hellos, the overall network sanity remains intact, as SCS implements correctly the keepalive mechanism.

If STP loose keepalives (BPDUs), it allows a blocking port to go to the forwarding state, resulting in potential network loops and eventually network meltdown. Conversely, if SCS loose keepalives (SCS Hellos), the neighborship is declared DOWN and the bridge must search for another way to reach that destination, resulting just in

a topology recalculation, as supposed. Moreover, the TTL field existence on SCS UF packets allows SCS to break loops within a few hops, minimizing the loop effect.

As seen above, all the configuration-driven STP issues are mitigated by SCS, just implementing correctly the keepalive mechanism, enforcing TTL countdown, making simple security validations and using adequate convergence timers.

Next, the system-driven STP issues are analysed.

System-driven STP Issues

Some inevitable system events could trigger loss of BPDUs and consequently STP network loops. For example:

- Software / Hardware errors
- Resource exhaustion
- Unidirectional links

Software/Hardware errors, Resource exhaustion and Unidirectional links

All three scenarios can be analysed together as, in STP, they all cause the same behaviour leaped by the *STP loop enabler feature*: loss of BPDUs, which lead to a blocking port transition into the forwarding state and, consequently, a network loop.

Every software and hardware device has errors or might suffer resource exhaustion, which effects cannot be predicted. SCS is not an exception and naturally remains vulnerable to them, like any other protocol. However, comparing to STP, SCS reduces the probability of such events introducing errors on SCS networks, as communication failures between bridges drive to network topology recalculation, not network loops.

Regarding unidirectional links, conversely to STP, in SCS a Tx link failure effectively brings a neighborship down, not the entire network!

For example, in Figure 2-15, if bridge B stops receiving BPDUs from bridge A, it will transition the port into STP Forwarding state and starts forwarding traffic, which causes a loop to occur.

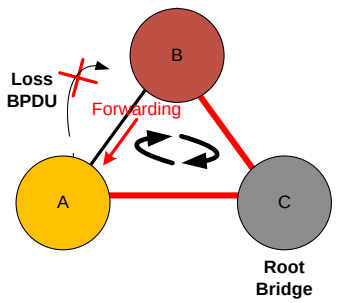


Figure 12-3 – Unidirectional link causes loop in STP

In SCS, that would never occur. If bridge B stops receiving SCS Hellos from A, B brings down A-B neighborship and no traffic flows via that link. As B stops sending SCS Hellos towards A, A would also bring the A-B neighborship down. That would take

about 3 seconds to occur and yet no loop as occurred. Besides, in Figure 12-3 scenario, a loop would never occur if SCS was running, as *flooded* traffic is unicasted between the adjacent neighbours and it is not relayed between them. That way, in the advent of communication difficulties in A-to-B direction, the worst case scenario would be a 3-second black hole from A hosts towards B hosts; A to C communication remained OK.

SCS completely mitigated the potential loop that might happen in STP networks, due to system-driven issues.

Protocol-driven STP Issues

The most relevant characteristics that contribute to resources misuse, potential network instability and unpredictable behaviour of STP over some of the most known network topologies are:

- Conservative STP timers
- Network diameter
- Least cost path
- Topology change mechanism
- Convergence black hole

Let's analyse them for STP and see SCS behaviour under the same circumstances.

Conservative STP timers

During topology changes, Forward Delay and Max Age timers induce unacceptable convergence times close to 50 or 30 seconds. Moreover, tuning the STP timers has several drawbacks and must be carefully planned.

In worst case scenarios, SCS convergence times are close to 3xSCS Hello Interval. However, only a few set of issues cause such big network instability. With correct high availability network design, a link failure would not cause relevant impact, as redundant paths are available.

The following lab was set up, to better compare SCS with STP behaviour regarding convergence times. Lab of Figure 12-4 tests communication impact between node 1 and node 5, upon link B2-B4 failure that causes topology recalculation.

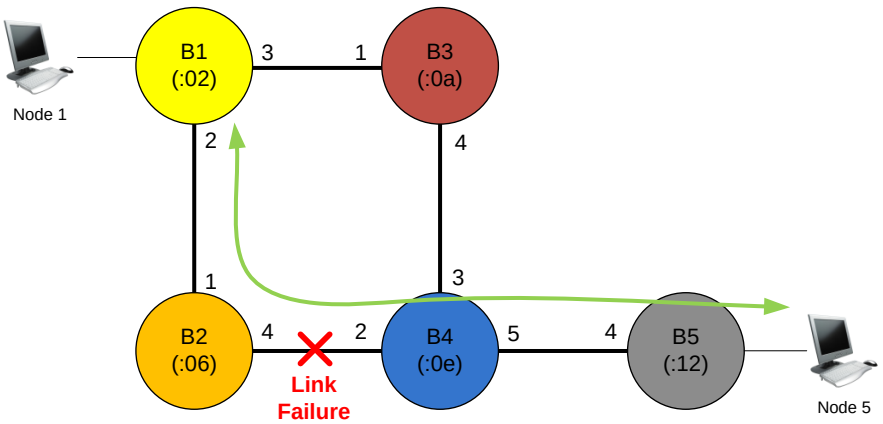


Figure 12-4 - Setup #1: Lab layout

Over Figure 12-4 network, STP would have the following active Spanning Tree topology, highlighted in blue in Figure 12-5.

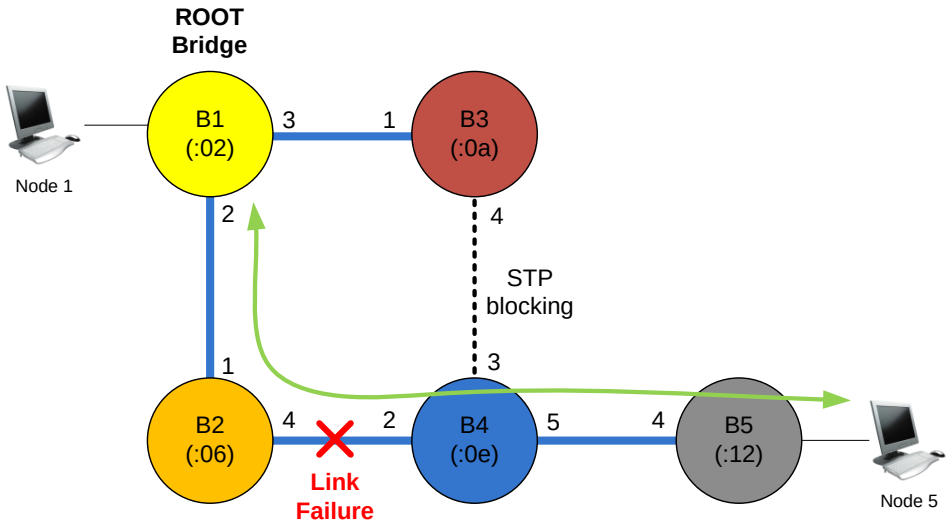


Figure 12-5 - Setup #1: STP Topology

Screenshots of Figure 12-6 show the Spanning Tree topology on the lab network, matching the one highlighted in blue.

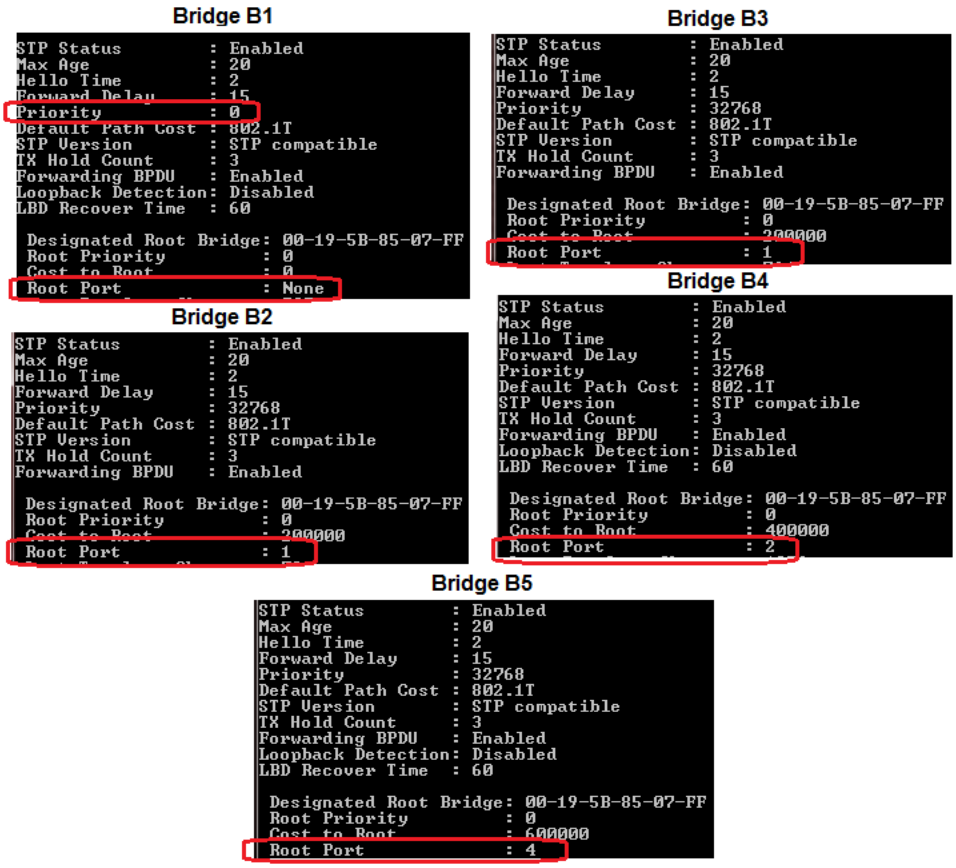


Figure 12-6 - Setup #1: STP Topology on D-Link switches

After the link failure between bridges B2 and B4, STP took about 30 seconds to converge into Figure 12-7 topology, causing big impact on communications from Node1 to Node5.

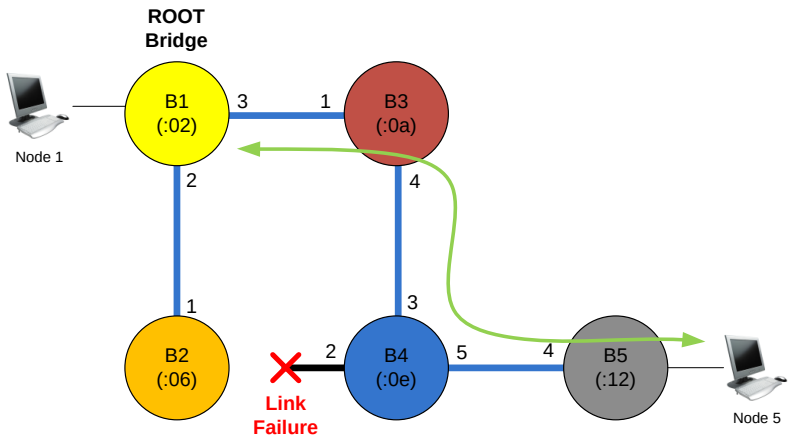


Figure 12-7 - Setup #1: STP recalculated topology

Figure 12-8 presents lab screenshots of B4 recalculating the STP topology, changing its root port from interface 2 to 3. Consequently, network suffered a 30-seconds downtime due to STP convergence, observed by the lack of ICMP packets with sequence numbers 39 towards 70.

Bridge B4

Before failure

```
STP Status          : Enabled
Max Age             : 20
Hello Time          : 2
Forward Delay       : 15
Priority            : 32768
Default Path Cost   : 802.1T
STP Version         : STP compatible
TX Hold Count       : 3
Forwarding BPDU     : Enabled
Loopback Detection: Disabled
LBD Recover Time    : 60

 Designated Root Bridge: 00-19-5B-85-07-FF
 Root Priority        : 0
 Cost to Root         : 400000
 Root Port            : 2
```

After failure

```
STP Status          : Enabled
Max Age             : 20
Hello Time          : 2
Forward Delay       : 15
Priority            : 32768
Default Path Cost   : 802.1T
STP Version         : STP compatible
TX Hold Count       : 3
Forwarding BPDU     : Enabled
Loopback Detection: Disabled
LBD Recover Time    : 60

 Designated Root Bridge: 00-19-5B-85-07-FF
 Root Priority        : 0
 Cost to Root         : 400000
 Root Port            : 3
```

Communication between Node1 and Node5

```
64 bytes from 10.10.10.1: icmp_seq=35 ttl=128 time=1.39 ms
64 bytes from 10.10.10.1: icmp_seq=36 ttl=128 time=1.71 ms
64 bytes from 10.10.10.1: icmp_seq=37 ttl=128 time=1.28 ms
64 bytes from 10.10.10.1: icmp_seq=38 ttl=128 time=1.60 ms
64 bytes from 10.10.10.1: icmp_seq=70 ttl=128 time=1.00 ms
64 bytes from 10.10.10.1: icmp_seq=71 ttl=128 time=1.07 ms
64 bytes from 10.10.10.1: icmp_seq=72 ttl=128 time=0.899 ms
64 bytes from 10.10.10.1: icmp_seq=73 ttl=128 time=1.21 ms
```

Figure 12-8 - Setup #1: STP convergence lab results

In SCS case, it is interesting to assess:

- Convergence times.
- TP and FT table changes on B4.
- DT table change on B3.

Figure 12-9 outputs the packet capture of ICMP traffic performed at node 1.

The link failure was simulated at t=100 seconds and the traffic loss is highlighted in black.

```
98.000000 IP 10.10.10.1 > 10.10.10.5: ICMP echo request, id 0, seq 28, length 64
98.024547 IP 10.10.10.5 > 10.10.10.1: ICMP echo reply, id 0, seq 28, length 64
99.000000 IP 10.10.10.1 > 10.10.10.5: ICMP echo request, id 0, seq 29, length 64
99.024150 IP 10.10.10.5 > 10.10.10.1: ICMP echo reply, id 0, seq 29, length 64
100.000000 IP 10.10.10.1 > 10.10.10.5: ICMP echo request, id 0, seq 30, length 64
101.000000 IP 10.10.10.1 > 10.10.10.5: ICMP echo request, id 0, seq 31, length 64
101.024289 IP 10.10.10.5 > 10.10.10.1: ICMP echo reply, id 0, seq 31, length 64
102.000000 IP 10.10.10.1 > 10.10.10.5: ICMP echo request, id 0, seq 32, length 64
102.024068 IP 10.10.10.5 > 10.10.10.1: ICMP echo reply, id 0, seq 32, length 64
103.000000 IP 10.10.10.1 > 10.10.10.5: ICMP echo request, id 0, seq 33, length 64
103.024578 IP 10.10.10.5 > 10.10.10.1: ICMP echo reply, id 0, seq 33, length 64
```

Figure 12-9 - Setup #1: SCS output

As Figure 12-9 outputs, only ICMP packet #30 was lost. SCS took only 1 second to forward traffic towards the alternative path B4-B3.

Please take into consideration that from bridge B4 to B1 there were two possible paths and this simulation forced traffic disruption. If, on the other hand, the traffic was flowing from B4 towards B1 using path via B3, the link failure B2-B4 would mean zero impact.

Prior to B2-B4 link failure, bridges B4 and B3 had the following table entries (only the relevant information is presented):

B4 TP Table		
Neighbour SCSID	**Interface**	**Metric**
B1	2	2
B1	3	2

B4 FT Table		
Remote Bridge SCSID	**outInterface**	**Delegate SCSID**
B1	2	B2

B3 DT Table		
Interface	**Source SCSID**	**Destination SCSID**
---	---	---

Table 16 - Bridges TP, FT and DT, before link issue

After link failure, the following changes were performed to allow B4 to reach B1:

B4 TP Table		
Neighbour SCSID	**Interface**	**Metric**
REMOVED: ~~B1~~	~~2~~	~~2~~
B1	3	2

B4 FT Table		
Remote Bridge SCSID	**outInterface**	**Delegate SCSID**
B1	2 → **3**	B2 → **B3**

B3 DT Table		
Interface	**Source SCSID**	**Destination SCSID**
1	**B1**	**B4**
4	**B4**	**B1**

Table 17 - Bridges TP, FT and DT, after link issue

Network diameter

IEEE recommends a maximum diameter of seven bridges, with default STP timers. That way, for large Layer 2 domains, STP is unsuitable. SCS, however, does not have such a limitation and even on ring topologies is quite appropriate delivering great performance.

STP incompatibility with larger network domains can be demonstrated using the *bizarre* network topology pictured in Figure 12-10.

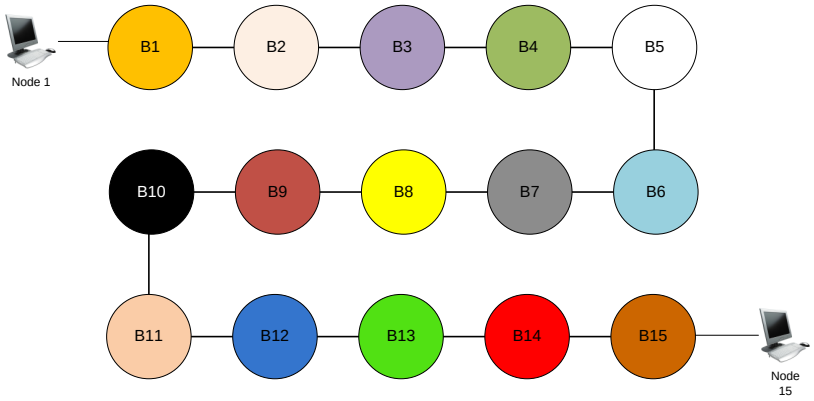


Figure 12-10 - Setup #2: network layout

Over large domains, STP has difficulties to maintain network stability. In the lab, nodes 1 (10.10.10.1) and 15 (10.10.10.15) were unable to communicate, like Figure 12-11 demonstrates.

```
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
From 10.10.10.15 icmp_seq=2 Destination Host Unreachable
From 10.10.10.15 icmp_seq=3 Destination Host Unreachable
From 10.10.10.15 icmp_seq=4 Destination Host Unreachable
From 10.10.10.15 icmp_seq=6 Destination Host Unreachable
From 10.10.10.15 icmp_seq=7 Destination Host Unreachable
From 10.10.10.15 icmp_seq=8 Destination Host Unreachable
From 10.10.10.15 icmp_seq=10 Destination Host Unreachable
From 10.10.10.15 icmp_seq=11 Destination Host Unreachable
From 10.10.10.15 icmp_seq=12 Destination Host Unreachable
From 10.10.10.15 icmp_seq=14 Destination Host Unreachable
From 10.10.10.15 icmp_seq=15 Destination Host Unreachable
```

Figure 12-11 - Setup #2: Node 15 cannot communicate with node 1

In ns3, the simulation revealed the SCS ease to deal with such large networks.

```
100.000000 ARP, Request who-has 10.10.10.15 (ff:ff:ff:ff:ff:ff) tell 10.10.10.1, length 50
100.070548 ARP, Reply 10.10.10.15 is-at 00:00:00:00:00:3b, length 50
100.070548 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 0, length 64
100.139584 ARP, Request who-has 10.10.10.1 (ff:ff:ff:ff:ff:ff) tell 10.10.10.15, length 50
100.139584 ARP, Reply 10.10.10.1 is-at 00:00:00:00:00:03, length 50
100.207836 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 0, length 64
101.000000 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 1, length 64
101.071851 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 1, length 64
102.000000 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 2, length 64
102.071941 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 2, length 64
103.000000 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 3, length 64
103.071627 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 3, length 64
104.000000 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 4, length 64
104.071789 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 4, length 64
105.000000 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 5, length 64
105.072224 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 5, length 64
106.000000 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 6, length 64
106.071914 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 6, length 64
```

Figure 12-12 - Setup #2: SCS simulation. Node 1 communicates with node 15

Examining TP and FT tables for bridge B1 (:02), Figure 12-13 shows the increasing metric for all neighbours in B1's TP table and, in FT, B1 relying in B2 (:06) to reach all other thirteen neighbours.

```
Node TP Table ------------
     for Bridge 02-06-00:00:00:00:00:02 @t=100 :
-----------------------------------------
Neighbor SCSID=00:00:00:00:00:06 / Inbound Interface=02-06-00:00:00:00:00:3d / Metric=1
Neighbor SCSID=00:00:00:00:00:0a / Inbound Interface=02-06-00:00:00:00:00:3d / Metric=2
Neighbor SCSID=00:00:00:00:00:0e / Inbound Interface=02-06-00:00:00:00:00:3d / Metric=3
Neighbor SCSID=00:00:00:00:00:12 / Inbound Interface=02-06-00:00:00:00:00:3d / Metric=4
Neighbor SCSID=00:00:00:00:00:16 / Inbound Interface=02-06-00:00:00:00:00:3d / Metric=5
Neighbor SCSID=00:00:00:00:00:1a / Inbound Interface=02-06-00:00:00:00:00:3d / Metric=6
Neighbor SCSID=00:00:00:00:00:1e / Inbound Interface=02-06-00:00:00:00:00:3d / Metric=7
Neighbor SCSID=00:00:00:00:00:22 / Inbound Interface=02-06-00:00:00:00:00:3d / Metric=8
Neighbor SCSID=00:00:00:00:00:26 / Inbound Interface=02-06-00:00:00:00:00:3d / Metric=9
Neighbor SCSID=00:00:00:00:00:2a / Inbound Interface=02-06-00:00:00:00:00:3d / Metric=10
Neighbor SCSID=00:00:00:00:00:2e / Inbound Interface=02-06-00:00:00:00:00:3d / Metric=11
Neighbor SCSID=00:00:00:00:00:32 / Inbound Interface=02-06-00:00:00:00:00:3d / Metric=12
Neighbor SCSID=00:00:00:00:00:36 / Inbound Interface=02-06-00:00:00:00:00:3d / Metric=13
Neighbor SCSID=00:00:00:00:00:3a / Inbound Interface=02-06-00:00:00:00:00:3d / Metric=14

Bridge Flood Table ------------
     for Bridge 02-06-00:00:00:00:00:02 @t=100 :
-----------------------------------------
Bridge SCSID = 00:00:00:00:00:0a / Interface =02-06-00:00:00:00:00:3d / Delegate to Bridge =00:00:00:00:00:06
Bridge SCSID = 00:00:00:00:00:0e / Interface =02-06-00:00:00:00:00:3d / Delegate to Bridge =00:00:00:00:00:06
Bridge SCSID = 00:00:00:00:00:12 / Interface =02-06-00:00:00:00:00:3d / Delegate to Bridge =00:00:00:00:00:06
Bridge SCSID = 00:00:00:00:00:16 / Interface =02-06-00:00:00:00:00:3d / Delegate to Bridge =00:00:00:00:00:06
Bridge SCSID = 00:00:00:00:00:1a / Interface =02-06-00:00:00:00:00:3d / Delegate to Bridge =00:00:00:00:00:06
Bridge SCSID = 00:00:00:00:00:1e / Interface =02-06-00:00:00:00:00:3d / Delegate to Bridge =00:00:00:00:00:06
Bridge SCSID = 00:00:00:00:00:22 / Interface =02-06-00:00:00:00:00:3d / Delegate to Bridge =00:00:00:00:00:06
Bridge SCSID = 00:00:00:00:00:26 / Interface =02-06-00:00:00:00:00:3d / Delegate to Bridge =00:00:00:00:00:06
Bridge SCSID = 00:00:00:00:00:2a / Interface =02-06-00:00:00:00:00:3d / Delegate to Bridge =00:00:00:00:00:06
Bridge SCSID = 00:00:00:00:00:2e / Interface =02-06-00:00:00:00:00:3d / Delegate to Bridge =00:00:00:00:00:06
Bridge SCSID = 00:00:00:00:00:32 / Interface =02-06-00:00:00:00:00:3d / Delegate to Bridge =00:00:00:00:00:06
Bridge SCSID = 00:00:00:00:00:36 / Interface =02-06-00:00:00:00:00:3d / Delegate to Bridge =00:00:00:00:00:06
Bridge SCSID = 00:00:00:00:00:3a / Interface =02-06-00:00:00:00:00:3d / Delegate to Bridge =00:00:00:00:00:06
```

Figure 12-13 - Setup #2: SCS B1 TP and FT table

Least cost path

STP least cost paths calculations are performed taking into consideration the communication towards the Root Bridge. This frequently creates on STP networks sub-optimal paths between non-root bridges, which cause over utilization of some link's bandwidth and bridge's switching engine. Besides non optimal paths, STP does not provide traffic load balancing.

SCS always creates optimal paths based on link's bandwidth, delivers traffic load balance between parallel links and provides redundant equal-cost paths between bridges. This is a great leap over STP features.

Suppose nodes communicating like Figure 12-14. The shortest path between node 5 and 3 is the one presented in green.

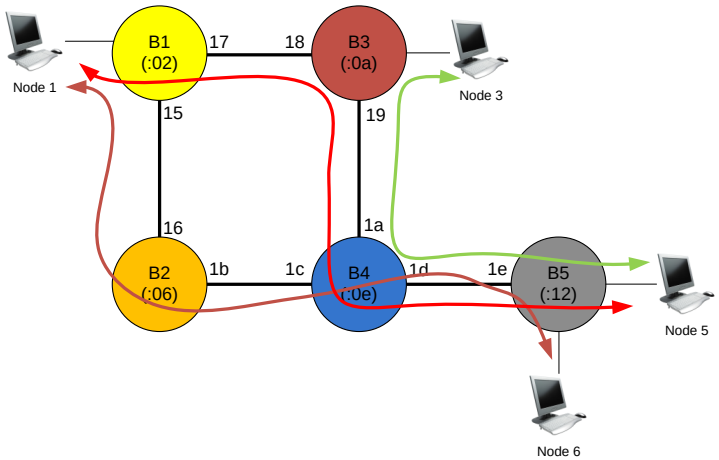


Figure 12-14 – Setup #3: Communication between nodes 1, 3, 5 and 6

For communications between nodes attached to bridge B5 and B1, there are two equal cost paths, red and purple, via B3 and B2, respectively.

SCS protocol provides Figure 12-14 connectivity, as designed and desired.

Running STP, however, no load balance would be possible via bridges 2/3, neither communication between node 5 and 3 follows the least cost path between bridges 5 and 3. Figure 12-15 illustrates traffic flow with STP.

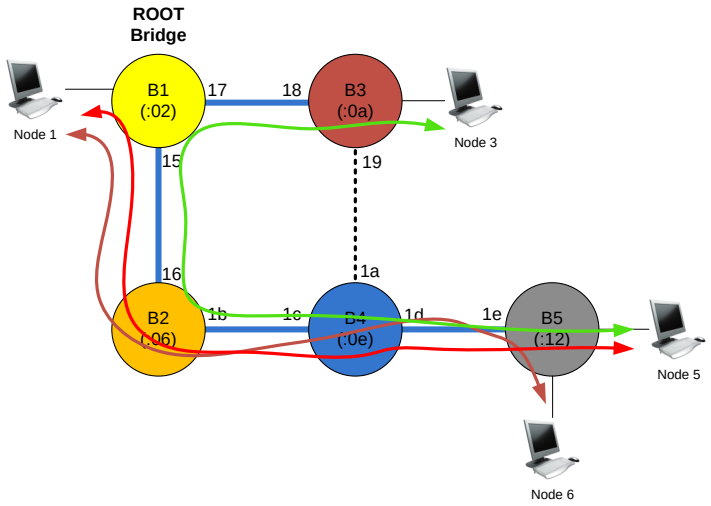


Figure 12-15 - Setup #3: Traffic flowing via STP topology

Figure 12-14 and Figure 12-15 illustrate, undoubtedly, SCS superiority over STP in terms of traffic distribution in redundant networks, via least cost paths.

Topology Change Mechanism

In some circumstances, even a host port flapping can force the STP network to be permanently in topology change status, causing the forwarding database's aging timers to be reduced to 15 seconds over a period of 35 seconds, by default. This could mean massive flooding in the network.

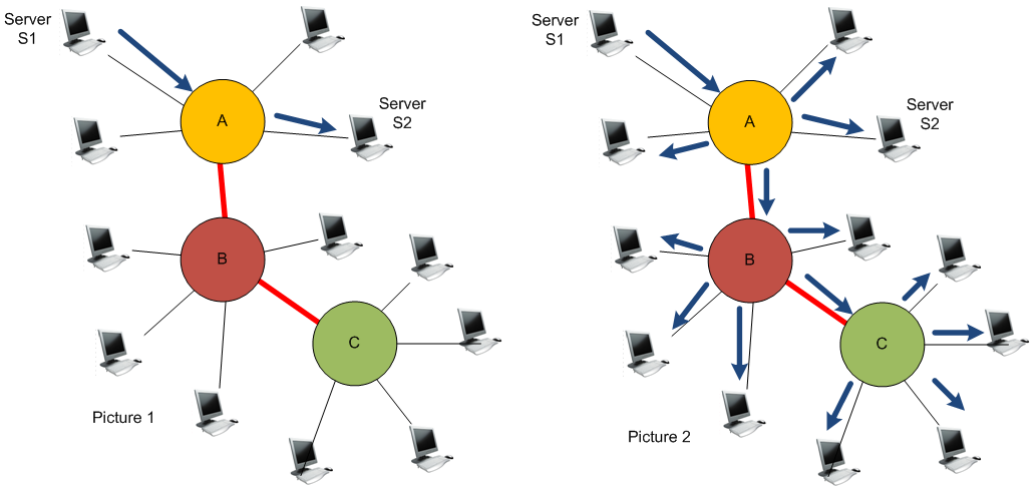


Figure 12-16 – Topology Change state issue

Topology changes on SCS networks only occur whenever a real topology change exists, meaning a bridge interface transiting from UP to DOWN or delayUP to UP. In such occurrences, there is no other way around and bridge's FwdT must be cleaned to speed up convergence times. Thus, traffic flooding will also occur in SCS networks, however, efficiency is achieved as only real topology changes trigger topology recalculations.

Figure 12-17 presents the lab setup that demonstrates traffic flood in the STP network, just because node 3 was connected to B3. Node 1 (10.10.10.1) is communicating with node 5 (10.10.10.15) and node's 3 link changed to up, causing bridge 3 to generate TCNs.

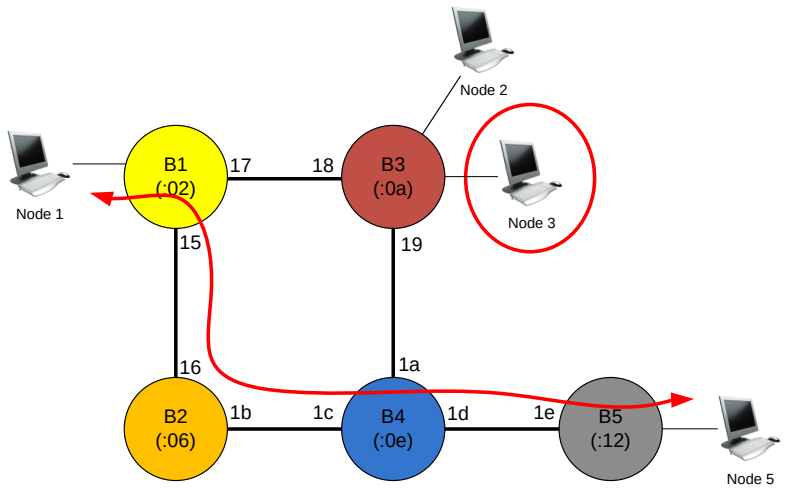


Figure 12-17 - Setup #4: Lab setup

This will cause all STP bridges to flood traffic throughout the network and Node 2 will start receiving unnecessary traffic, like Figure 12-18 shows and Figure 12-19 confirms.

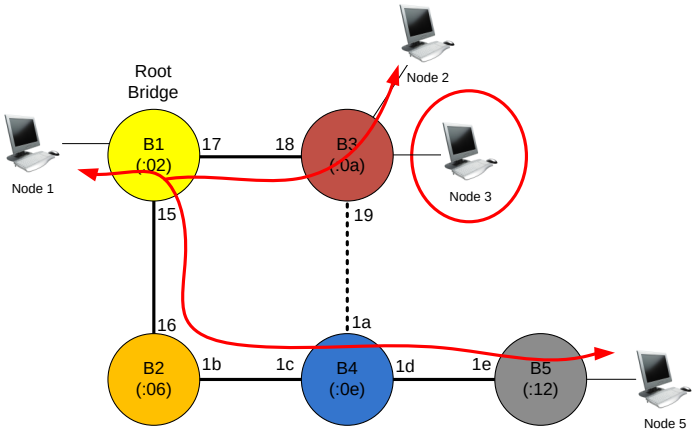


Figure 12-18 - Setup #4: STP network traffic flooding

Figure 12-19 shows a packet capture performed on node 2. First, we see the periodic BPDU transmission, in frames #12 towards #21, each 2 seconds. Until this point, traffic from node 1 to node 5 is unicasted, as supposed. However, at frame #22,

a BPDU signalling a topology change (TC bit set) is received and the traffic flood begins. In frame #23, unicast traffic from node 1 towards node 5 was captured by node 2, confirming the flooding.

No.	Time	Source	Destination	Protocol	Info
12	21.999679	00:1c:f0:a8:bd:2a	01:80:c2:00:00:00	STP	Conf. Root = 0/0/00:19:5b:85:07:ff Cost = 200000 Port = 0x8008
13	23.999760	00:1c:f0:a8:bd:2a	01:80:c2:00:00:00	STP	Conf. Root = 0/0/00:19:5b:85:07:ff Cost = 200000 Port = 0x8008
14	25.999843	00:1c:f0:a8:bd:2a	01:80:c2:00:00:00	STP	Conf. Root = 0/0/00:19:5b:85:07:ff Cost = 200000 Port = 0x8008
15	27.999678	00:1c:f0:a8:bd:2a	01:80:c2:00:00:00	STP	Conf. Root = 0/0/00:19:5b:85:07:ff Cost = 200000 Port = 0x8008
16	29.999761	00:1c:f0:a8:bd:2a	01:80:c2:00:00:00	STP	Conf. Root = 0/0/00:19:5b:85:07:ff Cost = 200000 Port = 0x8008
17	31.999596	00:1c:f0:a8:bd:2a	01:80:c2:00:00:00	STP	Conf. Root = 0/0/00:19:5b:85:07:ff Cost = 200000 Port = 0x8008
18	33.999681	00:1c:f0:a8:bd:2a	01:80:c2:00:00:00	STP	Conf. Root = 0/0/00:19:5b:85:07:ff Cost = 200000 Port = 0x8008
19	35.999512	00:1c:f0:a8:bd:2a	01:80:c2:00:00:00	STP	Conf. Root = 0/0/00:19:5b:85:07:ff Cost = 200000 Port = 0x8008
20	37.999599	00:1c:f0:a8:bd:2a	01:80:c2:00:00:00	STP	Conf. Root = 0/0/00:19:5b:85:07:ff Cost = 200000 Port = 0x8008
21	39.999683	00:1c:f0:a8:bd:2a	01:80:c2:00:00:00	STP	Conf. Root = 0/0/00:19:5b:85:07:ff Cost = 200000 Port = 0x8008
22	42.009513	00:1c:f0:a8:bd:2a	01:80:c2:00:00:00	STP	Conf. TC + Root = 0/0/00:19:5b:85:07:ff Cost = 200000 Port = 0x8008
23	43.235789	10.10.10.1	10.10.10.15	ICMP	Echo (ping) request id=0x0001, seq=1605/17670, ttl=128
24	44.009600	00:1c:f0:a8:bd:2a	01:80:c2:00:00:00	STP	Conf. TC + Root = 0/0/00:19:5b:85:07:ff Cost = 200000 Port = 0x8008
25	46.009429	00:1c:f0:a8:bd:2a	01:80:c2:00:00:00	STP	Conf. TC + Root = 0/0/00:19:5b:85:07:ff Cost = 200000 Port = 0x8008
26	48.009513	00:1c:f0:a8:bd:2a	01:80:c2:00:00:00	STP	Conf. TC + Root = 0/0/00:19:5b:85:07:ff Cost = 200000 Port = 0x8008
27	50.009346	00:1c:f0:a8:bd:2a	01:80:c2:00:00:00	STP	Conf. TC + Root = 0/0/00:19:5b:85:07:ff Cost = 200000 Port = 0x8008
28	52.009431	00:1c:f0:a8:bd:2a	01:80:c2:00:00:00	STP	Conf. TC + Root = 0/0/00:19:5b:85:07:ff Cost = 200000 Port = 0x8008

Figure 12-19 - Setup #4: Node 2 traffic capture

In the SCS protocol, host interfaces do not contribute to network topology. That way, node 3 link change is irrelevant for SCS network stability.

SCS independence over peripheral network events is another advantage of SCS over STP.

Convergence Black Hole

It is easy to demonstrate that a link *failure and restore* in STP creates a big impact on network connectivity. Figure 12-20 illustrates this problem.

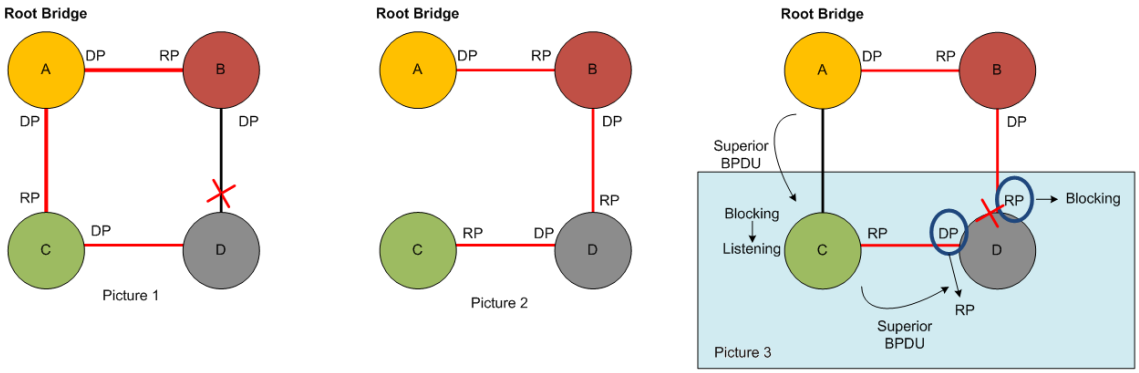


Figure 12-20 – Fast convergence creates black hole

The lab setup for this demonstration is pictured in Figure 12-21, where link B1-B2 is the one that will face problems.

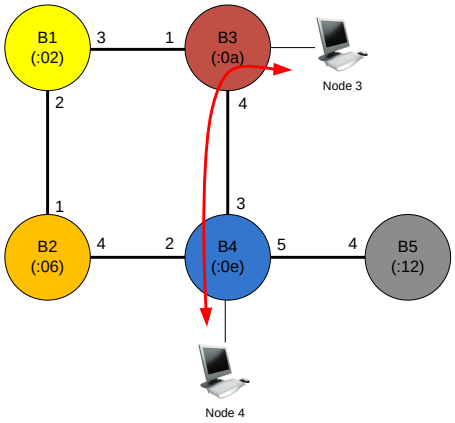


Figure 12-21 - Setup #5: Network topology

For SCS is quite simple: link B1-B2 failure is harmless for connectivity between node 3 (10.10.10.3) and node 4 (10.10.10.4), as SCS chooses the least cost path for communication. Simulating link failure at t=100seconds, Figure 12-22 shows the unaffected connectivity of SCS.

```
97.000000 IP 10.10.10.3 > 10.10.10.4: ICMP echo request, id 0, seq 27, length 64
97.015717 IP 10.10.10.4 > 10.10.10.3: ICMP echo reply, id 0, seq 27, length 64
98.000000 IP 10.10.10.3 > 10.10.10.4: ICMP echo request, id 0, seq 28, length 64
98.015972 IP 10.10.10.4 > 10.10.10.3: ICMP echo reply, id 0, seq 28, length 64
99.000000 IP 10.10.10.3 > 10.10.10.4: ICMP echo request, id 0, seq 29, length 64
99.015891 IP 10.10.10.4 > 10.10.10.3: ICMP echo reply, id 0, seq 29, length 64
100.000000 IP 10.10.10.3 > 10.10.10.4: ICMP echo request, id 0, seq 30, length 64
100.015574 IP 10.10.10.4 > 10.10.10.3: ICMP echo reply, id 0, seq 30, length 64
101.000000 IP 10.10.10.3 > 10.10.10.4: ICMP echo request, id 0, seq 31, length 64
101.015785 IP 10.10.10.4 > 10.10.10.3: ICMP echo reply, id 0, seq 31, length 64
102.000000 IP 10.10.10.3 > 10.10.10.4: ICMP echo request, id 0, seq 32, length 64
102.056125 IP 10.10.10.4 > 10.10.10.3: ICMP echo reply, id 0, seq 32, length 64
103.000000 IP 10.10.10.3 > 10.10.10.4: ICMP echo request, id 0, seq 33, length 64
103.015957 IP 10.10.10.4 > 10.10.10.3: ICMP echo reply, id 0, seq 33, length 64
104.000000 IP 10.10.10.3 > 10.10.10.4: ICMP echo request, id 0, seq 34, length 64
104.016369 IP 10.10.10.4 > 10.10.10.3: ICMP echo reply, id 0, seq 34, length 64
105.000000 IP 10.10.10.3 > 10.10.10.4: ICMP echo request, id 0, seq 35, length 64
105.016492 IP 10.10.10.4 > 10.10.10.3: ICMP echo reply, id 0, seq 35, length 64
```

Figure 12-22 - Setup #5: SCS response times

As STP is concerned, the results are far away from good! The active topology is illustrated in Figure 12-23.

Link B1-B2 failure has impact on the connectivity between node 3 (10.10.10.3) and node 4 (10.10.10.4). Furthermore, B2, B4 and B5 will be isolated from the remaining network for about 30 seconds.

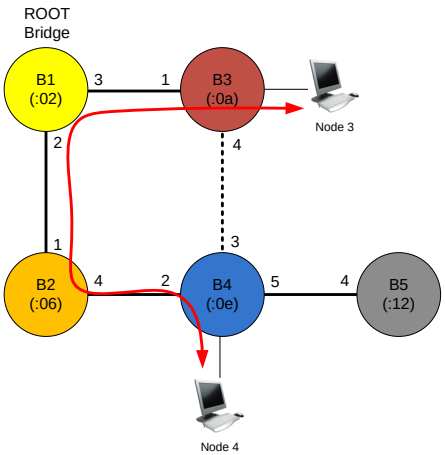


Figure 12-23 - Setup #5: STP active topology

Figure 12-24 presents the topology screenshots before any link failure.

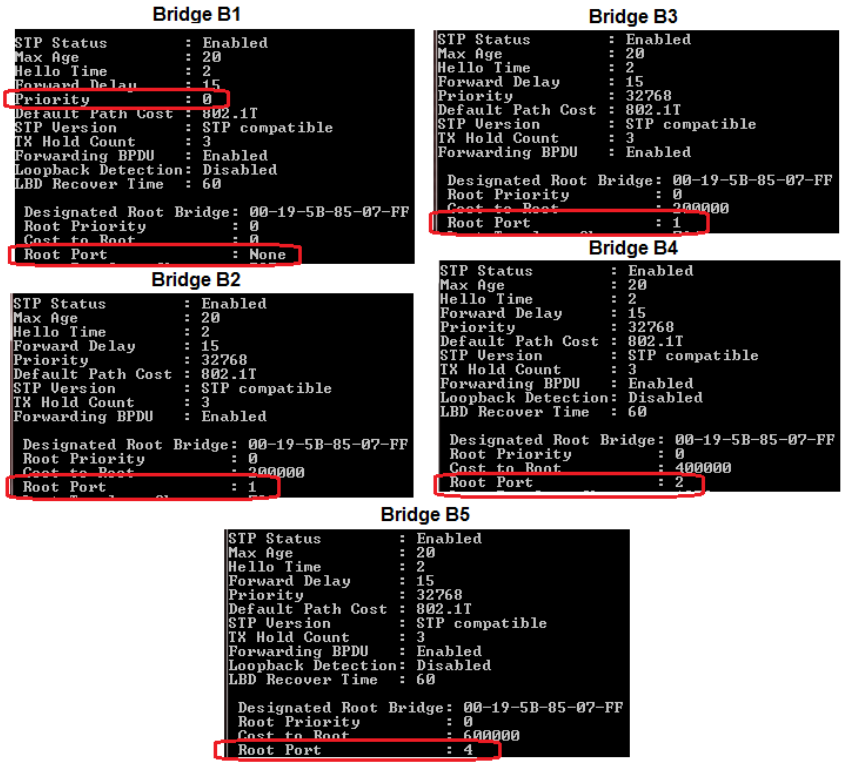


Figure 12-24 - Setup #5: STP active topology screenshots

After link B1-B2 failing, bridges B2 and B4 changed their root ports to 4 and 3, respectively, like Figure 12-25 shows.

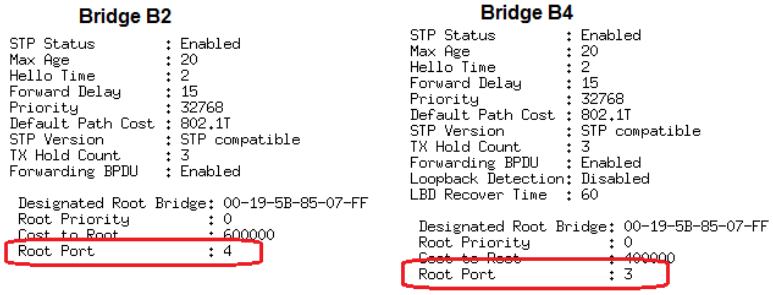


Figure 12-25 - setup #5: B2 and B4 STP root ports

B1-B2 is restored @ *icmp_seq=40*. Bridge B2 converges very quickly and 3 seconds after, B2, B4 and B5 get isolated from the network during 30 seconds. Figure 12-26 demonstrates the 30-seconds downtime in communications between node 3 and 4.

```
64 bytes from 10.10.10.3: icmp_seq=39 ttl=128 time=1.54 ms
64 bytes from 10.10.10.3: icmp_seq=40 ttl=128 time=1.42 ms
64 bytes from 10.10.10.3: icmp_seq=41 ttl=128 time=1.42 ms
64 bytes from 10.10.10.3: icmp_seq=42 ttl=128 time=1.24 ms
64 bytes from 10.10.10.3: icmp_seq=43 ttl=128 time=1.06 ms
64 bytes from 10.10.10.3: icmp_seq=74 ttl=128 time=1.54 ms
64 bytes from 10.10.10.3: icmp_seq=75 ttl=128 time=1.27 ms
64 bytes from 10.10.10.3: icmp_seq=76 ttl=128 time=1.34 ms
64 bytes from 10.10.10.3: icmp_seq=77 ttl=128 time=0.669 ms
64 bytes from 10.10.10.3: icmp_seq=78 ttl=128 time=1.23 ms
```

Figure 12-26 - Setup#5: ICMP between n3 and 4, after link B1-B2 restoration

SCS is immune to such events. Some link failures can indeed induce connectivity downtimes of about 3 seconds, in worst case scenarios. However, link restore never creates such impact, as SCS calculations are performed in parallel with traffic forwarding, allowing multiple redundant paths to be used at the same time.

Above section *Conservative STP timers* already presented the worst case scenario for SCS, with 3-seconds downtime, maximum.

As seen in all the above situations, STP misbehaves and misadjusts under a very wide range of situations, presenting convergence times completely inadequate for modern Layer 2 networks. On the other hand, SCS completely mitigates all those potential issues and significantly improves networking performance, convergence times and resource usage, creating always optimal and redundant paths, over all possible network architectures.

12.3 Rapid Spanning Tree Problems

One huge handicap of RSTP is *count to infinity* behaviour. Additionally, RSTP interaction with STP and *port role negotiation* are major contributors to slower convergence times, worsening RSTP performance.

Count to infinity

RSTP frequently exhibits the classic *count to infinity* behaviour, as *stale* BPDUs for a failed Root Bridge might persist in the network, flowing around *poisoning* the network, while the new information chases it. Figure 12-27, Figure 12-28 and Figure 12-29, recall the issue.

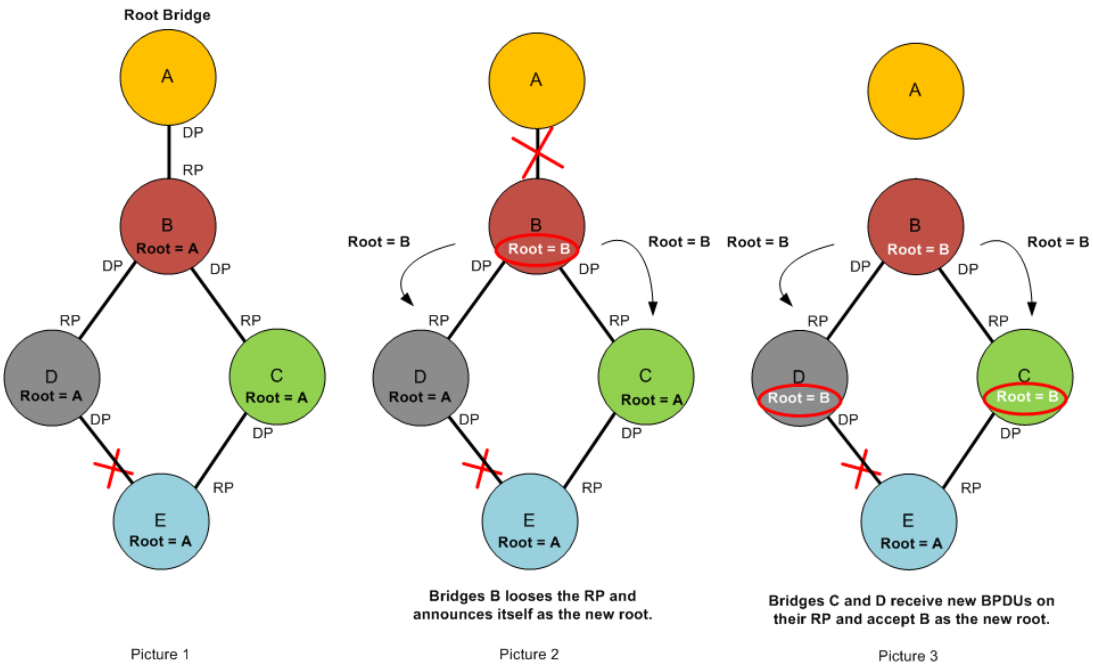


Figure 12-27 – C*ount to infinity*: new root advertisement

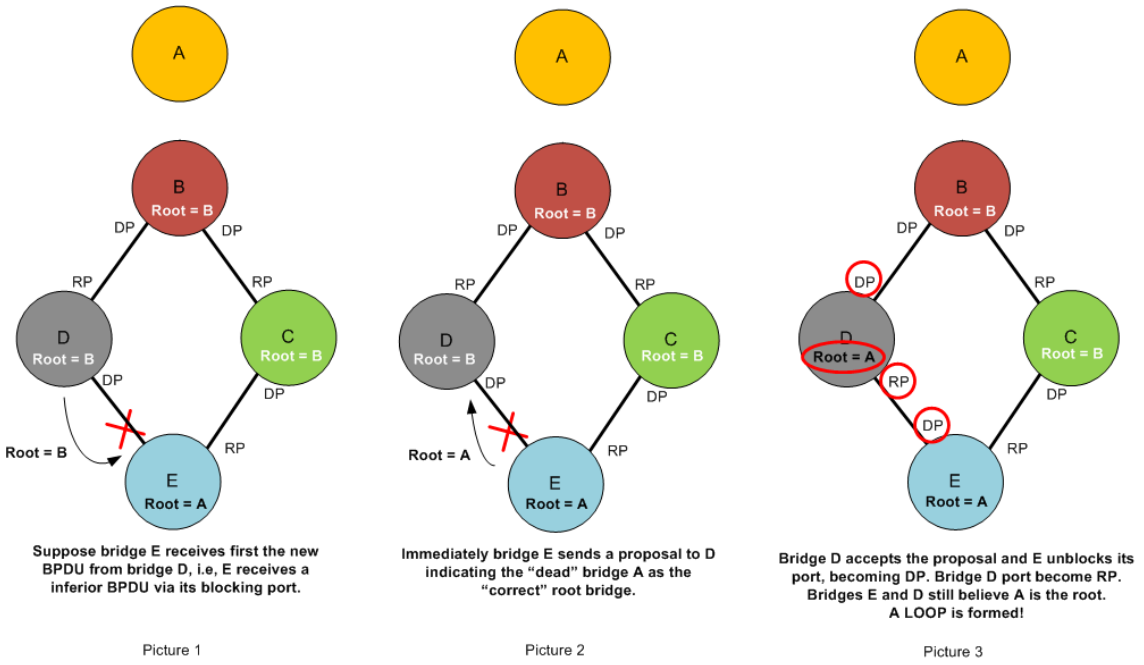


Figure 12-28 – Count to infinity: creating the loop

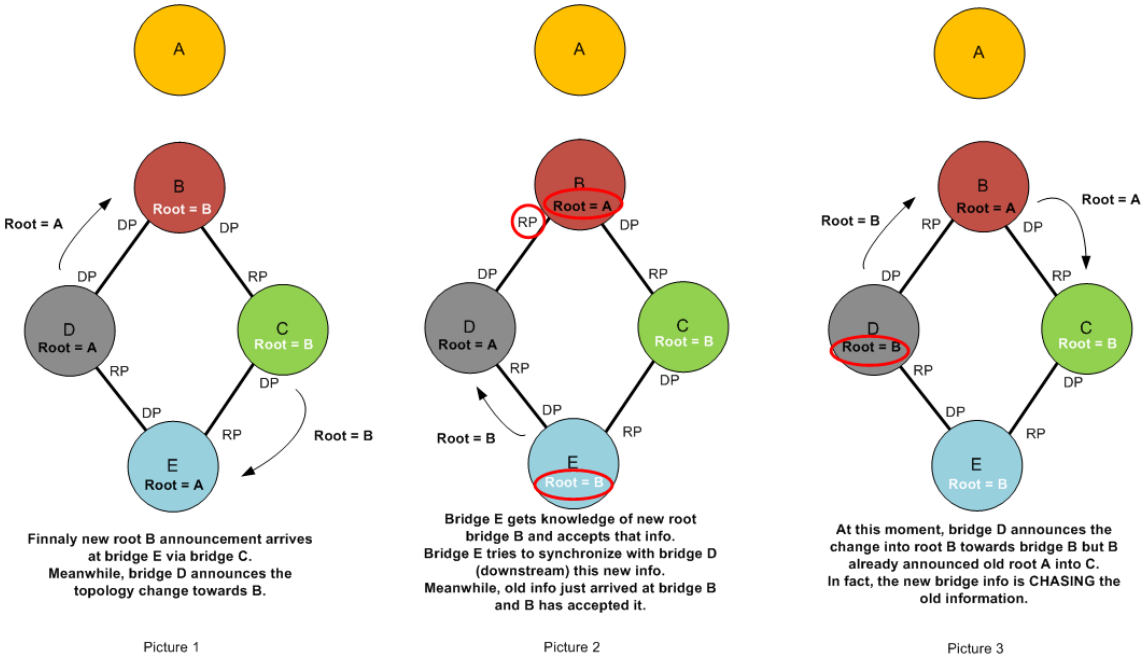


Figure 12-29 – Count to infinity: new information chases old information

SCS protocol does not exhibit the classic *count to infinity* behaviour. In the above scenario, bridge B (B4 on ns3 simulation) would announce the loss of neighborship with A (B5@ns3), triggering a SCS Update packet relative to bridge A with *clearFlag* set to '1', and sending it to bridges C (B2@ns3) and D (B3@ns3). Then, that info reaches E (B1@ns3). In sum, bridge A became isolated from the network, and all SCS bridges were notified about that fact and updated their tables accordingly.

Following is part of the output for the simulation of this particular link failure, using test bed #1 pictured on Figure 12-1 to simulate the SCS behaviour, where the above A to E bridges are simulated in ns3 as B5, B4, B3, B2 and B1 bridges, respectively.

```
(…)
t=102.002 : Bridge B4 removed from the NBTable the neighbor 00:00:00:00:00:12
                due to deadTimerExpired. NB Table Change => RECALCULATING TP!
t=102.002 : Bridge B4 sent an SCS TP Update Packet to B2 about B5 with clear flag=1
t=102.002 : Bridge B4 sent an SCS TP Update Packet to B3 about B5 with clear flag=1
t=102.002 : Bridge B4 removed the TP Table entry relative to B5
(…)
t=102.004 : Bridge B2 is processing the SCS Update received from B4 relative to B5
                with Clear Flag = 1 and metric=2
t=102.004 : Bridge B2 sent an SCS TP Update Packet to B1 regarding B5 with clear flag=1
t=102.004 : Bridge B2 removed the TP Table entry relative to B5
(…)
t=102.007 : Bridge B1 is processing the SCS Update received from B2 relative to B5
                with Clear Flag = 1 and metric=3
t=102.007 : Bridge B1 removed the TP Table entry relative to B5
(…)
```

Figure 12-30 - Setup #6: SCS simulation output

Even frequently exhibited by RSTP, the *count to infinity* behaviour is hard to simulate in lab. This could be fairly extrapolated to SCS simulations in ns-3. However, in SCS, if by some means "old" information poisons the network, the overall effect would be phantom entries created on SCS tables, leading to a non-existent bridge, whereas in RSTP, major network disruption occurs.

Port role negotiation

RSTP negotiates each port role transition, in order to guarantee a loop-free topology. In a ring topology, for example, with a certain number of bridges in the ring, if one of the two Root Bridge's ports fails, the negotiation overhead delays considerably the network convergence time to several seconds. Some studies advance that after 10 bridges, the convergence time goes above 3 seconds [2].

In a presence of a Root Bridge's link failure, all the bridge ports of half ring will have to change their role from *root port* into *designated port* and from *designated port* into *root port*. Figure 12-31 illustrates this situation.

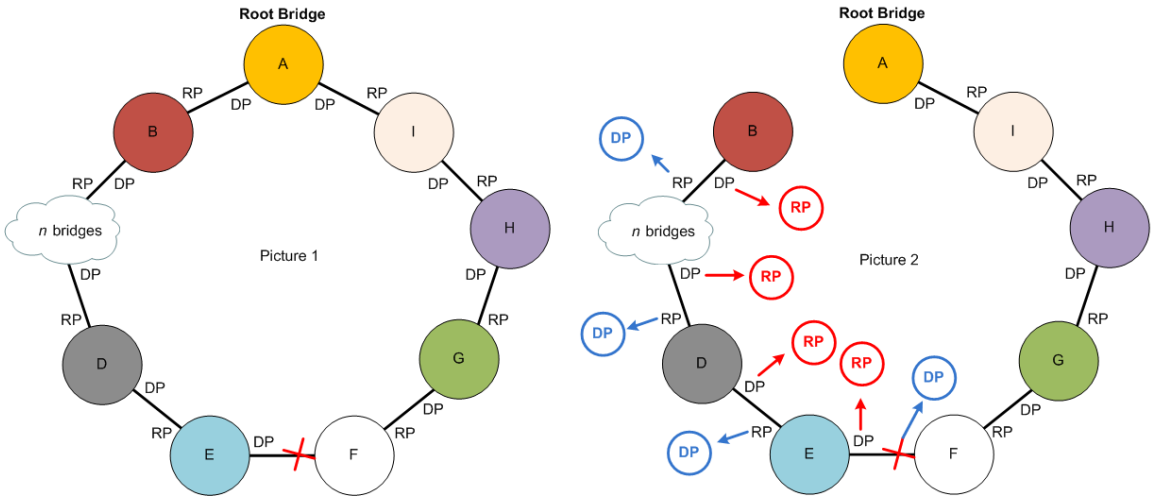


Figure 12-31 - RSTP converging in ring topologies

Usually, the negotiation between two bridges is quite fast, but during periods of convergence when several bridges send simultaneously BPDUs updating root path costs, RSTP restricts BPDU send rate to one per port per second. This BPDU transmission rate can add seconds to convergence total time [6].

The STP lab followed test bed #2, connecting the two edge bridges B1 and B15, forming a ring topology, like Figure 12-32 illustrates.

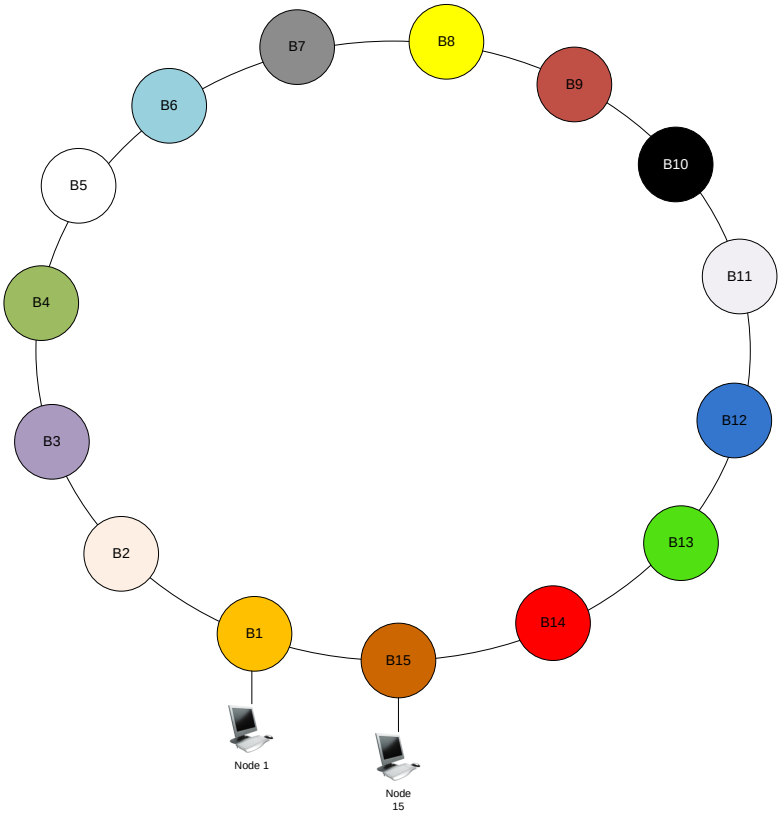


Figure 12-32 - Setup #7: Ring topology lab setup

Lab tests demonstrated the weak performance RSTP has in such ring topologies. To assess RSTP behaviour, a link failure was induced between bridges B1 and B15. The root bridge was B1 and node 1 and node 15 were communicating.

In such circumstances, being B1 the root bridge, the traffic flow between nodes 1 and 15 was performed via link B1-B15. Creating a failure in such connection, would force RSTP to renegotiate bridge's port roles, as Figure 12-31 pictures.

Figure 12-33 shows the downtime observed on traffic from node 15 to node 1.

```
64 bytes from 10.10.10.1: icmp_seq=25 ttl=128 time=0.694 ms
64 bytes from 10.10.10.1: icmp_seq=26 ttl=128 time=0.761 ms
64 bytes from 10.10.10.1: icmp_seq=27 ttl=128 time=0.831 ms
64 bytes from 10.10.10.1: icmp_seq=28 ttl=128 time=0.648 ms
64 bytes from 10.10.10.1: icmp_seq=29 ttl=128 time=0.717 ms
64 bytes from 10.10.10.1: icmp_seq=30 ttl=128 time=0.787 ms
64 bytes from 10.10.10.1: icmp_seq=31 ttl=128 time=0.853 ms
64 bytes from 10.10.10.1: icmp_seq=32 ttl=128 time=0.422 ms
64 bytes from 10.10.10.1: icmp_seq=63 ttl=128 time=204 ms
64 bytes from 10.10.10.1: icmp_seq=64 ttl=128 time=0.714 ms
64 bytes from 10.10.10.1: icmp_seq=65 ttl=128 time=0.771 ms
64 bytes from 10.10.10.1: icmp_seq=66 ttl=128 time=0.838 ms
```

Figure 12-33 - Setup #7: RSTP convergence time

When B1-B15 fails, the RSTP network has to change 14 port roles and a downtime of near 30 seconds was observed, confirming the above theory of BPDU transmission rate restriction and RSTP inadequacy for such network topologies.

SCS protocol, once more, performs superiorly over RSTP. The simulation performed on ns3 for the same network topology, demonstrates SCS response time of near 1 second. Figure 12-34 and Figure 12-35 shows SCS results on ns3 simulation, with B1-B15 failure induced at t=200 seconds.

```
198.000000 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 98, length 64
198.015546 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 98, length 64
199.000000 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 99, length 64
199.015836 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 99, length 64
200.000000 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 100, length 64
201.000000 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 101, length 64
201.071748 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 101, length 64
202.000000 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 102, length 64
202.071520 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 102, length 64
203.000000 IP 10.10.10.1 > 10.10.10.15: ICMP echo request, id 0, seq 103, length 64
203.071965 IP 10.10.10.15 > 10.10.10.1: ICMP echo reply, id 0, seq 103, length 64
```

Figure 12-34 - Setup #7: Node 1 packet capture

```
t=200 : Bridge B1 interface 00:00:00:00:00:59 is DOWN!
t=200 : Bridge B1 changed the NB State of interface 00:00:00:00:00:59 to DOWN!
t=200 : Bridge B1 removed from the NBTable the neighbor B15 due to link failure.
     NB Table Change => RECALCULATING TP!

t=200 : Bridge B1 removed the TP Table entry relative to B15
t=200 : Bridge B1 removed the TP Table entry relative to B14
t=200 : Bridge B1 removed the TP Table entry relative to B13
t=200 : Bridge B1 removed the TP Table entry relative to B12
t=200 : Bridge B1 removed the TP Table entry relative to B11
t=200 : Bridge B1 removed the TP Table entry relative to B10
t=200 : Bridge B1 removed the TP Table entry relative to B9

t=200 : Bridge B1 sent an SCS TP Update Packet to destination B2
     about entry related to B15 with clear flag=1
t=200 : Bridge B1 sent an SCS TP Update Packet to destination B2
     about entry related to B14 with clear flag=1
t=200 : Bridge B1 sent an SCS TP Update Packet to destination B2
     about entry related to B13 with clear flag=1
t=200 : Bridge B1 sent an SCS TP Update Packet to destination B2
     about entry related to B12 with clear flag=1
t=200 : Bridge B1 sent an SCS TP Update Packet to destination B2
     about entry related to B11 with clear flag=1
t=200 : Bridge B1 sent an SCS TP Update Packet to destination B2
     about entry related to B10 with clear flag=1
t=200 : Bridge B1 sent an SCS TP Update Packet to destination B2
     about entry related to B9 with clear flag=1
```

Figure 12-35 - Setup #7: Ns-3 script output

Figure 12-34 shows just one ICMP echo reply missing, at t=200.

Figure 12-35 illustrates the debug information provided by SCS. At t=200 bridge B1 declares link B1-B15 down, it removes from the TP table the seven affected entries and advertises that change towards bridge B2. Half-a-second after, bridge B1 has its TP table updated accordingly, like Figure 12-36 illustrates.

```
Node TP Table --------------------------------
     for Bridge 00:00:00:00:00:02 @t=200.522 :
----------------------------------------------
Neighbor SCSID=B2 / Inbound Interface=00:00:00:00:00:3d / Metric=1
Neighbor SCSID=B3 / Inbound Interface=00:00:00:00:00:3d / Metric=2
Neighbor SCSID=B4 / Inbound Interface=00:00:00:00:00:3d / Metric=3
Neighbor SCSID=B5 / Inbound Interface=00:00:00:00:00:3d / Metric=4
Neighbor SCSID=B6 / Inbound Interface=00:00:00:00:00:3d / Metric=5
Neighbor SCSID=B7 / Inbound Interface=00:00:00:00:00:3d / Metric=6
Neighbor SCSID=B8 / Inbound Interface=00:00:00:00:00:3d / Metric=7
Neighbor SCSID=B9 / Inbound Interface=00:00:00:00:00:3d / Metric=8
Neighbor SCSID=B10 / Inbound Interface=00:00:00:00:00:3d / Metric=9
Neighbor SCSID=B11 / Inbound Interface=00:00:00:00:00:3d / Metric=10
Neighbor SCSID=B12 / Inbound Interface=00:00:00:00:00:3d / Metric=11
Neighbor SCSID=B13 / Inbound Interface=00:00:00:00:00:3d / Metric=12
Neighbor SCSID=B14 / Inbound Interface=00:00:00:00:00:3d / Metric=13
Neighbor SCSID=B15 / Inbound Interface=00:00:00:00:00:3d / Metric=14
----------------------------------------------------------------------
```

Figure 12-36 - Setup #7: SCS Bridge B1 TP Table

Figure 12-37 shows the SCS Update messages exchanged, for bridge B1 side. For bridge 15 side, the similar exchanged occurred, but in the other half of the ring.

Counter-clockwise black arrows mean immediate responses from a neighbour bridge, whereas the red ones mean the remaining paths a bridge advertises downstream, after learning them from the upstream neighbour.

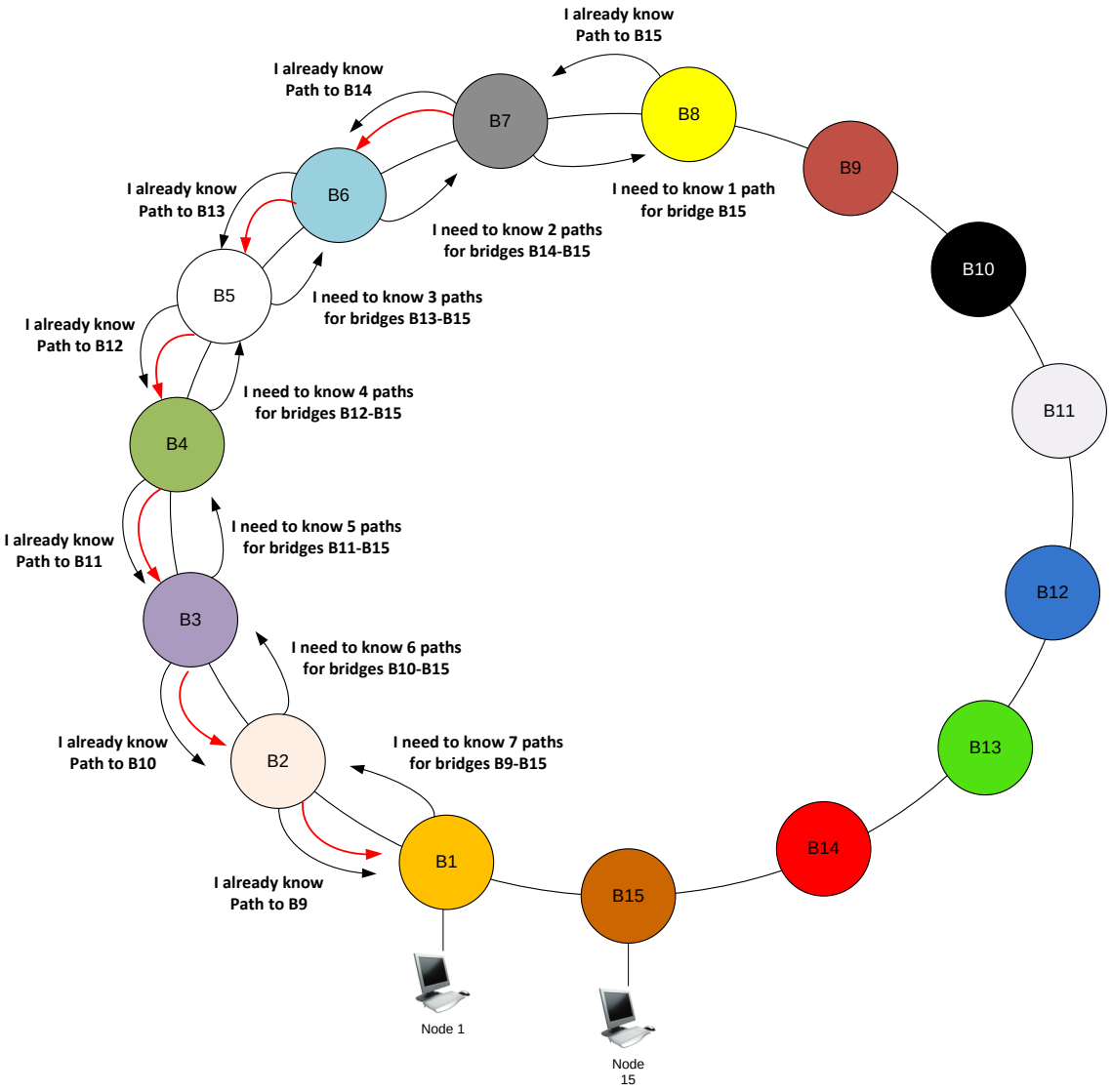


Figure 12-37 - Setup #7: SCS messages enchanged

After losing connectivity with bridge B15, bridge B1 needs to know how to reach seven paths, previously achieved via bridge B15. That way, bridge B1 asks B2 for those seven paths. Bridge B2 already knows the path to bridge B9, as B2-B9 path was shorter via B3 than previously via B1, so B2 immediately advertises B9 to B1. This message propagation continues in clockwise direction, decrementing the numbers of paths need to be known by the requester bridge.

RSTP Protocol Degradation

RSTP allows STP bridges to operate in a RSTP network. Anytime a bridge port detect legacy STP 802.1D bridges on the network, all RTSP interfaces are reverted back to 801.D STP specifications in order to avoid having bridges in the network that doesn't have the correct network information.

The lab setup for this section is the following:

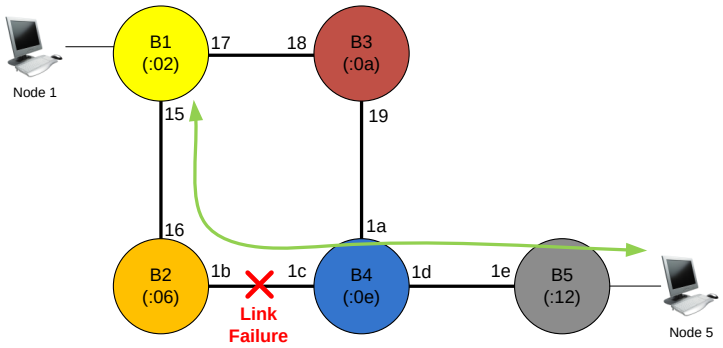


Figure 12-38 - Setup #8: RSTP protocol degradation setup

Two simulations were performed:

1. All bridges running RSTP.
2. B4 running STP and all others RSTP.

For each scenario, the convergence time was measured. Next table presents the results.

All running RSTP Convergence Time	B4 running STP Convergence Time
Aprox. 2 seconds (Figure 12-39).	Aprox. 30 seconds (Figure 12-40).

Table 18 - RSTP / STP convergence time.

As the above table demonstrates, STP bridges degrade RSTP network performance considerably.

```
64 bytes from 10.10.10.1: icmp_seq=37 ttl=128 time=0.987 ms
64 bytes from 10.10.10.1: icmp_seq=38 ttl=128 time=0.810 ms
64 bytes from 10.10.10.1: icmp_seq=39 ttl=128 time=0.948 ms
64 bytes from 10.10.10.1: icmp_seq=40 ttl=128 time=0.946 ms
64 bytes from 10.10.10.1: icmp_seq=42 ttl=128 time=0.836 ms
64 bytes from 10.10.10.1: icmp_seq=43 ttl=128 time=0.652 ms
64 bytes from 10.10.10.1: icmp_seq=44 ttl=128 time=0.846 ms
```

Figure 12-39 - Setup #8: Communication downtime with B4 running RSTP

```
64 bytes from 10.10.10.1: icmp_seq=106 ttl=128 time=0.735 ms
64 bytes from 10.10.10.1: icmp_seq=107 ttl=128 time=0.804 ms
64 bytes from 10.10.10.1: icmp_seq=108 ttl=128 time=0.873 ms
64 bytes from 10.10.10.1: icmp_seq=109 ttl=128 time=0.942 ms
64 bytes from 10.10.10.1: icmp_seq=110 ttl=128 time=0.510 ms
64 bytes from 10.10.10.1: icmp_seq=111 ttl=128 time=0.912 ms
64 bytes from 10.10.10.1: icmp_seq=142 ttl=128 time=0.719 ms
64 bytes from 10.10.10.1: icmp_seq=143 ttl=128 time=0.537 ms
64 bytes from 10.10.10.1: icmp_seq=144 ttl=128 time=0.602 ms
```

Figure 12-40 - Setup #8: Communication downtime with B4 running STP

SCS is incompatible with STP, as described in the *Self-Configurable Switches Overview* chapter. That way, SCS performance remains unchanged, despite of STP bridges connected to SCS networks.

In the above particular scenario, if all bridges were SCS, as soon as the STP bridge B4 was connected, SCS was going to self-defend against STP and B4 would never be allowed to participate in the network, connected according to Figure 12-38. Recall from the *Self-Configurable Switches Overview* chapter that SCS bridges prevent interfaces changing from *host* to *bridge interface*, i.e., if a SCS bridge detects another SCS bridge via a STP bridge, it shuts down the interface.

12.4 MSTP Problems

Besides still exhibiting *count to infinity* behaviour, MSTP adds two main drawbacks: *protocol complexity* and *interaction with legacy bridges*.

Protocol Complexity

MSTP is much more complex than legacy spanning tree protocols, is harder to understand, difficult to troubleshoot and requires additional training.

Misconfigurations of MSTP parameters might create unwanted region boundaries. Moreover, multi-region topologies and scenarios running MSTP with legacy STP/RSTP require design and troubleshooting expertise.

SCS is simple to understand and requires no configuration. Thus, misconfigurations and its potential side-effects are completely out of question. The only misconfiguration you can get is setting the wrong *Neighbour Key*, which does not allow the bridge to be connected to the network.

SCS protects itself against STP bridges, leaving the SCS network immune from STP issues propagation and its inferior performance.

Interaction with legacy bridges

MSTP creates boundary ports when senses legacy STP bridges, allowing them to connect to the MSTP Virtual Switch via the CST.

STP legacy bridges can induce unwanted instability to all MSTP network and greatly increases network topology complexity. Therefore, it is recommended to avoid connecting MSTP regions to legacy spanning tree domains.

Figure 12-41 presents a real example of two machines running legacy STP interconnected to a "*Cisco*" MSTP network. Note MSTP considers them peer-to-peer interfaces, but runs PVST (per-VLAN STP) on each interface, creating boundary ports.

```
MST0
 Spanning tree enabled protocol mstp
 Root ID   Priority   8192
        Address    0008.e3ff.xxxx
        Cost       0
        Port       1667 (Port-channel111)
        Hello Time   2 sec Max Age 20 sec  Forward Delay 15 sec

 Bridge ID Priority   12288  (priority 12288 sys-id-ext 0)
        Address    000f.3574.yyyy
        Hello Time   2 sec Max Age 20 sec  Forward Delay 15 sec

Interface      Role Sts Cost      Prio.Nbr Type
---------------- ---- --- --------- -------- --------------------------------
Gi1/2          Desg FWD 20000     128.2    Edge P2p
Gi3/15         Desg FWD 20000     128.271  Edge P2p
Gi3/34         Desg FWD 200000    128.290  P2p
Gi3/35         Desg FWD 20000     128.291  P2p
Gi4/14         Desg FWD 20000     128.398  Edge P2p
Gi9/21         Desg FWD 20000     128.1045 P2p Bound(PVST)
Te10/5         Desg FWD 2000      128.1157 P2p Bound(PVST)
Te11/6         Desg FWD 2000      128.1286 P2p
Gi13/10        Desg FWD 20000     128.1546 Edge P2p
Gi13/11        Desg FWD 200000    128.1547 P2p
Gi13/18        Desg FWD 20000     128.1554 Edge P2p
```

Figure 12-41 - MSTP interacting with legacy STP

As stated before, SCS networks are immune to issues raised by legacy STP bridges. To peak the immunization, STP bridges can be administratively forbidden to connect to SCS networks.

Misinterpreting IST

This is a particular issue of MSTP networks, already discussed on Misinterpreting IST section on page 46. However, it shows once again MSTP protocol complexity and the resulting misconfiguration and misinterpretations of the protocol, which SCS does not arise.

12.5 Network Topologies

This document classified the several STP flavours as inefficient, in numerous aspects, ranging from poor convergence times into weak resiliency to network changes. However, the single spanning tree created within the network is one of STP's major drawbacks, as reduces a well-designed, high-performance and redundant network, into a single communication path between all network bridges, reducing all investment in bandwidth and redundancy to simple dormant links.

Figure 12-42 shows a highly redundant network, segregated into core, distribution and access layers.

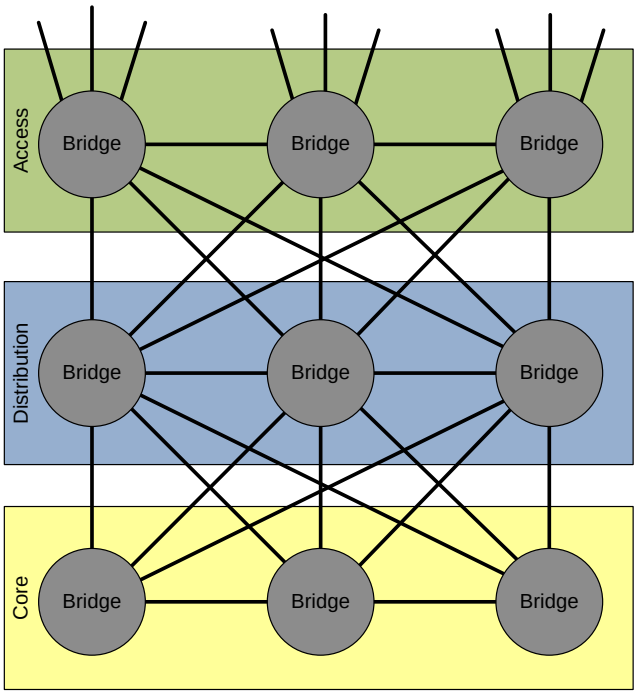


Figure 12-42 - Beautiful meshed network

Consider each connection between bridges as 10Gbps links. Thus, each layer is interconnected by a *bundle* of 3x 30Gbps, meaning 90Gbps, for a total capacity of 240 Gbps.

Let's see STP working on this great 240Gbps multi-layer redundant network. Figure 12-43 shows the *spectacular* result!

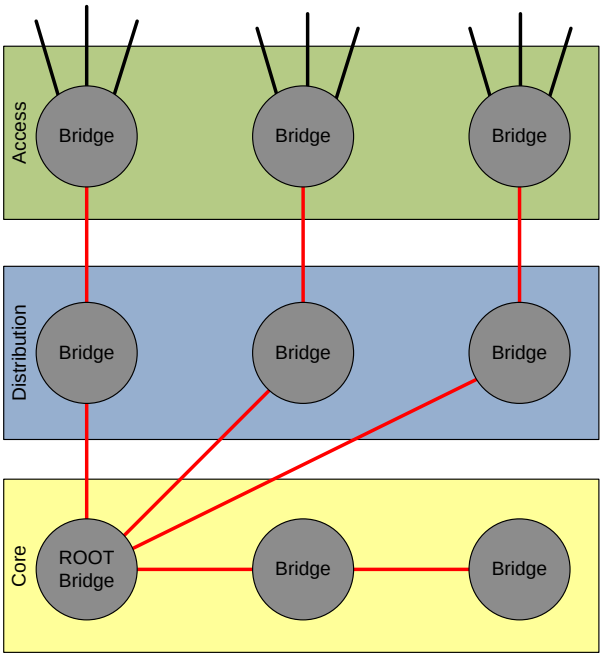


Figure 12-43 - Spanning Tree resulting topology

STP has the *magic* to eliminate 160 Gbps of network's capacity, near 67%, and reduce the 90Gbps uplinks between layers into one third, i.e., 30Gbps. STP takes a perfect, well meshed network and reduces it to a tree!

Moreover, Figure 12-43 illustrates one other big issue regarding STP. Reducing drastically the available network bandwidth, it will therefore overload the remaining forwarding links. Even for communications between devices attached on the edge, the Spanning Tree Protocols drive all traffic through the root bridge, meaning congestion on the uplinks towards the root bridge, for sure. Figure 12-44 illustrates uplinks overloading towards the root bridge, with unnecessary traffic between edge machines.

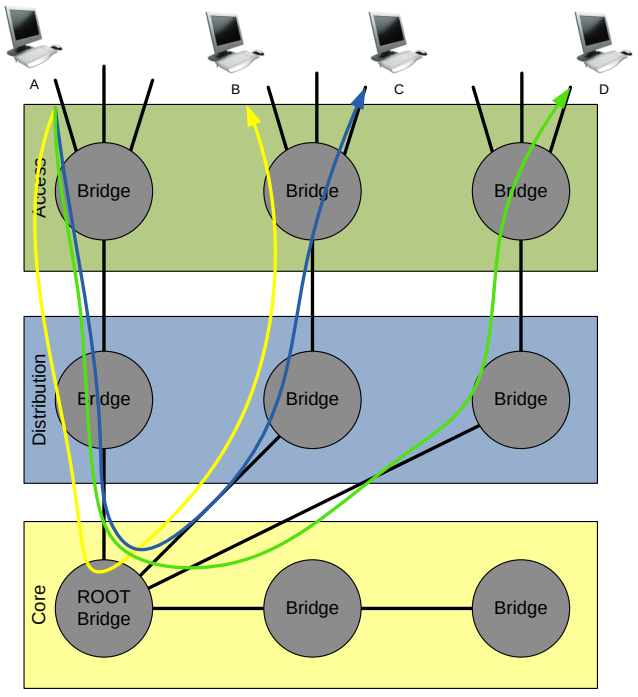


Figure 12-44 - Uplinks overloading towards the root bridge

SCS, in turn, would leave the great 240Gbps multi-layer redundant network as is, leveraging all the available bandwidth and bridges interconnections towards high performance bridging.

Figure 12-45 illustrates how SCS relieves the core and distribution layers of intra-access communications, choosing the best path towards the destinations.

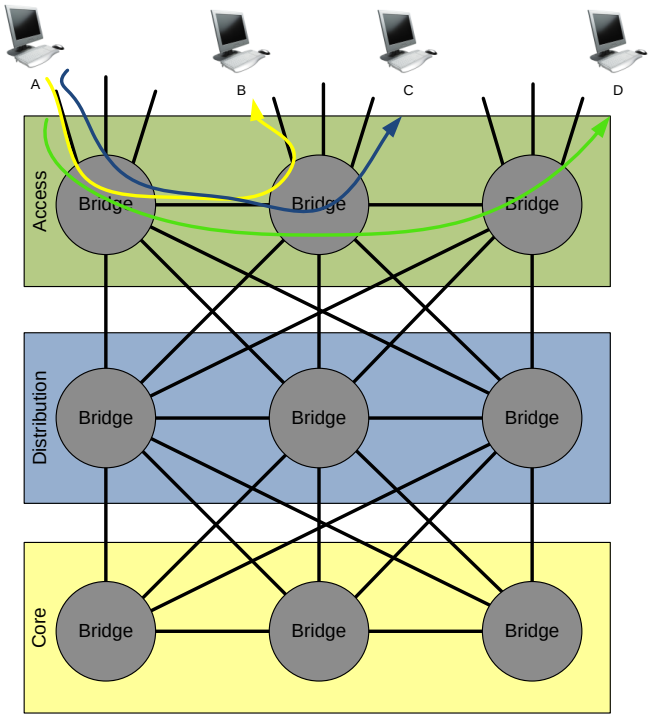


Figure 12-45 - SCS high performance network

12.6 SCS versus STP protocols

This chapter presented some of the most service impacting issues STP protocols face, both the original STP and its successor RSTP, but also the improved MSTP.

SCS addresses all those issues and presents a superior response in all of them.

Better performance, stability, resilience to network changes, topology independency, link load balance, operational redundant paths, optimal paths and zero-configuration, are some of SCS deliverables that STP cannot follow, as this chapter demonstrated.

Conclusions

This work has a single objective: replace the Spanning Tree Protocol.

Spanning Tree Protocol substitute should provide self-configuring capabilities, optimized forwarding paths for unicast and broadcast traffic, high performance and traffic load sharing over redundant paths, in which bridges are plugged transparently requiring zero-configuration.

Self-Configurable Switches protocol suits all these requisites.

In Part I, the standards Spanning Tree Protocol, Rapid Spanning Tree Protocol and Multiple Spanning Trees were analysed, their main features, processes and problems.

Part II presented and compared the two recent proposed standards from IEFT and IEEE, TRILL and 802.1aq SPB. Some considerations were produced regarding TRILL and SPB positions to replace the current standards.

Part III explains all the core concepts, processes and mechanisms of Self-Configurable Switches protocol and Part IV explain how SCS was simulated to assess its power, functionality and feasibility.

In the end, some experiments and simulations were performed to attest SCS behaviour against STP, RSTP and MST, revisiting the problems spanning tree protocols face and detailing how SCS overcomes all of them, one-by-one, providing at the same time all the aforementioned requisites necessary for a protocol to replace the Spanning Tree Protocol.

Final Conclusion

Temporary loops are a fact of life in networks. The question is: how do you detect them and deal with them and whether they affect the network performance.

Spanning Tree Protocol it's a *loop preventive* protocol; not *loop detective*. So, if a loop occurs in the network, it's not STP the one that will *break* it, most of the times. Moreover, STP failures in loop prevention are sometimes the ones responsible for loops to occur.

Rapid Spanning Tree Protocol can be seen as an evolution of the Spanning Tree Protocol standard and introduces convergence time optimizations and increase stability. However, RSTP is permeable to "*race conditions*", raising serious problems in redundant topologies.

The instance and region concepts of MST are good ideas. Nevertheless, running RSTP in the instances and multi-region topologies complexity are MST handicaps. Besides, running MSTP with legacy STP/RSTP requires high expertise in network design and troubleshooting.

SPB and TRILL are standard proposals. They are targeted to backbone or carrier networks and will increase considerably network's CAPEX and OPEX. Personally, I found TRILL unattractive; SPB "service" awareness is interesting.

For Layer 2 networks, SCS has a significant different paradigm than spanning tree protocols, maximizing the available redundancy and the investments made: the network has loops and they are not only allowed but also useful. STP breaks all the redundancy creating a single topology.

SCS continues to be a Layer 2 protocol, placing itself between the spanning tree protocols and the proposal standards TRILL and SPB, targeting network performance and stability optimization, delivering optimum forwarding, load balance, requiring no configuration neither substantial investments.

Future Work

There are some topics that can be further analysed and optimized in future work related with SCS protocol.

Concerning queries to the FwdT, the load balance engine for parallel links between adjacent neighbours can be optimized to deliver more intelligence to traffic distribution, as currently is simply a random choice between the multiple records.

One simple improvement could be performed on SCS code, in order to avoid bridges to propagate towards the hosts, the broadcast frames spoofed during the inverted flooding mechanism. Using a (new) small 1-bit flag on SCS UF packets, the bridges could signal adjacent neighbours about the flooding need towards the hosts, for the encapsulated Ethernet broadcast frame.

To further expand SCS functionality, there are three specific topics that can be explored: security, traffic segregation and multicast.

Regarding traffic segregation, SCS might allow traffic distribution over redundant topologies creating multiple SCS domains, redundant of each other. That way, besides resiliency, redundancy, fast convergence, optimal paths and load balancing, SCS would also provide traffic segregation.

Multicast traffic forwarding should be also further analysed, in order to deliver multicast traffic to only those bridges which request it.

Security should be always taken into consideration. SCS already delivers neighbour bridges authentication, to protect the SCS network against reckless insertion of new SCS bridges into the network. However, this clear-text 6-bit key is not suitable for security purposes, so it does not target secure communications neither bridge protection; it aims SCS network protection against careless insertion of bridges. To protect the bridging network, secure channels between adjacent bridges could be provided, for example, to protect against MITM attacks.

QoS and VLAN tagging are still possible to be employed, as the L2 information is not lost with SCS. VLAN topologies could be mapped into SCS domains, to create broadcast domain isolation and redundant topologies.

Ideally, to better understand its limitations and challenges on real world networks, a physical prototype development could be taken into consideration.

Bibliography

[1] http://www.computerworld.com/s/article/78803/All_systems_down
[2] http://ieee802.org/secmail/msg02970.html
[3] http://ieee802.org/secmail/msg02973.html
[4]http://www.networkworld.com/news/2006/050506-sun-radia-perlman-interview.html?page=3
[5] Khaled Elmeleegy, T. S. Eugene Ng. and Alan L. Cox. Supplemental Note on Count-to-Infinity Induced Forwarding Loops in Ethernet Networks.
[6] Andy Myers, T.S. Eugene Ng and Hui Zhang. Rethinking the Service Model: Scaling Ethernet to a Million Nodes.
[7] LAN/MAN Standards Committee of the IEEE Computer Society. IEEE Standard for Local and metropolitan area networks: Media Access Control (MAC) Bridges - 802.1D, 2004.
[8] 802.1s-2002 - IEEE Standards for Local and Metropolitan Area Networks - Amendment to 802.1Q Virtual Bridged Local Area Networks: Multiple Spanning Trees
[9] 802.1Q-2005 - IEEE Standard for Local and Metropolitan Area Networks---Virtual Bridged Local Area Networks
[10] 802.1Q-2011 - IEEE Standard for Local and metropolitan area networks--Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks
[11] ISO 7498:1984 - Information processing systems -- Open Systems Interconnection -- Basic Reference Model.
[12] ISO/IEC 7498:1994:1 - Information processing systems -- Open Systems Interconnection -- Basic Reference Model: The Basic Model.
[13] R. L. Graham, Pavol Hell: On the History of the Minimum Spanning Tree Problem, Annals of the History of Computing, Volume 7, Number 1, January 1985.
[14] Mark Huang, Eden Miller, Peter Sun, Charles Oji. Ethernet: An Engineering Paradigm. Structure, Practice, and Innovation in EE/CS, 1998.
[15] Routing Bridges (RBridges): Base Protocol Specification – IETF RFC 6325 http://tools.ietf.org/html/rfc6325
[16] TRILL: The ESADI Protocol. http://tools.ietf.org/html/draft-hu-trill-rbridge-esadi-04
[17] OSPF v2 – IETF RFC 2328.
[18] OSI ISIS Intra-domain Routing Protocol - ISO/IEC 10589:2002 / IETF RFC 1142.
[19] Internet Protocol Journal, Volume 14, Number 3, September 2011.
[20] Routing Without Tears; Bridging Without Danger – Radia Perlman, April 24, 2008. http://www.youtube.com/watch?v=N-25NoCOnP4
[21] RBridges: Transparent Routing. Radia Perlman, IEEE INFOCOM, 2004.
[22] Layer 2 Routing (sort of) and TRILL – LAME Journal, 16 May 2011, John Herbert.
[23] Network Fabric: TRILL for Server and Network People. Welcome RBridges. Greg Ferro, 30 March, 2009.
[24] TRILL in the Data Center: Look Before You Leap. Understanding Fundamental Issues with TRILL, Juniper Networks, Inc., 2011.
[25] IS-IS Extensions Supporting IEEE 802.1aq Shortest Path Bridging – IETF RFC 6329, April 2012.
[26] IEEE APPROVES NEW IEEE 802.1aq™ SHORTEST PATH BRIDGING STANDARD. May 8 2012.

http://standards.ieee.org/news/2012/802.1aq.html
[27] 802.1ag - Connectivity Fault Management.
http://www.ieee802.org/1/pages/802.1ag.html
[28] 802.1ah - Provider Backbone Bridges,
http://www.ieee802.org/1/pages/802.1ah.html
[29] Introduction to Shortest Path Bridging - Avaya Inc, September 2011.
[30] Network Virtualization using Shortest Path Bridging and IP/SPB - Avaya Inc, June 2011.
[31] Compare and Contrast SPB and TRILL - Avaya Inc, June 2011.
[32] GNU General Public License, version 2.
http://www.gnu.org/licenses/old-licenses/gpl-2.0.html