Traffic Rerouting Optimization in Network Recovery: A Performance Study

A Thesis
Submitted to the Faculty
of
Drexel University
by
Fei Bao
in partial fulfillment of the
requirements for the degree
of
Master of Science
September 2009
# Table of Contents

List of Tables . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . iv  
List of Figures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . vi  
Abstract . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . vii  
1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1  
  1.1 Network Recovery Process . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2  
  1.2 Network Recovery Characteristics: Restoration and Protection . . . . . . . . . . . . . . 3  
  1.3 Traffic Rerouting and Motivations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4  
  1.3.1 MPLS Traffic Engineering (TE) Restoration . . . . . . . . . . . . . . . . . . . . . . 4  
  1.3.2 MPLS Traffic Engineering (TE) Protection . . . . . . . . . . . . . . . . . . . . . . . 5  
  1.3.3 IP Traffic Restoration and Protection . . . . . . . . . . . . . . . . . . . . . . . . . . . 6  
  1.4 Contributions and Organization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7  
2 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8  
  2.1 Network Reliability and Failure Recovery . . . . . . . . . . . . . . . . . . . . . . . . . . 8  
  2.2 Traffic Rerouting . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9  
  2.3 OPNET . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9  
3 Network Failure Recovery Optimization Modeling . . . . . . . . . . . . . . . . . . . . . . 11  
  3.1 Model Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11  
  3.1.1 A 2-path model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11  
  3.1.2 Mathematical Modeling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12  
  3.1.3 Terms and notations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12  
  3.1.4 From continuous time Markov process to Discrete time Markov process . . . . . . . . 14  
  3.1.5 Bellman’s equations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15  
3.2 Uniformization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4.1</td>
<td>Threshold Optimal Control</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>4.2</td>
<td>Rerouting Traffic Optimization Algorithm</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>4.3</td>
<td>Optimization Model of Rerouting Flows with Different Bandwidth</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>4.4</td>
<td>Dynamic Programming Algorithm for the Model of Flows with Different Bandwidth</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>5.1</td>
<td>Simulation Description</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>5.1.1</td>
<td>Flow-based simulation</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>5.1.2</td>
<td>Customized routing protocol</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>5.1.3</td>
<td>IP route table setup</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>5.2</td>
<td>Numerical and Simulation Results</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>5.3</td>
<td>Testbed Setup</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>5.4</td>
<td>Experimental Results</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>5.4.1</td>
<td>All the flows have the same unit bandwidth</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>5.4.2</td>
<td>Flows may have different bandwidth</td>
<td>51</td>
</tr>
<tr>
<td>6</td>
<td>6.1</td>
<td>Conclusion</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>6.2</td>
<td>Limitations and Future Work</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bibliography</td>
<td>55</td>
</tr>
</tbody>
</table>
List of Tables

5.1 Measured Moving Cost and Holding Cost (bit/s) 50
List of Figures

1.1 Two Cycles of Network Recovery Process: recovery cycle and reversion cycle . . . . . . . 3

3.1 A 2-path network recovery procedure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
3.2 Mathematical model of the 2-path network . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
3.3 A 3D MDP model of network recovery . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
3.4 Markov transitions of the state \((0, 0, 0)\) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
3.5 Markov transitions of the state \((0, x_2, 0)\) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
3.6 Markov transitions of the state \((0, M, 0)\) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
3.7 Markov transitions of the state \((x_1, 0, 0)\) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
3.8 Markov transitions of the state \((M, 0, 0)\) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
3.9 Markov transitions of the state \((x_1, x_2, 0)\), when \(x_1 + x_2 < M\) . . . . . . . . . . . . . . . 19
3.10 Markov transitions of the state \((x_1, x_2, 0)\) when \(x_1 + x_2 = M\) . . . . . . . . . . . . . . . 20
3.11 Markov transitions of the state \((0, 0, 1)\) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
3.12 Markov transitions of the state \((0, y, 1)\) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
3.13 Markov transitions of the state \((0, M, 1)\) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
3.14 Uniformization of transitions of the state \((0, 0, 0)\) . . . . . . . . . . . . . . . . . . . . . . . . . 24
3.15 Uniformization of transitions of the state \((0, x_2, 0)\) . . . . . . . . . . . . . . . . . . . . . . . . . 24
3.16 Uniformization of transitions of the state \((0, M, 0)\) . . . . . . . . . . . . . . . . . . . . . . . . . 25
3.17 Uniformization of transitions of the state \((x_1, 0, 0)\) . . . . . . . . . . . . . . . . . . . . . . . . . 25
3.18 Uniformization of transitions of the state \((M, 0, 0)\) . . . . . . . . . . . . . . . . . . . . . . . . . 26
3.19 Uniformization of transitions of the state \((x_1, x_2, 0)\), when \(x_1 + x_2 < M\) . . . . . . . . . . . . . . 27
3.20 Uniformization of transitions of the state \((x_1, x_2, 0)\) when \(x_1 + x_2 = M\) . . . . . . . . . . . . . . 27
3.21 Uniformization of transitions of the state \((0, 0, 1)\) . . . . . . . . . . . . . . . . . . . . . . . . . 28
LIST OF FIGURES

3.22 Uniformization of transitions of the state \((0, y, 1)\) .................................................. 29
3.23 Uniformization of transitions of the state \((0, M, 1)\) .................................................. 30

5.1 OPNET simulation network ................................................................. 40
5.2 Optimal rerouting traffic \(\vec{x}^*\) with \(M = 30, \lambda = 0.15, \mu = 0.01, \lambda_f = 0.1, \mu_f = 0.01, c_2 = 2.0, c_1 = 1.8\). Top: \(c_2 > c_1 + \lambda_f c_m\), Bottom: \(c_2 < c_1 + \lambda_f c_m\) ........................................ 42
5.3 Optimal rerouting amount of flows \(x_1^*\) vs. \(c_1\) .................................................. 43
5.4 Optimal rerouting amount of flows \(x_1^*\) vs. \(c_2\) .................................................. 44
5.5 Optimal rerouting amount of flows \(x_1^*\) vs. \(\lambda_f\) with different \(\mu_f\) ..................... 45
5.6 Optimal rerouting amount of flows \(x_1^*\) vs. \(\lambda, \mu\) while keeping \(\lambda/\mu = 100.0\) .......................... 45
5.7 Logical network topology of the testbed .................................................. 47
5.8 Physical network topology of the testbed and an example of UDP flow .......................... 48
5.9 Optimal rerouting traffic \(\vec{x}^*\) with \(M\) from 50 to 100, \(\lambda = 0.1, \mu = 0.01, \lambda_f = 0.01019, \mu_f = 0.0195, c_2 = 224.01k, c_1 = 192.036k, c_m = 1046.8k\) ........................................ 49
5.10 Optimal rerouting amount of flows \(x_1^*\) vs. \(\lambda\) when \(\mu = 0.01\) .......................... 51
5.11 Long run time average optimal costs with varying \(\lambda\) .......................................... 52
Abstract
Traffic Rerouting Optimization in Network Recovery: A Performance Study
Fei Bao
Jaudelice Cavalcante de Oliveira, Ph.D.

In recent decades, the Internet has evolved from a special purpose computer network to a central platform of our daily communications. As it takes an increasingly important role in our everyday life, network reliability becomes even more critical. This is also one of the reasons why the Internet has been widely ubiquitous, due to its recovery ability. Internet protocol (IP) routing, such as Open Shortest Path First (OSPF) protocol, updates the forwarding tables to reflect alternative routes when the network topology changes. Due to the time consumed by the rerouting process after a failure, most recovery mechanisms are proposed to be proactive instead of reactive [1], [2], [3], [4], [5]. That is because in proactive approaches backup paths are pre-configured while reactive schemes such as that of OSPF usually are too slow to satisfy the recovery time requirements. After failures, all the traffic is moved to the backup path. After the traffic is transferred to the backup path, the fault on the primary path is repaired. Following the repair of the primary path, the traffic may or may not be switched back to the primary path. There are two important points that need to be considered when deciding on rerouting traffic: i) the primary path may be flapping (fails again), ii) traffic rerouting cost may be too high. In this thesis, we investigate the performance of a mathematical model for traffic rerouting after repairs. Our contribution is to verify the the optimal traffic rerouting threshold-type control algorithm with simulations and an experiment. The simulations were performed in OPNET and the experiments were built on Sun VM VirtualBox technology.
Chapter 1: Introduction

Nowadays our everyday life relies heavily on communication networks: business communications, phone calls, email, on-line banking, e-games and even watching TV or listening to music. The fact that our business and personal life depend more and more on the communication services results in the intensified interest in network reliability. During the past few years, the research community and industry have begun studying how to provide highly reliable networks and this trend is undoubtedly to continue in the future as the role of networks becomes more important in our life. Though there have been extensive research in network recovery, several problems remain unanswered due to its complex, fascinating and rapidly evolving features. Since network recovery is multi-dimensional in terms of characteristics, many criteria are required to be considered for a network recovery design. For example, the convergence time of the recovery process, the cost of traffic rerouting, etc. And it is these criteria that lead to various trade-offs during the decision-making process of network recovery design.

Though a wide variety of recovery schemes exists, all of them have a similar process composed of two cycles [6], [7]: the recovery cycle and, following it, the reversion or dynamic rerouting cycle. The recovery cycle detects a fault and restores traffic onto the backup paths. After it, the network is considered operational again. However, backup paths may have higher cost or be less optimal than primary paths. After the fault repair, the second cycle redirects the traffic from the backup path back to the working primary path [7]. In this case, the rerouting technique is called revertive or dynamic rerouting. This corresponding phase is called reversion cycle.

The reversion cycle can be planned well in advance. In a reversion, there is no need for a hasty operation as in the recovery cycle. Therefore, a well-controlled switch-back or traffic rerouting mechanism with minimal cost is typically preferred [7], [8], [9].

Our objective in this thesis is to investigate the performance of the traffic rerouting optimization model
in the reversion cycle. To do this, we build an OPNET simulation to verify a simple mathematical model describing the reversion (the so-called rerouting) process.

In this chapter, besides introducing background that are will be used in the modeling and analysis sections, we explain how and why this system is modeled in the way shown later in Chapter 3. This chapter is organized as follows: In Section 1.1 we present the two cycles of network recovery processes. The applicable phase and necessity of the network recovery is also presented. In Section 1.2, we list the basic two categories of recovery mechanisms of IP and multiprotocol label switching (MPLS) networks. We also describe the corresponding type of recovery schemes covered by the model used in this thesis. Traffic rerouting of IP and MPLS networks is summarized in Section 1.3. Although not all of these rerouting schemes are covered by the model that we investigate, our simulation and testbed results show that most popular schemes are able to be optimized by the model. The organization of this thesis is given in Section 1.4.

1.1 Network Recovery Process

In this section, we introduce the two successive cycles of network recovery process. Most research focuses on certain phases of these two cycles. As shown in Fig. 1.1, when a failure in the network occurs, it takes some time before a node adjacent to the failure detects the fault. This fault detection time depends on many factors such as the frequency of signals sent or the time of fault detection in a lower network layer and notification time towards upper layers. Moreover, all the abnormal information from various signals need to be collected and analyzed to derive the exact fault state. Once the failure is confirmed by a node, this node needs to send the failure notification to other nodes in the network. But in some cases, the node may hold on for some time to see whether the lower layer recovery scheme can repair the fault. For example, for an IP over Synchronous Optical Network (SONET) Synchronous Digital Hierarchy (SDH), the cable cut could be quickly repaired by an optical recovery mechanism so the IP layer becomes operational within 50 ms (the failure recovery requirement). One of the research topic in the network recovery field is on the hold-off time. In case there is flapping, the hold-off time needs to be longer, the so-called dampening is used to help the network be stable. If the failure still exists, fault notification messages are sent out to inform those nodes that are involved in the recovery process. After all the involved nodes receive the fault notification messages, to coordinate the recovery operation, the exchange of messages between nodes need some time, the so-called
recovery operation time (this is not the overall recovery time). The last thing of the recovery cycle is to reroute all the traffic from the failed path to the recovered path. There have been many significant contributions in these two time spans. The objective is to make the overall recovery cycle time less than 50-ms for the Fast Rerouting (FRR) either in IP or MPLS networks [10].

![Figure 1.1: Two Cycles of Network Recovery Process: recovery cycle and reversion cycle](image)

After the recovery cycle, the network is operational again. From Fig. 1.1, we see that the reversion cycle is a resemblance of the recovery cycle. When the fault is repaired, depending on the lower layer, the repair may need some time to be detected. Again, after the fault clearing time, a hold-off time may be needed to enforce waiting until the repaired path is stable. And then the repair notification messages are sent out. Similar to the recovery cycle, the reversion operation needs some time and the traffic on the backup path may be rerouted back to the primary path.

It is very common that backup paths are usually less optimal than the primary path (the failed one). However, considering the cost of rerouting traffic, it may or may not be optimal to reroute all the traffic back to the repaired primary path.

### 1.2 Network Recovery Characteristics: Restoration and Protection

In Section 1.1, we see that signaling in the two cycles plays a critical role in network recovery. A typical distinction of network recovery mechanism is made by signaling between protection and restoration [11].
CHAPTER 1. INTRODUCTION

Both require signaling but the difference lies in the timing of signaling. In the case of protection, the backup paths are pre-configured and fully signaled before a failure occurs. In the case of restoration, the backup paths are either pre-configured or dynamically allocated, but when a failure occurs, additional signaling is required to establish the restoration path. Obviously, the advantage of protection over restoration is its shorter recovery operation time. However, restoration brings flexibilities regarding to the failure scenarios they can recover from and usually requires less backup capacity due to their shared nature.

In terms of the 50-ms requirement, though many schemes (single or multiple dimensional) have been proposed, a common conclusion is the pre-configured backup path is preferred either with protection or with restoration. That is because the routing protocols such as Interior Gateway Protocol (IGP) needs too much time to converge. In summary, the IP routing convergence time is too long for applications with real-time demands [12]. The reason that the convergence process does not satisfy the FRR requirement is because of its reactive nature. It reacts to a failure after it has happened and it involves the routers in a domain. Many schemes are proposed to solve the slow convergence problem. We will discuss this further in Chapter 2. In summary, a common point in all the proposed solutions is that the backup path needs to be pre-configured. This is the reason that a pre-configured backup path is used in this work.

1.3 Traffic Rerouting and Motivations

As we have described in Section 1.1, beside the recovery cycle, the reversion cycle also plays a role in network recovery process. To make our discussion focus on the nature of the problem, we use the general phase reversion cycle to express the IP dynamic rerouting cycle and MPLS reversion cycle. In this section, we discuss the problem: how is the traffic rerouted on the restored primary path from the backup path?

1.3.1 MPLS Traffic Engineering (TE) Restoration

If we decide to apply MPLS TE restoration as our network recovery mechanism, when a link or a node fails, each TE label switched path (LSP) affected by the failure is rerouted to the backup path determined by its head-end link state router (LSR). When the failed component is restored, any head-end LSR has the possibility to reuse the restored primary path. This relies on the optimization of evaluating the minimal cost or maximal revenue for all the TE LSPs. Usually there are two configurations for a TE LSP.

1. Static paths: When the primary path is restored, the head-end LSR reevaluates whether it is a preferred
CHAPTER 1. INTRODUCTION

path. Notice that here the reevaluation involves two considerations: 
i) Although the primary path is restored, it is possible that it fails again in the future; 
ii) Grounded on the current state (some LSPs may have been moved back to the primary path), whether the primary path is still a least-cost path.

2. Dynamically configured: No static path is specified. TE LSPs are purely dynamic. The head-end LSR reevaluates whether the primary path is the least-cost path through either interior gateway protocol (IGP) or TE metric [8]. By the TE metric or IGP, it is obvious that the possible future failure of the primary path is ignored. Once the primary path is restored, in most scenarios, it is the preferred path unless the network topology cost changed during the repairing process.

The model chosen in this thesis covers both cases above.

Another issue of the MPLS TE restoration is the reversion trigger. The simplest method is timer driven. After a time interval, the head-end LSR reevaluates whether the current state is optimal. Obviously, this scheme bring unnecessary resource wastage. A more common mechanism is event driven. When there is a new flow arrival/departure or IGP Open Shortest Path First routing protocol (OSPF) link state advertisement (LSA) or Intermediate System to Intermediate System (IS-IS) link state packet, the head-end LSR triggers the optimization decision process. Since this is the most common practice, the model chosen for this study represents the system as a 3-dimension Markov process, and uniformization [13] is in then used to transfer the continuous time process to a discrete time process, which keeps the nature of event driven.

1.3.2 MPLS Traffic Engineering (TE) Protection

With MPLS TE protection, once a link or node fails, the head-end LSR switches the traffic onto the backup LSP. When the failed component is restored, the head-end LSR either immediately switches the traffic back to the primary TE LSP or keeps the traffic flowing over the backup TE LSP. The problem of the former mode is the packet reordering and multiple traffic disruption due to a flapping resource. The side effect of the latter mode is the higher cost of the backup path.

As we mentioned before, the objective of the model is to minimize the cost, therefore reusing the repaired primary path is considered. Since the chosen model is grounded on the flows (or connections, TE LSPs) instead of packets, the complexity of reordering packets has been minimized. This brings a new question, “should we use globally revertive or locally revertive?”. According to the definition in [14], in global
reversion the head-end TE LSP decides whether to reuse the restored primary path upon its optimization evaluation. The local reversion is simple, all the TE LSPs are moved back to the primary path. However, due to the possible flapping of the primary path and the limited TE LSP attributes view, the global reversion is usually preferred to the local reversion [14]. Hence, we only consider the global reversion case. The other problem of the traffic disruption caused by the primary path’s flapping (keeping on and off) is considered by the failure process parameter in the chosen model.

1.3.3 IP Traffic Restoration and Protection

Since the dynamic on-demand routing in IP restoration does not satisfy the 50-ms requirement of FRR, we do not consider the schemes in that category. Thus, no matter for restoration or protection, in an IP network, only the pre-configured backup path is applicable. Among the wide range of IP fast recovery mechanisms, three main schemes are introduced here.

1. Static backup IP route: almost all the routers support this feature. With the static backup IP route, once the primary path is repaired, unless a IP TE scheme is applied, all the traffic are either moved to the primary path or kept on the backup path.

2. Tunneling backup approach [15], [16]: To provide fast recovery against a link or a node failure, for each of its neighbors, a special “Not-via” address is assigned. Forwarding tables are pre-calculated for these “Not-via” addresses without counting the protected component. Notice the “Not-via” address is not an IP address. The drawbacks of this scheme is: i) local protection (less optimal backup path); ii) inconvenient rerouting traffic back from the backup path to the primary path.

3. Multiple routing configurations (MRC) [9]: all the failure events are predicted and a set of backup configurations each for a failed event is pre-configured. Once the primary path fails, the backup configuration corresponding to the failure is initiated in the router that detects the failure. Differently from pre-configure static IP route, this method is local. And also it does not need to notify the surrounding neighbors, which is required by the tunneling scheme. Since this IP fast recovery scheme has a great potential for improved load balancing, as Kvalbein and et. al points out in [9], it is also applicable in our chosen optimization model.
1.4 Contributions and Organization

There is no general model for all the network recovery mechanisms. The following are the recovery features covered by the model we chose to study:

- Fast recovery
- Pre-configured backup path (static backup, MPLS TE protection, TE LSPs restoration or IP MRC)
- Event driven
- Global rerouting or reversion
- Connection-based suboptimal

This work uses dynamic programming algorithm and OPNET to investigate the optimization of traffic rerouting during the reversion cycle of the network recovery process. Our contribution includes:

1. Build a simulation by using dynamic programming algorithms and OPNET to attain the traffic rerouting optimization.
2. Build a testbed with Sun VM VirtualBox to run experiments that verifies our model’s results.

This thesis is organized as follows: In Chapter 2 we summarize the related work. In Chapter 3, we describe the model in detail. Bellman’s equations are presented for each state of the model. Two dynamic programming algorithms of optimizing the traffic rerouting are presented in Chapter 4. We describe and explain the simulation and the test bed setup, experiment configuration, numerical, simulation and experimental results in Chapter 5. The conclusion and future work is presented in Chapter 6.
Chapter 2: Related Work

2.1 Network Reliability and Failure Recovery

A lot of work has been done to improve the network reliability corresponding to failures of nodes or links in the Internet, especially in IP networks [17]. In this section, we focus on some important contributions aimed at restoring connectivity without a dynamic global reconvergence.

The Internet Engineering Task Force has standardized a framework called IP fast reroute [15]. Within this framework, a tunnelling approach based on the so-called “Not-via” addresses to handle link and node failures [16]. To protect against the failure of a component, say $P$, a special Not-via address is created for this component at each of $P$’s neighbors. Forwarding tables are then calculated for these addresses without using the protected component. Through this way, all nodes get a path to each of $P$’s neighbors, without passing through (“Not-via”) $P$.

MRC in [9] is similar to IP fast reroute in that loop free backup next hops are found by doing shortest path calculations on a subset of the network. MRC covers link and node failures using the same mechanism, and is strictly pre-configured.

Equal Cost Multi-Path (ECMP) is emulated in [2]. The proposal is to use MPLS setting up virtual links where equal cost paths to a destination are needed. This makes it possible to use one ECMP path as backup when another fails. The method used in [2] separates mechanisms to protect against link and node failures. Again, their scheme is strictly pre-configured.

In [3], Narvaez et al. proposed to do a local restoration once a failure is detected. In this paper, messages are notified and sent only to the nodes involved. A similar approach which considers dynamic traffic engineering is also presented in [18]. Besides them, there are also other similar approaches [19], [20], [21], which we call local rerouting. They are usually not strictly pre-configured, and can hence not recover traffic in the
required 50-ms in FRR.

2.2 Traffic Rerouting

Though there are many papers discussing traffic balancing, they can be divided into the following categories.

Resource sharing schemes for differentiated service: traffic balancing for quality of service is also proposed in addition to the proposed differentiated service architecture in 1998. After that, many traffic balancing schemes have been published. The main ideas of these work is to split the traffic onto different routes so the network resources can be optimized for differentiated service levels in terms of fairness, utilization, etc. For example, in [22], fairness is considered as the optimal objective among the different virtual paths. In [23], a distributed bandwidth pushing scheme that can dynamically adjust the spare bandwidth distribution over the network is proposed to improve the resource utilization. Another research direction is dynamically adjusting the routes reacting to traffic changes [24]. The goal is still to fulfill quality of service requirements of different service levels. An investigation of the real performance of traffic balancing is given in [25].

Traffic engineering for network failure: Intra- and inter-autonomous system (AS) transient link failures are common in operational IP networks. Robust intra- and inter-AS traffic engineering schemes have been proposed to optimize network performance against transient link failures. For example, an inter-AS traffic engineering scheme is proposed in [26]. In [27], a Border Gateway Protocol (BGP) traffic rerouting mechanism is presented. The authors in [28] propose a intra-AS traffic engineering based IGP to protect the network from link failures. Some papers use the interaction between robust intra- and inter-AS traffic engineering [29] to achieve better network performance under both normal state and any single intra- or inter-AS link failure.

2.3 OPNET

Optimized Network Engineering Tools (OPNET) is a simulation tool system capable of simulating communication networks with detailed protocol modeling and performance analysis. OPNET includes a graphical modeler, a dynamic, event-scheduled simulation kernel, flow analysis and object-based modeling.

The latest OPNET version consists of three solutions: i) Application Performance Management; ii) Network Engineering Operations and Planning and iii) Network Research and Development (R&D). The main tools we used are as follows:

- OPNET Modeler Network Simulation. It is a network simulation tool set that accelerates the R & D
process for analyzing and designing communication networks, devices, protocols, and applications. Users can analyze simulated networks to compare the impact of different technology designs on end-to-end behavior. Modeler incorporates a broad suite of protocols and technologies, and includes a development environment to enable modeling of all network types and technologies including MPLS and IPV6.

- Flow Analysis. Flow Analysis (FLAN) has a fluid simulation engine where each traffic flow is modeled analytically instead of the packet based Discrete Event Simulation (DES). A fluid simulation engine can model network traffic faster and is more scalable than packet based simulation though this speed improvement is achieved by ignoring packet level phenomena. Since the model that we are investigating is flow-based, the simulation analysis we used with OPNET is also FLAN instead of DES. One problem of the FLAN is the link utilization could be greater than 1 in FLAN. This functionality was designed for analyzing overflow. In our simulations, we set the link utilization to 1 whenever it is greater than 1.
Chapter 3: Network Failure Recovery Optimization Modeling

3.1 Model Description

In computer networks, when there is a link or a node failure, the traffic flows over the path including the failed component are either dropped or rerouted if there is a secondary path. When the failure is restored, we may consider whether to move the traffic flows back to the primary path. A Markov Decision Process (MDP) is chosen to investigate the optimal policy of how to move the flows from the secondary path back to the primary path. The model was proposed by Z. Zhao in [30] and is here included for the sake of completeness.

3.1.1 A 2-path model

From [1, 2, 3, 4, 5], we see it is common to have a pre-configured backup path in the network fast recovery scenario. Therefore, we chose a 2-path model where there are two parallel paths between the source node $S$ and the destination node $D$. The primary path is labeled as $P$ and the secondary (backup) path as $B$. Each flow is assumed to have the same unit bandwidth (a general case where each flow has its individual bandwidth will be discussed later). The primary path $P$ could fail due to a node or a link failure. After the failure, all the traffic flows would be moved from the primary path $P$ to the backup path $B$. The primary path is restored after a while and then some or all traffic flows over the backup path $B$ may be moved back to the primary path $P$, see Fig.3.1 (LHS: before failure; MID: after failure; RHS: after restore).

Figure 3.1: A 2-path network recovery procedure
3.1.2 Mathematical Modeling

In the mathematical model, we look at each path as a loss system, which means the maximal amount of flows each path can hold is their capacity without additional buffers. Furthermore, we assume the arrival process of the traffic flows is a Poisson process. The service time of each flow is random, which is assumed to have an exponential distribution (see the blue arrows in Fig.3.2). Thus, we have a 2-D M/M/c/c. The service cost over the primary path is assumed to be a linear function of the occupancy of flows (see the cyan dash arrow in Fig.3.2). The failure events of the primary path form a Poisson process (see $\lambda_f$ in Fig.3.2). After the failure, all the flows are moved to the backup path $B$ (see the red arrow in Fig.3.2). The restore process requires a random time interval to recover the failed component. The random variables expressing the flows’ service time and components’ restore time are assumed to have an exponential distribution (see $\mu_f$ in Fig.3.2). After restoration, some flows may be moved back to the primary path $P$ with a cost (see the green arrow in Fig.3.2). The flows that are moved back to the primary path $P$ and those staying on the backup path $B$ will continue their service with corresponding costs (see the yellow arrows in Fig.3.2). The total amount of flows is assumed to be finite, limited either by the capacity of the primary path $P$ or the capacity of the backup path $B$. If the total number of flows over the primary path $P$ is greater than the backup path $B$, some flows have to be discarded during the failure rerouting. Thus, the remaining amount of flows that will be considered after the primary path recovery is still finite. Hence, we only consider the upper bound is the primary path capacity because the case in which the upper bound is the backup path capacity is very similar.

3.1.3 Terms and notations

- $M$: the capacity of the primary path defined as the maximal number of flows it can hold.
- $\lambda_f$: the failure rate of the primary path.
- $\mu_f$: the restoration rate of the primary path. The mean restore time is $\mu_f^{-1}$.
- $\lambda$: the arrival rate of traffic flows.
- $\mu^{-1}$: the average service time of traffic flows.
• $c_1$: the holding cost rate of a flow over the primary path $P$.

• $c_2$: the holding cost rate of a flow over the backup path $B$.

• $c_1 x_1 + c_2 x_2$: the holding cost flows from $S$ to $D$. The cost is assumed as a linear function of the current occupancy of flows.

• $c_m$: the moving cost rate of a flow. The total moving cost is a linear function of the amount of flows being moved $c_m x$.

• $(x_1, x_2, p_f)$: the current state, where $x_1, x_2$ are the occupancy of flows over the primary path $P$ and the backup path $B$, and $p_f$ is the current status of the primary path. When $p_f = 0$, the primary path works; when $p_f = 1$, the primary path fails.

There are two planes in Fig.3.3. The first plane is when $p_f = 0$, which means the primary path works fine. At the beginning, we have $y$ flows over the primary path $P$ and 0 over the backup path $B$. When $P$ fails, all $y$ flows are moved from $P$ to $B$. Thus, the system state changes from $(y, 0, 0)$ to $(0, y, 1)$ with transition rate $\lambda_f$. When $P$ is restored with rate $\mu_f$, we have to make a decision on how many flows will be moved back to $P$. The system state changes to $(x_1, x_2, 0)$ where $x_1 + x_2 \leq y$ (depending whether there is any flow serviced completely over $B$ during the restoring process of $P$). When the primary path works fine again, we may have an arrival of a new flow or departure of a flow either on $P$ or $B$. 
Figure 3.3: A 3D MDP model of network recovery

3.1.4 From continuous time Markov process to Discrete time Markov process

Since the service time is a exponential r.v., the remaining service time of each flow after moving still has the same mean $\mu^{-1}$.

Considering the system is in state $(x_1, x_2, p_f)$, expressed by $\vec{x}$, the next state will be $\vec{y}$ under the control $\vec{z}$ (we may or may not control over each state) with the probability $p(\vec{y}|\vec{x}, \vec{z})$. The time interval $\tau$ between the transition to state $\vec{x}$ and the transition to $\vec{y}$ (the next state) is exponentially distributed with parameter $\gamma$, where $\gamma$ is the sum of all possible maximal outgoing rates of a state. We will specify $\gamma$ later.

$$\mathbb{P}\{\text{transition time interval } > \tau | \vec{x}, \vec{z}\} = e^{-\gamma \tau}.$$ 

So the p.d.f. $f_p(\tau) = \gamma e^{-\gamma \tau}, \tau \geq 0$ for all $\vec{x}, \vec{z}$. The expectation $E\{\tau\} = \int_0^\infty \tau \gamma e^{-\gamma \tau} d\tau = \frac{1}{\gamma}$. The state
and control at any time $t$ are denoted by $\bar{x}(t)$ and $\bar{z}(t)$, respectively, and stay constant between transitions.

We use the following notation:

- $t_k$: the time of occupancy of the $k$-th transition. $t_0 = 0$.
- $\tau_k \triangleq t_k - t_{k-1}$: the $k$-th transition time interval.
- $\bar{x}_k \triangleq \bar{x}(k)$: we have $\bar{x}(t) = \bar{x}_k$ for $t_k \leq t < t_{k+1}$.
- $\bar{z}_k \triangleq \bar{z}(k)$: we have $\bar{z}(t) = \bar{z}_k$ for $t_k \leq t < t_{k+1}$. $\bar{z} = \{a, d, r_{12}, r_{21}\} \in \{0, 1\}^4$.

We consider a cost function of the form

$$\lim_{N \to \infty} E\left\{ \int_0^{t_N} c(\bar{x}(t), \bar{z}(t)) \, dt \right\}$$

where $c$ is a given function (holding and/or moving function) as described in the previous notations. Similar to discrete-time problems, a policy $Z = \{\bar{z}_0, \bar{z}_1, \ldots\}$ where $\bar{z}_k$ is a function mapping states to controls with $\bar{z}_k(\bar{x}) \in \bar{Z}(\bar{x})$ for all $\bar{x}$. Under $\bar{z}$, the control applied in the interval $[t_k, t_{k+1})$ is $\bar{z}_k(\bar{x}_k)$.

Because states stay constant between transitions (meaning during $\gamma^{-1}$), the cost function is given by

$$V(\bar{x}_0) = \sum_{k=0}^{\infty} E\left\{ \int_{t_k}^{t_{k+1}} e^{-t} c(\bar{x}_k, \bar{z}_k(\bar{x}_k)) \, dt \right\}.$$ 

because $\bar{x}_k, \bar{z}_k(\bar{x}_k)$ are constants in $[t_k, t_{k+1})$

$$\therefore \sum_{k=0}^{\infty} E\left\{ \int_{t_k}^{t_{k+1}} c(\bar{x}_k, \bar{z}_k(\bar{x}_k)) \, dt \right\} = \sum_{k=0}^{\infty} \left( E\{\tau\} E\{c(\bar{x}_k, \bar{z}_k)\} \right) = \sum_{k=0}^{\infty} \frac{1}{\gamma} E\{c(\bar{x}_k, \bar{z}_k)\}.$$ 

Thus, a continuous time Markov process with cost $\lim_{N \to \infty} E\left\{ \int_0^{t_N} c(\bar{x}(t), \bar{z}(t)) \, dt \right\}$ and transition rate $\gamma$ is equivalent to a discrete time Markov process with long-run cost per stage $\frac{c(\bar{x}, \bar{z})}{\gamma}$.

### 3.1.5 Bellman’s equations
From the previous subsection, we see that to find the cost of a state in the discrete time Markov process, we need to figure out the total outgoing rate and its neighboring states of this state. To make the discussion clear, we show Bellman’s equations for individual states first and then use uniformization to get the final Bellman’s equations. Fig.3.3 shows two plans in our MDP, so we split our Bellman’s equation groups into two categories $i)$ when $p_f = 0$ and $ii)$ when $p_f = 1$.

1. When $p_f = 0$, the primary path $P$ works fine. There are seven cases.

   $i)$ : $x_1 = 0, x_2 = 0, p_f = 0$.

   ![Figure 3.4: Markov transitions of the state $(0, 0, 0)$](image)

   From Fig.3.4, we see that when the current state is $(0, 0, 0)$, the only possible events to the primary path $P$ are the new arrival of a flow or a failure. There is no departure event. Given $\gamma_{0,0,0} = \lambda + \lambda_f$, we obtain

   $$V_n(0,0,0) = \frac{1}{\gamma_{0,0,0}} \{ \lambda V_{n-1}(1,0,0) + \lambda_f V_{n-1}(0,0,1) \}.$$  \hspace{1cm} (3.1)

   $ii)$ : $x_1 = 0, 0 < x_2 < M, p_f = 0$.

   In Fig.3.5, for the current state $(0, x_2, 0)$, the possible events to the primary path $P$ are the new arrival of a flow, a failure and a departure from the backup path $B$. Given $\gamma_{0,x_2,0} = \lambda + \lambda_f + x_2 \mu$. 
we obtain

\[ V_n(0, x_2, 0) = \frac{1}{\gamma_{0, x_2, 0}} \left\{ c_2x_2 + \lambda V_{n-1}(1, x_2, 0) + \lambda_f V_{n-1}(0, x_2, 1) + x_2\mu V_{n-1}(0, x_2 - 1, 0) \right\}. \]  

(3.2)

\text{iii) : } x_1 = 0, x_2 = M, p_f = 0.

As shown in Fig.3.6, when \( x_2 = M \), all flow arrivals will be blocked. Here, we don’t consider the case where overflowed flows are rerouted to the backup path. There are only failure and departure events possible. Given \( \gamma_{0, M, 0} = \lambda_f + M\mu \), we obtain

\[ V_n(0, M, 0) = \frac{1}{\gamma_{0, M, 0}} \left\{ c_2M + \lambda_f V_{n-1}(0, M, 1) + M\mu V_{n-1}(0, M - 1, 0) \right\}. \]  

(3.3)
iv) $0 < x_1 < M, x_2 = 0, p_f = 0.$

![Markov transitions of the state $(x_1, 0, 0)$](image)

**Figure 3.7:** Markov transitions of the state $(x_1, 0, 0)$

Fig. 3.7 shows that when there is no flows on the backup path $B$ and the primary path $P$ is not filled up with flows, there are new arrival, departure and failure events. Given the total outgoing rate $\gamma_{x_1,0,0} = \lambda + \lambda_f + x_1 \mu$, we obtain

$$V_n(x_1, 0, 0) = \frac{1}{\gamma_{x_1,0,0}} \{ c_1 x_1 + \lambda V_{n-1}(x_1 + 1, 0, 0) + \lambda_f [V_{n-1}(0, x_1, 1) + c_m x_1]$$

$$+ x_1 \mu V_{n-1}(x_1 - 1, 0, 0) \}.$$  

(3.4)

v) $x_1 = M, x_2 = 0, p_f = 0.$

Fig. 3.8 shows the case in which the primary path is full of flows and no flow arrivals will be admitted. Thus, only departure and failure events are possible. Given $\gamma_{M,0,0} = \lambda_f + M \mu$, we obtain

$$V_n(M, 0, 0) = \frac{1}{\gamma_{M,0,0}} \{ c_1 M + \lambda_f [V_{n-1}(0, M, 1) + c_m M] + M \mu V_{n-1}(M - 1, 0, 0) \}.$$  

(3.5)

vi) $x_1 > 0, x_2 > 0, x_1 + x_2 < M, p_f = 0.$
From Fig. 3.9, we see that there are 4 next states of \((x_1, x_2, 0)\). Except one failure neighbouring state \((x_1, x_2, 1)\), the other three are reached through one arrival and two departure events. Note there is no arrival event on the backup path causing \(x_2\) to be changed to \(x_2 + 1\) since we only reroute flows at the failure or restoration time instant. There is no traffic balancing here. Given 
\[
\gamma_{x_1, x_2, 0} = \lambda + \lambda_f + (x_1 + x_2) \mu,
\]
we obtain
\[
V_n(x_1, x_2, 0) = \frac{1}{\gamma_{x_1, x_2, 0}} \left\{ c_1 x_1 + c_2 x_2 + \lambda V_{n-1}(x_1 + 1, x_2, 0) + \lambda_f [V_{n-1}(0, x_1 + x_2, 1) + c_m x_1] 
+ x_1 \mu V_{n-1}(x_1 - 1, x_2, 0) + x_2 \mu V_{n-1}(x_1, x_2 - 1, 0) \right\}.
\] (3.6)
(x_1 - 1, x_2, 0) \xrightarrow{\lambda} (x_1, x_2, 0) \xrightarrow{x_1 \mu} (0, M, 1) \xrightarrow{x_2 \mu} (x_1, x_2 - 1, 0)

\(\lambda\)
\(x_1 \mu\)
\(x_2 \mu\)

**Figure 3.10:** Markov transitions of the state \((x_1, x_2, 0)\) when \(x_1 + x_2 = M\)

The only difference between this case, Fig.3.10, and the previous one, Fig.3.9, is that there is no arrival being admitted. Given \(\gamma_{x_1, x_2, 0} = M\), we obtain

\[
V_n(x_1, x_2, 0) = \frac{1}{\gamma_{x_1, x_2, 0}} \{c_1 x_1 + c_2 x_2 + \lambda f[V_{n-1}(0, M, 1) + c_m x_1]
+ x_1 \mu V_{n-1}(x_1 - 1, x_2, 0) + x_2 \mu V_{n-1}(x_1, x_2 - 1, 0)\}.
\] (3.7)

2. When \(p_f = 1\), the primary path \(P\) fails, so there are 0 flows on it. All the flows are over the backup path \(B\). Suppose that the amount of flows over \(B\) is \(y\), there are three cases: i) \(y = 0\); ii) \(0 < y < M\); and iii) \(y = M\).

i) : \(y = 0, p_f = 1\). As shown in Fig.3.11, besides admitting the new arrival of flows, there is only one possible state to go when the primary path \(P\) is restored.

\[
V_n(0, 0, 1) = \frac{1}{\gamma_{0,0,1}} \{\lambda V_{n-1}(0, 1, 1) + \mu f V_{n-1}(0, 0, 0)\}.
\] (3.8)

where \(\gamma_{0,0,1} = \lambda + \mu f\).

ii) : \(0 < y < M, p_f = 1\). In Fig.3.12, to make the outgoing transitions clear, we do not draw the incoming transitions. From Fig.3.12, we see besides the new arrival and departure of flows,
CHAPTER 3. NETWORK FAILURE RECOVERY OPTIMIZATION MODELING

Figure 3.11: Markov transitions of the state (0, 0, 1)

there are a set \( \Omega = \{ (x_1, x_2, 0) | x_1 \geq 0, x + 2 \geq 0, x_1 + x_2 = y, y < M \} \) of state to go when the primary path \( P \) is restored. The control in the dynamic programming equations allows us to select one state with the minimal cost from the \( \Omega \).

Figure 3.12: Markov transitions of the state (0, y, 1)
\( V_n(0, y, 1) = \frac{1}{\gamma_{0,y,1}} \{ \lambda V_{n-1}(0, y+1, 1) + y\mu V_{n-1}(0, y-1, 1) \}
\]

\[ + \mu_f \min \{ V_{n-1}(0, y, 0) + 0, V_{n-1}(1, y-1, 0) + c_m, \]

\[ \ldots, V_{n-1}(x, y-x, 0) + c_m x, \ldots, V_{n-1}(y, 0, 0) + c_m y \} \}

(3.9)

where \( \gamma_{0,y,1} = \lambda + y\mu + \mu_f \).

\( iii) : y = M, p_f = 1 \). Similar to Fig.3.12 and (3.9), the only difference (see Fig.3.13) is there is no arrival being admitted. Thus, we obtain

![Figure 3.13: Markov transitions of the state (0, M, 1)](image-url)
\[ V_n(0, M, 1) = \frac{1}{\gamma_{0,M,1}} \{ M\mu V_{n-1}(0, M-1, 1) + \mu_f \min\{ V_{n-1}(0, M, 0) + 0, V_{n-1}(1, M-1, 0) + e_m, \ldots, V_{n-1}(x, M-x, 0) + e_{mx}, \ldots, V_{n-1}(M, 0, 0) + e_{mM}\} \} \]  

(3.10)

where \( \gamma_{0,M,1} = M\mu + \mu_f \).

### 3.2 Uniformization

In the previous section, we presented the Bellman’s equations for individual states. We see that \( \gamma \) varies as the occupancy of flows over the two paths \( P \) and \( B \) changes. To do uniformization, we need to define a common \( \gamma \) that is state-independent for all the state. Define

\[ \gamma = \lambda + \lambda_f + \mu_f + M\mu. \]

By this definition, we get the maximal possible outgoing rate of all the states in the system. Grounded on the new definition of \( \gamma \), we add dummy transitions \( (\gamma - \gamma_{x_1,x_2,p_f}) \) to each state in Fig.3.3, 3.4, ... Fig.3.12. Hence, the new Bellman’s equations are derived as follows.

1. When \( p_f = 0 \), again, there are seven cases.
   
   \( i \) : \( x_1 = 0, x_2 = 0, p_f = 0 \).
   
   In Fig.3.4, we added dummy transitions \( \gamma - \gamma_{0,0,0} = \mu_f + M\mu \), which starts from \( (0, 0, 0) \) and goes back to \( (0, 0, 0) \). Using the new \( \gamma \), we get

\[ V_n(0, 0, 0) = \frac{1}{\gamma} \{ \lambda V_{n-1}(1, 0, 0) + \lambda_f V_{n-1}(0, 0, 1) + (\mu_f + M\mu) V_{n-1}(0, 0, 0) \}. \]  

(3.11)

\( ii \) : \( x_1 = 0, 0 < x_2 < M, p_f = 0 \).
Comparing Fig.3.15 to Fig.3.5, we see that the added dummy transitions of the current state 
\((0, x_2, 0)\) is \(\gamma - \gamma_{0,x_2,0} = \mu_f + (M - x_2)\mu\). Using \(\gamma\) and the dummy transitions, We get

\[
V_n(0, x_2, 0) = \frac{1}{\gamma}\{c_2 x_2 + \lambda V_{n-1}(1, x_2, 0) + \lambda_f V_{n-1}(0, x_2, 1) \\
+ x_2 \mu V_{n-1}(0, x_2 - 1, 0) + (\mu_f + (M - x_2)\mu) V_{n-1}(0, x_2, 0)\}. \tag{3.12}
\]

\(\text{iii) : } x_1 = 0, x_2 = M, p_f = 0.\)
Similiarly, in Fig.3.16, to use $\gamma$, we added dummy transitions $\lambda + \mu_f$.

$$V_n(0, M, 0) = \frac{1}{\gamma} \{ c_2 M + \lambda_f V_{n-1}(0, M, 1) + M \mu V_{n-1}(0, M-1, 0) + (\lambda + \mu_f)V_{n-1}(0, M, 0) \} \cdot$$

\[ (3.13) \]

\[ iv) : 0 < x_1 < M, x_2 = 0, p_f = 0. \]
Fig. 3.17 and Fig. 3.17, we see that by adding dummy transitions $\mu_f + (M - x_1)\mu$, we obtain

$$V_n(x_1, 0, 0) = \frac{1}{\gamma} \{ c_1 x_1 + \lambda V_{n-1}(x_1 + 1, x_2, 0) + \lambda_f [V_{n-1}(0, x_1, 1) + c_m x_1] $$

$$+ x_1 \lambda V_{n-1}(x_1 - 1, 0, 0) + (\mu_f + (M - x_1)\mu) V_{n-1}(x_1, 0, 0) \}.$$  (3.14)

\(v)\) \(x_1 = M, x_2 = 0, p_f = 0.\)

![Figure 3.18: Uniformization of transitions of the state \(M, 0, 0\)](image)

Fig. 3.18 shows that by adding the dummy transitions $\lambda + \mu_f$, we obtain

$$V_n(M, 0, 0) = \frac{1}{\gamma} \{ c_1 M + \lambda_f [V_{n-1}(0, M, 1) + c_m M] + M \mu V_{n-1}(M - 1, 0, 0) $$

$$+ (\lambda + \mu_f) V_{n-1}(M, 0, 0) \}.$$  (3.15)

\(vi)\) \(x_1 > 0, x_2 > 0, x_1 + x_2 < M, p_f = 0.\)

Denote \(x_1 + x_2 = L\), then the dummy transitions we add in Fig. 3.19 are $\mu_f + (M - L)\mu$. We
obtain

\[
V_n(x_1, x_2, 0) = \frac{1}{\gamma} \{ c_1 x_1 + c_2 x_2 + \lambda V_{n-1}(x_1 + 1, 0, 0) + \lambda_f [V_{n-1}(0, x_1 + x_2, 1) + c_m x_1] \\
+ x_1 \mu V_{n-1}(x_1 - 1, x_2, 0) + x_2 \mu V_{n-1}(x_1, x_2 - 1, 0) \\
+ (\mu_f + (M-L)\mu) V_{n-1}(x_1, x_2, 0) \}. \tag{3.16}
\]

\(\text{vii) } x_1 > 0, x_2 > 0, x_1 + x_2 = M, p_f = 0.\)
Similar to (3.16), in Fig.3.20, the dummy transitions are \(\lambda + \mu f\), and we obtain

\[
V_n(x_1, x_2, 0) = \frac{1}{\gamma_{x_1, x_2, 0}} \left\{ c_1 x_1 + c_2 x_2 + \lambda f[V_{n-1}(0, M, 1) + c_m x_1] \right.
\]

\[
+ x_1 \mu V_{n-1}(x_1 - 1, x_2, 0) + x_2 \mu V_{n-1}(x_1, x_2 - 1, 0)
\]

\[
+ (\lambda + \mu f)V(x_1, x_2, 0) \right\}.
\]

(3.17)

2. When \(p_f = 1\), there are three cases: i) \(y = 0\); ii) \(0 < y < M\); and iii) \(y = M\).

i) \(y = 0, p_f = 1\), shown in Fig.3.21.

\[
V_n(0, 0, 1) = \frac{1}{\gamma} \left\{ \lambda V_{n-1}(0, 1, 1) + \mu f V_{n-1}(0, 0, 0) \right.
\]

\[
+ (\lambda + \mu f)V_{n-1}(0, 0, 0) \right\}.
\]

(3.18)

ii) \(0 < y < M, p_f = 1\), shown in Fig.3.22. Note that the outgoing transition \(\mu f\) has multiple choices but only one state with the minimal cost will be selected.
iii) $y = M, p_f = 1$. Similar to (3.19), from Fig.3.23, we get

\[
V_n(0, M, 1) = \frac{1}{\gamma} \{ M \mu V_{n-1}(0, M-1, 1) + (\lambda + \lambda_f) V_{n-1}(0, M, 1) \\
+ \mu_f \min\{V_{n-1}(0, M, 0) + 0, V_{n-1}(1, M-1, 0) + c_m, \\
\ldots, V_{n-1}(x, M-x, 0) + c_m x, \ldots, V_{n-1}(M, 0, 0) + c_m M \}\}. \tag{3.20}
\]
Figure 3.23: Uniformization of transitions of the state \((0, M, 1)\)
Chapter 4: Dynamic Programming Optimization Model Analysis and Extension

In the previous chapter, we describe the dynamic programming optimization model, proposed in [30], for rerouting flows with unit bandwidth. In this chapter, we replicate the analysis carried out in [30], which shows that the optimal control is a threshold control policy, under some conditions. This means that when the total amount of traffic that could be rerouted back to the primary path $P$ is less than a threshold, we should move all the traffic back to $P$ after $P$ is restored; if the total amount of traffic that could be moved is greater than the threshold, we only move the threshold amount of traffic back to $P$. An algorithm for calculating the threshold is presented.

Following [30], the simplified model where all the flows have the same unit bandwidth is extended to a more general model where flows have different bandwidth. Another dynamic programming algorithm is given to find out the selected set of flows to be rerouted grounded on the threshold found in the simplified model. This algorithm is sub-optimal but efficient.

This chapter is organized as follows: In Section 4.1, we show the optimal control of the simplified model where all the flows have the same unit bandwidth is a threshold policy. The algorithm for finding out the threshold is presented in Section 4.2. In Section 4.3, the simplified model of rerouting flows each with the same bandwidth is extended to a general model of rerouting flows each with its own bandwidth. A dynamic programming solution for selecting the flows to be rerouted based on the optimization result from the simplified model is presented in Section 4.4.

4.1 Threshold Optimal Control

From the previous chapter, we see the control of this model is actually in (3.19) and (3.20). In this section, we derive a threshold control policy as the optimal control policy from (3.16) and (3.19), (3.20), as in [30]. Under this threshold policy, we are able to determine how many traffic are being rerouted back to the primary...
CHAPTER 4. DYNAMIC PROGRAMMING OPTIMIZATION MODEL ANALYSIS AND EXTENSION

path $P$ when it is restored.

**Theorem 1.** [30] Define $V(x_1, x_2, 0) \triangleq \lim_{n \to \infty} V_n(x_1, x_2, 0)$ and $\Omega \triangleq \{(x_1, x_2)| x_1 \geq 0, x_2 \geq 0, x_1 + x_2 = L, L = 1, 2, \ldots, M\}$. For a certain $L$ in the set $\Omega$: i) if $c_2 < c_1 + \lambda f c_0$, $V(x_1, x_2, 0)$ is an increasing function of $x_1$; ii) if $c_2 > c_1 + \lambda f c_0$, $V(x_1, x_2, 0)$ is a decreasing function of $x_1$.

**Proof [30]:** Define $\Delta_n(x_1, x_2) \triangleq V_n(x_1, x_2, 0) - V_n(x_1 + 1, x_2 - 1, 0), x_2 \geq 1$. Denote $\Delta(x_1, x_2) \triangleq \lim_{n \to \infty} \Delta_n(x_1, x_2)$. From the definition, we get

$$\Delta(x_1, x_2) = \lim_{n \to \infty} (V_n(x_1, x_2, 0) - V_n(x_1 + 1, x_2 - 1, 0))$$

For a sequence of functions $\Delta_n(x_1, x_2)$, if $\forall n$, we have $\Delta_n(x_1, x_2) > 0, \forall (x_1, x_2) \in \Omega$, then $\Delta(x_1, x_2) > 0$, which means $V(x_1, x_2, 0)$ is a decreasing function of $x_1$ in the set of $\Omega$.

To show $\Delta_n(x_1, x_2) > 0, \forall n$ and $(x_1, x_2)$ for each $L$ in $\Omega$, we use induction as follows:

i) When $n = 0$, since the initial value of the cost function does not affect the final result, we are free to set for all $(x_1, x_2) \in \Omega$, the value function $V_0(x_1, x_2, 0) = 0$. It is easy to show when $n = 1, \forall (x_1, x_2) \in \Omega$ and $x_2 \geq 1$, if $c_2 > c_1 + \lambda f c_0$

$$\Delta_1(x_1, x_2) = \frac{1}{\gamma}(c_2 - c_1 - \lambda f c_0) > 0.$$  

ii) Assume at the $n_{th}$ step, $\forall (x_1, x_2) \in \Omega, x_2 \geq 1$, we have $\Delta_n(x_1, x_2) > 0$. Then at the $(n + 1)$th step, we obtain

$$\Delta_{n+1}(x_1, x_2) = \frac{1}{\gamma} \left\{c_2 - c_1 + \lambda (V_n(x_1 + 1, x_2, 0) - V_n(x_1 + 2, x_2 - 1, 0)) + \lambda f (V_n(0, L, 0) + c_0 x_1 - V_n(0, L, 0) - c_0 (x_1 + 1)) + \lambda x_1 \mu V_n(x_1 - 1, x_2, 0) - (x_1 + 1) \mu V_n(x_1, x_2 - 1, 0) + \lambda x_2 \mu V_n(x_1, x_2 - 1, 0) - (x_2 - 1) \mu V_n(x_1 + 1, x_2 - 2, 0) + ((M - L) \mu + \mu f) V_n(x_1, x_2, 0) - ((M - L) \mu + \mu f) V_n(x_1 + 1, x_2 - 1, 0) \right\}$$
CHAPTER 4. DYNAMIC PROGRAMMING OPTIMIZATION MODEL ANALYSIS AND EXTENSION

After some derivation, we obtain

\[
\Delta_{n+1}(x_1, x_2) = \frac{1}{\gamma} \left\{ c_2 - c_1 - \lambda f c_m \right. \\
+ \lambda [V_n(x_1 + 1, x_2, 0) - V_n(x_1 + 2, x_2 - 1, 0)] \\
+ x_1 \mu [V_n(x_1 - 1, x_2, 0) - V_n(x_1, x_2 - 1, 0)] \\
+ (x_2 - 1) \mu [V_n(x_1, x_2 - 1, 0) - V_n(x_1 + 1, x_2 - 2, 0)] \\
+ ((M - L) \mu + \mu_f) [V_n(x_1, x_2, 0) - V_n(x_1 + 1, x_2 - 1, 0)] \left\}. \right.
\]

By substituting the definition of \(\Delta_n(x_1, x_2)\), we obtain

\[
\Delta_{n+1}(x_1, x_2) = \frac{1}{\gamma} \left\{ c_2 - c_1 - \lambda f c_m \right. \\
+ \lambda \Delta_n(x_1 + 1, x_2) + x_1 \mu \Delta_n(x_1 - 1, x_2) + \\
+ (x_2 - 1) \mu \Delta_n(x_1, x_2) + ((M - L) \mu + \mu_f) \Delta_n(x_1, x_2) \left\}. \right.
\]

By substituting the definition of \(\Delta_n(x_1, x_2)\), we obtain

\[
\Delta_{n+1}(x_1, x_2) = \frac{1}{\gamma} \left\{ c_2 - c_1 - \lambda f c_m \\
+ \lambda \Delta_n(x_1 + 1, x_2) + x_1 \mu \Delta_n(x_1 - 1, x_2) + \\
+ (x_2 - 1) \mu \Delta_n(x_1, x_2) + ((M - L) \mu + \mu_f) \Delta_n(x_1, x_2) \right\}. \quad (4.1)
\]

We see the last two lines of (4.1) are composed of coefficients and \(\Delta_n(i, j)\). Each item of the last two lines in (4.1) is greater than 0 according to the induction assumption. Since \(c_2 > c_1 + \lambda f c_m\), the first line is also greater than 0. Hence, \(\Delta_{n+1}(x_1, x_2)\) is shown to be greater than 0 under that same condition. By induction, we show \(\Delta(x_1, x_2) > 0\), \(\forall (x_1, x_2) \in \Omega\) and \(x_2 \geq 1\). Thus, the cost function \(V(x_1, x_2, 0)\) is shown to be a decreasing function when \(c_2 > c_1 + \lambda f c_m\). Applying the same method, it is easy to show that when \(c_2 < c_1 + \lambda f c_m\), \(V(x_1, x_2, 0)\) is an increasing function of \(x_1\), ■

**Corollary 1** [30].

i): If \(c_2 > c_1 + \lambda f c_m\), the optimal (minimum) control of (3.19) and (3.20) is a threshold-type control. If the amount of flows on the backup path \(B\) is less than the threshold, all the flows should be moved back to the primary path \(P\) when it is restored to minimize the long-term cost of the system. If the amount of flows on the backup path \(B\) is greater than the threshold, only the amount of flows equaling to the threshold value should be moved back to \(P\).

ii): If \(c_2 \leq c_1 + \lambda f c_m\), the optimal (minimum) control is to keep all the flows on the backup path \(B\) until their remaining service is completed.

Proof [30]: The proof of Corollary 1 is grounded on Theorem 1. i): When \(c_2 > c_1 + \lambda f c_m\), from
Theorem 1., we show the cost function of $V(x_1, x_2, 0)$ is a decreasing function of $x_1$. Considering $c_m x_1$ is an increasing function of $x_1$, our optimal control policy of achieving the minimal sum of $V(x_1, x_2, 0) + c_m x_1$ is a trade-off for selecting a threshold $x_1$ to achieve the minimum cost.

\[ ii) \]: When $c_2 \leq c_1 + \lambda f c_m$, from Theorem 1., we show that if $c_2 < c_1 + \lambda f c_m$ then $V(x_1, x_2, 0)$ is an increasing function of $x_1$. Since $c_m x_1$ is also an increasing function of $x_1$, $V(x_1, x_2, 0) + c_m x_1$ is an increasing function of $x_1$. Hence, the minimum of $V(x_1, x_2, 0) + c_m x_1$ is achieved when $x_1 = 0$. That is to say, no flows should be moved from the backup path $B$ back to the primary path $P$ after $P$ is restored.

If $c_2 = c_1 + \lambda f c_m$, it is easy to check

\[ [V(x_1, x_2, 0) + c_m x_1] - [V(x_1 + 1, x_2 - 1, 0) + c_m(x_1 + 1)] = -c_m < 0. \]

Similar to the above analysis, $V(x_1, x_2, 0) + c_m x_1$ is an increasing function of $x_1$. Again, no flows should be moved from the backup path $B$ back to the primary path $P$ after $P$ is restored.

4.2 Rerouting Traffic Optimization Algorithm

From the previous modeling and analysis, we see that under the condition $c_2 > c_1 + \lambda f c_m$, our optimal control is a threshold policy. In this section we present how to attain the threshold given a system with the parameters $\lambda, \lambda f, \mu, \mu f, M$, see Algorithm 4.1.

4.3 Optimization Model of Rerouting Flows with Different Bandwidth

So far, the simplified model is grounded on the assumption that all the flows over the two paths $P$ and $B$ have the same unit bandwidth. In practice, this is not true. In this section, a more general model from [30], where each flow has its own bandwidth is presented.

First, we look at the $(x_1, x_2)$ as the amount of traffic instead of number of flows. For instance, $x_1, x_2$ are expressed in bytes. Through this way, each flow with its own bandwidth in the new the previous model could be looked as a bunch of “flows with a unit bandwidth”. Thus, the previous model is transferred to a traffic division optimization model.

If the total traffic is less than the threshold, we just follow the Corollary 1. to reroute all the flows. If the total traffic is greater than the threshold, there are two cases. \[ i) \]: If a flow is allowed to be divided and
Algorithm 4.1 Find out $x^*_1$ through $\min \{ V_n[i, j, k] + c_{mi} \}$

$$\gamma = \lambda + \lambda_f + \mu_f + M \mu$$

for all $0 \leq i, j \leq M$, $i + j \leq M$ and $k = 0, 1$ do

$V[i, j, k] \leftarrow 0$

end for

while $\Delta_{max} - \Delta_{min} \geq \Delta_{min} \epsilon$ do

for $i = 0$ to $M$ do

for $j = 0$ to $M - i$ do

$preV = V[i, j, k]$

if $k = 0$ then

Calculate $V[i, j, 0]$

else if $k = 1$ then

if $i = 0, j = 0$ then

$V[0, 0, 1] = \frac{1}{\epsilon} \{ \lambda V[0, 1, 1] + \mu_f V[0, 0, 0] + (\lambda_f + M \mu)V[0, 0, 0] \}$

else

$y = i + j$

Calculate $\text{mincost}(y)$ \{ $\text{mincost}$ returns the minimal cost of $V[i, j, k] + c_{mi}$ over $y = i + j$ \}

if $i + j < M$ then

$V[0, y, 1] = \frac{1}{\epsilon} \{ \lambda V[0, y + 1, 1] + \mu_f V[0, y - 1, 1] + (\lambda_f + (M - y) \mu)V[0, y, 1] + \mu_f \cdot \text{mincost}(y) \}$

else if $i + j = M$ then

$V[0, M, 1] = \frac{1}{\epsilon} \{ M \mu V[0, M - 1, 1] + (\lambda + \lambda_f)V[0, M, 1] + \mu_f \cdot \text{mincost}(M) \}$

end if

end if

end if

end if

$\Delta = V[i, j, k] - preV$;

if $\Delta_{min} > \Delta$ then

$\Delta_{min} = \Delta$

end if

if $\Delta_{max} < \Delta$ then

$\Delta_{max} = \Delta$

end if

end for

end while

Calculate $\text{mincost}(y)$

$\text{min} = V[0, y, 0]$

for $i = 1$ to $y$ do

if $\text{min} > V[i, y - i, 0] + c_{mi}$ then

$\text{min} = V[i, y - i, 0] + c_{mi}$

$x^*_1 = i$

end if

end for

$x^*_1[y] = x^*_1$ \{ Store the optimal control threshold value corresponding to $y$ \}

return $\text{min}$
transferred through two paths, then our conclusion from the previous model can be applied directly into the new model. \( ii \): However, the traffic division is not preferred in many realistic scenarios, in which case, the optimal threshold \( x_1^* \) that is obtained from the simplified model where all the flows have the same unit bandwidth may not be achieved in the new model.

In this section, we focus on solving the latter case \( ii \). Grounded on the threshold \( x_1^* \) obtained previously, a dynamic programming algorithm is proposed to solve this problem in the new model.

Suppose there are \( k \) flows over the two path \( P, B \), we index them by \( i = 1, 2, \ldots, k \).

- \( b_i \): the bandwidth of \( i \)th flow.
- \( p_i \): the selection decision of \( i \)th flow. If \( p_i = 0 \), then the \( i \)th flow is not selected to be rerouted. Otherwise, it is picked up to be moved back from \( B \) to \( P \).

Given the optimal control \( x_1^* \), our goal is to find out the maximum utilization upper bounded by \( x_1^* \). Mathematically this knapsack problem can be formulated as:

\[
\begin{align*}
\text{maximize} & \quad \sum_{i=1}^{k} b_i p_i \\
\text{subject to} & \quad \sum_{i=1}^{k} b_i p_i \leq x_1^* \quad p_i \in \{0, 1\}.
\end{align*}
\]  

(4.2)

4.4 Dynamic Programming Algorithm for the Model of Flows with Different Bandwidth

We see (4.2) is a utilization knapsack problem. Define \( J(i, b) \) to be the maximum value of the utilization that can be attained with the bandwidth no more than \( b \) using flows with index up to \( i \).

\[
J(i, b) = J(i - 1, b)1(b_i > b) + \max\{J(i - 1, b), J(i - 1, b - b_i) + b_i\}1(b_i \leq b)
\]  

(4.3)

Running the above algorithm with \( J(k, x_1^*) \), we are able to obtain the optimal selection set, in which the selected flows have the total traffic amount adding up to no more than \( x - 1^* \). If a table is used to store the
previous calculation result, it is easy to check this algorithm requires \( O(k \cdot x^*_1) \) time and \( O(k \cdot x^*_1) \) space, see Algorithm 4.2.

Algorithm 4.2 Calculate \( J(i, b) \) by recursions

```plaintext
if \( i > 0, b > 0 \) then
    for \( j = 1 \) to \( k \) do
        if \( b_j > b \) then
            res = \( J(i - 1, b) \)
        else
            res = max\{\( J(i - 1, b) \), \( b_j + J(i - 1, b - b_j) \)\}
        end if
    end for
else
    res = 0
end if
return res
```
Chapter 5: Numerical, Simulation and Experimental Results

In this chapter, we describe the simulations performed in OPNET and then show numerical and simulation results for the simple model where all the flows have the same unit bandwidth. These results show the relationship nature of the optimal control and the network parameters. Moreover, the results from Algorithm 4.1 are also the basis of the extended model where each flow has its own bandwidth. After that, we describe our testbed setup. Our testbed is built upon Sun VM VirtualBox virtual machine technology. A virtual network was built, acting as a real network. With this testbed, we run two experiments, one for the simple model with each flow having the same unit bandwidth, the other for the extended model with each flow having its own bandwidth. The numerical, simulation and experimental results are shown together.

5.1 Simulation Description

In this section, we describe our OPNET simulation. The research license was used. Though the academic version can be downloaded and used directly, it has many limitations. The main point is the routers have to be an exact router in reality. In the performance analysis of the optimization model, the router type is not specified and only standard route protocols are available. So we chose the research version.

5.1.1 Flow-based simulation

In the OPNET modeler simulation modes, we select the flow mode instead of packet mode. Flows are rerouted in case of link failure if the routing protocols react to the failure and recompute the routes. The following is a list of the issues that have to be taken into account to attain the results.

1. In OPNET, the routing protocols being used must detect the failure or recovery and react to it. So we disable the simulation efficiency mode for the routing protocol we used. We do this by going to the Global Attributes tab of the Configure Simulation dialog box, and setting the "RIP Sim Efficiency", "OSPF Sim Efficiency” etc. to "Disabled".
CHAPTER 5. NUMERICAL, SIMULATION AND EXPERIMENTAL RESULTS

2. In OPNET, flow-based simulation uses hybrid simulation efficiency techniques to increase simulation speed. When flows are used, there are no explicit packets being generated. Instead, 2 tracer packets are generated during the course of the simulation. If these packets do not get sent out during the time of failure, then the traffic is oblivious of the failure or recovery. Hence, the default 2 is too small to be able to guarantee failures or recovery to be detected. One way solving this problem is to configure explicit traffic using Application and Profile Configuration objects. However, in this thesis, we use flows and modify a few parameters as follows:

- We set 100 tracer packets during the simulation by setting "Tracer Packets Per Interval" attribute under "Traffic Characteristics" compound attribute on the flow globally using the Global Attribute of the same name (Tracer Packets Per Interval).
- In order to see which route (the primary path or the backup path) is taken during failure or reversion, we set these two routes to be recorded not once per flow (which is the default), but for all packets. This is being done by setting the attribute "Record Route Option" under "Traffic Characteristics" on the flow object.
- We configure the failure time to be 50-100 seconds.

3. While running the simulation, we get the link statistics, through selecting "Protocols → IP → Demands → Display Routes" for "Configured Demands", and the link utilization and the route during the period of the failure and reversion are attained.

5.1.2 Customized routing protocol

Since none of the existing implemented protocols in OPNET can be used for our simulation, we have to add our customized routing protocol. The key points of how to do so are as follows:

1. Register the Custom Protocol. We use the function int Ip_Cmn_Rte_Table_Custom_Rte_Protocol_Register (char* custom_rte_protocol_label_ptr) to register the customized routing protocol with IP. The returned integer is the routing protocol ID which is used in all calls to Ip_Cmn_Rte_Table API functions. Besides registering with IP, we also need register in the oms process, the attribute named protocol of the process handle is set to the same string used for registering with IP.
2. In the \textit{ip\_dispatch} process model, we add the name of the routing protocol to the list of symbol maps of the \textit{ip\_router\_parameters} \textit{\rightarrow} \textit{Interface Information} \textit{\rightarrow} \textit{Routing Protocols attribute}. This adds the custom routing protocol to the list of protocols that can be enabled on an interface.

3. Once receiving the start interrupt from IP, the routing protocol should identify the interfaces on which it is enabled. To do this, we borrow most of the code from \textit{rip\_init\_rt_table} function.

\subsection*{5.1.3 IP route table setup}

As shown in Fig. 5.1, there are four subnets with mask 192.168.0.x/24, 192.168.1.x/24, 192.168.2.x/24, 192.168.3.x/24 and 192.168.4.x/24. \(S\) is the source node and \(D\) is the destination node. \(A, B, C\) are routers. The direct link between \(A, B\) is considered as the primary path and the longer path through \(C\) is the backup path. All the configurations are the same as the model. The time interval between two successive flow arrivals is exponential. The service time of flows is also exponential. The link failure events consists of a Poisson process. The repair time is exponential [30].

\begin{center}
\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure51}
\caption{OPNET simulation network}
\end{figure}
\end{center}
We use the API functions `Ip_Cmn_Rte_Table_Entry_Add` and `Ip_Cmn_Rte_Table_Entry_Delete` to insert and remove routes into the route table. `ip_cmn_rte_table` is used to add backup routes. The ports are set by using `ip_rte_addr` to set `IpPortInfo` for the index and name of the interfaces. The simulation network topology is shown in Fig.5.1.

One problem that we met while setting up the backup route is multiple links between two nodes are automatically combined as one link. So we have to allow multiple next hops in our customized route protocol in order to setup the backup path. Our customized route protocol ID must be no less than IPC_INITIAL_CUSTOM_RTE_PROTOCOL_ID. Otherwise, the customized route protocol would be considered as a standard route protocol and the traffic rerouting in the customized route protocol does not work.

5.2 Numerical and Simulation Results

From all the plots in this section (Fig.5.2, Fig.5.3, Fig.5.4, Fig.5.5 and Fig.5.6), we see that when the number of tracker packets per flow is set to 100, the simulation results match the numerical results perfectly. That is because we use flow-based simulation which fits the model very well with large tracker packets.

Fig.5.2 shows the amount of rerouting and remaining flows in the two cases of the model of flows with the same unit bandwidth: i) \( c_2 > c_1 + \lambda_f c_m \) and ii) \( c_2 < c_1 + \lambda_f c_m \). From Fig.5.2, we see when \( c_2 > c_1 + \lambda_f c_m \), the threshold is \( x^*_1 = 10 \). When the total amount of flows \( x_1 + x_2 \) is less than or equal to \( x^*_1 \), we see that all the flows are rerouted back to the repaired primary path. When the total amount of flows \( x_1 + x_2 \) is greater than \( x^*_1 \), we rerouted 10 flows back to the primary path. When \( c_2 < c_1 + \lambda_f c_m \), no flows should be moved back to the primary, see the bottom of Fig.5.2.

Fig.5.3 shows the optimal amount of rerouting flows with various \( c_1 \). The parameters are configured as \( M = 100, \lambda = 1.0, \mu = 0.01, \lambda_f = \mu_f = 0.01, c_2 = 2.5, c_m = 10.0 \). In this configuration, the restoration rate setup seems to not make much sense in practice since it makes the average restoration time equal to the time interval between two failure events (usually, we have shorter restoration time). But later (Fig.5.5), we see the optimal threshold does not depend on the restoration time.

We vary \( c_1 \) from 0.75 to 2.0. Since after \( c_1 > 1.6 \), the calculation results are all 0, Fig.5.3 only shows \( c_1 \) from 0.75 to 1.35. From Fig.5.3, we see that the optimal rerouting amount of flows \( x^*_1 \) decreases as the \( c_1 \) increases. When \( c_1 \) is small, the calculated threshold is upper bounded by \( M \). All the flows are rerouted.
Figure 5.2: Optimal rerouting traffic $\vec{x}^\dagger$ with $M = 30$, $\lambda = 0.15$, $\mu = 0.01$, $\lambda_f = 0.1$, $\mu_f = 0.01$, $c_2 = 2.0$, $c_1 = 1.8$. 

Top: $c_2 > c_1 + \lambda_f c_m$, Bottom: $c_2 < c_1 + \lambda_f c_m$

to the restored primary path. When $c_1$ increases, the calculated threshold decreases to 0. Notice when the threshold is calculated to 0, the condition $c_2 > c_1 + \lambda_f c_m$ is still satisfied. This numerical result shows that under the condition $c_2 > c_1 + \lambda_f c_m$, even for optimal control of threshold type, it is possible that no flows are going to be rerouted (the same as the control under $c_2 < c_1 + \lambda_f c_m$). That is because although keeping all the flows on the backup path costs more than transferring them to the primary path, the difference of the costs between the flows being serviced on the two paths is too small to be worthy of even moving a single flow from the backup path back to the primary path. Thus, from the numerical calculation, we see after $c_1 > 1.46$, the threshold control exists but is equal to 0.
Fig. 5.4 shows the optimal amount of rerouting flows with various $c_2$ under the condition $c_2 > c_1 + \lambda_f c_m$. The parameters are configured as $M = 100, \lambda = 1.0, \mu = 0.01, \lambda_f = \mu_f = 0.01, c_1 = 1.0, c_m = 5.0$ and we vary $c_2$ from 1.5 to 2.5. From Fig.5.4, we see that the optimal rerouting amount of flows $x^*_1$ is an increasing function of $c_2$. When $c_2$ is small, though $c_2 > c_1 + \lambda_f c_m$, no flows are rerouted. Similar to the explanation of Fig.5.3, the reason is the benefit attained from the hold cost saving of servicing the flows on the primary path instead of on the backup path is less than the cost of moving a flow from the backup path to the primary path. As the $c_2$ increases, the calculated threshold also increases. When $c_2$ becomes greater, the threshold is equal to $M$, in which case all the flows are rerouted.

Fig. 5.5 shows the optimal amount of rerouting flows vs. log-scale $\lambda_f$ under the condition $c_2 > c_1 + \lambda_f c_m$. The parameters are configured as $M = 100, \lambda = 1.0, \mu = 0.01, \mu_f = 0.01, c_1 = 2.0, c_m = 1.0, c_2 = 2.52$ and we vary $\lambda_f$ from 0.001 to 0.5. From Fig.5.5, we see that the optimal rerouting amount of flows $x^*_1$ is a decreasing function of $\lambda_f$. When the failure event happens with a very low probability($\lambda_f$ is small), we do not move flows between the primary and backup paths many times. Thus, the cost caused by the move is small. We prefer moving all the flows from the backup path to the primary path after the primary path is restored. As the failure rate increases, the probability that the primary path works fine decreases. This makes
CHAPTER 5. NUMERICAL, SIMULATION AND EXPERIMENTAL RESULTS

Numerical results Simulation results

![Graph showing optimal rerouting amount of flows $x_1^*$ vs. $c_2$](image)

**Figure 5.4:** Optimal rerouting amount of flows $x_1^*$ vs. $c_2$

the moving cost caused by the failure events to increase. When it exceeds the benefit brought by transferring flows on the primary path instead of on the backup path, we do not reroute any flow back to the primary path.

Moreover, from Fig.5.5, we see that the optimal thresholds calculated through $\mu_f = 0.01$ overlaps the results of varying $\mu_f$. This result shows the restoration rate $\mu_f$ does not affect the moving decision attained from the Algorithm 4.1. From this result, we are free to set the restoration time (which makes the parameter setup $\mu_f = 0.01$ not matter) since our control does not depend on it.

Fig.5.6 shows the optimal amount of rerouting flows vs. log-scale $\lambda, \mu$ under the condition $c_2 > c_1 + \lambda_f c_m$. The parameters are configured as $M = 100, \lambda_f = 100.0, \lambda_f = 0.01, \mu_f = 0.01, c_1 = 1.85, c_m = 10.0, c_2 = 2.0$. We vary $\lambda, \mu$ from 0.1 to 100. From Fig.5.6, we see the optimal threshold also depends on the service time $\mu^{-1}$. When $\lambda$ and $\mu$ increase while keeping the offered load $\frac{\lambda}{\mu} = 100.0$ unchanged, the optimal threshold decreases. This result can be explained as follows. Though the average number of flows on the two paths does not change, the cost of servicing flows decreases due to the average service time $\mu^{-1}$ decreasing. So the value of $V(x_1, x_2, \mu_f)$ decreases (3.11) - (3.20). Thus, the weight of $V(x_1, x_2, 0)$ in the minimum goal $V(x_1, x_2, 0) + c_m x_1$ decreases. Because $V(x_1, x_2, 0)$ is a decreasing function of $x_1$ and $c_m x_1$ is an increasing function of $x_1$, it is easy to see that $x_1^*$ decreases.

5.3 Testbed Setup
As a verification of our model, we used Sun Virtual Machine (VM) VirtualBox to build a virtual network testbed. The host system is Ubuntu Long Time Stable version (LTS) 8.04. All the virtual machines (VM1, VM2, VM3, VM4, VM5) are installed on the host. To eliminate the unnecessary resource assumption, VM1 is installed Deli Linux. VM2 is using Puppy. TinyMe is installed in VM3. To make sure the backup path has higher cost, there are two hops VM4 and VM5 between the source VM1 and the destination VM2. VM4 and VM5 are running Windows XP and Windows 2008 respectively. The transfer protocol being used is User Datagram Protocol (UDP).
According to the virtual machine features, we compared two testbed setup schemes: 

i) The network of VM VirtualBox virtual machines is built through Network Address Translation (NAT).

ii) The network is built through bridged networking.

The advantage of using NAT is that the failure event is similar to what really happens in practice. In Linux, to save resources, a virtual machine only listens for a certain amount of time (in TinyMe, it follows an exponential distribution) for UDP data on a particular port. As a consequence, NetBios name resolution does not always work in Linux. We use this feature of NAT to generate the failure events on the primary path. Since this problem does not happen when we use Windows Internet Name Service (WINS), the backup path composed of VM4 (Windows xp) and VM5 (Windows 2008) always works fine.

With bridged networking, virtual machines are allowed to intercept data from the physical network and inject data into it. That is to say, bridged networking creates a new network interface in software. In summary, every virtual machine looks as though it had its own network card and were physically connected to the network through the host. Although using this mechanism allows us to set up a real routing table with preferred route (primary path $P$) and backup path $B$, there are two issues which makes us decide to not use it. One reason is the failure events have to be generated by our own code. In a simple simulation, we find the performance of the selected model is closely related to the distribution of the failure process. Generating failure events makes this experiment more like a simulation running on the testbed. The other reason relates to whether the use of a real routing table does not affect our results since they are pre-configured.

For the reasons stated above our experiments use the NAT scheme.

The logical topology of the network is shown in Fig.5.7. The source address is 10.45.0.1. The destination address is 10.45.0.2. The primary path $P$ is through 10.45.0.3. The backup path $B$ has two hops: 10.45.0.4 and 10.45.0.5. The flows are transferred from the source VM1 to the destination VM2 through $P$ (the brown path in Fig.5.7 when $P$ works or $B$ (the red path in Fig.5.7 when $P$ fails).

The virtual physical network is shown in Fig.5.8. We see that actually all the virtual machines are running on the host. We use iperf packet generator of Datagram Congestion Control Protocol (DCCP) [31] to generate the UDP traffic with constant bit rate (CBR). A modified tool ttcp (Test Transmission Control Protocol (TCP)) is used to send and receive constant bit rate (CBR) UDP flow from VM1 to VM3. To do so with
NAT, we used port forwarding. For example, as for a flow (connection), we forward the port 138 of VM1 to 5138 of the host. Sending data out through 138 of VM1 is actually sending data on the port 5138 of the host. At VM1, we use its numeric IP address 10.45.0.1 to avoid the NAT failure. We implemented a socket forwarding program (a proxy program) at VM3 to forward all the packets received at the port 5138 of the host to 138 of VM2, which is forwarded to 5238 of the host. This proxy program at VM3 logs the traffic so we can calculate the time average of the buffer usage, which is used as the cost rate of the primary path. In this example, we forward the port 138 of VM3 to the host 6138. Thus, the packet forwarding service looks as though sending data from 138 of VM3 but actually from 6138 of the host to 138 of VM2 (actually 5238 of the host).

When the listening time of NetBios service in VM3 expires, the name resolution of VM3 is no longer available. Thus, the transmission fails due to UDP proxy failure. The receiver of `ttcp` at VM2 generates an error message. This message triggers the same proxy program running at VM4 to start to listen 5138 of the host. Then, VM4 forwards the UDP packets to VM5 and VM5 forwards them to VM2. This process is controlled by the main program running at the host. In this way, all the flows are moved to the backup path $B$. The cost rate on the path $B$ is calculated as the time average of the total buffer usage at VM4 and VM5.
While moving flows, the main program running at the host reboots VM3 through Sun VM VirtualBox command. After all the network services of VM3 successfully start (after checking the VM3’s network status through VM VirtualBox), the proxy program at VM3 starts to listen the sender of ttcp at VM1 at 5137 of the host. At the same time, VM4 also listens to this port. The main program implementing Algorithm 4.1 and 4.2 and running at the host determines the packets of which connections would be forwarded through VM3 or VM4. The cost of moving flows is calculated by the usage change of Java virtual machine heap.

The Ethernet card is e1000. Since DCCP is a datagram-based protocol, the original iperf and ttcp created by M. Muuss and T. Slattery treats DCCP as if it were TCP, by stuffing as many bytes into the socket as possible. We changed the VirtualBox receive and send buffer sizes from the default 64KB to 1024KB, the maximum stack size, to avoid dropping packets. Though in the virtual machine, the capacity of the network cable is ideally infinite, it is actually upper bounded by the communication threads capacity, especially when there are 5 virtual machines running at the same time. The socket forwarding program and optimization control program are implemented in Java. Unix bash scripts are used to attain the information and control Sun VM VirtualBox virtual machines.

### 5.4 Experimental Results

In this section, we implement two experiments: one for the simpler model where all the flows have the same unit bandwidth, the other for the extended model where the flows have different bandwidth.

#### 5.4.1 All the flows have the same unit bandwidth
To verify the simple model where all the flows have the same unit bandwidth, we use *iperf* traffic generator to generate UDP traffic [31]. For each traffic flow, the modified *ttcp* client [31] at VM1 sends a 1024kbps CBR UDP flow. Although the minimum of the maximum packet size of *TinyMe, XP, 2008* is 8164 bytes (excluding 20 bytes of IP header and 8 of UDP header), due to the limitation of Sun VM VirtualBox NAT, the packet size is set to be 1472 (excluding 28 IP and UDP headers). The maximum number of flows (connections) is 100. As we mentioned before, the time averages of the total buffer usage on the primary path $P$ and on the backup path $B$ are respectively counted as $c_1x_1$ and $c_2x_2$. We measure the total runtime memory usage increment of Java proxy programs caused by switching $x$ flows between the two paths by summing the three Java packet forwarding object heaps on both paths as $c_mx$ [32]. All of these are measured as event driven, which means the measurement is logged whenever there is an event (arrival, departure, failure, etc).

![Figure 5.9: Optimal rerouting traffic $x^*$ with $M$ from 50 to 100, $\lambda = 0.1$, $\mu = 0.01$, $\bar{\lambda}_f = 0.01019$, $\bar{\mu}_f = 0.0195$, $\bar{c}_2 = 224.01k$, $\bar{c}_1 = 192.036k$, $\bar{c}_m = 1046.8k$.](image)

We ran the experiment 51 times. Each time we allow the maximum number of connections to be a constant from 50 to 100. The time interval between two successive flows arrival is exponential with mean 10 seconds. The lifetime of each flow generated is an exponential random variable with mean 100 seconds. This is done to guarantee that no connections are rejected because of exceeding $M$. The UDP receiving failure event is measured as the primary path failure. According to *TinyMe* developers’ statement, the *TinyMe* has
Table 5.1: Measured Moving Cost and Holding Cost (bit/s)

<table>
<thead>
<tr>
<th>M</th>
<th>Real Number of flows</th>
<th>Buffer Usage Increment of JVM Heap</th>
<th>Buffer Usage on P</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>12</td>
<td>68401243</td>
<td>9585845</td>
</tr>
<tr>
<td>59</td>
<td>11</td>
<td>69504585</td>
<td>10585876</td>
</tr>
<tr>
<td>61</td>
<td>15</td>
<td>60027699</td>
<td>11012112</td>
</tr>
<tr>
<td>63</td>
<td>17</td>
<td>85753856</td>
<td>15731589</td>
</tr>
<tr>
<td>69</td>
<td>14</td>
<td>94329242</td>
<td>17304748</td>
</tr>
<tr>
<td>72</td>
<td>10</td>
<td>77178470</td>
<td>14158430</td>
</tr>
<tr>
<td>75</td>
<td>11</td>
<td>102904627</td>
<td>18877907</td>
</tr>
<tr>
<td>81</td>
<td>10</td>
<td>85753856</td>
<td>15731589</td>
</tr>
<tr>
<td>82</td>
<td>10</td>
<td>60027699</td>
<td>11012112</td>
</tr>
<tr>
<td>87</td>
<td>11</td>
<td>94329242</td>
<td>17304748</td>
</tr>
<tr>
<td>91</td>
<td>10</td>
<td>68603085</td>
<td>12585271</td>
</tr>
</tbody>
</table>

an exponential distribution time when listening to UDP port (means the time interval between failures is exponential). In our experiment, since the delay of message sending and processing is ignored, the measured interval does not strictly have an exponential distribution. We stop the experiment when we have received at least 20 failure events. TinyMe’s rebooting time is measured as the primary path repair time. Though it is claimed that TinyMe’s booting time is less than 30 seconds, in our experiment, the measured average booting time is between 35 to 40, which could be caused by the virtual machine’s nature of sharing processor and memory. In addition to the shutdown and hold-off time, the measured average rebooting time (repairing time) of TinyMe is 51.4 seconds. The average memory usage by forwarding UDP packets of a flow on VM3 is 1573158.912 bits or 192.036 k bytes, The total average memory usage by forwarding UDP packets of a flow on VM4 and VM5 is 1835089.92 bits or 224.01 k bytes. The average memory usage increment of Java proxy programs heap of a flow is 1046.8 k bytes. The average time interval between two failures is 98.1354 seconds.

Table 5.1 shows some of our measurements where 9 flows are moved from the backup path B back to the primary path P after P is repaired. From Table 5.1, we see that in different experiments where all parameter configurations are the same except that the allowed maximum number of flows are different, the one-time moving cost and the holding cost rates varies in every experiment. However, from Fig. 5.9, we see that the average optimal control over all these experiments (M does not affect any results as long as it is large enough.) is approximately equal to the numerical results. Thus, for limited computing resources, we can make the calculation off-line instead of “on the fly” if we know these parameters or their average values. The off-line calculation can give us a nearly optimal solution even though not as good as the “on the fly” computation.
Fig. 5.10 shows the optimal amount of rerouting flows vs. log-scale \( \lambda \) under the condition \( c_2 > c_1 + \lambda f e_m \). The parameters are configured as \( M = 100, \mu = 0.01, \bar{\lambda}_f = 0.01019, \bar{\mu}_f = 0.0195, c_2 = 2, c_1 = 1.5, c_m = 10 \). We vary \( \lambda \) from 0.1 to 100. From Fig. 5.10, the optimal threshold increases with \( \lambda \). When \( \lambda \) is small, the number of flows being serviced seen at the time instant of a new flow arrival is very small. As \( \lambda \) increases, more traffic is transferred to the backup path, the cost of running flows on the backup path increases. Thus, the optimal threshold increases until it is equal to the capacity \( M \).

5.4.2 Flows may have different bandwidth

In this subsection, we run two experiments. The capacity limitation of the traffic on both paths is 100 Mbytes. In the first experiment, we generate up to 50 “flows”. To generate different bandwidth, we use a continuous uniform distribution within \([0.001 MB, 1.999 MB]\) of mean 1Mbps and floor function to get an integer (in kbytes unit) bandwidth. In the second experiment, we generate up to 10 “flows”. For both experiments, \( \mu = 0.01, \bar{\lambda}_f = 0.0101, \bar{\mu}_f = 0.0205, c_2 = 2, c_1 = 1.5, c_m = 10 \). Again we use a continuous uniform distribution within \([0.001 MB, 9.999 MB]\) of mean 5Mbps and floor function to get an integer (in kbytes unit) bandwidth. We use these two experiments to compare the performance of Algorithms 4.2. Their
performance is shown as in Fig. 5.11.

From Fig. 5.11, we see that as the traffic arrival rate $\lambda$ increases, the long run time average optimal cost increases. That is because more and more flows are being serviced in the system. It is also easy to see that our first experiment (50 flows) result has greater optimal cost than the numerical result. That is because running Algorithm 4.2 is suboptimal if a flow is not allowed to be split to be transferred to two paths. We also see that in the second experiment (20 flows), the optimal cost is worst. That is because when the bandwidth of flows are greatly different (from 1KB to 5MB), the optimization attained from Algorithm 4.2 “wastes” some available traffic amount calculated by Algorithm 4.1 due to the nature of flows (non-splitable). Though this seems to bring greater cost than complete traffic splitting, it avoids most packet reordering work at the receiver end, which is a high cost operation.
Chapter 6: Conclusion and Future Work

6.1 Conclusion

In this thesis, we investigate an optimization model of traffic rerouting in the reversion (or rerouting) cycle of network recovery process. Due to the wide variety of network recovery mechanisms, there is no simple model that is able to cover all mechanisms. The model that we investigate covers a family of network recovery schemes which has the features: fast recovery, pre-configured backup path, event driven global rerouting, flow/connection-based cost optimization. Our contribution includes:

- We used dynamic programming algorithms to implement a flow-based simulation with OPNET modeler. Through the simulation results, as long as the number of tracker packets per flow per second are large enough, the OPNET simulation attains the same optimal control as the theoretical analysis.
- We built a testbed with Sun VM VirtualBox Networking and verify the numerical calculations.

6.2 Limitations and Future Work

However, this model itself and the simulation still have some limitations that could be addressed in the future.

- Exponential distribution assumptions: The traffic arrival process is known not to be a Poisson process (even not a self-similar process). The failure event time interval may not be exponential, either. All these assumptions are the basis of the selected Markov Decision Process Model.
- The same service time assumption for a flow on the two paths: In the selected model, we use the memory usage as the cost and assume a flow’s initially requested service time does not change when the path is changed. This is true in some scenarios, but not always true. In other cases, the backup path
may have a longer delay than the primary path. In such cases, the cost of switching traffic between two paths may be largely affected by packet reordering.

- Linear cost assumption: For the holding cost and one-time moving cost, we assume that they are a linear function of the number of flows (or the traffic amount). But actually during our experiment, we find that it is possible that they are not linear functions, especially when flows have different bandwidth. However, in our virtual machine experiment, moving a flow does not cost the same as actually moving a flow as in a real network.

- Simulation only implements the flows with fixed bandwidth. It would be better if the simulation could include flows with random bandwidth.

- A simple network topology limitation: A large network topology in OPNET would greatly extend the simulation results which may not be easy to be implemented by the testbed.

- Testbed limitations: Though there are many benefits from using virtual machines to setup our testbed, one side effect of using this technology is the scalability. After running 5 virtual machines, the system resources assigned to each virtual machine is very limited. This limited the number of flows in our experiment.

- Experiment Limitations: Other limitations of the maximum number of flows (or connections) is the UDP flows that iperf generates. When the number of flows exceed a certain number (between 100-200), all the memory it uses is recollected. This results in an automatic exit of iperf. Thus, in our experiment, we do not test the case where there are thousands of flows.
Bibliography


