2010

Coded based protection in mesh networks

Cole, Aaron

Coded Based Protection in Mesh Networks

By: Aaron Cole

Supervised by: Dr. Hassan Naser

August 2010

A Thesis submitted in partial fulfillment of the requirements of the M.Sc.Eng degree in

Electrical and Computer Engineering

Faculty of Engineering

Lakehead University

Thunder Bay, Ontario

Your file Votre référence
ISBN: 978-0-494-71917-6
Our file Notre référence
ISBN: 978-0-494-71917-6

Abstract

Since the Internet revolution of the 1990s; ever increasing levels of connectivity have been integrated into society. This has ushered in the era of globalization and a new plateau in prosperity. Credit for this accomplishment can be placed firmly on our communication networks. However, our incorporation of telecommunications into society has led to a dependency on it. Our escalating reliance on telecommunications has made society highly susceptible to fault occurrences. Consequently, the field of network survivability is required to maintain reliability in our telecommunications infrastructure. Mesh networks have been touted as the successor to the ring based networks of the past due to their efficiency and scalability. Unfortunately, mesh networks owing to their complexity have not been able to obtain restoration times comparable to its predecessor. This issue has led to a polarization of survivability schemes, where restoration time is pitted against redundancy requirements.

In order to mitigate this problem; network coding based survivability algorithms are being proposed. Network coded based protection uses coding theory to linearly combine disjoint connections. This permits restoration times comparable to dedicated mesh schemes while having significantly less redundancy requirements. We propose three schemes of coded survivability known as Source Coded Protection, Multiple Source Coded Protection, and Network Coded Protection. From these three schemes, eight novel heuristic algorithms have been created.

Two of these heuristic algorithms have been designed to generate realizations of Source Coded Protection. The first of these algorithms, Near Optimal Source Coded Protection produces highly capacity efficient version of Source Coded Protection. On the other hand,

Fast Source Coded Protection also produces Source Coded Protection, but with a minimal amount of computation time.

Adding to those two algorithms, three more have been designed for Multiple Source Coded Protection. Two of these algorithms have been designed to improve capacity utilization and computation time. Re-optimized Multiple Source Coded Protection creates a capacity efficient version of Multiple Source Coded Protection, while Fast Source Coded Protection creates the same scheme in a short time period. The third algorithm for Multiple Source Coded Protection known as Single Stream Multiple Source Coded Protection generates a specific version of Multiple Source Coded Protection that requires less decoding at the destination.

The last three algorithms are designed to create Network Coded Protection. Due to the greater varieties of the Network Coded Protection scheme, each algorithm generates its own version of the technique. Neighbor Decoded Protection creates a specific variation of Network Coded Protection that guarantees at most one hop restorations during fault events. The second Network Coded Protection algorithm known as Trunk Coded Protection, uses predetermined coded paths to ease the economic burden of providing an entire network with coded protection functionality. Lastly, the Stream Based Network Coded Protection algorithm uses a novel stream based sharing technique to generate Network Coded Protection in capacity efficient manner.

In house C++ simulations were used to compare the performance metrics of these eight heuristic algorithms with two established benchmark algorithms used for survivability. These two benchmark algorithms represent both edges of the choice between restoration time and redundancy. Through a comparison of the proposed algorithms with the bench-

mark algorithms, it will be shown that these coded based algorithms allow network operators to have both low restoration time and a reduced redundancy requirement.

Acknowledgment

I would like to thank my supervisor Dr. Hassan Naser for his guidance and Patience throughout while completing this thesis. It is my firm belief that the best professors only need to provide a guiding hand to their graduate students.

I would also like to extend my thanks to my family and friends. Without all your love and support, this would not be possible.

Contents

List of Algorithms

List of Figures

List of Tables

Nomenclature

α_{ij} A binary number which determines whether a piece of information X_j exists in a coded route $R_{wi}(r)$.

Γ_n The set of neighbor nodes for a node n.

$A_\gamma(s)$ The availability of a protection set s for a traffic demand.

A_s The availability of a system.

b The bandwidth required by traffic demands.

$C(r)$ The cost of routing a connection request r.

$C(s)$ The critical node of a protection set s.

$C_\gamma(ij)$ The cost of routing a connection over link ij for a route $R_\gamma(r)$.

C_j An n-dimensional vector used to linearly combine information sources into a stream j.

$D(r)$ The destination node of a traffic demand r.

$D(s)$ The destination node of a protection set s for MSCP based algorithms.

$D_\gamma(s)$ The available destinations for route γ.

d_{max} The pseudo min-cut between two nodes in the network.

F_s The time a system is in a failed state.

$G = (N, J)$ A graph abstraction of a real network that contains a set of links J and nodes N.

O_s The time a system is in an operational state.

$P_l(s)$ a set of phantom links between the phantom source node $P_s(s)$ and the source nodes $S(r)$ of the traffic demands r in the protection set s.

$P_n(s)$ A phantom network for the protection set s.

$P_s(s)$ a phantom source node for a RMSCP protection set s.

r A traffic demand in the network.

$R[ij]$ The real link represented by the phantom link ij.

$R_\gamma(r)$ A set of links representing path γ for a traffic demand r.

$R_c(s)$ A set of coded links and nodes for a protection set s.

$R_s(s)$ A set of shared links and nodes for a protection set s.

$S(r)$ The source node of a traffic demand r.

x_j A piece of information.

z A zone consisting of a set of nodes.

FITS Failures In Time: The number of failures experienced by a system in a billion hours.

MTBF Mean Time Before Failure

MTTR Mean Time To Repair

SLA Service Level Agreement

1. Introduction

1.1. Motivation

The global communications network has exploded in size since its introduction at the dawn of the information age. From its humble beginnings with the telegraph, it has expanded to meet humanities insatiable demands for first voice and now data. Resting on the achievements of this network is the information age, where the distance between two points provides no hindrance to cooperation and integration. Globalization is not possible without the services provided by our global communications network. With all of society virtually dependent on our communication networks for the performance of daily tasks, it is monumentally important that we ensure that events that can disrupt our communications are mitigated quickly and efficiently. Thus network survivability schemes are required to handle these disruptions. Normally, network designers are given a choice between quick dedicated protection and efficient shared restoration. For dedicated protection, large quantities of bandwidth are reserved so that two copies of the same information can be sent disjointly from source to destination. This is costly and not an economically acceptable option for creating survivable networks. For shared restoration, every scheme is based off the automatic re-request concept where by a secondary route is used to circumvent a network failure. From a time sensitive point of view this can cause congestion for client networks operating over the network failure. This problem is especially true if a large amount of

connections have failed simultaneously. To resolve these issues, a forward error correction based protection scheme can be utilized.

1.2. Thesis Contributions

The contributions of this thesis are in the further development of coded based survivability. Presented in this thesis are three survivability schemes for providing coded based survivability. Adding to that, in order to create realizations of those three schemes, eight heuristic algorithms have been proposed. The contributions of this thesis can be further summarized as follows:

1. A source coding based survivability scheme known as *Source Coded Protection (*SCP*)*. By fragmenting and linearly combining traffic between a source-destination pair over multiple diverse paths, source coded protection protects a connection against single link failures. Since this technique uses the concepts of forward error correction and network coding, it combines the attributes of negligible restoration time with efficient capacity consumption. Together these attributes produce a technique that is ideally suited for platinum service level agreements in logical networks. Two heuristic algorithms have been created to provide SCP. These algorithms are *Near Optimal Source Coded Protection* (NOSCP) and *Fast Source Coded Protection* (FSCP). Each of these algorithms are designed to maximize a specific quality of routing algorithms while generating SCP. NOSCP strives to produce the most capacity efficient version of SCP. However, it requires a significant amount of time to perform the path computations. On the other end, FSCP generates paths in a short period of time, but with a less capacity efficient result.

2. A multiple source coding based survivability scheme termed *Multiple Source Coded*

*Protection (*MSCP*)*. This technique combines the benefits associated with SCP with the ability to protect traffic from multiple source nodes. By removing the single source-destination pair and data fragmentation constraints of SCP, this novel technique produces favorable results under a greater variety of conditions. Due to the unique properties of MSCP, three heuristic algorithms have been generated for it. These three algorithms are presented as *Fast Multiple Source Coded Protection* (FMSCP), *Re-optimized Multiple Source Coded Protection* (RMSCP), and *Single Stream Multiple Source Coded Protection* (SSMSCP). Like the realizations for SCP, each of the algorithms for MSCP attempt to maximize a performance criteria of the routing algorithm or function of MSCP. FMSCP reduces the MSCP routing problem so that it can be solved with the minimum number of path computations. However, it performs this function with a reduced level of network capacity efficiency. RMSCP re-optimizes previously established connections in a network with new connection requests to produce a capacity efficient result. However, like NOSCP for SCP, RMSCP requires more computational time to resolve the new path for each of the connections. As opposed to FMSCP and RMSCP, which use different techniques that create same version MSCP, SSMSCP creates a specialized variant of MSCP. In SSMCP, the MSCP scheme is generated such that it is computationally easier for destinations to decode information.

3. A network coding based survivability scheme known as *Network Coded Protection* (NCP). MSCP only allowed coding if connection requests were heading to the same destination. This can limit the sharing opportunities available in the network to an unacceptable level. Thus a coded survivability scheme is required that allows sharing between multiple sources and destinations. NCP is a coded based survivability scheme that obtains this functionality. It does this by merging the attributes of traditional survivability techniques with network coding. Three heuristic algorithms

have been generated from NCP. These heuristic algorithms are *Neighbor Decoded Protection* (NDP), *Trunk Coded Protection* (TCP), and *Stream Based Network Coded Protection* (SBNCP). Each of these algorithms generates a specific form of NCP that is ideal for different applications. NDP has been designed to produce a form of NCP that guarantees at most one hop restorations for all connections. This is ideal for large networks, where the distance between the source and destination can create unbearable delays during restoration events. On the other hand, TCP has been created to allow network operators more control over the coding operations required for network survivability. It uses predefined coded trunks through the network to provide survivability. With this, operators can reduce the complexity and cost of certain areas of a network. Lastly, SBNCP has been designed to produce the most generalized form of NCP. It does this using a new technique known as *Stream Based Sharing* (SBS). With this technique, SBNCP produces a more capacity efficient form of NCP than the previous two heuristic algorithms.

Each of the heuristic algorithms has been simulated using an in house C++ simulation of the *Global Crossing Network* (GCN) and *National Science Foundation Network* (NSFNET). In order to properly compare the presented heuristic algorithms, two benchmark algorithms have also been simulated. These benchmark algorithms are *Simple Pool Sharing* (SPS) and *Shortest Pair Dedicated Path Protection* (SPDPP). Together they represent opposite edges of the efficiency versus performance spectrum. With these two benchmarks as upper and lower bounds on performance, the heuristic algorithms have been compared. They are compared using demand blocking probability, redundancy, network capacity utilization, availability, and restoration time.

1.3. Thesis Outline

This thesis is outlined as follows. Chapter 2 introduces the background knowledge required for an understanding of network survivability. It begins with a treatise on transport network failures and their affect on customers. Following that is the concept of network survivability and the aspects that are important to this thesis. The last part of the chapter is reserved for a brief explanation coded based survivability.

Chapter 3 presents in detail the three coded based survivability schemes proposed in this thesis. For improved understanding, the schemes are presented in order of complexity. SCP is the simplest of the coded schemes and therefore is presented first. Likewise, since MSCP is only a generalization of SCP, it follows it within the chapter. Lastly, the NCP scheme is presented. Due to the additional complexity and numerous variations, extra time will be spent on this concept.

After each of the three survivability schemes has been explained, the eight different realizations can be introduced. The explanation of these algorithms are presented in chapter 4. In order to maintain continuity with chapter 3, all the algorithms for SCP will be presented first, followed by MSCP and then NCP. The realizations for SCP and MSCP are presented in order of capacity efficiency. This is because explanations of each technique flows smoothly from the capacity efficient to the reduced computation time version. Thus for SCP, NOSCP is explained first and FSCP second. On the other hand, RMSCP is explained first for MSCP. This is followed by FMSCP. SSMSCP is explained last because its unique form of MSCP would disrupt the important relationship between RMSCP and FMSCP. For the NCP realizations, a different approach is taken. Since each algorithm generates a different form of NCP, they are ordered differently. Since NDP can be under-

stood as a modification of SSMSCP using NCP principles, it is explained first. Likewise, since TCP shares some routing features in common with the MSCP heuristic algorithms, it follows NDP. At the end of the chapter is SBNCP. Included with the aspects of SBMCP is an explanation of the SBS technique and how it is used in the algorithm. Since the SBS technique used in SBMSCP is different from all the other approaches to routing used by the previously mentioned algorithms, it is given special attention.

With all the heuristic algorithms proposed for this thesis explained, chapter 5 can focus on the simulation results. In order to understand the results for each of the algorithms, chapter 5 begins with a brief explanation of each of the performance metrics. Following that are descriptions of the Global Crossing and National Science Foundation networks. Chapter 5 concludes with comparisons of each of the proposed and benchmark algorithms. Finally, chapter 6 is dedicated to our concluding remarks and proposals for future work.

Following the conclusion, all the proposed algorithms are presented in algorithmic form in appendix A. Afterward, an overview of the benchmark algorithms is presented in this thesis in appendix B. These two benchmark algorithms can be used as a foundation for understanding some of the algorithms presented in this thesis. Therefore, it is important that they be briefly explained with the contributions of the thesis. Furthermore, because of its extensive use in this thesis, the min-cut max flow theorem is attached to this thesis as appendix C. Lastly, for quick reference, the path search algorithms used throughout this thesis by the proposed and benchmark algorithms are included as appendix E.

2. Background

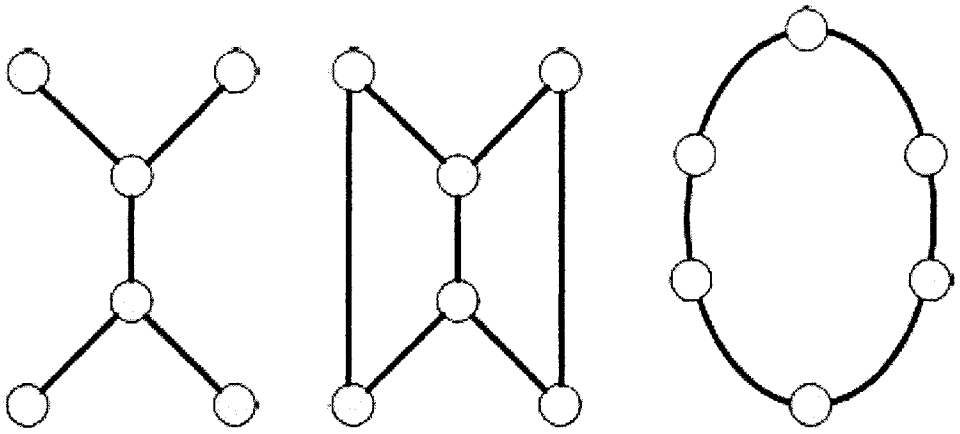
2.1. Survivability

The idea of network survivability has been around since the dawn of the information age. Even the design of the original ARPANET was designed to ensure that communication should be available after large sections of the network fail from nuclear war. As mentioned in section D.1, the primary concern today is handling the network failures associated with day to day problems. These problems take the form of single-link failures as mentioned in section D.2.1. Node failure incidents are not considered day to day events and are usually protected by redundant hardware [1]. No matter how the failure occurs or what it's effects are on the network topology we want to restore connections as fast as economically possible. Due to the computerization of traffic rerouting, the speed of connection recovery has been reduced to a technical issue. As mentioned in section D.3 the connection recovery speed problem is now determined by it's effect on client layers of the physical network. This works along the concept that as long as the client layer doesn't have time to detect a major problem neither will the customer. However, some customers may want to purchase extremely fast restoration times and high levels of availability for their traffic. While at the same time some customers may not even want their traffic restored, if it reduces the cost. Therefore, allowing customers to select their desired service level agreement is required. Survivability techniques that take this into consideration will have distinct advantages over others. From this we can define network survivability as

Network survivability is the ability of a network to withstand failure events. It demonstrates the resilience of the network against failure events and is measured in terms of the reliability, restorability and end-to-end availability of the light path [2].

2.1.1. Physical Network Requirements

A fundamental requirement for survivable networks is the ability for data to always have a possible connection between every source-destination pair after a failure has occurred. Figure 2.1a is a tree like graph that does not meet this fundamental requirement for survivability. In a failure event there is no alternate connection possibility between each source-destination pair. Since an overwhelming majority of all failures are of the single link type, this requirement translates into the need for bi-connected networks. In figure 2.1b there are no single link failure scenarios that will separate a node from the network. Figure 2.1c depicts the smallest possible bi-connected network as a ring, which is why it has been given preference for survivability in first generation SONET networks.



(a) Tree Like Graph (b) Bi-Connected Graph (c) Ring Graph

Figure 2.1.: Network Types

2.1.2. Survivability Techniques

There are many different schemes capable of providing survivability to a network. Figure 2.2 depicts a tree of the different methods. With the myriad of choices available, network designers are left to make the decisions. Where do we provide the survivability features. That is; do we protect links, paths, channels, or segments? If we choose to protect links, what kind restoration architecture do we overlay on these links to make them survivable? Likewise, if we choose to protect paths, do we pre-compute and assign the backup path before the failure occurs or do we restore traffic afterward? This choice is not black and white, as there are many variations that will be eluded to later. Moreover, if we choose to pre-compute backup paths for demands, do we allow it to share the redundant capacity with other connections?

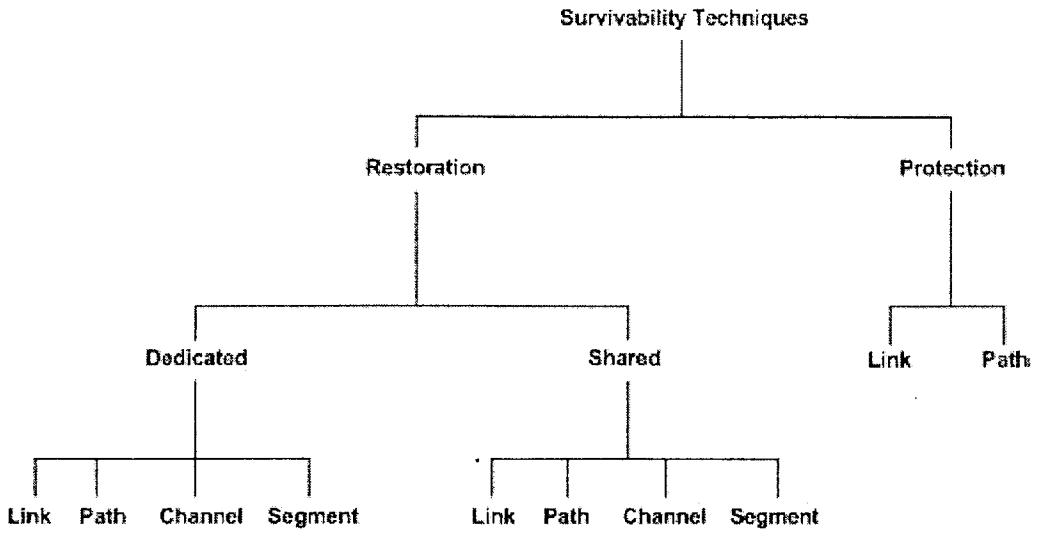


Figure 2.2.: Survivability Techniques

Until recently it was thought that you could only effectively use restoration techniques with link based survivability in ring networks. This ideology led the simple yet inefficient SONET ring based protection. However, this is changing as new path based schemes are being invented. Carrying on with this trend, this thesis utilizes path based schemes. The remainder of this section is devoted to path survivability. For more information on the other techniques available, refer to [2, 3, 4, 5].

2.1.2.1. Path Survivability

In both path protected and restorable networks, survivability is created from the end-to-end perspective of a connection. This technique can trace its roots to the late 1990s when path survivability became more competitive with link survivability [6, 7]. With path survivability, demands are provided with a working and backup route. For most path-based survivability schemes, the working route is used exclusively in normal operation, while the

backup route is reserved until a failure occurs. When a failure occurs on the working path of a demand, the source node will switch the demand onto the backup route. An exception to this is dedicated backup path protection, which will be explained with the other survivability techniques throughout this subsection. As shown in figure 2.3, when a working connection fails, information is switched at the source node onto a backup connection.

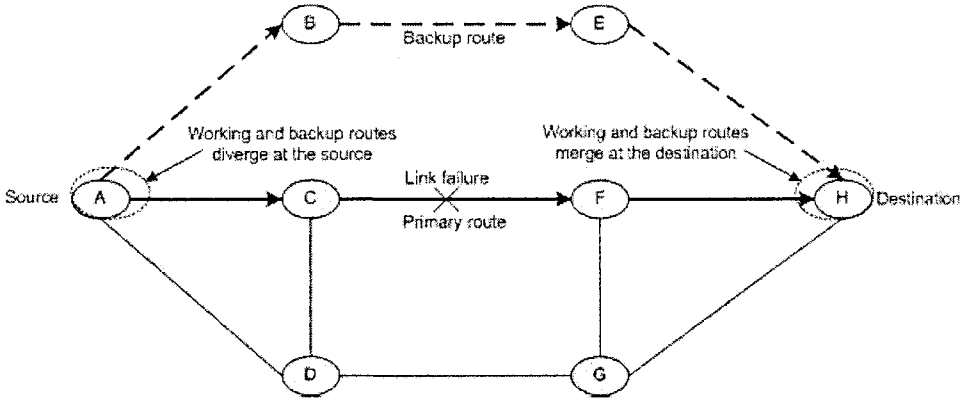


Figure 2.3.: Path Protection

Table 2.1 depicts several advantages and disadvantages associated with using path based survivability schemes. First of all, from the advantageous side, the minimum possible redundancy requirements are exhibited by path perspective schemes. This advantage stems from the availability of all links in the network for finding the optimal backup paths. Additionally, since connections are protected individually, service level agreements can be offered to individual demands. Link and segment-based schemes can not provide that service. Furthermore, since protection is offered at the ends of the network, it can be easily provided by higher layer logical connections. This can allow network operators to design traditional low intelligence fast transport layers. This can also allow networks the option of using simpler higher layer failure detection and correction protocols, such as the coded based protection techniques presented in this thesis. Unfortunately, these advantages come with

a series of disadvantages. When a failure event occurs many individual connections need to be rerouted as opposed to one span in link restoration schemes. This can heavily congest the network when it is already on its knees. Adding to that, each of those individual connections require an on-line computed backup route determined by heuristic algorithms, which can not always provide optimal results. Since most path protection schemes involve some form of sharing, a great deal of signaling is required to setup connections. This added signaling results in an increased restoration time. Lastly, with increased sharing and the possibility of long working and backup paths, traffic demands can experience lower levels of availability. In the worse cases this can be low enough to violate certain SLAs.

Table 2.1.: Path Perspective Advantages and Disadvantages

Advantages	Disadvantages
Individual service level agreements	Greater failure response complexity
Minimal redundancy requirements	Additional routing complexity
Simpler transport networks	Added signaling requirement
Reduced transport layer signal monitoring requirement	Longer restoration time
	Reduced availability

With path based survivability, a new dimension of choice is given to network operators. Before with link based survivability, network redundancy was utilized for survivability at the physical layer. Path based survivability is not limited in this regard, since resources can be allocated at any physical or logical layer in the network. This has led to a wealth of studies on where survivability should be added to a network[8, 9]. From the studies, it has been concluded that lower layers have a higher reliability and higher layers have a greater efficiency.

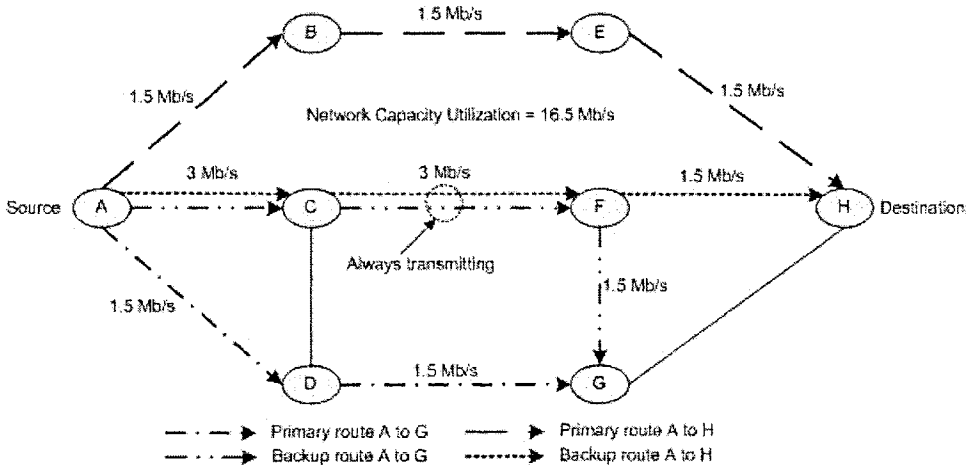
2.1.2.2. Dedicated Backup Path Protection

In *Dedicated Backup Path Protection* (DBPP) multiple diverse connections with identical information are setup between a source-destination pair [10]. Since both signals are identical, in failure events the destination node chooses the best of the two. This results in extremely high availability and a negligible restoration time. Unfortunately, this requires that at least 100% of the working routes capacity utilization be reserved for the backup connection. In networks with low nodal degrees this redundancy requirement can skyrocket much higher. The concept behind DBPP is represented in figure 2.4a. In this figure, there two 1.5 Mbps demands routed in a simple network. The first demand has a working connection along the path ABEH, and a backup connection along ACFH. Since the number of links along the backup route is equal to the number of links along the working route, the redundancy is 100%. Likewise, the second 1.5 Mbps demand has a working connection that goes along the path ADG and a backup connection that goes along ACFG. This results in a 133% redundancy requirement. Adding both demands together we get a total network capacity utilization of 16.5 Mb/s. This is opposed to the 7.5 Mbps required to route only the working paths for the traffic demands.

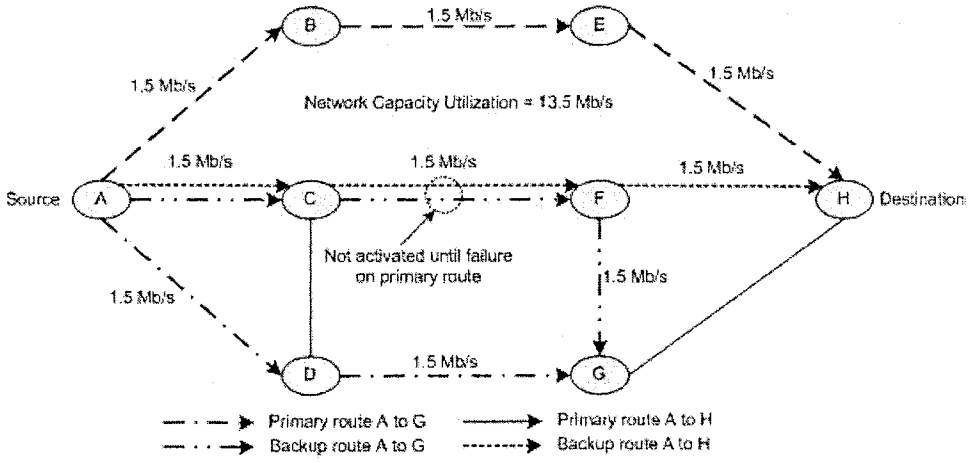
2.1.2.3. Shared Backup Path Protection

In *Shared Backup Path Protection* (SBPP) schemes, demands are allowed to share redundant capacity along their backup routes[11]. This sharing is usually done under the condition that both demands are disjoint from one another. Figure 2.4b depicts how a SBPP technique could be used to protect two 1.5 Mbps demands. As with figure 2.4a the two demands have working paths going over ABEH and ADG respectively. The two demands also have backup paths going over ACFH and ACFG. Additionally, since the two working paths are disjoint, they can share capacity along their backup routes. Since the redundant capacity is being shared between multiple demands, they can not use it until

a failure occurs. Furthermore, when a failure is repaired, the demand must revert to the working route, so that another demand can have the opportunity to utilize the shared capacity. This is opposed to DBPP which is non-revertive. With the redundancy reduction provided by SBPP, the total network capacity utilization is only 13.5 Mb/s. That is a 18% reduction over the DBPP network in figure 2.4a.



(a) Dedicated Backup Path Protection



(b) Shared Backup Path Protection

Figure 2.4.: Path Survivability

2.1.2.4. Generalized Path Survivability Schemes

As mentioned earlier, the choice between DBPP and SBPP is not black and white. Table 2.2 depicts the seven different variations available. Each of the seven schemes is distinguished by the three major components utilized in the failure recovery procedure. These three components are the alternate failure recovery route, the channel assignment along that route, and the optical switches cross-connecting the signal. Generally speaking, if a component is independent of any failure then it may be assigned before the specific event occurs. The only path survivability techniques which assign every component in advance of a failure are *Dedicated Backup Path Protection* and *Shared Backup Path Protection with Pre-assigned Channels*. The fundamental difference between the two is that the former sends two signals simultaneously, while the latter does not send redundant information until a fault occurs on the working path. All other calculations and setup operations are performed in advance. The difference between the two schemes is that in the latter the redundant signal is not sent between the source destination pair until after the failure occurs. The remaining five schemes are dependent on where the failure occurs and must re-actively perform computation and setup procedures. For example, in *Shared Backup Path Protection with Non Pre-assigned Channels*, the failure recovery route is computed and assigned before the failure occurs but no channel computations are performed. Of the remaining techniques, two utilize pre-planned maps. These two techniques *Shared Backup Path Protection with Preplanned Maps (routes and channels)* and *Shared Backup Path Protection with Preplanned Maps (routes only)* utilize preplanned maps to simplify the failure rerouting problem using a backup network map. This map is a network consisting of all the reserved bandwidth available for survivability. In a fault event, a connection can

be quickly rerouted over the backup network map. Unfortunately, this technique requires that the network be able to rapidly isolate the fault and quickly reroute over the redundant network. The remaining techniques, *Re-provisioning* and *Shared Backup Path Protection with Pre-assigned Backup Paths and no Reservations* are entirely reactive in nature. For them, best effort approaches are used. In the former technique, the failure recovery route and its channel allocation procedures are performed re-actively after the failure. While the latter method computes the failure recovery route in advance without any capacity reservations.

Table 2.2.: Path Based Survivability Classifications (Data from [12])

Category	Failure recovery route		Channel assignment on failure recovery route		Cross-connect on failure recovery route	Failure specific
	Computed	Assigned	Computed	Assigned		
DBPP	before	before	before	before	before	no
SBPP with pre-assigned channels	before	before	before	before	after	no
SBPP with non pre-assigned channels	before	before	after	after	after	yes (channel only)
SBPP with preplanned maps (routes and channels)	before	after	before	after	after	yes (route and channel)
SBPP with preplanned maps (routes only)	before	after	after	after	after	yes (route only)
Re-provisioning	after	after	after	after	after	yes (route and channel)
SBPP with pre-assigned backup paths and no reservation	before	after	after	after	after	no

2.1.3. Performance Metrics

When determining the general performance of a survivability technique, four criterion can be used. These properties are derived from the SLAs mentioned in section D.4 and physical properties of all networks. These properties are provided with a general description

in table 2.3. The first two metrics, restoration time and connection availability represent the requirements in SLAs for quick failure recovery. The second two metrics represent the economic problems of network survivability. These are the requirements of both simple and efficient solutions.

Table 2.3.: Basic Survivability Performance Metrics

Criteria	Definition
Restoration Time	Time required to restore the connection through the network
Availability	Percentage of time that a connection is available
Redundancy	Additional resources required to protect traffic
Complexity	Computation difficulty associated with the technique

2.1.3.1. Restoration Time

When a fault occurs along the working path of traffic demand r, the network must go through the restoration process to transfer the flow of data to the backup route. The restoration time is the period of time taken from when the the fault occurs to when the connection switches to the backup route. This time can be viewed as the amount of down-time experienced by a demand when a fault occurs. Thus it can be used to define where a survivability technique fits in the restoration target ranges depicted in figure D.1. In [11], a technique was presented for determining the restoration time for a demand. In this thesis, the number of hops along the reactive section of the backup route is used as a metric for restoration time. As a longer reactive section correlates with a longer restoration time.

2.1.3.2. Availability

The availability is a measure how often a system is in a functional state. Thus the availability can be described as the percentage of time a system is not experiencing a fault event. The availability can be calculated as a function of the mean time between failures (MTBF)

and mean time to repair (MTTR). The MTBF is the average time between failure events. It is measured in FITS, which is the number of failures experienced by a system in a billion hours. As mentioned earlier, optical cables experience a MTBF of 500 $^{FIT}/_{Mile}$. Likewise, the MTTR is the time required to repair a failure. As mentioned earlier, the MTTR for optical cables is around 5.2 hours. Using those two metrics, equation 2.1 can be used to determine the availability A_s of a system.

$$A_s = \frac{MTBF}{MTBF + MTTR} \tag{2.1}$$

Equation 2.1 is used to predetermine the availability of individual links in a network. Likewise, the availability of a traffic demand is the percentage of time the connection is operational. A connection's availability can also be predetermined with method depicted in equation 2.1. However, since the MTBF and MTTR for a connection are determined by the set of links traversed by the demand, it is easier to predetermine a connection's availability as a function of link unavailabilities. This is because the unavailability of a connection U_c is the sum of each link's unavailability U_j along the connection's route $R(r)$. Thus, the unavailability of a connection can be expressed as equation 2.2.

$$U_c = \sum_{j \in R(r)} U_j \tag{2.2}$$

Afterward, the availability of the connection A_c can be easily determined from the unavailability A_c using the relationship in equation 2.3.

$$A_c = 1 - U_c \tag{2.3}$$

By utilizing this methodology, an estimation of the connection availability for a demand can be generated when it is routed. Afterward, in order to determine the true connection availability over the life of a demand a different technique can be utilized. This technique

consists of simply comparing the amount of time a connection was in a failed state F_s with the time it was in an operation state O_s. This technique used to determine a connection's availability is summarized with equation 2.4.

$$A_S = \frac{O_s}{F_s + O_s} \tag{2.4}$$

According to their SLAs, connections are offered specific availability guarantees. These availabilities are usually offered using the nines notation. The nines notation is a method of segmenting availability into discrete levels that can be offered by service providers. As depicted in table 2.4, each nines category corresponds to a different quality of service. Service providers will use the previously mentioned method for predetermining the availability of a connection, in order to ensure that a traffic demand does not violate a SLA.

Table 2.4.: Nines Notation

Availability	Downtime
90% (1-nine)	36.5 $^{days}/_{year}$
99% (2-nines)	3.65 $^{days}/_{year}$
99.9% (3-nines)	8.76 $^{hours}/_{year}$
99.99% (4-nines)	52 $^{minutes}/_{year}$
99.999% (5-nines)	5 $^{minutes}/_{year}$
99.9999% (6-nines)	31 $^{seconds}/_{year}$

Due to the difficulty in obtaining the high levels of availability required in SLAs, service providers will use survivability techniques. These techniques allow the service provider to significantly reduce the combined failure rate of the connection. When survivability techniques are utilized, it is equivalent to adding a redundant system in parallel to the primary system. From this, a connection can be considered operational if either of the systems are available. Therefore the availability of the connection with path survivability

can be predetermined with equation 2.5, where U_w and U_b are the working and backup connection unavailabilities.

$$A_s = 1 - U_w U_b \tag{2.5}$$

2.1.3.3. Hop Requirement

The redundancy of a connection is a measure of how efficiently a survivability scheme protects information. As presented in equation 2.6, It is measured as the sum of the bandwidth $b_2(ij)$ reserved on a link $\{ij\}$ along the backup route $R_2(r)$ over the sum of bandwidth $b_1(ij)$ reserved of a link $\{ij\}$ along the working route $R_1(r)$.

$$R = \frac{\sum_{\{ij\} \in R_2(r)} b_2(ij)}{\sum_{\{ij\} \in R_1(r)} b_1(ij)} \tag{2.6}$$

When using the redundancy requirement as a performance metric, a degree of caution is required. Some path survivability techniques produce backup routes that are short in length compared with their working routes. However these schemes may also have unnecessarily long working paths. Therefore, even though the connections appear to have low redundancy requirement they in fact utilize more of the networks resources. In order to mitigate this issue, the hop requirement should be measured. The hop requirement can be calculated using equation 2.7. Since this metric provides a greater accuracy than the redundancy requirement, it will be used extensively to compare the results of each proposed algorithm.

$$H = \sum_{\{ij\} \in R_2(r)} b_2(ij) + \sum_{\{ij\} \in R_1(r)} b_1(ij) \tag{2.7}$$

2.1.4. Real Time Routing Concerns

If you want to route a connection request r through a network in real time, heuristic algorithms are required. There are many algorithms capable of creating paths through a

network. One such method is Dijkstra's algorithm, which has a popular following among practicing engineers [13]. It is a simple algorithm for solving the shortest path through a network that does not contain negative arcs. Due to its popularity, it has become the foundation to many complex algorithms, some of which will be mentioned in this section. When utilizing any shortest path algorithm in a survivable routing application, there are a few problems with creating working and backup paths.

In order to provision a demand r with a working path $R_1(r)$ and a backup path $R_2(r)$, at least two iterations of a shortest path algorithm are required. Since at least one iteration is required for each of the paths. However, if successive iteration techniques are utilized, then that number can be much larger. Unfortunately, determining $R_1(r)$ and $R_2(r)$ sequentially in simple two step procedures is suboptimal and prone to the trap topology problem [13].

2.1.4.1. Sub-optimality

Ideally, when determining the least cost $R_1(r)$ and $R_2(r)$ combination, the relationship in equation 2.8 should exist for the total cost $C(r)$ of the connection request. For the equation, $C_1(ij)$ and $C_2(ij)$ represent the cost of using a link $\{ij\}$ along the working $R_1(r)$ and backup routes $R_2(r)$ for a connection request r.

$$C(r) = min\left[\sum_{\{ij\}\in R_1(r)} C_1(ij) + \sum_{\{ij\}\in R_2(r)} C_2(ij)\right] \tag{2.8}$$

This unique situation where $R_1(r)$ and $R_2(r)$ are minimized together is known as the shortest pair. Unfortunately, utilizing a shortest path algorithm in a simple two-step approach might not be able to produce this optimal solution. For a simple two-step solution to be

optimal the relationship in equation 2.9 must be true.

$$min\left[\sum_{\{ij\}\in R_1(r)} C_1(ij)\right] + min\left[\sum_{\{ij\}\in R_2(r)} C_2(ij)\right] = min\left[\sum_{\{ij\}\in R_1(r)} C_1(ij) + \sum_{\{ij\}\in R_2(r)} C_2(ij)\right] \tag{2.9}$$

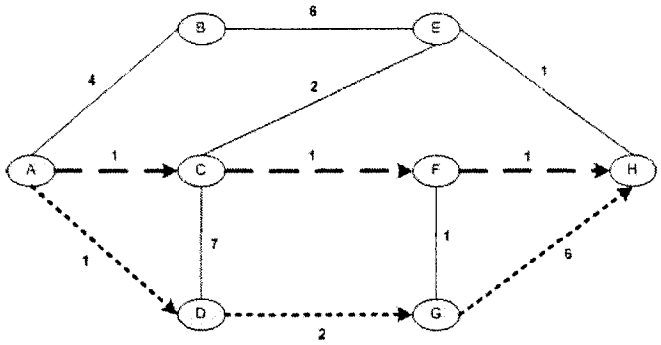
For example in figure 2.5, there is a simple network. In the network, a demand r wishes to route $R_1(r)$ and $R_2(r)$ from node A to H. For cost reduction reasons, the demand should be routed so that the relationship in 2.9 exists. In a simple two-step approach, $R_1(r)$ is routed according to equation 2.10 using any shortest path algorithm.

$$min\left[\sum_{\{ij\}\in R_1(r)} C_1(ij)\right] \tag{2.10}$$

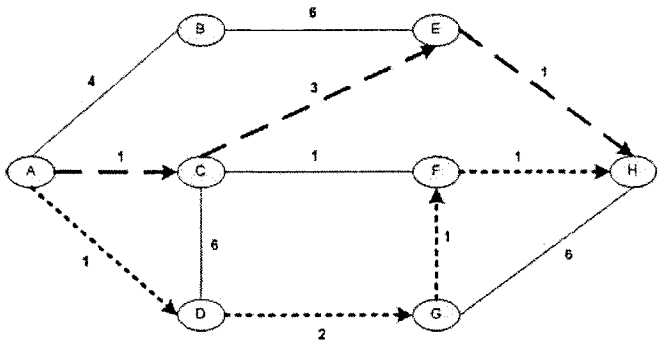
As depicted in figure 2.5a, this produces the path ACFH for $R_1(r)$. Following that, $R_2(r)$ is disjointly routed according to the backup path variant of equation 2.11.

$$min\left[\sum_{\{ij\}\in R_2(r)} C_2(ij)\right] \tag{2.11}$$

This produces the path ADGH and a total cost *C(r)* of 12. However, in this graph the actual minimum cost determined by equation 2.8 is 10 when $R_1(r)$ and $R_2(r)$ are routed over ACEH and ADGFH respectively. This scenario, depicted in figure 2.5b is in fact the shortest pair of paths available. This discrepancy in total costs between the optimal and simple two-step solutions is caused by $R_1(r)$ unfairly utilizing links that should be given to $R_2(r)$. Thus we can not rely on simple two-step algorithms to provide the shortest pair.



(a) Network with Both $R_1(r)$ and $R_2(r)$



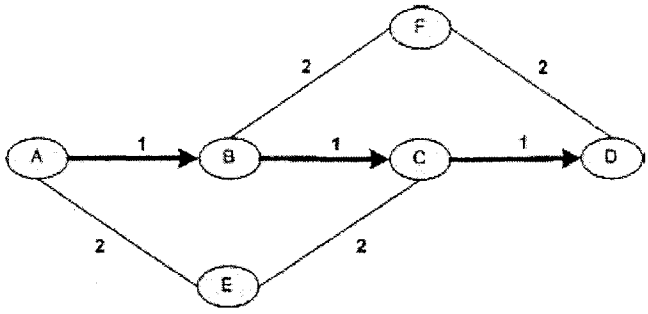
(b) Network With Shortest Pair $R_1(r)$ and $R_2(r)$

Figure 2.5.: Sub-Optimality of Two-Step Solutions

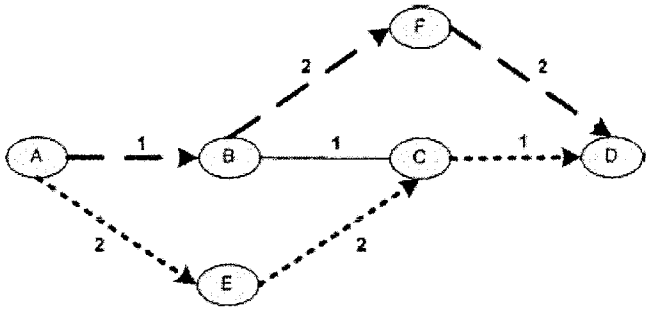
2.1.4.2. Trap Topology

In addition to the problem of sub-optimality associated with simple two-step algorithms, they are prone to the dreaded trap topology problem. In this problem, the shortest path for $R_1(r)$ will block the path for $R_2(r)$. This situation is best understood using figure 2.6. In figure 2.6a, $R_1(r)$ is routed using a shortest path algorithm and obtains a path of ABCD. Afterward, the network can not form a disjoint backup route $R_2(r)$ given that $R_1(r)$ has blocked its path. Therefore, it is a trap, and this demand must be declined. However, a set of paths can exist in the network if the connections are routed as they are in figure 2.6b.

This erroneous result produced by simple two-step algorithms could unnecessarily prevent customers from establishing connections through a network.



(a) Network with $R_1(r)$



(b) Shortest Pair for $R_1(r)$ and $R_2(r)$

Figure 2.6.: Trap Topology

2.1.4.3. One Step Solutions and their Limitations

To resolve the problems of suboptimal pairs and trap topologies a one-step path computation solution is required. There are two algorithms that are capable of performing one-step solutions. These algorithms are Suurballe's algorithm [14] and Bhandari's algorithm [13]. The former of the two algorithms relies on graph transformations to allow multiple iterations of Dijkstra's algorithm to determine the optimal shortest disjoint pair. The latter of the two uses multiple iterations of a modified variant of Dijkstra's algorithm introduced in

[13] that is capable of handling negative links to solve the problem. Due to the simplicity of Bhandari's algorithm which does not involve graph transformations, it is generally preferred over Suurballe's algorithm [2]. Due to its usage in this thesis, the Modified Dijkstra's algorithm and Bhandari's algorithm have been included in Appendix E as Algorithms E.2 and E.5. Additionally, if Bhandari's algorithm is run for k iterations, it can be used to resolve the k-shortest path set [13]. This variant has also been included in the reference section as algorithm E.6. Unfortunately, there are major limitations associated with both shortest pair algorithms and their variants. The link costs $C_1(ij)$ can not be arbitrarily manipulated between each iteration of the shortest path search used in the algorithms. This inhibits the establishment of sharing based link costs for backup routes. Adding to that, SBPP schemes need to know the working route $R_1(r)$ to determine link costs $C_2(ij)$ for the backup route $R_2(r)$. Since the shortest pair algorithms resolve both paths simultaneously, they can not be used for survivability techniques which require two steps. There have been attempts to find a middle ground between two-step and one-step techniques like the Iteration Restoration Dijkstra [15], but these solutions are suboptimal and require many iterations to complete. Many of the algorithms in this thesis generate shortest path sets using one step algorithms.

2.1.4.4. Re-optimized Solutions

Networks can provision connection requests using on-line algorithms. These algorithms are provided with information about the current state of the network. As more connection requests enter and leave the network, the routes utilized by the connections may become suboptimal. That is, with a change in the connections in the network, superior routing and sharing opportunities might become available. Re-optimization takes advantage of these opportunities by offering network administrators the option of readjusting the routes used

by those connections. Re-optimization can be performed on a regular basis, or as a one time event. This quality is of particular importance to survivability techniques that depend on sharing between specific demands that are not very common. An example of this is when sharing can only occur if two demands are disjoint and heading to the same destination. In this thesis one of the proposed algorithms utilizes re-optimization to improve its results.

2.2. Coded Survivability

At the turn of the millennium, the idea of linearly combining independant data connections was proposed to increase throughput [16]. Originally this idea was proposed to improve latency in satellite communications, however it was expanded to include mesh networks [17]. The potential of this technology has produced a wealth of literature on improving our networks through network coding [18, 19, 20].

Recently, the idea of using coding to provide survivability to networks has been proposed by the authors of [21]. Although originally proposed to work in conjunction with protection cycles, the concept has since branched off so that it can be used on its own to provide survivability [22]. In [23], the authors proposed a strategy for providing 1+N protection. This scheme was designed to provide fully proactive protection against single link failures for N connections using a single redundant path. The technique used coding to linearly combine all N connections onto a redundant circuit. Modulo two additions were specifically proposed as the method of choice to make the linear combinations. This method was therefore posed to provide survivability without the need for fault localization or signal rerouting. Thus, the problem of network survivability would be reduced to an error correction problem.

In [24], the authors proposed using network protection codes (NPC) so that the network

survivability could be reduced to an error correction coding issue. Therefore, we can represent every scheme with a generator matrix G. The generator matrix defines the sets of linear combinations sent over a set of disjoint connections. Each element in the matrix g_{ij} is a binary variable existing in $\mathbf{F}_2$ space that determines whether a traffic demand i is linearly combined onto disjoint connection j. In its simplest form, for a scheme to protect n working paths, k connections must carry uncoded data and $m = n-k$ connections must carry coded data. In this scheme, each source transmits a column vector $(\; g_{1j} \quad g_{2j} \quad \cdots \quad g_{(n-1)j} \;)^T$ in $\mathbf{F}_2^{n-1}$. For single link failures we can define a simplified generator matrix G that can be used to encode survivability for $n-1$ sources as in equation 2.12.

$$G = \begin{bmatrix} 1 & 0 & \cdots & 0 & 1 \\ 0 & 1 & \cdots & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 1 \end{bmatrix}_{(n-1)\times n} \tag{2.12}$$

The columns in the matrix consists of of a set of disjoint connections between the sources and receivers. The encoded connection consists of a linearly combined set of the data from each source node. When there is only one coded connection the graph looks like figure 2.7. This technique has been used successfully to generate the generalized 1+N protection scheme [23]. Later on, that work was expanded to protect against possible node failures in [25]. This was done by increasing the minimum required redundancy to be greater than the nodal degree. To further study coded protection, this thesis includes several novel heuristic algorithms. These algorithms were invented to provide coding based protection using heuristic algorithms

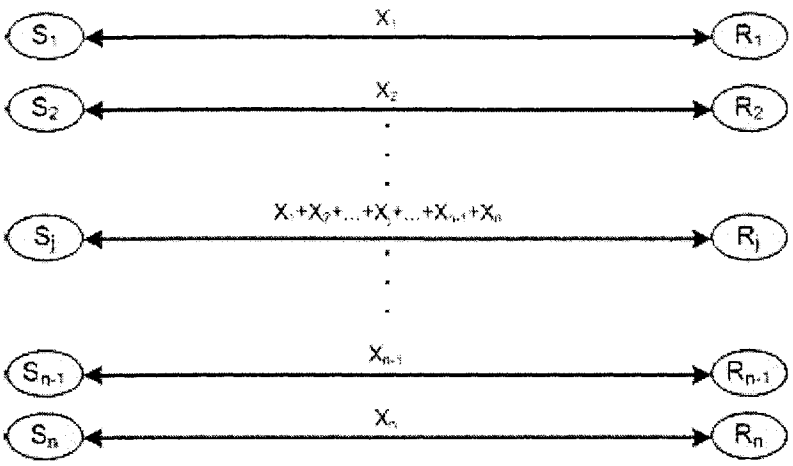
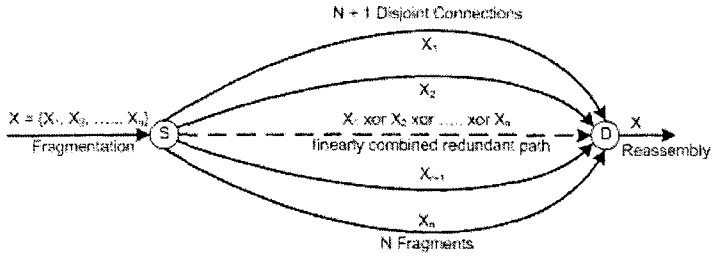


Figure 2.7.: Network Coded Protection Against Single Link Failures (from [24])

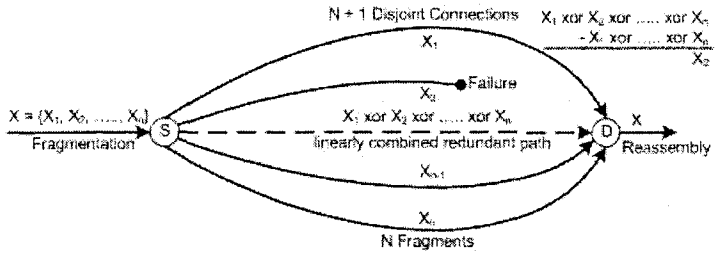
3. Proposed Coded Protection Schemes

3.1. Source Coded Protection

Source Coded Protection (SCP) is the simplest coded-based path survivability technique presented in this thesis. This survivability scheme attempts to emphasize the importance of utilizing a minimum amount of computations and control signaling to protect information. Normally, in order to minimize the amount of computations and signaling, a dedicated backup path protection (DBPP) technique is required. Since all the computations and signaling are performed before a fault event, no restoration computations or signaling are necessary. This would allow the network to remain as simple as possible and dedicated to fast and efficient information transfer. However, with DBPP comes inefficient capacity utilization. This added redundancy requirement has been the weapon for opponents to the technique. Fortunately, with coding based survivability, information can be fragmented by a source node, encoded together and sent disjointly to the destination. This leads to reduction in bandwidth requirements over DBPP. Figure 3.1 depicts the general operation of a connection request r protected by SCP.



(a) Normal Operation



(b) Failure Operation

Figure 3.1.: Source Coded Protection Appearance

Depicted in figure 3.1a are the normal working conditions for a demand X protected using SCP. Under normal conditions a demand X at a source node S is separated into N fragments. These fragments are sent disjointly over N working routes $R_w(r) = \{R_{w1}(r),\ R_{w2}(r)...,\ R_{wN}(r)\}$ to a destination node D. These working routes $R_w(r)$ are supplemented by a single disjoint backup route $R_b(r)$ containing a linear combination of all the fragments of demand X, i.e $X_1 \oplus X_2 \oplus .... \oplus X_n$. If a fault event occurs along one of the working routes $R_{wi}(r)$, the destination node will be able to decode the missing data using the redundant connection $R_b(r)$. For example, in figure 3.1b, fragment X_2 is lost due to a fault in the network. However, the destination still has enough information to decode X_2 by performing the following operation.

$$X_2 = (X_1 \oplus X_2 \oplus .... \oplus X_n) \oplus (X_1 \oplus X_3 \oplus .... \oplus X_n)$$

This allows the network survivability issue to be reduced to a simple error correcting code

problem. Since the fragmentation and reassembly are performed by the source and destination nodes, the intermediate nodes will not require any special computations or signaling procedures. As presented in table 3.1, from the path assignment and recovery perspective, SCP is identical to DBPP. Unlike DBPP that requires at least a 100% redundancy to protect traffic, SCP can provide survivability with a redundancy requirement as low as $\frac{1}{N}$, where N is the number of fragments. Assuming each path has a normalized cost of one, DBPP requires 2 units of network capacity, while SCP only requires $\frac{N+1}{N}$. However, the scheme is bounded by the min-cut between the source and destinations nodes. In the worst case situation, where the min-cut between two nodes is only two, SCPs performance is equal to DBPP. As the min-cut increases to 3 and 4, best case capacity utilization approaches to 1.33 and 1.25 respectively. In chapter 3, two novel heuristic algorithms have been designed to perform SCP. The first of these algorithms was designed to provide the most capacity efficient set of connections for SCP. The second algorithm, provides a much faster solution but with less efficient results.

Table 3.1.: Source Coded Protection Characteristics

Failure recovery route		Channel assignment on failure recovery route		Cross-connect on failure recovery route	Failure specific
Computed	Assigned	Computed	Assigned		
before	before	before	before	before	no

3.2. Multiple Source Coded Protection

Even with all the benefits of SCP, there are a few aspects of it that make it undesirable. First and foremost, the level of sharing is limited by the minimum nodal degree of the

source and destination nodes. If either the source or destination node has a degree of two, SCP is essentially DBPP. However, coded based protection has no requirement that forces it to allow sharing from only one source. By removing the single source constraint from SCP, this nodal degree problem can be partially mitigated, so that the amount of sharing is only dependent on the degree of the destination node. Adding to that, by removing the single source constraint of SCP, fragmentation becomes unnecessary, therefore removing the complexity associated with it. This new survivability technique is called Multiple Source Coded Protection (MSCP). As with SCP and DBPP, MSCP also performs all assignment operations before the failure occurs. Thus, as depicted in table 3.2, the failure recovery route, channel assignments and cross-connects are all computed in advance and independent of any failure.

Table 3.2.: Multiple Source Coded Protection Characteristics

Failure recovery route		Channel assignment on failure recovery route		Cross-connect on failure recovery route	Failure specific
Computed	Assigned	Computed	Assigned		
before	before	before	before	before	no

Due to the removal of the single source restriction, MSCP has a slightly different operation. Since source nodes can be distinct from one another, coding must be performed inside the network. In SCP a single coded route was relied upon to protect connections, in MSCP this can require a set of coded routes $R_{wj}(r)$. Figure 3.2 has been employed to further illustrate the operation of MSCP. In the figure, every source in a *protection set s* simultaneously sends two identical disjoint connections into the network. Within the network, these identical

connections are linearly combined onto separate coding streams $R_{wj}(r)$. Each of these routes are linearly independent, disjoint from one another, and stream converge on the destination node. In order to provide survivability to a protection set of n sources, $n+1$ coded routes are required.

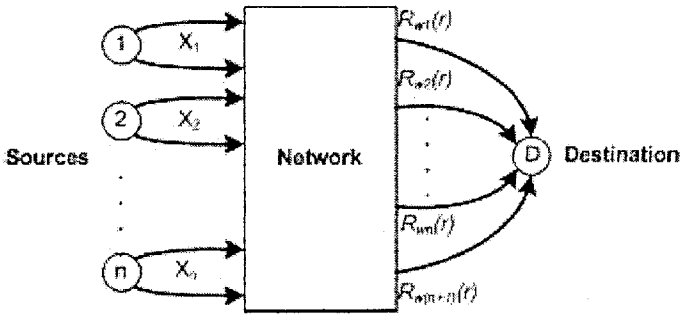


Figure 3.2.: Multiple Source Coded Protection Appearance

The destination node will receive the $n+1$ streams $R_{wj}(r)$. Each of these streams will have a header with the coding vector C_j in equation 3.1.

$$C_j = \left\{ \begin{array}{cccc} \alpha_{j1} & \alpha_{j2} & \cdots & \alpha_{jn} \end{array} \right\}_{1\times n} \tag{3.1}$$

The coding vector is an n-dimensional vector, used to determine which information sources have been linearly combined into stream j. For the purposes of providing survivability, each element $\alpha_{ij} \in \{0, 1\}$. That is, α_{ij} is a binary number, which determines whether a piece of information x_j exists in a coded route $R_{wi}(r)$. With this information from each coding vector, the destination node can generate the matrix in equation 3.2. This matrix contains a set of $n+1$ linear equations, each representing a disjoint stream $R_{wj}(r)$. Since the matrix has has $n+1$ equations and n unknowns it can be considered over-defined. Adding to that,

since every equation is linearly independent, the matrix is full rank. Thus, any set of n linear equations from the matrix can be used to solve the source messages.

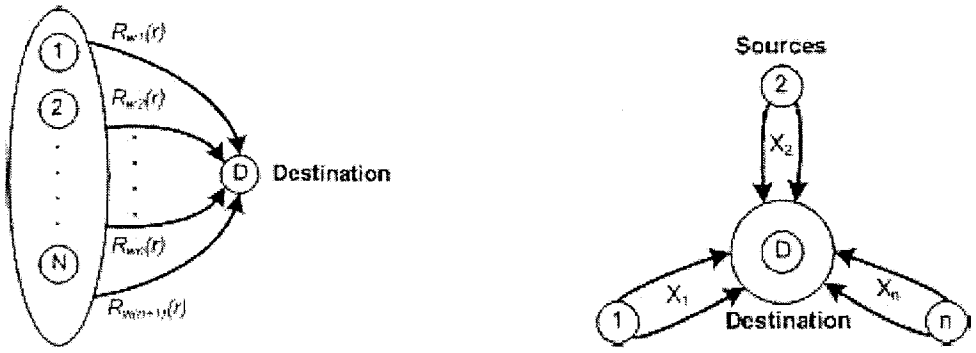
$$\left[\begin{array}{cccc|c} \alpha_{11}x_1 & \alpha_{12}x_2 & \cdots & \alpha_{1n}x_n & R_{w1}(r) \\ \alpha_{21}x_1 & \alpha_{22}x_2 & \cdots & \alpha_{2n}x_n & R_{w2}(r) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \alpha_{n1}x_1 & \alpha_{n2}x_2 & \cdots & \alpha_{nn}x_n & R_{wn}(r) \\ \alpha_{(n+1)1}x_1 & \alpha_{(n+1)2}x_2 & \cdots & \alpha_{(n+1)n}x_n & R_{w(n+1)}(r) \end{array}\right]_{(n+1)\times n} \tag{3.2}$$

In a single failure event, one link in the network will be disabled. Since every stream is disjoint, only one can be affected by the fault. Likewise, since each source sends two disjoint connections into the network, and each connection is linearly combined into different streams, no pair of connections can be lost. Therefore, the effect of the fault can be reduced to a loss of one linear equation at the destination. Since each linear equation is linearly independent, a failure will create a standard $n \times n$ or well-defined matrix. With the well-defined matrix, each source message can be solved as if no failure occurred.

Unlike SCP, the tree structures required for coded routes in MSCP require more capacity than point-to-point connections. However, these trees formations are required for encoding to occur. Thus it is important to minimize the collective size of these trees. For an n source node protection set, $2n$ disjoint branches are required to connect to $n+1$ coded routes. This number of disjoint branches required corresponds to a pair of disjoint connections from each source node. Simultaneously, the $n+1$ coded routes are required so that a full rank well-defined matrix will be maintained in any single failure event. The disjoint branch pair from each source node only needs to connect to two disjoint streams, since connecting to more will only increase capacity usage and offer no benefits. With that, MSCP requires at least one and at most $n-1$ coded routes. The number of coded routes is bounded at the high end when the branches from each source are paired onto the $n+1$ routes. That is, when each coded route contains at most two pieces of information. From that, at most

$n-1$ coded routes will be required for any protection set.

The capacity utilization of MSCP can be determined as a range bounded by SCP and DBPP. These bounds are depicted in figure 3.3. In figure 3.3a, if all the sources are collocated, then MSCP performs like SCP. In this instance, like SCP the normalized capacity utilization is $\frac{N+1}{N}$. On the other end, in figure 3.3b is the worst case scenario. In this scenario, the source nodes are farther away from each other than they are from the destination for both the working and backup routes. Therefore, it costs the same amount to send two disjoint connections to the destination as it would to linearly combine them. This situation is identical to DBPP and therefore has a capacity utilization of two.



(a) Most Efficient Scenario (b) Least Efficient Scenario

Figure 3.3.: Performance Bounds for MSCP

There are a few drawbacks associated with MSCP. The worst of these is the new linear combining requirement imposed on intermediate nodes in the network. Unlike in SCP, MSCP must have a complex transport layer, where connections must be buffered and linearly combined into streams. This requirement of transport networks will be exhibited

by the remaining network coding schemes. Fortunately, as network coding becomes more prominent in networks for its increased throughput property, this issue will become less important. In chapter 3, three novel heuristic algorithms have been created to perform MSCP. These algorithms will be explained in chapter 4.

3.3. Network Coded Protection

The previously proposed protection schemes all relied exclusively on coding to provide survivability to networks. SCP allowed a source to fragment a traffic demand into multiple connections that can be encoded together for survivability. MSCP expanded this coded attribute to allow many sources to share coded routes. Using their respective techniques, both of these schemes significantly reduced the redundancy requirement over DBPP. However, they have a fatal weakness that can not be solved with coding alone. Both schemes are dependent on nodal degrees. SCP is dependent on the nodal degree of the source and destination nodes, while MSCP is dependent only on the destination node. When sharing is limited in that fashion, high levels of sharing are improbable. The goal of Network Coded Protection (NCP) is to create a scheme that is capable of allowing sharing between connections that have different source and destination nodes. Table 3.3 summarizes the differences between each of the coded protection schemes presented in this thesis.

Table 3.3.: Difference Between Coded Protection Schemes

Scheme	Number of Sources	Number of Destinations
Source Coded Protection	One	One
Multiple Source Coded Protection	Many	One
Network Coded Protection	Many	Many

The single destination constraint was imposed on SCP and MSCP because a centralized location was required to decode each coded stream. NCP strives to decentralize this operation so that intermediate nodes in the network can contribute to decoding operations. If decoding operations are allowed within the network, then there are three general points where a connection can be fully decoded. Figure 3.4 depicts these three possible decoding points. The first condition is where decoding occurs at the source. This situation represents the uncoded connections reminiscent of the worst case scenario for MSCP and DBPP. Under the bottom condition, the stream is decoded at the destination node. This is how SCP and MSCP normally operate. In the figure there is a middle condition between the fully coded and uncoded scenarios. If a coded route can be decoded at some intermediate node in the network, then the single destination constraint can be relaxed.

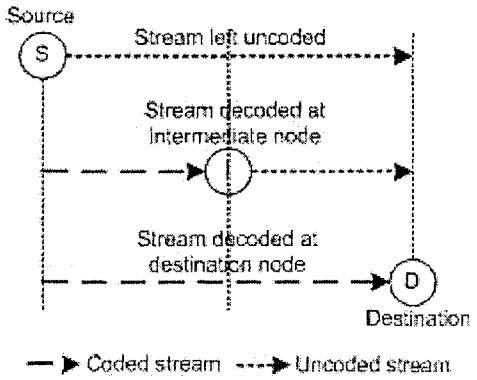


Figure 3.4.: The General Decoding Points for Coded Based Protection

This situation where an intermediate node decodes the coded stream is the premise behind NCP. In NCP, every traffic demand has its information decoded before or at the destination. For a set of traffic demands protected together in a protection set s, there is a node where all information must be decoded. This node is referred to as the *critical node C(s)* of

the protection set s. The critical node acts as the placeholder for the destination node in MSCP. Furthermore, in a coded route, the critical node separates the coded section from the uncoded section. In the coded section of a coded route, information can be encoded and decoded. However, in the uncoded section, since it is after the critical node, no coding is allowed. This organization for a coded route for a protection set is depicted in figure 3.5.

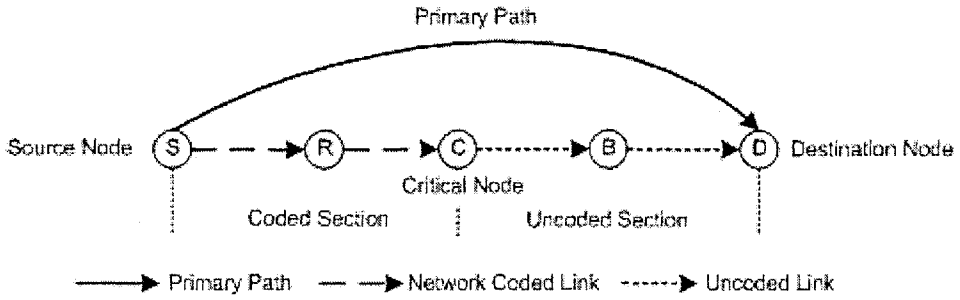


Figure 3.5.: Network Coded Protection General Layout

In order to help differentiate nodes before and after $C(s)$, the set of nodes before $C(s)$ shall be known as *root nodes* $R_n(s)$ and the nodes after shall be known as *branch nodes* $B_n(s)$. Adding to that, the set of links used to interconnect nodes in the coded section shall be known as *root links* $R_l(s)$ and the set of links used to interconnect nodes in the uncoded section shall be known as *branch links* $B_l(s)$. The root and branch distinctions have been given to nodes and links in a NCP stream because of their collective tree-like appearance. This general form is depicted in figure 3.6.

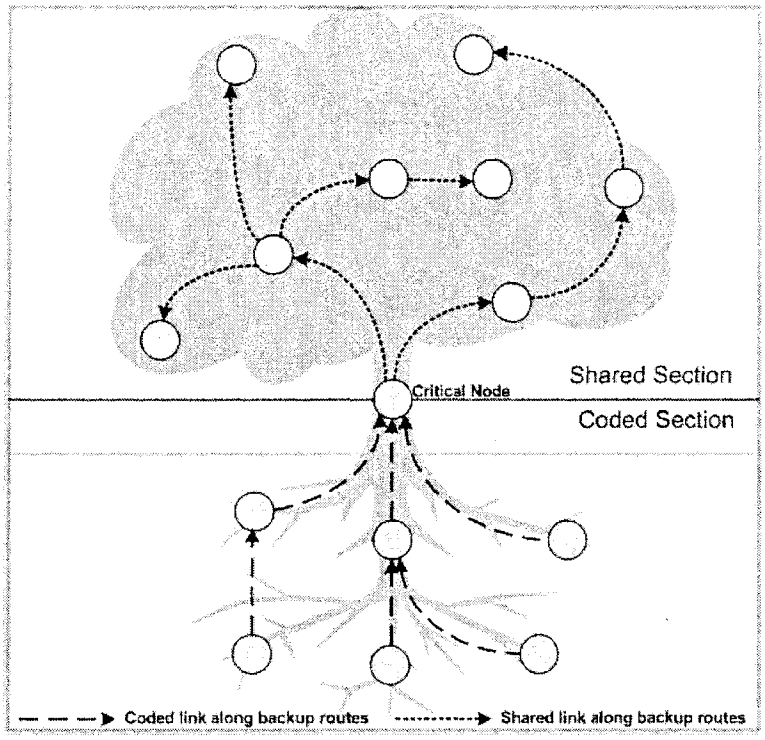


Figure 3.6.: Network Coded Tree Appearance

This tree depicts the general structure of a coded route for NCP. The coded section contains the root nodes $R_n(s)$ and links $R_l(s)$ of the protection set s. The roots of the tree defines the coded section. Each source for the traffic demands in the protection set connect to the coded route via the route nodes and links. Likewise, the uncoded or shared section is depicted as the branches of the tree. Each destination for the traffic demands in the protection set are connected to the coded route by the branch nodes and links. Branch links $B_l(s)$ can be either dedicated to an individual connection or shared between multiple demands in the protection set. For redundancy reduction reasons, this section should consist of shared capacity that can be used to route the last mile of a NCP demand in fault events. Thus, the uncoded section should be considered shared for NCP. In fault events, the failed connection will be forwarded from $C(s)$ to the destination node over $B_l(s)$. Since the cross-

connects setup is dependent on the failure type and the set of channels is dependent on the cross-connections, they should be determined after the failure occurs. As depicted in table 3.4, this gives network coded protection two separate protection characteristics. The coded section is failure independent and dedicated to all the connections in the protection set, while the shared section is basically a last mile variant of SBPP.

Table 3.4.: Network Coded Protection Characteristics

Section	Failure recovery route		Channel assignment on failure recovery route		Cross-connect on failure recovery route	Failure specific
	Computed	Assigned	Computed	Assigned		
Coded Section	before	before	before	before	before	no
Shared Section	before	before	after	after	after	yes

Obviously, to ensure single failure survivability, the working path $R_1(r)$ of each demand $r \in s$ should be disjoint from each other and from the set of root links $R_l(s)$ and branch links $B_l(s)$ in a protection set s. Fortunately though, there are no disjointness requirements between the set of root $R_l(s)$ and branch links $B_l(s)$. Three heuristic algorithms have been created to perform NCP. All of them will be presented in chapter 3.

3.3.1. Secondary Connections

As described for MSCP, in order to give a coded section the ability to decode any connection in a fault event, each stream requires two copies of every connection. However, the tree structure described in figure 3.7 only provides one connection to the coded section of the NCP protection set. The result depicted in figure 3.7, is a coded section that linearly

combines every connection onto a single channel. No decoding operations can be performed as long as the coded section only has one copy of every signal.

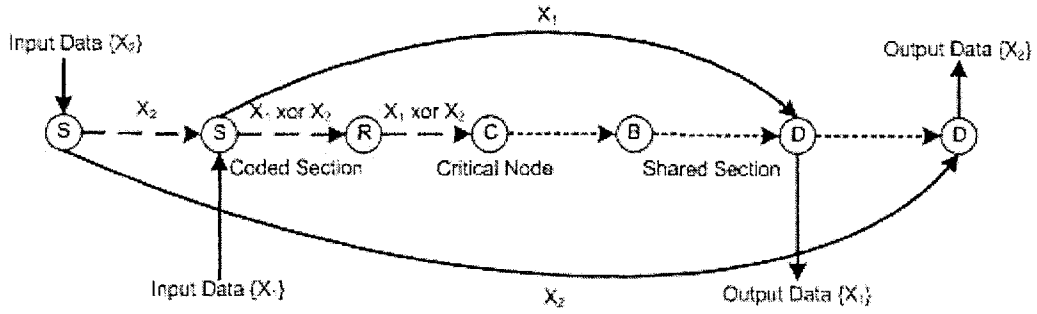


Figure 3.7.: Decoding Problem for Network Coded Protection

The solution is to provide a second connection from the working route $R_1(r)$ of each demand $r \in s$ to any node in the root section $R_n(s)$. The function of this secondary connection is to remove its demand's information from the stream. A simple exclusive OR operation can remove the redundant data from the coded section of the protection set. Under normal conditions, this will result in every connection being removed from the coded route before the critical node. The general appearance of the secondary connection is presented in figure 3.8.

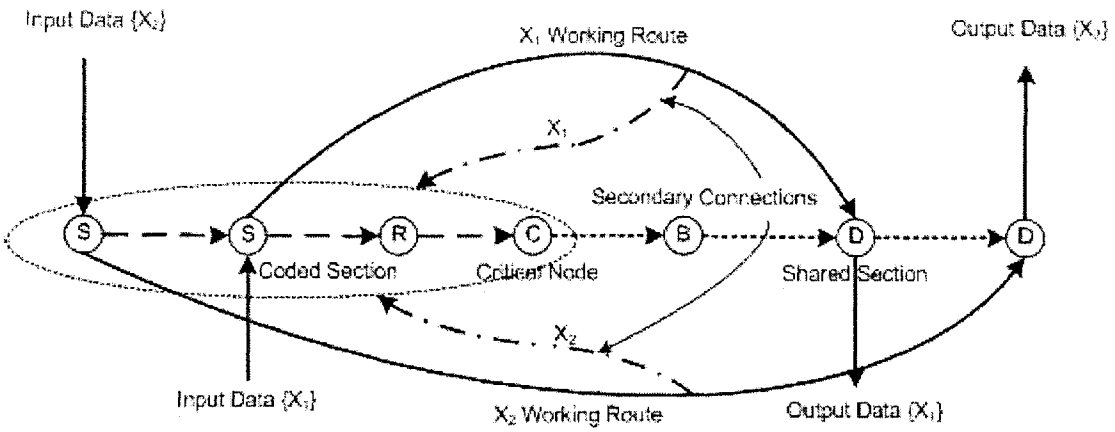
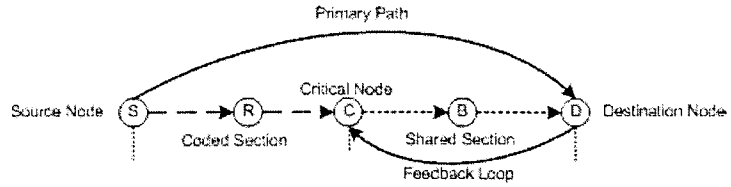


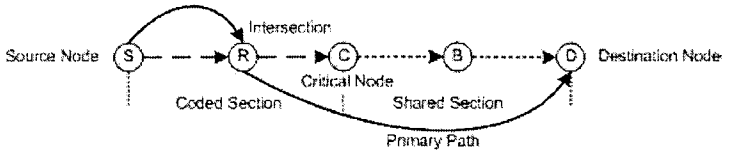
Figure 3.8.: Network Coded Protection Secondary Connection Solution

In a fault event, the secondary connection associated with the failed demand can be disabled, so that the failed demand's traffic can be forwarded through the shared section to the its destination. To do this only one working route in the protection set can fail at a time. Therefore disjointness must be enforced between working connections and the coded route. Since every connection except the failed connection will be removed from the coded section, the critical node $C(s)$ will be provided with a redundant copy of the information for that connection. Using the shared section, the critical node will establish a connection to the destination node. When the failed connection is repaired the secondary connection will be re-enabled. This will remove the redundant connection from the stream and disable the connection over the shared section.

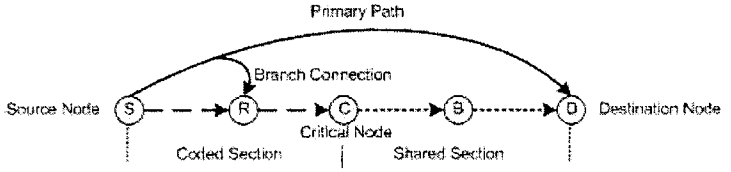
There are three methods available for creating secondary connections. As portrayed in figure 3.9, these techniques are known as the feedback loop, intersection arc, and branch connection. Each is designed to provide specific advantages over the other two.



(a) Feedback Loop



(b) Intersection Arc

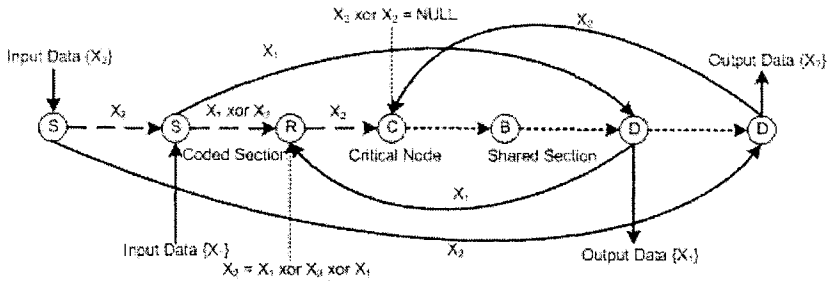


(c) Branch Connection

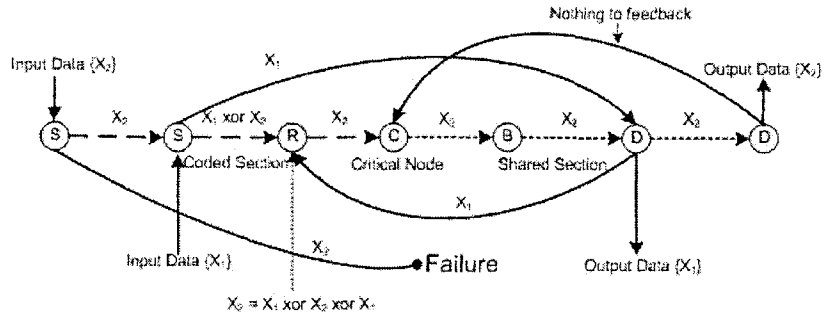
Figure 3.9.: Network Coded Protection Secondary Connection Types

3.3.1.1. Feedback Loop

The feedback loop is the simplest of all feedback types. It is an extension based solution to the secondary connection problem. In it, the working path is appended after the destination so that it connects to one of the root nodes of its protection set. Figure 3.10 has been employed to provide an example of the operation of a feedback loop.



(a) Normal Operation



(b) Failure Operation

Figure 3.10.: Network Coded Protection Feedback Loop Example

In figure 3.10, the normal operation and fault operation of the feedback loop are depicted. In it, two connections X_1 and X_2 are protected by a coded route with feedback connections. Under normal conditions, the connections are encoded and decoded in the coded section using the feedback loops. If a fault occurs along the working path of X_2, the feedback loop will have nothing to feedback. Thus, the coded section will not be able to remove X_2 from the coded stream. Therefore, at the critical node the only connection left will be X_2, which will be forward through the shared section to the destination.

As mentioned in table 3.5, the feedback loop has a very simple routing operation. Routing from the destination to the root nodes can be done with a simple modification to the end conditions of Dijkstra's algorithm. Adding to that, since the feedback loop is appended

to the end of the working route, the working route must be fully functional in order to maintain. Thus, in fault events the feedback loop can be automatically disabled at the destination node. Therefore, without signaling the coded route of the protection set, it will establish the redundant path to the destination node. In false positive events where the feedback loop fails but not the working route, a special update message can be sent to the coded route notifying it of the failure type.

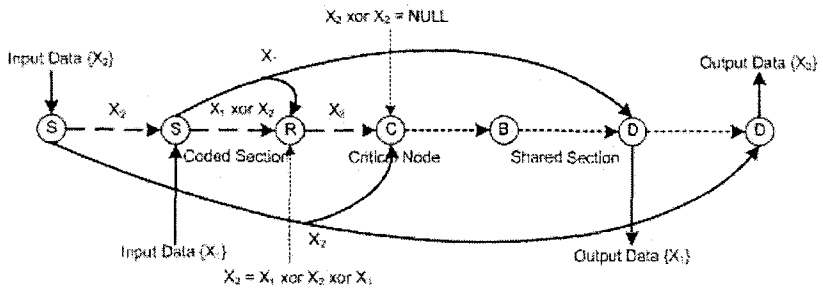
Table 3.5.: Feedback Loop Characteristics

Benefits	Drawbacks
Simple routing	Greatest redundancy requirement
Minimum protection signaling	Greatest delay

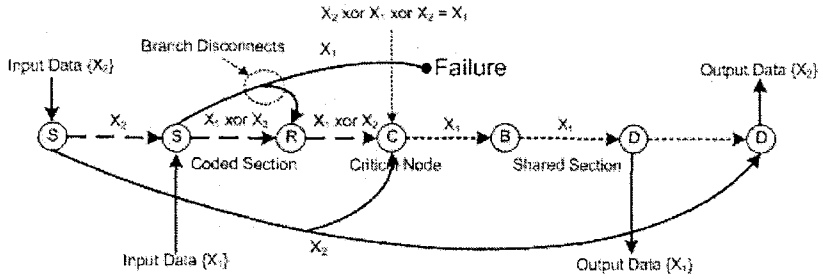
Unfortunately, a feedback loop can require a significant amount of redundant capacity. Furthermore, large delays can be accrued by routing a connection from first the source to destination and then to one of the root nodes. However, delays are a fundamental problem associated with network coding and we believe that as the technique becomes more mainstream, the problem will become less evident. Adding to that, since the feedback loop is only required to decode redundant data, it does not need to be disjoint from either the working path or protection stream. If the working path and feedback loop fail simultaneously, the stream can still establish a redundant path to the destination. Likewise, if the stream and the feedback loop fail simultaneously, the working connection for the demand will still be operational. Moreover, since the purpose of the feedback is to provide redundant data to the stream, other connections may utilize its route to linearly combine data into the stream. That is, the feedback loop essentially becomes part of the set of $R_n(s)$ and $R_l(s)$.

3.3.1.2. Branch Connection

The branch connection is a generalization of the feedback loop. Instead of forwarding connections to the coded section of the stream from the destination, the branch connection allows any intermediate node along the working route of the traffic demand can do it. Like the feedback loop, if the working path already intersects the coded section of the stream, no branch connection is required. In the case of a branch connection, an intersection can be interpreted as a zero length branch. Also like the feedback loop the branch connections can be added to $R_n(s)$ and $R_l(s)$ of the stream, so that other demands can use them too. Figure 3.11 has been employed as an example of the branch connection.



(a) Normal Operation



(b) Failure Operation

Figure 3.11.: Network Coded Protection Branch Connection Example

In figure 3.11 an example of the normal and fault operation of the branch connection is presented. During normal operation each demand provides a branch connection from their

working route to the coded section of the protection set. These branch connections are used to decode and remove each connection from the coded route. When a fault occurs, a signal will be sent from the node adjacent to the failure to the branch connection to disable it. With the branch connection disabled, the critical node will receive the redundant copy of the failed signal. The critical node will then forward the redundant connection through the shared section to the destination node. When the failure is repaired, the adjacent node will signal the branch to re-establish its connection. The branch connection will forward the working connection to the coded route, where it is used to decode the redundant data. Lastly, when the critical node loses its redundant information for the failed connection it will disable the shared section. Table 3.6 summarizes the benefits and drawbacks of using the branch connection.

Table 3.6.: Branch Connection Characteristics

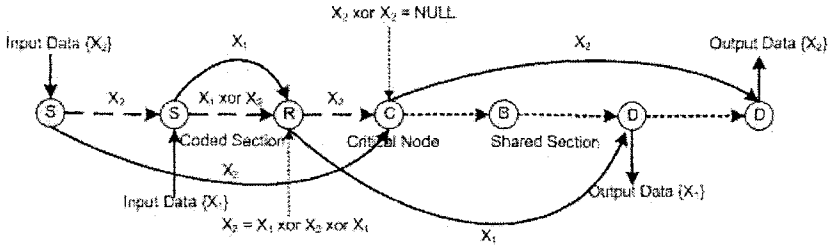
Benefits	Drawbacks
Least redundancy requirement	Greatest protection signaling requirement
	Difficult to route

Since the branch connection takes the most general approach to the secondary connection problem, it can produce the smallest redundancy requirement. In fact if the source is close enough to its destination a branch connection can be disregarded altogether. This scenario is where the branch occurs at the source node and results in the connection not using the coded route of the protection set until the failure occurs. Of course, this option is dependent on the restoration time and latency requirements of the SLA. However, the branch connection has a few drawbacks associated with it. Branch connections require significantly more signaling than feedback loops. In fact, if there is a branch from the

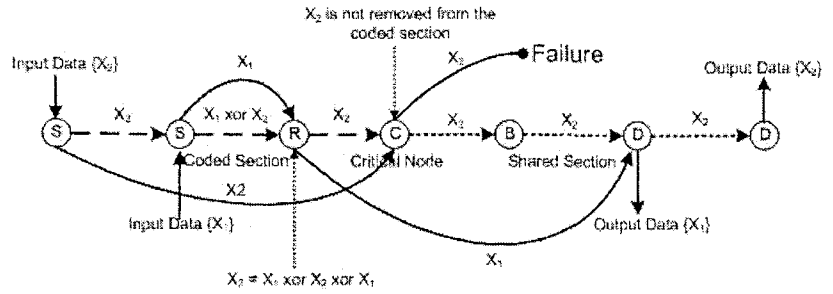
source node to the coded section, the restoration time will be comparable to some SBPP schemes. Adding to that, there is a degree of complexity associated with routing branch connections. In its most general sense, routing the branch connection is akin to routing a tree from the source node to the destination and one of the root nodes such that the capacity utilization is minimized and the delay is acceptable.

3.3.1.3. Intersection Arc

The intersection arc is a technique that forces the working path to provide the redundant data without using any feedback loops or branch connections. Ideologically it strives to create the ideal situation where the working path intersects the coded section of the protection set. Figure 3.12 has been employed to summarize the operation of the intersection arc on the coded route of a protection set.



(a) Normal Operation



(b) Failure Operation

Figure 3.12.: Network Coded Protection Intersection Arc Example

In figure 3.12a, under normal operation, the intersection point acts as a decoding point for the coded route. In the figure, two traffic demands X_1 and X_2 are both utilizing a coded route for protection. The redundant information for X_1 is connected to the coded section of the protection set, where it is linearly combined with demand X_2. Demand X_1 also has a working route that intersects a root node in the coded section before proceeding to its destination. At the intersection point, the information from the working route of demand $X_1$is used to decode its information from the coded section. As with all other secondary connections, this can be done with an exclusive OR operation. This event is allowed, since the working route of X_1 is disjoint from the working route of X_2 and the coded route.

Figure 3.12b depicts the failure operation for a network coded stream with intersection arcs. If a failure occurs across the working route of demand X_2 then, like a normal failure in path protected networks, the failure will first be detected by the nodes immediately adjacent the failure. The node on the source end will propagate a failure message to the intersection point with the coded route. At the intersection point the decoding operation will be canceled so that the redundant data is propagated to the critical node. At the critical node, the redundant data will be forwarded down the shared section to the destination. This reduces the restoration time since the error propagation only needs to travel to the intersection point and the backup path only needs to be setup from the critical node to the destination.

For the intersection arc, since no secondary connection is required then the additional redundancy and extra signaling are also not required. Unfortunately, by forcing the working route of a connection to intersect the coded route of a protection set, the working path can become significantly longer. This may affect the latency of connections, possibly violating

SLAs. Adding to that, routing a working path that intersects a coded route such that the minimum possible capacity is utilized is complex and difficult to route. These benefits and drawbacks of using the intersection arc are summarized in table 3.7.

Table 3.7.: Intersection Arc

Benefits	Drawbacks
No branches or feedback loops	Increased working path length
	Difficult to route

3.3.2. Coded Section Setup

The coded section layout is the most important aspect of network coded protection. The layout and critical node of a stream can dictate how capacity efficient it is and how quickly restoration can occur for demands in its protection set. Adding to that, if some nodes are not capable of performing coding operations, coded sections must be designed to not incorporate those nodes. Thus, the placement of the coded section of the stream is of extreme importance. There are generally two options available for positioning of coded sections for protection sets: *Pre-established Trunks* and *Demand Generated Formations*. Each technique takes a radically different approach to the placement of the coded section. The first technique utilizes predefined coded sections called *trunks*, which connections can utilize for protection. That is, potential coded sections are defined when the network is created. This provides network operators with the option of ensuring certain quality of service levels at the cost of efficiency. The latter technique relies on connections to generate the placement of the coded section with path search algorithms. This allows for efficient results but removes control from operators.

3.3.2.1. Pre-established Trunks

Pre-established Trunks are operator specified coded sections. They can be defined by the availability of network coding compatible nodes or through manual placement. When connections are added to the network, each will have to determine if it should be protected by a certain trunk. An example of this technique is in figure 3.13, where demands X_1 and X_2 want to be provisioned in the network. Demand X_1 wants to be routed from node B to node H. If it routes its working connection along BEH, then the trunk is available for protection. To solidify its protected status, the demand can route a disjoint backup connection along BC in order to connect to the trunk. Likewise, it can route a branch connection from its working route at node E to the trunk close to its destination at node H to remove its information under normal conditions. Finally, a shared backup route can be created from the critical node at node F to the demand's destination at node H to restore X_1 in fault events.

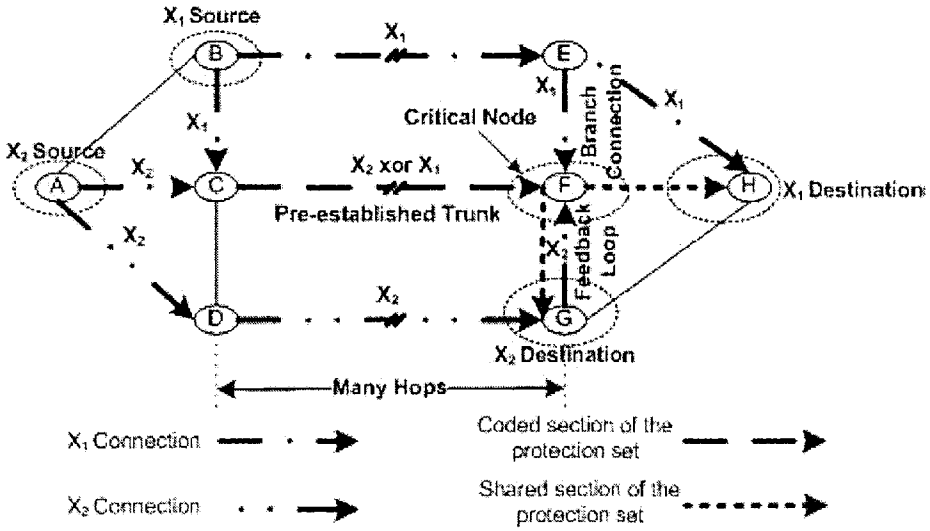
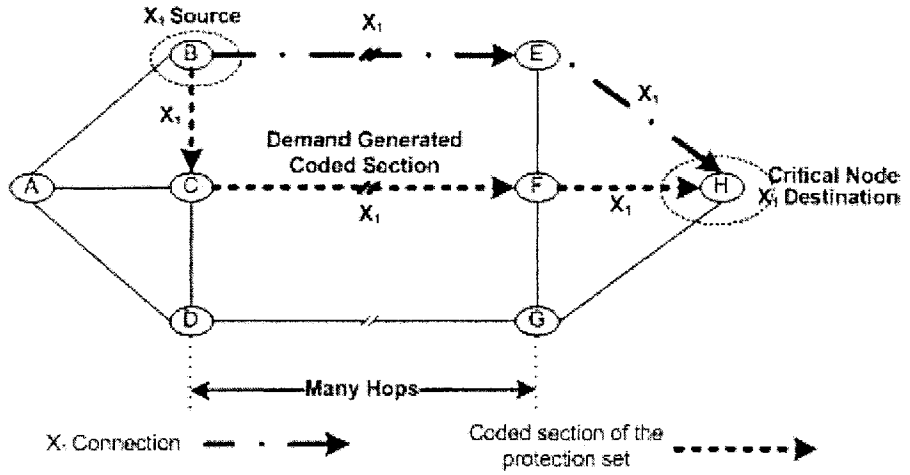


Figure 3.13.: Pre-Established Trunks

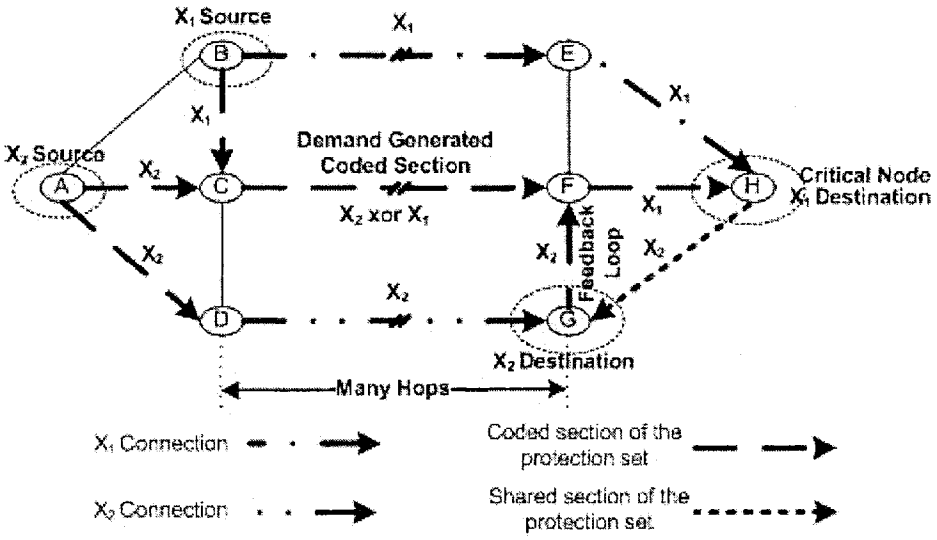
The second connection X_2 can be routed in the same fashion as the first. In order to get from its source node A to the destination at G it can follow the path ADG. Since no disjointness requirements are violated it can route a backup connection to the trunk from its source and from the critical node to its destination. A simple feedback loop can be used to provide the secondary connection to the trunk. In this instance, only nodes C and F require network coding functionality, yet overall capacity has been reduced and both connections can have one hop restorations from the critical node to their destinations.

3.3.2.2. Demand Generated Formations

Demand Generated Formations is a coded section creation method that closely resembles the techniques used by SCP and MSCP. That is, the network coded section is initially created from the backup route of a connection. This operation is summarized as figure 3.14a. In it, demand X_1 requires a connection from node B to node H. The primary route can be routed as normal with a shortest path algorithm. Afterward the demand needs to determine if it will use a pre-existing protection set or create a new one. The simplest method for determining if a connection should generate a protection set is to base the decision on the available existing protection sets. If there are no efficient protection sets available, then a new one should be created. The critical node can be set as the destination node or the highest degree node along the backup route. This setup requires that the entire network is capable of performing coding operations. Under conditions where only a subset of the intermediate nodes can perform coding, those nodes can perform all the linear combining operations. The other nodes in the coded route will only forward the coded information closer to the critical node. However, the critical node must be coding compatible or else the decoding process will not function correctly.



(a) Initial Stream Creation



(b) Second Connection

Figure 3.14.: Demand Generated Formations

As in figure 3.14b, when a second demand X_2 enters the network it will be given the option of joining the NCP protection set generated from demand X_1. There are four operations that must be completed in order for X_2 to join a protection set already containing X_1.

1. The working route for X_2 must be disjoint from the working route for X_1 and the coded route of the protection set.
2. X_2 must be able to route from its source node to the coded section of the protection set via a connection disjoint from its working route.
3. A route must be available from the critical node of the protection set to the destination of traffic demand X_2.
4. A route must be available from the destination of X_2 to the coded section of the protection set.

In figure 3.14b the backup route for traffic demand X_2 is provisioned using the four operations. First of all, X_2 has a working route provisioned from its source at node A to its destination at node G along the route ADG. Since, Its is disjoint from the coded route BCFH and the working route of traffic demand X_1, X_2 can join the protection set. In order to join this protection set, a backup route is routed from the source at node A to the coded route of the protection set at node C. Likewise, a connection is routed from the critical node of the protection set at node H to the destination of X_2 at node G. Finally, a secondary connection is required from the working route to the coded section. For this example, X_2 uses a feedback loop from its destination at node G to the coded section of the protection set at node F.

4. Proposed Coded Protection Algorithms

4.1. Source Coded Protection Heuristic Algorithms

In this thesis we are proposing two SCP based heuristic algorithms. These algorithms are *Near Optimal Source Coded Protection* (NOSCP) and *Fast Source Coded Protection* (FSCP). As specified in table 4.1, these techniques represent the trade offs between speed and efficiency. NOSCP produces an almost optimal set of paths for SCP. However, this can require many iterations of a shortest path algorithm. Conversely, FSCP has been created to establish a set of paths for SCP in a fraction of the time required by NOSCP. However, the results generated by the technique can be less optimal if the available capacity in the network is limited.

Table 4.1.: Source Coded Protection V. Fast Source Coded Protection

Technique	Benefit	Drawback
Optimal Source Coded Protection	Provides the optimal set of paths for a connection	Requires many iterations to complete
Fast Source Coded Protection	Reduces the number of iterations	Provides suboptimal paths for the connection

The problem with the heuristic algorithms for SCP is determining the optimal number of fragments that should be created in order to minimize capacity. Conventional wisdom would dictate that as the number of fragments is increased the capacity should decrease. However, increasing the number of fragments does not necessarily decrease the capacity

required for the demand. This issue can be explained with table 4.2. For a SCP connection fragmented into N pieces, the capacity utilized is $\frac{(N+1)}{N}A_{N+1}$, Where A_{N+1} is the average capacity requirement for each of the $N+1$ paths. Since shorter routes are always favored over longer ones, as the number of fragments is increased, the average hops per path also increases. Thus, there is a point where the benefit from fragmenting is negated by the increase in path length. At this point it is no longer beneficial to increase fragmentation. This point can be determined with equation 4.1. If equation 4.1 is true, then it is more capacity efficient to use $N-1$ fragments with N paths instead of N fragments with $N+1$ paths.

$$\frac{(N+1)}{N}A_{N+1} > \frac{N}{(N-1)}A_N \tag{4.1}$$

However, in the case of SCP algorithms, it is possible that $A_{N+1} < A_N$ after the point specified by equation 4.1. Since the size of a fragment is inversely proportional to the number of fragments. Therefore, as fragments increase, capacity requirements on each link decrease. Thus, the available links might increase, freeing up a shorter path set. This event occurs when the available capacity A_{ij} on a link $\{ij\}$ can be bounded by equation 4.2 for a bandwidth request b. In equation 4.2, the bounds are created by an increase in fragmentation from *N-1* pieces to N pieces. This circumstance may be able to generate the situation after the point specified by equation 4.1 where $A_{N+1} < A_N$.

$$\frac{b}{N} \leq A_{ij} < \frac{b}{(N-1)} \tag{4.2}$$

Fortunately, this situation only occurs when a network is heavily congested and rarely affects the relationship between A_{N+1} and A_N in the drastic ways specified by equation 4.2. Thus, in order to allow a SCP based algorithm to solve in a reasonable degree of time we let equation 4.1 decide the endpoint. That is, when the relationship in equation 4.1 exists then a SCP algorithm should end and provision the demand with the N paths

created in the previous iteration.

Table 4.2.: Capacity Requirements for Source Coded Protection

Number of fragments	Size of fragment	Number of paths	Average capacity required per path	Capacity utilized
2	$\frac{1}{2}$	3	A_3	$\frac{3}{2}A_3$
3	$\frac{1}{3}$	4	A_4	$\frac{4}{3}A_4$
N	$\frac{1}{N}$	$N+1$	A_{N+1}	$\frac{(N+1)}{N}A_{N+1}$

Unfortunately, discarding the last iteration means that a set of $N+1$ paths must be created only to be discarded, because it is less capacity efficient than the N paths set. This problem can be partially mitigated by determining in advance when $N+1$ disjoint paths is impossible and therefore unnecessary to calculate. This can be done with a min-cut max-flow check. As mentioned in section ??, the min-cut can be used to determine the maximum number of disjoint paths available between a source destination pair. For SCP, $N+1$ disjoint paths are required for a demand subdivided into N fragments. Therefore, we can bound the maximum number of paths set for SCP by the min-cut between the source and destination. However, determining the true min-cut of a graph is computationally significant and may mitigate the benefits of using the min-cut max-flow theorem. Thus we are using a pseudo min-cut determinator technique. This method is presented in section C.1 of appendix C.

4.1.1. Near Optimal Source Coded Protection

As mentioned earlier *Near Optimal Source Coded Protection* (NOSCP) strives to generate a near optimal set of paths for SCP. In order to determine the number of paths required in the near optimal set of connections we have to generate all the path sets until the relationship

in equation 4.1 exists. As mentioned in section 2.1.4, if a simple step by step approach is taken to discovering the path set, it will be suboptimal and prone to the trap topology. This is unacceptable for SCP, since high diversity is a fundamental requirement to achieve the benefits of the scheme. Thus, a shortest path set algorithm is required. Therefore we inherit the inflexibility of link costs and additional computation time associated with the shortest path set algorithms.

Since the availability of links changes with fragmentation, shortest path set algorithms must be rerun whenever fragmentation increases. For NOSCP, each time fragmentation increases, a shortest path set algorithm is run for the new set of paths. NOSCP can be best explained with the flowchart in figure 4.1.

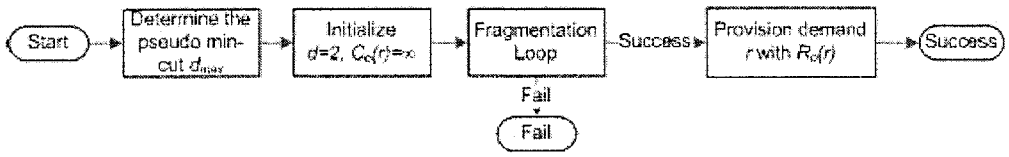


Figure 4.1.: Near Optimal Source Coded Protection Flowchart

In NOSCP, when a connection request r enters a network protected with NOSCP, the following procedure will be performed. First, the pseudo min-cut will be determined for the source-destination pair. This provides the algorithm with an upper bound for the number of paths possible with the algorithm. Afterward, the algorithm is setup to the starting parameters. For any SCP based algorithm to function correctly, it requires a min-cut of at least three. If the min-cut between a source-destination pair is two then coded fragmentation is not possible. Thus, NOSCP must decline connection requests between source-destination pairs with a min-cut of two. If a min-cut of at least three is available

then the algorithm will begin the path finding operation. The algorithm is designed to increase fragmentation in a looped fashion until the endpoint cost function in equation 4.1 becomes true. Figure 4.2 depicts the flowchart for the fragmentation loop in NOSCP.

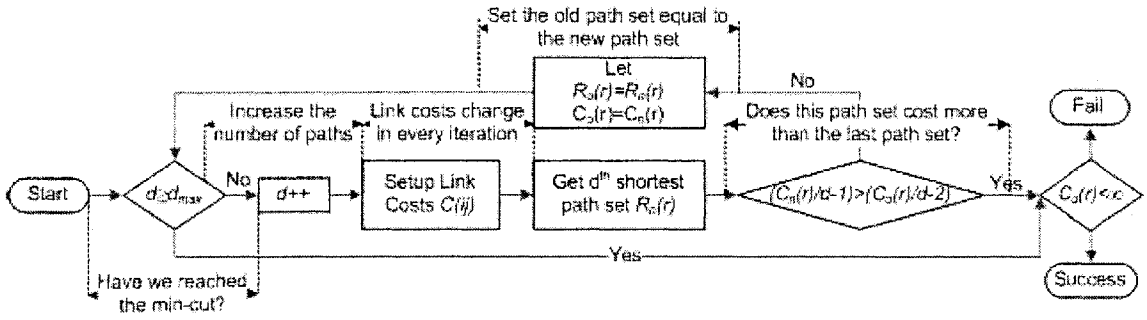


Figure 4.2.: Near Optimal Source Coded Protection Fragmentation Loop Flowchart

Inside every iteration of the fragmentation loop the link costs are assigned and a new shortest path set is calculated. The cost function in each iteration can be defined with equation 4.3, where d is the number of disjoint paths being generated in that iteration. The cost of a link for routing is only dependent on whether a fragment can fit inside the available capacity on the link.

$$C(ij) = \begin{cases} \infty & \frac{b}{d-1} > A_{ij} \\ \epsilon & \frac{b}{d-1} \leq A_{ij} \end{cases} \tag{4.3}$$

Initially, in the first iteration with two fragments, each is $\frac{1}{2}$ the bandwidth b required. If there is enough capacity on the link to provision the connection request, the link cost $C(ij)$ will be set to some small number ε, otherwise it will be set to ∞. After the link costs are setup, the algorithm will run the shortest path set algorithm for three routes in the first iteration. The number of paths d generated by the algorithm is incremented

in each iteration. For the first iteration, two routes will be used for the fragments and one will be used as the protection path. As the number of paths d increases so does the number of fragments. For every level of fragmentation only one path is devoted to coded information. At the end of every iteration, the endpoint cost function in equation 4.1 is used to determine if the cost $C_n(r)$ of the new path set $R_n(r)$ is less than the cost $C_o(r)$ of the previous iterations path set $R_o(r)$. For computational reasons, the endpoint cost function in the algorithm is modified into equation 4.4. It is different from equation 4.1 because the original endpoint cost function is based on the number of fragments and the average path length, which is good for explaining the concept but inefficient for heuristic algorithms. The endpoint function used in the algorithm instead uses the combined cost of all the routes on the new path set $C_n(r)$ and old path set $C_o(r)$. It also uses the number of disjoint paths instead of the number of fragments. Functionally though, both equations are equal.

$$\frac{C_n(r)}{d-1} > \frac{C_o(r)}{d-2} \tag{4.4}$$

If the costs are equal in equation 4.4, the path set with more routes is favored because it will distribute the capacity more evenly throughout the network. If the shortest path set algorithm fails to find $R_n(r)$, its corresponding cost $C_n(r)$ will be ∞. Since the initial old path set cost $C_o(r)$ is set to ∞ the endpoint cost function will ensure that the algorithm does not end prematurely without a path set. Thus for any iteration of the algorithm, if no acceptable path set has been found thus far, it will always allow another iteration to occur with increased fragmentation. If the endpoint cost function is true, then the algorithm will check if a path set is available. The easiest way to determine this is with the cost of the previous iteration $C_o(r)$. If $C_o(r) = \infty$ then no good path set has been found and the connection request should be rejected. Otherwise the connection should provisioned on the set of routes defined by $R_o(r)$.

For a connection that should be fragmented into N pieces, NOSCP requires up to $\frac{N^2+5N}{2}$ iterations of a Dijkstra's algorithm to complete. In order to mitigate some of the computation time requirements, the pseudo min-cut described earlier is utilized. If the optimal set of paths for a SCP demand has N fragments with the pseudo min-cut of $N+1$, the number of Dijkstra iterations required will be reduced to $\frac{N^2+3N-4}{2}$. The NOSCP algorithm is depicted in algorithmic form as algorithm A.1 in appendix A.

4.1.2. Fast Source Coded Protection

NOSCP provided an efficient set of paths for a SCP demand. However, it required up to $\frac{N^2+5N}{2}$ iterations of Dijkstra's algorithm to solve the path set. Therefore, a SCP demand with an optimum set of three fragments could require up to twelve iterations to complete. This is because the algorithm may have to generate the shortest triplet, quadruplet, and quintuplet before finishing. The problem was partially mitigated with the pseudo min-cut so that NOSCP would require only seven iterations if the min-cut was reached with the optimum set. This corresponds to the situation where the NOSCP algorithm only generates the shortest triplet and quadruplet. We were forced to do this many operations because the link costs $C(ij)$ change when the amount of fragmentation increases. If the algorithm were to attempt to get the shortest set with changing link costs, negative cycles could destabilize it. Thus, for NOSCP, the shortest path set must be reset whenever fragmentation increased. If the link costs are not changed when fragmentation increases, the algorithm doesn't have to reset the shortest path set algorithm. This reduces the required number of iterations in the worst case scenario from $\frac{N^2+5N}{2}$ to $N+2$. Modifying the algorithm in this regard reduces the number of iterations of Dijkstra in the previous example with three fragments from twelve to five iterations without the benefits of the pseudo min-cut,

and seven to four iterations with the pseudo min-cut. Since four routes are required for a SCP demand with three fragments, this is the minimum number of iterations possible for generating that many paths.

This novel heuristic algorithm for providing SCP is known as *Fast Source Coded Protection* (FSCP). In it the link costs $C(ij)$ are only calculated once instead of each time the fragmentation increases. Thus, the Algorithm can use the path sets of the previous iterations while generating the next path. The operation of FSCP has been summarized as a flowchart in figure 4.3.

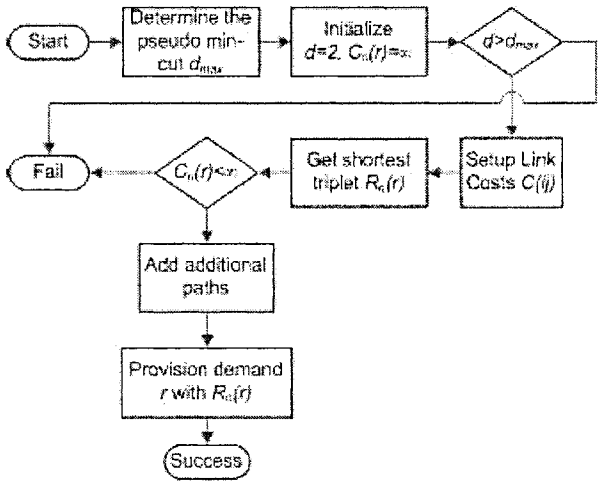


Figure 4.3.: Fast Source Coded Protection Flowchart

Functionally, FSCP is very similar to the NOSCP algorithm. Both use the same endpoint cost function and pseudo min-cut technique. However, they differ in respect to how the internal loops are performed. In NOSCP, each iteration consists of a link cost update and a K-shortest path set algorithm. FSCP is completely different in this respect, since its

internal loop only adds a path to the previously created set. In order for this path to generate a shortest set each iteration also incorporated features of the shortest path set algorithms. Initially, the FSCP algorithm starts off with a check if SCP is possible for the source-destination pair, using the pseudo min-cut. If three disjoint connections are not possible, then SCP can not be created and the connection request must be denied. If SCP is possible, then the link costs are setup for the first and only time with a fragmentation of two. If fragmentation increases afterward, the benefits of recalculating the link costs are ignored. Fortunately, the loss of these benefits are only evident when the network is congested.

After setting the link costs, FSCP gets the shortest triplet using the K Shortest Path Algorithm. This technique is reminiscent of NOSCP, which also initially gets the shortest triplet. However, in NOSCP this triplet may be discarded if more iterations are allowed with increased fragmentation. FSCP utilizes this triplet as the foundation for the internal loop iterations. This creates the major difference between NOSCP and FSCP. In NOSCP if the initial triplet calculation fails, the algorithm can look for a four disjoint path solution with increased fragmentation. In FSCP, since link costs are not affected by fragmentation, if the shortest triplet fails, the connection request is declined. Fortunately, this difference is only noticeable when the network is congested. Afterward, the algorithm enters the internal loop depicted as a flowchart in figure 4.4.

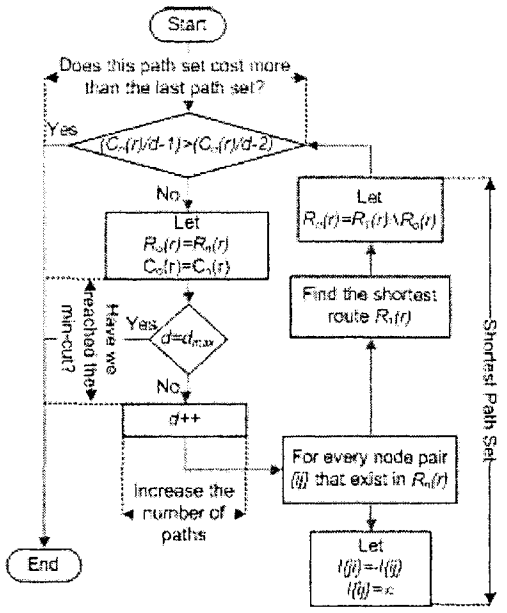


Figure 4.4.: Fast Source Coded Protection Internal Loop Flowchart

Inside the internal loop flowchart, the endpoint cost function described in equation 4.4 is used to determine when the algorithm finishes. After, at the beginning of each iteration the pseudo min-cut determines if another path is possible before the next path is calculated. Following that, the fragmentation is increased and the $d-1$ shortest path set is converted into the d shortest path set. This structure is basically the same as the internal loop structure of the K-shortest path algorithm depicted as algorithm E.6. That is, the internal loop follows the same three major steps as a loop in the K-shortest path algorithm.

1. The existing path is modified with shortest path set characteristics.
2. An additional route is generated.
3. The two routes are merged to create a new shortest path set.

For additional information on the K-shortest path set algorithm, please refer to [13]. FSCP is depicted in algorithmic format as algorithm A.2 in appendix A.

4.2. Multiple Source Coded Protection Heuristic Algorithms

Presented in this section are three algorithms for performing MSCP. These algorithms are *Re-optimized Multiple Source Coded Protection* (RMSCP), *Fast Multiple Source Coded Protection* (FMSCP), and *Single Stream Multiple Source Coded Protection* (SSMSCP). As described in table 4.3, The first technique has been designed to take full advantage of the possibilities available for coding multiple connections together. It strives to produce an near optimal set of routes connecting multiple demands from different sources to a single destination. It does this by re-optimizating pre-existing traffic demands. The second algorithm, provides a simplified approach by modifying the available destinations for the backup route based on previous paths in the protection set. By utilizing this method re-optimization can be avoided. This drastically reduces the overall complexity of the routing algorithm. However, without re-optimization the capacity efficiency of FMSCP is significantly reduced. This problem is slightly mitigated by the use of shortest path set characteristics for the working and backup connections in FMSCP. As opposed to RMSCP and FMSCP, which generate a generalized form of MSCP, the third algorithm, SSMSCP creates a specific variety of the MSCP scheme. Instead of allowing coding on any of the routes in a protection set, SSMSCP only allows a single route in the protection set contain coded data. This generates a MSCP protection scheme that appears like SCP to the destination. That is, in SSMSCP the destination will receive only one coded path containing the linear combination of all the traffic demands in the protection set. This reduces the amount of decoding required at the destination at the cost a of a greater capacity usage. The following three sections will explain in detail the operation of these heuristic algorithms.

Table 4.3.: Re-optimized Multiple Source Coded Protection V. Single Stream Multiple Source Coded Protection

Technique	Benefit	Drawback
Re-optimized Multiple Source Coded Protection	Greatest capacity efficiency	Requires re-optimization
Fast Multiple Source Coded Protection	Reduced routing complexity	Not as capacity efficient or easy to decode at the destination
Single Stream Multiple Source Coded Protection	Minimal decoding required at the destination	Least capacity efficient

4.2.1. Re-optimized Multiple Source Coded Protection

Re-optimized Multiple Source Coded Protection (RMSCP) is a heuristic algorithm for providing the generalized form of MSCP. That is, it aspires to allow N connection requests to encode their data together onto $N+1$ disjoint connections. For these N connections to encode together they must share a destination with a nodal degree of $N+1$. Since this algorithm re-establishes paths for existing demands when a new one is added to the network, it requires re-optimization. That is, all the routes chosen for a set of demands must be reconfigured, so that they perform MSCP while using the minimum amount of bandwidth. The initial problem with re-optimization is the decision of which demands should be optimized together. For standard shared mesh techniques this poses a great problem for re-optimization. There are limitless combinations of demands available for re-optimization. Determining an efficient set will be a computationally complex task. Fortunately, since MSCP requires that demands travel to the same destination for coding to occur, it limits the available options to an acceptable level. Like in SCP schemes, the min-cut theorem is important for providing sharing in RMSCP. However, Since sharing in RMSCP is only dependent on the nodal degree on the destination node, the previously described method

of determining the min-cut requires a slight modification. As mentioned earlier, the pseudo min-cut used the nodal degree of the source and destination nodes to determine the min-cut. Since RMSCP utilizes multiple source nodes, it is not dependent on source nodal degrees. Therefore, the pseudo min-cut for RMSCP should only be based on the nodal degree of the destination node. Thus it can be described by equation 4.5, where $|\Gamma_D|$ is the nodal degree of the destination node.

$$d_{max} = |\Gamma_D| \tag{4.5}$$

The flowchart in figure 4.5 has been employed to explain the routing operation for RMSCP. Due to the complexity of RMSCP, several of the blocks in the flowchart can be further expanded into additional flowcharts. These flowcharts will be presented as the RMSCP flowchart is explained.

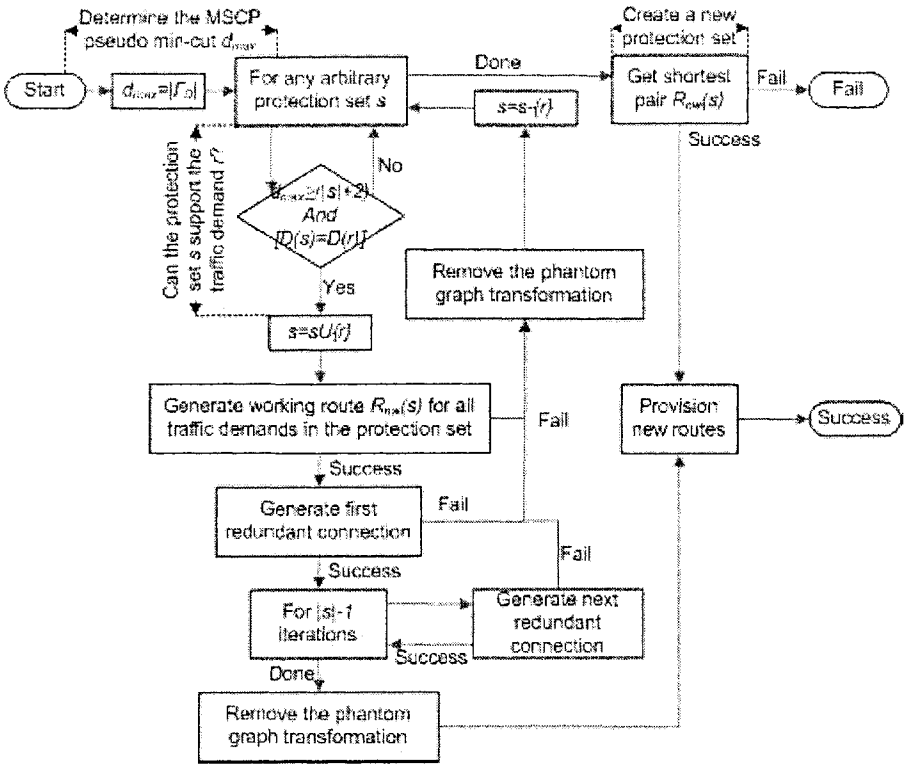


Figure 4.5.: Re-optimized Multiple Source Coded Protection Flowchart

In the RMSCP, whenever a new demand enters the network, re-optimization will be attempted. To facilitate the re-optimization, an arbitrary protection set of pre-existing demands will be chosen to be reorganized. A connection request can join an existing protection set if equation 4.6 is true. In the equation, $A(s)$ determines if a connection request r can potentially join a protection set s.

$$A(s) = \begin{cases} 0 & D(r) \neq D(s) \\ 0 & d_{max} < (|s| + 2) \\ 1 & otherwise \end{cases} \tag{4.6}$$

In order for a connection request r to join a pre-existing protection set s, it must have the same destination $D(r)$ as the protection set $D(s)$. This is because the protection sets for MSCP based algorithms must have a common destination. The second criterion is used to ensure that protection sets that have reached thier min-cut can not be used for more sharing. In it, if the number of traffic demands in the protection set $|s|$ plus the two addition routes required for the coded redundant path and connection request is greater than d_{max}, then the protection set can not be used. Either way, the algorithm will proceed to the next available protection set and test its availability with equation 4.6. If the algorithm does not find an available protection set, a shortest pair will be generated for a new protection set s, containing the connection request r. Otherwise, if the algorithm finds a protection set that is capable of handling an additional demand it will proceed with the re-optimization procedure. Before performing any re-optimization operations, the connection request r is added to the set of traffic demands that exist in the protection set s. This allows the algorithm to re-optimize all the pre-existing traffic demands and the connection request together. The re-optimization procedure consists mainly of a modified variant of the K-Shortest Paths Algorithm and a redundant connection generator. The modified variant of the K-Shortest Paths Algorithm used for RMSCP generates the shortest set of for all of the traffic demands in the protection set. Its operation is depicted in figure 4.6.

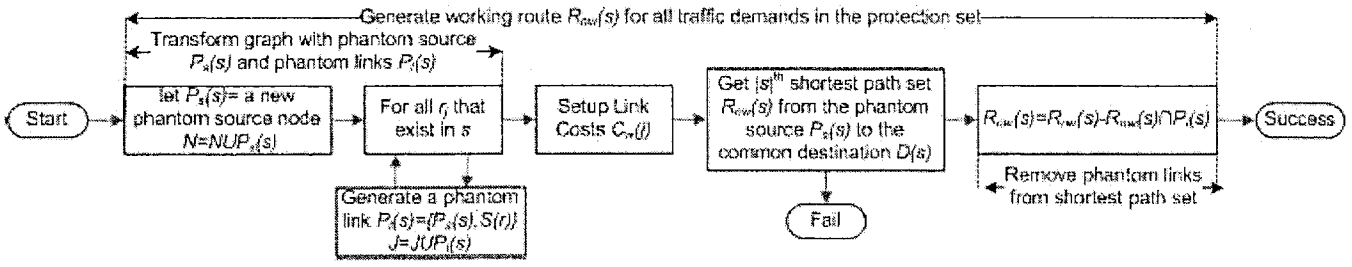


Figure 4.6.: Working Route Generator Flowchart

To generate a shortest path set from a set of source nodes to a single destination, a special graph transformation is required. This transformation is depicted in figure 4.7. The transformation consists of a phantom source node $P_s(s)$ and a set of phantom directional links $P_l(s)$ between $P_s(s)$ and the source node $S(r_j)$ of each traffic demand r_j in the protection set s. This phantom node and its directional links are only created for routing purposes. They do not represent any real connections in the network.

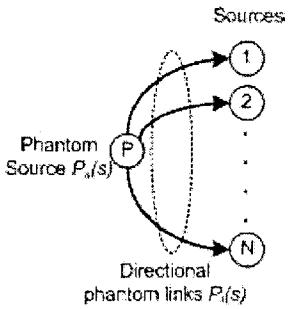


Figure 4.7.: Phantom Source Graph Transform Appearance

With the phantom source node and its directional links, the K Shortest Paths Algorithm can determine the new $|s|$ disjoint paths $R_{nw}(s)$ required for the working routes in the protection set. From the set of working routes $R_{nw}(s)$, each traffic demand r_j has a corresponding working route $R^j_{nw}(s)$. Generating the path set is done by treating the phantom source node $P_s(s)$ as the source node for the K Shortest Paths Algorithm. As depicted in figure 4.8, each of the $|s|$ disjoint paths from the phantom source node connects through a traffic demand source $S(r_j)$ from the protection set to the destination node $D(s)$.

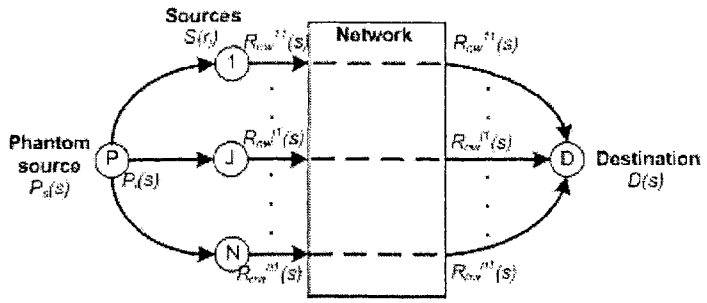


Figure 4.8.: K Shortest Paths Algorithm with Phantom Source Appearance

For the path search operation, the link costs are set according to equation 4.3. Of course, the phantom links generated in the graph transformation are not based on any physical links and therefore have unlimited available capacity A_{ij}.

$$C_w(ij) = \begin{cases} \infty & b > A_{ij} \\ \epsilon & b \leq A_{ij} \end{cases} \tag{4.7}$$

If the shortest path set can not be generated because of network congestion or differences between the pseudo min-cut and the true min-cut, the algorithm will remove the connection request from the protection set and cycle into the next available protection set.

If a shortest paths set can be generated between the phantom source node and destination node, the algorithm will proceed to the next stage. Since the K Shortest Paths Algorithm was routed from the phantom source node to the destination, it will include the phantom links. These phantom links have no function in the shortest paths set and must be removed before survivability can be added. Equation 4.8is employed to remove all possible phantom links $P_l(s)$ from the working route set $R_{nw}(s)$.

$$R_{nw}(s) = R_{nw}(s) - R_{nw}(s) \cap P_l(s) \tag{4.8}$$

Figure 4.9 also portrays this removal operation. The phantom links $P_l(s)$ are only removed from the new shortest path set $R_{nw}(s)$ and not from the graph abstraction of the network. This is because the phantom source node and its links are required for the redundant connections.

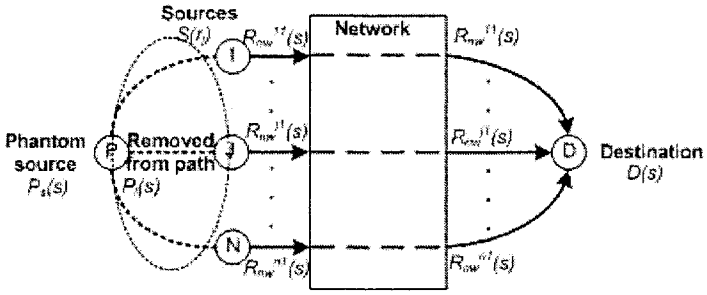


Figure 4.9.: Phantom Connection Removal Operation

In order for RMSCP to provide protection to a set of demands, it requires that there is one more path than there are demands in the protection set. Thus, the RMSCP algorithm must generate a redundant path between one of the source nodes $S(r_j)$ and the destination node $D(s)$ of the protection set. Since it is not known which source node is the optimal source node to have the redundant connection from, the routing will be done from the phantom source node $P_s(s)$. In this case the shortest path algorithm will be able to find the shortest available redundant path out of all the paths from each the source nodes $S(r_j)$. Figure 4.10 has been employed to explain how the first redundant path is generated.

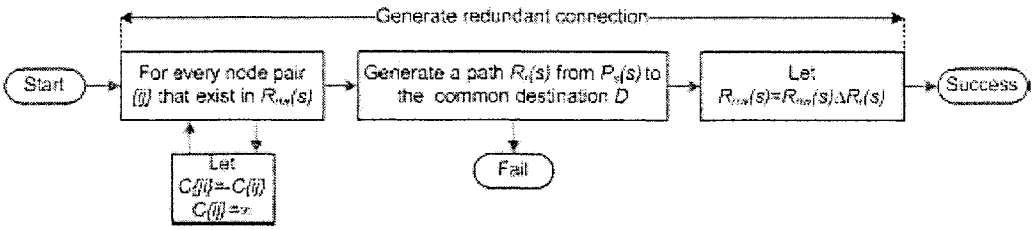


Figure 4.10.: Redundant Connection Generator Flowchart

From the phantom source node, the redundant connection can be routed using shortest path set principles, so that it creates another shortest path to the destination without suffering from the trap topology problem. This event is depicted with figure 4.11 as an additional route connecting from the phantom source through any one of the sources to the destination node. In fact, this additional connection turns the $|s|$ shortest paths set into the $|s|+1$ shortest paths set $R_{nw}(s)$. After this procedure, a traffic demand r_j with the redundant connection will have two routes between its source $S(r_j)$ and the common destination $D(s)$. These two routes will be known as $R_{nw}^{j1}(s)$ and $R_{nw}^{j2}(s)$.

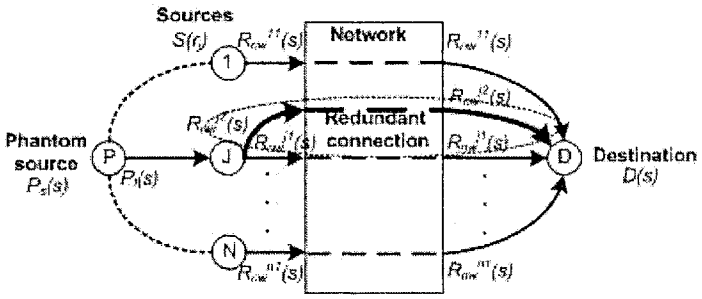


Figure 4.11.: Redundant Connection Appearance

The redundant connection added in the previous iteration is enough to provide protection to the one demand which shares a source node with it. However, there are still $|s|-1$

demands in the protection set that require a redundant connection in the network. These demands will use branch connections to carry their redundant information into paths that carry linearly independent data. The flowchart in figure 4.12 depicts the set of operations required to create the redundant branch connections.

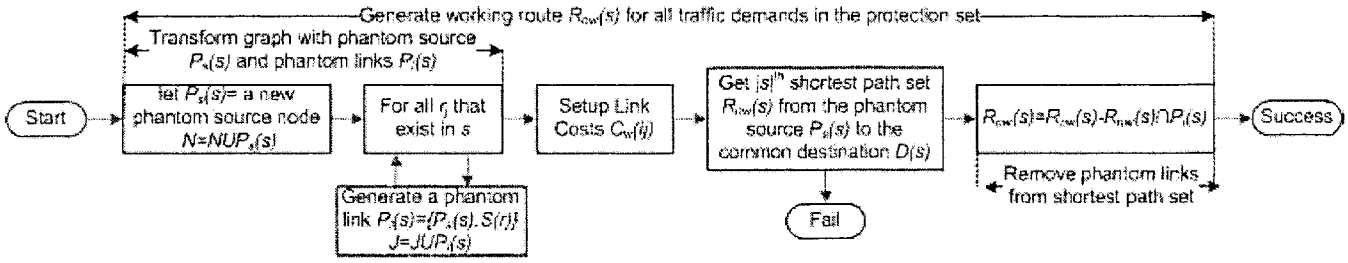


Figure 4.12.: Redundant Branch Connection Generator Flowchart

Normally the redundant branches can connect to any route as long as they are disjoint from the individual demand's primary route and no linearly dependent routes are generated. However, in RMSCP the two connections $R_{nw}^{j1}(s)$ and $R_{nw}^{j2}(s)$ in the previous stage is already carry linearly dependent information. This is unacceptable for solving missing demands in fault events at the destination and must be mitigated. For this reason, the first branch connection generated by the algorithm should go to one of the intermediate nodes along $R_{nw}^{j1}(s)$ or $R_{nw}^{j2}(s)$. Thus, the available destinations $D_b(s)$ for the branch connections are determined with equation 4.9.

$$D_b(s) = D_b(s) \cup R_{nw}^{j1}(s) \cup R_{nw}^{j2}(s) \tag{4.9}$$

Figure 4.13 depicts this situation with a branch connection routed to any of the intermediate nodes in the redundant connection or its linearly dependent primary route. To perform this special routing operation a modified variant of Dijkstra's operation is required. This special

Dijkstra's algorithm is modified so that it ends if it incorporates any of the intermediate nodes along the two linearly dependent routes. This modification has been included in appendix E as algorithm E.3. In order to ensure that the shortest possible branch is taken by the algorithm, the phantom source $P_s(s)$ is utilized.

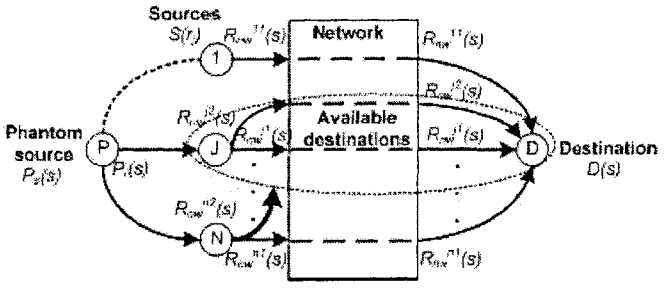


Figure 4.13.: First Redundant Branch Connection Appearance

The previous iteration involving the redundant path connection was the last path search that relied on the shortest path set link assignments. Since, the branch connection does not benefit from the interlacing of paths used to create the shortest path set, it can be routed using the normal approach. To maintain disjointness between all connections, the branch connection can not use any links used by any of the working routes already defined. Thus the path cost for the branch connection must be defined by equation 4.10.

$$C_r(ij) = \begin{cases} \infty & \{ij\} \in R_{nw}(s) \\ \infty & else if\, b > A_{ij} \\ \epsilon & else if\, b \leq A_{ij} \end{cases} \tag{4.10}$$

The phantom source node $P_s(s)$ is used as the source node for the branch connection for two reasons. First off, so that the real source with the shortest branch connection will be generated in each iteration. Secondly, so that no real source node can be branched from

more than once. The second criteria is maintained with the phantom links between each of the source nodes and the phantom source node. When the phantom source generates a branch connection through a real source node, the phantom link between them becomes part of the new working path set $R_{nw}(s)$. Therefore, in the proceeding iterations, the link cost $C_r(ij)$ for that phantom link will be infinite, forcing the branch connection through another real source node $S(r_j)$. The branch connection itself with the path that it merged with becomes a new redundant route $R_{nw}^{j2}(s)$ for a traffic demand r_j in the protection set s.

In the first iteration of the branch connection generator, the branch had to be routed to one of the linearly dependent connections. In the remaining $|s| - 2$ iterations of the generator, the set of nodes which can be a destination for a branch perpetually increase. When a branch connection is routed though one of the real sources $S(r_j)$, the nodes along its primary route $R_{nw}^{j1}(s)$ and branch connection $R_{nw}^{j2}(s)$ become available as destinations for proceeding iterations. Thus, equation 4.9 is performed in every iteration of the branch connection generator. This situation is depicted in figure 4.14, where the oval of available destinations increases with every iteration. However, it is important to note that each branch connection is still bounded by the link costs defined by equation 4.10. That is, no redundant connection can be routed over the same bidirectional link as a another branch connection or the working paths. Since the branch connections are merged with the working route $R_{nw}(s)$ in every iteration this can be done easily by recalculating the link costs in equation 4.10. In the final iteration of the branch iterator, the phantom node will route through the last real source to a node along any of the connections in working routes R_w.

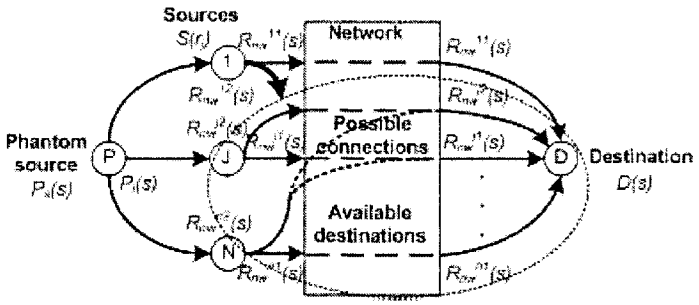


Figure 4.14.: Second Redundant Branch Connection Addition Appearance

After the final branch connection has been routed, the routes required to create MSCP is essentially complete. The last remaining operation, is the final removal of the phantom source $P_s(s)$ and its phantom links $P_l(s)$ from the new route set $R_{nw}(s)$ and the graph abstraction of the network. This is done by using the set of operations depicted in figure 4.15.

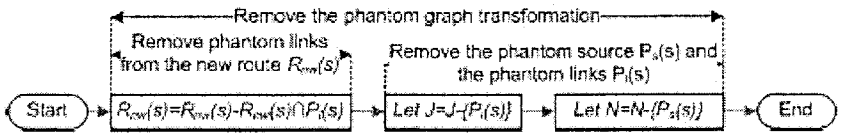


Figure 4.15.: Phantom Transformation Removal Flowchart

Removing the phantom network is basically a three step operation. First all the phantom links $P_l(s)$ are removed from the working route set $R_{nw}(s)$ using equation 4.11.

$$R_{nw}(s) = R_{nw}(s) - R_{nw}(s) \cap P_l(s) \tag{4.11}$$

Afterward, all the phantom links $P_l(s)$ can simply be removed from the set of links J in the network. Lastly, the phantom source $P_s(s)$ can be removed from the set of nodes N

that are in the network. This operation is also performed if the algorithm fails to generate a working or redundant route and must find a different protection set.

The final operation required for RMSCP is the re-provisioning of the entire protection set s. Before and during the routing of a new connection request r to an existing protection set s, the protection set still had its traffic provisioned on an existing set of working routes $R_w(s)$. However, the algorithm generated a new set of working routes $R_{nw}(s)$. Since during the routing phase the capacity used by the existing set of routes $R_w(s)$ was not taken into consideration, the traffic demands r_j in the protection set can be switched over to the new route $R_{nw}(s)$ without any effects of their service quality. The other option was to consider the resources used by $R_w(s)$ in the calculations for the new path set $R_{nw}(s)$. This would improve the performance of the algorithm under congested conditions, but would also generate a drop in service every time a connection request r was added to the protection set s. The set of operations is depicted in further detail in figure 4.16.

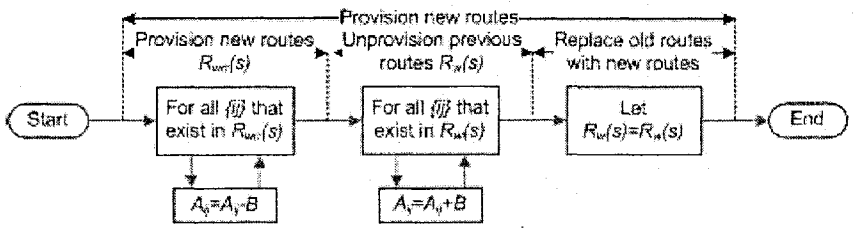


Figure 4.16.: Protection Set Re-provision Flowchart

In the figure, the new route $R_{nw}(s)$ is provisioned before the old route $R_w(s)$ is un-provisioned. Thus, the traffic demands in the protection set will not experience any loss in service.

After the re-provisioning is complete, the RMSCP algorithm is complete. The completed appearance of the MSCP created by RMSCP is displayed as figure 4.17. In it, the branch connections from each of the source nodes are routed to any of the nodes that were available to it in its iteration of the branch connection generator.

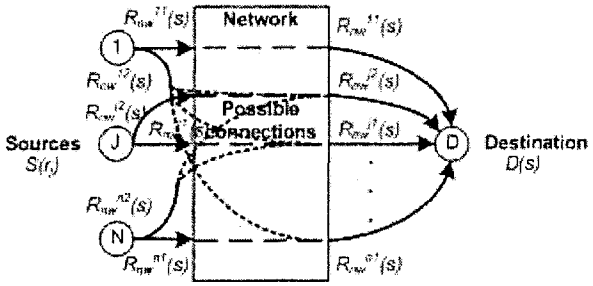


Figure 4.17.: Final Re-optimized Multiple Source Coded Protection Appearance

Since each source is given a primary connection and a branch or redundant connection that are disjoint, it is impossible for a single failure event to disable a source. Likewise, since the available destinations for each branch connection are sequentially increased with each iteration there are no possibilities for linear dependent information streams. Algorithm A.3 in appendix A has been employed to further explain the operation of RMSCP. To aid in its readability, the redundant path generator and the branch redundant path generator has been displayed after the main aspects of the algorithm.

4.2.2. Fast Multiple Source Coded Protection

Fast Multiple Source Coded Protection (FMSCP) is a heuristic algorithm designed to be a quick and effective technique for routing MSCP connections. Unlike RMSCP mentioned earlier, FMSCP does not use re-optimization. FMSCP relies on a modified version of Bhandari's Algorithm to generate MSCP with shortest path set characteristics. Since FMSCP

utilizes these characteristics, it is not prone to the trap topology problem. Likewise, since the algorithm is a modification of Bhandari's algorithm it only requires two iterations of Dijkstra's algorithm to resolve a connection request. This is much faster than RMSCP, since the $2 \times |s|$ iterations of Dijkstra's algorithm required to complete a RMSCP connection request can be significantly greater than FMSCP. Figure 4.18 has been employed to explain the operation of FMSCP.

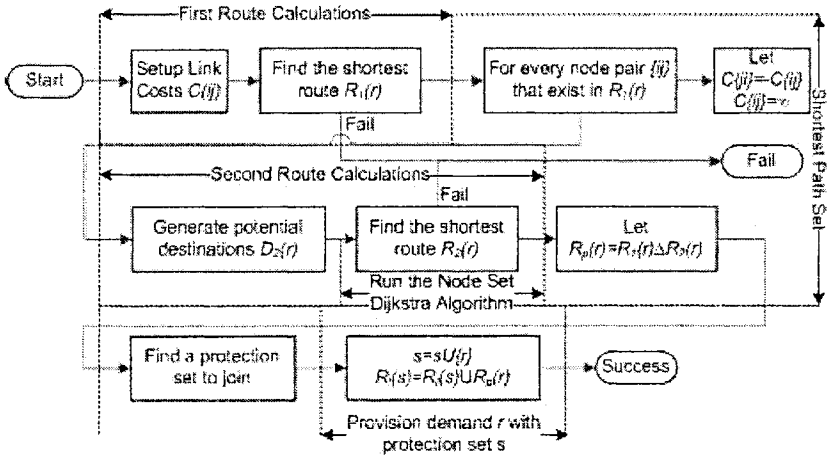


Figure 4.18.: Fast Multiple Source Coded Protection Flowchart

As with all algorithms that use shortest path set characteristics, the link costs $C(ij)$ are assigned only once at the beginning. The link costs will be generated based on the available capacity in the network. If a link $\{ij\}$ does not have enough available capacity A_{ij} to route the connection request, it will set the cost $C(ij)$ equal to ∞. Otherwise, $C(ij)$ can be set to some small number ϵ. Afterward, the path $R_1(r)$ will be generated from the source to the destination node of the connection request.

Following the creation of the first route $R_1(r)$, the potential destinations for the redundant path $R_2(r)$ can be determined. These potential destinations represent the intermediate

nodes along the routes in available protection sets. To determine which nodes can be used as potential destinations, the algorithm must find out which protection sets are available for sharing. The availability of a protection set $A_1(s)$ is determined by equation 4.12.

$$A_1(s) = \begin{cases} 0 & D(r) \neq D(s) \\ 0 & else\, if\, R_1(r) \in R_l(s) \\ 1 & otherwise \end{cases} \tag{4.12}$$

There are two constraints on imposed on a protection set s. First, the destination of all the traffic demands $D(s)$ in the protection set s, must be the same as the destination of the $D(r)$ connection request r. This is a requirement for all MSCP based algorithms. Second, the first route $R_1(r)$ of the connection request r, must be disjoint from all links $R_l(s)$ in the protection set s. This is necessary to maintain the disjoint requirements of survivability algorithms.

As mentioned earlier, in MSCP sharing can only occur if multiple demands share the same destination node $D(s)$. Thus, FMSCP requires that this constraint be imposed on any potential protection set s. Adding to that, in order to ensure that no two connections fail simultaneously, no link can be utilized twice in a protection set s. Therefore, a connection request can only join a pre-existing protection set if it is disjoint from it. If those constraints are met, a connection request can join a pre-existing protection set. In order to conceptualize the available protection sets that can be used by a connection request, the possible destinations $D_2(r)$ for the second route $R_2(r)$ are manipulated. The second route $R_2(r)$ is given the option of linearly combining it's contents with any existing path $R_l(s)$ of any available protection set s. This is done by routing $R_2(r)$ to any of the intermediate nodes $R_n(s)$ along the routes $R_l(s)$ traversed by any available protection set s. This operation can be performed using the operations in figure 4.19.

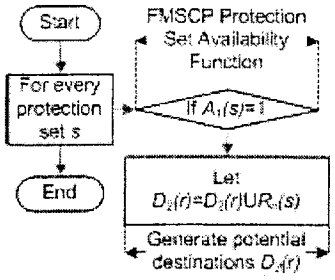


Figure 4.19.: Fast Multiple Source Coded Protection Potential Destination Generator

Routing to any of a set of nodes can be done using the Node Set Dijkstra algorithm in appendix E. The Node Set Dijkstra algorithm is a variant of Dijkstra's algorithm that will end if any of the potential destinations are incorporated. The first potential destination to be incorporated becomes the destination for the path search operation.

To ensure that $R_1(r)$ does not generate a trap topology, shortest pair attributes from Bhandari's Algorithm are imposed on it before $R_2(r)$ is routed. Likewise, after $R_2(r)$ is routed, shortest pair modifications must be performed on it and $R_1(r)$ to generate the shortest pair $R_p(r)$. Since a protection set is depicted as a set of possible destination nodes $D_2(r)$ and multiple protection sets do not have to be node disjoint, it is possible that a connection request can choose from multiple protection sets after routing is completed. It is not important which of the available protection sets is chosen, so it can be determined by an arbitrary technique. Thus, in order to determine which protection sets can be used with $R_2(r)$, the routed protection set availability equation $A_2(s)$ is required. The original protection set availability equation determined which protection sets could be routed to by $R_2(r)$. After $R_2(r)$ has been routed, the routed protection set availability equation is required to determine which of the available protection sets were routed to. This routed protection set availability equation is depicted as equation 4.13.

$$A_2(s) = \begin{cases} 0 & D(r) \neq D(s) \\ 0 & else\, if\, R_1(r) \in R_l(s) \\ 0 & else\, if\, d \notin R_n(s) \\ 1 & otherwise \end{cases} \tag{4.13}$$

The difference between the original protection set availability equation and the routed protection set availability lies in one additional case statement. A protection set is only available if the destination d of $R_2(r)$ is an intermediate node along the route $R_l(s)$ in the protection set s. If $R_2(r)$ could not share with any of the protection sets, it will route its destination d to the true destination $D(r)$ for the connection request r. Under this circumstance, the shortest pair $R_p(r)$ will be used to create a new protection set s. This operation is depicted as a flowchart in figure 4.20.

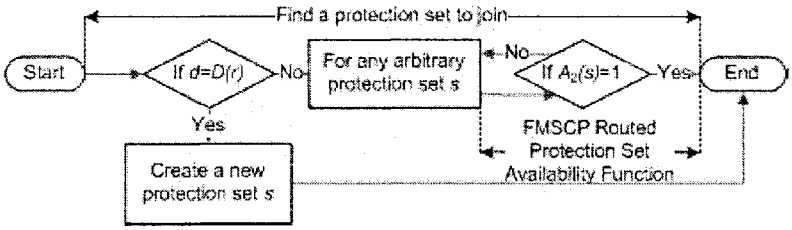


Figure 4.20.: Fast Multiple Source Coded Protection Protection Set Joining Operation

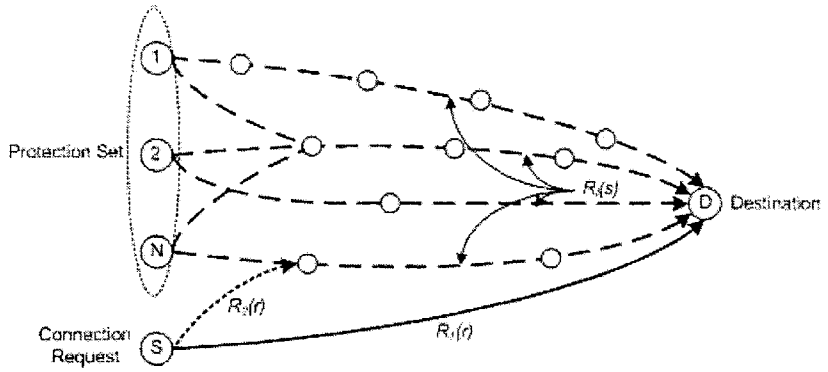


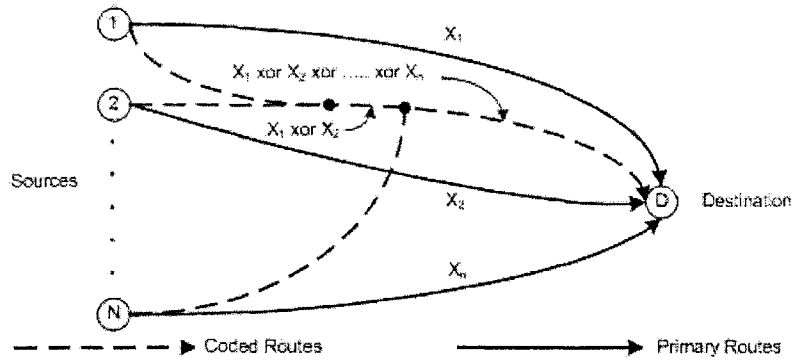
Figure 4.21.: Fast Multiple Source Coded Protection Operation

Figure 4.21 has been employed to further illustrate the appearance of FMSCP. In the figure, an unconstrained route $R_1(r)$ is initially generated for a connection request r. If $R_1(r)$ is disjoint from every link that exists in $R_l(s)$ for a protection set s then the nodes along the route $R_l(s)$ will be added to the available destinations $D_2(r)$ for route $R_2(r)$. If $R_2(r)$ is routed to one of the destinations created by s then the shortest pair $R_p(r)$ and connection request r will be added to the protection set s. If $R_2(r)$ connects to the destination node $D(r)$ for connection request r then $R_p(r)$ will be used to generate a new protection set s. Algorithm A.4 in appendix A has been created to further explain how the FMSCP algorithm works.

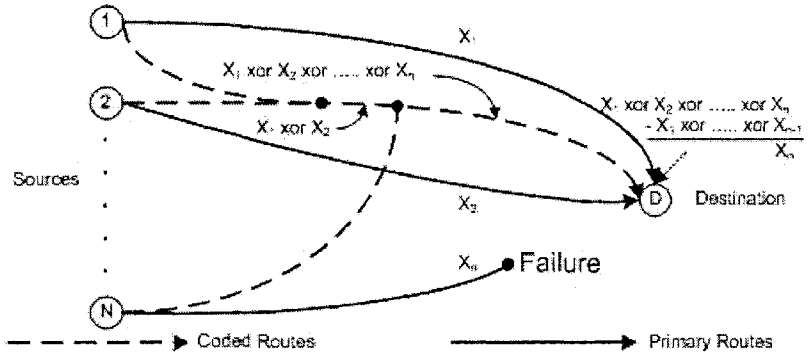
4.2.3. Single Stream Multiple Source Coded Protection

Single Stream Multiple Source Coded Protection (SSMSCP) is a heuristic algorithm designed to perform MSCP protection using only one coded path. In RMSCP and FMSCP any path between the protection set and the destination could be encoded with multiple demands. Although efficient, this method puts strain on the computation capabilities of the destination by forcing it to always decode the $|s|$ demands in the protection set s. To

reduce the number of calculations at the destination, it may be desirable to make MSCP appear like SCP at the destination node. That is, SCP normally sends its working information through uncoded channels and uses a single redundant channel as the coded redundant route. Thus, under normal conditions SCP does not require any decoding. Likewise, during single link failure events, linearly combining all connections together using an exclusive OR operation recovers the missing information. However, in RMSCP and FMSCP a complex matrix decoding technique is required to perform the same function. Figure 4.22a has been employed to illustrate how SSMSCP appears under normal conditions. Under these normal conditions, the destination receives uncoded connections from each of the source nodes. In this instance, no decoding is required for any of the information. In figure 4.22b, the failure event for SSMSCP is depicted. If any of the uncoded connections fail due to a single link failure event, the coded connection can be used to decode the missing information. If the coded connection is affected by the single failure event, the uncoded connections will still be able to carry their information to the destination.



(a) Normal Operation



(b) Under Failure

Figure 4.22.: Single Stream Multiple Source Coded Protection Appearance

Since SSMSCP is meant to be a simplest heuristic algorithm for MSCP, it is based on the FMSCP routing technique instead of complex re-optimization. From FMSCP, SSMSCP inherits the shortest pair characteristics and Node Set Dijkstra algorithm. Thus, SSMSCP is immune to the trap topology problem and uses multiple available destinations for its second path calculation. The operation of SSMSCP is depicted as a flowchart in figure 4.23.

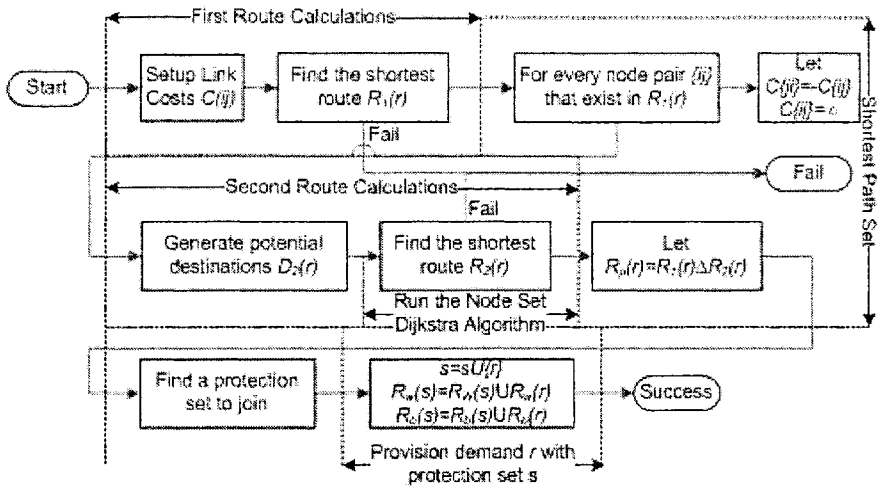


Figure 4.23.: Single Stream Multiple Source Coded Protection Flow Chart

In order to route a connection request r with SSMSCP a slightly modified version of the heuristic algorithm for FMSCP is required. In fact, the only major difference between the two routing algorithms is in the assignment of available destinations for the second route $R_2(r)$. In SSMSCP each protection set s distinguishes between the working routes $R_w(s)$ and backup route $R_b(s)$. When the backup route for a new connection request is established only the nodes $R_{bn}(s)$ along the backup route $R_b(s)$ for a protection set s are available as destinations $D_2(r)$ for the backup route $R_2(r)$. Thus, the SSMSCP potential destination generator is slightly different from the FMSCP version. This new version is depicted in figure 4.24.

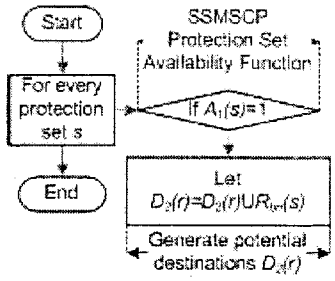


Figure 4.24.: Single Stream Multiple Source Coded Protection Potential Destination Generator

Functionally, the protection set availability equation $A_1(s)$ for SSMSCP is the same as FMSCP, however since a protection set s now has working $R_w(s)$ and backup $R_b(s)$ links, the equation is different. This new protection set availability equation is presented as equation 4.14.

$$A_1(s) = \begin{cases} 0 & D(r) \neq D(s) \\ 0 & else\, if\, R_1(r) \in (R_w(s) \cup R_b(s)) \\ 1 & otherwise \end{cases} \tag{4.14}$$

The protection set joining operation for SSMSCP is also similar to that of FMSCP. The only difference is in the routed protection set availability equation $A_2(s)$. This new method is presented as equation 4.15.

$$A_2(s) = \begin{cases} 0 & D(r) \neq D(s) \\ 0 & else\, if\, R_1(r) \in (R_w(s) \cup R_b(s)) \\ 0 & else\, if\, d \notin R_{bn}(s) \\ 1 & otherwise \end{cases} \tag{4.15}$$

The actual SSMSCP protection set joining operation is the same as FMSCP. For reference it is depicted here as figure 4.25.

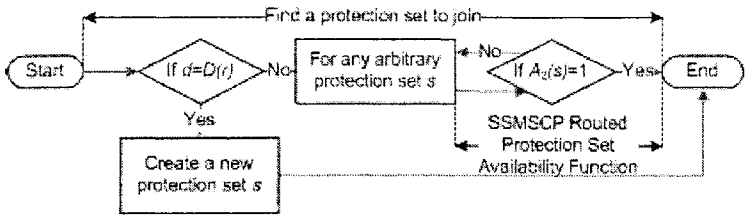


Figure 4.25.: Single Stream Multiple Source Coded Protection Protection Set Joining Operation

Likewise, at the end of the routing algorithm, the working route $R_w(r)$ and backup route $R_b(r)$ of the shortest pair $R_p(r)$ must be individually added to the working $R_w(s)$ and backup $R_b(s)$ routes of the protection set s. The SSMSCP algorithm is further summarized in algorithmic form as algorithm A.5 in appendix A.

4.3. Network Coded Protection Heuristic Algorithms

Network Coded Protection is significantly more complex than MSCP and SCP. Connection requests protected by a NCP scheme have a proactive coded section and a semi-reactive shared section. Therefore, unlike the algorithms for SCP and MSCP, NCP heuristic algorithms must utilize one of the three secondary connections to decode the redundant information. This makes a NCP connection request a four part routing problem. How do we determine:

1. The working route
2. The backup route coded section
3. The backup route shared section

4. The working route secondary connection

In every MSCP and SCP routing technique, no backup route shared section or working route secondary connection were required because the destination was the decoding point for a protection set. However, in NCP since this constraint is relaxed, additional effort is required to route demands. Furthermore, in RMSCP and FMSCP, the working and backup routes can be mixed together to improve the efficiency. Since NCP uses a semi-reactive approach to survivability instead of pure protection, working and backup paths must be separated like in SSMSCP. Of particular concern for routing is the backup route. It consists of two sections different in operation and method of sharing. Of these, the coded section can be routed easily using algorithms with shortest paths set attributes. However, the shared section, depending on its method of conceptualizing sharing can not be routed using shortest paths set attributes. An example of this is the link cost sharing concept used by pool sharing algorithms. Without shortest path set attributes, the trap topology problem can cause NCP to be unattractive. Thus, there are essentially three methods for handling this problem.

- Pre-determine the route of one of the sections.
- Solve it as a special constrained simple two step problem.
- Use graph transformations to represent individual sharing opportunities.

The following three algorithms each use one of these techniques to generate NCP.

4.3.1. Neighbor Decoded Protection

Neighbor Decoded Protection (NDP) is NCP heuristic algorithm designed to guarantee at most one hop restoration times to demands. It performs this function by constraining the layout of the shared section so that it can be no more than one hop. Modifying the general

layout of NCP so that one hop restorations are guaranteed produces a new structure. This new structure can be specified with figure 4.26.

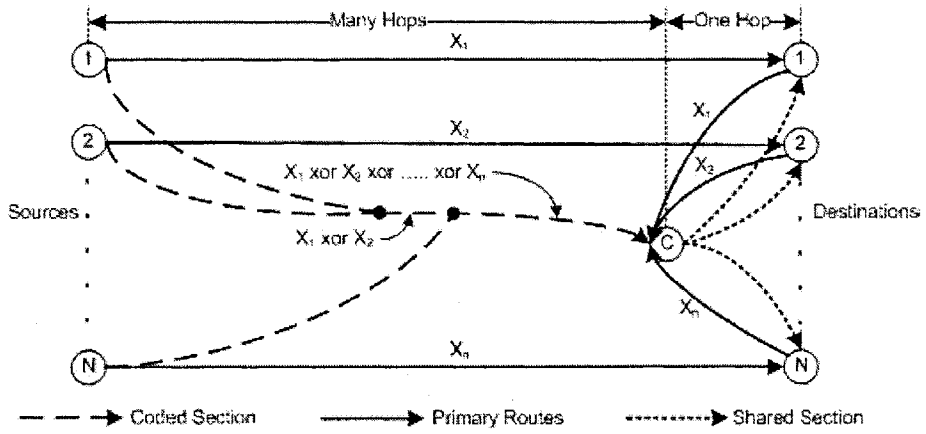


Figure 4.26.: Neighbor Decoded Protection Normal Operation Appearance

In the figure, the coded section is unchanged from the general NCP layout described in section 3.3. Likewise, the primary path is still provisioned disjointly from the coded and shared section of the protection set. The major difference is in the shared section. The shared section can only consist of the zone of nodes that are at most one hop away from the critical node. To ensure that restoration times are not increased by the secondary connections, a special variant of the feedback loop is used exclusively for NDP. This variant of the feedback loop is constrained so that it traverses the same bidirectional link as the shared link. By limiting the options of the feedback loop in this respect, it can only be at most one hop and it can be predetermined without an additional step. The constraint only affects the algorithm under heavily loaded conditions where there isn't enough available bandwidth to provision the feedback loops. Thus, NDP eliminates the need for shortest path searches for both the shared section and the secondary connection. The failure operation

for NDP is the same as the general NCP technique described in section 3.3. However, for further reference, the failure operation specifically for NDP is depicted as figure 4.27.

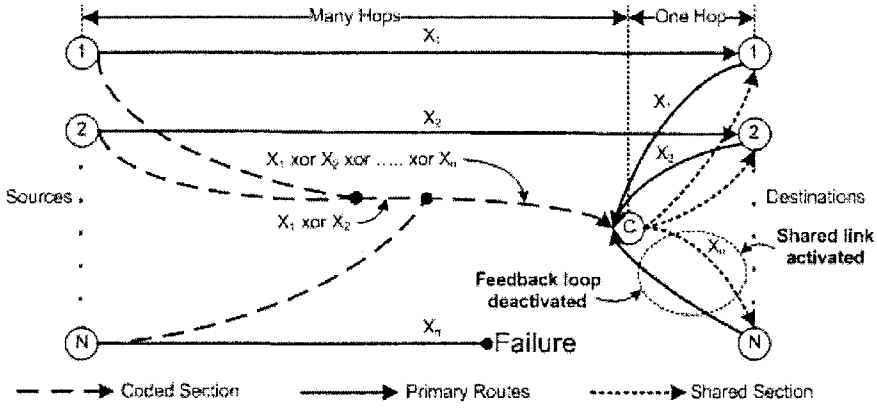


Figure 4.27.: Neighbor Decoded Protection Failure Operation Appearance

As with all NCP heuristic algorithms, the primary routes are disjoint from the coded and shared sections of a protection set. Thus, in a failure event either the primary route of a traffic demand or some part of the coded/shared section of the protection set fails. If a part of the coded/shared section fails, normal operation is maintained. Likewise, if a primary route fails, redundant information can be decoded at the critical node and forwarded to the destination through the one hop shared section.

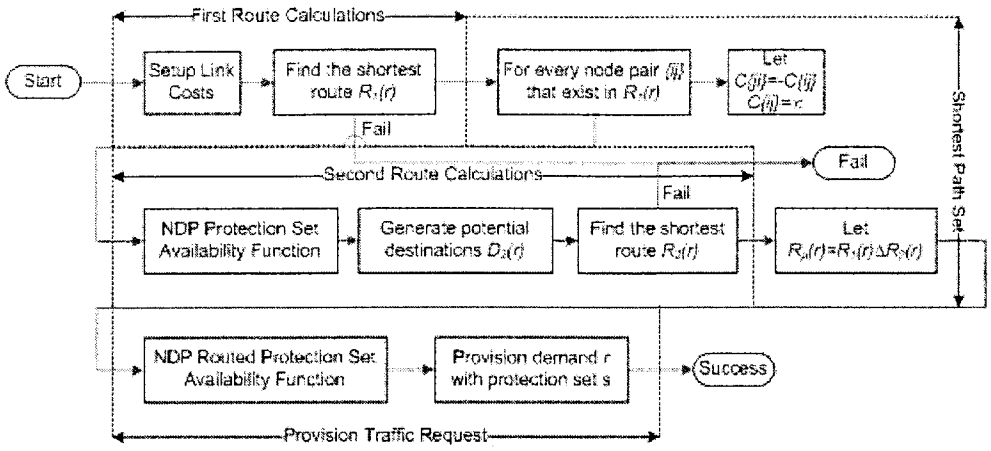


Figure 4.28.: Neighbor Decoded Protection Flowchart

Figure 4.28 illustrates the operation of NDP as a flowchart. By removing the path search operations for both the shared section and the secondary connection, the NDP heuristic algorithm can be simplified into a variant of the SSMSCP algorithm. Like SSMSCP, NDP uses the Node Set Dijkstra algorithm, shortest set characteristics, and the protection set availability equations to route the primary route and the coded section. Initially, the link costs for the algorithm are setup according to equation 4.16. This ensures that only the links that are capable of provisioning the connection request are used in the path search operations.

$$C(ij) = \begin{cases} \infty & b > A_{ij} \\ \epsilon & b \leq A_{ij} \end{cases} \tag{4.16}$$

Afterward, the first route $R_1(r)$ between the source and destination $D(r)$ of the connection request r can be determined. In order to gain shortest path set characteristics, $R_1(r)$ is used to modify the link costs of the graph. With that completed, the protection set availability equation can be used to determine which protection sets can be used by the connection request. Since NDP generates at most one hop restorations for any traffic demand r, the

critical node *C(s)* of the protecting set s must be a neighbor of the destination node *D(r)*. Thus, a connection request r can not join a protection set s if it is not a neighbor of the critical node *C(s)*. This is conceptualized as equation 4.17, where $\Gamma_{C(s)}$ is the set of nodes that are neighbors of the critical node $C(s)$.

$$D(r) \in \Gamma_{C(s)} \tag{4.17}$$

Even, if the destination is a neighbor of the critical node, the protection set still might not be available for restoration purposes. Two critierion are required for disjoint reasons:

- The working route of a traffic demand must be disjoint from its backup route.
- The working route of a traffic demand must disjoint from any traffic demands it shares resources with.

To ensure that these two criterion are met, equation 4.18 must not be true.

$$R_1(r) \in (R_s(s) \cup R_c(s) \cup R_w(s) \cup \{C(s)D(r)\} \cup \{D(r)C(s)\}) \tag{4.18}$$

Of particular importance to routing the backup route for NDP is the bidirectional link which will contain shared section and feedback loop. This is because the bidirectional link between the critical node and the destination for the connection request might not have enough capacity to add the connection request. For this bidirectional link, the bandwidth b of a connection request is required in both directions. Specifically, one unit of bandwidth is required from the critical node *C(s)* to the destination *D(r)* for the restoration route and another unit is required for the feedback loop from the destination *D(r)* to the critical node *C(s)*. Fortunately though, if there already is a traffic demand in the protection set that heads to the same destination as the connection request then the connection request can share the capacity reserved on the bidirectional link. This is conceptualized as having the

link from the critical node *C(s)* to destination *D(r)* *{C(s)D(r)}* in the shared section $R_s(s)$ and the link from the destination to critical node *{D(r)C(s)}* in the coded section $R_c(s)$ of the protection set s. Otherwise, if no other traffic demand in the protection set heads to the same destination as the connection request, the connection request must provision the capacity on the bidirectional link. Therefore, there must be enough free capacity on both directions of the link to provision connection request. If neither of these two criterion are met, the connection request can not join the protection set. The protection set availability equation for NDP is presented as equation 4.19.

$$A_1(s) = \begin{cases} 0 & D(r) \notin \Gamma_{C(s)} \\ 0 & else if\, R_1(r) \in (R_s(s) \cup R_c(s) \cup R_w(s) \cup \{C(s)D(r)\} \cup \{D(r)C(s)\}) \\ 1 & else if\, \{C(s)D(r)\} \in R_s(s)\, and\, \{D(r)C(s)\} \in R_c(s) \\ 1 & else if\, A_{C(s)D(r)} \geq b\, and\, A_{D(r)C(s)} \geq b \\ 0 & otherwise \end{cases} \tag{4.19}$$

Once all the available protection sets have been determined, the potential destinations for the second route $R_2(r)$ can be created using the Node Set Dijkstra's algorithm. The second route $R_2(r)$ will be established between the source node and any of the potential destinations created from the protection sets or the real destination of the connection request. Afterward, shortest path set modifications will be performed to make the two routes $R_1(r)$ and $R_2(r)$ into the shortest pair $R_p(r)$. Based on the destination of the second route $R_2(r)$ the routed protection set availability function can be used to determine which protection set the connection request should join. Unlike FMSCP and SSMSCP which could use a simple modification of the original protection set availability equation to determine which protection set to share with, NDP requires a three step procedure. This is because there are

three types of protection sets that the connection request can join based on the destination $D(r)$ of the connection request. The protection set can have already provisioned:

1. the link $\{C(s)D(r)\}$ between the critical node $C(s)$ and the destination $D(r)$, in its shared section $R_s(s)$ and the link $\{D(r)C(s)\}$ between the destination and the critical node in its coded section $R_c(s)$ for a previous traffic demand.
2. the link $\{D(r)C(s)\}$ between the destination and the critical node in its coded section $R_c(s)$ for a previous traffic demand.
3. nothing.

Obviously, in order to minimize capacity usage, the connection request should first attempt to join a protection set that already has the links for the coded and shared section already provisioned. This corresponds to the situation where a protection set has already added a traffic demand heading to the same destination as the new connection request. This is the ideal scenario, since no additional bandwidth provisioning is required to generate the shared section and feedback loop. If no protection set has these characteristics, the algorithm will use a protection set that has already provisioned the feedback connection from the destination to the critical node in the coded section. In this situation, the capacity of the connection request only has to be provisioned for the shared section. Otherwise, if no protection sets are available with a pre-existing coded link between the destination and the critical node, the connection request will have to provision the shared section and feedback loop. If the connection request still is unable to find a suitable protection set it will create a new protection set. Figure 4.29 depicts a block diagram of the NDP routed protection set availability function.

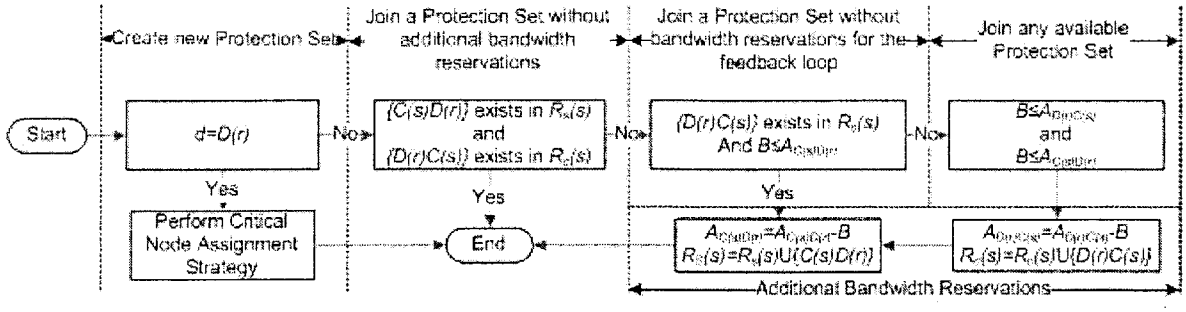


Figure 4.29.: Neighbor Decoded Protection Routed Protection Set Availability Function

If a connection request can not share with a pre-existing protection set it must create a new one. When a new protection set is created, considerable care must be put into setting the critical node. Since NDP protection sets are partially reliant on the nodal degree of the critical node for the number possible of traffic demands, higher degrees should be favored. Thus when a new, protection set s is created, a high degree node must be selected to be the critical node $C(s)$. From the shortest pair $R_p(r)$, three nodes are candidates to become the critical node of a new NDP protection set. As depicted in figure 4.30, these are the destination node and the nodes from each of the paths immediately preceding it.

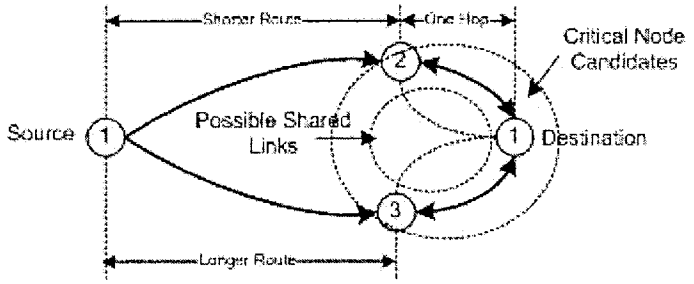


Figure 4.30.: Neighbor Decoded Protection Critical Node Assignment Strategy Appearance

In order to ensure that critical nodes are given high degrees, the highest degree node is chosen. Figure 4.31 depicts the flowchart for the critical node assignment strategy. In

the event of a tie between the nodal degree of two or three of the nodes the destination $D(r)$ is favored to become the critical node $C(s)$. Otherwise, the second last node $D_l(r)$ along the longer route becomes the critical node $C(s)$. Longer routes are favored because the intermediate nodes along the routes can be used as the coded section for future traffic demands. If the longer route can not be chosen because of insufficient capacity, the second last node $D_s(r)$ along the shorter route can be utilized. Otherwise, the algorithm will revert back to the destination $D(r)$ as the choice for the critical node $C(s)$.

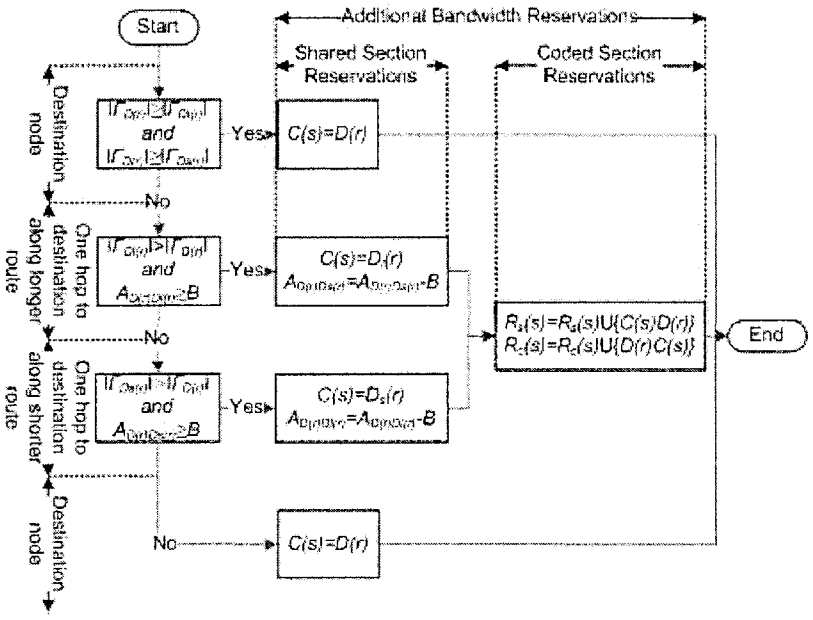


Figure 4.31.: Neighbor Decoded Protection Critical Node Assignment Strategy Flow Chart

The NDP technique is depicted in algorithmic form as algorithm A.6 in appendix A. In order to ease the understanding of the algorithm it has been divided into three pieces: The main algorithm, the NDP Routed Protection Set Availability Function, Critical Node Assignment Strategy.

4.3.2. Trunk Coded Protection

Trunk Coded Protection (TCP) is a NCP scheme that uses predefined coded trunks through a network. This allows network designers to decide in advance which ports on switches require linear combining functionality. When predefined trunks are used in a network, a few challenges arise. There are basically three problems for TCP:

1. Determining the placement of the trunks inside a network.
2. Generating the shared section.
3. Creating efficient secondary connections.

The placement of coded trunks in a network is an obvious problem for TCP. With a generalized approach, determining the most capacity efficient set of trunks is a difficult task worthy of further research. However, for the purposes of this thesis, a constrained method of establishing trunks has been utilized. This approach is based on the true goals of NCP. NCP has been designed to provide restoration times equivalent to Dedicated Path Protection (DPP), while using significantly less capacity. In a network, the adverse effects of poor restoration times is most apparent in long distance connections. In these long distance connections, restoration can take longer than the 200ms maximum, referred to in section D.3. Thus, TCP should have trunks that connect distant points in a network. The best way to conceptualize this distance problem is to subdivide a network into many zones z. Each zone z will represent a local group of nodes where restoration times are acceptable for SLAs. Thus a zone z can be considered a subset of the nodes in the network where traffic demands can use a normal Shared Backup Path Protection (SBPP) based techniques for survivability. On the other hand, traffic demands between nodes in separate zones can use protections sets s, built from inter-zone trunks $R_t(z)$ to reduce their restoration times. Figure 4.32 depicts a simplified representation of a single inter-zone trunk for a four zone

network. The trunk in the network is designed to protect traffic demands heading to zone B by using a predefined coded section that extends into every other zone.

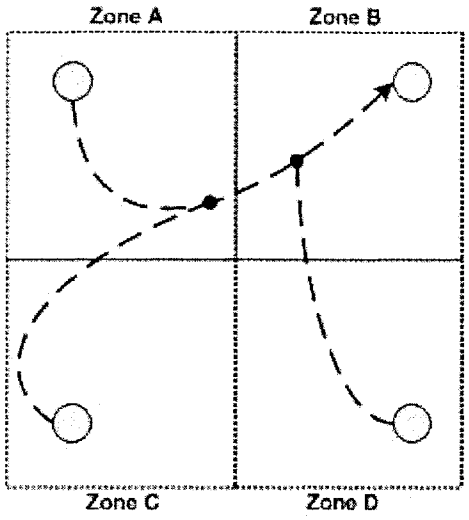


Figure 4.32.: Inter-Zone Trunks

By utilizing this technique, the problem of assigning trunk and the average restoration times are reduced. However, generating an efficient set of inter-zone trunks is also beyond the scope of this thesis. Therefore, arbitrary trunk assignments have been made for the analysis of TCP.

The second problem for TCP is how to conceptualize the shared section $R_s(s)$. Since, TCP is designed to work in conjunction with a standard restoration scheme for intra-zone traffic, it is important that the two techniques be compatible. Thus, the shared section should consist of a modified SBPP technique. As mentioned in section B.2, SPS is an efficient and quick technique for providing SBPP. It uses the link costs along for the backup route to conceptualize available shared capacity in the network. SPS can be used to provide

survivability to inter-zone connections from the critical node $C(s)$ to the destination $D(r)$ and for intra-zone connections using the same spare capacity matrix. For that reason, SPS will be used to create the shared section $R_s(s)$ for TCP. From that, there are essentially four stages to routing a TCP connection request.

1. Route a primary route $R_1(r)$ for the connection request r that intersects the trunk $R_t(z)$.
2. Route an inter-zone backup route $R_2(r)$ from the source $S(r)$ of connection request r to the trunk $R_t(z)$.
3. Route the intra-zone backup route $R_3(r)$ from the critical node $C(t)$ of the trunk to the destination $D(r)$.

These steps required to route a TCP connection request are conceptualized as the flowchart in figure 4.33.

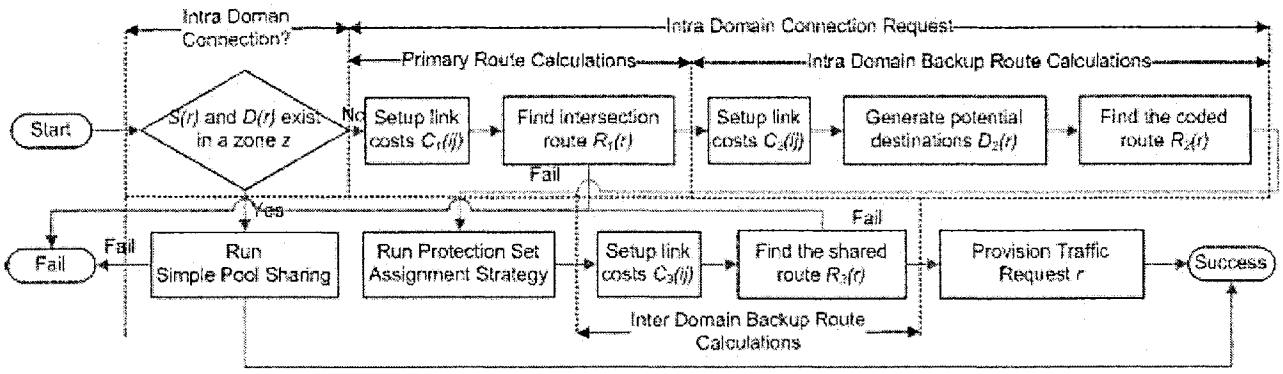


Figure 4.33.: Trunk Coded Protection Flowchart

Before performing any routing operation, the algorithm must determine if a trunk is required for the connection request. If the identity in equation 4.20 is true for any zone z, then both

the source and destination are in the same zone and can be protected by the SPS technique.

$$S(r) \cup D(r) \subset z \tag{4.20}$$

In that case, SPS can be used to provide protection to the traffic demand. Else, an inter-zone trunk is required for the connection request. Clearly, the coded trunk $R_t(z)$ used for protection should be for the zone z that contains the destination *D(r)* for the connection request r. With that, the critical node *C(t)* at the end of the trunk t can be used to provision the last mile of the backup connection over the shared capacity. Since the coded trunk $R_t(z)$ is known in advance, a routing algorithm can provision the primary route $R_1(r)$ so that it includes an intersection arc. This will remove the requirement for a feedback loop or branch connection to provide the secondary connection to the coded trunk. However, provisioning a primary route with an efficient intersection arc requires a special set of operations. This set of operations shall be known as the *Intersection Routing Technique* (IRT). The goal of IRT is to generate the shortest intersection of a route $R_1(r)$ between source *S(r)* and destination *D(r)* with an intermediate node $I(r) \in R_t(z)$. In order to ensure that the intersection takes the shortest possible route, *I(r)* should be chosen such that the relation in equation 4.21 is true. Where $I_s(r)$ and $I_d(r)$ are the costs from the source *S(r)* to the intersection *I(r)* and from the intersection *I(r)* to the destination *D(r)*

$$C_1(r) = min\left(C\left[I_s(r)\right] + C\left[I_d(r)\right]\right) \tag{4.21}$$

To generate the conditions in equation 4.21, Dijkstra's algorithm must be used to determine the route $R_s(r)$ from the source *S(r)* to all the potential intersection points $R_t(z)$ and from the intersection points to the destination *D(r)*. This operation is already completed by Dijkstra's algorithm. The algorithm only needs to be modified so that the endpoint conditions are different. Normally in Dijkstra's algorithm, the procedure ends when the

shortest path to the destination node is determined. In the node set variant of Dijkstra's algorithm, this is modified so that if a shortest route is created for any of the potential destinations, the procedure ends. For the IRT variant of Dijkstra's algorithm, the end point condition is set to when the shortest route has been generated for all the nodes that exist in the set of potential destinations. This modified variant of Dijkstra's algorithm is depicted as algorithm E.4 in appendix E.

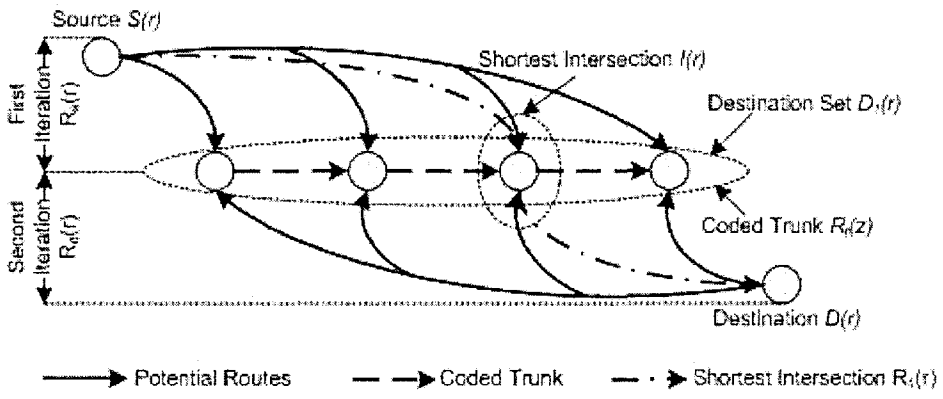


Figure 4.34.: Intersection Routing Technique Diagram

The general outline of the IRT is depicted in figure 4.34. In the figure, two iterations of Dijkstra are used to create the intersection connection.

1. Route $R_s(r)$ is generated from the source $S(r)$ to all the potential intersections defined by the destination set $D_1(r)$. Where $D_1(r)$ consists of all the nodes along the coded trunk $R_t(z)$

2. Route $R_d(r)$ is generated from the destination $D(r)$ to all the potential intersections defined by the destination set $D_1(r)$.

Since the links in the network are directional, routing from the destination to a set of sources is normally not allowed. However, if the link costs $C(ij)$ are reversed so that $C(ij) = C(ji)$ and $C(ji) = C(ij)$ this method can be used by the routing algorithm. Afterward, a path can be created by combining the two routes such that equation 4.21 mentioned earlier is true. This operation is summarized in figure 4.35.

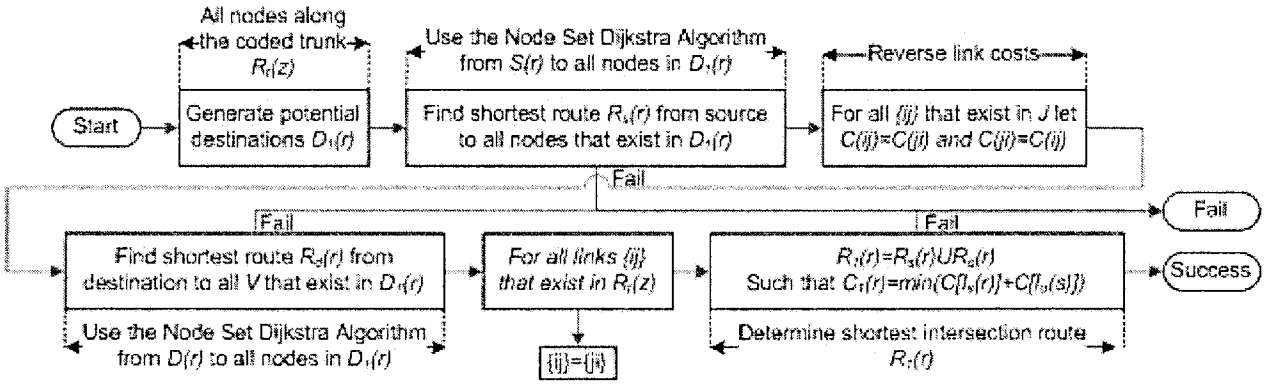


Figure 4.35.: Intersection Routing Technique Flowchart

To ensure that the intersection path $R_1(r)$ and the trunk $R_t(z)$ remain link disjoint from each other, the link costs $C_1(ij)$ must be setup according to equation 4.22. By setting up the link costs such that the links along the trunk can not be used by the working route of the connection request, disjointness is maintained between them.

$$C_1(ij) = \begin{cases} \infty & \{ij\} \in R_t(z) \\ \infty & else\, if\, b > A_{ij} \\ \epsilon & else\, if\, b \leq A_{ij} \end{cases} \tag{4.22}$$

After generating the intersection path, an intra-zone backup path $R_2(r)$ can be created from the source node $S(r)$ to the coded trunk $R_t(z)$. As with all backup routes, $R_2(r)$ must be disjoint from the working route $R_1(r)$. Therefore, the cost of a link $\{ij\}$ for $R_2(r)$ will be set according to equation 4.23.

$$C_2(ij) = \begin{cases} \infty & \{ij\} \in R_1(r) \\ \infty & else\, if\, b > A_{ij} \\ \epsilon & else\, if\, b \leq A_{ij} \end{cases} \tag{4.23}$$

After generating a backup route $R_2(r)$ for r, the connection request must be added to a protection set s using the TCP protection set assignment strategy. This operation has been summarized as figure 4.36.

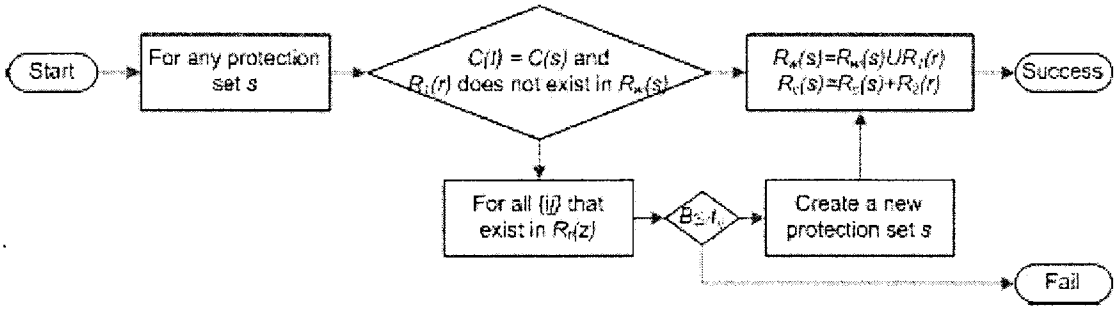


Figure 4.36.: Trunk Coded Protection Protection Set Assignment Strategy

There are two requirements for joining a pre-existing protection set.

1. The protection set must use the same coded trunk as the connection request.
2. The working route $R_1(r)$ of the connection request r must be disjoint from the pre-existing working routes $R_w(s)$ in the protection set s.

The availability of a protection set is determined by equation 4.24. For simplicity, the first protection set deemed available will be used be the connection request.

$$A_1(s) = \begin{cases} 0 & C(s) \neq C(t) \\ 0 & else\, if\, R_1(r) \in R_w(s) \\ 1 & otherwise \end{cases} \tag{4.24}$$

If no protection set is available, the algorithm will attempt to generate a new one for the connection request. In order to generate a new protection set, there must be enough

available capacity on the trunk to handle an additional protection set. This availability requirement is quantized as equation 4.25. From that, a new protection set can be generated if $A(j) = 1$ for all links $\{ij\} \in R_t(z)$.

$$A(ij) = \begin{cases} 0 & b \leq A_{ij} \\ 1 & otherwise \end{cases} \tag{4.25}$$

Following that, the connection request can be added to the new or existing protection set by adding the working $R_1(r)$ and inter-zone backup $R_2(r)$ routes of the connection request r to the working $R_w(s)$ and backup $R_c(s)$ routes of the protection set s.

after the protection set assignment strategy for TCP, the algorithm will generate the intra-zone backup route using a variant of SPS. In the variant, the amount of reserved capacity required on a link $\{ij\}$ for the inter-zone backup route $R_3(r)$ is determined by equation 4.26. Since all traffic demands in a protection set s use the same resources in the shared section $R_s(s)$, the union of their working routes $R_w(s)$ must be used for all capacity requirement calculations. Therefore the amount of shared capacity required on link $\{ij\}$ for a new connection request r is determined by equation 4.26.

$$T_{ij}(r, s) = b + max_{\forall\{kl\} \in R_w(s)}[\theta_{\{kl\}\{ij\}}] \tag{4.26}$$

Adding to that, since each traffic demand in a protection set enters the network at a different time and the link costs are based on the union of all working routes in the protection set, the cost of a link for the intra-zone route is slightly different from the backup route for SPS. For TCP, a link along an intra-zone route for a new connection request can be reserved without additional capacity if the link is already being used for another intra-zone route $R_3(r)$ by a traffic demand in the protection set. Thus the link costs for the inter-zone

backup route can be determined using equation 4.27.

$$C_3(ij) = \begin{cases} \infty & \{ij\} \in R_1(r) \\ \epsilon & else\,if\ \{ij\} \in R_s(s) \\ \epsilon & else\,if\,T_{ij}(r,s) \leq B_{ij} \\ T_{ij}(r,s) - B_{ij} & else\,if\,T_{ij}(r,s) - B_{ij} \leq A_{ij} \\ \infty & otherwise \end{cases} \tag{4.27}$$

The operation of TCP is further summarized as algorithm A.7 in appendix A. To ease in the understanding of the algorithm, it has been split into three sections: the main algorithm, the intersection route technique, the protection set assignment strategy.

4.3.3. Stream Based Network Coded Protection

Stream Based Network Coded Protection (SBNCP) is a novel heuristic algorithm based on the *Stream Based Sharing* (SBS) concept. With the SBS concept, SBNCP generates the coded section and the shared section of the coded route simultaneously. This avoids the inefficiency of the two step process usually required to create the shared and coded sections of the backup route. SBS can be considered a method of conceptualizing available shared capacity using graph transformations. The general outline for SBS is depicted in figure 4.37.

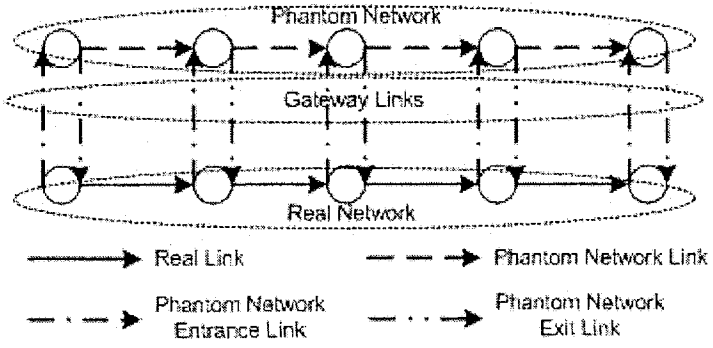


Figure 4.37.: Stream Based Sharing Concept

In the figure, the links $P_l(s)$ in the phantom network $P_n(s)$ represents the possibility of establishing a connection over shared capacity on links in J with the protection set s. Separating the phantom network from the real network are entrance $P_e(s)$ and exit $P_x(s)$ links. These phantom links can have their costs manipulated to control their availability. That is, if a connection request meets the necessary constraints to use the shared capacity with the protection set s on link $\{ij\}$, the cost of the corresponding link in the phantom network will be made acceptable.

For SBNCP, the general phantom network $P_n(s)$ appearance is depicted in figure 4.38. In the figure, for each protection sets s there is a phantom network $P_n(s)$ of all the coded $R_c(s)$ and shared $R_s(s)$ nodes and links. The cost of using the links in this phantom network $P_n(s)$ of a protection set s is set to some small number ϵ.

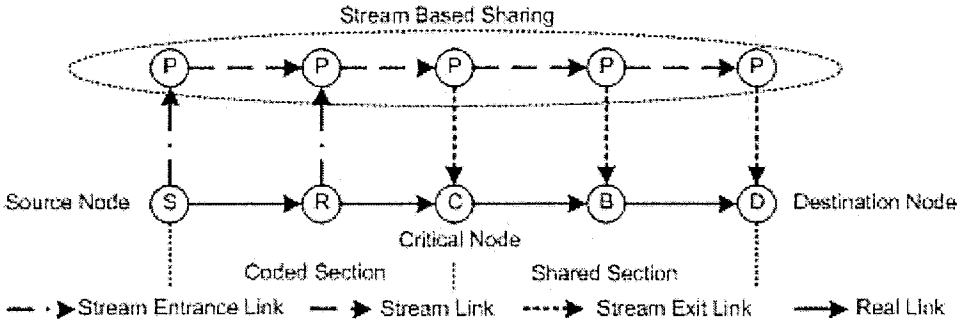


Figure 4.38.: Stream Based Network Coded Protection Appearance

The availability of using the protection set for survivability is determined by the cost of links in the phantom networks. To ensure that the working route $R_1(r)$ of a connection request do not attempt to use any of the shared capacity, access through the phantom network is

denied. This is done by setting the cost of all the phantom links to ∞. For the backup route $R_2(r)$ of the connection request, the phantom links must be setup so that the path can only enter the phantom network in coded section $R_c(s)$ and exit in the shared section $R_s(s)$.

SBNCP provisions a new connection request r similarly to NCP and SBNCP. The operation of SBNCP can be reduced into the following steps.

1. Generate a primary route $R_1(r)$ between the source $S(r)$ and destination $D(r)$ of the connection request r.
2. Using the protection set availability equation $A_2(s)$, determine which protection sets s are available for the connection request.
3. Modify the link costs $C_2(j)$ for the backup route $R_2(r)$ so that it can find a path through the phantom network $P_n(s)$, for all available protection sets..
4. Route the backup route $R_2(r)$ between the source $S(r)$ and destination $D(r)$ of the connection request r.
5. Translate the backup route $R_2(r)$ through the phantom network into coded $R_c(r)$ and shared $R_s(r)$ links in the real network.
6. Route the branch connection $R_3(r)$ between the source $S(r)$ of the connection request r and the nodes in the coded section $R_l(s)$ of the protection set s.

This operation is summarized as a flowchart in figure 4.39.

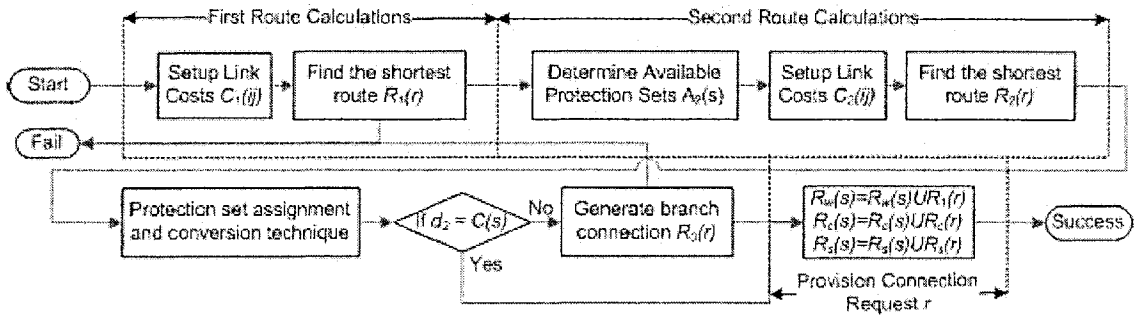


Figure 4.39.: Stream Based Network Coded Protection Flowchart

Like all algorithms, SBNCP must first generate a working route $R_1(r)$ for the connection request r. Thus, it must be routed over dedicated capacity and not shared capacity represented by a phantom network $P_n(s)$. To conceptualize this, the cost of a link $C_1(ij)$ that exists in the phantom network $P_n(s)$ is ∞. Otherwise the link cost will be controlled by the available capacity on the link. To summarize this the cost of a link $\{ij\}$ for the first route is determined by equation 4.28.

$$C_1(ij) = \begin{cases} \infty & j \in P_n \\ \infty & else\, if\, b > A_{ij} \\ \epsilon & else\, if\, b \leq A_{ij} \end{cases} \tag{4.28}$$

Following the generation of the first route $R_1(r)$, the algorithm determines which protection sets are available for sharing. In order for one to be available, the first route $R_1(r)$ must be disjoint from the shared $R_s(s)$, coded $R_c(s)$, and working $R_w(s)$ links of the protection set s. This is conceptualized as the protection set availability equation in equation 4.29.

$$A_2(s) = \begin{cases} 0 & R_1(r) \in (R_s(s) \cup R_c(s) \cup R_w(s)) \\ 1 & otherwise \end{cases} \tag{4.29}$$

By using the protection set availability equation, the second route $R_2(r)$ can be designed to take advantage of the shared capacity in the phantom networks. This is done by manipulating the cost of the links in the phantom network of available protection sets. Since these phantom links are essentially free, they are assigned the cost of some small number ϵ. To numerically distinguish this from the cost of a standard link, they will be assigned a different number β, where $\infty \gg \beta \gg \epsilon$. From that, the link costs for the second route $R_2(r)$ can be determined by equation 4.30.

$$C_2(ij) = \begin{cases} \epsilon & \{ij\} \in P_n(s)\, and\, A_2(s) = 1 \\ \infty & else if\ \{ij\} \in R_1(r) \\ \infty & else if\ b > A_{ij} \\ \beta & else if\ b \leq A_{ij} \end{cases} \tag{4.30}$$

After generating the second route $R_2(r)$, it may contain phantom links that exist in $P_l(s)$. These links represent shared capacity on a protection set s. These links need to be converted into real links $R[ij]$ that are part of the coded $R_c(s)$ and shared $R_s(s)$ sections of a protection set s. Figure 4.40 has been employed to explain the operation. The figure depicts the protection set assignment and conversion technique.

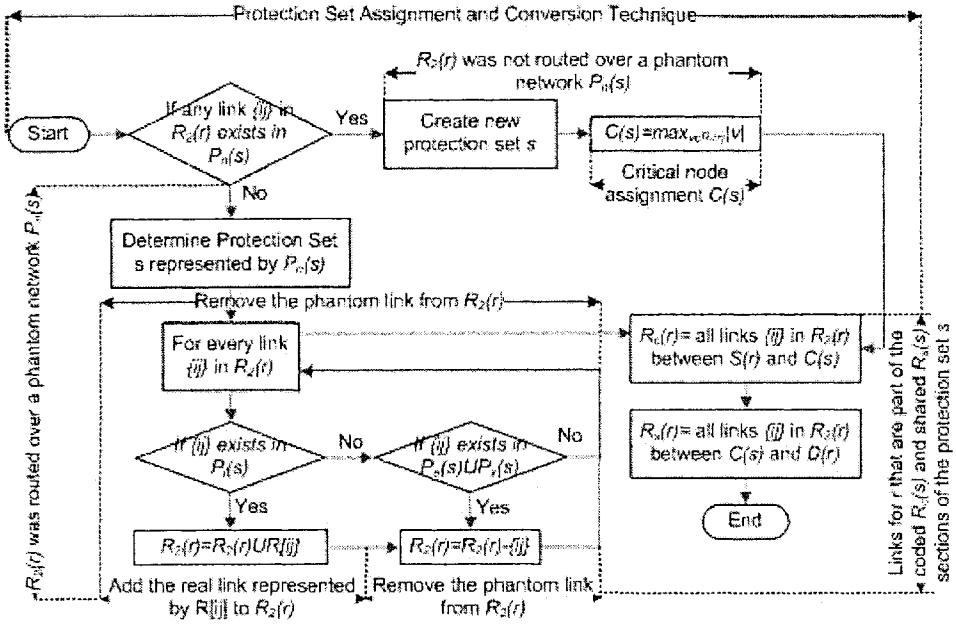


Figure 4.40.: Stream Based Network Coded Protection Protection Set Assignment and Conversion Technique

This technique has two directions based on the second route $R_2(r)$. Either $R_2(r)$ contains

phantom links that exist in $P_l(s)$ or it does not. If the former of the two options is true, then the shared resources of a protection set have been used by the connection request. In that case, there are two operations are required to convert it into links for a protection.

1. Determine the protection set represented by the phantom links $\{ij\}$.
2. Convert phantom links $\{ij\}$ into their real link equivalents $R[ij]$.

The first of these operations is required to determine which protection set s they belong too. This is easily done using a unique identifier of the link that determines which protection set it belongs to. Likewise, another identifier can be used to determine which real link $R[ij]$ is represented by $\{ij\}$. With that information, $R_2(r)$ can be modified so that all phantom links are replaced with real links. This is done by adding all links $R[ij]$ represented by $\{ij\}$ to $R_2(r)$ and removing all phantom intra-network $P_l(s)$, entrance $P_e(s)$, and exit $P_x(s)$ links from $R_2(s)$.

The other direction taken by the technique is for the instance where $R_2(r)$ does not contain any phantom inter-network links $\{ij\}$. In this circumstance, no protection set was used for the connection request. Therefore, the connection must create a new protection set s. $R_2(r)$ will be used to generate the coded $R_c(s)$ and shared $R_s(s)$ sections of the new protection set s. Along this route, the critical node *C(s)* is assigned based on equation 4.31. that is, the critical node *C(s)* for the new protection set s will be set to the highest degree node along $R_2(r)$.

$$C(s) = max_{v \in R_2(r)} |v| \tag{4.31}$$

Independent of which direction the protection set assignment and conversion technique takes for the connection request, it must subdivide the second route $R_2(r)$ into coded $R_c(r)$ and shared $R_s(r)$ routes. The coded route $R_c(r)$ is created from all the links j between the

source $S(r)$ and the critical node $C(s)$. Likewise, the shared route $R_s(r)$ is created from all the links j between the critical node $C(s)$ and the destination $D(r)$.

After routing the backup route for the connection request a secondary connection must be generated. For SBNCP, the secondary connection is a branch connection. Figure 4.41 depicts the set of operations required to generate the branch connection.

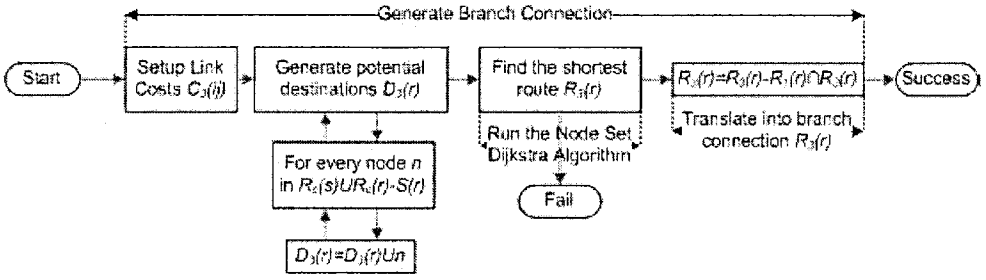


Figure 4.41.: Stream Based Network Coded Protection Branch Connection Generator

First off, the branch connection link costs $C_3(ij)$ must be setup such that it connects from the working route $R_1(r)$ to one of the nodes in the coded section $R_c(s)$ of the protection set s. This can be done by setting the cost of the links used by $R_1(r)$ to be a really small number ϵ compared to a standard link cost of β, where $\infty \gg \beta \gg \epsilon$. This link cost for the branch connection is defined by 4.32.

$$C_3(ij) = \begin{cases} \infty & \{ij\} \in P_n \\ \epsilon & else if\ \{ij\} \in R_1(r) \\ \infty & else if\ b > A_{ij} \\ \beta & else if\ b \leq A_{ij} \end{cases} \tag{4.32}$$

A depth search first routing algorithm like Dijkstra's algorithm will proceed up the shortest route defined by $R_1(r)$ until a short branch can be formed that connects to the coded

section $R_c(s)$ of the protection set s. The potential destinations for the node set Dijkstra's operation are presented as equation 4.33.

$$D_3(r) = R_c(s) \cup R_c(r) - S(r) \tag{4.33}$$

In order to prevent branches from occurring from the source node, it is removed from the potential destinations $D_3(r)$. Afterward, the third route $R_3(r)$ can be transformed into a branch connection using equation 4.34.

$$R_3(r) = R_3(r) - R_3(r) \cap R_1(r) \tag{4.34}$$

Equation 4.34 removes all links that are used by both $R_1(r)$ and $R_3(r)$ from $R_3(r)$. This creates a route that has a source that is one of the nodes along $R_1(r)$ and a destination that will be in the coded section $R(s)$ of the protection set s.

The SBNCP heuristic algorithm is also depicted in algorithmic form as algorithm A.8 in appendix A. To simplify the algorithm, the protection set assignment and connection technique and branch connection generator are presented after the main algorithm.

5. Results

5.1. Simulated Algorithms

In this chapter, the heuristic algorithms of chapter 4 will be compared with the benchmark algorithms of chapter B. The algorithms were simulated using an in house C++ simulation model. Each of the eight proposed algorithms in chapter 4 are based off one of the three different protection schemes proposed in chapter 3. All of these algorithms and schemes are depicted together with the benchmark algorithms in figure 5.1.

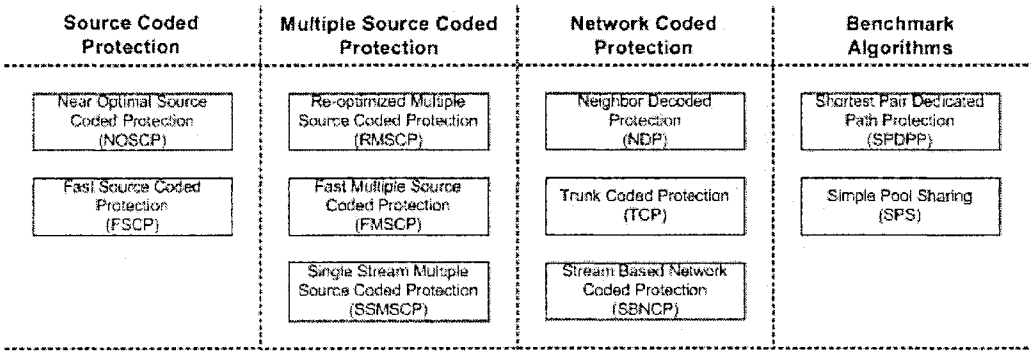


Figure 5.1.: Simulated Algorithms

5.2. Performance Criteria

There are five metrics used in this thesis to compare the proposed and benchmark algorithms. These performance metrics are depicted in table 5.1. These five metrics together provide details on the effectiveness, efficiency, and recovery speed of each survivability algorithm.

Table 5.1.: Simulation Performance Metrics

Metric	Definition
Availability	Average percentage of a demand's life that is spent in an operational state
Restoration Failure Rate	Probability that a demand will not be restorable when its primary path fails
Hop Requirement	Average number of hops required to provision the working and backup routes of a traffic demand.
Demand Blocking Probability	The probability that a traffic request can not be provisioned in the network.
Restoration Hops	Average number of hops along the restoration route for a traffic demand

The effectiveness of an algorithm can be determined with two metrics:

- Availability
- Restoration Failure Rate

The availability can be interpreted as the percentage of a traffic demand's life that is spent in a operational state. It is calculated in the simulation as equation 5.1. This metric provides a good indicator on how well a technique insulates traffic demands from failure events. For comparison purposes, the availability of each traffic demand in the network is averaged to generate an availability for the network.

$$Availability = \frac{Demand\,lifetime - Failure\,time}{Demand\,time} \tag{5.1}$$

As a complement to availability, the effectiveness of an algorithm is also determined by its restoration failure rate. The restoration failure rate is calculated as the probability that a traffic demand will be unrestorable when a fault event occurs. This provides another viewpoint of how effective a technique provides survivability to a network. For the simulations,

the restoration failure rate is calculated as equation 5.2. Like availability, the restoration failure rate is averaged among all traffic demands in the network for the results displayed in the thesis. With this metric we can compare the frequency of failures as opposed to just the percentage of time a traffic demand is available.

$$Restoration\,Failure\,Rate = \frac{Unsuccessful\,restoration\,events}{Restoration\,events} \tag{5.2}$$

The efficiency of an algorithm measures how much resources are required to provide survivability to traffic demands. In these simulations, the hop requirement is used to determine the efficiency. The hop requirement is the number of links required to provide the working and backup path from the source to the destination node of a traffic demand. For shared capacity, the first traffic demand to require the capacity records it in its hop requirement. Subsequent, traffic demands using the shared capacity reserve it without cost. Like all the previously mentioned metrics, the hop requirement of each traffic demand in the network is averaged to generate the mean hop requirement for a traffic demand in the network. In the specific case for SCP based algorithms (NOSCP and FSCP), the hop requirement is modified into equation 5.3. This is done because the SCP based algorithms both fragment traffic demands into a smaller size. This allows a proper comparison of the SCP based algorithms.

$$SCP\,Hop\,Requirement = Hop\,Requirement \times Fragement\,Size \tag{5.3}$$

The demand blocking probability is an important metric for determining how often a survivability technique can generate traffic demands. It measures the number of times, traffic demands were accepted and rejected during the life of a network. Algorithms that have more constraints on routing tend to have higher demand blocking probabilities. It is calculated

using equation 5.4.

$$Demand\,Blocking\,Probability = \frac{Rejected\,demands}{Accepted\,demands + Rejected\,demands} \tag{5.4}$$

This metric is important because some techniques require specific conditions in order to provide survivability. For example, the SCP based algorithms both require at least three disjoint connections between the source and destination nodes. This is not always possible and is reflected as a high blocking probability for SCP based algorithms. Furthermore, some algorithms may be greedy in the use of certain links in the network. For example, TCP uses pre-defined trunks. Once the capacity of these pre-defined trunks is reached, no more traffic demands can be provisioned.

The last and arguably most important metric for showing the advantages of coded based protection is the number of restoration hops. With restoration based algorithms, when a fault event occurs on the working path of a traffic demand, a backup route must be provisioned. This time critical endeavor generally produces a time delay proportional to the number of hops that must be provisioned along the backup route. Because of this concept, the average number of restoration hops is recorded from each technique for comparison.

5.3. Network Maps

In these simulations, two network maps have been utilized. The first of these maps is the *National Science Foundation Network* (NSFNET). This network is depicted in figure 5.2. NSFNET contains 16 nodes interconnected by 25 bidirectional links. These links have an average fiber length of 733 Miles. From this, the network has an average nodal degree of 3.13 and a highest nodal degree of 4. These statistics are summarized in table 5.2.

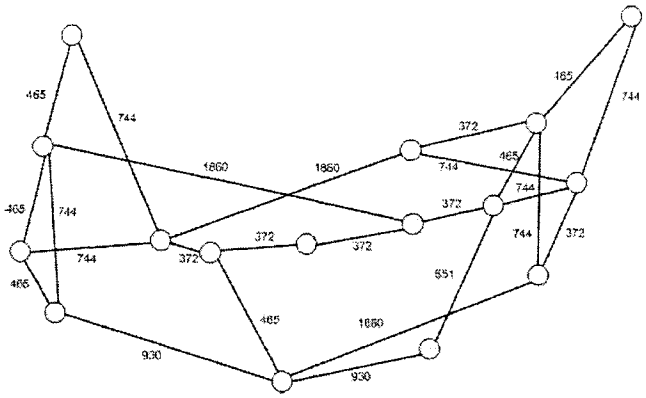


Figure 5.2.: National Science Foundation Network Map

Table 5.2.: National Science Foundation Network Statistics

Links	Nodes	Average Nodal Degree	Highest Nodal Degree	Average Fiber Length (Miles)
25	16	3.13	4	733

For TCP, the NSFNET had to be adjusted so that it contained zones and inter-zone trunks. These zones and trunks are depicted in figure 5.3. The network has been subdivided into three zones: Zone A, Zone B, and Zone C.. Each zone has a trunk that interconnects it with the other two zones.

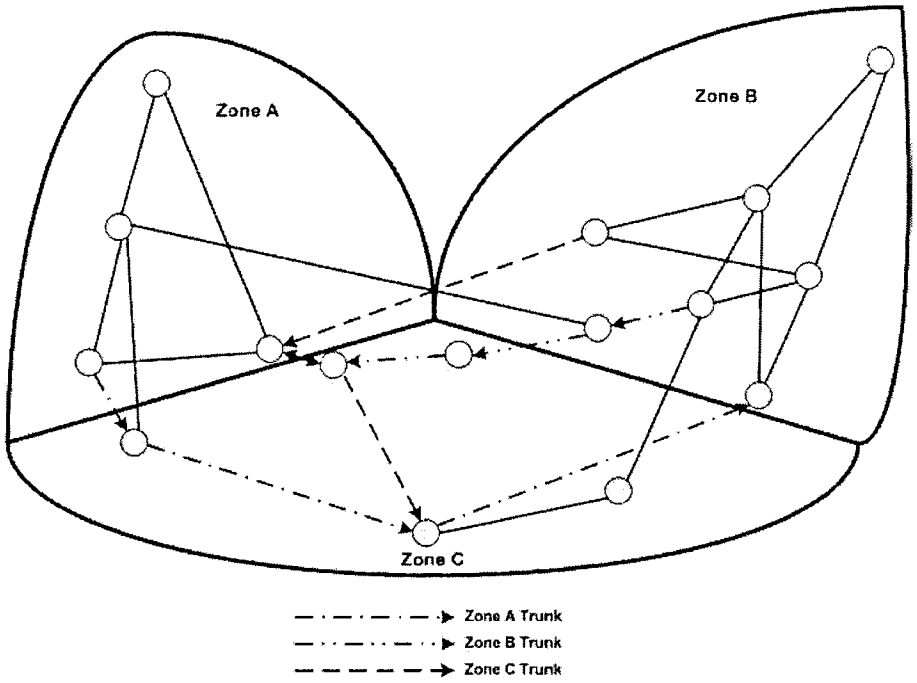


Figure 5.3.: National Science Foundation Network Zones and Trunks for Trunk Coded Protection

In addition to the NSFNET, all algorithms have also been simulated on the *Global Crossing Network* (GCN).The GCN is depicted in figure 5.4. This network is much larger than the NSFNET and will require many more hops to route a traffic demand from source to destination. In the GCN, there are 27 nodes and 38 bidirectional links. Like the NSFNET, the maximum nodal degree in the GCN is 4. However, the ratio of nodes to links is smaller in the GCN. In the GCN the average nodal degree is 2.81, much lower than the 3.13 average nodal degree in the NSFNET. Additionally, the average link length in the network is only 440 Miles, much smaller than the 733 Miles required in the NSFNET. These statistics are depicted in table 5.3.

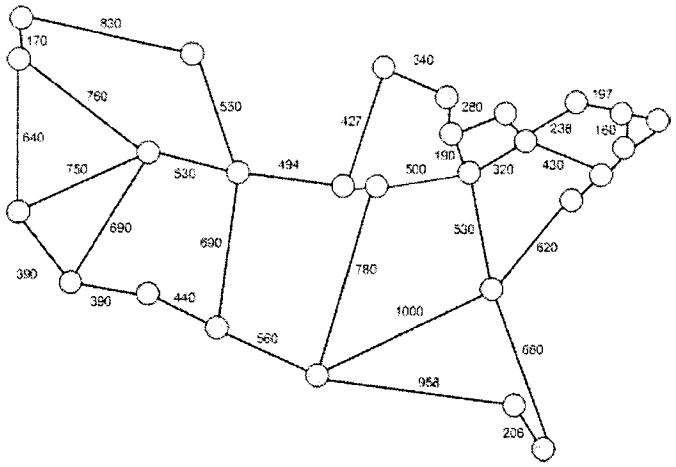


Figure 5.4.: Global Crossing Network Map

Table 5.3.: Global Crossing Network Statistics

Links	Nodes	Average Nodal Degree	Highest Nodal Degree	Average Fiber Length (Miles)
38	27	2.81	4	440

Like the NSFNET, to generate TCP, the GCN had to be modified so that it contained zones and inter-zone trunks. These zones and trunks are depicted in figure 5.5. Like the NSFNET, The GCN has also been subdivided into three zones: Zone A, Zone B, and Zone C. Each zone has a trunk that interconnects it with the other two zones.

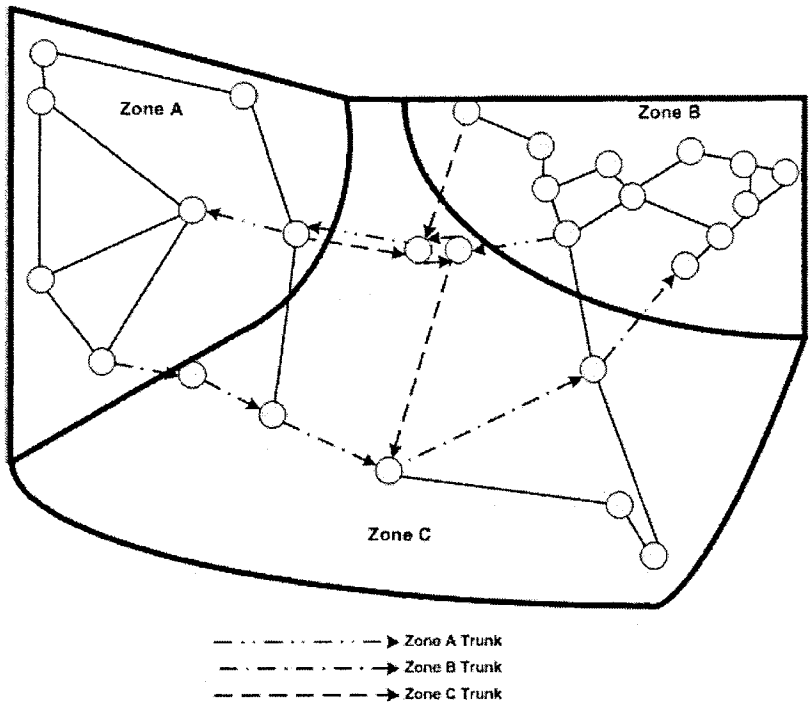


Figure 5.5.: Global Crossing Network Zones and Trunks for Trunk Coded Protection

5.4. Network Model

In order to properly produce a set of metrics to compare the algorithms in each network, a set of simulation statistics is required. These statistics are as follows:

- Each link in the network consists of two directional links between the adjacent nodes.
- All directional links have a capacity of 1 Gbps. This was done to maintain backwards compatibility with previous research in the field.
- The failure rate of each link is based on 500 $^{\text{FITs}}/_{\text{Mile}}$. This value is based on the crawford study in reference [26].
- A link failure affects both directional links contained within it. This is done as opposed to allowing directional links to fail independantly. It is our belief that fault events will

affect both directional links simultaneously. Thus this technique will more accurately represented failure events.

- All link failures are independant of each other, creating the potential for multiple failures in the network. This allows the algorithms to be simulated under more realistic conditions

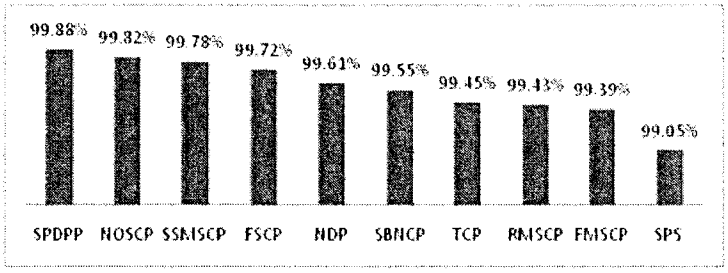
- 10 hours are required to restore a failed link. This was done to maintain backwards compatibility with previous research in the field.
- Traffic demands are generated between a random source node and random destination node in the network.
- All traffic demands require a capacity of 100 Mbps. This was done to maintain backwards compatibility with previous research in the field.
- The simulation and all traffic demands are held in the network for 1 year.
- Each survivability algorithm is simulated 5 times for each network with 30-70 traffic demands, with a total of 400 iterations per algorithm. The 30-70 traffic demands correspond roughly to network loads of between 25%-80%. However, this number varies between algorithms, due to thier capacity requirements.

5.5. Simulation Results

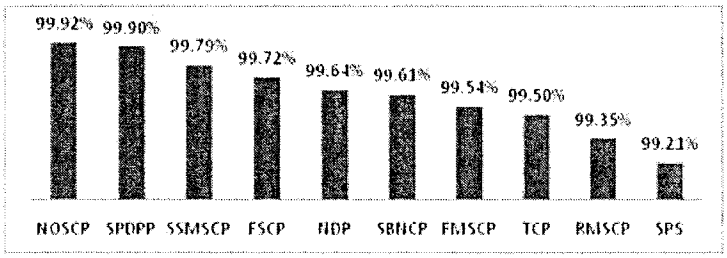
5.5.1. Availability

The average traffic demand availability provides a good metric of how effectively a scheme protects a demand from failure events. In single failure simulations, since all of the algorithms use disjoint routes, the availability would be 100%. However, in these simulations, multiple failures are not prohibited. This allows us to compare how the schemes and algorithms would perform when failures beyond their level of protection occur. The average availability for traffic demands in the NSFNET are presented in figure 5.6a. Likewise, the average availability for traffic demands in the GCN are presented in figure 5.6b. Since, the availability of traffic demands are generally independent of the number of demands in the network, their results are presented as a bar graph. The following section, will explain

the results for each of the different proposed algorithms compared with the benchmark algorithms.



(a) National Science Foundation Network



(b) Global Crossing Network

Figure 5.6.: Average Availability

5.5.1.1. Source Coded Protection

The two algorithms NOSCP and FSCP produce the same form of SCP and therefore produce comparable results. Furthermore, since SCP based algorithms do not allow sharing between multiple traffic demands, the both FSCP and NOSCP produces results similar to SPDPP. Unfortunately, since the SCP based algorithms split their capacity requirement onto many routes, the number of links traversed by an average demand is increased over SPDPP. Since more links generally means a greater probability of failure, the SCP algorithms produce results that are slightly weaker than SPDPP. Since FSCP can produce less efficient results

when links become congested, it requires more hops and therefore has a lower availability

5.5.1.2. Multiple Source Coded Protection

Availability is generally lower for MSCP based algorithms (RMSCP, FMSCP, SSMSCP) than in SCP based algorirthms. This is because sharing of resources between traffic demands is allowed. Thus, traffic demands in a protection set can utilize protection resources, preventing other demands from being recovered in failure events. This culminates in a reduced availability when compared to the SCP algorithms and SPDPP. Generally, in networks that can have multiple failures, lower sharing translates into higher availability. Since the number of traffic demands in a protection set are limited by the nodal degree of the destination node, sharing is kept to a minimum in MSCP.

The highest availability of all MSCP algorithms is seen with SSMSCP, which provides an average availability to traffic demands that approaches the SCP algorithms. This is because sharing is limited to only the designated backup route of the protection set. On the other hand, RMSCP and FMSCP both allow sharing with any of the routes in the protection set. This increases the sharing potential and reduces the availability. This method of sharing along any of the routes in the protection set is only used in RMSCP and FMSCP and not any of the other proposed or benchmark algorithms. Thus, the availability of RMSCP and FMSCP are reduced so that they are more comparable with SPS.

5.5.1.3. Network Coded Protection

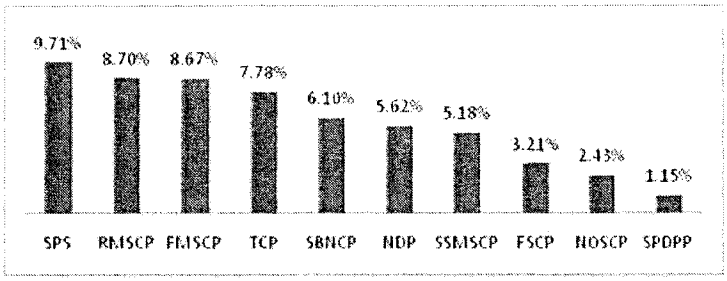
The availability of the NCP based algorithms (NDP, TCP, SBNCP) derive their statistics from their foundation of SSMSCP, in that they only allow sharing along backup routes. This gives them a relatively high availability compared to SPS. However, since sharing is not as constrained for the NCP algorithms as it is for SSMSCP, the results do become

weaker. Both NDP and SBNCP have average demand availabilities that are almost as good as SSMSCP. The reduction is mainly caused by the loss of the destination nodal degree constraint imposed on MSCP algorithms.

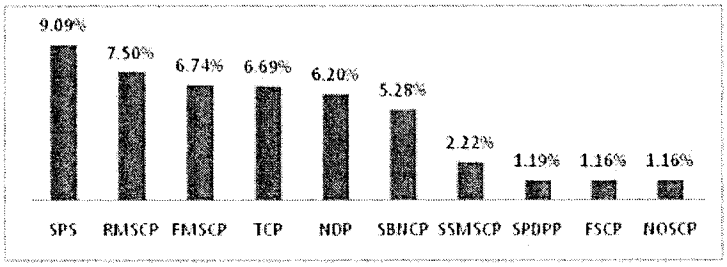
TCP produces the lowest availability of the NCP based algorithms because it contains a pool sharing based component. Unlike the other two NCP based algorithms, TCP permits sharing between traffic demands that are not in the same protection set. This improves the efficiency of the algorithm at the cost of availability.

5.5.2. Restoration Failure Rate

The average restoration failure rate is a measure of the probability that a demand can not be restored in a failure event. It is closely related to the availability of a traffic demand, as it influences how many times a demand will be unrestorable during failure events. The average restoration failure rate results are presented as a bar graph in figure 5.7. Within the figure, the average restoration failure rate for the NSFNET and GCN are presented in figures 5.7a and 5.7b respectively. The following section explains the results exhibited by each of the algorithms presented in this thesis.



(a) National Science Foundation Network



(b) Global Crossing Network

Figure 5.7.: Restoration Failure Rate

5.5.2.1. Source Coded Protection

Both NOSCP and FSCP produce low restoration failure rates. This is because both of them are not sharing resources with other traffic demands. Therefore the only way a SCP based algorithm can fail to restore a traffic demand in a failure event, is if two of its disjoint routes have failed. This situation is highly unlikely, giving SCP algorithms a low restoration failure rate. This failure rate is still higher than SPDPP, which only has two routes that can fail, as opposed to the N possible routes for SCP based algorithms.

5.5.2.2. Multiple Source Coded Protection

The restoration failure rate for the MSCP based algorithms (RMSCP, FMSCP, SSMSCP) form the middle ground between the low failure rates of SPDPP and the higher failure

rates of SPS. Like the availability metric, the limited sharing potential of MSCP prevents competition in protection sets for redundant resources. This is most apparent in SSMSCP, which because of its increased constraints on sharing, produces the lowest failure rate of the MSCP algorithms.

Similar to the availability results, the increased sharing potential of RMSCP and FMSCP increase their restoration failure rates over SSMSCP. From them, since RMSCP re-optimizes traffic demands so more sharing can occur, it produces the greatest level of sharing and therefore the highest restoration failure rate. This gives it a restoration failure rate closer to that of SPS.

5.5.2.3. Network Coded Protection

NDP, TCP, and SBNSCP are all based on SSMSCP, in that they have distinct backup and working routes. However, unlike SSMSCP all NCP algorithms include a shared section where bandwidth is reserved but not assigned. Of the three NCP based algorithms, NDP and SBNCP only allow sharing with traffic demands in a protection set. Thus, the possibility of failing to restore a connection in failure event is affected only by the other demands in the protection set. This allows NDP and SBNCP to produce restoration failure rates comparable to SSMSCP. Since NDP and SBNCP are less constrained than SSMSCP, they have a slightly higher restoration failure rate.

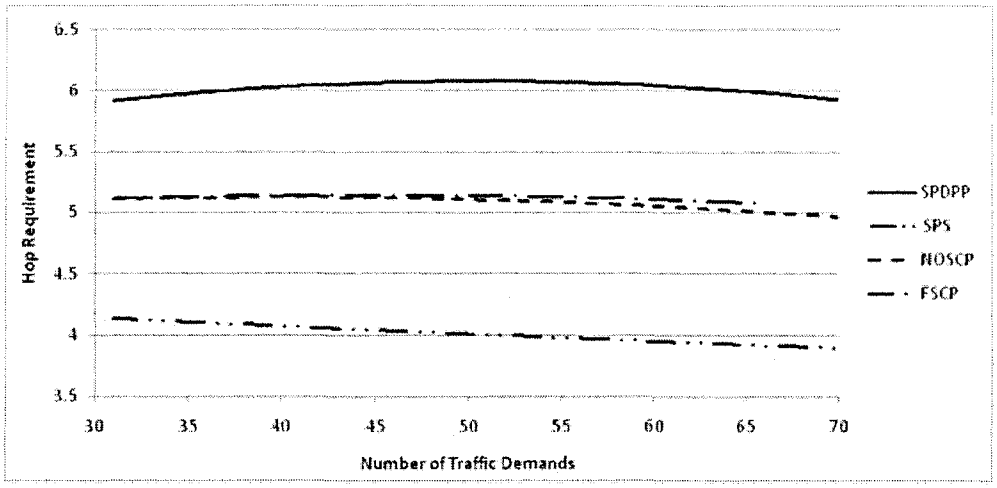
However, unlike the other two algorithms, TCP uses an additional link based sharing concept to route the intra-zone portion of its backup paths. This allows traffic demands that are not part of the protection set to preempt resources required to restore demands protection by TCP during multiple failure situations. Thus, TCP has a much higher restoration

failure rate than the other two algorithms. Since sharing is still constrained, the restoration failure rate for TCP is still not as high as SPS. It is in fact, comparable to the FMSCP and RMSCP, because all three have increased levels of sharing.

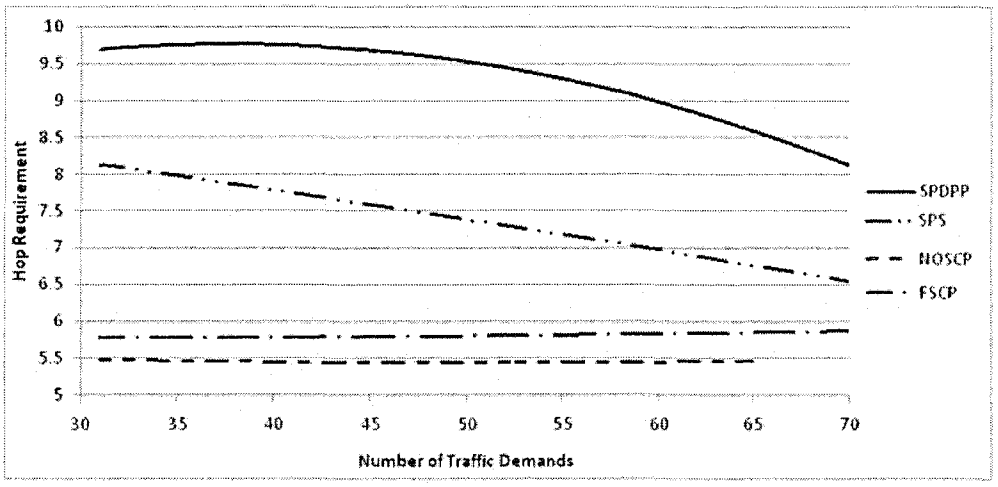
5.5.3. Hop Requirement

5.5.3.1. Source Coded Protection

The hop requirement results for NOSCP and FSCP are presented in figures 5.8a and 5.8b respectively. For the NSFNET the average hop requirement for the two algorithms fit neatly between SPDPP at the high end and SPS in the low end. In general, five hops worth of bandwidth are required for a traffic demand protected by NOSCP or FSCP. However, in the GCN, both FSCP and NOSCP seem to have performed far superior to SPS. Unfortunately, this result is skewed by the structure of the GCN. As depicted earlier in figure 5.6b, the GCN can be subdivided into two networks connected in the middle by two links. Since both SCP based algorithms require at least three disjoint paths between the source and destination, any cross network connection requests must have been declined. This had the unfortunate effect of giving both SCP based algorithms low hop requirements. This problem is further explained with the demand blocking ratio.



(a) National Science Foundation Network



(b) Global Crossing Network

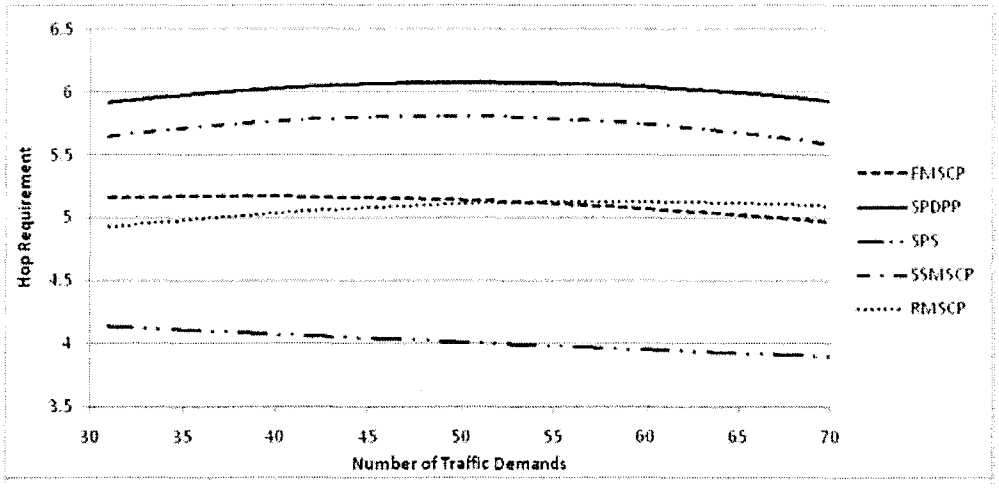
Figure 5.8.: Source Coded Protection Hop Requirement

5.5.3.2. Multiple Source Coded Protection

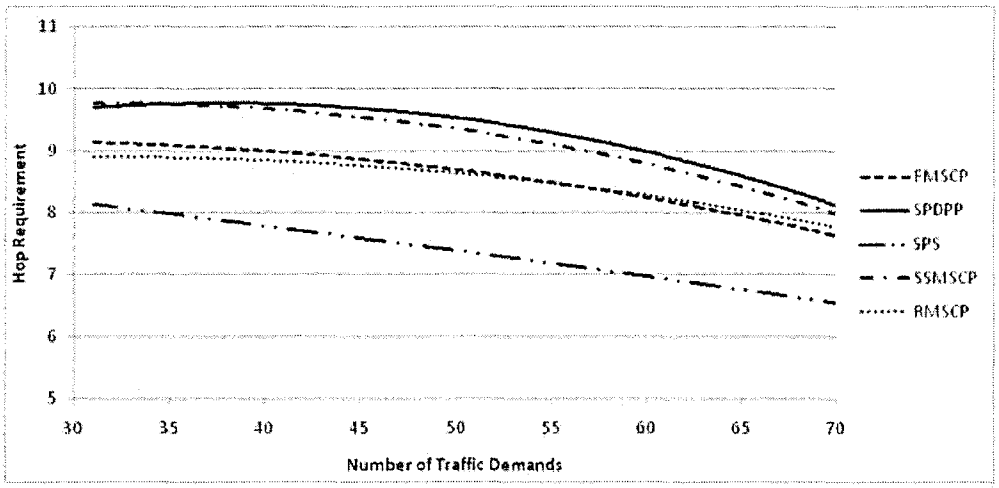
The hop requirements for the three MSCP algorithms (RMSCP, FMSCP, SSMSCP) vary widely between each other and in both of the networks. In the NSFNET depicted in figure 5.9a, the algorithms perform well. Closest to SPDPP is the SSMSCP algorithm. Since this

technique only allows sharing along the backup route, the sharing potential is quite limited. As such the reduction in bandwidth is minimal between it and the benchmark algorithm. On the other hand, the FMSCP algorithm reduced the bandwidth requirement by one full hop from SPDPP. In fact it is only one hop above the ideal SPS hop requirement. On the other hand, RMSCP did not perform as well as expected. This can be attributed to the fact that RMSCP does not unprovision traffic demands before re-optimizing them. This reduces the effectiveness of the algorithm but prevents blips in service when traffic demands switch to their new route. From this, as the network becomes more congested, RMSCP becomes increasingly less effective.

The results for the algorithms in the GCN are depicted in figure 5.9b. The advantages of all the algorithms over SPDPP are reduced in this network. This is most likely because of the increased number of nodes in the network. Since all MSCP based algorithms require the same destination node to share capacity, the increased number of nodes in the GCN reduce their efficiency. So much so, that SSMSCP is performs very similar to SPDPP. Fortunately though, both FMSCP and RMSCP perform well enough to fit comfortably in between SPDPP and SPS. However, RMSCP still maintains its efficiency problem that prevents it from clearly surpassing FMSCP.



(a) National Science Foundation Network



(b) Global Crossing Network

Figure 5.9.: Multiple Source Coded Protection Hop Requirement

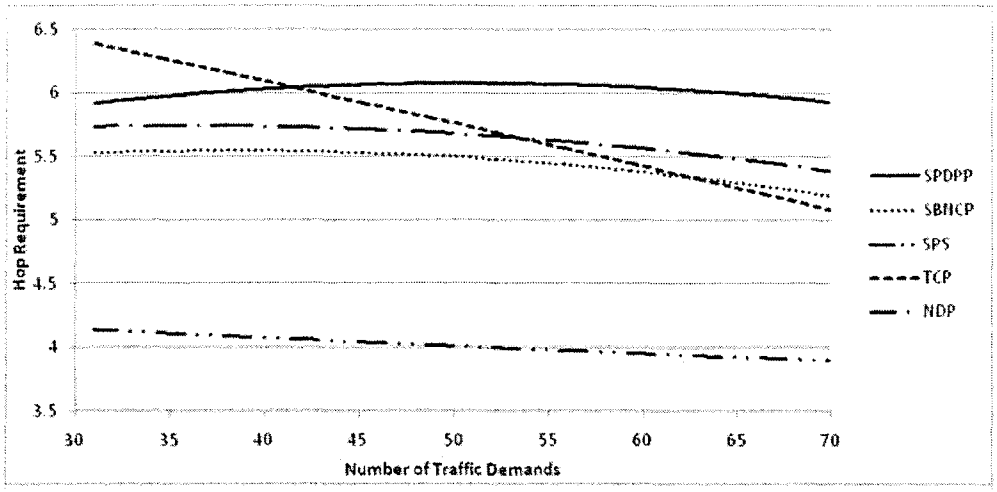
5.5.3.3. Network Coded Protection

The NCP based algorithms (NDP, TCP, SBNCP), because of their unique formations generate the most interesting results. Since all three NCP based algorithms have many similarities with SSMSCP, they inherit some of its aspects. However, each algorithm also has

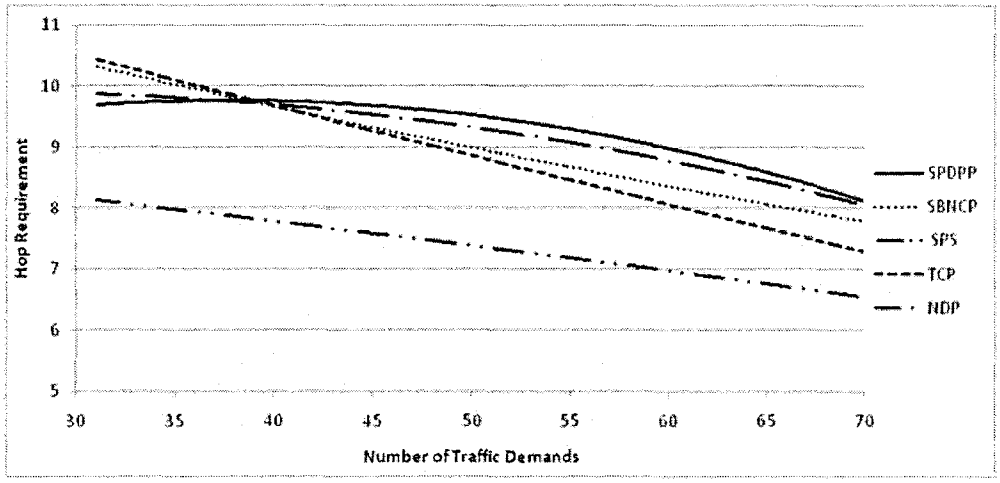
special characteristics which affect the hop requirements.

The results for the NCP based algorithms in the NSFNET are depicted in figure 5.10a. In these results, TCP is the most visible algorithm. It requires more hops than SPDPP when the network is lightly loaded. This is because TCP must provision an entire trunk for all inter-zone traffic demands that can not find an available protection set. Thus, under light loaded conditions SPDPP requires less hops. Fortunately though, this problem disappears as more traffic demands are added to the network. On the other hand, NDP and SBNCP both perform much better than SPDPP. Thus, the advantages of those two algorithms outweigh the constraints imposed on them, even under low network utilization.

The results for the NCP based algorithms in the GCN are depicted in figure 5.10b. The one aspect of the results which garners the most interest is the hop requirements of each of the algorithms under low network utilization. All of the algorithms require more hops than SPDPP when the network is lightly loaded. This is because all of the NCP algorithms require secondary connections, which increases the hop requirements. Under light loaded conditions, this secondary connection outweighs the capacity reduction advantages of the algorithms. Fortunately though, this problem is quickly mitigated as more traffic demands are added to the network. Of the three algorithms, TCP appears to perform the best. However, this can be partially attributed to the SPS based intra-zone traffic demands in the network. In reality, TCP probably performs much like SBNCP in the GCN.



(a) National Science Foundation Network



(b) Global Crossing Network

Figure 5.10.: Network Coded Protection Hop Requirement

5.5.4. Demand Blocking Probability

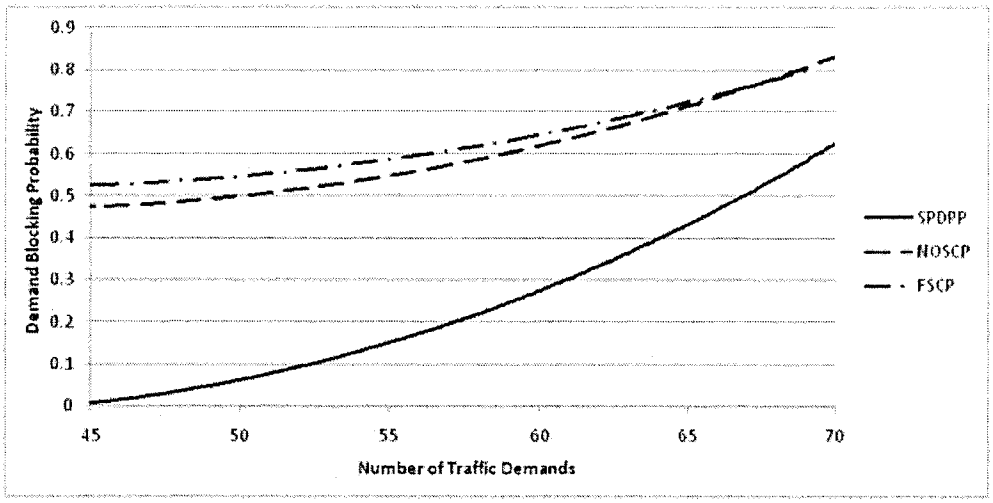
The demand blocking probability is the probability of declined traffic demands over accepted traffic demands. Because SPS does not experience network congestion at the same point as the proposed algorithms, it has been omitted from some of the results.

5.5.4.1. Source Coded Protection

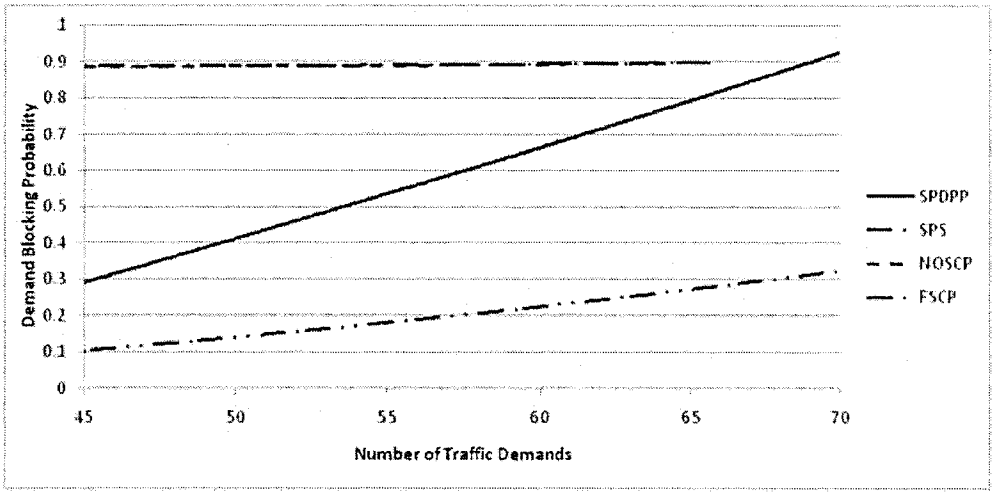
The demand blocking probability is where SCP based algorithms (FSCP, NOSCP) under perform. This is because they require at least three disjoint routes between the source and destination nodes. If either the source or destination node of a traffic demand do not have a nodal degree of three, then it must be declined. This generates blocking events even when the network is not congested. Thus, the demand blocking probability for SCP based algorithms in the NSF network depicted in figure 5.11a is much higher than either of the benchmark algorithms. A special case occurs in the GCN in figure 5.11b. In the GCN, the network can be subdivided into two halves connected by two links. Two links does not provide enough disjoint routes for traffic demands with source and destination nodes in difference halves of the network. Thus, all these demands are blocked by the SCP algorithms. This provides the SCP algorithms with a significantly greater demand blocking probability. Therefore, only the NSF results for SCP will be compared with the benchmark algorithms.

Demand blocking probability is also one instance where FSCP and NOSCP perform differently. As mentioned in chapter 4, NOSCP adjusts the link costs between path set search iterations while FSCP does not. Because of this, FSCP requires significantly less iterations than NOSCP. However, it also prevents FSCP from taking advantage of congested links where a fragmented traffic can fit but an unfragmented traffic demands can not fit. This increases the demand blocking probability of FSCP over NOSCP. This attribute is most apparent in results for the NSFNET. In the NSFNET, under the lighter load conditions FSCP has a higher blocking probability. As an unintended result of FSCP having a higher blocking probability in the light loaded area, it favors traffic demands that are easier to route. This slightly improves the blocking probability of FSCP under congested conditions. However, when the network becomes congested, both FSCP and NOSCP still produce unacceptable blocking probabilities. As mentioned earlier, pairing the SCP algorithms with

SPDPP would significantly reduce this problem in practical applications.



(a) National Science Foundation Network

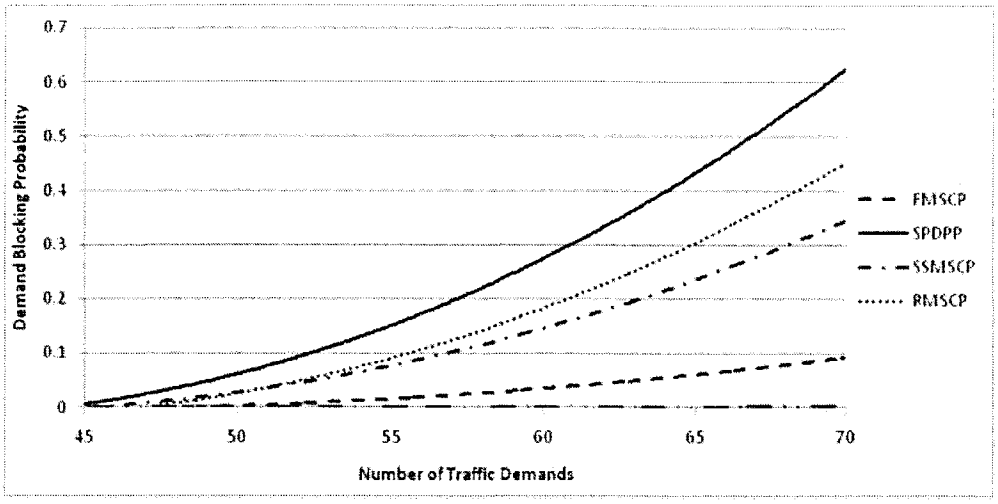


(b) Global Crossing Network

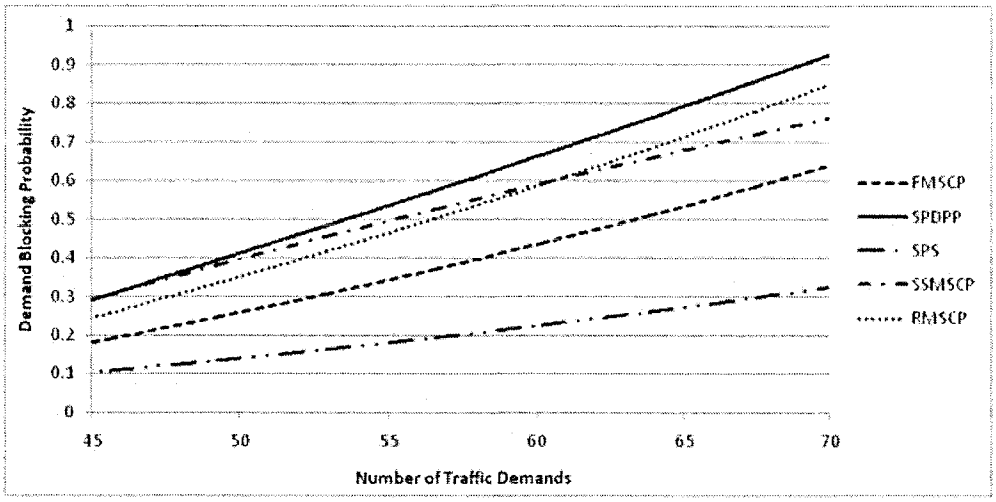
Figure 5.11.: Source Coded Protection Demand Blocking Probability

5.5.4.2. Multiple Source Coded Protection

MSCP algorithms (RMSCP, FMSCP, SSMSCP) were all designed so that traffic demands with the same destination could share resources. Thus, they all have significantly smaller blocking probabilities than SPDPP. The simulation results for the NSFNET and GCN are depicted in figures 5.12a and 5.12b respectively. For these results both networks produced similar results excluding the different magnitudes. The best performer of the MSCP algorithms is FMSCP. It produces results that get reasonably close to that of SPS. This can be attributed to the availability of all paths in the protection set for sharing opportunities. In the middle, between FMSCP and RMSCP is SSMSCP. Because SSMSCP could only use the designated backup route to share capacity, it had to block more traffic demands than FMSCP. Like always, RMSCP produced results that were unexpected at the time. Since RMSCP attempts to re-optimize all the traffic demands in the protection set without releasing capacity, it has added difficultly sharing resources. This results in RMSCP favoring many small protection sets instead of a few large protection sets. The latter of these two would be significantly more efficient and reduce the blocking probability.



(a) National Science Foundation Network



(b) Global Crossing Network

Figure 5.12.: Multiple Source Coded Protection Demand Blocking Probability

5.5.4.3. Network Coded Protection

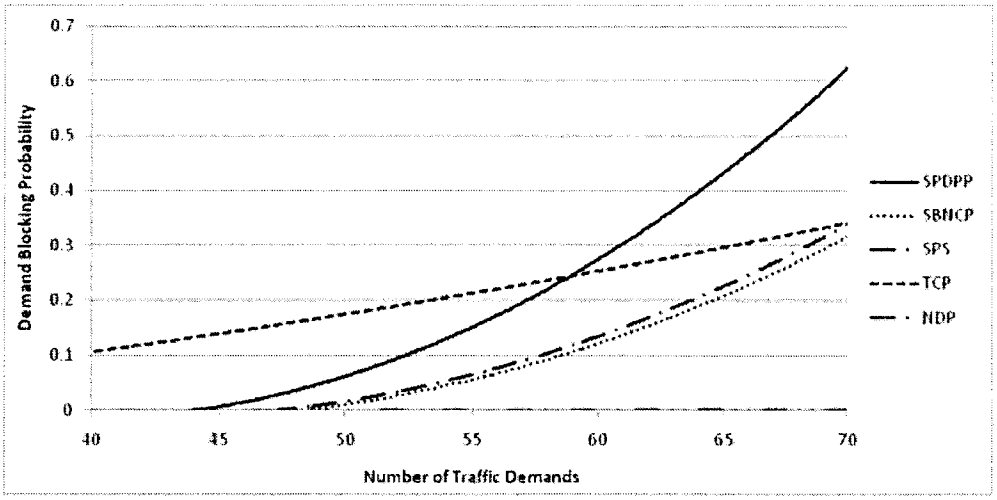
The NCP based algorithms (NDP, TCP, SBNCP) all generate interesting blocking probabilities. The blocking probability for the NCP algorithms in the NSFNET are depicted

in figure 5.13a. Likewise, the blocking probability of the NCP algorithms in the GCN are depicted in figure 5.13b.

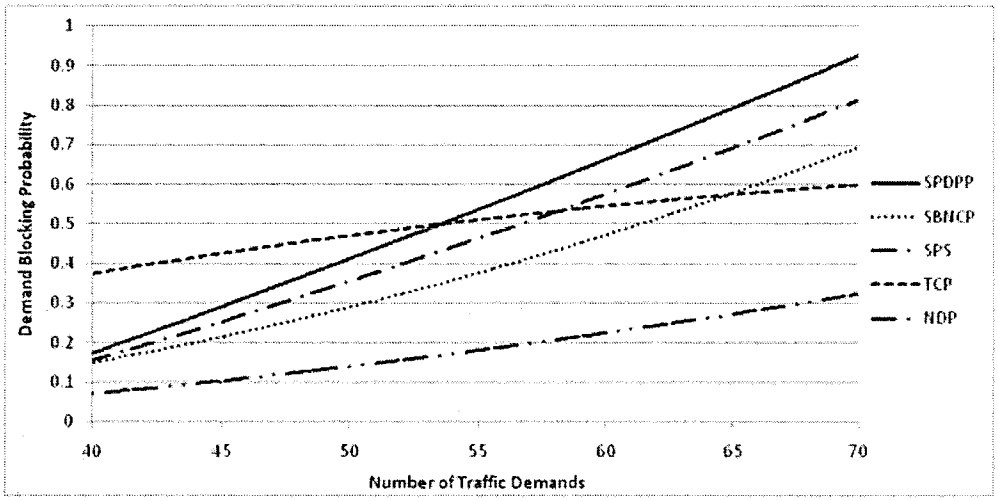
The most visible algorithm of these results is TCP. The TCP algorithm seems to be generating a linearly increasing blocking probability, where every other algorithm follows an exponential growth pattern. This is because TCP tends to fill up the network with trunks before finding efficient sharing opportunities. As a result, TCP is always more selective in which traffic demands are accepted into the network. By doing this, TCP favors intra-zone traffic demands, reducing the overall hop requirement and blocking probability in congested networks. This quality of TCP is visible in both the NSFNET and GCN.

At the other end, NDP generates results that are comparable to the SSMSCP algorithm mentioned in the previous section. This is expected because NDP is very similar in design to SSMSCP. However, since NDP is very constrained in comparison to the other NCP based algorithms, it has a weaker performance. This is most apparent in the GCN, where TCP and SBNCP utilize their sharing advantages to generate desirable blocking probabilities. However, NDP can not provide the same advantages in this situation. Reducing the neighbor constraint so that more than one hop is allowed for the shared section may reduce this problem.

SBNCP produced results that were expected. Since SBNCP still only allows sharing along a designated backup route it can not produce blocking probabilities comparable to FMSCP. Nor can SBNCP compare with the unconstrained SPS for blocking probability. As such SBNCP seems to have generated fair results for NCP. However, it is envisioned that adjusting SBNCP so that few secondary connections are required may free up capacity for improved efficiency and blocking probabilities.



(a) National Science Foundation Network



(b) Global Crossing Network

Figure 5.13.: Network Coded Protection Demand Blocking Probability

5.5.5. Restoration Hops

The restoration hops metric truly portrays the advantages of NCP based algorithms (NDP,TCP,SBNCP). With only a few restoration hops, a network can recover from failures within the time re-

quired to provide high level SLAs. Of all the algorithms, the following do not have any restoration hops:

- Shortest Pair Dedicated Path Protection (benchmark algorithm)
- Near Optimal Source Coded Protection
- Fast Source Coded Protection
- Re-optimized Multiple Source Coded Protection
- Fast Multiple Source Coded Protection
- Single Stream Multiple Source Coded Protection

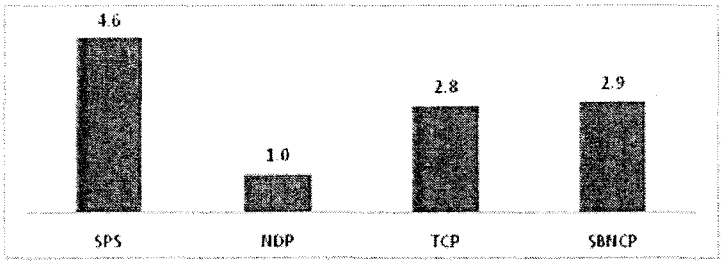
This is because each of them has redundant information pro-actively provided to the destination. The following algorithms have reactive components and therefore require restoration hops:

- Simple Pool Sharing (benchmark algorithm)
- Neighbor Decoded Protection
- Trunk Coded Protection
- Stream Based Network Coded Protection

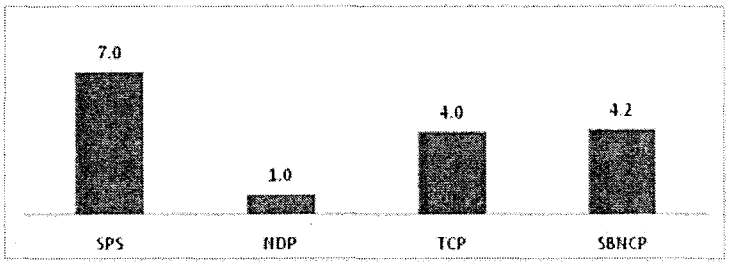
These four algorithms have their average restoration hop requirements depicted in figures 5.14a and 5.14b respectively. For this metric, SPS is the only benchmark algorithm. Since the number of restoration hops required by a traffic demand is uncorrelated with the number of demands, the results will be compared as a bar chart for each network. SPS, having a reactive backup path, obviously requires the greatest number of hops. This corresponds to a requirement of 4.5 hops in the NSFNET and 7 hops in the GCN. As designed, NDP generates at most a one hop restoration route. Since one hop restoration routes are generated for most of the traffic demands, the average restoration hop requirement for NDP is

approximately one. This quality of NDP is apparent in both the NSFNET and GCN.

The final two algorithms produce results in between SPS and NDP. Both create this characteristic between SPS and NDP using different techniques. TCP has pre-defined trunks that force traffic demands to provision coded routes up to the zone of the destination. From there the number of hops to the destination is limited. The zones for TCP were designed so that there would only be a few hops between each node. Unfortunately, these routes can be blocked because of disjointness requirements. This has pushed up the restoration hop requirement for TCP unintentionally high levels. On the other hand, SBNCP used demand generated formations created with the SBS concept. Since traffic demands are not forced into trunks, they can take more direct paths between source and destination. From that, since at least some portion of the route must be devoted to the coded section, the number of restoration hops must be smaller than it would be in a reactive scheme like SPS.



(a) National Science Foundation Network



(b) Global Crossing Network

Figure 5.14.: Restoration Hops

6. Conclusions and Future Research

6.1. Contributions

The contributions of this thesis are in three novel protection schemes for logical networks. These schemes are:

- *Source Coded Protection* (SCP)
- *Multiple Source Coded Protection* (MSCP)
- *Network Coded Protection* (NCP)

SCP offered coded based protection for one traffic demand by fragmenting and encoding data over disjoint routes. From this, two algorithms were generated based on the trade off between speed and efficiency. These algorithms are:

- *Near Optimal Source Coded Protection* (NOSCP)
- *Fast Source Coded Protection* (FSCP)

NOSCP provided an efficient set of routes using several iterations of a path search algorithm. On the other hand, FSCP significantly reduced the number of path search operations with reduced efficiency in congested networks.

MSCP removed the single traffic demand constraint and fragmentation requirements apparent in SCP. With MSCP traffic demands are linearly combined in the network to reduce redundancy. From this scheme, three algorithms were presented based off on efficiency, speed, and complexity. These algorithms are:

- *Fast Multiple Source Coded Protection* (FMSCP)
- *Single Stream Multiple Source Coded Protection* (SSMSCP)
- *Re-optimized Multiple Source Coded Protection* (RMSCP)

FMSCP provided a quick method for generating MSCP. However, it required that destination nodes perform computationally complex decoding. To mitigate this problem a variant was created called SSMSCP, which removed the complexity issue at the cost of efficiency. At the other end, RMSCP was designed to generate an efficient form of MSCP by re-optimizing previously established traffic demands with new traffic requests.

The last scheme proposed was NCP. All previous schemes required that traffic demands share the same destination node. NCP removed this constraint. By removing this constraint, a shared restoration route was required from the decoding point to the destination node. Routing this shared route with the coded route was the major challenge for NCP. Three proposed solutions to this problem were created as three algorithms. The algorithms were:

- *Neighbor Decoded Protection* (NDP)
- *Trunk Coded Protection* (TCP)
- *Stream Based Network Coded Protection* (SBNCP)

NDP simplified the shared route to one hop so that neighbors of the decoding point can share protection resources. At the same time, TCP defined the coded section in advance using pre-defined trunks. This removed most of the coded section calculations, so that the working and shared routes could be optimized together. Meanwhile, SBNCP used the *Stream Based Sharing* (SBS) technique to conceptualize the coded and shared routes into

phantom links. This allowed the coded and shared routes to be generated together.

All of these algorithms were compared with two benchmark algorithms to determine there effectiveness. The two benchmark algorithms were:

- *Shortest Pair Dedicated Path Protection* (SPDPP)
- *Simple Pool Sharing* (SPS)

6.2. Summary of Results

To summarize the results, each of the proposed schemes and algorithms have advantages for different applications. In networks where traffic demands can not be linearly combined, FSCP and NOSCP can be used to provide survivability. If provisioning speed is an issue then FSCP is the ideal choice. Otherwise, if efficiency is more important, NOSCP is a good choice. To mitigate the demand blocking ratio problem of SCP based algorithms, SPDPP can be substituted when three disjoint routes do not exist.

However, if some nodes in the network can be made to linearly combine traffic demands then TCP may be an option. This technique also performs well in large networks where trunks can be used to reduce restoration times. The algorithm provides efficient results and provides network designers with the option of choosing which nodes should be capable of linearly combining information.

On the other hand, if a network can linearly combine traffic demands at any point in the network, two approaches can be taken:

- Low cost and simple decoding.

- Efficient and complex decoding.

The low cost simple decoding technique is used in SSMSCP, NDP, and SBNCP. The choice of among these three algorithms is based on the restoration time requirements. If a minimum restoration time is desired, then SSMSCP can be used. Otherwise, if a one hop restoration is acceptable, NDP can be chosen. On the other end, if restoration time is not a great issue, SBNCP can be selected.

However, if complex decoding can be performed by the destination node, the FMSCP and RMSCP algorithms are desirable. Their good availabilities, low hop requirements, and proactive nature make them ideal survivability techniques for traffic demands.

6.3. Future Work

Due to the emphasis on introducing different coded protection algorithms, this thesis did not have the opportunity to look into the depth on constraints that can be added. Among these potential constraints for future work are:

- SRLG constraints
- Wavelength continuity constraints
- Partial coding capable networks

In addition to these potential constraints, variants of the proposed algorithms were omitted. This is especially true for the algorithms based on NCP. The NCP scheme was proposed with possible variants. However, from these six possibilities only three algorithms were realized. More over, these three algorithms used drastically different techniques for conceptualizing the coded and shared sections. Thus, many different algorithms for NCP not

presented in this thesis are possible.

Above that, some algorithms could be modified slightly to produce superior results. For example, SBNCP could have the cost of phantom networks modified so that traffic demands favor protection sets that have a critical node equal to their destination node. This would reduce the requirement for costly secondary connections. Modifying SBNCP in this respect may let NCP inherit some of the attributes of FMSCP when the critical node and destination node are the same.

Adding to that, RMSCP could be designed such that capacity is released before optimization occurs. This would improve the performance of RMSCP significantly, but cause blips in service as traffic demands switch to their new routes.

Of particular importance for future work would be on an optimized version of TCP. For this thesis, the zones and trunks of TCP were generated manually. This sub-optimal assignment reduced the acceptability of TCP. Optimizing the zones and trunks for TCP would significantly improve its performance. Furthermore, in this thesis each zone was provided with one trunk. If trunks were designed between zone pairs, they may become more practical and efficient.

Lastly, the NDP algorithm was designed to generate at most one hop restorations. It is our belief that two hop restorations could still be easily calculated with acceptable restoration times.

Bibliography

[1] L. Wrobel, *Disaster Recovery Planning For Telecommunications.* Artech House, 1990.

[2] W. D. Grover, *Mesh Based Survivable Network - Options and Strategies for Optical MPLS, SONET and ATM Networking.* Prentice Hall, 2004.

[3] K. N. S. Rijiv Ramaswami, *Optical Networks - A Practical Prespective.* Morgan Kaufmann, 2002.

[4] K. B. Thomas E. Stern, Georgios Ellinos, *Multiwavelength Optical Networks.* Cambridge, 2 ed., 2009.

[5] J.-F. L. R. R. Eric Bouillet, Georgios Ellinas, *Path Routing in Mesh Optical Networks.* Wiley, 2007.

[6] R. C. L. H. F. L. T. L. N. R. M. R. Y Ellison, D. A. Fisher, "Survivable network systems: An emerging discipline," tech. rep., Carnegie Mellon, May 1999.

[7] T.-H. Wu, "Emerging technologies for fiber network survivability," *IEEE Communications Magazine*, vol. 1, pp. 58–74, February 1995.

[8] L. V. Andrea Fumagalli, "Ip restoration vs. wdm protection: Is there an optimal choice," *IEEE Network*, vol. 1, pp. 34–41, 2000.

[9] B. M. Laxman Sahasrabuddhe, S. Ramamurthy, "Fault managment in ip-over-wdm networks: Wdm protection versus ip restoration," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 1, pp. 21–33, 2002.

[10] B. S. Ramamurthy, "Survivable wdm mesh networks, part 1 - protection," in *IEEE INFOCOM*, vol. 2, pp. 744–751, March 1999.

[11] B. S. Ramamurthy, "Survivable wdm mesh networks, part 2 - restoration," in *IEEE INFOCOM*, vol. 2, March 1999.

[12] R. R. J.-F. L. S. C. K. B. Georgios Ellinas, Eric Bouillet, "routing and restoration architecture in mesh optical networks," *Optical Networks Magazine*, vol. 1, pp. 91–106, 2003.

[13] R. Bhandari, *Survivable Networks : Algorithms for Diverse Routing.* Kluwer Academic Publishers, 2 ed., 1999.

[14] R. T. J. Suurballe, "A quick method for finding shortest pairs of disjoint paths," *Networks*, vol. 14, pp. 325–336, 1984.

[15] M. Gong, "Integrated mechanisms for qos and restoration in mesh transport networks," Master's thesis, Lakehead University, 2007.

[16] Z. Z. Raymond W. Yeung, "Distributed source coding for satellite communications," *IEEE Transactions on Information Theory*, vol. 45, pp. 1111–1120, May 1999.

[17] N. C. Z. Z. Raymond W. Yeung, Shuo-Yen Robert Li, *Network Coding Theory.* now, 2006.

[18] E. S. Christina Fragouli, "Network coding fundamentals," *Foundations and Trends in Networking*, vol. 2, no. 1, pp. 1–133, 2007.

[19] E. S. Christina Fragouli, "Network coding applications," *Foundations and Trends in Networking*, vol. 2, no. 2, pp. 135–269, 2007.

[20] D. S. L. Tracy Ho, *Network Coding: an Introduction.* Cambridge University Press, 2008.

[21] A. E. Kamal, "1+n protection in optical mesh networks using network coding over p-cycles," in *IEEE Globecom*, 2006.

[22] A. E. Kamal, "1+n protection against multiple faults in mesh networks," in *IEEE International Conference on Communications (ICC)*, 2007.

[23] A. E. Kamal, "A generalized strategy for 1+n protection," in *ICC 2008 proceedings*, 2008.

[24] A. E. K. Salah A. Aly, "Network protection codes against link failures using network coding," in *IEEE Clobecom*, 2008.

[25] A. E. K. Salah A. Aly, "Network coding-based protection strategy against node failures," in *IEEE INFOCOM*, Department of Electrical and Computer Engineering, Iowa State University, 2009.

[26] D. Crawford, "Fiber optic dig-ups, causes and cures," in *Network Reliability: A Report to the Nation - Compendium of Technical Papers*, National Engineering Consortium, 1993.

[27] H. H.Naser, "Modelling and simulation of mesh networks with path protection and restoration," *Ninth Symposium on Computers and Communication (ISCC 2004)*, vol. 2, pp. 779–785, July 2004.

[28] Z. Gao, "End-to-end shared restoration in multi-domain networks," Master's thesis, Lakehead University, 2008.

[29] H. Naser and H. T. Mouftah, "Enhanced pool sharing: a constraint-based routing algorithm for shared mesh restoration networks," *Journal of Optical Networking*, vol. 3, pp. 303–323, May 2004.

[30] Sosnosky, "Service applications for sonet dcs distributed restoration," *IEEE Journal on Selected Areas in Communications*, vol. 12, pp. 59–68, January 1994.

A. Proposed Survivability Techniques in Algorithmic Form

A.1. Near Optimal Source Coded Protection

Algorithm A.1 Near Optimal Source Coded Protection

Definitions:

- $G(N, J)$: a network G with a set of nodes N, set of links J.
- d_{max}: the minimum nodal degree of both the source and destination nodes. Calculated as $min(|\Gamma_S|, |\Gamma_D|)$.
- d: the current number of disjoint routes being generated by the algorithm.
- b: bandwidth requested by connection requests.
- $C(ij)$: cost of using link $\{ij\}$ for the path of demand r.
- $R_n(r)$: the set of all new routes $R_n^d(r)$ generated for demand r.
- $R_o(r)$: the set of all old routes $R_o^d(r)$ generated for demand r in the previous iteration.
- $C_n(r)$: the total cost of the new set of links for demand r. Calculated as $\sum_{d \in R_n(r)} \sum_{\{ij\} \in R_n^d(r)} C(ij)$
- $C_o(r)$: the total cost of the previous set of links for demand r. Calculated as $\sum_{d \in R_o(r)} \sum_{\{ij\} \in R_o^d(r)} C(ij)$

Algorithm:

1. Let $d_{max} = min(|\Gamma_S|, |\Gamma_D|)$, $d = 2$, $C_o(r) = \infty$
2. If $d \geq d_{max}$
   go to step 6
3. $d++$, $\forall \{ij\} \in J$ let $C(ij) = \begin{cases} \infty & \frac{b}{d-1} > A_{ij} \\ \epsilon & \frac{b}{d-1} \leq A_{ij} \end{cases}$
4. Run d^{th} Shortest Paths Set Algorithm
   $R_n(r)$: the d routes generated by the Shortest Paths Set Algorithm
5. If $\left\lceil \frac{C_n(r)}{d-1} \right\rceil \leq \left\lceil \frac{C_o(r)}{d-2} \right\rceil$
   $R_o(r) = R_n(r)$, $C_o(r) = C_n(r)$, go to step 2
6. If $C_o(r) < \infty$
   Provision demand r with $R_o(r)$
7. END

A.2. Fast Source Coded Protection

Algorithm A.2 Fast Source Coded Protection

Definitions:

- $G(N, J)$: a network G with a set of nodes N, set of links J.
- d_{max}: the minimum nodal degree of both the source and destination nodes. Calculated as $min(|\Gamma_S|, |\Gamma_D|)$.
- d: the current number of disjoint routes being generated by the algorithm.
- b: bandwidth requested by connection requests.
- $C(ij)$: cost of using link $\{ij\}$ for the path of demand r.
- $R_n(r)$: the set of all new routes $R_n^d(r)$ generated for demand r.
- $R_o(r)$: the set of all old routes $R_o^d(r)$ generated for demand r in the previous iteration.
- $C_n(r)$: the total cost of the new set of links for demand r. Calculated as $\sum_{d \in R_n(r)} \sum_{\{ij\} \in R_n^d(r)} C(ij)$
- $C_o(r)$: the total cost of the previous set of links for demand r. Calculated as $\sum_{d \in R_o(r)} \sum_{\{ij\} \in R_o^d(r)} C(ij)$

Algorithm:

1. Let $d_{max} = min(|\Gamma_S|, |\Gamma_D|)$, $d = 3$, $C_o(r) = \infty$, if $d > d_{max}$
   go to step 11
2. $\forall \{ij\} \in J$ let $C(ij) = \begin{cases} \infty & \frac{b}{d-1} > A_{ij} \\ \epsilon & \frac{b}{d-1} \leq A_{ij} \end{cases}$
3. Run d^{th} Shortest Paths Set Algorithm
   $R_n(r)$: the d routes generated by the Shortest Paths Set Algorithm
4. If $C_n(r) \geq \infty$ go to step 11
5. If $\left\lceil \frac{C_n(r)}{d-1} \right\rceil \leq \left\lceil \frac{C_o(r)}{d-2} \right\rceil$ go to step 10
6. $R_o(r) = R_n(r)$, $C_o(r) = C_n(r)$, if $d_{max} = d$
   go to step 10
7. $d++$, $\forall \{ij\} \in R_n(r)$
   $C(ji) = -C(ij)$
   $C(ij) = \infty$
8. Run the Modified Dijkstra's Algorithm
   $R_1(r)$: the route generated by the Modified Dijkstra's Algorithm
9. $\forall \{ij\} \in R_1(r) \cup R_n(r)$
   $R_n(r) = R_1(r) \triangle R_n(r)$
   go to step 5
10. Provision demand r with $R_o(r)$
11. END

A.3. Re-optimized Multiple Source Coded Protection

Algorithm A.3 Re-optimizing Multiple Source Coded Protection

Definitions:

- $G(N, J, P)$: a network G with a set of nodes N, set of links J, and set of protection sets P.
- d_{max}: the minimum nodal degree of both the source and destination nodes. Calculated as $d_{max} = |\Gamma_D|$.
- r: a traffic request.
- r_j: traffic demands that already exist in the protection set s.
- b: bandwidth required by connections.
- $C(ij)$: cost of using link $\{ij\}$ for the path of demand r.
- s: a set of traffic demands protected together.
- $R_{nw}(s)$: the new working connections for all the traffic demands in the protection set s.
- $R_{nw}^{j1}(s)$:the new first route of the traffic demand r_j in the protection set s.
- $R_{nw}^{j2}(s)$: the new second route of the traffic demand r_j in the protection set s, generated using the redundant path $R_r(s)$.
- $D(s)$: the common destination of all the traffic demands in the protection set s.
- $D(r)$: the destination for the traffic request r.
- $S(r_j)$: the source node of the traffic demand r_j.
- $P_s(s)$: the phantom source node for the protection set s.
- $P_l(s)$: a directional link that connects the phantom source $P_s(s)$ to a source node $S(r_j)$.
- $R_{rj}(s)$: the redundant connection for the traffic demand r_j in the protection set s.
- $R_w(s)$: the old working connections for all the traffic demands in the protection set s.

Algorithm:

1. For any protection set $s \in P$,

   If $(|s| + 2 \leq d_{max})$ and $(D(s) = D(r))$

   Go to step 4

2. Run Shortest Pair Algorithm from $S(r)$ to $D(r)$

   $R_{nw}(s)$: the two routes generated by the Shortest Paths Set Algorithm

   If $(R_{nw}(s) = \emptyset)$, go to step 9, else go to step 10

3. Let $s = s \cup r$

4. Generate working route $R_{nw}(s)$ for all traffic demands in the protection set

   Let $P_s(s)$= a new phantom source node, $N = N \cup P_s(s)$

   $\forall (r_j \in s)$, generate a phantom link $j = \{P_s(s), S(r)\}$, $P_l(s) = j \cup P_l(s)$

   $J = J \cup P_l(s)$

   $\forall \{ij\} \in J$ let $C_w(ij) = \begin{cases} \infty & b > A_{ij} \\ \epsilon & b \leq A_{ij} \end{cases}$

   Run $|s|^{th}$ Shortest Paths Set Algorithm from $P_s(s)$ to $D(s)$

   $R_{nw}(s)$: the $(|s| + 1)$ routes generated by the Shortest Paths Set Algorithm

   $R_{nw}^{j1}(s)$: the working route generated for the traffic demand r_j

   If $(R_{nw}(s) = \emptyset)$, $s = s - \{r\}$, Run Remove Phantom Graph Transformation, go to step 2

   $R_{nw}(s) = R_{nw}(s) - R_{nw}(s) \bigcap P_l(s)$

5. Run Redundant Path Generator:

   if Fail, $s = s - \{r\}$, Run Remove Phantom Graph Transformation, go to step 2

6. For $|s| - 1$ iterations,

   Run Branch Redundant Path Generator

   If fail, $s = s - \{r\}$, Run Remove Phantom Graph Transformation, go to step 2

7. Run Remove Phantom Graph Transformation

8. Provision new routes

   $\forall j \in R_{nw}(s)$ let $A_j = A_j - B$

   $\forall j \in R_w(s)$ let $A_j = A_j + B$

   $R_w(s) = R_{nw}(s)$

9. Done

Redundant Path Generator:

1. $\forall \{ij\} \in R_{nw}(s)$
   $C(ji) = -C(ij)$
   $C(ij) = \infty$
2. Run the Modified Dijkstra Algorithm from $P_s(s)$ to $D(s)$
   Let $R_{rj}(s)$ = set directed links $\{ij\} \in J$ along the shortest route
   Let r_j= the traffic demand which was provided the redundant path $R_{rj}(s)$
   If $(R_r(s) = \emptyset)$, go to step 2
3. $\forall \{ij\} \in R_w(s) \cup R_r(s)$
   $R_{nw}(s) = R_{nw}(s) \triangle R_r(s)$

Branch Redundant Path Generator:

1. Let $D_b(s) = D_b(s) \cup R_{nw}^{j1}(s) \cup R_{nw}^{j2}(s)$
2. $\forall \{ij\} \in J$ let $C_r(ij) = \begin{cases} \infty & \{ij\} \in R_{nw}(s) \\ \infty & b > A_{ij} \\ \epsilon & b \leq A_{ij} \end{cases}$
3. Run the Node Set Dijkstra Algorithm from $P_s(s)$ to a node in $D_b(s)$
   Let $R_{rj}(s)$ = set directed links $\{ij\} \in J$ along the shortest route
   Let r_j= the traffic demand which was provided the redundant path $R_{rj}(s)$
   If $(R_{rj}(s) = \emptyset)$, Fail
4. Let $R_{nw}(s) = R_{nw}(s) \cup R_{rj}(s)$

Remove Phantom Graph Transformation:

1. $R_{nw}(s) = R_{nw}(s) - R_{nw}(s) \bigcap P_l(s)$
2. $J = J - \{P_l(s)\}$
3. $N = N - \{P_s(s)\}$

A.4. Fast Multiple Source Coded Protection

Algorithm A.4 Fast Multiple Source Coded Protection

Definitions:

- $G(N, J, P)$: a network G with a set of nodes N, set of links J, and set of protection sets P.
- s: a set of connection requests that are protected together.
- $R_l(s)$: The set of links used by a set of connection requests in the protection set s.
- $R_n(s)$: The set of nodes and links used by a set of connection requests in the protection set s.
- $D(s)$: The common destination node for the set of connection requests s.
- $D(r)$: the destination node for connection request r.
- $D_2(r)$: The available destinations for $R_2(r)$.
- A_{ij}: available bandwidth on link $\{ij\}$.
- b: bandwidth requested by connection requests.
- $R_1(r)$: set of links along the first path of demand r.
- $R_2(r)$: set of links along the second path of demand r.
- $R_p(r)$: set of links along the shortest pair of demand r.
- $C(ij)$: cost of using link $\{ij\}$ for a path.

Algorithm:

1. $\forall \{ij\} \in J$ let $C(ij) = \begin{cases} \infty & b > A_{ij} \\ \epsilon & b \leq A_{ij} \end{cases}$
2. Run the Modified Dijkstra Algorithm

   Let $R_1(r)$ = set directed links $\{ij\} \in J$ along the shortest route

   If $(R_1(r) = \emptyset)$, Fail
3. $\forall \{ij\} \in R_1(r)$

   $C(ji) = -C(ij)$

   $C(ij) = \infty$
4. $\forall s \quad$ if $A_1(s) = \begin{cases} 0 & D(r) \neq D(s) \\ 0 & else if\ R_1(r) \in R_l(s) \\ 1 & otherwise \end{cases}$

   $D_2(r) = D_2(r) \cup R_n(s) \cup D(r)$
5. Run the Node Set Dijkstra Algorithm

   Let $R_2(r)$ = set directed links $\{ij\} \in J$ the shortest route

   Let d = the last node incorporated along the shortest route

   If $(R_2(r) = \emptyset)$, Fail
6. $\forall \{ij\} \in R_1(r) \cup R_2(r)$

   $R_p(r) = R_1(r) \triangle R_2(r)$
7. If $(d = D(r))$, create a new protection set s such that $s = s \cup \{r\}$, go to step 9
8. For an arbitrary protection set s,

   If $A_2(s) = \begin{cases} 0 & D(r) \neq D(s) \\ 0 & else if\ R_1(r) \in R_l(s) \\ 0 & else if\ d \notin R_n(s) \\ 1 & otherwise \end{cases}, \quad s = s \cup \{r\}$

   Else, go to step 8
9. Let $R_l(s) = R_l(s) \cup R_p(r)$

A.5. Single Stream Multiple Source Coded Protection

Algorithm A.5 Single Stream Multiple Source Coded Protection

Definitions:

- $G(N, J, P)$: a network G with a set of nodes N, set of links J, and set of protection sets P.
- s: a set of connection requests that are protected together.
- $R_w(s)$: The set of links used as working routes for the set of connection requests.
- $R_b(s)$: The set of links traversed by backup route of the set of connection requests in the protection set s.
- $R_{bn}(s)$: The set nodes traversed by backup route of the set of connection requests in the protection set s.
- $D(s)$: The common destination node for the set of connection requests s.
- $D(r)$: the destination node for connection request r.
- $D_2(r)$: The available destinations for $R_2(r)$. Initially $D_2(r) = \{D(r)\}$.
- A_{ij}: available bandwidth on link $\{ij\}$.
- b: bandwidth requested by connection requests.
- $R_1(r)$: set of links along the first path of demand r.
- $R_2(r)$: set of links along the second path of demand r.
- $R_p(r)$: set of links along the shortest pair of demand r, where $R_p(r) = R_w(r) \cup R_b(r)$.
- $R_w(r)$: set of links along the working route of the shortest pair for demand r.
- $R_b(r)$: set of links along the backup route of the shortest pair for demand r.
- $C(ij)$: cost of using link $\{ij\}$ for a path.

Algorithm:

1. $\forall \{ij\} \in J$ Let $C(ij) = \begin{cases} \infty & b > A_{ij} \\ \epsilon & b \leq A_{ij} \end{cases}$

2. Run the Modified Dijkstra Algorithm

   Let $R_1(r)$ = set directed links $\{ij\} \in J$ along the shortest route

   If $(R_1(r) = \emptyset)$, Fail

3. $\forall \{ij\} \in R_1(r)$

   $C(ji) = -C(ij)$

   $C(ij) = \infty$

4. $\forall s \in P$ if $A_1(s) = \begin{cases} 0 & D(r) \neq D(s) \\ 0 & else\, if\, R_1(r) \in (R_w(s) \cup R_b(s)) \\ 1 & otherwise \end{cases}$

   $D_2(r) = D_2(r) \cup R_{bn}(s)$

5. Run the Node Set Dijkstra Algorithm

   Let $R_2(r)$ = set directed links $\{ij\} \in J$ the shortest route

   Let d = the last node incorporated along the shortest route

   If $(R_2(r) = \emptyset)$, Fail

6. $\forall \{ij\} \in R_1(r) \cup R_2(r)$

   $R_p(r) = R_1(r) \triangle R_2(r)$

7. If $(d = D(r))$, create a new protection set s such that $s = s \cup \{r\}$, go to step 9

8. For an arbitrary protection set s,

   If $A_2(s) = \begin{cases} 0 & D(r) \neq D(s) \\ 0 & else\, if\, R_1(r) \in (R_w(s) \cup R_b(s)) \\ 0 & else\, if\, d \notin R_{bn}(s) \\ 1 & otherwise \end{cases}$, $s = s \cup \{r\}$

   Else, go to step 8

9. Let $R_w(s) = R_w(s) \cup R_w(r)$, $R_b(s) = R_b(s) \cup R_b(r)$

A.6. Neighbor Decoded Protection

Algorithm A.6 Neighbor Decoded Protection

Definitions:

- $G(N, J, P)$: a network G with a set of nodes N, set of links J, and set of protection sets P.
- s: a set of connection requests that are protected together.
- Γ_v: the neighbour nodes of a node v.
- $R_w(s)$: The set of links used as working routes for the set of connection requests.
- $R_c(s)$: The set of links traversed by coded route of the set of connection requests.
- $R_{cn}(s)$: The set of nodes traversed by coded route of the set of connection requests.
- $R_s(s)$: The set of links traversed by shared route of the set of connection requests.
- $D(r)$: the destination node for connection request r.
- $D_2(r)$: The available destinations for $R_2(r)$. Initially $D_2(r) = \{D(r)\}$.
- A_{ij}: available bandwidth on link $\{ij\}$.
- b: bandwidth requested by a connection request.
- $R_1(r)$: set of links along the first path of demand r.
- $R_2(r)$: set of links along the second path of demand r.
- $R_p(r)$: set of links along the shortest pair of demand r. $R_p(r) = R_w(r) \cup R_b(r)$.
- $R_w(r)$: set of links along the working route of the shortest pair for demand r.
- $R_b(r)$: set of links along the backup route of the shortest pair for demand r.
- $C(ij)$: cost of using link $\{ij\}$ for a path.
- $D_l(r)$: the second last node along the longer path of a shortest pair $R_p(r)$.
- $D_s(r)$: the second last node along the longer path of a shortest pair $R_p(r)$.

Algorithm:

1. $\forall \{ij\} \in J$ let $C(ij) = \begin{cases} \infty & b > A_{ij} \\ \epsilon & b \leq A_{ij} \end{cases}$
2. Run the Modified Dijkstra Algorithm

   Let $R_1(r) =$ set directed links $\{ij\} \in J$ along the shortest route

   If $(R_1(r) = \emptyset)$, Fail
3. $\forall \{ij\} \in R_1(r)$

   $C(ji) = -C(ij)$

   $C(ij) = \infty$
4. $\forall s \in P \quad \text{if } A_1(s) = \begin{cases} 0 & D(r) \notin \Gamma_{C(s)} \\ 0 & else\, if\, R_1(r) \in (R_s(s) \cup R_c(s) \cup R_w(s) \cup \{C(s)D(r)\} \cup \{D(r)C(s)\}) \\ 1 & else\, if\, \{C(s)D(r)\} \in R_s(s)\, and\, \{D(r)C(s)\} \in R_c(s) \\ 1 & else\, if\, A_{C(s)D(r)} \geq b\, and\, A_{D(r)C(s)} \geq b \\ 0 & otherwise \end{cases}$

   $D_2(r) = D_2(r) \cup R_{cn}(s)$
5. Run the Node Set Dijkstra Algorithm

   Let $R_2(r) =$ set directed links $\{ij\} \in J$ the shortest route

   Let $d =$ the last node incorporated along the shortest route

   If $(R_2(r) = \emptyset)$, Fail
6. $\forall \{ij\} \in R_1(r) \cup R_2(r)$

   $R_p(r) = R_1(r) \triangle R_2(r)$
7. Run NDP Routed Protection Set Availability Function
8. $s = s \cup \{r\}$, $R_w(s) = R_w(s) \cup R_w(r)$

   END

NDP Routed Protection Set Availability Function:

1. If $d = D(r)$

   Perform Critical Node Assignment Strategy

   END

2. If any $s \in P$, If $A_1(s) = 1$ and $C(s) = D(r)$ and $d \in R_c(s)$

   $R_c(s) = R_c(s) \cup R_b(r)$

   END

3. If any $s \in P$, If $A_1(s) = 1$ and $\{C(s)D(r)\} \in R_s(s)$ and $\{D(r)C(s)\} \in R_c(s)$ and $d \in R_c(s)$

   $R_c(s) = R_c(s) \cup R_b(r)$

   END

4. If any $s \in P$, If $A_1(s) = 1$ and $\{D(r)C(s)\} \in R_c(s)$ and $B \leq A_{C(s)D(r)}$ and $d \in R_c(s)$

   $R_s(s) = R_s(s) \cup \{C(s)D(r)\}$, $R_c(s) = R_c(s) \cup R_b(r)$ and $A_{C(s)D(r)} = A_{C(s)D(r)} - B$

   END

5. If any $s \in P$, If $A_1(s) = 1$ and $B \leq A_{C(s)D(r)}$ and $B \leq A_{D(r)C(s)}$ and $d \in R_c(s)$

   $R_s(s) = R_s(s) \cup \{C(s)D(r)\}$ and $A_{C(s)D(r)} = A_{C(s)D(r)} - B$

   $R_c(s) = R_c(s) \cup R_b(r) \cup \{D(r)C(s)\}$ and $A_{D(r)C(s)} = A_{D(r)C(s)} - B$

   END

Critical Node Assignment Strategy:

1. If $|\Gamma_{D(r)}| \geq |\Gamma_{D_l(r)}|$ and $|\Gamma_{D(r)}| \geq |\Gamma_{D_s(r)}|$
   $C(s) = D(r)$, $R_c(s) = R_c(s) \cup R_b(r)$
   END
2. If $|\Gamma_{D_l(r)}| > |\Gamma_{D(r)}|$ and $A_{D(r)D_l(r)} \geq B$
   $C(s) = D_l(r)$
   $A_{D(r)D_l(r)} = A_{D(r)D_l(r)} - B$,
   $R_s(s) = R_s(s) \cup \{D(r)D_l(r)\}$, $R_c(s) = R_c(s) \cup R_b(r) - \{D(r)D_l(r)\}$
   go to step 5
3. If $|\Gamma_{D_s(r)}| > |\Gamma_{D(r)}|$ and $A_{D(r)D_s(r)} \geq B$
   $C(s) = D_s(r)$
   $A_{D(r)D_s(r)} = A_{D(r)D_s(r)} - B(r)$
   $R_s(s) = R_s(s) \cup \{D(r)D_s(r)\}$, $R_c(s) = R_c(s) \cup R_b(r) - \{D(r)D_s(r)\}$
   go to step 5
4. $C(s) = D(r)$, $R_c(s) = R_c(s) \cup R_b(r)$, END
5. $R_s(s) = R_s(s) \cup \{C(s)D(r)\}$ and $R_c(s) = R_c(s) \cup \{D(r)C(s)\}$
   END

A.7. Trunk Coded Protection

Algorithm A.7 Trunk Coded Protection

Definitions:

- $G(N, J, P, Z)$: a network G with a set of nodes N, set of links J, set of protection sets P, and set of zones Z.
- s: a set of connection requests that are protected together.
- z: a set of nodes that are protected together with an inter-zone trunk.
- $R_w(s)$: The set of links used as working routes for the set of connection requests.
- $R_c(s)$: The set of links and nodes traversed by inter-zone route of the set of connection requests.
- $R_s(s)$: The set of links and nodes traversed by intra-zone route of the set of connection requests.
- $R_t(z)$: The set of links along the trunk for zone z.
- $R_{tn}(z)$: The set of nodes along the trunk for zone z.
- $D(r)$: the destination node for connection request r.
- $D_1(r)$: The available destinations for $R_s(r)$ and $R_d(r)$.
- $D_2(r)$: The available destinations for $R_2(r)$. Initially $D_2(r) = \{D(r)\}$.
- A_{ij}: available bandwidth on link $\{ij\}$.
- b: bandwidth requested by demand r.
- $R_1(r)$: set of links along the working path of demand r.
- $R_2(r)$: set of links along the inter-zone path of demand r.
- $R_3(r)$: set of links along the intra-zone path of demand r.
- $C_1(ij)$: cost of using link $\{ij\}$ for the working path $R_1(r)$.
- $C_2(ij)$: cost of using link $\{ij\}$ for the inter-zone path $R_2(r)$.
- $C_3(ij)$: cost of using link $\{ij\}$ for the intra-zone path $R_3(r)$.
- $I(r)$: an intersection point along the coded route $R_t(z)$.
- $C[I_s(r)]$: cost of a route from the source $S(r)$ to an intersection point $I(r)$.
- $C[I_d(r)]$: cost of a route from the destination $D(r)$ to an intersection point $I(r)$
- B_{ij}: the total shared backup bandwidth required on link $\{ij\}$. Calculated as $B_{ij} = max_{\forall\{kl\}\in J}[\theta_{\{kl\}\{ij\}}]$.
- $T_{ij}(r)$: the maximum amount of bandwidth required on link j if a link on the working path $R_1(r)$ fails. Calculated as $T_{ij}(r, s) = b + max_{\forall\{kl\}\in R_w(s)}[\theta_{\{kl\}\{ij\}}]$.

Algorithm:

1. $\forall \{ij\} \in J$ let $C_1(ij) = \begin{cases} \infty & \{ij\} \in R_t(z) \\ \infty & else\, if\, b > A_{ij} \\ \epsilon & else\, if\, b \leq A_{ij} \end{cases}$

2. Run the Intersection Routing Technique

   Let $R_1(r)$ = set directed links $\{ij\} \in J$ along the shortest route

   If $(R_1(r) = \emptyset)$, Fail

3. $\forall \{ij\} \in J$ let $C_2(ij) = \begin{cases} \infty & \{ij\} \in R_1(r) \\ \infty & else\, if\, b > A_{ij} \\ \epsilon & else\, if\, b \leq A_{ij} \end{cases}$

4. $\forall v \in R_{tn}(z)$ let $D_2(r) = D_2(r) \cup v$

5. Run the Node Set Dijkstra Algorithm

   Let $R_2(r)$ = set directed links $\{ij\} \in J$ the shortest route

   Let d = the last node incorporated along the shortest route

   If $(R_2(r) = \emptyset)$, Fail

6. Run Protection Set Assignment Strategy

7. $\forall \{ij\} \in J$ let $C_3(j) = \begin{cases} \infty & \{ij\} \in R_1(r) \\ \epsilon & else\, if\ \{ij\} \in B_l(s) \\ \epsilon & else\, if\, T_{ij}(r,s) \leq B_{ij} \\ T_{ij}(r,s) - B_{ij} & else\, if\, T_{ij}(r,s) - B_{ij} \leq A_{ij} \\ \infty & otherwise \end{cases}$

8. Run the Modified Dijkstra Algorithm

   let $R_3(r)$ = set directed links $\{ij\} \in J$ along the shortest route

   If $(R_3(r) = \emptyset)$, Fail

9. $\forall \{ij\} \in R_w(s)$ and $\{kl\} \in R_3(r)$ if $j \notin R_s(s)$

   $\theta_{\{ij\}\{kl\}} = \theta_{\{ij\}\{kl\}} + b$

   END

Intersection Routing Technique:

1. $\forall v \in R_{tn}(z)$ let $D_1(r) = D_1(r) \cup v$
2. Run the Intersection Routing Technique Dijkstra's Algorithm from the source node *S(r)* to all of the potential intersections in $D_1(r)$

   Let $R_s(r)$ = set directed links $\{ij\} \in J$ along the spanning tree

   If $(R_3(r) = \emptyset)$, Fail
3. $\forall \{ij\} \in J$ let $C(ij) = C(ji)$ and $C(ji) = C(ij)$
4. Run the Intersection Routing Technique Dijkstra's Algorithm from the source node *D(r)* to all of the potential intersections in $D_1(r)$

   Let $R_d(r)$ = set directed links $\{ij\} \in N$ along the spanning tree

   If $(R_3(r) = \emptyset)$, Fail
5. $\forall \{ij\} \in R_d(r)$ let $\{ij\} = \{ji\}$
6. $R_1(r) = R_s(r) \cup R_d(r)$ such that $C_1(r) = min(C[I_s(r)] + C[I_d(r)])$

Protection Set Assignment Strategy:

1. For any protection set s if $A_1(s) = \begin{cases} 0 & C(s) \neq C(t) \\ 0 & else if\, R_1(r) \in R_w(s) \\ 1 & otherwise \end{cases}$

   Let $s = s \cup \{r\}$, $R_w(s) = R_w(s) \cup R_1(r)$, $R_c(s) = R_c(s) + R_1(r)$
2. Else, if $\forall \{ij\} \in R_t(z)$, $b \leq A_{ij}$

   Create a new protection set s such that $s = s \cup \{r\}$, $R_w(s) = R_w(s) \cup R_1(r)$, $R_c(s) = R_c(s) + R_1(r)$
3. Else, Fail

A.8. Stream Based Network Coded Protection

Algorithm A.8 Stream Based Network Coded Protection

Definitions:

- $G(N, J, P, P_n)$: a network G with a set of nodes N, set of links J, set of protection sets P, and phantom networks P_n.
- s: a set of connection requests that are protected together.
- $P_n(s) = [P_v(s), P_l(s), P_e(s), P_x(s)]$: A phantom network for a protection set s that contains phantom nodes $P_v(s)$, phantom network entrance links $P_e(s)$, phantom network exit links $P_x(s)$, and phantom inter-network links $P_l(s)$.
- $R_w(s)$: The set of links and nodes used as working routes for the set of connection requests in s.
- $R_c(s)$: The set of links and nodes traversed by coded section of the protection set s.
- $R_s(s)$: The set of links and nodes traversed by shared section of the protection set s.
- $D(r)$: the destination node for connection request r.
- $D_2(r)$: The available destinations for $R_2(r)$. Initially $D_2(r) = \{D(r)\}$
- d_2 :The destination chosen for $R_2(r)$.
- A_{ij}: available bandwidth on link $\{ij\}$.
- b: bandwidth requested by connection requests.
- $R_1(r)$: set of links along the working path of demand r.
- $R_2(r)$: set of links along the backup path of demand r.
- $R_c(r)$: set of links along the coded section of demand r.
- $R_s(r)$: set of links along the shared section of demand r.
- $R_3(r)$: set of links along the secondary connection of demand r.
- $C_1(ij)$: cost of using link $\{ij\}$ for the first path $R_1(r)$.
- $C_2(ij)$: cost of using link $\{ij\}$ for the backup path $R_2(r)$.
- $C_3(ij)$: cost of using link $\{ij\}$ for the secondary connection $R_3(r)$.
- $R[ij]$: the real version of a phantom link $\{ij\}$.

Algorithm:

1. $\forall \{ij\} \in J \cup P_n$ let $C_1(ij) = \begin{cases} \infty & \{ij\} \in P_n \\ \infty & else\, if\, b > A_{ij} \\ \epsilon & else\, if\, b \leq A_{ij} \end{cases}$

2. Run the Dijkstra's Algorithm

   Let $R_1(r)$ = set directed links $\{ij\} \in J$ the shortest route

   If $(R_1(r) = \emptyset)$, Fail

3. $\forall s \in S$ let $A_2(s) = \begin{cases} 0 & R_1(r) \in (R_s(s) \cup R_c(s) \cup R_w(s)) \\ 1 & otherwise \end{cases}$

4. $\forall \{ij\} \in J \cup P_n$ let $C_2(ij) = \begin{cases} \epsilon & \{ij\} \in P_n(s)\, and\, A_2(s) = 1 \\ \infty & else\, if\ \{ij\} \in R_1(r) \\ \infty & else\, if\, b > A_{ij} \\ \beta & else\, if\, b \leq A_{ij} \end{cases}$

5. Run the Dijkstra's Algorithm

   Let $R_2(r)$ = set directed links $\{ij\} \in J$ the shortest route

   Let d_2 =the destination chosen be $R_2(r)$

   If $(R_2(r) = \emptyset)$, Fail

6. Run Protection Set Assignment and Conversion Technique

7. If $[d_2 \neq C(s)]$, Run Branch Connection Generator

8. Let $R_c(s) = R_c(s) \cup R_c(r)$ $R_s(s) = R_s(s) \cup R_s(r)$ $R_w(s) = R_w(s) \cup R_1(r)$

Protection Set Assignment and Conversion Technique:

1. $\forall s \in S$ if $R_2(r) \cap P_n(s) = \emptyset$

   Go to step 3

2. $\forall \{ij\} \in R_2(r)$

   If $\{ij\} \in P_i(s)$

   $R_2(r) = R_2(r) \cup R[ij]$

   If $\{ij\} \in [P_e(s) \cup P_x(s) \cup P_l(s)]$

   $R_2(r) = R_2(r) - \{ij\}$

   Go to step 4

3. Let s= a new protection set in S, where $C(s) = max_{v \in R_2(r)} |v|$

4. Let $R_c(r)$= all links $\{ij\} \in R_2(r)$ between *S(r)* and *C(s)*, $R_s(r)$= all links $\{ij\} \in R_2(r)$ between *C(s)* and *D(r)*

Branch Connection Generator:

1. $\forall \{ij\} \in J \cup P_n$ let $C_3(ij) = \begin{cases} \infty & \{ij\} \in P_n \\ \epsilon & else\, if\ \{ij\} \in R_1(r) \\ \infty & else\, if\, b > A_{ij} \\ \beta & else\, if\, b \leq A_{ij} \end{cases}$

2. $\forall v \in [R_c(s) \cup R_c(r) - S(r)]$ let $D_3(r) = D_3(r) \cup v$

3. Run the Node Set Dijkstra Algorithm

   Let $R_3(r)$ = set directed links $\{ij\} \in J$ the shortest route

   Let d = the last node incorporated along the shortest route

   If $(R_3(r) = \emptyset)$, Fail

4. Let $R_3(r) = R_3(r) - R_3(r) \cap R_1(r)$

B. Benchmark Algorithms

For path survivability schemes, there is always a trade off between speed and efficiency. Quick and inefficient protection is always pitted against slow and efficient restoration. From these schemes many heuristic algorithms have been designed. In order to properly compare coded based survivability with the traditional protection and restoration schemes, at least two benchmark algorithms must be selected. It is important that these two algorithms represent the best their parent schemes can offer. From that, a protection heuristic algorithm should be chosen which minimizes the capacity usage required for protection without unnecessarily sacrificing its simplicity and speed. Likewise, a heuristic algorithm should be selected which attempts to reduce restoration time without adversely affecting capacity usage. By selecting algorithms in this fashion, they can act as acceptability boundaries for their coded counterparts. For most of the different performance qualities captured from the results, these two algorithms should sit on opposing extremes. From that, it will be unacceptable for a coded heuristic algorithm to generate results that are not between these two bounds. This will allow a proper comparison of the qualities of each of the coded survivability schemes and their heuristic algorithms. The two benchmark algorithms selected for this purpose are *Shortest Pair Dedicated Path Protection* (SPDPP) and *Simple Pool Sharing* (SPS). The following two sections will briefly explain the operation of each of the heuristic algorithms.

B.1. Shortest Pair Dedicated Path Protection

Shortest Pair Dedicated Path Protection (SPDPP) is a standard path based protection scheme that can be used in any network. It generates dedicated backup path protection, a completely proactive survivability scheme. As stated in table B.1, the failure route, channel assignments and cross-connects are assigned and configured before the fault occurs. This allows the backup connection to proactively send data to the destination in advance. If there is a fault occurrence along one of the links on the primary path, the destination node will immediately switch the connection to the backup route. Thus, it does not matter where the failure occurs as the redundant capacity is dedicated and proactively used for survivability. Since this algorithm generates dedicated backup paths, it can be used to create platinum SLA connections. SPDPP does not violate any other constraints required to use a one step solution. Therefore, it can be given shortest path set characteristics.

Table B.1.: Shortest Pair Dedicated Path Protection

Failure recovery route		Channel assignment on failure recovery route		Cross-connect on failure	Failure specific
Computed	Assigned	Computed	Assigned	recovery route	
before	before	before	before	before	no

Shortest path set characteristics is a term to describe an algorithm which can resolve shortest path sets instead of taking a simple two step approach. Since, this technique is reused in several of the algorithms presented in this thesis, a brief explanation is being provided. By having this functionality, the algorithm reduces its capacity usage and becomes immune to the trap topology problem. Shortest path set characteristics can only be added to a heuristic algorithm if the backup route link costs do not depend on the working route and no link in the network has a lower cost for the backup route than it had for the working route. Reference [13] provides an excellent proof and a few examples of these constraints

required for shortest path set characteristics.

If an algorithm does not require that the working path be determined before the backup path and doesn't change its link costs, it can to generate a shortest pair. Adding shortest path set characteristics to an algorithm takes at least two route modifications. These modifications are illustrated in figure B.1. In the figure, after the first route is generated the links along that route are transformed. They are transformed so that the cost of directional link *C(ij)* connecting node j to node i in the first route $R_1(r)$ is set to infinite and the cost of its opposite direction pair *C(ji)* is set to $-C(ij)$. With these modifications the second route $R_2(r)$ can be routed partially over the negative cost links to reduce its path cost. Afterward, in order to generate the shortest pair $R_p(r)$, any bidirectional links that exist in both $R_1(r)$ and $R_2(r)$ are removed. If shortest triplets or quadruples are desired, then at the end of the algorithm, the shortest path $R_1(r)$ can be set to the shortest pair $R_p(r)$ and modified again for a third route. Therefore, the number of paths in the shortest paths set is only limited by the min-cut between the source and destination. For a detailed explanation of this concept please refer to [13].

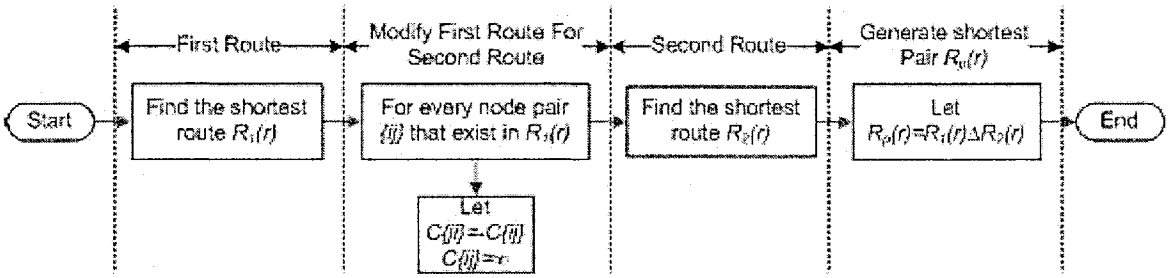


Figure B.1.: Shortest Path Set Modifications

Using shortest path set modifications, SPDPP generates two paths for a connection request r. From an algorithmic point of view, SPDPP can be setup with two iterations of a path search algorithm. For a given network $G = (N, J)$, where N is a set of nodes and J is a

set of links, a shortest first path $R_1(r)$ can be determined by running any shortest path algorithm. For path computation purposes, the cost of using a link $C_1(ij)$ will be set according to equation B.1.

$$C_1(ij) = \begin{cases} \infty & b(r) > A_{ij} \\ \epsilon & b(r) \leq A_{ij} \end{cases} \tag{B.1}$$

In the equation, the cost of using a link $C_1(ij)$ for $R_1(r)$ of demand r will be set to ∞ if the bandwidth of demand r is greater than the available capacity A_{ij} on link $\{ij\}$. Otherwise, the link cost is set to some small number ϵ. The second path $R_2(r)$ can be determined afterward with link costs $C_2(ij)$ set by the shortest path set modifications. Thus the link costs $C_2(ij)$ for the second route $R_2(r)$ can be determined with equation B.2 where link $\{ji\}$ is the opposing directional link pair of link $\{ij\}$.

$$C_2(ij) = \begin{cases} \infty & \{ij\} \in R_1(r) \\ -C_1(ji) & \{ji\} \in R_1(r) \\ C_1(ij) & otherwise \end{cases} \tag{B.2}$$

Afterward, in order to generate the shortest pair $R_p(r)$, an exclusive disjunction of routes $R_1(r)$ and $R_2(r)$ is performed. This means that any bidirectional links that exist in both $R_1(r)$ and $R_2(r)$ are removed. To further explain this concept, a short example has been included.

$$\{A, B, C, D\} \triangle \{C, D, E, F\} = \{A, B, E, F\}$$

In the example, The exclusive disjunction is performed on two sets of letters. Any letters contained in both sets are removed from the result. Using this concept, bidirectional links contained within both paths are removed. For an extensive treatise on this topic, please refer to [13]. Algorithm B.1 has been employed to further explain the operation of SPDPP.

Algorithm B.1 Shortest Pair Dedicated Path Protection

Definitions:

- $G(N, J)$: a network G with a set of nodes N and set of links J.
- $C_1(ij)$: cost of using link $\{ij\}$ for the working path.
- $C_2(ij)$: cost of using link $\{ij\}$ for the backup path.
- $\{ji\}$: the directional link opposite link $\{ij\}$ in a bidirectional link.
- A_{ij}: available bandwidth on link $\{ij\}$.
- $b(r)$: bandwidth requested by demand r.
- $R_1(r)$: set of links along the first path of demand r.
- $R_2(r)$: set of links along the second path of demand r.
- $R_p(r)$: set of links along the shortest pair of demand r.

Algorithm:

1. $\forall \{ij\} \in J$ let $C_1(ij) = \begin{cases} \infty & b(r) > A_{ij} \\ \epsilon & b(r) \leq A_{ij} \end{cases}$
2. Run shortest path algorithm

   $R_1(r)$ = the set of links traversed by the shortest path
3. $\forall \{ij\} \in J$ let $C_2(ij) = \begin{cases} \infty & \{ij\} \in R_1(r) \\ -C_1(ji) & \{ji\} \in R_1(r) \\ C_1(ij) & otherwise \end{cases}$
4. Run shortest path algorithm

   $R_2(r)$ = the set of links traversed by the shortest path
5. $R_p(r) = R_1(r) \cup R_2(r)$

   $\forall \{ij\} \in R_1(r)$ if $(\{ji\} \in R_2(r))$

   $R_P(r) = R_P(r) - \{ij\}$

B.2. Simple Pool Sharing

Simple Pool Sharing *(SPS)* is a SBPP with non pre-assigned channels scheme presented in [27]. It is a semi proactive scheme designed for logical layer networks. To this end, it puts special emphasis on determining an efficient shared backup route. In order to allow more efficient routing assignments, channels are not determined until after the fault occurs. Thus the set of cross-connects required to route the connection can only be determined after fault isolation occurs. Since SPS was designed to operate at the logical layer, channel assignments were considered of limited importance. These characteristics of SPS are summarized in table B.2. Because failure recovery can be guaranteed in single link failure situations, SPS can be relied upon to provide gold SLAs. Additionally, due to the innovative strategy this algorithm employs, it produces close to optimal redundancy levels

Table B.2.: Pool Sharing

Failure recovery route		Channel assignment on failure recovery route		Cross-connect on failure recovery route	Failure specific
Computed	Assigned	Computed	Assigned		
before	before	after	after	after	yes

To provision demand r in a network $G(N, J)$ protected by SPS, a two step approach is required. The procedure for setting up the working path $R_1(r)$ for the demand is equivalent to that of SPDPP. The cost of a link $C_1(j)$ is determined by the ability of the link to support the additional bandwidth $b(r)$ required demand r. Therefore the cost of a link $C_1(j)$ can

be determined by equation B.3.

$$C_1(j) = \begin{cases} \infty & b(r) > A_j \\ \epsilon & b(r) \leq A_j \end{cases} \tag{B.3}$$

Once the working path has been setup, a spare capacity matrix is used to determine the link cost of the backup route $C_2(j)$. The spare capacity matrix is seen in equation B.4 with elements θ_{ij}. In the matrix, each element θ_{ij} is the spare capacity required on link j if link i fails. To allow sharing across all links in the network a $(J \times J)$ square matrix is required, where J is the number of links in the network. Fortunately, a link only requires a column of the matrix in order to get enough information for sharing computations. This allows for SPS to be used as a distributed algorithm when creating network survivability.

$$\Phi = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} & \cdots & \theta_{1J} \\ \theta_{21} & \theta_{22} & \theta_{23} & \cdots & \theta_{2J} \\ \theta_{31} & \theta_{32} & \theta_{33} & \cdots & \theta_{3J} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \theta_{J1} & \theta_{J2} & \theta_{J3} & \cdots & \theta_{NJ} \end{bmatrix} \tag{B.4}$$

To determine amount of capacity that must be reserved on a link j, the maximum of all requirements for spare capacity from all the other links can be utilized. By using the spare capacity matrix, this can be determined by calculating the maximum of the column corresponding to link j. Mathematically, this required backup capacity B_j can be calculated as $max_{\forall i \in J}[\theta_{ij}]$. When computing the backup route for a demand r, a method is required to determine how much additional bandwidth must be reserved on link j if the working path $R_1(r)$ of demand r fails. This can be determined by calculating the maximum of the spare capacity requirement of every link on the working path for link j added to the bandwidth

requirement $b(r)$ of demand r. For simple pool sharing this is calculated as

$$T_j(r) = b_r + max_{\forall i \in R_1(r)}[\theta_{ij}]$$

If the bandwidth required to protect the demand $T_j(r)$ is less than the amount of bandwidth B_j that link j has already reserved for protection, then the backup path $R_2(r)$ can be routed on that link without using any additional capacity. Under these conditions the cost of routing over the link will be set to some small number ϵ. If $T_j(r) > B_j$, then for demand r to utilize link j for it's backup connection $T_j(r) - B_j$ additional bandwidth must be reserved. As long as the link has enough available capacity A_j to reserve the additional bandwidth, the link can be used. In this case the cost of using the link is set to the amount of additional capacity $T_j(r) - B_j$ required to route demand r over link j. If there is not enough available capacity on link j then the link cost $C_2(j)$ for $R_2(r)$ must be set to ∞. From that, the cost of a link $C_2(j)$ for the backup route $R_2(r)$ can be determined by equation B.5.

$$C_2(j) = \begin{cases} \infty & j \in R_1(r) \\ \epsilon & else\, if\, T_j(r) \leq B_j \\ T_j(r) - B_j & else\, if\, T_j(r) - B_j \leq A_j \\ \infty & otherwise \end{cases} \tag{B.5}$$

If the connection request r can be given a working route $R_1(r)$ and a backup route $R_2(r)$ then the demand will be accepted. After accepting the demand, the spare capacity matrix will update all matrix elements with the new redundancy requirements. these updates can be calculated as $\theta_{ij} = \theta_{ij} + b(r)$ for all $i \in R_1(r)$ and $j \in R_2(r)$. For further reference, SPS is further explained as algorithm B.2.

Algorithm B.2 Simple Pool Sharing

Definitions:

- $G(N, J)$: a network G with a set of nodes N and set of links J.
- $C_1(j)$: cost of using link j for the working path.
- $C_2(j)$: cost of using link j for the backup path.
- A_j: available bandwidth on link j.
- $b(r)$: bandwidth requested by demand r.
- $R_1(r)$: set of links along the working path of demand r.
- $R_2(r)$: set of links along the backup path of demand r.
- Φ: the backup bandwidth square matrix where θ_{ij} is the amount of backup bandwidth required on link j for link i.

$$\Phi = \begin{matrix} \theta_{11} & \theta_{12} & \theta_{13} & \cdots & \theta_{1J} \\ \theta_{21} & \theta_{22} & \theta_{23} & \cdots & \theta_{2J} \\ \theta_{31} & \theta_{32} & \theta_{33} & \cdots & \theta_{3J} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \theta_{J1} & \theta_{J2} & \theta_{J3} & \cdots & \theta_{JJ} \end{matrix}$$

- B_j: the total shared backup bandwidth required on link j. Calculated as $B_j = max_{\forall i \in J}[\theta_{ij}]$
- $T_j(r)$: the maximum amount of bandwidth required on link j if a link on the working path $R_1(r)$ fails. Calculated as $T_j(r) = b_r + max_{\forall i \in R_1(r)}[\theta_{ij}]$

Algorithm:

1. $\forall j \in J$ let $C_1(j) = \begin{cases} \infty & b(r) > A_j \\ \epsilon & b(r) \leq A_j \end{cases}$
2. Run shortest path algorithm
   - $R_1(r)$ = the set of links traversed by the shortest path
3. $\forall j \in J$ let $C_2(j) = \begin{cases} \infty & j \in R_1(r) \\ \epsilon & else\, if\, T_j(r) \leq B_j \\ T_j(r) - B_j & else\, if\, T_j(r) - B_j \leq A_j \\ \infty & otherwise \end{cases}$
4. Run shortest path algorithm
   - $R_2(r)$ = the set of links traversed by the shortest path
5. For all links $i \in R_1(r)$ and $j \in R_2(r)$ 184
   - $\theta_{ij} = \theta_{ij} + b(r)$

C. Min-Cut Max-Flow Theorem

The min-cut max-flow theorem is an important concept in both network survivability and network coding. For example, let a network be represented by a graph abstraction $G = (N, J)$, where N is a set a nodes and J is a set of links. In this network, we have a source node $S \in N$ that is sending information to a destination node $D \in N$. A cut is a set of edge removals that completely separates a node S from node D. The min-cut is the minimum size cut set that will completely separate the two nodes. For example, in figure C.1 removing the edges covered by the min-cut will completely separate the source and destination. Every other cut set will be larger than the min-cut depicted in the figure. If the min-cut associated with the source destination pair is h, then it can be said that the max-flow is proportional to h, where the max-flow is the highest possible rate of information transfer. That is, given that the min-cut is h, we can find at most h disjoint paths between the two nodes. This concept is used extensively in this thesis to determine when sharing can occur.

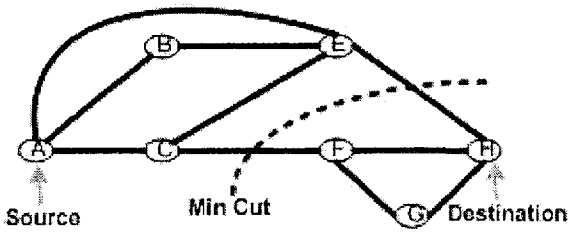


Figure C.1.: Min-Cut

C.1. Pseudo Min-Cut Determinator

The Pseudo min-cut determinator is a simplified method for approximating the min-cut between two nodes in a network. In this technique we determine the pseudo min-cut d_{max} with equation C.1.

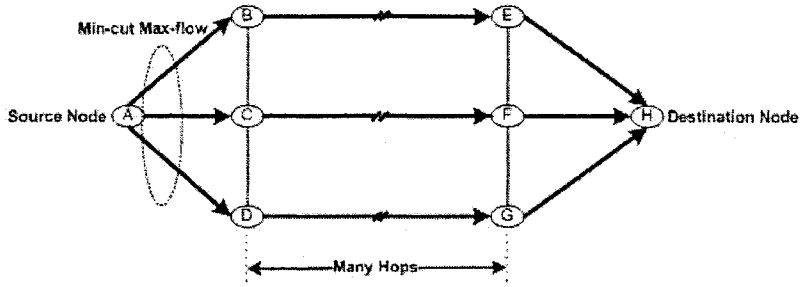
$$d_{max} = min\left(\left|\Gamma_S\right|, \left|\Gamma_D\right|\right) \tag{C.1}$$

From the equation the pseudo min-cut d_{max} is calculated as the minimum nodal degree of the source $|\Gamma_S|$ and destination nodes $|\Gamma_D|$. This technique is computationally insignificant compared with determining a new $N+1$ set of paths. Using this pseudo min-cut approach generates two possible scenarios.

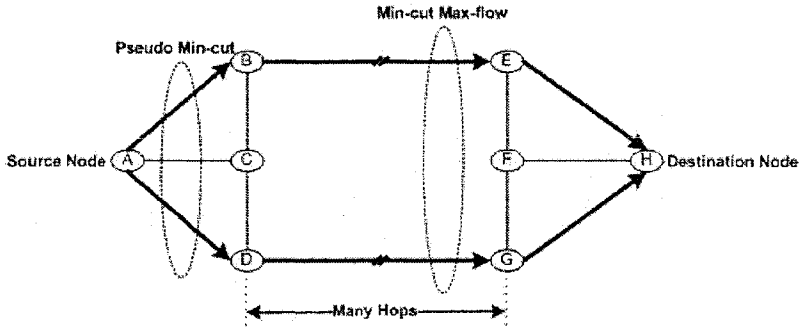
- The min-cut is equal to the pseudo min-cut.
- The min-cut is less than the pseudo min-cut.

The pseudo min-cut can not be less than the true min-cut because then it would be the min-cut. Thus, there can only be the two scenarios mentioned above. These two possible situations are depicted in figure C.2. Figure C.2a depicts the scenario where the min-cut is equal to the pseudo min-cut. In this instance, the technique produces the same results as a complex min-cut algorithm without the required solution time. In figure C.2b, the pseudo min-cut is greater then the actual min-cut. This situation is where both the source and destination nodes have more incident links then the available disjoint connections through the intermediate nodes. This scenario is as infrequent and barely affects normal operations. It is very unlikely that the available disjoint connections for a source-destination pair are determined by the structure of the intermediate nodes in the network. It is significantly more likely that the nodal degree of either the source or destination nodes will be the

bottleneck. As mentioned earlier, the effect of the pseudo min-cut being greater than the true min-cut is minimal. It only means that the algorithm will attempt to get another $N + 1$ set of paths. Thus the pseudo min-cut is used extensively in the SCP and MSCP heuristic algorithms.



(a) Min-cut is equal to pseudo min-cut



(b) Pseudo min-cut is greater than min-cut

Figure C.2.: Pseudo Min-Cut Scenarios

D. Transport Network Failures

The telecommunication networks that are moving valuable information throughout the world are engineering marvels. They are so important to the continuance of society that they have been declared critical infrastructure by our government. The mere mention of unscheduled downtime causes waves of panic throughout those most dependent on the system. Thus, it is monumentally important that we prevent or mitigate the problem of transport network failures. Normally with an asset this important it would be a simple choice to enact a series of procedures to prevent these outages. However, it has proved to be futile to protect such a mega structure from harm. When a telecommunication network traverses over 100,000 miles it is impossible to fully protect against damage. Failures will occur irrespective of how deep the cables are buried, how many warning signs are placed in the area, or how sturdy the cable carrying conduits are made [1]. It is a statistical certainty that portions of our telecommunications networks will fail. This certainty has been quantized as 4.39 fiber cuts/1000 miles annually [2]. This corresponds to approximately 500 FITS/mile or 500 failures in 10^9 hours of operation. Additionally, this failure probability has been further subdivided into metro and long haul networks. Where by the former experiences 13 fiber cuts/1000 miles and the latter has 3 fiber cuts/1000 miles.

D.1. Causes and Durations

This problem with the inevitability of failures in our telecommunication network became of increasing concern during the early 1990s as fiber networks came into prevalence. This led to the Crawford study [26] on the causes and durations of network failures. Figure D.1 presents the the probability of failure by cause from 160 failures recorded in the study.

The most prevalent cause of failures was due to contractor dig-ups, which accounted for a slight majority of the incidents. The second leading cause of failures were vehicle accidents involving overhead lines.

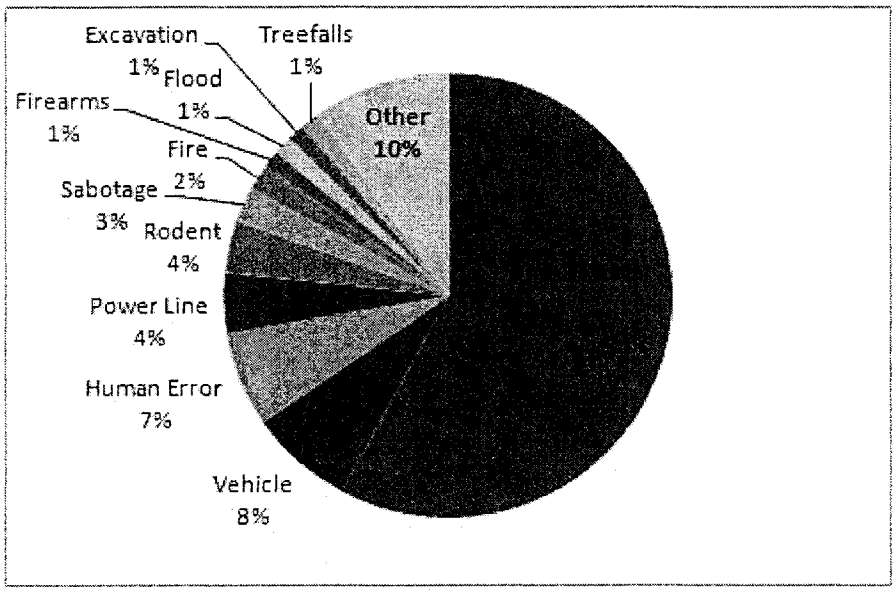


Figure D.1.: Fiber Optic Failures By Cause (data from [26])

Additionally, the Crawford study included details regarding the repair time and restoration time required after each failure. Figure D.2 depicts the statistics for these repair times and restoration times. The mean repair time was 14.2 hours with a maximum of over 100 hours. Likewise, the restoration of service took an average of 5.2 hours to complete. An important aspect of this study was that all recorded events were single-failures. This realization has led to a focus in research on creating networks that are resilient against single-failure scenarios.

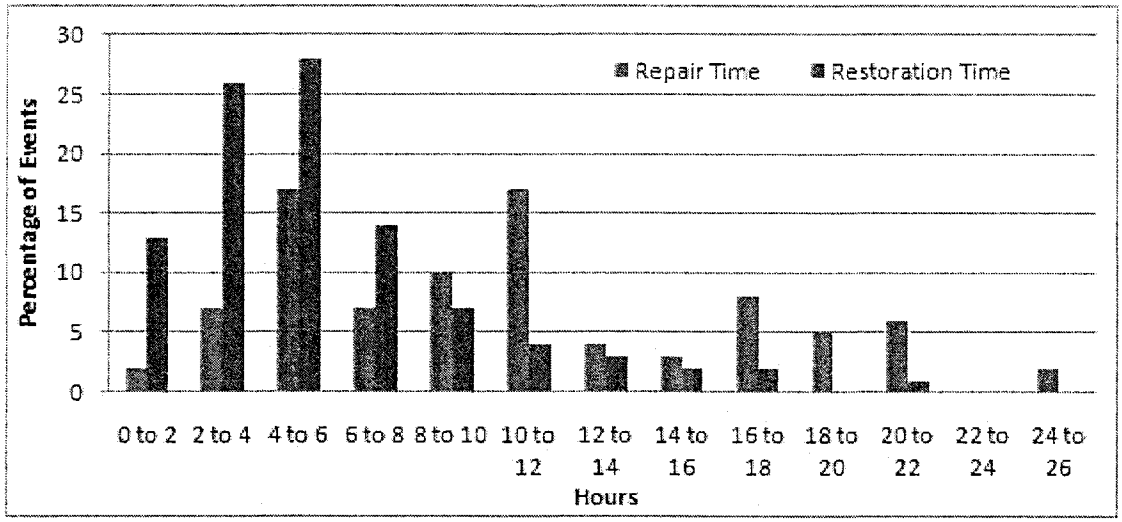


Figure D.2.: Histogram of Service Restoration and Repair Times (data from [26])

D.2. Failure Scenarios

There are two general types of failures that a telecommunications network can experience.

1. Single Link Failure
2. Node Failure

In the following sections; concise descriptions will be provided for each of the failure situations. These descriptions will provide details on what causes each failure how often they occur and what restoration/protection type is ideal for resolving them.

D.2.1. Single Link Failures

The link is the most identifiable point of failure for a network. Figure D.3 shows the effect of a link failure in a network. At the physical layer this is the most common failure type. Thus most network survivability techniques in the literature are concerned only with

this kind of failure [13, 28, 12, 15, 27, 29, 10, 11]. As mentioned earlier the probably of failure associated with single link is 500 FITs/mile. Due to the ability of network operators to simplify all other failure types into a set of link failures, the single link failure situation can be protected by most survivability techniques. An exception to this is the traditional approach to protecting against node failure scenarios, which will be alluded to in section D.2.2. All of the survivability algorithms presented in this thesis are based on the single link failure concept.

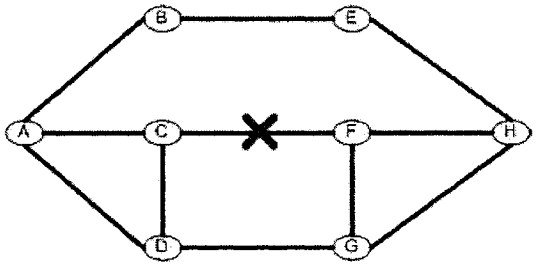
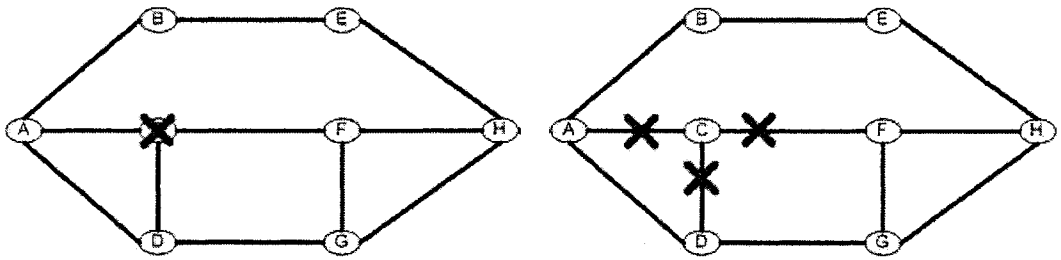


Figure D.3.: Single Link Failure

D.2.2. Node Failures

Node failures are the least common but most devastating of all failure scenarios. In a node failure two devastating events occur simultaneously. To begin with, all local traffic that connects directly to that node is immediately disconnected from the larger network. If the node is a gateway for simpler one-connected access networks, the traffic will be unrecoverable until the fault is repaired. Adding to that, every link that is incident to the failed node will lose functionality. This relationship between node failures and the corresponding link failures is summarized in figures D.4a and D.4b respectively. In high degree nodes, one node failure can result in the appearance of several concurrent link failures. Since it is reasonable to assume that a significant portion of the network will

be affected by such a failure, it is generally understood that it is more economical to use redundant hardware to protect against node failures, instead of rerouting [3].



(a) Node Failure (b) Node Failure Effect on Network Operation

Figure D.4.: Node Failures

D.3. Impacts

Network failures have a variety of impacts on customers and carriers. The loss of revenue associated with the failure of a major trunk group has been quoted as $100000/minute or more [2]. There is also a loss of reputation experienced by carriers when these disruptions occur [1]. In extreme cases this may lead to violations of service level agreements (SLAs), resulting in further decreases in revenue and reputation. Due to this high cost and loss of reputation, it has become increasingly important that service outages are minimized. Clearly, it is of the greatest importance that carriers attempt to reroute customer traffic over redundant connections while repairs are performed. If traffic can be restored over redundant circuitry in the order of a few seconds, the problems associated with failures become purely technical with minimal customer impact. The technical effects of outage durations were originally summarized by [30] and updated in [2] to include more recent communication protocols. Table D.1 presents these effects as target ranges for telecommunication carriers.

Table D.1.: Traffic Restoration Target Ranges(Data from [30])

Target Range	Duration	Characteristics
Protection Switching	<50ms	System re-frames
1	50ms-200ms	<5% voice band disconnects, SS7 switch overs, SMDS and ATM cell rerouting may begin
2	200ms-2s	DS1 CGA activates, TCP/IP protocol back off
3	2s-10s	All switched circuit services disconnect, private line disconnects, X.25 disconnects, TCP session timeouts start, hello protocol affected
4	10s-5min	All calls and data services are terminated, TCP/IP application layer programs timeout, routers flood network with LSA
Undesirable	5min-30min	Minor societal/business effects, noticeable Internet brownout
Unacceptable	>30min	Major societal impacts

The most desirable objective is the protection switching target range. In this target range restoration occurs in less than 50ms. Restoration times in this range are usually associated with 1+1 automated protection switching (APS) but with improvements in techniques and technology it might become obtainable by other techniques. In this range, the transmission system will only register a "hit". Client layers will perceive this as a damaged frame and will attempt retransmission. Due to high cost and excessive performance associated with this range, there has been discussion as to whether this should be a target [2]. However, it is still considered the telecommunication standard for voice grade quality. After the protection switching range is the 1^{st} target range. During this time period, Switched Multimegabit Date Service (SMDS) and Asynchronous Transfer Mode (ATM) cell rerouting may begin, however this is only a concern as the duration approaches 200ms. Fortunately, TCP/IP only performs retransmissions, which will not permanently degrade it's quality of service.

In the second target range, some carrier group alarms (CGA) for old Digital Signal 1 (DS1) circuits trip and the TCP/IP protocol starts performing exponential back-off and window resizing operations. But otherwise all connections at the client layers remain intact. Due to minimal amount of adverse effects associated with the 2^{nd} target range, it has been touted as the defacto standard for IP traffic restoration [2]. Following the 2^{nd} target range, the severity of the technical issues associated with the outage becomes unacceptable. At the third target range and beyond, multiple client layers may attempt to perform their own connection restoration techniques, leading to suboptimal results.

D.4. Service Level Agreements

To provide customer assurances that connections will be available when required and disruptions will be minimal, service level agreements have been created [3]. The levels of service offered are summarized in table D.2. While these standards have not been officially defined yet, they still provide a good idea of the different guarantees that can be offered.

Table D.2.: Classes of Service

Service Level	Description
Platinum	Highest level of service and fastest restoration time. Restoration time is typically 50ms
Gold	High availability and fast restoration times. Restoration time is a few hundred milliseconds
Silver	Best effort services. Typically involves re-provisioning of connections
Bronze	No protection is provided with this service
Lead	Lowest availability and lowest priority. Consists of preemptable connections

From table D.2, the best service level offered is platinum. The platinum service level is

associated with restoration times in the protection switching target range and very high availability guarantees. The second level of service is gold. This service level is usually associated with protection and restoration schemes that have a shared backup route. Its restoration time is usually associated with the second target range. The silver service level is associated with best effort services like IP rerouting and lower layer connection re-provisioning. At this level, attempts will be made to restore traffic when a failure occurs but capacity will not be proactively reserved for the connection. The bronze service level is used for unprotected traffic. No attempts will be made to restore the traffic until the failed hardware has been repaired. The lowest service level is lead. At this level, the traffic consists of connections temporarily routed over the reserved capacity for higher service level connections. If the higher service level connection that reserved the capacity requires it, the lead level connection will be preempted.

E. Fundamental Algorithms

Algorithm E.1 Dijkstra

Definitions:

- $G(N, J)$: a network G with a set of nodes N and set of links J.
- $D \in N$: the destination node.
- *d(A):* denote the distance of vertex $A \in N$ from source vertex $S \in N$. where $d(S) = 0$.
- *P(A):* denote the predecessor of vertex A along the path.
- Γ_A:set of neighbor vertices of vertex A.
- $l(ij)$: the cost of link from vertex i to vertex j.
- V: the nodes that have been visited by the algorithm.

Algorithm:

1.
   - $d(S) = 0$,
   - $d(A) = \begin{cases} L(SA) & A \in \Gamma_S \\ \infty & otherwise \end{cases}$
   - $V = N - \{S\}$
   - $P(A) = S \quad \forall A \in V$

2.
   - Find $j \in V$ such that $d(j) = min\,[d(i)]$, $i \in V$.
   - $V = V - \{j\}$
   - if $(j = D)$, END; otherwise go to step 3

3.
   - $\forall i \in \Gamma_j \cap V$, if $d(j) + l(ij) < d(i)$, set
     $d(i) = d(j) + l(ij)$, $P(i) = j$
   - Go to Step 2

Algorithm E.2 Modified Dijkstra

Definitions:

- $G(N, J)$: a network G with a set of nodes N and set of links J.
- $D \in N$: the destination node.
- *d(A):* denote the distance of vertex $A \in N$ from source vertex $S \in N$. where $d(S) = 0$.
- *P(A):* denote the predecessor of vertex A along the path.
- Γ_A:set of neighbor vertices of vertex A.
- $l(ij)$: the cost of link from vertex i to vertex j.
- V: the nodes that have been visited by the algorithm.

Algorithm:

1.
   - $d(S) = 0$,
   - $d(A) = \begin{cases} L(SA) & A \in \Gamma_S \\ \infty & otherwise \end{cases}$
   - $V = N - \{S\}$
   - $P(A) = S \quad \forall A \in V$

2.
   - Find $j \in V$ such that $d(j) = min\,[d(i)]$, $i \in V$.
   - $V = V - \{j\}$
   - if $(j = D)$, END; otherwise go to step 3

3.
   - $\forall i \in \Gamma_j \cap V$, if $d(j) + l(ij) < d(i)$, set
     $d(i) = d(j) + l(ij)$, $P(i) = j$
     $V = V \cup \{i\}$
   - Go to Step 2

Algorithm E.3 Node Set Dijkstra

Definitions:

- $G(N, J)$: a network G with a set of nodes N and set of links J.
- $D \subset N$: a set of potential destination nodes.
- *d(A):* denote the distance of vertex $A \in N$ from source vertex $S \in N$. where $d(S) = 0$.
- *P(A):* denote the predecessor of vertex A along the path.
- Γ_A:set of neighbor vertices of vertex A.
- $l(ij)$: the cost of link from vertex i to vertex j.
- V: the nodes that have been visited by the algorithm.

Algorithm:

1.
   - $d(S) = 0$,
   - $d(A) = \begin{cases} L(SA) & A \in \Gamma_S \\ \infty & otherwise \end{cases}$
   - $V = N - \{S\}$
   - $P(A) = S \quad \forall A \in V$

2.
   - Find $j \in V$ such that $d(j) = min\,[d(i)]$, $i \in V$.
   - $V = V - \{j\}$
   - if $(j \in D)$, END; otherwise go to step 3

3.
   - $\forall i \in \Gamma_j \cap V$, if $d(j) + l(ij) < d(i)$, set
     $d(i) = d(j) + l(ij)$, $P(i) = j$
     $V = V \cup \{i\}$
   - Go to Step 2

Algorithm E.4 Intersection Routing Technique Dijkstra's Algorithm

Definitions:

- $G(N, J)$: a network G with a set of nodes N and set of links J.
- $D \subset N$: a set of potential destination nodes.
- *d(A):* denote the distance of vertex $A \in N$ from source vertex $S \in N$. where $d(S) = 0$.
- *P(A):* denote the predecessor of vertex A along the path.
- Γ_A:set of neighbor vertices of vertex A.
- $l(ij)$: the cost of link from vertex i to vertex j.
- V: the nodes that have been visited by the algorithm.

Algorithm:

1.
   - $d(S) = 0$,
   - $d(A) = \begin{cases} L(SA) & A \in \Gamma_S \\ \infty & otherwise \end{cases}$
   - $V = N - \{S\}$
   - $P(A) = S \quad \forall A \in V$

2.
   - Find $j \in V$ such that $d(j) = min\,[d(i)]$, $i \in V$.
   - $V = V - \{j\}$
   - if $(D \notin V)$, END; otherwise go to step 3

3.
   - $\forall i \in \Gamma_j \cap V$, if $d(j) + l(ij) < d(i)$, set
     $d(i) = d(j) + l(ij)$, $P(i) = j$
     $V = V \cup \{i\}$
   - Go to Step 2

Algorithm E.5 Bhandari's Algorithm

Definitions:

- $G(N, J)$: a network G with a set of nodes N and set of links J.
- $D \in N$: the destination vertex.
- *d(A):* denote the distance of vertex $A \in N$ from source vertex $S \in N$. where $d(S) = 0$.
- *P(A):* denote the predecessor of vertex A along the path.
- Γ_A:set of neighbor vertices of vertex A.
- $l(ij)$: the cost of link from vertex i to vertex j.
- V: the nodes that have been visited by the algorithm.
- $R_1(r)$: the first set of links along the paths between source vertex S and destination vertex D.
- $R_2(r)$: the second set of links along the paths between source vertex S and destination vertex D.
- $R_P(r)$: the final set of links along the edge disjoint shortest pair between source vertex S and destination vertex D.

Algorithm:

1. Run the Modified Dijkstra Algorithm
   - let $R_1(r)$ = set directed links $\{ij\} \in N$ along the shortest route
   - if $(R_1(r) = \emptyset)$, Fail
2. $\forall \{ij\} \in R_1(r)$
   $l(ji) = -l(ij)$
   $l(ij) = \infty$
3. Run the Modified Dijkstra Algorithm
   - let $R_2(r)$ = set directed links $\{ij\} \in V$ the shortest route
   - if $(R_2(r) = \emptyset)$, Fail
4. $\forall \{ij\} \in R_1(r) \cup R_2(r)$
   $R_P(r) = R_1(r) \triangle R_2(r)$

Algorithm E.6 K Shortest Path Set Algorithm

Definitions:

- $G(N, J)$: a network G with a set of nodes N and set of links J.
- $D \in N$: the destination vertex.
- *d(A):* denote the distance of vertex $A \in N$ from source vertex $S \in N$. where $d(S) = 0$.
- *P(A):* denote the predecessor of vertex A along the path.
- Γ_A:set of neighbor vertices of vertex A.
- $l(ij)$: the cost of link from vertex i to vertex j.
- V: the nodes that have been visited by the algorithm.
- $R_1(r)$: the first set of links along the paths between source vertex S and destination vertex D.
- $R_2(r)$: the second set of links along the paths between source vertex S and destination vertex D.
- $R_P(r)$: the final set of links along the edge disjoint shortest pair between source vertex S and destination vertex D.
- k: the current number of disjoint paths.

Algorithm:

1. Run the Modified Dijkstra Algorithm
   - let $R_1(r) =$ set directed links $\{ij\} \in N$ along the shortest route
   - $k = 1$
   - if $(R_1(r) = \emptyset)$, Fail
2. $\forall \{ij\} \in R_1(r)$
   $l(ji) = -l(ij)$
   $l(ij) = \infty$
3. Run the Modified Dijkstra Algorithm
   - let $R_2(r) =$ set directed links $\{ij\} \in V$ the shortest route
   - $k = k + 1$
   - if $(R_2(r) = \emptyset)$, Fail
4. $\forall \{ij\} \in R_1(r) \cup R_2(r)$
   $R_P(r) = R_1(r) \triangle R_2(r)$
   $R_1(r) = R_P(r)$
5. if $k < K$, go to step 2