5-2010

Diversifying The Internet

Yong Liao
University of Massachusetts Amherst

Follow this and additional works at: https://scholarworks.umass.edu/open_access_dissertations

Part of the Electrical and Computer Engineering Commons

Recommended Citation
https://scholarworks.umass.edu/open_access_dissertations/210

This Open Access Dissertation is brought to you for free and open access by ScholarWorks@UMass Amherst. It has been accepted for inclusion in Open Access Dissertations by an authorized administrator of ScholarWorks@UMass Amherst. For more information, please contact scholarworks@library.umass.edu.
DIVERSIFYING THE INTERNET

A Dissertation Presented
by
YONG LIAO

Submitted to the Graduate School of the University of Massachusetts Amherst in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2010

Electrical & Computer Engineering
DIVERSIFYING THE INTERNET

A Dissertation Presented

by

YONG LIAO

Approved as to style and content by:

__________________________
Lixin Gao, Chair

__________________________
Weibo Gong, Member

__________________________
Aura Ganz, Member

__________________________
Arun Venkataramani, Member

__________________________
Christopher V. Hollot, Department Chair
Electrical & Computer Engineering
ABSTRACT

DIVERSIFYING THE INTERNET

MAY 2010

YONG LIAO
B.Sc., UNIVERSITY OF SCIENCES AND TECHNOLOGY OF CHINA
M.Sc., GRADUATE SCHOOL OF CHINESE ACADEMY OF SCIENCES
Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Lixin Gao

Diversity is a widely existing and much desired property in many networking systems. This dissertation studies diversity problems in Internet, which is the largest computer networking system in the world. The motivations of diversifying the Internet are two-fold. First, diversifying the Internet improves the Internet routing robustness and reliability. Most problems we have encountered in our daily use of Internet, such as service interruptions and service quality degradation, are rooted in the inter-domain routing system of Internet. Inter-domain routing is policy-based routing, where policies are often based on commercial agreements between ASes. Although people know how to safely accommodate a few commercial agreements in inter-domain routing, for a large set of diverse commercial agreements, it is not clear yet what policy guidelines can accommodate them and guarantee convergence. Accommodating diverse commercial agreements not only is needed for ASes in Internet to achieve their business goals, it also provides more path diversity in inter-domain
routing, which potentially benefits the inter-domain routing system. However, more reliable and robust routing cannot be achieve unless the routing system exploits the path diversity well. However, that is not the case for the current inter-domain routing system. There exist many paths in the underlying network, but the routing system cannot find those paths promptly. Although many schemes have been proposed to address the routing reliability problem, they often add significant more complexity into the system. The need for a more reliable inter-domain routing system without adding too much complexity calls for designing practical schemes to better exploit Internet path diversity and provide more reliable routing service.

The increasing demands of providing value-added services in Internet also motivates the research work in this dissertation. Recently, network virtualization substrates and data centers are becoming important infrastructures. Network virtualization provides the ability to run multiple concurrent virtual networks in the same shared substrate. To better facilitate building application-specific networks so as to test and deploy network innovations for future Internet, a network virtualization platform must provide both high-degree of flexibility and high-speed packet forwarding in virtual networks. However, flexibility and forwarding performance are often tightly coupled issues in system design. Usually we have to sacrifice one in order to improve the other one. The lack of a platform that has both flexibility and good forwarding performance motivates the research in this dissertation to design network virtualization platforms to better support virtual networks with diverse functionalities in future Internet. The popularity of data centers in Internet also motivates this dissertation to studying scalable and cost-efficient data center networks. Data centers with a cluster of servers are already common places in Internet to host large scale networking applications, which require huge amount of computation and storage resources. To keep up with the performance requirements of those applications, a data center has to accommodate a large number of servers. As Internet evolves
and more diverse applications emerge, the computation and storage requirements for data centers grow quickly. However, using the conventional interconnection structure is hard to scale the number of servers in data centers. Hence, it is of importance to design new interconnection structures for future data centers in Internet.

Four interesting topics are explored in this dissertation: (i) accommodating diverse commercial agreements in inter-domain routing, (ii) exploiting the Internet AS-level path diversity, (iii) supporting diverse network data planes, and (iv) diverse interconnection networks for data centers. The first part of this dissertation explores accommodating diverse commercial agreements in inter-domain routing while guaranteeing global routing convergence, so as to provide more path diversity in Internet. The second part of this dissertation studies exploiting the path diversity in Internet by running multiple routing processes in parallel, which compute multiple paths and those paths can complement each other in case one path has problems when dynamics present in the routing system. The third part of this dissertation studies supporting concurrent networks with heterogeneous data plane functions via network virtualization. Two virtual network platforms are presented, which achieve both high-speed packet forwarding in each virtual network and high degree of flexibility for each virtual network to customize its data plane functions. The last part of this dissertation presents a new scalable interconnection structure for data center networks. The salient feature of this new interconnection structure is that it expands to any number of servers without requiring to physically upgrading the existing servers.
TABLE OF CONTENTS

Page

ABSTRACT ......................................................... iv

LIST OF TABLES .................................................. xiv

LIST OF FIGURES ................................................ xvi

CHAPTER

1. INTRODUCTION ................................................. 1

1.1 Internet Overview ............................................. 1

   1.1.1 Commercial Agreements and Internet Topological Structure .... 2
   1.1.2 Internet Routing and Inter-domain Routing Policies ............. 3
   1.1.3 Internet Data Plane ........................................ 5

1.2 Motivations of Diversifying the Internet .......................... 6

   1.2.1 Internet Routing Convergence and Reliability ................. 6
   1.2.2 Providing Value-added Service in Internet ..................... 7

1.3 Dissertation Overview ........................................ 9

   1.3.1 Diverse Commercial Agreements .............................. 9
   1.3.2 Inter-domain Path Diversity ................................. 10
   1.3.3 Network Virtualization Platforms ............................ 11
   1.3.4 Data Center Networks ...................................... 12

1.4 Dissertation Outline ......................................... 13

2. SAFE INTER-DOMAIN ROUTING UNDER DIVERSE COMMERCIAL AGREEMENTS .......... 14

2.1 Introduction ................................................. 14

2.2 Background, Motivation and Overview ........................... 16
2.2.1 Routing Policies, Routing Safety and Robustness ........................................ 16
2.2.2 Practical Routing Policy Guidelines Accommodating Transit and Peering Agreements ....................................................... 17
2.2.3 Diverse Commercial Agreements ................................................................. 18
2.2.4 Accommodating Mutual Transit Agreements: An Overview .................. 20

2.3 Admissible Paths for Accommodating Mutual Transit Agreement ........ 21
2.3.1 AS Graph Model ......................................................................................... 21
2.3.2 AS Paths with Steps ................................................................................. 21
2.3.3 Admissible Path Set .................................................................................. 22
   2.3.3.1 Not allowing valley paths .................................................................. 22
   2.3.3.2 Allowing valley-free paths with steps ............................................... 23

2.4 Classes of Paths and Ranking of the Paths ............................................... 25
2.4.1 Classes of Paths in the Admissible Path Set ............................................. 25
2.4.2 Ranking the Paths ..................................................................................... 27
   2.4.2.1 Ranking between customer paths and MTran paths .......................... 27
   2.4.2.2 Ranking between provider paths and MTran paths .......................... 28
   2.4.2.3 Ranking between peer paths and MTran paths ............................... 30
   2.4.2.4 Ranking among MTran paths .......................................................... 31

2.5 Policy Guidelines for Accommodating Mutual Transit Agreements .... 32
2.5.1 The Policy Guidelines ............................................................................ 32
   2.5.1.1 The 1-step policy ............................................................................ 32
   2.5.1.2 The k-step policy ........................................................................... 33
   2.5.1.3 The any-step policy ...................................................................... 34
2.5.2 Safety and Robustness of the Policy Guidelines ................................. 34

2.6 Practical Implications .................................................................................. 37
2.6.1 Realizing the Policy Guidelines in BGP .................................................. 37
2.6.2 Safely Establishing Mutual Transit Agreements .................................... 38
2.6.3 Handling Selective Mutual Transit Agreements .................................... 39

2.7 Experimental Evaluation .......................................................................... 40
2.7.1 Potential Peering Agreements that Can Be Safely Converted to Mutual Transit Agreements .................................................. 41
2.7.2 Improving Fault Tolerance by Extending Peering into Mutual Transit ........................................ 42

2.8 Conclusion .................................................. 45

2.9 Appendix .................................................... 45

2.9.1 Dispute Wheel ................................................ 45

2.9.2 The Safeness and Robustness of the k-step Policy Guideline .................................................... 45

3. RELIABLE INTER-DOMAIN ROUTING THROUGH MULTIPLE COMPLEMENTARY ROUTING PROCESSES ............................................................ 50

3.1 Introduction .................................................. 50

3.2 Background of Inter-domain Routing and a Review of Related Work ........................................ 53

3.2.1 Transient Routing Problems in Inter-domain Routing ......................................................... 53

3.2.2 Related Work Addressing the Unreliable Problem of BGP and Their Limitations ........................................ 55

3.3 Multi-Process Routing: Basic Design Principles ............................................................ 57

3.3.1 Routing Events ............................................... 58

3.3.2 Multiple Routing Processes ..................................... 59

3.3.3 Downhill Node Disjointness ....................................... 59

3.3.4 From Control Plane to Data Plane ..................................... 65

3.4 A multi-process routing protocol ........................................... 66

3.4.1 The STAMP Protocol ........................................... 66

3.4.2 Properties of STAMP ........................................... 69

3.5 Data plane design ........................................... 75

3.5.1 Packet Forwarding ........................................... 75

3.5.2 Switching between Routing Processes ..................................... 76

3.5.3 Effectiveness of the Switching Mechanism ..................................... 78

3.6 Experimental evaluation ........................................... 80

3.6.1 Evaluating STAMP’s Performance ..................................... 80

3.6.2 Performance under Failure–Comparison to Other Schemes ........................................ 83

3.6.3 Evaluating Partial Deployment ..................................... 85

3.7 Conclusion .................................................... 87
4. PDP: PARALLELIZING DATA PLANE IN VIRTUAL NETWORK SUBSTRATE

4.1 Introduction .......................................................... 88
4.2 Related Work: Virtual Network Platform State of Art ............... 90
  4.2.1 VINI ................................................................. 90
  4.2.2 Trellis ............................................................. 91
  4.2.3 VRouter ........................................................... 91
  4.2.4 Supercharging PlanetLab ....................................... 92
  4.2.5 Source Code Merging ............................................ 93
  4.2.6 Summary .......................................................... 93
4.3 The Design of PdP ................................................... 95
  4.3.1 Basic Ideas ....................................................... 95
  4.3.2 PdP Node Architecture ........................................ 96
  4.3.3 The Management Host and The Multiplexer/Demultiplexer ... 97
  4.3.4 The Forwarding Engine ....................................... 99
  4.3.5 A PdP Node Prototype ........................................ 101
4.4 Experiment Evaluation .............................................. 102
  4.4.1 Experiment Setting ............................................. 103
  4.4.2 Packet Forwarding Speed ..................................... 103
  4.4.3 Forwarding with Large Routing Table ....................... 106
  4.4.4 TCP Throughput ............................................... 107
  4.4.5 Packet Out-of-Order in TCP .................................. 108
  4.4.6 Transmission Delay ........................................... 109
4.5 Conclusion .......................................................... 110

5. EUROPA: EFFICIENT USER MODE PACKET FORWARDING FOR NETWORK VIRTUALIZATION .... 111

5.1 Introduction .......................................................... 111
5.2 Packet Forwarding in Software Routers ............................ 113
  5.2.1 Kernel Mode Forwarding ..................................... 113
  5.2.2 Conventional User Mode Forwarding .......................... 114
  5.2.3 Identify the Causes of Slow User Mode Forwarding ......... 115
5.3 Efficient User Mode Packet Forwarding ............................ 117
  5.3.1 Overview .......................................................... 118
    5.3.1.1 Directly controlling the devices .......................... 118
5.3.1.2 Sharing memory between KM and UM .......................... 118
5.3.1.3 Avoiding expensive system calls ......................... 119

5.3.2 The Design of Efficient User Mode Packet Forwarding .... 119
5.3.2.1 The packet pool ........................................ 120
5.3.2.2 The rxRing and txRing ................................. 121

5.3.3 A Complete Example ........................................ 122

5.4 EUROPA: Flexible and High-performance Network
Virtualization .............................................. 124
5.4.1 EUROPA Architecture ........................................ 124
5.4.2 Virtual Routers ............................................. 125
5.4.3 Packet Forwarding in Virtual Networks Hosted by
EUROPA ..................................................... 125
5.4.3.1 Design of EUROPAKM .................................... 126
5.4.3.2 Design of EUROPAUM .................................... 127

5.4.4 Discussion .................................................. 128
5.4.4.1 Security and Isolation ..................................... 128
5.4.4.2 Overhead of packet classification ...................... 129
5.4.4.3 Overhead of polling packet state ..................... 129

5.5 Experimental Evaluation ....................................... 130
5.5.1 Experiment Setting ........................................... 130
5.5.2 UDP Experiments ............................................ 131
5.5.2.1 Single virtual router .................................... 131
5.5.2.2 Multiple virtual routers ............................... 133

5.5.3 TCP Experiments ............................................. 134
5.5.3.1 Single virtual router .................................... 134
5.5.3.2 Multiple virtual routers ............................... 135

5.5.4 Forwarding Performance and Packet Pool Size ........ 135

5.6 Conclusion .................................................... 136

6. DPILLAR: SCALABLE DUAL-PORT SERVER
INTERCONNECTION FOR DATA CENTER
NETWORKS ....................................................... 138
6.1 Introduction ................................................................. 138
6.2 Background and Related Work ........................................ 140
  6.2.1 Interconnection Networks in Data Centers ...................... 140
  6.2.2 Related Work in Data Center Networks ......................... 141
6.3 Interconnection of DPillar ............................................ 143
  6.3.1 Network Structure of DPillar .................................... 143
  6.3.2 Topological Properties of DPillar ................................ 145
    6.3.2.1 Number of servers DPillar accommodates .................. 146
    6.3.2.2 Number of switches used by DPillar ....................... 146
    6.3.2.3 Bisection width of DPillar ................................. 147
  6.3.3 Cost of Building DPillar ........................................ 149
6.4 Routing in DPillar Network ......................................... 150
  6.4.1 Routing Algorithm ................................................ 150
  6.4.2 Overhead of Routing and Forwarding ............................ 153
  6.4.3 Longest Path in DPillar ......................................... 153
  6.4.4 Traffic Distribution in All-to-All Communication ............. 154
6.5 Handling Failure and Balancing Load in DPillar ................. 155
  6.5.1 Discovering Failures ............................................. 155
  6.5.2 Bypassing Failures ............................................... 155
    6.5.2.1 The ring phase ............................................. 156
    6.5.2.2 The helix phase ............................................. 156
  6.5.3 The Fault-tolerant Routing Algorithm ......................... 160
  6.5.4 Traffic-aware Routing in DPillar Networks .................... 160
6.6 Evaluations ............................................................. 162
  6.6.1 Implementing DPillar Routing & Measuring the Performance .... 162
    6.6.1.1 Implementation ............................................. 162
    6.6.1.2 Performance measurement .................................... 163
  6.6.2 Simulation Evaluations .......................................... 165
    6.6.2.1 Average Path Length ....................................... 165
    6.6.2.2 Average Path Length in Failure-tolerant Routing ........ 166
6.7 Conclusion ...................................................... 167

7. CONCLUSION ................................................... 168

BIBLIOGRAPHY ................................................... 171
<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Number of disconnected AS pairs under tier-1 de-peering</td>
</tr>
<tr>
<td>2.2</td>
<td>Number of disconnected AS pairs under tier-1 AS partition</td>
</tr>
<tr>
<td>3.1</td>
<td>Number of destinations without downhill node disjoint paths</td>
</tr>
<tr>
<td>3.2</td>
<td>Number of ASes with max($\Phi_k$)=1</td>
</tr>
<tr>
<td>3.3</td>
<td>Number of ASes having transient problems in single link failure</td>
</tr>
<tr>
<td>3.4</td>
<td>Number of ASes having transient problems in multiple link failures</td>
</tr>
<tr>
<td>3.5</td>
<td>Number of ASes having transient problems in single node failure</td>
</tr>
<tr>
<td>4.1</td>
<td>Percentages of out-of-order packets when the PdP node has one, two, and three FE guest machines. The classifier in the MH distributes packets to FE guest machines in a round robin manner</td>
</tr>
<tr>
<td>4.2</td>
<td>Packet out-of-order when the classifier uses different strategies to distribute packets to FE guest machines</td>
</tr>
<tr>
<td>4.3</td>
<td>The round trip time experiment results</td>
</tr>
<tr>
<td>5.1</td>
<td>CPU cycles consumed in invoking system calls</td>
</tr>
<tr>
<td>5.2</td>
<td>Overhead of copying packets between user space and kernel</td>
</tr>
<tr>
<td>5.3</td>
<td>A slot in the packet pool can have five states. This table shows the meaning of each state</td>
</tr>
<tr>
<td>6.1</td>
<td>Comparison between different data center interconnection networks. Parameter $n$ is the number of ports each switch has; $N$ represents the total number of servers; $U_s$ is the unit price of a $n$-port switch; $D$ is the network diameter. For DCell, FiConn, and BCube, $l$ is the number of recursively construction level. For DPillar, $k$ is the number of server columns</td>
</tr>
</tbody>
</table>
6.2 The cost of the networking devices, including switches and cables, when using switches with different number of ports to build DPillar networks with four columns of servers. ................. 149
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Example of sibling relation established between merging ASes.</td>
<td>19</td>
</tr>
<tr>
<td>2.2</td>
<td>Uphill/downhill paths and uphill/downhill paths with steps. The dashed lines represent the AS paths. (a) is an uphill path; (b) is an uphill path with step; (c) is a downhill path; (d) is a downhill paths with step.</td>
<td>22</td>
</tr>
<tr>
<td>2.3</td>
<td>Examples of valley paths, with and without steps. In (a) and (b), an AS transits traffic for its two providers; in (c) and (d), ASes with mutual transit agreements transit traffic for their respective providers; in (e) and (f), two peering ASes transit traffic for their respective providers. Allowing valley paths does not make economic sense in general.</td>
<td>23</td>
</tr>
<tr>
<td>2.4</td>
<td>Examples of admissible paths with steps in $P_1$. The dashed lines are the AS paths. Note that steps do not have to appear in those six kinds of admissible paths in $P_1$.</td>
<td>24</td>
</tr>
<tr>
<td>2.5</td>
<td>Examples of $uMTran$ paths and $dMTran$ paths. In (a)<del>(c), AS $a$ has $dMTran$ path to $d$. In (d)</del>(f), AS $a$ has $uMTran$ path to $d$.</td>
<td>26</td>
</tr>
<tr>
<td>2.6</td>
<td>Policy dispute if $dMTran$ path $\succ$ customer path.</td>
<td>28</td>
</tr>
<tr>
<td>2.7</td>
<td>Policy dispute if $uMTran$ path $\succ$ customer path.</td>
<td>28</td>
</tr>
<tr>
<td>2.8</td>
<td>Policy dispute if $uMTran$ path $\succ$ provider path.</td>
<td>29</td>
</tr>
<tr>
<td>2.9</td>
<td>Policy dispute if provider path $\succ$ $dMTran$ path.</td>
<td>29</td>
</tr>
<tr>
<td>2.10</td>
<td>Policy dispute if peer path $\succ$ $dMTran$ path.</td>
<td>30</td>
</tr>
<tr>
<td>2.11</td>
<td>Policy dispute if $uMTran$ path $\succ$ peer path.</td>
<td>30</td>
</tr>
<tr>
<td>2.12</td>
<td>Policy dispute can arise if $dMTran_j \succ dMTran_i$ where $j &gt; i$.</td>
<td>31</td>
</tr>
</tbody>
</table>
2.13 Policy dispute can arise if $\text{uMTran}_j \succ \text{uMTran}_i$, where $j > i$. 

2.14 Examples of $s_0\text{Cycle}$ and $s_1\text{Cycle}$. 

2.15 Per-neighbor based selective mutual transit agreement. 

2.16 Percentage of peering links which can be safely expanded into mutual transit agreements for $k$ ranges from 1 to 6. 

2.17 Number of disconnected AS pairs in case of access link failures when peering ASes are not willing to do mutual transit. 

2.18 A dispute wheel $W = (\mathcal{U}, \mathcal{Q}, \mathcal{R})$ of size $m$. 

3.1 An example to show transient routing problems of BGP. 

3.2 Complementary paths to the same destination. 

3.3 The CDF of $\Phi_k$. 

3.4 Percentage of non-tier-1 ASes to which each tier-1 AS have two downhill node disjoint paths under incremental deployment of STAMP (at tier-1 ASes only). 

4.1 Example of a PdP node. The dashed arrows represent incoming, unprocessed packets. The solid arrows represent outgoing, processed packets. 

4.2 The structure of a management host. 

4.3 The structure of a forwarding engine. 

4.4 $vnet_1 \sim vnet_3$ get equal processing power in (a) and (b). In (a) $vnet_2$ and $vnet_3$ has two FE guest machines. In (b) $vnet_2$ (and $vnet_3$) has one FE guest machine only. 

4.5 A PdP node prototype. 

4.6 The experiment testbed. 

4.7 Packet forwarding speed and packet loss rate in UDP traffic experiment.
4.8 Packet forwarding speed when the packet size is 512 bytes or 1500 bytes. The input speed for each experiment is set to saturate the 1 Gbps link. ......................................................... 105

4.9 Packet forwarding speed in case of large routing table. (a) plots the peak packet forwarding speed. For (b) and (c), the packet input speed is set to saturate the 1 Gbit Ethernet link. .................. 106

4.10 The TCP throughput experiment results. ................................. 107

5.1 Packet forwarding in kernel mode Click software router. .............. 114

5.2 The conventional packet forwarding in user mode Click ............... 115

5.3 Slot state transition diagram. ................................................. 121

5.4 rxRing and txRing structure. ................................................. 122

5.5 Steps of forwarding a packet. ................................................. 123

5.6 The architecture of EUROPA .................................................. 124

5.7 Diagram of EUROPAKM ....................................................... 126

5.8 Diagram of EUROPAUM ....................................................... 127

5.9 Experiment testbed. .............................................................. 131

5.10 UDP packet forwarding speed. ............................................. 132

5.11 UDP packet forwarding speed vs. number of virtual routers. ......... 133

5.12 TCP throughput .............................................................. 134

5.13 TCP throughput vs. the number of virtual routers. ....................... 135

5.14 UDP and TCP performance vs. packet pool size. ....................... 136

6.1 The vertical view of a DPillar network. .................................... 144

6.2 A DPillar network in two-dimension. The number in each circle is the label of that server. ......................................................... 145

6.3 A (4, 3) DPillar network example. ......................................... 156
6.4 Configure the IP addresses of each host according to the host column and label information. ................................. 162

6.5 The testbed network. Server $P$ forwards packets between $A$ and $B$. ......................................................... 163

6.6 The number of CPU cycles one table lookup operation consumes in conventional IP forwarding. ............................. 164

6.7 The server CPU usage under various traffic load. The packet size is 1024 bytes. .................................................. 164

6.8 Comparison of DPillar routing and shortest path routing. .............. 166

6.9 Path length in fault-tolerant routing. ........................................ 166
Diversity is often a much desired property in many real world systems, such as ecosystems and social systems. This dissertation studies the diversity problems in Internet, the largest computer networking system in the world. As the only global interconnected network, Internet is extraordinarily complicated and it is very hard, if not impossible, for us to fully understand. Internet presents lots of diversity in many aspects. Domains in Internet has various relationships among them to define their behaviors in participating the global network; topological connectivity diversity exists in both the independently administrated domains and cross domains in Internet; various networking applications and services running in Internet have diverse performance requirements from the underlying infrastructures.

At first glance, Internet is already diverse enough and it is hard for us to justify any additional diversifying to Internet. However, in-depth research work reveals that there still exist lots of room, and more importantly, lots of need, for us to diversify Internet. Before answering the question “why diversify the Internet?” in details, in the following section 1.1, I first present a brief overview of Internet and introduce necessary backgrounds closely related to the research work presented in this dissertation.

1.1 Internet Overview

The Internet we are referring to today is originated from a research project in the late 1960's called the ARPANET [1], whose first permanent link was established in the
end of 1969. Over the four decades of development, the initial ARPANET with four routers has grown to be the global Internet that connects countless number of hosts in the world. The Internet today has also gone far beyond its initial academic research purpose and becomes an infrastructure available to the public. Today’s Internet is an indispensable part of many people’s daily lives. It is estimated that the Internet user population is more than 1.5 billion as of 2009 [2].

1.1.1 Commercial Agreements and Internet Topological Structure

Since the opening of Internet to commercial interests in 1988, many commercial Internet Service Providers (ISPs) have connected into the global Internet. Today’s Internet consists of about 30,000 independently administered autonomous systems (often called ASes) and one AS can have up to several hundred routers. An AS is usually an ISP that provides service to other ISPs or end-users. The connections among ISPs in Internet are based on various types of commercial agreements. In order to connect into Internet, an ISP usually buys connection from one or multiple other ISPs and becomes the customer of those ISPs. Those ISPs selling their connections to other ISPs are the providers of the customer ISPs. If an AS chooses to connect with multiple providers, it is said that the AS is multi-homed [3]. The customer-provider relationship between ASes is based on a so-called transit agreement [4], where the customer AS pays to the provider AS for all traffic coming from and going to the provider AS. Two ASes can also establish an agreement where they swap traffic to their respective customers without any monetary exchange. This is the peering agreement [4]. Each of the two ASes entering into a peering agreement is referred to as the peer of each other. The core of Internet is a small number of ASes that connect with each other into a full-mesh by establishing peering agreements among them. Those core ASes are often referred to as the tier-1 ASes. To have more reliable connection, two tier-1 ASes usually connect with each other at multiple locations.
Tier-1 ASes do not have any providers; they usually have their own networks that may large enough to cover an entire continent. Some smaller ASes buy connections from one or multiple tier-1 ASes and become customers of those tier-1 ASes. Those smaller ASes are often referred to as the tier-2 ASes. The tier-2 ASes sell their connections to even smaller scale ISPs, which are often called the tier-3 ASes. Because of this business model, i.e., smaller ASes buy connections from larger ASes, the Internet AS-level topology demonstrates a hierarchical structure [5].

As Internet is evolving, its topological structure also changes. Studies have shown that Internet is getting increasingly denser [6,7]. Factors contributing to this trend include more ASes connected into Internet and the prevalence of multi-homing and peering. The densely connected Internet implies the existence of much AS-level path diversity in Internet [8].

1.1.2 Internet Routing and Inter-domain Routing Policies

Routing in Internet faces many challenges such as scalability, stable, and efficiency. Internet routing consists of two components, routing within each individual AS and routing among the ASes, which are referred as intra-domain routing and inter-domain routing, respectively [9]. Routing within the internal network of an AS is entirely controlled by the AS itself. Intra-domain routing usually means computing the “shortest” path between routers within an AS. Many intra-domain routing protocols have been proposed and standardized [10–14]. The routing between ASes, however, is much more complicated as it requires the coordination among all ASes in the global Internet. Although many inter-domain routing protocols have been proposed [15–20], the Border Gateway Protocol (BGP) [21] is the only one actually deployed in Internet. BGP is a path vector routing protocol where a piece of reachability information basically consists of a destination and an AS path. The AS path represents a series of
ASes among which one can reach the destination. The reachability information to a destination is referred to as a \textit{path} or \textit{route} to that destination.

Different from intra-domain routing, inter-domain routing is dictated by the \textit{routing policies} of each individual AS. Generally saying, routing policies are the rules guiding how an AS propagates its reachability information to other ASes and how an AS selects its routes according to the reachability information learned from other ASes. The BGP routing protocol standardized by IETF (Internet Engineering Task Force) defines only the mechanisms to realize routing policies; the protocol itself does not have any constraints on what routing policies ASes should adopt. Hence, essentially an AS can be free to autonomously choose any routing policies. This freeness is a double-bladed sword: it provides the ability for ASes to independently choose their routing policies to achieve their own business goals and maximize their interests, but there is no guarantee of \textit{routing convergence}, which is often referred to as \textit{routing safety} as well. A routing system is safe if from any initial state, this routing system always converges into a unique final stable state where no one changes its routes any more. It is well known that the use of arbitrary routing policies in Internet inter-domain routing can lead to \textit{routing oscillation}, i.e., a set of ASes change their routes to some destinations forever without reaching a final decision [22–24].

It is computational hard to determine whether the routing policies adopted by a set of ASes will lead to routing oscillation or not [25]. Hence, people are looking for sufficient conditions that can guarantee convergence [26–28] in inter-domain routing. In practice, the routing policies adopted by an AS depend on many issues [29]. However, because most ASes in the current Internet are commercial ISPs, the commercial agreements among ASes are usually the dominating factors that determine their routing policies. The first practical and safe policy guidelines studied in [27] conclude the policy configurations into two simple rules, i.e., \textit{prefer customer} and \textit{no valley path}. “Prefer customer” constrains an AS to select a path learned from a customer if such a
path is available; “no valley path” constrains an AS not to propagate a path learned from a provider or peer to another provider or peer. If ASes in Internet follow those two guidelines and the Internet AS-level topology is hierarchical\(^1\), the inter-domain routing system guarantees to be safe. More importantly, there are strong economic incentives for ASes in Internet to follow those guidelines. Because customers always pay for any traffic transited by their providers, preferring customer paths can generate more revenue for the provider ASes. An AS allowing valley path basically means a customer transiting traffic for its two providers. That does not make economic sense as the customer needs to pay both its providers for doing that.

1.1.3 Internet Data Plane

Although Internet has complicated topological structure and control plane, its data plane is rather simple and uniform. Basically, the Internet data plane does best-effort based packet switching. For each received packet, a router first stores the packet in its cache and then forwards the packet out according to the forwarding table lookup result. If the router’s cache is full, packets are dropped according to certain strategies. Routers do not attend to recover any dropped packets. Instead, lossless packet transmission is achieved via end-to-end mechanisms. This design decision complicates the end hosts and simplifies the network, which largely contributes the scalability of Internet, where enormous amount data are exchanged every moment around the world.

In practice, some ISPs may have sophisticated data plane functions deployed within their own networks, so as to differentially forward packets and provide cer-

\(^1\)That is, the customer-provider relationships between ASes in Internet do not contain any cycles, for example, if \(u\) is a customer of \(v\) and a provider of \(w\), \(w\) cannot be a provider of \(v\), global routing safety can be guaranteed. No cycle exists in the customer-provider relationships is a rather realistic assumption because small ASes usually have large as their providers. In most cases, an AS serving a metropolitan area is likely to have a regional provider, and a regional AS is likely to have a national provider; it is unlikely that a nationwide AS would be a customer of a metropolitan-area AS [5].
tain degree of QoS guarantee. However, when packets are forwarded across domains, one can only assume them to be forwarded in a best-effort manner, because most routers in Internet treat packets equally without applying any priorities in forwarding them, even those packets contain data for different applications with divergent performance requirements. More importantly, the Internet data plane sometimes is totally a black box to applications or end users. That is, applications and end users have little control of the data plane. For the various applications running in the Internet, one can only see packet in and out, but rarely know and have little control of what happened in the routers.

1.2 Motivations of Diversifying the Internet

Although Internet already has lots of diversities in many aspects, there still exist lots of room and need for us to diversify Internet, so that Internet can provide more robust, reliable, and diverse services.

1.2.1 Internet Routing Convergence and Reliability

It is commonly aware that Internet is not as reliable as some counterpart networks, such as the telephone network. Research shows that the problem lies mainly in the control plane of Internet, i.e., the Internet routing system, specially, the Internet inter-domain routing system [30–38]. Inter-domain routing is driven by the business interests of each AS in Internet, where different ASes apply their autonomously chosen routing policies to achieve certain commercial goals and maximize their interests. Hence, how to accommodate various commercial agreements between ASes, and in the meanwhile without jeopardizing the health of the global inter-domain routing system, is of extreme importance. However, our knowledge in configuring routing policies to safely accommodate different commercial agreements is quite limited. There are guidelines supporting a few commonly existing agreements [27]. But
for a broad range of diverse agreements, some of which already present in the current Internet [5,39] and more are expected to appear in the future [40–43], it is unknown yet regarding how to handle them while preserving inter-domain routing safety. This under-investigated issue motivates the dissertation to study accommodating diverse commercial agreements in inter-domain routing.

Besides providing the guidelines for ASes in Internet to configure routing policies and achieve their business goals, accommodating diverse commercial agreements also has the benefits of introducing more Internet path diversity. Many measurements have shown that Internet is a much densely connected network where there are usually many different paths between two nodes [6–8]. However, more path diversity does not necessarily mean better routing service, unless the path diversity is used well by the routing system. The current inter-domain routing system does not make full use of the existing path diversity. Because of the policy routing nature of inter-domain routing, sometimes even an AS has a path to some destination, the AS may choose not to let some of its neighbors know such a path. Besides, an AS selects only one path to a destination and disseminates that path to other domains, even that AS has multiple paths to the same destination. The hiding of available alternative path information, although contributes to the scalability of inter-domain routing, is the root causes of many routing problems when the inter-domain routing system is experiencing dynamics [34–37]. Therefore, how to make better use of the path diversity existing in Internet, and in the meantime preserve the many desired properties of the current inter-domain routing system, also motivates the research work presented in this dissertation.

1.2.2 Providing Value-added Service in Internet

As Internet grows and evolves, there are increasing demands to provide more “value-added” services in Internet. Two types of infrastructures for hosting new net-
work services and innovative network applications are gaining considerable attention recently, i.e., network virtualization platforms [44–52] and data centers [53–59]. Network virtualization is a powerful way to run multiple concurrent virtual networks in a shared substrate, where each virtual network can be customized for certain special purposes, so as to facilitate testing and deploying new services and network innovations. To attract practical applications to be deployed in the platform, a network virtualization platform has to provide both high degree of flexibility for each virtual network to customize its functions, and the ability for each virtual network to forward packets at high speed. Although several platforms have been proposed [46–49], it is hard for them to achieve both high degree of flexibility and high packet forwarding speed, which are essential properties contributing to the success of a network virtualization platform.

As more services and application in Internet require huge amount of computation power and storage space [60–64], data centers are becoming increasingly important infrastructure in today’s Internet and more data centers are likely to appear in future Internet. To keep up with the rapid growing computation and storage demands of applications, a data center must accommodate a large number of servers. Hence, one of the essential requirements for data centers in future Internet is that they must scale to huge number of servers, e.g., hundreds of thousands or even millions of servers. However, the conventional tree-based data center network architecture requires using high-end networking devices, such as high-speed core switches, in order to scale the number of servers in a data center. Usually, those high-end devices are quite expensive. More importantly, the intrinsic limitation of available hardware, e.g., number of ports in the high-end core switches, often has a hard constraint on how many servers can be connected into a data center. The need to connect huge number of servers motivates the work presented in this dissertation in designing cost-
efficient and high performance interconnection structures to build future data centers in Internet.

1.3 Dissertation Overview

This dissertation presents my work in addressing the issues pointed out in section 1.2. Four problems are studied, i.e., extending the routing policy diversity to accommodate more complicated commercial agreements in the inter-domain routing system of Internet; exploiting Internet AS path diversity to eliminate the impact of transient routing problems in the inter-domain routing system; facilitating the deployment of diverse data plane functions via network virtualization; and designing diverse interconnection structures for data centers in Internet. The following presents a brief overview for each of the four topics studied in this dissertation.

1.3.1 Diverse Commercial Agreements

Two connecting ASes in Internet often have certain commercial agreements between them regarding how to exchange their traffic and how to settle monetary payment. Because of economic incentives, the routing policies adopted by ASes are usually based on their commercial agreements with other ASes. In today’s Internet, two widely existing agreements are transit and peering agreements. Commercial agreements between ASes are, however, continuously evolving, and commonly take many diverse forms beyond the transit and peering agreements. Their existence and evolution are driven by the business interests of ISPs and other players, the competitive marketplace, and the constantly changing Internet structure. Furthermore, the future economic structure of the Internet is likely to take many different forms [40–42], and this by itself calls for a broader set of commercial agreements. So far the only practical safe and robust routing policy guideline known to us is the Gao-Rexford’s policy guideline [27], which is applicable for transit and peering agreements only.
The first part of this dissertation studies the problem of accommodating the diverse *mutual transit agreements* in inter-domain routing [65]. A series of routing policy guidelines have been proposed, which accommodate mutual transit agreements with increasing broader meanings. Those policy guidelines are provably safe and robust when the Internet AS-level topology satisfies certain mild constraints. I use experimental results to show that the majority of peering links in today’s Internet can safely bear the richer mutual transit semantics. In addition, the benefits of peering links entering into mutual transit agreements is also evaluated, and shown to substantially improve the resiliency to a wide range of failures.

1.3.2 Inter-domain Path Diversity

The long convergence delay of the inter-domain routing system has long been realized [30–33]. Although there are a series of research works to speed up routing convergence [66–71], the distributed computation nature of inter-domain routing, which is mandatory for scalability, inevitably leads to domains across the global Internet to use inconsistent information in path computing. Hence, the transient behaviors in inter-domain routing cause lots of problems [34–36,38,72], which can greatly impact the performance of applications running in Internet [73].

The second part of this dissertation presents a scheme that adopts multiple parallel routing processes to explore the Internet AS-level path diversity, so as to eliminate the impact of transient behaviors in the inter-domain routing system [74,75]. This multi-process routing scheme seeks to improve inter-domain routing reliability with minimal changes or added complexity to the current routing system. The goal is to use BGP pretty much “as-is”, to preserve our operational knowledge and expertise with BGP, and minimize the deployment hurdles. In this scheme, each AS runs multiple slightly extended BGP processes. Those processes compute *complementary* paths. Specifically, each process selects paths to ensure that across all single network event
that affects routing, at least one routing process maintains a “reliable” end-to-end path, i.e., a path free of routing failures or loops.

1.3.3 Network Virtualization Platforms

As the Internet evolves, increasingly diverse network applications will be deployed to accommodate business and social needs. Often, network applications call for strikingly divergent performance requirements in terms of security, predictability, and throughput. However, in current Internet, only one type of data plane service is offered, i.e., best-effort data packets forwarding. Although physically separate networks could be constructed to meet these varied service requirements of different applications, a common physical substrate minimizes equipment investment, operating cost, and power consumption. Network virtualization, which supports the simultaneous operation of multiple virtual networks over a shared substrate, provides a powerful way to customize each network to a specific purpose and service requirement. For those who are using the shared substrate, a virtual network is like a physically existing network being exclusively used by them [44–46,76,77]. The goal of network virtualization is promising, but it is challenging to build such a shared network substrate that can host multiple concurrent virtual networks. Not only the flexibility of customizing each virtual network is required, high-speed packet forwarding is also highly desired in order to host realistic applications in virtual networks.

The third part of this dissertation presents two network virtualization platforms that achieve both high degree of flexibility and high speed packet forwarding [50,52,78]. Both platforms run virtual network data planes in virtual machines so as to achieve the flexibility to customize virtual network data planes. But these two platforms use different approaches to compensate the data plane performance degradation due to virtualization. The first platform adopts the parallel processing of multiple machines to achieve close to the best known software router packet forwarding speed;
the second platform achieves high speed packet forwarding by adopting a novel user mode packet forwarding scheme, which eliminates the overhead of invoking system calls and copying packet between user space and kernel.

1.3.4 Data Center Networks

Data centers consisting of a cluster of servers have become important infrastructure in Internet to provide large scale storage and computation service [60–64]. The conventional data center networks use switches to build a tree-like interconnection structure to connect servers [79]. However, as the demand for storage and computation grows quickly, the conventional data center interconnection networks become the bottleneck to limit the number of servers that can be accommodated in data centers [58]. When more servers are connected into a data center, the conventional data center network has to upgrade the switches to higher switching capability. However, high-end commodity switches are considerably more expensive. More importantly, the intrinsic limitations of the current switching hardware have hard constraints on how fast packets can be forwarded. Hence, new diverse interconnection structures are needed to accommodate huge number of servers in future data centers.

The fourth part of this dissertation proposes DPillar [80], a new server interconnection structure for data center networks. Different from the conventional data center networks with tree structures, DPillar adopts a server-centric approach, where a server is not only a computation and storage workstation, but also an intermediate nodes relaying traffic for other servers. All servers in DPillar are commodity dual-port PCs. As most server-class PCs in market and in existing data centers already have two ports, one primary port and one backup port, there is no need to physically upgrade servers in building DPillar. All switches used in DPillar are identical plug-and-play layer-2 Ethernet switches. Because low-end layer-2 Ethernet switches are inexpensive
and require little configuration, DPillar can be easily scaled to accommodate any number of servers.

1.4 Dissertation Outline

The rest of this dissertation is organized as follows. Chapter 2 presents the practical routing policy guidelines that safely accommodate diverse commercial agreements in Internet. The multi-process inter-domain routing scheme is presented in chapter 3. Chapter 4 and 5 present two pieces of work in building fully customizable and high-speed network virtualization infrastructure. Chapter 6 presents a server-centric data center network that can connect large number of servers in a cost efficient manner. Chapter 7 concludes this dissertation.
CHAPTER 2

SAFE INTER-DOMAIN ROUTING UNDER DIVERSE COMMERCIAL AGREEMENTS

2.1 Introduction

The Internet consists of thousands of inter-connected autonomous systems (ASes). Each AS enters into certain commercial agreements with a few other ASes so as to attain global reachability to the Internet. These commercial agreements determine how and what traffic the ASes exchange and thereby dictate their inter-domain routing policies. Two typical commercial agreements are transit and peering agreements. Commercial agreements between ASes are, however, continuously evolving and commonly take many forms beyond the above two agreements.

For example, one ISP may acquire or merge with another ISP. Since it is often not economically feasible to physically merge the two existing networks, the relationship between the two ASes needs to be redefined: the two ASes may now want to use each others’ providers to reach certain destinations (i.e., the two ASes now provide transit to each other). As another example, an AS might establish a private transit agreement with one of the neighbors for a particular customer (an instance of selective transit), while establishing a peering agreement with that neighbor for the rest of its customers. Similarly, two physically co-located enterprise networks might establish a mutual backup agreement, where one provides transit service to the other only when the other’s link to its own provider fails or is in maintenance. By entering into various forms of more diverse commercial agreements, the ASes not only can achieve additional cost savings, they can also enhance the service reliability and availability
to their customers. Furthermore, the future economic structure of the Internet is likely to take many different forms [40–42], and this by itself calls for a broader set of commercial agreements.

Yet, broadening the set of commercial agreements that can be accommodated in inter-domain routing is challenging. Commercial agreements dictate the routing policies adopted in each AS, and it is well known that the use of “arbitrary” routing policies can lead to routing oscillations [25]. So far, the only known practical safe and robust routing policy is Gao and Rexford’s policy guideline [27], which is applicable only for transit and peering agreements, with their extension to “backup” agreement [81]. Arbitrary agreements, such as an AS transiting traffic between any two other ASes, have been shown to possibly cause persistent routing oscillations [82]. Clearly, some caution is in order when contemplating more general agreements.

This chapter studies the routing policies that remain safe and robust while accommodating a broader range of commercial agreements. In particular, this chapter focuses on those cases where two ASes are willing to provide connectivity for each other to reach the rest of the Internet, i.e., they transit traffic for each other, and therefore establishing the so-called mutual transit agreements [5]. This kind of agreements already exists in Internet, but people have not fully understood how to accommodate them yet. More importantly, with the increasing diversity of Internet, we can expect more ASes, especially those ASes having peering agreements, would enter into various complex agreements such as the mutual transit agreements in the future. To provide the guidelines on handling the diverse and complex mutual transit agreements, this chapter introduces routing policies that expose increasing larger sets of paths and shows that those paths are indeed the types of needed paths in accommodating the diverse and complex mutual transit agreements. These policies are provably safe and robust, as long as the AS-level topology satisfies certain constraints. Experimental results show that the majority of the peering links in today’s Internet can safely bear
the richer mutual transit semantics. In addition, the benefits of peering links entering into the more diverse mutual transit agreements are also evaluated, and shown to substantially improve the resiliency to a wide range of failures.

The rest of this chapter is organized as follows. Section 2.2 presents relevant background on inter-domain routing policies, the motivations for accommodating diverse commercial agreements, and a brief overview of the chapter. Section 2.3 details the set of admissible paths produced by the mutual transit agreements. Section 2.4 specifies the rules of ranking those paths to avoid policy disputes. Section 2.5 presents the routing policies and formally establishes their safety and robustness properties. The practical implications of the proposed routing policies are discussed in section 2.6. Section 2.7 presents the experiments aimed at evaluating the benefits of extending peering agreements in the current Internet to more diverse mutual transit agreements. Section 2.8 concludes this chapter.

2.2 Background, Motivation and Overview

This section first provides some background on inter-domain routing policies and how they relate to routing safety and robustness. Then this section discusses AS business relations (or commercial agreements) that dictate the routing policies used in practice, and outlines the Gao-Rexford policy guideline. The argument here is that there exist more diverse and complex commercial agreements in reality, but how to accommodate those agreements is not clear yet. Therefore, studying this problem is both valuable in theory and needed in practice.

2.2.1 Routing Policies, Routing Safety and Robustness

In essence, routing policies specify two things: (i) the paths that are exposed or opened to neighbors, via the export policies, and (ii) preferences or ranking of the paths learned from neighbors, via the import policies. It is well known that without
any restriction on policies, so-called “policy disputes” may arise [23,26] and lead to routing oscillation. To avoid such a situation, certain limitations must be applied to routing policies (import or export policies, or both). Griffin et al. introduce the notions of routing safety and robustness [25,26]. Informally, a set of routing policies are said to be safe if the resulting routing system always converges to a unique stable state. Such routing policies are robust if they are safe under any topology changes (e.g., link failures). Furthermore, a sufficient condition for safety and robustness is identified in [26]: if a set of routing policies do not lead to a dispute wheel, they are safe and robust (see appendix 2.9.1 for the definition of dispute wheel). The problem of safety and robustness in policy routing is further investigated in [82]. The authors show that if ASes are allowed to arbitrarily filter their routes, a safe and robust routing has to constrain the route ranking to be selecting the route with the shortest weighted path length.

The safe path vector protocol is proposed in [83], which includes a mechanism to dynamically detect oscillation induced by policy dispute. This is further extended in [84], which resolves the oscillation by letting an AS select some less preferred but more stable route when that AS detects itself is involved in policy dispute. Jaggard et al. study the routing safeness problem in class based path vector systems in [28]. Sobrinho studies the convergence of path vector routing protocol using the Routing Algebra framework in [85,86]. Based on the Routing Algebra framework, a meta routing language is proposed in [87], which can be used to describe and construct safe routing protocols.

2.2.2 Practical Routing Policy Guidelines Accommodating Transit and Peering Agreements

Fortunately, in reality the routing policies adopted by ASes are dictated by the commercial agreements they have with other ASes and their own business interests.
The most common agreements are *transit* where the provider AS provides transit service to the customer AS, and *peering* where two peering ASes agree to swap traffic between their respective customers without monetary settlement [4]. Taking these two common business relations into account, Gao and Rexford present the *prefer customer* and *no valley path* policy guideline, which guarantees safety and robustness if the AS topology does not contain any provider-customer cycle [27]. This topological constraint is fairly mild because an AS usually chooses other ASes of bigger size or coverage\(^1\) than itself as the providers [27]. An AS serving a metropolitan area is likely to have a regional provider, and a regional AS is likely to have a national provider; it is unlikely that a nationwide AS would be a customer of a metropolitan-area AS.

### 2.2.3 Diverse Commercial Agreements

While the *transit* and *peering* agreements are the most common ones, far more diverse and complex commercial agreements exist. A perhaps better known and easier to understand example is the *sibling* relation [4,5], where two ASes provide transit service to each other. This relation could be established because: an ISP owns two ASes in two geographical regions, or an AS merges with or acquires another AS. At first glance, it seems that a sibling relation could be treated as two separate “provider-customer” relations and then apply the Gao-Rexford policy guideline. Such a treatment, however, would lead to a major technical problem: it violates the (mild) topological constraint under which the Gao-Rexford policy guideline is proved to be safe and robust. We use a realistic example shown in Figure 2.1 to illustrate the potential issues. In the middle of 2007, Tiscali (AS3257) acquired Pipex Broadband (AS5413) [88]. Both Tiscali and Pipex bought their transit service from TeliaSonera (AS1299), which is a tier-1 ISP [89]. Before their merging, both Tiscali and Pipex

---

\(^1\)The size of an AS could be quantified by its traffic volume, degree in the AS graph, etc. The coverage of an AS is usually the geographical area that AS covers.
use TeliaSonera to reach some destination prefix $p$. However, if they treat each other as customers, Tiscali would prefer Pipex’s route to $p$ and Pipex would prefer Tiscali’s route too. This is basically a DISAGREE scenario described in [26]. Routing oscillation may occur because no unique stable state exists in the DISAGREE scenario. As there is no systematic guideline for handling the sibling relation yet, when two ASes are merging, they usually have to treat each other as peers. This is a conservative treatment that much under-utilizes the connections between the two merging ASes, as they only use those connections to reach each other’s customers.

![Figure 2.1. Example of sibling relation established between merging ASes.](image)

Besides the sibling relation, another example of diverse agreements is that two paid peering ASes may have some special agreements for certain destinations, where they provide transit to each other only for those destinations. For other destinations, they exchange customer traffic as the standard peering agreement.

Except for the backup agreement studied in [81], *until now*, it is not clear what practical policy guidelines are needed to accommodate more diverse commercial agreements, e.g., the sibling relation, the case of peering relation with special mutual transit arrangement, and so forth, while ensuring the safety and robustness of the global inter-domain routing system. In practice, some ASes or ISPs perhaps use a few local tweaks for their own business interests, with little concern or respect for the safety and robustness of the global routing system. Hence it is of both theoretical interest
and practical value to understand when and how we can accommodate those diverse agreements in a safe and robust manner. This chapter is devoted to this problem.

2.2.4 Accommodating Mutual Transit Agreements: An Overview

We focus on how to safely accommodate the more diverse and complex mutual transit agreements in inter-domain routing. In general, two ASes having a mutual transit agreement means they are willing to provide each other the connectivity to reach the rest of the Internet [5]. For example, the sibling relation discussed above is one type of mutual transit agreement. This chapter presents the routing policy guidelines to accommodate a broad range of mutual transit agreements. Those mutual transit agreements have various semantics regarding what paths the ASes entering into those agreements can expose to each other. First, we study accommodating the mutual transit agreement where two ASes expose to each other their provider, customer, and peer paths, which is most likely what happens when two ASes are merging. Then the semantic of mutual transit is expanded, so that an AS can also announce certain paths learned from their mutual transit neighbors to other neighbors with which they have mutual transit agreements. Finally, the mutual transit agreement with the broadest meaning is considered, i.e., two ASes entering into an agreement where they announce to each other all their paths.

To study the aforementioned diverse and complex mutual transit agreements, section 2.3 studies the type of paths that should be exposed to neighbors, in order to support various mutual transit agreements. How to rank those admissible paths is discussed in section 2.4. Section 2.5 presents a series of policy guidelines that allow progressively larger sets of admissible paths, and therefore, accommodate the mutual transit agreements with progressively broader meanings. The safe and robust properties of those routing policy guidelines can be formally established.
2.3 Admissible Paths for Accommodating Mutual Transit Agreement

This section first introduces an abstract AS graph model that captures the complex nature of mutual transit agreements. Next, the concept of admissible paths is introduced. The admissible paths, essentially, specify the export policy of our policy guidelines in accommodating mutual transit agreements.

2.3.1 AS Graph Model

The AS-level topology is modeled as a graph $G = (V, E)$, where the nodes are ASes and the edges represent the agreements among ASes. An edge in $G$ can be undirected, directed, or bi-directed. An undirected edge $(u−v)$ presents a peering agreement between $u$ and $v$; a directed edge $(u→v)$ represents a transit agreement where $u$ is the provider of $v$; and a bi-directed edge $(u↔v)$ represents a mutual transit agreement between $u$ and $v$. Let $E$ denote the set of undirected edges, $→E$ the set of directed edges, and $↔E$ the set of bi-directed edges. Obviously, $E = E ∪ →E ∪ ↔E$.

2.3.2 AS Paths with Steps

A “path” $P = u_0u_1 \ldots u_m$, $(m \geq 0)$, in graph $G = (V, E)$ is an ordered sequence of distinct nodes. We say $P$ is a downhill (resp., uphill) path if all edges in path $P$ are directed edges and any node (except the first one) is a customer of its previous node in $P$ (resp., any node is a provider of its previous node). That is, $P$ is a downhill (resp., uphill) path if $\forall i \in [0, m−1], (u_i→u_{i+1}) \in →E$ (resp., $(u_{i+1}→u_i) \in →E$). Path $P$ is referred to as a “step” if all edges in $P$ are bi-directed edges, i.e., $\forall i \in [0, m−1], (u_i↔u_{i+1}) \in ↔E$. In particular, step $P$ is called a $k$-step if it contains $k$ bi-directed edges. Path $P$ is referred to as a downhill path with steps if no segment of $P$ is an uphill path.

\footnote{The path to an AS itself is considered as a downhill path, i.e., $P$ is a downhill path of $u_0$ if $m = 0$.}
path and it contains at least one bi-directed edge, i.e., \( \exists i \in [0, m - 1], (u_{i+1} \rightarrow u_i) \in \overrightarrow{E} \) and \( \exists j \in [0, m - 1], (u_j \leftrightarrow u_{j+1}) \in \overrightarrow{E} \). \(^3\)\(^3\) P is referred to as an *uphill path with steps* if all edges in \( P \) are either bi-directed edges or directed edges that are uphill path segments, and there is at least one directed edge and one bi-directed edge in \( P \). That is, \( P \) is an uphill path with steps if \( \exists i, j \in [0, m - 1], (u_{i+1} \rightarrow u_i) \in \overrightarrow{E}, (u_j \leftrightarrow u_{j+1}) \in \overrightarrow{E}, \) and \( \exists f \in [0, m - 1], (u_f \rightarrow u_{f+1}) \in \overrightarrow{E} \). If the widest step in a downhill path with steps \( P \) is a *k*-step, \( P \) is referred to as a *downhill path with k-steps*. Uphill path with \( k \)-steps can be similarly defined. See Figure 2.2 for an illustration of uphill/downhill paths (with steps).

\[^3\]A path with only bi-directed edges is a downhill path with steps.

Figure 2.2. Uphill/downhill paths and uphill/downhill paths with steps. The dashed lines represent the AS paths. (a) is an uphill path; (b) is an uphill path with step; (c) is a downhill path; (d) is a downhill paths with step.

2.3.3 Admissible Path Set

Now we illustrate what kind of paths should be permitted to accommodate the mutual transit agreements.

2.3.3.1 Not allowing valley paths

In general, no *valley paths* should be allowed, because opening valley paths essentially asks ASes to transit traffic for their providers. Given that customers must pay their providers for all traffic going to or coming from them, such a practice does
not make economic sense in general. The “valley paths” here have broader meanings than those in the Gao-Rexford policy guideline. A path $P$ is said to have a valley if it contains a downhill segment (with or without steps) followed by an uphill segment (with or without steps); or it contains a downhill segment (with or without steps), followed by an undirected edge, then an uphill segment (with or without steps). A path contains a valley is a valley path. See Figure 2.3 for an illustration.

![Figure 2.3](image-url)

Figure 2.3. Examples of valley paths, with and without steps. In (a) and (b), an AS transits traffic for its two providers; in (c) and (d), ASes with mutual transit agreements transit traffic for their respective providers; in (e) and (f), two peering ASes transit traffic for their respective providers. Allowing valley paths does not make economic sense in general.

### 2.3.3.2 Allowing valley-free paths with steps

It is necessary to permit the valley-free paths with steps to accommodate the mutual transit agreements. When two ASes $u$ and $v$ have a mutual transit agreement where they expose to each other their provider routes, customer routes, and peer routes, the result is that an AS path including $u$ and $v$ has a 1-step, i.e., edge $(u \leftrightarrow v)$. Figure 2.4 provides a depiction of six types of valley-free paths with 1-step. Further,
if AS $u$ and $v$ also announce to each other their routes learned from other neighbors with which they have mutual transit agreements, we will see valley-free paths with steps wider than one. In general, if the step width is no more than some number $k$, we define the set of admissible paths $\mathcal{P}_k$ in Definition 2.3.1.

**Figure 2.4.** Examples of admissible paths with steps in $\mathcal{P}_1$. The dashed lines are the AS paths. Note that steps do not have to appear in those six kinds of admissible paths in $\mathcal{P}_1$.

**Definition 2.3.1 ($\mathcal{P}_k$)** The set of admissible paths, $\mathcal{P}_k$, includes: (i) uphill paths with steps of width at most $k$, (ii) downhill paths with steps of width at most $k$, (iii) paths consisting of an uphill segment followed by a downhill segment and there is no steps wider than $k$, (iv) paths consisting of an uphill segment followed by an undirected edge, then followed by a downhill segment, and there is no steps wider than $k$.

Clearly, $\mathcal{P}_{k+1} \supset \mathcal{P}_k$, and in particular, $\mathcal{P}_k \supset \mathcal{P}_0$, where $\mathcal{P}_0$ is the collection of admissible paths under the Gao-Rexford policy guideline, which covers only the transit and the peering agreements. As we have mentioned, an AS path with only bi-directed
edges is a downhill path with steps, therefore, an $m$-step path, where $m \leq k$, is an admissible path in $\mathcal{P}_k$.

Here we provide some interpretations for the admissible path sets $\mathcal{P}_k$. First, opening the *valley-free paths with 1-step*, i.e., those paths in $\mathcal{P}_1 \cap \overline{\mathcal{P}_0}$, allows two ASes to have a mutual transit agreement where they expose to each other all paths except the paths learned from other mutual transit neighbors. If two ASes have a mutual transit agreement where they also announce to each other certain paths learned from other mutual transit neighbors, it is necessary to expand the admissible path set to $\mathcal{P}_k$ where $k > 1$, as steps wider than one can appear in valid AS paths. Further, if two mutual transit neighbors expose to each other all their paths, essentially the admissible path set should be $\mathcal{P}_\infty$.

### 2.4 Classes of Paths and Ranking of the Paths

We have seen that the mutual transit agreements give rise to the admissible path sets including the valley-free paths with steps. The next natural question would be how to rank these paths and set-up the preferences. Appropriate path ranking is important, otherwise “policy disputes” may arise. In this section, we first classify paths in the admissible path sets. Then we study the ranking of those paths.

#### 2.4.1 Classes of Paths in the Admissible Path Set

In admissible path set $\mathcal{P}_k$, we still have provider paths, customer paths, and peer paths, which accommodate the transit and peering agreements. If AS $a_0$ learns path $P$ from a provider (resp., customer, peer) and $P \in \mathcal{P}_k$, we say $P$ is a provider (resp., customer, peer) path of $a_0$. Besides those three types of paths, in set $\mathcal{P}_k$ where $k > 0$, there is another type of paths, i.e., the paths learned from the neighbors with whom the mutual transit agreements are established. If two ASes have a mutual transit agreement, we call them *MTran neighbors* and the link between them a *MTran link*. 
The routes learned from an MTran neighbor are referred as \textit{MTran paths} or \textit{MTran routes}.

If AS \(a_0\) and \(a_1\) are MTran neighbors, we further distinguish the paths that \(a_1\) exports to \(a_0\) into those paths going downhill and those paths going uphill in the AS hierarchy. Given an AS graph \(G = (V, E)\), a path \(P = a_0a_1...a_mQ\) (\(m \geq 1\)) learned by \(a_0\) from its MTran neighbor \(a_1\) is called a \(d_mMTran\) path if \((a_i \leftrightarrow a_{i+1}) \in \vec{E}\) \((\forall i \in [0, m-1])\) and \(Q\) is a customer path of \(a_m\). In other words, a \(d_mMTran\) path has an \(m\)-step at the \textit{beginning}, which is followed by a segment going downhill in the AS hierarchy. Likewise, we say \(P\) is an \(u_mMTran\) path of \(a_0\) if \(Q\) is a provider path or peer path of \(a_m\). When the context is clear, we sometimes drop the index \(m\), and use the terms \(dMTran\) and \(uMTran\) paths to refer to any \(d_mMTran\) and \(u_mMTran\) paths in \(\mathcal{P}_k\) \((m \leq k)\). Note that a route to a prefix owned by the AS itself is considered as a customer route of that AS, so a path consisting of a series of bi-directed edges is a \(dMTran\) path too. That is, \(P\) is a \(dMTran\) path if \(Q = \text{null}\).

Figure 2.5 depicts some examples of \(dMTran\) and \(uMTran\) paths.

![Figure 2.5](image-url)

\textbf{Figure 2.5.} Examples of \(uMTran\) paths and \(dMTran\) paths. In (a)\textasciitilde(c), AS \(a\) has \(dMTran\) path to \(d\). In (d)\textasciitilde(f), AS \(a\) has \(uMTran\) path to \(d\).
Because of opening the valley-free paths with steps, paths learned by an AS from its provider, customer, or peer may also contain steps. Besides the initial step, a \( dMTran \) or \( uMTran \) route may contain additional steps in other portions of the path.

### 2.4.2 Ranking the Paths

Now we have classified paths in \( P_k \) into provider, customer, peer, \( uMTran \), or \( dMTran \) paths. For the rankings between provider path, customer path, and peer path, similar to the Gao-Rexford policy guideline, we should prefer customer path over peer path and provider path; we do not have to enforce any preference between peer path and provider path. The unspecified cases are the rankings between \( MTran \) paths and the other types of paths, and the rankings among \( MTran \) paths. In the following, we focus on the path rankings when \( MTran \) paths (including both \( uMTran \) paths and \( dMTran \) paths) are involved.

#### 2.4.2.1 Ranking between customer paths and \( MTran \) paths

For customer path and \( dMTran \) path, we should prefer customer path over \( dMTran \) path; otherwise policy dispute as shown in Figure 2.6 can occur. Here ASes \( a, b, c \) are \( MTran \) neighbors of each other and they have AS \( d \) as their customer. AS \( d \) has a customer path to some destination prefix \( p \). If \( a, b, c \) announce their customer paths to \( MTran \) neighbors and they prefer \( dMTran \) paths over customer paths, i.e., letting \( R_1 \succ R_2 \) represent preferring route \( R_1 \) over \( R_2 \), if we have \( abdp \succ adp \) at \( a \), \( bcdp \succ bdp \) at \( b \), and \( cadp \succ cdp \) at \( c \) (the dashed lines are the preferred paths of \( a \), \( b \), and \( c \) in Figure 2.6), there is a dispute wheel.\(^4\) Setting \( \text{customer paths} \succ dMTran \text{ paths} \) resolves this problem.

\(^4\)Actually, this is a case of \textit{dispute ring} [82]. The presence of dispute ring implies persistent oscillation.
For customer path and \textit{uMTran} path, we should prefer customer path over \textit{uMTran} path to avoid policy dispute. We use Figure 2.7 to explain the reason. In Figure 2.7, AS \(a, b, c\) have both customer paths and \textit{uMTran} paths (the \textit{uMTran} path of \(a\) is \(aa'bdp\), the \textit{uMTran} path \(b\) is \(bb'cdp\), and the \textit{uMTran} path of \(c\) is \(cc'adp\)). If they prefer \textit{uMTran} paths over customer paths, a dispute wheel occurs. Setting \textit{customer paths} \(\succ\) \textit{uMTran paths} resolves this policy dispute.

\begin{figure}[h]
\centering
\includegraphics[width=0.4\textwidth]{figure26.png}
\caption{Policy dispute if \textit{dMTran} path \(\succ\) customer path.}
\end{figure}
\begin{figure}[h]
\centering
\includegraphics[width=0.4\textwidth]{figure27.png}
\caption{Policy dispute if \textit{uMTran} path \(\succ\) customer path.}
\end{figure}

Ranking the customer paths to be more preferred than MTran paths not only solves the potential routing oscillation, it also makes economic sense. Because customers always pay for their traffic transited by their providers, customer paths should always be more preferred than other paths.

\subsection*{2.4.2.2 Ranking between provider paths and MTran paths}

For provider path and \textit{uMTran} path, we use Figure 2.8 to discuss the ranking. This example is similar to Figure 2.6 except that \(d\) is a provider of \(a, b, c\). Here \(a, b, c\) learn \textit{u1MTran} paths from each other. A policy dispute arises if \(a, b, c\) prefer the \textit{u1MTran} paths announced by their MTran neighbors. This problem can be resolved if provider paths are preferred over \textit{uMTran} paths. Hence we have \textit{provider paths} \(\succ\) \textit{uMTran paths}.

Preferring provider routes over \textit{uMTran} paths also has economic justification. Consider the case that an AS has both a provider path and an \textit{uMTran} path, the
latter one goes through a provider of an MTran neighbor. If the two ASes belong to a single (merged) ISP, it is better to shift the traffic “off-the-net” as soon as possible, rather than carrying it “on-the-net” between the two ASes, as the ISP eventually needs to pay a provider to transit the traffic to the destination. Even those two ASes are two separately owned ASes with mutual transit agreement, sending the traffic through the \( uMTran \) paths instead of the provider paths would not benefit either of them. The reason is that besides one of them must pay a provider to transit the traffic, it also incurs additional cost to carry the traffic “on-the-net” between them.

![Figure 2.8](image1.png)  
**Figure 2.8.** Policy dispute if \( uMTran \) path \( \succ \) provider path.

![Figure 2.9](image2.png)  
**Figure 2.9.** Policy dispute if provider path \( \succ \) \( dMTran \) path.

Now we show the ranking between provider path and \( dMTran \) path. In Figure 2.9, \( d \) is an MTran neighbor of \( a, b, c \); \( d \) has a customer path to destination \( p \). Supposing \( a \) is a provider of \( b \) and \( b \) is \( c \)'s provider, we have \( abdp \succ adp \) at \( a \) and \( bcdp \succ bdp \) at \( b \). If \( a \) is also a provider of \( c \) and \( c \) prefers provider paths over \( dMTran \) paths, we have \( cadp \succ cdp \) at \( c \) and this leads to a dispute wheel. No policy dispute would arise if \( c \) prefers the \( dMTran \) path via \( d \) over the provider path via \( a \). Hence we should have \( dMTran \) paths \( \succ \) provider paths.

There are also economic incentives for ASes to prefer \( dMTran \) path over provider paths. Sending traffic to providers always increases the cost. However, using \( dMTran \) paths usually will not cost more, considering two mutual transit ASes usually do not charge each other (e.g., two merging ASes). Besides, the MTran neighbor can
be benefited because it will certainly send the traffic to customers and change the customers.

One thing worthy of highlighting here is that as we prefer dMTran paths over provider paths and prefer provider paths over uMTran paths, it implies that dMTran paths should be preferred over uMTran paths.

2.4.2.3 Ranking between peer paths and MTran paths

It is not too hard to see that dMTran paths should be preferred over peer paths; otherwise the policy dispute shown in Figure 2.10 may arise. Here a, b, c are peers and they are MTran neighbors of d. If a, b, c announce their dMTran paths learned from d to each other and a, b, c prefer their peer paths, there will be a dispute wheel. This dispute problem is resolved by preferring dMTran paths over peer paths. Hence we have \( dMTran \text{ paths} \succ peer \text{ paths} \). Again, such a ranking makes economic sense: Two ASes having a mutual transit agreement usually belong to the same ISP (such as merging ASes and siblings). Since a dMTran path goes through a customer of the MTran neighbor, sending the traffic through an MTran neighbor always benefits that neighbor, as its customer always pays.

\[ \text{Figure 2.10. Policy dispute if peer path } \succ dMTran \text{ path.} \]

\[ \text{Figure 2.11. Policy dispute if uMTran path } \succ peer \text{ path.} \]

For peer paths and uMTran paths, we use the example in Figure 2.11 to show how we should rank them. Here a, b, c are MTran neighbors and they have d as a peer. AS d announces its customer path to peers a, b, c; and a, b, c announce their
peer paths to each other. Therefore, $a, b, c$ have peer paths via $d$ and $uMTran$ paths via their MTran neighbors. A dispute wheel will exist if $uMTran$ paths are preferred over peer paths. This problem can be resolved if we prefer peer paths over $uMTran$ paths, i.e., \( \text{peer paths} \succ \text{uMTran paths} \).

### 2.4.2.4 Ranking among MTran paths

As we have seen, for the ranking among MTran paths, $dMTran$ paths should be more preferred than $uMTran$ paths. Here we discuss the rankings among multiple $dMTran$ paths or $uMTran$ paths. For $i < j$, we have: $d_iMTran$ paths $\succ d_jMTran$ paths, and $u_iMTran$ paths $\succ u_jMTran$ paths; otherwise policy dispute may arise.

In Figure 2.12, $a, b, c, d$ have mutual transit agreements. AS $d$ has customer paths to destination $p$ and $a, b, c$ learn $d_1MTran$ paths from $d$. ASes $a, b, c$ announce their $d_1MTran$ paths to each other so they also learn $d_2MTran$ paths. If $a$ prefers the $d_2MTran$ path learned from $b$, $b$ prefers the $d_2MTran$ path via $c$, and $c$ prefers the $d_2MTran$ path via $a$, a policy dispute occurs. This policy dispute can be resolved if $a, b, c$ all prefer their respective $d_1MTran$ paths over the $d_2MTran$ paths. Likewise, if $d$ has a provider path to destination $p$, as shown in Figure 2.13, we must have $u_iMTran$ paths $\succ u_jMTran$ paths if $i < j$, in order to avoid policy disputes.

![Figure 2.12. Policy dispute can arise if $dMTran_j \succ dMTran_i$ where $j > i$.](image1)

![Figure 2.13. Policy dispute can arise if $uMTran_j \succ uMTran_i$ where $j > i$.](image2)
Again, the rules for ranking among MTran paths economically make sense. As
the traffic will eventually be sent to some AS which is not an MTran neighbor, it is
better shift the traffic “off-the-net” as soon as possible.

From the above discussions, we see that the path ranking can be uniquely deter-
mined. We have customer path $\succ$ dMTran path $\succ$ provider path $\succ$ uMTran path, and
customer path $\succ$ dMTran path $\succ$ peer path $\succ$ uMTran path. For multiple dMTran
paths or uMTran paths, the one started by the least number of MTran links should
be preferred.

2.5 Policy Guidelines for Accommodating Mutual Transit
Agreements

We are now in a position to formally and completely specify the policy guidelines
which can accommodate the diverse mutual transit agreements. After presenting the
guidelines, we formally establish their safety and robustness properties.

2.5.1 The Policy Guidelines

We present three instances of policy guidelines, which progressively accommodate
mutual transit agreements with increasing broader meanings. Policy 2.5.1 accommod-
dates the agreement where two MTran neighbors open to each other their provider,
customer, and peer paths. Policy 2.5.2 further allows certain MTran paths to be an-
nounced to MTran neighbors. Finally, Policy 2.5.3 accommodate the mutual transit
agreement where two MTran neighbors can open any paths to each other. The safety
and robustness of those three policies are discussed in section 2.5.2.

2.5.1.1 The 1-step policy

Policy 2.5.1, denoted as the 1-step policy, accommodates the mutual transit agree-
ment where two MTran neighbors expose to each other all their paths except the
MTran paths. Because MTran paths are not announced to MTran neighbors, consecutive MTran links will not appear in any AS paths. Essentially, if this policy is adopted by ASes in Internet, the valid AS paths will be valley-free paths and valley-free paths with steps not wider than one. In other words, the admissible path set of Policy 2.5.1 is $\mathcal{P}_1$.

Policy 2.5.1 (1-step policy)

<table>
<thead>
<tr>
<th>EXPORT POLICY</th>
</tr>
</thead>
<tbody>
<tr>
<td>• To Customer: <em>announce all routes</em></td>
</tr>
<tr>
<td>• To Peer: <em>announce customer and $d_1$MTran routes</em></td>
</tr>
<tr>
<td>• To MTran: <em>announce customer, peer, and provider routes</em></td>
</tr>
<tr>
<td>• To Provider: <em>announce customer and $d_1$MTran routes</em></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IMPORT POLICY</th>
</tr>
</thead>
<tbody>
<tr>
<td>• customer $\succ d_1$MTran $\succ$ provider $\succ u_1$MTran</td>
</tr>
<tr>
<td>• customer $\succ d_1$MTran $\succ$ peer $\succ u_1$MTran</td>
</tr>
</tbody>
</table>

We believe that the valley-free paths with steps allowed by the 1-step policy are most likely what are used in reality by some ISPs today, because usually an AS has no more than one mutual transit neighbor (so no consecutive bi-directed edges will appear in any AS paths).

Policy 2.5.2 (k-step policy)

<table>
<thead>
<tr>
<th>EXPORT POLICY</th>
</tr>
</thead>
<tbody>
<tr>
<td>• To Customer: <em>announce all routes</em></td>
</tr>
<tr>
<td>• To Peer: <em>announce customer and $d_i$MTran routes</em> $\forall i \leq k$</td>
</tr>
<tr>
<td>• To MTran: <em>announce customer and provider routes; announce $d_i$MTran and $u_i$MTran routes</em> $\forall i &lt; k$</td>
</tr>
<tr>
<td>• To Provider: <em>announce customer and $d_i$MTran routes</em> $\forall i \leq k$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IMPORT POLICY</th>
</tr>
</thead>
<tbody>
<tr>
<td>• customer $\succ d_i$MTran $\succ$ $d_j$MTran ($\forall j &gt; i$) $\succ$ provider $\succ u_i$MTran $\succ$ $u_j$MTran ($\forall j &gt; i$)</td>
</tr>
<tr>
<td>• customer $\succ d_i$MTran $\succ$ $d_j$MTran ($\forall j &gt; i$) $\succ$ peer $\succ u_i$MTran $\succ$ $u_j$MTran ($\forall j &gt; i$)</td>
</tr>
</tbody>
</table>

2.5.1.2 The k-step policy

For a fixed $k > 1$, the export policy specified in Policy 2.5.2 further extends the set of admissible paths from $\mathcal{P}_1$ to $\mathcal{P}_k$, i.e., any valley-free paths with steps of width at most $k$. We call Policy 2.5.2 the $k$-step policy. Essentially, the $k$-step policy allows
an AS to announce certain MTran paths to its MTran neighbors too, i.e., announcing to MTran neighbors the MTran paths staring with steps of width smaller than $k$, and therefore, the admissible path set is $\mathcal{P}_k$.

**Policy 2.5.3 (any-step policy)**

<table>
<thead>
<tr>
<th>EXPORT POLICY</th>
</tr>
</thead>
<tbody>
<tr>
<td>• To Customer: announce all routes</td>
</tr>
<tr>
<td>• To Peer: announce customer and dMTran routes</td>
</tr>
<tr>
<td>• To MTran: announce all routes</td>
</tr>
<tr>
<td>• To Provider: announce customer and dMTran routes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IMPORT POLICY</th>
</tr>
</thead>
<tbody>
<tr>
<td>• customer $\succ d_i$MTran $\succ d_j$MTran ($\forall j &gt; i$) $\succ$ provider $\succ u_i$MTran $\succ$ $u_j$MTran ($\forall j &gt; i$)</td>
</tr>
<tr>
<td>• customer $\succ d_i$MTran $\succ d_j$MTran ($\forall j &gt; i$) $\succ$ peer $\succ u_i$MTran $\succ u_j$MTran ($\forall j &gt; i$)</td>
</tr>
</tbody>
</table>

**2.5.1.3 The any-step policy**

Lastly, Policy 2.5.3, named the any-step policy, allows valley-free paths with steps of any width. In other words, the admissible path set is $\mathcal{P}_\infty$. In a sense, Policy 2.5.3 opens the maximally extensible set of paths in accommodating mutual transit agreements, as it allows any paths to be announced to any MTran neighbors.

**2.5.2 Safety and Robustness of the Policy Guidelines**

The safety and robustness of Policy 2.5.1 $\sim$ Policy 2.5.3 can be guaranteed when the AS graph $G$ has certain topological properties. Remember that the Gao-Rexford policy guideline guarantees routing safety and robustness when the AS graph $G$ is acyclic, i.e., the directed edges in graph $G$ do not form any cycles. When ASes enter into mutual transit agreements and the bi-directed edges are presented in graph $G$, we need to re-establish the topological properties that guarantee routing safety and robustness.

An ordered sequence of nodes, $C = u_0 \ldots, u_m u_{m+1}$, where $m > 1$ and $u_{m+1} = u_0$, is a *cycle with steps* if all directed edges in $C$ point to the same direction, and there are at least one directed edge and one bi-directed edge in $C$. Further, if the longest
step in $C$ has $k$ consecutive bi-directed edges, $C$ is referred to as a *cycle with $k$-steps*, or $s_k\text{Cycle}$ in short. For example, we will refer to a directed cycle (without steps) as a $s_0\text{Cycle}$. See Figure 2.14 for an illustration of $s_0\text{Cycle}$ and $s_1\text{Cycle}$.

![Figure 2.14. Examples of $s_0\text{Cycle}$ and $s_1\text{Cycle}$.
](image)

To capture the topological properties of the AS graph that guarantee the safety and robustness of the policy guidelines presented in this chapter, we define the graph family $\text{ASG}_k$ as in the following Definition 2.5.1.

**Definition 2.5.1 ($\text{ASG}_k$)** An graph $G$ is $s_k\text{Cycle-free}$ if it contains no $s_h\text{Cycles}$, $0 \leq h \leq k$. The collection of all $s_k\text{Cycle-free}$ graphs is denoted as $\text{ASG}_k$.

Note that there may be a $s_h\text{Cycle}$ ($h > k$) in a graph $G \in \text{ASG}_k$. Hence we have the following relation: $\text{ASG}_{k+1} \subset \text{ASG}_k$. In particular, $\text{ASG}_0$ is the family of acyclic AS graphs, where there is no cycle in the provider-customer relation. The Gao-Rexford policy guideline is safe and robust for AS graph $G \in \text{ASG}_0$.

The $k$-step policy guarantees routing safety and robustness as long as the AS graph $G$ has no $s_k\text{Cycles}$ (i.e., $G \in \text{ASG}_k$), as stated in the following Theorem 2.5.1. The detailed proof of Theorem 2.5.1 is in appendix 2.9.2. The basic idea of the proof is to show that the routing system cannot contain any dispute wheel, given AS graph $G \in \text{ASG}_k$, the admissible path set $\mathcal{P}_k$, and the path rankings as specified in the import policy of the $k$-step policy. By the sufficient condition established in [26], the $k$-step policy is safe and robust.
Theorem 2.5.1 For any AS graph $G \in \text{ASG}_k$, the $k$-step policy is safe and robust.

As special cases of Theorem 2.5.1, we have the following Corollary 2.5.2, which establishes the safety and robustness of the 1-step policy. The 1-step policy accommodates the mutual transit agreement where provider, customer, and peer paths can be exposed to MTran neighbors, but MTran paths are not allowed to be announced to MTran neighbors. Therefore, among the three policy guidelines present in this chapter, the safety and robustness of the 1-step policy require the least restrictions to the AS graph $G$, i.e., AS graph $G \in \text{ASG}_1$.

Corollary 2.5.2 For any AS graph $G \in \text{ASG}_1$, the 1-step policy is safe and robust.

Finally, if the AS graph $G$ is sCycle-free, i.e., $G \in \text{ASG}_\infty$, the any-step policy is safe and robust. This fact is formally stated in Corollary 2.5.3. As we can see, the any-step policy has the least constraints on what paths can be exposed to MTran neighbors, but in the meanwhile, we have to place the most restrictive assumption on the AS topology, namely, the AS graph $G$ contains no sCycles for any $i$ (thus is strictly hierarchical), to guarantee the safety and robustness of the any-step policy.

Corollary 2.5.3 For any AS graph $G \in \text{ASG}_\infty$, the any-step policy is safe and robust.

We can see the progression from Policy 2.5.1 to Policy 2.5.3. The 1-step policy accommodates the mutual transit agreement where two MTran neighbors expose to each other all paths except their MTran paths; the $k$-step policy accommodates the mutual transit agreement where two MTran neighbors are willing to expose certain MTran paths to each other; and the any-step policy supports the mutual transit agreement where any paths can be exposed to MTran neighbors. Although the any-step policy has the least limitations for two MTran neighbors to set-up their agreement, it asks for the most constrained topological properties from the underlying AS graph in order to guarantee routing safety and robustness. On the other hand, the 1-step
policy imposes the least topological constraints to the AS graph (just slightly more than what the Gao-Rexford policy guideline requires). But the 1-step policy does not allow an AS to announce any MTran paths to any MTran neighbors. The trade-off here is that, if the routing policy accommodates more diverse and complex commercial agreements, the AS topology must satisfy more restrictive properties in order to guarantee routing safety and robustness.

2.6 Practical Implications

In this section, we discuss some practical implications of the policy guidelines presented in this chapter. We show how these policies can be realized in BGP without significant configuration effort. Other practical issues are also discussed, such as which ASes can safely establish mutual transit agreements, and how to handle selective mutual transit.

2.6.1 Realizing the Policy Guidelines in BGP

Realizing the policies put forth in section 2.5 does not require significantly more effort beyond the configuration of BGP needed today. In realizing the 1-step policy, the only extra care required is to distinguish the $d_1MTran$ and $u_1MTran$ routes. For the $k$-step policy and the any-step policy, we also need the initial step width index $i$ in $d_iMTran$ and $u_iMTran$ routes to rank those routes. In the following, we provide an example implementation to show how such information can be incorporated in the BGP community attribute. Note that the extra effort is only imposed on the configuration of those ASes having mutual transit agreements.

Recall that the 4 octets community attribute are typically represented as $x:y$ (an AS:VALUE pair), where the first two octets $x$ denote the AS number, and the second two octets $y$ denote the value. We define the value $y$ in such a matter that the first octet $y_1$ in $y = y_1:y_2$ represents the type of routes: customer, $dMTran$, 

37
peer, provider, or $uMTran$ routes. For $dMTran$ and $uMTran$ routes, the second octet $y_2$ represents the initial step width. When an AS imports a route from a customer, peer or provider, it sets the community attribute $y_1$ to customer, peer or provider accordingly\footnote{Depending on the arrangement between neighboring ASes, the community attribute may in fact be set by the neighboring AS before the route is exported.}, and sets $y_2 = 0$. Before exporting a customer route to an MTran neighbor, it sets the community attribute with $y_1 = dMTran$ and $y_2 = 1$. Likewise, before exporting a provider (or peer) route to an MTran neighbor, it sets $y_1 = uMTran$ and $y_2 = 1$. Hence when an AS imports a route from an MTran neighbor, the $y_1:y_2$ value can indicate whether it is a $dMTran$ or $uMTran$ route and the initial step width. If an AS needs to further export an MTran route to another MTran neighbor, it simply increments $y_2$ before exporting it. On the other hand, if this AS exports a $dMTran$ or $uMTran$ route to a customer, peer or provider, it resets the community attribute (in particular, setting $y_2 = 0$, $y_1 =$ customer, peer, or provider) before exporting it.

### 2.6.2 Safely Establishing Mutual Transit Agreements

Certain care must be taken when establishing mutual transit agreements between ASes, because the safety and robustness of the policy guidelines presented in this chapter hinge on some AS graph topological properties. However, given that the provider-customer relation is usually acyclic, it immediately implies that any two tier-1 ASes can establish a mutual transit agreement where they expose to each other all their paths, and the AS graph has no $sCycles$. Besides, any two stub ASes can also safely establish a mutual transit agreement where they announce to each other all their paths, and the resulting AS graph is still $sCycle$-free. Note that it is especially a useful insight that stub ASes can safely establish mutual transit agreements because the majority of the ASes in the Internet are stubs.
In general, for ASes other than stub ASes and tier-1 ASes, as long as mutual transit agreements are established only between ASes of similar size and coverage, one can ensure the resulting AS graph is free of sCycles and the policies guidelines presented in this chapter guarantee safe and robust routing. It is worthy to mention that it is to its own advantage of an AS to establish mutual transit agreements only with ASes of similar size and coverage. Otherwise, the larger AS would rather be a provider of the smaller AS to generate higher revenue.

2.6.3 Handling Selective Mutual Transit Agreements

In the discussion before, we assume a mutual transit agreement between two ASes is set for all prefixes, i.e., an MTran link has a unique meaning. In reality, however, mutual transit can be applied selectively so that the semantics of a link vary for different sets of prefixes. A realistic example could be that two peering ASes agree on using their peering link to do mutual transit only for certain destinations. Ideally, we could configure different policies for different prefixes. However, configuring policies for each prefix is not practical as there are hundreds of thousands of prefixes in the Internet. Doing policy configuration in a per-neighbor based manner is more realistic. We show an example in Figure 2.15, which is similar to the example in Figure 2.1. Here Tiscali and Pipex can have a selective mutual transit agreement so that Tiscali is willing to transit traffic for Pipex’s customer c and Pipex is willing to transit traffic to Tiscali’s customer a. Likewise, the BGP community attribute can be used to realizing the per-neighbor based mutual transit configuration. Tiscali and Pipex can locally agree on some community number to indicate the mutual transit agreement for certain prefixes. When Tiscali imports routes from customer a, Tiscali uses import filters to assign a community number to those routes. That community number should be preserved when Tiscali announces those routes to Pipex, so that Pipex will further announce them to its providers, as Pipex agrees to do mutual transit.
2.7 Experimental Evaluation

Having presented the routing policies which can safely accommodate the diverse MTran agreements, in this section we explore how the MTran agreements can affect the inter-domain routing, assuming ASes will establish those agreements. We first study the extent to which peering agreements could be extended to MTran agreements. The reason for choosing peering agreements is that they are the most natural candidates to enter into the MTran agreements (peering relationships are typically established between ASes of similar size). It not only can lead to potential routing oscillation, but also does not makes sense economically for two ASes with direct or indirect customer-provider relation to have an MTran agreement. After studying how many peering links can be potentially safely converted to MTran links, we proceed to quantify one of the benefits the conversion could afford, i.e., the ability to tolerate a wider range of failures.

We carry out our investigation by performing a number of experiments on an AS graph derived from the BGP tables archived by Routeviews [90]. We use 160 BGP table snapshots archived in January 2008 as our data set. The AS relationships are inferred using the algorithm in [5]. To speed up our experiments, we eliminate all stub ASes in the AS graph and consider only those transit ASes [91]. Note that due to the limited views available from Routeviews, the number of inferred peering links is likely to under-estimate the actual number. As a result, expanding peering agreements to
mutual transit agreements can be expected to have even larger impact than shown in our results.

2.7.1 Potential Peering Agreements that Can Be Safely Converted to Mutual Transit Agreements

For the policies that require $s_k$Cycle-free AS graph $G$, we want to identify the number of peering agreements that could be converted to mutual transit agreements, i.e., assuming the involved ASes wished to do so. In practice, many of those ASes may decide not to enter into such agreements even if it is feasible to do that. However, barring such detailed knowledge, this number provides a useful estimation of the potential for greater routing choices that the policies presented in this chapter can afford. Exploring the benefits achievable from that diversity is the topic of section 2.7.2.

We use the following heuristic algorithm to identify which peering links could safely adopt mutual transit agreement. We construct an AS graph $G$ whose directed edges $\vec{E}$ are transit links, bi-directed edges $\vec{E}$ are MTran links, and undirected edges $\vec{E}$ are peering links. For each undirected edge $e \in \vec{E}$, we test whether $e$ is in any $s_i$Cycles ($i \leq k$). If the answer is negative, we count it as expandable to mutual transit agreement and change it to a bi-directed edge in $G$. The test is performed until all peering links have been examined. The experiment is carried out for different values of $k$ and the results are reported in Figure 2.16. As we can see, when $k = 1$, about 78% of peering links can be safely enter the more diverse mutual transit agreements. This figure drops down to about 50% when $k \geq 2$, and stays approximately at that level as $k$ increases further.

In order to understand how these link-level results mapped onto ASes, we counted the number of ASes that have at least one peering link that could safely establish mutual transit agreement. When $k = 1$, among all transit ASes with peering neigh-
bors, 96% of them can expand at least one of their peering agreements into mutual transit agreement. This drops down to about 62% when $k > 1$.

### 2.7.2 Improving Fault Tolerance by Extending Peering into Mutual Transit

After investigating how many peer links and ASes with peer agreements could enter into mutual transit agreements, in this subsection, we evaluate the potential benefit of converting the peering links that can be safely converted into mutual transit agreements. The benefit we focus on is routing reliability. In particular, we are interested in a few common failure scenarios and how the expanded mutual transit agreements can help to tolerate those failures. In our experiments, we compare the Gao-Rexford policy guideline (which accommodate only the transit and peering agreements) to the 1-step policy and the any-step policy specified in section 2.5.

For each failure scenario, we compare the number of reachable AS pairs before and after a failure. If AS $u$ can reach AS $v$ and AS $v$ can reach AS $u$ using paths permitted by the corresponding routing policy, we say $(u, v)$ is a reachable AS pair. If $(u, v)$ is a reachable AS pair and it becomes unreachable after the failure, we say

![Figure 2.16](image-url)  

**Figure 2.16.** Percentage of peering links which can be safely expanded into mutual transit agreements for $k$ ranges from 1 to 6.
is a disconnected AS pair. We have considered the following three categories of failures.

**Access link failures:** Access links are the links connecting an AS to its providers. An AS with a peer neighbor can tolerate access link failures by expanding its peer agreement into mutual transit agreement. That is, if all access links of an AS fail, the peering neighbor can transit its traffic. We ran 50 instances of failure experiments. In each instance, one AS among all the ASes that can safely convert one of their peer links into mutual transit agreement is selected, and all its access links are failed. We count the number of disconnected AS pairs in each experiment instance. The results of disconnected AS pairs for the Gao-Rexford policy is presented in Figure 2.17. As we can see, a significant number of AS pairs become disconnected when using the Gao-Rexford policy. However, under either the 1-step or the any-step policies, no AS pairs are disconnected in this failure scenario.

![Figure 2.17](image-url)

**Figure 2.17.** Number of disconnected AS pairs in case of access link failures when peering ASes are not willing to do mutual transit.

**Tier-1 de-peering:** This corresponds to a scenario where two tier-1 ASes decide to de-peer. As the study in [91] shows, tier-1 de-peering can have a huge impact on the reachability of ASes single-homed to the de-peered tier-1 ASes. We select some well-known tier-1 AS pairs [89] and let them de-peer in our experiments. The number of disconnected AS pairs after the de-peering is presented in Table 2.1. Not unexpect-
edly, the 1-step policy does not offer any improvement over the Gao-Rexford policy. In contrast, the any-step policy is able to entirely eliminate any loss of connectivity. This is because the any-step policy allows AS paths with multiple consecutive peering links (now they have the mutual transit semantics) to be used, the de-peered tier-1 ASes can use other tier-1 ASes to bypass the failed peering link.

<table>
<thead>
<tr>
<th>peering link</th>
<th># of disconnected AS pairs</th>
<th>Gao-Rexford</th>
<th>1-step</th>
<th>any-step</th>
</tr>
</thead>
<tbody>
<tr>
<td>1239 - 3356</td>
<td>546</td>
<td>546</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1239 - 7018</td>
<td>294</td>
<td>294</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>701 - 1239</td>
<td>273</td>
<td>273</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>701 - 3356</td>
<td>338</td>
<td>338</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1. Number of disconnected AS pairs under tier-1 de-peering.

**AS partition:** This last scenario considers failures that partition a tier-1 AS into two disconnected components. Using the NetGeo data [92], we classify the US customers of a tier-1 AS into three categories: east coast customers, west coast customers, and other customers. We assume that after a partition the east coast customers and west coast customers of the tier-1 AS cannot reach each other through that tier-1 AS. We test two well-known tier-1 ASes, Quest and AT&T, and present the results of disconnected AS pairs in Table 2.2. As in the tier-1 de-peering scenario, the any-step policy offers full protection against the AS partition failure. This is again because it allows a second tier-1 AS to transit the traffic between the east coast and west coast customers of the partitioned tier-1 AS.

<table>
<thead>
<tr>
<th>tier-1 AS</th>
<th># of disconnected AS pairs</th>
<th>Gao-Rexford</th>
<th>1-step</th>
<th>any-step</th>
</tr>
</thead>
<tbody>
<tr>
<td>209 Quest</td>
<td>86</td>
<td>86</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7018 AT&amp;T</td>
<td>113</td>
<td>113</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2. Number of disconnected AS pairs under tier-1 AS partition.
2.8 Conclusion

This chapter studies the fundamental problem of safely accommodating a broader range of commercial agreements in inter-domain routing. We propose a series of policy guidelines which support mutual transit agreements with progressively richer semantics. We study the safety and robustness of those policy guidelines. Based on those theoretical insights, we further discuss how the diverse mutual transit agreements can be safely established and be easily implemented in BGP. We also demonstrate the benefits in terms of routing reliability in various representative failure scenarios, if the peering agreements in Internet could be extended to mutual transit agreements.

2.9 Appendix

2.9.1 Dispute Wheel

The safety and robustness of our routing policy guidelines are established by a sufficient condition proved in [26], i.e., no dispute wheel ensures safety and robustness. A dispute wheel $W$ of size $m$, as shown in Figure 2.18, is a triple $(U, Q, R)$, where $U$ is a sequence of $m$ nodes $u_0, u_1...u_{m-1}$; $Q$ is a sequence of $m$ non-empty paths $Q_0, Q_1...Q_{m-1}$; and $R$ represents $m$ non-empty paths $R_0, R_1...R_{m-1}$. This triple is such that for each $0 \leq i < m$, we have (1) $R_i$ is a path from $u_i$ to $u_{i+1}$; (2) $Q_i$ and $R_iQ_{i+1}$ are valid paths at $u_i$; and (3) $u_i$ prefers $R_iQ_{i+1}$ over $Q_i$. All subscripts are to be interpreted modulo $m$.

2.9.2 The Safeness and Robustness of the k-step Policy Guideline

We first prove the following Lemma 2.9.1, which says that if a dispute wheel exists when the k-step policy is adopted, the rim of the dispute will not be all MTran links.

Lemma 2.9.1 For any AS graph $G \in ASG_k$, if there is a dispute wheel $W = (U, Q, R)$ by adopting the k-step policy, the rim of $W$ cannot have only MTran links.
Proof Given an AS graph $G \in \mathcal{ASG}_k$, we first assume a dispute wheel $W = (\mathcal{U}, Q, \mathcal{R})$ of size $m$ exists in a routing system adopting the k-step policy and $R_i$ has only MTran links, $\forall i \in [0, m-1]$. Obviously, because $u_i$ prefers $R_i Q_{i+1}$ over $Q_i$, $\forall i \in [0, m-1]$, $Q_i$ cannot be customer route of $u_i$; $Q_i$ cannot be provider route or peer route of $u_i$ either. Therefore, $\forall i \in [0, m-1]$, $Q_i$ must be MTran path of $u_i$. Besides, all $Q_i$s are $uMTran$ routes of $u_i$, or all $Q_i$s are $dMTran$ routes of $u_i$.

**Case 1:** If $\forall i \in [0, m-1]$, $Q_i$ is an $uMTran$ route of $u_i$, let $\mathcal{H}(R)$ be the width of the step at the beginning of path $R$, we have

$$
\begin{align*}
\mathcal{H}(R_0) + \mathcal{H}(Q_1) & \leq \mathcal{H}(Q_0) \\
\mathcal{H}(R_1) + \mathcal{H}(Q_2) & \leq \mathcal{H}(Q_1) \\
& \vdots \\
\mathcal{H}(R_{k-1}) + \mathcal{H}(Q_0) & \leq \mathcal{H}(Q_{k-1})
\end{align*}
$$

(2.1)

From (2.1), we have $\sum_{i=0}^{k-1} \mathcal{H}(R_i) \leq 0$, which is impossible because $\min(\mathcal{H}(R_i)) = 1$.

**Case 2:** If $\forall i \in [0, m-1]$, $Q_i$ is a $dMTran$ route of $u_i$, similarly, we can derive a contradiction too.

Hence, we know that the rim of $W$ cannot have only MTran links.
The following Lemma 2.9.2 further states that if the \textit{k-step} policy is adopted and there exists a dispute wheel \( W \), the \textit{rim} of \( W \) must be a \( s_h \text{Cycle} \) where \( h \leq k \).

\textbf{Lemma 2.9.2} If a dispute wheel \( W = (U, Q, R) \) exists in a routing system adopting the \textit{k-step} policy, \( R_0 R_1 \ldots R_{m-1} \) must be a \( s_h \text{Cycle}, h \leq k \).

\textbf{Proof} Assuming there is a dispute wheel \( W = (U, Q, R) \) when the \textit{k-step} policy is adopted, without loss of generality, we first consider the case where \( Q_0 \) is a customer route of \( u_0 \).

When \( Q_0 \) is a customer route of \( u_0 \), \( R_0 Q_1 \) must a customer route of \( u_0 \) too as it is preferred by \( u_0 \). Because no valley path (with or without steps) is allowed, \( R_0 \) should be a downhill path (with steps) from \( u_0 \) to \( u_1 \) and \( Q_1 \) can only be customer path or \textit{dMTran} path of \( u_1 \). For either case, \( R_1 Q_2 \) must be either a customer path or \textit{dMTran} path of \( u_1 \) so that \( u_1 \) can prefer \( R_1 Q_2 \) over \( Q_1 \). Again because there is no valley path (with or without steps), \( R_1 \) should be a downhill path (with steps) from \( u_1 \) to \( u_2 \). By keeping doing this, eventually we will have \( R_0 R_1 \ldots R_{m-1} \) is a downhill path (with steps) from \( u_0 \) to itself. Also because Lemma 2.9.1 guarantees that \( R_0 R_1 \ldots R_{m-1} \) cannot have only \textit{MTran} links, it should be a \( sCyle \).

Next we show that \( R_0 R_1 \ldots R_{m-1} \) cannot have a segment with more than \( k \) consecutive \textit{MTran} links. Assuming the rim of \( W \) has such a segment, it must be located at the concatenation point of \( R_i \) and \( R_{i+1} \). Let \( \mathcal{H}(R) \) and \( \mathcal{T}(R) \) represent the width of the step at the beginning and at the end of path \( R \), respectively. Without loss of generality, we assume

\[
\mathcal{T}(R_{m-1}) + \mathcal{H}(R_0) > k
\]  

(2.2)

This also implies \( R_0 Q_1 \) is a \textit{MTran} path of \( u_0 \). We consider the following two cases:
Case 1: If $R_0Q_1$ is an $uMTran$ path $u_0$, $Q_0$ must be an $uMTran$ path $u_0$ too. Because $u_0$ prefers $R_0Q_1$, we have

$$H(R_0Q_1) \leq H(Q_0) \quad (2.3)$$

Also because $R_{m-1}Q_0$ is a valid path path of $u_{m-1}$, it should not have steps wider than $k$, i.e.,

$$T(R_{m-1}) + H(Q_0) \leq k \quad (2.4)$$

From (2.3) and (2.4), we can derive $T(R_{m-1}) + H(R_0Q_1) \leq k$. This contradicts with (2.2) because $H(R_0Q_1) \geq H(R_0)$.

Case 2: If $R_0Q_1$ is a $dMTran$ path of $u_0$, $Q_0$ can be a $dMTran$ path, a peer path, a provider path, or an $uMTran$ path of $u_0$. Case 2.1: If $Q_0$ is a $dMTran$ of $u_0$, we can derive a contradiction similar to case 1. Case 2.2: If $Q_0$ is a provider path, a peer path, or an $uMTran$ path of $u_0$, $R_{m-1}Q_0$ must be an $uMTran$ path or a provider path of $u_{m-1}$. Because $u_{m-1}$ prefers $R_{m-1}Q_0$ over $Q_{m-1}$, $Q_{m-1}$ must an $uMTran$ path or a provider path of $u_{m-1}$. Hence, $R_{m-2}Q_{m-1}$ is an $uMTran$ path or a provider path of $u_{m-2}$. By keeping doing this, we can derive that $R_0Q_1$ is an $uMTran$ path or a provider path of $u_0$, this contradicts with the assumption that $R_0Q_1$ is a $dMTran$ path of $u_0$.

Based on the above case 1 and 2, inequation (2.2) should not hold. Therefore, the rim of $W$ is a $s_hCycle$ where $h \leq k$.

Similarly, for other cases where $Q_0$ is a provider path, a peer path, a $dMTran$ path, or an $uMTran$ path of $u_0$, we can have the same conclusion, i.e., $R_0R_1...R_{m-1}$ is a $s_hCycle$ where $h \leq k$.

Now we are in the position to prove Theorem 2.5.1.

48
Proof When the $k$-step policy is adopted and a dispute wheel exists, Lemma 2.9.2 tells us that the rim of the dispute wheel must be a $s_h$ Cycle where $h \leq k$. This contradicts the fact that the AS graph $G \in \mathcal{ASG}_k$. Therefore, the dispute wheel does not exist and the $k$-step guarantees routing safety and robustness. $\blacksquare$
CHAPTER 3

RELIABLE INTER-DOMAIN ROUTING THROUGH MULTIPLE COMPLEMENTARY ROUTING PROCESSES

3.1 Introduction

With the increasing popularity of time-sensitive or interactive Internet applications such as VoIP, video streaming, on-line gaming, etc, it has become ever more important for the Internet routing system to provide “reliable” end-to-end paths. As basic as this requirement is, it has proven challenging, because the distributed nature of Internet routing decisions, something that scalability mandates, introduces unavoidable latency when reacting to network changes (such as link/node failures). This has been particularly evident in inter-domain routing, where the shortcomings of the de facto standard routing protocol, BGP, are well known [38]. For instance, BGP may take as long as 30 minutes to converge after certain routing events [33], and during those periods “transient” routing loops and loss of network reachability frequently occur. Measurement studies [34–37] have shown that 55% to 85% of short-lived routing failures are due to transient routing failures during BGP convergence, and that transient loops account for up to 90% of all packet losses.

Researchers have sought to address this challenge and proposed several approaches to improve inter-domain routing reliability. One approach is to speed-up BGP convergence; hence limiting the duration and thereby impact of transient routing loops and failures [66–71]. In particular, faster convergence will occur if obsolete routing information is rapidly removed across routers, e.g., by propagating additional information such as root cause information (RCI) that can be used to invalidate routes affected
by a common failure. Another approach to limiting the impact of transient loops and failures is to compute backup paths that can supplement the “best path” selected by BGP. As with approaches to speed up BGP convergence, enabling the selection of good backup paths calls for making additional routing information available. This need for additional information introduces overhead and modifications that can affect the odds of successful deployment. For example, both indicative rerouting [93] and R-BGP [94] assumes the availability of RCI to remove inconsistent routing information. As discussed later, RCI is currently not available in BGP, and its incorporation in BGP requires careful design and adds considerable implementation complexity to an already-fragile system. Besides, implementing RCI entails revealing substantial details about the underlying physical network topology. This may not always be possible, e.g., for privacy or security reasons, and when it is, the need to disseminate that much additions information can have a significant effect on the complexity of the routing system.

In this chapter presents my research work that seeks to improve inter-domain routing reliability with minimal changes or added complexity to the current routing system [75]. The goal is to use BGP pretty much “as-is,” and in particular without resorting to RCI, to preserve current operational knowledge and expertise and minimize deployment hurdles. In realizing this goal, our basic idea is to have each AS run multiple (two) very slightly extended BGP processes that exploit the AS-level path diversity of the Internet to compute complementary paths. Specifically, each process selects paths to ensure that across all network events that affect routing, at least one routing process maintains a “reliable” end-to-end path, i.e., a path free of routing failures or loops. In other words, the routing processes complement each other across the space of possible network events. As we demonstrate in this chapter, in addition to being feasible with minimal changes to BGP, this approach offers protection against a broader range of routing events than existing alternatives.
Although the intuition behind computing complementary paths is straightforward, translating it into reality in the context of inter-domain routing is challenging. Not only do distributed computations have to be coordinated, but the resulting paths need to be policy compliant. This in itself, has been shown to be a hard problem. Our goal is, therefore, to develop an approach that allows the distributed computation of disjoint AS paths, while accommodating existing policy constraints and relying on the BGP protocol with as few changes as possible. In tackling this problem, we first identify possible simplifications brought about by the current Internet structure and common routing policies. In particular, we establish that complementary routing solutions can be obtained by focusing only on the “downhill” portion of paths, i.e., the segments that extend from provider ASes to customer ASes towards the destination. This affords some simplifications, but the problem remains hard, and we introduce a simple heuristic whose performance we demonstrate through extensive experiments on the current inter-AS topology.

After devising a practical and effective scheme for realizing complementary routing processes, we turn our attention to defining how to use them. In particular, identifying when to switch from one process to another, and which one is offering a working path. Given the distributed nature of inter-domain routing, such decisions cannot be coordinated across ASes. Hence, to avoid replacing one problem with another, it is also important to ensure that these switching decisions do not themselves introduce loops or outages. We propose a simple approach to this problem and argue its effectiveness and safety.

In summary, the main contributions of our work are two-fold: (i) we devise a simple and practical scheme for significantly enhancing the reliability of inter-domain

\footnote{Computing disjoint inter-domain paths was investigated in [95], which established that for common policy constraints computing disjoint paths in a ToR (Type-of-Relationship) graph is NP-hard unless the graph is acyclic.}
routing; and (ii) we do so in a manner that leverages existing experience with BGP protocol, and which can be incrementally deployed with minimum disruption.

The rest of this chapter is organized as follows. Section 3.2 provides necessary background information on inter-domain routing and reviews related works. Section 3.3 discusses the motivation and basic design principles behind our multi-process routing scheme. Details on its design and realization are given in section 3.4 and section 3.5. Section 3.6 is devoted to an extensive evaluation of the scheme and its performance. Section 3.7 concludes this chapter.

3.2 Background of Inter-domain Routing and a Review of Related Work

In this chapter we first provide a brief overview of BGP and lay out our assumptions. We then introduce the transient routing problems faced by BGP, outline representative proposals for limiting or eliminating them. We focus on eBGP, which controls exchanges of routing information between ASes. Other related works are also briefly touched on.

3.2.1 Transient Routing Problems in Inter-domain Routing

A basic requirement for any routing system is safeness, i.e., the ability to converge to a stable state for any initial state and combination of routing events. BGP is an incremental path vector protocol that accommodates a wide spectrum of routing policies, so that without constraints on their generality its safeness cannot be guaranteed [25]. In practice, however, neighboring ASes usually engage in bilateral agreements, also called AS relationships, which determine and constrain their routing policies. The two most common ones are i) customer-provider relationships where a customer AS pays a provider to transit its traffic, and ii) peer-peer relationships where two ASes agree to swap traffic of their respective networks (and their customer
ASes) for free. Because of the economic interests defined by the relationships, ASes typically follow two common routing policies, *prefer-customer*—an AS always selects customer routes (routes learned from a customer) whenever available, and *valley-free*—an AS does not advertise provider/peer routes (routes learned from a provider/peer) to other providers/peers. Assuming that the customer-provider relationships between ASes are *acyclic*, which holds in practice, the BGP protocol has been shown to be *safe*, if every AS adopts these two policies [27]. This chapter assumes that these two policies are adopted by all ASes across the Internet.

The safeness of BGP notwithstanding, it only implies that routing “eventually” converges. However, during convergence affected ASes can experience *transient* loss of reachability, commonly referred to as *transient routing failures* [37]. Moreover, inconsistency in routing information across ASes during convergence can also result in *transient routing loops*. We use an example, shown in Figure 3.1, to illustrate these transient routing problems. In the figure, AS 1 is the destination, with the selected path to that destination shown next to each AS. If the link between AS 6 and AS 1 fails, AS 6 loses its path and sends a withdrawal to AS 5. AS 5 will not announce its alternate path to AS 6 until its MRAI (Minimum Route Advertisement Interval) timer expires. AS 6, therefore, experiences a transient failure until AS 5 announces its alternate path. Similarly, if the link between AS 2 and AS 1 goes down, AS 2 sends withdrawal messages to AS 3 and AS 4. Both of them will then switch to their alternate paths, with AS 4 using path 4:3:2:1 and AS 3 using path 3:4:2:1. There is a transient loop between AS 3 and AS 4 until one of them announces its route to the other.

The relatively slow convergence of BGP after many network events, therefore results in frequent transient routing problems that are major contributors to network

---

2Namely, the provider of any AS cannot be a customer of that AS’s customer, or a customer of a customer, and so on.
Figure 3.1. An example to show transient routing problems of BGP.

performance degradations [37]. Hence limiting or eliminating the impact of transient routing failures has become the focus of much recent research activity.

3.2.2 Related Work Addressing the Unreliable Problem of BGP and Their Limitations

One of the main reasons behind BGP’s slow convergence is the so-called path exploration phenomenon [31]. As alluded to earlier, its impact can be limited through the availability of additional information such as RCI on the cause of route changes [69,70]. However, as shown in [70], correctly implementing RCI to expose path dependencies calls for careful design that can add significant complexity to BGP. For example, one approach is to annotate the AS_PATH attribute in BGP update messages with the ingress/egress router information of each AS on the path. Apart from the additional network bandwidth and router memory overhead, changes in RCI may also induce more routing update messages, thus increasing BGP dynamics.

Nevertheless, extensions such as RCI open the door for improvements to BGP’s slow convergence, and the R-BGP scheme of [94] relies on it to eliminate transient problems under network instability involving a single link failure. The basic idea behind R-BGP is to pre-select a fail-over route that is used when the primary route
fails. For illustration purposes, consider the configuration of Figure 3.1. Under R-BGP, each AS selects a “most disjoint” route as its fail-over route and advertises it only to the next-hop AS on its best route, e.g., AS 4 selects route 4:5:6:1 as its fail-over route and announces it to AS 2 only. In the case of single link failures, R-BGP uses RCI to expose path dependencies and avoid transient loops. For instance, when link (1:2) fails, upon receiving withdrawals from AS 2, both AS 3 and AS 4 rely on RCI to remove their alternate routes, 3:4:2:1 and 4:3:2:1, and therefore do not attempt to use them. Note though that this results in AS 3 being left with no valid routes. To “compensate” for this complete loss of connectivity induced by RCI, R-BGP allows AS 3 to continue forwarding packets along its old primary route 3:2:1, namely, to AS 2. In turn, AS 2 forwards packets along its fail-over route 2:4:5:6:1. Hence, in this scenario, R-BGP successfully eliminated transient problems caused by the single link failure.

Extending this success to other failure scenarios is unfortunately not immediate, and in particular it can be shown that R-BGP cannot handle scenarios involving the failure of multiple links adjacent to a single AS; a reasonably common and realistic occurrence. For example, multiple links adjacent to the same AS can fail at the same time due to the crash of a border router. Or a policy change in an AS can lead to route withdrawals to several neighbors. We illustrate this through an example using again the network of Figure 3.1. Suppose that AS 2 connects to AS 1 and AS 4 via a single border router, and connects to AS 3 through another router. Suppose the former router crashes, so that both the primary path 2:1 and the fail-over path 2:4:5:6:1 are now invalid. Upon receiving withdrawals from AS 2 with embedded RCI, AS 3 invalidates all its routes but continues forwarding packets to AS 2 in conformance with R-BGP’s rule. Those packets are, however, dropped at AS 2 since its fail-over route is unavailable. Hence both AS 3 and AS 2 lose connectivity to AS 1, in spite of the availability of an alternate route for both of them (3:4:5:6:1 for AS 3 and 2:3:4:5:6:1
for AS 2). Note that AS 3’s inability to learn about that alternate route is caused by R-BGP’s limitation to have an AS announce its fail-over route only to the next-hop on the current best route, e.g., AS 4 announces its fail-over route only to AS 2.

Limitations in existing solutions to exploit the rich path diversity available in the Internet topology, and more importantly the significant added cost implied by the additional mechanisms such as RCI that they require, are the principal motivations behind this chapter’s attempt at designing a new solution to improve the reliability of inter-domain routing. Before we proceed with the description of our proposed scheme, we complete this section by quickly mentioning a few other relevant works with a similar goal.

Other Related Work. Qiu et al. proposed the indicative re-routing in [93], but their scheme only reduces the chance of transient loop or failure. Xu et al. proposed a multi-path inter-domain routing scheme in [96]. Their focus is on how to explore the richness of connectivity in the current Internet and provide value-added services. The consensus routing proposed in [97] aims to take advantage of a distributed protocol to build a consensus view of the network. However, when network changes are in the form of node/link failures, consensus may not always be established in time to prevent transient failures.

3.3 Multi-Process Routing: Basic Design Principles

In this section, we present key aspects of the approach we propose to improve the reliability of inter-domain routing, and establish several properties that motivate its subsequent investigation. Before we move on to these key aspects, we first introduce routing events that multi-process routing protocol aim to handle.
3.3.1 Routing Events

Routing events are the underlying network events (or “root causes”) such as a link failure or recovery, BGP session reset or re-establishment, router crash or recovery, policy change. Based on the underlying causes and their manifestations, we classify routing events into three classes: a route withdrawal event triggers one or multiple ASes (incident to the event) to withdraw (or replace) affected routes, e.g., due to failures; a route addition event triggers one or multiple ASes to announce new routes, e.g., due to recovery; and a route change event triggers an AS to announce route updates (but no withdrawals) due to a policy change. Note that under our definition, a single route event can trigger multiple concurrent route updates (which may originate from different ASes). For instance, as shown by the example in section 3.2.2, the crash of a border router connected to several neighbor ASes may trigger a route withdrawal from each of these ASes. Instead of treating these route withdrawals as separate “failure events,” we consider them as caused by a single (route withdrawal) event. Another example is a policy change within an AS (a route change event) that affects multiple neighbor ASes. An important goal of our scheme is to ensure the reliable delivery of packets against all disruptions associated with a single routing event as defined above. Note that this is a more stringent requirement than that of previous similarly motivated proposals, e.g., R-BGP, which as illustrated earlier did not deal with some multiple correlated “route failures” triggered by a single underlying routing event.

3 These ASes will detect the failure of their respective BGP session with the said router, but may not know the underlying root cause!
3.3.2 Multiple Routing Processes

Given the above set of route events, our approach to avoiding the transient loops and failures can give rise to combine a control plane mechanism and a closely related data plane mechanism.

In the control plane, we seek to provide robustness to any single network event, by identifying two distinct sets of routing decisions in each AS that are complementary in how they are affected by single network events. This is realized by having multiple slightly modified BGP routing processes running in parallel in each AS. Those processes can, for example, be differentiated through the use of different TCP port numbers. Paths selected by different routing processes should satisfy a key property, namely, node disjointness, i.e., not share any common AS nodes except the source and destination (recall that our focus is on AS-level path, with each AS a “node” in the path). This ensures that they are not affected by the same sets of events. The use of separate routing processes that select distinct (disjoint) paths, provides considerably more flexibility than relying on a single routing process that selects one best path, which must then be supplemented by finding a good fail-over path among the remaining available paths. For example, as shown in Figure 3.2 and using the same AS topology as in Figure 3.1, complementary processes in each AS compute two disjoint paths, the red and blue paths, to destination AS 1. The availability of these two node-disjoint paths ensures that one of them remains operational in the presence of any single node (AS) and multiple link (from the same AS) failures. In contrast, as shown in in section 3.2.2, such “multiple” failures disrupt routing even when a protection scheme such as R-BGP is in use.

3.3.3 Downhill Node Disjointness

Having stated our overall goal, namely, computing node disjoint paths, we pause to point out that realizing it while preserving BGP’s policy-based distributed com-
putations is non-trivial, i.e., recall [95]. In section 3.4, we present our approach based on using two essentially standard BGP processes together with some simple coordin-
ination rules within each AS. Besides computational challenges, requesting full path disjointness can also limit the choices available to routing processes running in an AS. Fortunately, full path disjointness is not necessary to realize paths that are com-
plementary in the sense that they are not affected by the same route event. As we shall see, the hierarchical structure of the Internet together with the common policies guiding permissible routing choices, allow us to “simplify” this requirement. Before presenting this simplification, we introduce two definitions needed to characterize AS paths and establish the result.

Because of the valley-free routing policy, an AS path usually goes through a se-
quence of customer-to-provider links, possibly followed by a peer-to-peer link, and
finally a series of provider-to-customer links. We define the uphill portion of an AS
path as a sequence of customer-to-provider links followed by a peer-to-peer link (if it exists) and the ASes at the two ends of each link, except the AS whose next hop is its customer. The downhill portion of an AS path is a sequence of provider-to-customer links, together with the ASes at the two ends of each link.
Using the above definition of uphill and downhill portions of a path, we will demonstrate that complementary routing processes only have to ensure node disjointness in the downhill portions of their paths. Specifically, they need to produce downhill node disjoint paths, as formally stated below:

**Definition 3.3.1** Two AS paths are downhill node disjoint paths if the downhill portions of both paths do not have any AS in common.

**Route withdrawals.** To establish that downhill node disjointness is sufficient to ensure that the routing processes are complementary under any single routing event, we first show that it is sufficient for any single route withdrawal event.

The reason why node disjointness is needed only in the downhill portion is that under the constraints of common routing policies, network events in the uphill portion of a path will not trigger transient loops or failures during BGP convergence. In other words, a link or AS failure or a policy change in a higher tier AS (provider) does not create transient failures or loops at an AS while its BGP process adapts to the changes. This is more formally expressed in Lemma 3.3.1.

**Lemma 3.3.1**  A route withdrawal event in the uphill portion of an AS path to a destination does not produce transient routing loops or failures during BGP convergence.

**Proof** We first introduce the definition of routing graph. The routing graph for one destination AS $p$ is a direct graph $G = (V, E)$. A node $v \in V$ represents an AS. A direct edge $\overrightarrow{e} = (u, v) \in E$ if $v$ is the next hop of $u$ to reach $p$. When BGP is in a stable state, the correspond routing graph is a DAG (Direct Acyclic Graph) and a node has one outgoing edge if there exists a policy compliant path for that node.

We consider the scenario where there is a route withdrawal event caused by a node (AS) failure. Route withdrawal events caused by link failure and policy change can be proved similarly. Suppose the failed AS is $u$ and $u$ is in the uphill portion of
an AS path to reach \( p \). Before the failure of \( u \), the BGP system is in a stable state and the routing graph is a DAG.

Since \( u \) is the in uphill portion of an AS path, \( u \) has only provider path to \( p \) and \( u \) never announces its path to providers/peers (valley-free policy). The failure of \( u \) affect only the direct/indirect customers of \( u \). Let \( C_s \) represent those direct single-homed customers of \( u \) and \( C_m \) represent those multi-homed direct customers of \( u \), which have policy compliant alternate paths after \( u \) fails. The routing graph changes from \( G \) to \( G' \) as following after the failure of \( u \). ASes in \( C_s \) remove their outgoing edges. Those ASes do not have alternate paths and there is nothing we can do. AS \( v \in C_m \) removes it outgoing edge pointing to \( u \) and add a direct edges pointing to another provider \( u' \). During this process, AS \( v \) does not have transient routing failure because it still has outgoing edge. AS \( v \) does not have transient loop either, because the result routing graph \( G' \) is still a DAG. Suppose \( G' \) has a cycle \( C \) after removing edge \( \overrightarrow{e} = (v, u) \) and adding edge \( \overrightarrow{e_1} = (v, u') \), we have \( \overrightarrow{e_1} \in C \). Since \( u \) never announces its path to providers, any AS using an path going through \( u \) must be a direct/indirect customer of \( u \). Since \( \overrightarrow{e_1} \in C \), \( u' \) is a direct/indirect customer of \( v \), which is a contradiction. Therefore, \( G' \) must be a DAG.

Recursively, we can prove that any AS \( v \) who uses a path with \( u \) in the uphill portion will not have transient routing loop or failure, as long as there still exists a policy compliant path from \( v \) to \( p \) after \( u \) fails.

The basic ideas of Lemma 3.3.1 are as follows. If AS \( V \) uses a provider route to reach a destination \( p \), \( V \) should have no peer or customer route to \( p \). Hence, during a change of the route to \( p \) at AS \( V \), e.g., after a route withdrawal, AS \( V \) cannot forward packets to \( p \) originating from its own customers “back” to lower tiers ASes because \( V \) never learned a route from them (note that since \( V \) does not have customer path to \( p \), it will not advertise \( p \) to its peers/providers, and so should never receive a packet addressed to \( p \) from them). Hence, transient routing loops will not occur. Since \( V \)
uses provider paths, the failure of \( V \) affects only \( V \)'s customers who use a provider path (via \( V \)). For those customers who have another path, their providers (other than AS \( V \)) should announce that path to them before. So they will not lose connectivity.

Lemma 3.3.1 states that complementary routing processes only need to focus on the downhill portions of the paths they select. As a result, section 3.4, which introduces our design for complementary routing processes, focuses on selecting paths that are downhill node disjoint.

Before proceeding with characterizing another set of events we need not be concerned with when it comes to their impact on routing, we highlight some implicit assumptions in the above discussion. Specifically, we have assumed that withdrawals propagate quickly, which is standard in current BGP implementations, and that transient problems (loops or failures), therefore, arise primarily because of latency introduced by timers that delay the propagation of route announcements. We believe this to be a reasonable assumption and an accurate reflection of when and why problems do arise during BGP convergence. Hence, problems in the uphill direction will not impact routing whenever a feasible alternate exists, i.e., it will be discovered and used very rapidly.

**Route additions and changes.** We now proceed to identify two other categories of events, namely, route addition and route change events, for which it can also be established that transient problems cannot arise. Intuitively, adding a link gives BGP more choices in selecting paths, which cannot cause routing failures (note that we are considering eBGP). Provided that adding the link does not violate the cycle-free property of AS relationships, no transient loop can arise during convergence either. Similarly, an AS changing its best path selection will not result in transient failures, since all ASes still have their paths. The reason for the absence of transient loops is that the path change remains compliant with common routing policies. So that the direction (either uphill or downhill) of each AS' path does not change. We formally
state those properties in Lemma 3.3.2. Note that we consider eBGP only. This lemma does not hold for iBGP.

**Lemma 3.3.2** No transient routing loop or failure will occur after a route change event or route addition event.

**Proof** We first consider route addition event caused by adding a link between two ASes. Route addition event caused by AS policy change can be proved similarly.

Suppose a link is added between AS $u$ and AS $v$. Without losing generality, we consider how AS $u$ and its neighbors reach destination $p$ only. If AS $u$ selects $v$ as its next hop and $v$ is a customer, $u$ will announce that path to all of its neighbors. AS $u$ will not withdraw any path. If AS $u$ selects $v$ as its next hop and $v$ is a peer or provider of $u$, $u$’s old path must be a peer path or a provider path. AS $u$ will announce the new path to its customers. AS $u$ will not send any withdrawal message to its peers and providers, since $u$ did not announce a path to them before (valley free path policy). Since no withdrawal message is sent out, all nodes still have outgoing edges in the routing graph of $p$. Transient routing failure does not occur.

Now we consider transient routing loop. If $u$ selects the path announced by $v$ as its best path and its is a provider/peer path, $u$ will announce that path to its customers. So, an AS $u'$, which is the direct/indirect customers of $u$, may switch from one provider path to another provider path. According to the proof of Lemma 3.3.1, no transient routing loop will occur when an AS switches from one provider path to another provider path. Suppose at some time the BGP system is in state $S'$ and the routing graph $G'$ is a DAG. For each direct/indirect provider $w$ of $u$, it either switches from a provider path to a customer path via $u$ or switches from one customer path to another customer path. For both cases, the other end of $w$’s outgoing edge is changed from $x$ to a customer $y$. The outgoing edge of AS $y$ should point to a customer of $y$, otherwise it violates the valley free path policy. So $w$ changing its outgoing edge
makes the routing graph transit from $G'$ to $G''$ and $G''$ is still a DAG, otherwise, we can derive that $y$ is a provider of $w$, which is impossible. Therefore, no transient routing loop occurs.

In case of a route change event, we consider an AS $u$ changes from a customer path to another customer path. AS $u$ changes from a provider (or peer) to another provider (or peer) path can be provided similarly. Note that AS $u$ cannot change from a customer path to a provider (peer) path if that customer path is still available, since it violates the prefer-customer policy.

In the initial state $S$ the routing graph $G$ is a DAG. The outgoing edge of $u$ points to customer $v$. At some time, node $u$ changes its outgoing edge and points to another customer $v'$. Since no AS loses path, every node in the new routing graph $G'$ has an outgoing edge in this new state $S'$. If there is a cycle in $G'$, that cycle must have edge $(u, v')$ in it. Since $u$ changes to a customer path, we can derive that $u$ is a customer if $v'$, which is not possible. So there is no cycle in $G'$. Similarly, we can prove an AS changes from a provider/peer path to another provider/peer does not create transient loop/failure either.

Lemma 3.3.2 highlights that route addition and route change events are not events one really needs to be concerned with in designing complementary routing processes.

3.3.4 From Control Plane to Data Plane

As mentioned earlier, our approach relies on both a control plane and a data plane component. Complementary routing processes offer the possibility of uninterrupted packet delivery in the presence of failure, by ensuring that one of the paths is always unaffected by the failure. Hence, by simply switching to the unaffected path, packets can avoid transient failures or loops. The simplicity of this statement notwithstanding, it hides several challenges beyond the design of complementary routing processes. First and foremost are the criteria that trigger switching from the current
path to one computed by another routing process. According to the properties we have identified via Lemma 3.3.1 and Lemma 3.3.2, only route withdrawal events can cause transient problems. Therefore, we can embed the "event type" information into each update message\(^4\) to assist in the detection of a possible failure on the current path and the identification of a complementary one. A closely coupled issue is the translation of the path switching decision into corresponding packet forwarding actions. We expand on these issues in section 3.5, while the next section is devoted to our design of complementary routing processes. The discussion in the rest of this chapter is limited to two complementary routing processes in each AS, a red process and a blue process that correspondingly compute red and blue paths. Those two paths must in turn be complementary under all single AS-level changes.

### 3.4 A multi-process routing protocol

We first introduce our scheme, the Selective Announcement Multi-Process routing protocol (STAMP), and then formally establish its properties.

#### 3.4.1 The STAMP Protocol

Each AS has two routing processes, red and blue, running in parallel. The red (or blue) process selects path among those announced by the red (or blue) processes of neighboring ASes.

As indicated in section 3.3.3, our design goal for STAMP is to ensure that red and blue paths are downhill node disjoint. A straightforward approach is to have the red and blue processes locally coordinate their choices, and simply select the two most downhill node disjoint paths among the candidates they receive from neighbors. However, such "local" greedy decisions may not be good "globally". That is, a lo-

\(^4\)Note that this calls for just “one bit” of information, which is a far cry from what is required to support RCI.
cal greedy choice might result in neighboring ASes not having any downhill disjoint paths. This problem can be eliminated through the use of more sophisticated algorithms, e.g., [95,98], but this comes at the cost of significant added complexity, and more importantly major changes to the BGP protocol; something we want to avoid. Therefore, we opt for a simpler alternative, where we perform selective announcements of paths by the two processes to control their propagation, while essentially preserving the standard BGP path selection process. This ensures that we impose only minor changes to existing BGP implementations, and as we will demonstrate through analysis as well as experiments, is quite effective in achieving our goal of providing downhill node disjoint paths.

Next, we describe the rules that govern these selective path announcements. Note that since we only require disjointness in the downhill portion, path announcements to peers and customers are not selective. Therefore, selective path announcements are required only to providers. Let us first consider how a multi-homed AS (i.e., an AS with multiple providers) announces its own prefixes. The AS selects a subset of its providers as blue providers, and announces its prefixes to these providers through its blue process only, and to rest of its providers through its red process only. This “splitting” of the colors with which the origin AS announces its prefixes to the providers ensures that the red and blue paths for each prefix reach it through different last hop providers. For a single-homed origin ASes, the splitting occurs at their first multi-homed direct/indirect provider. Note that in spite of the splitting, all prefixes are announced to all providers and only their color is affected. As we shall see, because STAMP does not impose any constraint on an AS on which colored path is its “primary” path for a prefix, the splitting of announcements on the basis of colors should have a minimal or no effect on the ability of an AS to use the same path as BGP would have selected to reach a prefix.
Having described how announcements initially proceed from the origin AS, we turn next to how transit ASes announce paths to prefixes they do not originate, i.e., receive from customers. The goal is for them to announce either red or blue routes to providers so that it is impossible for any direct/indirect provider to have both red and blue paths going through them. Clearly, if a transit AS has only red (blue) customer routes, it announces a red (blue) route to all providers. When a transit AS has both red and blue customer routes, it then has to decide which process announces to providers. One approach is to give the blue one a higher priority and announce only the blue route to providers when there are both red and blue customer routes. However, such a strict priority can severely affect the odds that a red path can propagate to all ASes. To reduce the likelihood of such an outcome, we introduce two measures. First, we require that in its initial split between blue and red providers, the origin AS selects a single blue provider. Second, to facilitate the propagation of red paths to as many providers as possible while ensuring the propagation of at least one blue path, we impose that this one blue provider to be a “locked” blue provider. This locking is indicated through a new path attribute, Lock, which takes a value of 1 when set. When the origin AS of a prefix announces it to its unique locked blue provider, it sets Lock = 1. Upon receiving a blue path with Lock = 1, the provider proceeds to announce this blue path to all its own providers. However, only one such announcement has its Lock attribute set to 1 (to the locked provider), and all others will have their Lock = 0. Second, a transit AS that received a blue path announcement with its Lock = 0 is not required to propagate it further if it also has a red path for the same prefix.

The main purpose of the Lock attribute is to ensure at least one provider propagates the blue path, and that red paths are propagated whenever a blue path is received with Lock = 0. Of course, if an AS has only a blue path, that path will be propagated anyhow. This ensures that all customer paths are propagated to providers.
Since route exchanges between ASes are asynchronous, an AS has to decide which process announces to providers based on what routes it has received so far. Therefore, it can happen that the AS needs to backtrack its decisions. If the red routing process of the AS announced route to providers and its blue path learns a customer route with $Lock = 1$. The red process should withdraw its routes from providers and let the blue process proceed (the red process does not need to withdraw routes announced to customers and peers since only announcing to providers is selective). Similarly, if the blue process of an AS has customer routes with $Lock = 0$ only and latter the red process learns a customer routes, the blue process should withdraw its routes from providers and let the red process proceed.

### 3.4.2 Properties of STAMP

Having described STAMP, we next establish its safeness before showing that the two paths discovered by STAMP are complementary in the sense that under any single routing event (as defined earlier), at least one of the processes does not experience transient problems. We will also present a necessary and sufficient condition for all ASes to have both red and blue routes.

To show that STAMP is safe, we first note that the only difference between a routing process in STAMP and a BGP process is that a STAMP routing process selectively announces routes to providers. Selectively announcement only limits the route announcement. As such, each STAMP routing process is safe as long as the prefer-customer and valley-free policies are followed. Therefore, the conclusion that STAMP is safe readily follows from BGP’s safeness.

In exploring the complementarity of the red and blue processes, note that the two processes never announce their best routes to the same providers. Hence, if both the red and blue routing processes of an AS have paths to a prefix, the paths must be downhill node disjoint paths. Based on Lemma 3.3.1, red and blue routing
processes are, therefore, complementary under any one route withdrawal event. Similarly, Lemma 3.3.2 tells us that the red and blue routing processes are complementary under any route addition or change event. Therefore, we have Theorem 3.4.1.

**Theorem 3.4.1** Under any one routing event, the red and blue routing processes in STAMP are complementary.

**Proof** We define the state of an AS. If an AS is selected by one of its customers at a locked blue provider, we define it is in state \( \text{LB} \). If an AS is not selected as a locked blue provider, and it announces blue (or red) path to providers, that AS is in state \( B \) (or \( R \)).

We first consider the route addition event caused by adding a link between two ASes. Route addition event caused by policy change can be proved similarly. Suppose a link is added between AS \( u \) and AS \( v \).

- **AS \( u \) is a provider of AS \( v \):** 1) If \( u \) is in state \( R \) and \( v \) is in state \( B \), AS \( u \) announces both red and blue (if \( u \) has one) paths to \( v \); AS \( v \) announces a blue path to \( u \). Both \( u \) and \( v \) do not withdraw any path so no transient problem will occur for either of them. 2) if \( u \) is in state \( B \) and \( v \) is in state \( R \), after the link between \( u \) and \( v \) is added, \( u \) withdraws its blue path and announces the red path learned from \( v \). It is equivalent to have a route withdrawal event in blue process and a route addition event in red process. The red process will not have any transient problems according to Lemma 3.3.2. 3) For other combinations of the states of \( u \) and \( v \), neither the red process nor the blue process will withdraw any paths, adding link \((u, s)\) is a route addition event for both routing processes. So none of them will have transient problems.

- **AS \( v \) is a provider of AS \( u \):** We ignore this case because it is equivalent to the previous case.
• AS \( u \) and AS \( v \) have peer-peer relationship: In this case, neither \( u \) nor \( v \) has new customer path. Therefore, none of them will change state so that no path will be withdrawn. Both red and blue routing processes will not have transient routing problems.

Second, we consider the route withdrawal event in which an AS \( u \) fails all its links. It is equivalent to a node failure event in which node \( u \) fails.

• AS \( u \) has red customer path and blue provider/peer path: For blue routing process, the \( u \) use provider path. Therefore, \( u \) is not in the downhill portion of any red path. According to Lemma 3.3.1, the blue process will not have any the failure of \( u \) will not create transient problems.

• AS \( u \) has blue customer path and red provider/peer path: This case if equivalent to the previous case.

• Both red and blue processes of \( u \) use provider/peer paths: For both red and blue processes, AS \( u \) is in the uphill portion, the failure of \( u \) will not cause transient problems.

• Both red and blue processes of \( u \) have customer paths: In this case, only one routing process of \( u \) announces to providers (assuming it is the red process). The blue process of \( u \) announces to peers and customers only. Therefore, \( u \) is not in the downhill portion of any blue path. The failure of \( u \) will not create transient problems for blue process.

One special case is that an AS \( u \) fails multiple connected links and AS \( u \) is the source AS we are considering. In this case, AS \( u \) will not have transient failure.

Proving the safeness is rather straightforward according to [27, Theorem 5.1]. Although each routing process selectively announces paths to providers and peers,
since all ASes still prefer customer path and no valley path is allowed, the routing system is safe.

Theorem 3.4.1 notwithstanding, we note that while selective announcement is effective in ensuring that the two processes are complementary, it does not guarantee that all ASes have both red and blue routes to a given prefix. Nevertheless, since each AS has a locked (blue) provider, we know that a blue route must propagate to all ASes (although obviously not always as a customer route). In other words, STAMP ensures that all ASes have a blue route for a given prefix. The same guarantee, however, does not apply to red routes. This is because a locked blue route can block red routes from propagating upward to providers (and eventually reaching a tier-1 AS). Next, we formally study under what conditions ASes can be guaranteed to have both red and blue paths for a destination prefix. In section 3.6.1 we perform an experimental study over today’s Internet topology to evaluate the actual likelihood of ASes having both red and blue paths when using STAMP.

In order to make sure that all ASes have both red and blue routes, we need to guarantee that they have a red route. Since STAMP already announces its red routes to all customers and peers, we only need to ensure that the red route is propagated to a tier-1 AS. Therefore, we have the following Lemma:

**Lemma 3.4.2** For any destination prefix, all ASes have downhill node disjoint red and blue routes, if and only if at least one tier-1 AS has a red customer route.

**Proof** Since a blue path is always announced to wards uphill direction, the blue process of each AS must have a blue path because announcing to peers and customers are not selective. Similarly, if the red process of a tier-1 AS has a red path, the red processes of all ASes can have a red since we do not selectively announce to peers and customers.
Next, we derive conditions for ensuring that at least one tier-1 AS has a red customer route. First, we derive a necessary condition for a red customer route to be propagated to a tier-1 AS. From Lemma 3.4.2, one can see that the sub-graph between the destination AS and the tier-1 ASes is crucial in determining whether all ASes can have red and blue routes. We formally define this sub-graph as the AS hierarchy graph as follows. For a destination AS $u$, the AS hierarchy graph of $u$, $G(u)$, is a sub-graph of the AS topology graph that includes only the direct/indirect providers of $u$ and the customer-provider (directed) links between them. In order to ensure that a red route is propagated to a tier-1 AS, it is necessary that there are at least two node-disjoint paths from $u$ to tier-1 ASes, i.e., the min-cut of the AS hierarchy graph is at least two. We formally state this in Theorem 3.4.3.

**Theorem 3.4.3** For any destination AS, $u$, if all ASes have both red and blue routes to $u$, then the minimum cut of $G(u)$ is at least two.

**Proof** Proving the existence of a cut vertex being a sufficient condition is straightforward, since all paths from $p$ to tier-1 ASes must include that cut vertex. Therefore, we can not find two node disjoint paths.

Now we assume tier-1 ASes (we can image all tier-1 ASes as one “super-node”) do not have two node disjoint paths to reach $p$ and there is no cut vertex in $p$’s AS hierarchy graph. If tier-1 ASes do not have two node disjoint paths to $p$, the cut between tier-1 ASes and node $p$ is 1. Therefore, there must exist a cut vertex according to the max-flow min-cut theorem. Here comes a contradiction. So existence of a cut vertex is also a necessary condition.

Although a min-cut of at least two is a necessary condition for all ASes to have both red and blue routes, it is obviously not sufficient. Recall that the locked blue provider is determined in a distributed manner. The wrong selection of a locked blue provider might block all red customer routes. We therefore derive next a necessary
and sufficient condition for all ASes to have both red and blue routes. That is, if the locked blue providers do not block all possible paths that a red route could propagate towards a tier-1 AS, then because the red route propagation has the next highest priority after the propagation of the locked blue route, this ensures one of the remaining open paths to a tier-1 AS is discovered. Hence, all ASes will have both red and blue routes. We formally state this in Theorem 3.4.4.

**Theorem 3.4.4** *For any destination AS, u, if u is connected with at least one tier-1 AS after removing from G(u) all locked blue direct and indirect providers of u, then all ASes have both red and blue routes to u.*

**Proof** If p is in a connected component C of the remaining graph and C has at least a tier-1 AS, there must be a path from p to some tier-1 AS and that path has customer-provider links only. So the remaining graph being connected is a sufficient condition.

Now we assume tier-1 ASes do not have another path to reach p after removing all locked blue providers and the customer-provider links between them, but p is in a connected component C which as at least a tier-1 AS. Because C has only direct/indirect providers of p, there must exist a policy compliant path from p to that tier-1 AS in C, which uses customer-to-provider links only. Here comes the contradiction. So C has at least a tier-1 AS is the necessary condition as well.

Intuitively, given the Internet topology, the necessary condition in Theorem 3.4.3 is extremely likely to hold. We confirm this for the current Internet topology in section 3.6.1 (it is true for around 98.5% of destination ASes). We also expect the sufficient condition in Theorem 3.4.4 to be satisfied in most cases, even under random selection of locked blue providers. In fact, as shown in section 3.6.1, under a random selection of locked blue providers, around 92.0% of destination ASes satisfy the sufficient condition. In addition, we will show that this percentage can be further
improved to about 98.2% simply by having the origin AS “intelligently” select its locked blue provider. This is close to the maximum possible figure of 98.5%, which accounts for the fact that irrespective of the scheme used to select blue and red routes, it is impossible to find downhill node disjoint paths for about 1.5% of all ASes.

Before proceeding with describing in the next section how red and blue routes are used to forward packets, we would like to emphasize that although STAMP performs selective announcement, STAMP can always ensure that one of its routing process selects a customer route if there is a customer route. That is, the selective announcement does not limit the chance that an AS can use a customer route. This is because one of the routing processes always announces its route to providers. Therefore, as we will see in the next section, we will leave each source AS to determine which routing process to use when sending the packets.

3.5 Data plane design

Once STAMP has computed routes, how these are used in forwarding packets is obviously of importance. In this section, we present a data plane design that addresses this issue.

3.5.1 Packet Forwarding

Under normal conditions, packets should be forwarded consistently using routes computed by routing process of the same color. For that purpose, we define a color bit in the packet header, indicating whether the packet should be forwarded using the red or blue routes\(^5\). This color bit can be set by the source AS. The color bit might be changed by a transit AS upon detecting that the corresponding route is currently

\(^5\)Note that this could also be realized using virtual interfaces based forwarding [94], with the addition that routers on each side should agree to the binding of the interface to a particular color. There is no conceptual difficulty to that approach, but using virtual interfaces and tunnels introduces its own overhead and complexity.
experiencing instabilities (more on this below). Such a change should, however, be performed only once to avoid potential loops [99]. We therefore define another bit in the packet header, a change bit, to record whether the color bit of a packet has been changed or not. Two of the DS (Differentiated Services) bits in IP header could be used for that purpose.

Allowing the source AS to determine the color of its packets has a number of implications. First, transit ASes can receive packets from either color for a given destination, and must then be able to forward them using either route. There is clearly a cost to such a capability; one that goes beyond examining extra bits in packet headers and that extends to storing additional information in forwarding tables, i.e., the next hop(s) associated with routes of different colors. While this certainly represents an overhead, it can be kept relatively small through the use of intelligent data structures. Second, it is possible to ensure that whenever there is a customer route to deliver a packet, the source AS can choose the color that provides a customer route. Furthermore, it is possible to ensure that any transit AS does not have to use a provider route to deliver packets whenever a customer route exits, if the source AS selects its default route carefully.

3.5.2 Switching between Routing Processes

The previous section dealt with implementing the forwarding decisions given the routing process (or color) of the packet. However, it is possible to change the routing process used (or color) if the current process potentially experiences transient problems. In this section, we focus on how this switching is performed. Recall that our goal is to always switch to a routing process without transient problems. Although the idea is straightforward, we need to address two issues in designing such a switching mechanism; (1) detecting problematic behavior in routing processes, (2) identifying the routing process that does not experience transient problems.
Detecting potential transient problems. ASes adjacent to where a routing event originated, e.g., a link/node failure/recovery, can easily detect and identify the root cause of the event. For example, a link addition/route change will not cause transient problems but a link failure would, and the first AS heard of the event has the necessary information to tell whether the event can lead to transient problem. ASes not directly adjacent to where the routing event occurred, need to rely on update messages to infer potential transient problems. According to Lemma 3.3.2, transient problems arise only when a routing process loses some routes. Therefore, in order assist in “recording” if the routing event that originally triggered an update was associated with a loss of a route, we add a new path attribute $ET$ (Event Type) to update messages. The $ET$ attribute is 1-bit of information that indicates whether the message was caused by losing a route ($ET = 0$) or not ($ET = 1$).

In STAMP, updates can be sent not only because of events that affect one process, but also because of interactions between processes\(^6\). Rules for setting the $ET$ bit must, therefore, account for these different scenarios. Withdrawal messages all have $ET = 0$, irrespective of their cause. If a process generates an update message for a route because of an adjacent link/node failure/recovery or policy change, i.e., it is the origin AS of the event, the update message has $ET = 1$ if it is a recovery; and $ET = 0$ if it is a failure. If a process announces an update because itself received an update message $M$ from one of its neighbors that resulted in a change of its best route or a new route, the process copies $M$’s $ET$ attribute into the update messages sent to its neighbors. Finally, if a routing process of an AS announces a route to a neighbor, because the other process withdrew its route, the update message has $ET = 1$.

\(^6\)For example, an AS used to announce an unlocked blue route to providers, but after learning a red customer route, it should announce this red route to providers and its blue process must withdraw its blue route.
The switching mechanism. Given the availability of the $ET$ attribute, we have the following mechanism for switching routes. If an AS is using the best route computed by one process and that process loses that route (receives an update message with $ET = 0$ or the adjacent link/node fails), the AS switches to the route selected by the other process. If both processes receive update messages with $ET = 0$, either process that still has a route can be used.

Changing the packet color should be “temporarily”. Once the routing process having transient problems converges and has a new path to the destination, an AS should stop changing the color of packets to that destination. Although the intuition behind this statement is simple, implementing it is not trivial because it is hard to decide when a routing process has converged or not. We adopt a heuristic in which a switch back timer $t$ is used. If a routing process has not had dynamics for $t$ seconds, we assume it has converged. Instead of setting $t$ to a constant value, we can use mechanisms such as an exponential backoff algorithm to dynamically adjust $t$.

### 3.5.3 Effectiveness of the Switching Mechanism

With the above switching mechanism and the downhill node disjoint paths computed by STAMP, we can achieve reliable packet forwarding. In case of a single routing event, the only disruption in packet forwarding can be as short as the time needed for ASes adjacent to the routing event (such as a link failure) to detect that event. The effectiveness of this switching mechanism is formally established in Theorem 3.5.1.

**Theorem 3.5.1** In case of a single routing event, no packet will be looped or black-holed after ASes adjacent to where the routing event occurs detect that event.

**Proof** We consider ASes who have received routing updates caused by the routing event and ASes who have not yet separately.
First, if a policy compliant path to destination $p$ still exists after the routing event occurs, each AS should have at least one routing process which yields a route to destination $p$, because red and blue processes in the STAMP scheme complement each other (Theorem 3.4.1). When an AS $A$ detects the $EventType$ attribute in routing update message received by one process (suppose the red process) is route withdrawal, it will switch to the route computed by the blue process. All packets from $A$ to $p$, including packets from other ASes and forwarded by $A$, will be delivered according to the route computed by the blue process of each intermediate AS.

Second, after the actual routing event occurs, for example a link is down, and before $A$ receives any update message relevant to that event, $A$ continues forwarding packets according to the route computed by it current routing process (suppose it is red), which may have transient loop/failure. Those packets can also be delivered to destination $p$, as long as the ASes adjacent to the failed link have detected the failure. When the packets from $A$ are forwarded among the route computed by the red processes $A$’s downstream intermediate ASes, they will eventually arrive at some AS who has detected the unfavourable dynamics. That AS must change the color bit of those packets. From that point on, those packets will be forwarded according to the route computed by blue process of each AS. Since blue process complements the red process, those packets will be successfully delivered.

The basic ideas behind Theorem 3.5.1 are followings. According to Theorem 3.4.1, for one routing event, at least one routing process in STAMP has no transient problems. Also note only losing routes creates transient problems (Lemma 3.3.2). If only one routing process of an AS receives updates with $ET = 0$, the other one should not have transient problems. If both routing processes of an AS receive updates with $ET = 0$, the routing event must either 1) occurs in the uphill portions of both red and blue paths of that AS (e.g, those two paths share a link in the uphill portions and that link fails), or 2) that AS has multiple link failures. In the first case, neither pro-
cess has transient problems (Lemma 3.3.1). In the second case, suppose the AS has both red and blue customer routes. One of its routing process (e.g., red) must have a provider route since only one process announces to providers (here the blue one). Even if the AS loses all customer routes (the first links of both red and blue paths fail), one process still has a provider path which does not include the failed links. So the routing process that still has a route will not have any transient problem.

3.6 Experimental evaluation

Given that STAMP may not always succeed in discovering blue and red paths at all ASes even when they exist, we first evaluate STAMP’s performance along that dimension. Next, we compare STAMP with other schemes under various failure scenarios by simulations, and finally we study its benefits in the context of partial deployments.

In order to carry out meaningful and realistic evaluations, we conduct our experiments using BGP routing tables collected by the RouteViews project [90], which we use to construct an AS topology of the Internet. We infer the underlying AS relationships using Gao’s algorithm [5].

3.6.1 Evaluating STAMP’s Performance

Given our reconstructed AS topology, we proceed to evaluate the odds for ASes to have both red and blue paths to destination prefixes when using STAMP.

The metric. Can STAMP ensure that all ASes have both blue and red paths to a destination? This depends on the AS topology as well as on how the locked blue provider is selected at each AS. Assume that the locked blue provider is selected randomly among all providers of an AS. Given the AS topology, we can then compute the odds that all ASes have both blue and red paths to a destination. Let $\Phi_m$ be the probability that all ASes have both red and blue routes to multi-homed AS $m$,
and denote $\lambda$ as the number of all possible paths from $m$ to any tier-1 AS. If path $l_i$, $1 \leq i \leq \lambda$, is selected as the “locked blue path” from $m$ to a tier-1 AS and a disjoint path from $m$ to another tier-1 AS exists, we say that $l_i$ is a “good” locked blue path since we know that STAMP can still find a red path. If there are $\lambda'$ good locked blue paths, $\Phi_m = \frac{\lambda'}{\lambda}$. For a single-homed AS $s$, $\Phi_s = \Phi_m$ if $m$ is the first multi-homed (direct/indirect) provider of $s$.

**Destinations without downhill node disjoint paths.** For destination AS $k$, if $\Phi_k = 0$, then the min-cut of $G(k)$, the AS hierarchy graph of AS $k$, is one. That is, no matter how locked blue providers are selected, we cannot ensure that all ASes have both red and blue path to $k$ (Theorem 3.4.3). The number of ASes with $\Phi_k = 0$ is shown in Table 3.1. We can see that only a small number of ASes have AS hierarchy graphs with a min-cut of one. This means that the Internet AS topology is to a large extent diverse enough to provide downhill node disjoint paths to most ASes.

<table>
<thead>
<tr>
<th></th>
<th># of destinations with $\Phi=0$</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>single-homed</td>
<td>125</td>
<td>0.46%</td>
</tr>
<tr>
<td>multi-homed</td>
<td>289</td>
<td>1.1%</td>
</tr>
<tr>
<td>overall</td>
<td>414</td>
<td>1.56%</td>
</tr>
</tbody>
</table>

Table 3.1. Number of destinations without downhill node disjoint paths

**The distribution of $\Phi_k$ across all destinations.** In Figure 3.3, we plot the CDF (Cumulative Distribution Function) of $\Phi_k$ for all destinations. We see that less than 10% of destinations have $\Phi_k \leq 0.7$. Conversely, more than 75% of destination ASes have a probability greater than 0.9 that all other ASes can reach them through both red and blue paths. On average, all ASes have both red and blue paths to any destination AS with probability 0.92.

**Smart selection of first hop blue provider.** In the previous experiment, each AS selects its locked blue provider randomly. However, if the origin AS (or the first multi-homed provider if the origin AS is single-homed) “intelligently” selects its blue
provider, the chance that all ASes have red and blue paths to that destination can be improved. Specifically, assume there are \( \nu_a \) paths from \( k \) to tier-1 ASes via \( k \)'s provider \( a \); and \( \nu'_a \) of them are “good” locked blue paths. The probability that all ASes have red and blue paths to \( k \) when \( k \) selects \( a \) as its locked blue provider is then

\[
\Phi_k^a = \frac{\nu'_a}{\nu_a}.
\]

Assuming that AS \( k \) can compute \( \Phi_k^a \) for all its providers, something that can be done off-line relatively easily and periodically since topology does not change often, it can select the lock provider \( u \) that maximizes \( \Phi_k^u \).

We investigate the benefits of such an approach in Table 3.2, which reports the number of ASes for which \( \max(\Phi_k^u) = 1 \), i.e., are guaranteed to be reachable by all ASes through node disjoint red and blue paths. Note that while the table reports a total percentage of 97.3%, when accounting for the fact that about 1.5% of ASes cannot have two node-disjoint paths no matter how they are selected (Table 3.1), this improves to 98.8% of all ASes for which this is feasible.

<table>
<thead>
<tr>
<th></th>
<th># of ASes with max(( \Phi_k^u ))=1</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>single-homed</td>
<td>10160</td>
<td>38.1%</td>
</tr>
<tr>
<td>multi-homed</td>
<td>15794</td>
<td>59.2%</td>
</tr>
<tr>
<td>overall</td>
<td>25954</td>
<td>97.3%</td>
</tr>
</tbody>
</table>

Table 3.2. Number of ASes with max(\( \Phi_k^u \))=1
3.6.2 Performance under Failure—Comparison to Other Schemes

The previous experiments focused on evaluating STAMP’s ability to provide protections against any single routing event/failure, which does not account for the actual impact of each possible failure scenario, e.g., some failures may not have an impact even for ASes for which STAMP did not succeed in identifying both red and blue paths. In order to better assess STAMP’s actual benefits in the presence of failures, we developed an event-driven simulator to replicate routing dynamics. Our simulator is lightweight and highly efficient, which can simulate networks with thousands of ASes. We implemented BGP, R-BGP, and STAMP in the simulator. For all protocols, both processing and transmission delays are modeled by a random variable uniformly distributed in $[10 ms, 20 ms]$. The BGP MRAI timer is peer-based and its value is set to 30 seconds multiplied by a random factor uniformly distributed within $[0.75, 1.0]$.

**Single link failure.** We simulate routing convergence after a multi-homed AS fails one of its provider links. The destination AS is randomly selected across 100 simulation instances. The average (across all 100 scenarios) number of ASes having transient problems is shown in Table 3.3. BGP has more than 6,000 ASes experiencing transient problems. Although R-BGP handles single link failure very well, it requires RCI mechanism, which as argued earlier adds significant complexity to the routing system. Nevertheless, we include it as a benchmark against which to compare STAMP. Note that without RCI, R-BGP results in over 2,000 ASes being affected in some ways by failures. STAMP has about 350 ASes that experience transient problems. Considering that the actual Internet is likely to be more densely connected than the partial AS topology derived from BGP tables, STAMP should perform better in practice.

**Multiple link failures** Next, we consider scenarios where multiple links fail simultaneously (or policy changes affect multiple ASes). We distinguish between two cases: i) the two failed links are not connected to the same AS; and ii) the two failed links...
are connected to the same AS, which corresponds the case that one router within an
AS fails or an AS changes it policy resulting withdrawing route from two neighbors.
In the first case, an origin AS fails one of its provider links and another randomly
selected indirect provider link (multi-hop away from the origin AS). In the second
case, an origin AS fails a link to one of its providers and that provider also fails one
of its own provider links.

The average number of ASes experiencing transient problems are presented in Ta-
ble 3.4. When the two failed links are not connected to the same AS, both STAMP
and R-BGP perform similarly, while when the two failed links are connected to the
same AS, STAMP experiences about half fewer problems than R-BGP. This is be-
cause multiple link failures at the same AS correspond to a “single” routing event for
STAMP; something against which its node-disjoint path selection offers protection. A
similar set of conclusions hold in the presence of single node (AS) failures, which cor-
respond to an AS withdrawing a route from all its neighbors. Note that when R-BGP
is not afforded the benefit of RCI, its performance again degrades significantly.

**Single node (AS) failure** The other scenario we consider is node (AS) failure,
which means all links attached to that AS fail or an AS withdrawing its route from all
neighbors. The experiment results are presented in Table 3.5. In case of single node
failure, both BGP and R-BGP have a considerable number of ASes experiencing tran-
sient problems. The reason for R-BGP’s poor performance is that R-BGP heavily
relies on the provider of the origin AS to detour the traffic. If the provider AS fails,
a large number of ASes will have transient problems. Here, STAMP performs similar
to the single link failure because we let the origin AS “branch” its announcement to

<table>
<thead>
<tr>
<th># of ASes</th>
<th>BGP</th>
<th>R-BGP without RCI</th>
<th>R-BGP</th>
<th>STAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>percentage</td>
<td>24.76%</td>
<td>7.86%</td>
<td>0%</td>
<td>1.34%</td>
</tr>
</tbody>
</table>

Table 3.3. Number of ASes having transient problems in single link failure
(a) two failed links are not connected to the same AS

<table>
<thead>
<tr>
<th></th>
<th>BGP</th>
<th>R-BGP without RCI</th>
<th>R-BGP</th>
<th>STAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td># of ASes</td>
<td>10314.5</td>
<td>4242.6</td>
<td>861.4</td>
<td>845.7</td>
</tr>
<tr>
<td>percentage</td>
<td>38.66%</td>
<td>15.9%</td>
<td>3.22%</td>
<td>3.17%</td>
</tr>
</tbody>
</table>

(b) two failed links are connected to the same AS

<table>
<thead>
<tr>
<th></th>
<th>BGP</th>
<th>R-BGP without RCI</th>
<th>R-BGP</th>
<th>STAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td># of ASes</td>
<td>12071.2</td>
<td>3803.4</td>
<td>761.4</td>
<td>366.8</td>
</tr>
<tr>
<td>percentage</td>
<td>45.2%</td>
<td>14.3%</td>
<td>2.85%</td>
<td>1.49%</td>
</tr>
</tbody>
</table>

Table 3.4. Number of ASes having transient problems in multiple link failures

different providers so even one provider totally fails, other ASes still have another path computed by one of their routing processes.

<table>
<thead>
<tr>
<th></th>
<th>BGP</th>
<th>R-BGP without RCI</th>
<th>R-BGP</th>
<th>STAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td># of ASes</td>
<td>7721.8</td>
<td>3376.5</td>
<td>2504.2</td>
<td>327.5</td>
</tr>
<tr>
<td>percentage</td>
<td>28.94%</td>
<td>12.7%</td>
<td>9.39%</td>
<td>1.23%</td>
</tr>
</tbody>
</table>

Table 3.5. Number of ASes having transient problems in single node failure

From the simulation studies, we can conclude that STAMP performs much better than standard BGP in all failure scenarios we considered. Although R-BGP handles single link failure very well, it requires RCI information, which is not trivial to implement. Without RCI, R-BGP performs much worse than STAMP in all failure scenarios in our simulations. For multiple link failures, depending on the locations of those failed links, STAMP performs comparable or better than R-BGP. In case of single node failure, STAMP performs much better than other schemes.

3.6.3 Evaluating Partial Deployment

The previous sections demonstrated STAMP’s ability to provide most ASes in the Internet downhill node disjoint paths to most destinations with minimal impact to the operation of the current BGP protocol, and more importantly to eliminate a majority of transient problems appearing in BGP convergence. In this section, we turn to the
important practical issue of incremental deployment. Specifically, the previous results assume that all ASes in the Internet deploy STAMP, but this is unlikely to occur, at least not immediately. A natural question is, therefore, to ascertain how much of these benefits remain if STAMP is deployed only in a fraction of all ASes. The natural candidates for such an initial deployment are tier-1 ASes, since they are the core of the Internet and responsible for delivering a significant percentage of the Internet traffic. We conducted a set of experiments to evaluate STAMP’s “coverage”, i.e., to how many destination ASes tier-1 ASes have two downhill node disjoint paths, if only tier-1 ASes adopt STAMP.

Since we assume that no customer AS feed tier-1 ASes with “colored” routes, the tier-1 ASes need to assign colors to their routes. We assume each tier-1 AS $T$ acts according to the following rules in assigning colors to its customer routes: If $T$ has two disjoint customer routes to a destination, $T$ randomly assigns colors to them (one red and one blue) and announces them to other tier-1 ASes; If $T$ does not have two disjoint customer routes to a destination, $T$ randomly assigns a color to its best route and announces it to other tier-1 ASes.

![Figure 3.4](image_url)

**Figure 3.4.** Percentage of non-tier-1 ASes to which each tier-1 AS have two downhill node disjoint paths under incremental deployment of STAMP (at tier-1 ASes only).

Ten ASes were selected as tier-1. For each of them, we count the number of destinations for which they have two downhill node disjoint red and blue routes.
The results are shown in Figure 3.4. Note that even under our simple random color assignment rule, all tier 1 ASes have two downhill disjoint paths to more than 70% of destination ASes. On average, tier-1 ASes have two downhill disjoint paths to about 75% of destination ASes.

3.7 Conclusion

The chapter proposed a multi-process routing solution, STAMP, to mitigate transient problems experienced by today's inter-domain routing. STAMP seeks to accomplish this while requiring minimal changes to the current inter-domain routing protocol, BGP, and its implementations. STAMP is based on running two slightly extended BGP processes in each AS, which compute complementary AS routes. The chapter establishes that this can be realized by focusing only on the downhill portion of AS paths, and using a simple heuristic for path selection. STAMP was evaluated through extensive experiments, which showed that compared to BGP, it could yield substantial improvements in routing stability. These improvements were comparable, and for some important failure scenarios, better than those of previous proposals that also called for more extensive and potentially complex modifications to BGP. Equally if not more important, STAMP can be deployed incrementally across the Internet, and we showed that its deployment at tier-1 ASes only could already deliver a significant improvement in Internet routing stability.
CHAPTER 4

PDP: PARALLELIZING DATA PLANE IN VIRTUAL NETWORK SUBSTRATE

4.1 Introduction

Network virtualization provides a powerful way to facilitate testing and deploying network innovations over a shared substrate. Currently the network research community is focusing on building a shared, wide-area experimental platform to support a broad range of research in networking and distributed systems. To that end, and more importantly, toward the long term goal of providing a global infrastructure in which multiple virtual networks, each customized to a specific purpose, could run concurrently, the virtual network substrate must have four key properties: (1) isolation between virtual networks to minimize the interference among them; (2) flexibility to customize the virtual networks; (3) high-speed data plane packet processing performance to facilitate realistic experiments and attract long term applications; and (4) low cost in building that platform to lower the barrier of wide-area deployment.

The challenge of building such a virtual network substrate is that the four properties, i.e., isolation, flexibility, high performance, and low cost, are often tightly coupled issues in system design so that usually we have to compromise one in order to improve another one. For example, special purpose hardware can achieve better packet processing performance but it can cost significantly more than commodity hardware. Another dilemma is that in order to achieve better performance, the data plane functions of a virtual network should have direct access to the hardware or run in the privileged domain of the hardware. However, opening low-level and close-to
hardware programming interfaces usually results in peer isolation among virtual networks. A buggy function implemented in one virtual network can crash the whole system, e.g., shut down a machine hosting multiple virtual networks. Or a malicious user of the virtual network platform can easily affect other virtual networks residing at the same substrate. In order to prevent such a situation from happening but still offering the desired performance benefits, Prior works [48,100] propose to design a set of well-tested building blocks which have direct access to the hardware or run in the privileged domain of the hardware. Virtual networks can assemble those building blocks to implement desired functions. However, that compromises the flexibility because the virtual networks are limited to the set of provided building blocks.

In this chapter, a virtual network platform called PdP is presented [50,78]. PdP is built from commodity hardware. The basic ideas behind the design of PdP are two-fold. First, both the control plane and data plane of a virtual network run in virtual machines to provide the isolation among virtual networks and the flexibility to customize each virtual network. Second, there are multiple physical machines serving as “forwarding engines” in a PdP node. To achieve high speed packet processing, a virtual network can have multiple virtual machines (hosted by the forwarding engines) running in parallel to serve as its data plane. Note that it is possible to combine PdP with virtual network platforms based on special purpose hardware, in which case PdP supports highly customized virtual network service; while the special hardware based data forwarding elements such as network processors support virtual network services that can be composed with the set of building block provided by the network processors. Therefore, PdP can complement the special purpose hardware based solutions.

Although the basic idea behind PdP is promising, implementing this platform is challenging. First, for a PdP node, it is important to ensure that the packet processing performance scales with the number of forwarding engines. The machine
which coordinates the forwarding engines should not become the bottleneck. Second, parallel packet forwarding by multiple forwarding engines can lead to out-of-order packets and thereby resulting in degraded performance for the up-layer applications such as applications using TCP. Therefore, it is important to reduce the amount of out-of-order packets.

In summary, we make three main contributions in our work. (I) To the best of our knowledge, PdP is the first virtual network platform which provides both high degree of customization and viable data processing performance. (II) PdP is the first platform demonstrating the scalability of parallelizing packet processing in virtual networks. (III) We have built a proof-of-concept PdP node prototype using off-the-shelf commodity hardware and open source software. Our experiments show promising results.

The rest of this chapter is organized as follows. Section 4.2 presents related work on virtual network platforms. Section 4.3 details the design of PdP. Section 4.4 presents the experiment evaluation results. Section 4.5 concludes this chapter and projects our future work.

4.2 Related Work: Virtual Network Platform State of Art

In this section we briefly review the existing virtual network platforms and point out the pros and cons of each platform.

4.2.1 VINI

The VINI platform is presented in [46]. Each physical node in VINI is virtualized in the operating system level. Operating system-level virtualization is where the kernel of an operating system allows for multiple isolated user-space instances (instead of just one). Such instances (often called containers, VEs, VPSs or jails) look and feel like a real server, from the point of view of its owner [101]. A user of the VINI platform
is allocated with a slice in a set of physical machines. A slice is basically a container. Using VINI platform, user can build overlay networks in which a virtual link is a UDP tunnel. The actual packet forwarding is done by Click modular router running in user mode. This design decision, although not efficient in terms of packet forwarding speed, is very feasible because of the customizability of Click. Besides, user can be free of customizing Click in various ways without interfering other users’ virtual networks, which is a much desired property for an infrastructure supporting concurrent multiple virtual networks.

4.2.2 Trellis

As a followup work of VINI, the Trellis platform [103] provides faster packet forwarding speed. Trellis still adopts the operating system level virtualization. That is, a user of Trellis is allocated with a set of containers in some physical machines. A virtual link in Trellis is an Ethernet tunnel. An ethernet tunnel is ended in the “root context” of the physical machine. The root context is outside of the virtual host container. That is, an ethernet tunnel is corresponding to a virtual interface in the kernel of the operating system of the physical machine. Therefore, packets of a virtual network are handled by the native operating system kernel of the physical machine. So the (aggregate) forwarding speed can approach the limit of the physical hardware. The design decision of Trellis is much performance oriented. The performance improvement of Trellis can not benefit users who need to customize their packet forwarding process. To fulfill customized packet forwarding, a user of Trellis has to run Click in user mode, which loses all the performance benefits of Trellis.

4.2.3 VRouter

The VRouter project [104] adopts a different form of virtualization technology, which is so called full virtualization [105]. In particular, the VRouter project uses the Xen [106] platform to implement virtual routers in commodity hardware [107].
Xen consists of a hypervisor running above the hardware. The hypervisor slices the machine into multiple domains, which are essentially virtual machines residing above the hypervisor, each running its own operating system. There is one Isolated Driver Domain, commonly known as dom0 in Xen, which has special privileges allowing it to host and execute the device drivers. Xen can have multiple Guest Domains. A guest domain, or domU has no direct access to devices and relays such access through dom0. The performance study in [107] shows that running each virtual router entity in one domU has unacceptable poor forwarding performance. However, offloading the forwarding plane of every virtual router into a single separate domain with direct I/O to all interfaces yields a viable solution in terms of packet forwarding performance. But doing that loses all the protection, isolation and flexibility afforded by Xen.

4.2.4 Supercharging PlanetLab

The Supercharging PlanetLab Platform (SPP) is presented in [100]. SPP uses network processor (NP) to implement its data plane. To enable multiple applications to use the network process resources concurrently, SPP supports both multiple NP subsystems and sharing of individual NP subsystems. SPP provides a generic application structure. Each application has a “slice manager” (SM) and a “fast path” (FP). The SM is the control plane of the slice and is running on a general purpose compute server. The FP is the data plane which handles packet forwarding and is implemented in network processor. The SM manages the FP through a generic control interface. The FP has some generic function blocks which the SM can control them to implement different forwarding scheme, such as IPv4 forwarding or DHT based forwarding. The generic function blocks include packet parse, lookup, and header format, which are mapped to different micro engines (ME) in network processor to achieve high throughput by pipelining the operations. For example, the lookup block
in FP provides a generic lookup capability, using TCAM. It treats the lookup key as an opaque bit string with 112 bits.

4.2.5 Source Code Merging

The source code merging scheme [48] provides a set of function elements which can run in the privileged domain of underlying hardware. As a result, a virtual network is limited to assemble its data plane using the provided elements.

4.2.6 Summary

All existing systems can be classified into two classes, i.e., software based systems (VINI and Trellis) and special hardware-enhanced systems (SPP or some systems using NetFPGA [108]). All systems have different tradeoff between flexibility and forwarding performance. The hardware-enhanced systems, although have much higher forwarding throughput, are usually not so convenient to support customized forwarding function. For example, SPP hardwires a “generic” forwarding function block into the network processor subsystem. Therefore, users are limited to make use of only what the forwarding function block can provide, which is basically a TCAM based lookup function. If the function of the hardware is not pre-defined and uses can program the hardware such as network processor or NetFPGA board [109], we will face two major problems. First, programming those hardware is usually difficult and requires additional learning or trading in coding. Second, which is more important, opening the hardware to user programming can make it hard to isolate one user from other users. For example, the buggy user code can halt the network processor or NetFPGA board which may require a reboot to restore working condition.

The throughput of software-based systems is limited by the native packet forwarding speed of commodity hardware (desktop or server class PC) and the operating system (Linux). However, as PC processor architecture are rapidly adopting the multi-core paradigm, we can expect software based systems will close the gap
between general purpose processor and special purpose processor such as NP or NetFPGA \cite{110}. Trellis and VRouter can approach the limit of native hardware forwarding speed. However, they have considerable constraints on customizing packet forwarding. If a Trellis user wants to do customized packet forwarding, he has to run user mode Click in his slice, which means small packet forwarding speed. Various studies have shown that user mode packet forwarding can hardly exceed about 50K packets per second. Running Click in kernel mode can achieve native forwarding speed. However, we usually cannot trust a customized Click module running in kernel mode because a code bug of the customized Click module can crash the physical machine, i.e., very poor isolation between different users. VRouter needs to put all data forwarding into dom0 in order to achieve acceptable forwarding performance, which also suffers from the poor isolation issue.

All existing systems can be classified into two classes, i.e., general-purpose hardware based systems (using PC and unix-like OS) and special-purpose hardware enhanced systems (using NP or NetFPGA \cite{109}). Among general-purpose hardware based systems, VINI suffers poor forwarding speed but Trellis and VRouter can approach the limit of native hardware forwarding speed. However, Trellis and VRouter have considerable constraints on customizing packet forwarding if good isolation between virtual networks needs to be retained. The special-purpose hardware enhanced systems, although have much faster forwarding speed, are usually not so flexible to support customized forwarding functions. Opening the hardware programming interface to users cannot help too much because \((a)\) programming those hardware is usually difficult and requires additional learning or trading in coding; \((b)\) more importantly, opening the hardware to user programming can make it hard to isolate one user from others. The buggy user code can halt the network processor or NetFPGA board so that a reboot would be required to restore working condition.
Compared with the other platforms, PdP provides good isolation and flexibility properties with little packet processing performance compromise and affordable cost increasing.

4.3 The Design of PdP

In this section, we first describe the basic ideas behind PdP. Then we present the design of PdP in details.

4.3.1 Basic Ideas

The design goal of PdP is to provide maximum flexibility and isolation to virtual networks with minimal compromise in packet processing. For a virtual network, both the control plane and the data plane run in guest machines and the virtualization mechanism (which slices a host machine into one or more guest machines) provides the necessary isolation among different virtual networks. Running the control plane and the data plane in guest machines has certain overhead. Although this overhead may not be an issue for the control plane functions, it can significantly degrade the data plane performance, because essentially the packet processing functions run in the unprivileged domain of the underlying hardware. To compensate the performance degradation of running the data plane in guest machine, we assign one virtual network multiple guest machines to perform the packet processing task. With the parallel processing in multiple guest machines, a virtual network in PdP can achieve better data plane performance than the virtual networks in other platforms with similar degree of isolation and flexibility. In other words, PdP trades cost (having multiple physical machines to perform the data plane tasks of virtual networks) for better flex-

How many guest machines should be assigned to one virtual network and how much packet processing power one guest machine should have depend on the requirement the virtual network.
ibility, isolation, and performance. Since PdP is built from cost-efficient commodity hardware and open source software, the cost increasing should not be substantial.

4.3.2 PdP Node Architecture

A PdP node actually consists of a cluster of machines. One of the machines is the management host (denoted by MH) and there are multiple machines acting as the forwarding engines (denoted by FEs). A multiplexer/demultiplexer machine (denoted by MD), under the control of the MH, distributes incoming packets to FEs and merges the outgoing packets from FEs. Both the MH and the FEs are sliced into guest machines using operating system level virtualization mechanism [101]. A guest machine hosted in the MH is called a MH guest machine and a guest machine hosted in some FE is called a FE guest machine. We choose OS level virtualization because it is efficient and provides good isolation among guest machines. For one virtual network in PdP, its control plane runs in a MH guest machine. Depending on how much packet processing power a virtual network claims, one or more FE guest machines can be allocated to the virtual network to perform its data plane tasks. Slicing the FEs into how many FE guest machines and assigning which FE guest machines to each virtual network are important issues we need to consider. We will discuss this in detail when we present the design of the FE in Section 4.3.4.

The MD in the PdP node coordinates multiple FE guest machines of the virtual networks hosted in the FEs. Once receiving a packet, the MD first decides which virtual network that packet belongs to and then sends it to corresponding FE guest machines for processing, such as address lookup and traffic shaping. After a packet is processed, it is returned to the MD. At that time, the packet is tagged with necessary information (e.g., the outgoing interface) for the MD to decide how to dispatch it.

We show an example of the PdP node in Figure 4.1. It hosts three virtual networks, i.e., red, blue, and green. There are three MH guest machines, each of which runs the
control plane of a virtual network. Packets belonging to virtual networks are classified and distributed from the MD to the FE guest machines. The red and blue virtual networks require little packets processing power so that one FE is sliced into two FE guest machines, with one guest machine serving for the data plane of the red and blue virtual network, respectively. The green network requires much more processing power so two FE guest machines, each has all the processing power of one FE, are assigned to the green network. After packets being processed, they are returned to the MD with necessary tags and the MD dispatches those packets according to those tags.

4.3.3 The Management Host and The Multiplexer/Demultiplexer

Figure 4.2 depicts the basic structure of a management host. The control plane of each virtual network runs in the guest machines hosted by the MH. For simplicity, we implement the multiplexer/demultiplexer inside the MH\(^2\). The multiplexer/demultiplexer functions are implemented by the packet classifier and packet

\(^2\)Note that the multiplexer/demultiplexer can be implemented using another dedicate machine or special purpose hardware.
dispatcher running in the OS kernel of the MH. The packet classifier and dispatcher perform simple tasks and should process packets at high speed. For each incoming packet, the packet classifier first checks whether the packet belongs to a virtual network (e.g., the packet is encapsulated in UDP). If it does, the packet classifier further finds out which virtual network that packet belongs to and sends it to corresponding FE guest machine. The mapping between the virtual networks and their FE guest machines should be established when creating the virtual networks.

![Diagram of a management host structure](image)

**Figure 4.2.** The structure of a management host.

After a packet is processed by the FE guest machine and sent back to the MH, it should be properly tagged. The packet dispatcher checks the tags of the packet to see whether this packet should be sent out or it should be delivered to a local MH guest machine. If it should be sent out, the packet dispatcher simply sends the packet to the outgoing interface (note the packet has already been properly encapsulated by some FE guest machine and it has a tag to indicate the outgoing interface). If the FE guest machine labels a packet as local delivery (e.g., it is a routing update message), the packet dispatcher delivers it to the corresponding MH guest machine.
4.3.4 The Forwarding Engine

The other important component in a PdP node is the FE. Here we first present the structure of the FE and then discuss the problem of how to allocate the processing power of the FEs to virtual networks.

Structure of FE. The structure of an FE is shown in Figure 4.3. Each FE is sliced into one or more guest machines using OS level virtualization as well. A packet belonging to a virtual network is delivered to the packet processing function running inside the a FE guest machine. The packet processing function processes each packet according to the control plane of that virtual network and marks the processed packet with a set of simple tags. The tags can be some fields in the header of a lightweight encapsulation mechanism used between the MH and the FEs. The tags include information such as whether the packet should be locally delivered, or how the packet should be forwarded out by the dispatcher running in the MH.

Allocating the Processing Power of FEs. How to allocate the processing power of the FEs is of importance, because packet out-of-order resulting from parallelizing packet processing can impact the performance of the up-layer protocols such as TCP.
Informally, the basic principle should be not slicing the FEs too finely to avoid such a situation that there are a lot of “fragmented” FE guest machines. For example, suppose there are three FEs \((FE_1 \sim FE_3)\) and they are allocated to three virtual networks \((vnet_1 \sim vnet_3)\). Slicing and allocating the FEs according to either Figure 4.4(a) or Figure 4.4(b) satisfies the processing power requirement of each virtual network. However, the slicing of FEs as in Figure 4.4(a) may cause \(vnet_2\) and \(vnet_3\) to have lots of out-of-order packets. Slicing the FEs as in Figure 4.4(b) is a better choice because all three virtual networks have their required processing power and none of them has the packet out-of-order problem\(^3\).

![Figure 4.4](image)

**Figure 4.4.** \(vnet_1 \sim vnet_3\) get equal processing power in (a) and (b). In (a) \(vnet_2\) and \(vnet_3\) has two FE guest machines. In (b) \(vnet_2\) (and \(vnet_3\)) has one FE guest machine only.

In order to minimize the impact of packet out-of-order to a virtual network, we should assign minimal number of FE guest machines to serve for its data plane. Suppose there are \(n\) virtual networks \((vnet_1 \sim vnet_n)\) and \(vnet_i\) requires \(R_i\) processing power. Also suppose we have enough FEs in the PdP node and each FE has \(C\) processing power. Slicing the FEs and allocating their processing power can be formulated as a “bin packing problem”, which is NP-hard [111]. That is, if \(R_i = kC + r_i\), \((r_i < C)\), we should first allocate \(k\) FE guest machines to \(vnet_i\) and each of them has all the processing power of one FE. Then finding the minimum number of FEs to “pack” the \(n\) remainders \((r_1 \sim r_n)\) is the classic bin packing problem. Considering

\(^3\)Note that if no single FE guest machine can satisfy the requirement of a virtual network, we have to assign multiple FE guest machines to that virtual network.
that new virtual networks are created in PdP and old ones are removed from PdP, we develop an heuristic algorithm to decide the slicing and allocation of FEs in an online manner. Our algorithm adopts a heuristic similar to the classic “best fit” heuristic used in solving the bin packing problem. The details of the algorithm is omitted to save space.

Algorithm 1: \texttt{SliceAlloc}(R)

\textbf{Input:} \( R \), the processing power requirement of a virtual network \( vnet \).

\textbf{Output:} The “best fit” slicing and assignment of FEs which satisfies the requirement of \( vnet \).

\begin{align*}
1 & \quad r = R \% C; k = (R - r)/C; \\
2 & \quad \text{for } i = 0; i < k; i ++ \text{ do} \\
3 & \quad \quad \text{find an idle FE, create one guest machine with } C \text{ processing power in it, assign that guest machine to } vnet; \\
4 & \quad \text{FE} = \text{null}; \text{min} = \text{BIG\_NUM}; \\
5 & \quad \text{for } \text{FE}_i \in \text{all FEs AND } \text{leftPower}(\text{FE}_i) > r \text{ do} \\
6 & \quad \quad \text{if } (\text{leftPower}(\text{FE}_i) - r) < \text{min} \text{ then} \\
7 & \quad \quad \quad \text{min} = \text{leftPower}(\text{FE}_i) - r; \text{FE} = \text{FE}_i; \\
8 & \quad \text{create a guest machine in } \text{FE}_i \text{ with processing power } r \text{ and assign it to } vnet;
\end{align*}

4.3.5 A PdP Node Prototype

We have built a proof-of-concept PdP node prototype as shown in Figure 4.5. All machines are Linux PCs and we use OpenVZ [112] to slice the MH and the FEs. The packet classifier and dispatcher running inside the MH are implemented by kernel mode Click [113]. The packet processing function of each virtual network running in the FE guest machines is implemented by user mode Click.

The PdP node prototype has two external physical interfaces, \( A \) and \( B \). We assign two virtual interfaces to each virtual router hosted in this PdP node, one mapped
to each physical interface⁴, and a virtual router forwards packet between those two interfaces. The classifier in the MH classifies packets belonging to different virtual networks based on the UDP port numbers (assuming the virtual links in a virtual network are UDP tunnels). If a virtual network has multiple FE guest machines, the classifier sends packets to them in a round-robin manner. The packet processing function, which runs in the FE guest machines, processes each packet, encapsulates the packet with proper UDP/IP header, labels the packet a tag to indicate the outgoing interface, and sends it back to the MH. According to the tag labeled to the packet, the dispatcher in the MH sends that packet out via either interface A or interface B. In this PdP node prototype, the prototype field in the Ethernet header is reused as the tag to indicate the outgoing interface.

4.4 Experiment Evaluation

This section evaluates the packet processing performance of PdP. We focus on IP forwarding but the basic conclusions of our experiments also apply to virtual networks using protocols other than TCP/IP. Our experiments show that the raw

---

⁴Note that this is for purpose of testing and prototyping. In reality, each virtual router hosted by the PdP node can have any number of virtual interfaces.
packet forwarding speed of PdP scales with the number of FEs and it can match the best known forwarding speed of software router running in commodity hardware.

4.4.1 Experiment Setting

Figure 4.6 shows the testbed in our experiments. Two Linux PCs are connected by the router machine through Gbit Ethernet links, where the router machine is a PdP node. We test three settings in which the number of FEs varies from one to three. For comparison purpose, we also test the scenarios where the router machine is one Linux PC running user mode Click in its guest machine and running Click software router in kernel mode. All PCs are equipped with 2.4~3.0 GHz CPU, 1G RAM, and Gbit Ethernet adapters.

![Figure 4.6. The experiment testbed.](image)

4.4.2 Packet Forwarding Speed

We first use UDP traffic to test the raw packet forwarding speed of PdP. We configure the routing table to have only two routes, which point to the source host and the destination host respectively. The source host runs the udpgen tool shipped with Click to send UDP packets to the destination host. The udpgen tool runs in kernel mode and can send out packets at very high speed. The destination host runs the udpcount tool in Click to count the number of received UDP packets.

We had the experiments in which we create multiple concurrent virtual networks. The aggregate forwarding speed of the PdP node, when the number of virtual networks varies from one to three, does not show noticeable difference. To save space, throughout this section we present only the results where there is only one virtual...
network. If it is not stated explicitly, each FE hosts only one guest machine and the FE guest machine has all the processing power of the FE.

![Diagram](image)

**Figure 4.7.** Packet forwarding speed and packet loss rate in UDP traffic experiment.

We configure the source host to send out 64-byte UDP packets at a fixed speed ranging from $10^3$ packets per second (pps) to $1100^3$ pps. The forwarding speed of the router machine, when it is a PdP node, user mode Click router, or kernel mode Click router, is plotted in Figure 4.7(a). We also plot the packet loss rate at the router machine in Figure 4.7(b). As shown in Figure 4.7, when the input speed is lower than certain threshold, the forwarding speed always increases proportionally as input speed increases and the loss rate remains to be zero. The peak forwarding speed of PdP is proportional to the number of FEs and the peak speed of PdP with three FEs matches the peak speed of kernel mode Click. After the input speed exceeds the threshold, the packet loss rate becomes larger; the forwarding speed of user mode Click and PdP drops down but the kernel mode Click router maintains a constant forwarding speed. The reason is that the kernel mode Click sets the Ethernet interface into polling mode [102] so as to prevent the *receive livelock* [114]. On the contrary, the packets receiving and sending in user mode Click and PdP (in FE guest machines) are driven by the OS interrupt procedures. With increasing numbers of input packets, the interrupt processing can eventually starve all other system tasks, resulting in low
forwarding speed [102]. We believe that if the Linux native driver supports polling mode, the forwarding of PdP (and the user mode Click) would maintain the peak speed even the input speed is higher than its peak forwarding speed.

![Graphs showing packet forwarding speed for 512-byte and 1500-byte packets.](image)

**Figure 4.8.** Packet forwarding speed when the packet size is 512 bytes or 1500 bytes. The input speed for each experiment is set to saturate the 1 Gbps link.

We also test the forwarding performance of PdP in case of other two packet sizes, 512 bytes and 1500 bytes, in which we set the packet input speed to saturate the 1 Gbps link. For 512-byte packet, the maximum packet input speed is about 230K pps and the maximum packet input speed for 1500-byte packet experiment is about 80K pps. We plot the results in Figure 4.8. Our tests show that the forwarding speed gets lower for larger packets but having more FEs still achieves faster forwarding speed. PdP node with three FEs can match the speed of kernel mode Click in both the 512-byte packets experiment and the 1500-byte packets experiment. Equipping the PdP node with two FEs instead of one can double its speed. However, increasing the number of FEs from two to three does not show proportional forwarding speed enhancement because of the bandwidth limit of Gbit Ethernet link.
4.4.3 Forwarding with Large Routing Table

Note that in the above experiments, because the routing table has only two routes, the IP address lookup time is ignorable due to the “warm cache” effect [115]. To study the forwarding performance of PdP in case of large routing table, we download a BGP routing table from RouteViews [90] and extract about $170K$ IP prefixes. We repeat the above experiments with this large routing table. To avoid the warm cache effect, the source host sends out UDP packets with randomly selected unicast destination IP addresses. In the router machine, the *nexthop* of all routes (including the default route) is set to the destination host. The experiment results are plotted in Figure 4.9.

![Figure 4.9](image.png)

**Figure 4.9.** Packet forwarding speed in case of large routing table. (a) plots the peak packet forwarding speed. For (b) and (c), the packet input speed is set to saturate the 1 Gbit Ethernet link.
We can see that PdP still performs better than the user mode Click software router and matches the speed of kernel mode Click. However, the forwarding speed gets lower when using large routing table, especially for 64-byte packets experiment. For large packets, the forwarding speed does not show much degradation because the input speed is slow (due to the link bandwidth limit) and the IP address lookup time is not the significant part in packet processing.

4.4.4 TCP Throughput

So far the experiments using UDP traffic test only the raw packet forwarding speed of PdP. Most popular network applications use TCP protocol and the actual throughput achieved by TCP depends on more factors such as packet reordering, round trip time etc. In this experiment, we evaluate the performance of PdP in terms of TCP throughput. The router machine is configured with two routes in its routing table. An iperf server runs in the destination host and an iperf client running in the source host sends TCP traffic to the iperf server. We do not change any TCP-related parameters of iperf but use the default values. The TCP throughput is plotted in Figure 4.10.

![Figure 4.10. The TCP throughput experiment results.](image-url)
Our experiments show that PdP with one FE guest machine achieves similar TCP throughput as user mode Click IP router. Even the MH distributes packets to FE guest machines in a round-robin manner and there are a lot of out-of-order packets (as will be shown in Section 4.4.5), PdP with two or three FE guest machines demonstrates significant improvement of TCP throughput compared with user mode Click and PdP with one FE guest machine.

4.4.5 Packet Out-of-Order in TCP

Packet out-of-order is a challenging problem for parallel processing based systems. The following experiment is to quantify how PdP affects packet out-of-order in TCP. We use iperf to generate a TCP session and capture all the packets at the destination host. Then we use the Expert Info tool in wireshark [116] to analyze the out-of-order packets. The percentages of out-of-order packets, in case of the PdP node having one, two, and three FEs, are shown in Table 4.1. When counting the number of packets, we ignore those ACK messages sent by the destination host to the source.

<table>
<thead>
<tr>
<th></th>
<th>one FE</th>
<th>two FEs</th>
<th>three FEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of out-of-order pkts</td>
<td>0.31%</td>
<td>10.19%</td>
<td>13.02%</td>
</tr>
</tbody>
</table>

Table 4.1. Percentages of out-of-order packets when the PdP node has one, two, and three FE guest machines. The classifier in the MH distributes packets to FE guest machines in a round robin manner.

When there are more than one FE guest machines, about 10% ~ 13% packets are out-of-order packets and there is no significant difference between the experiments using two and three FE guest machines. Note that although considerable number of packets are out-of-order, as shown in Section 4.4.4, we still have decent TCP throughput.

Next we evaluate how the strategy of the packet classifier running in the MH affects the packet out-or-order. We use two identical FEs in the PdP node and
each FE hosts one guest machine. We tune the setting of OpenVZ so that one FE guest machines has 75% of the CPU cycles of an FE and the other FE guest machines has 50% of the CPU cycles of an FE. The packet classifiers uses two packet distributing strategies. One is sending packets in round-robin manner; the other is sending different number of packets based the allocated CPU cycles of the guest machines, i.e., for every 5 packets, sending packets 1, 3, 5 to the FE guest machine with 75% CPU cycles and sending packets 2, 4 to the FE guest machine with 50% CPU cycles. The results are presented in Table 4.2

<table>
<thead>
<tr>
<th>% of out-of-order pkts</th>
<th>round-robin</th>
<th>proportional</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.27%</td>
<td>10.02%</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2. Packet out-of-order when the classifier uses different strategies to distribute packets to FE guest machines.

The results in Table 4.2 show that less out-of-order packets will occur if the packet classifier takes into account the packet processing power of FE guest machines. Note that packet out-of-order is a rather complicated problem and it is not clear whether distributing packets based on the CPU cycles of a FE guest machine is the best strategy. We are actively investigating how other strategies affect the packet out-of-order and to what extent the up-layer protocol performance is impacted.

4.4.6 Transmission Delay

As every packet needs to traverse two PCs in a PdP node, it introduces the overhead in terms of transmission delay. We use the following experiment to evaluate the transmission delay overhead of PdP. Here the source host in Figure 4.6 uses ping to send ICMP packets to the destination host. We record average round-trip-time (RTT) reported by ping and show the results in Table 4.3.

---

5Note that CPU cycle is just one aspect of the packet processing power.
<table>
<thead>
<tr>
<th>RTT (ms)</th>
<th>user Click</th>
<th>PdP</th>
<th>kernel Click</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.208</td>
<td>0.296</td>
<td>0.132</td>
</tr>
</tbody>
</table>

Table 4.3. The round trip time experiment results.

From the results in Table 4.3 we can see that one PdP node adds about 0.17ms additional delay to the RTT, compared with the RTT of the kernel mode Click router. The addition delay is 0.09ms compared with the RTT of user mode Click router. According to the measurement study in [117], most hosts in Internet are about 14 hops away from a university probing site and the average RTT from those hosts to the probing site is about 80ms. Therefore, if PdP is widely deployed in Internet and each PdP node adds 0.17ms additional RTT delay, the total additional RTT delay would be about 2.4ms, which is ignorable considering an 80ms average RTT.

4.5 Conclusion

This chapter presents PdP, a full programmable and high speed virtual network platform. PdP is built from cost-efficient commodity hardware and open source software. A virtual network hosted in PdP can have complete control over its control plane and data plane without interfering other virtual networks. The key ideas behind PdP are two-fold: running virtual network control plane and data plane in guest machines for better isolation and flexibility; having multiple guest machines working in parallel to achieve high speed packet processing. We have built a prototype of the PdP node. The performance measurement shows very promising results for both UDP and TCP traffic.
5.1 Introduction

The intrinsic volatile nature of experimenting network innovations requires that a virtual network must be highly flexible and customizable. It is often required to tune various aspects of a virtual network. For example, a virtual network may be created to test new routing protocols, and therefore, its control plane needs to be customized. One may also experiment with new packet forwarding functions in a virtual network, such as queuing schemes or new addressing mechanisms, which cannot be realized without a customized data plane. In addition to the flexibility requirement, to experiment and test network innovations in a realistic environment, and more importantly, to attract long term deployment of new applications, a network virtualization platform should provide good data plane performance as well. It is desirable that the overhead of virtualization is minimized, so that the data plane performance of the platform can closely approach the full potential of the underlying hardware.

Achieving both high degree of flexibility and high performance is challenging. To guarantee the isolation between virtual networks so as to provide the flexibility to do customization, both control plane and data plane of a virtual network should run in the unprivileged domain of the hardware, which can introduce overhead. Although this overhead may not be an issue for the control plane functions, it can largely impact the data plane performance. For example, the VINI platform [46,118] provides high
degree of flexibility by running virtual network data planes in operating system user mode, but the packet forwarding speed of VINI is much slower than what the hardware can potentially achieve. Running the data plane of a virtual network in OS kernel mode, as what Trellis [47] does, can achieve much better performance. However, Trellis is limited in its ability to customize data plane in a virtual network due to the constraint imposed by the forwarding function provided by the kernel. The PdP platform presented in Chapter 4 provides high-degree of flexibility and high-speed packet forwarding by parallelizing the data plane. However, each forwarding engine machine still achieves much slower forwarding speed than what a software router running in the same hardware can achieve.

This chapter explores how to build a network virtualization platform that can achieve high degree of flexibility without sacrificing data plane performance. I propose EUROPA, a virtual network platform built from commodity hardware. EUROPA puts flexibility as its first design goal. Hence, the data plane of a virtual network hosted in EUROPA has to run in a virtual machine and essentially run in OS user mode, so that the virtual network can be granted the full control of its data plane. A new user mode packet forwarding scheme is designed for EUROPA, which can achieve high forwarding speed. Unlike the conventional ways of forwarding packets in user mode, this new user mode packet forwarding scheme uses shared memory to store packets and eliminates the overhead of copying packets between user space and kernel space; this scheme also avoids the overhead of invoking system calls by letting user mode forwarding process and OS kernel independently poll the state of a packet. Experimental results show that although an EUROPA virtual network runs its data plane in OS user mode, it can achieve close to the best known software router data plane performance.

The rest of this chapter is organized as followings. Section 5.2 examines different types of software routers running in commodity hardware and study why software routers using the conventional user mode packet forwarding have degraded perfor-
mance. The basic ideas of high-speed user mode packet forwarding is presented in section 5.3. How to translate this rationale into the design of the EUROPA platform is presented in section 5.4. Section 5.5 provides the experimental evaluation results. Section 5.6 concludes this chapter.

5.2 Packet Forwarding in Software Routers

It has been shown that running software router in the kernel mode or privileged domain of commodity hardware can achieve reasonably good forwarding speed [49, 110], but user mode software router has much worse performance [50,118]. However, user mode software router does provide some unbeatable advantages in building a virtual network platform, such as the flexibility to customize each virtual network and the isolation among different virtual networks. This section examines the packet forwarding procedures software routers running in OS kernel mode and user mode. This section focuses on the widely used Click modular router [113]. The study here can lead us to identify the reasons for the degraded performance of conventional user mode software router.

5.2.1 Kernel Mode Forwarding

Figure 5.1 depicts the process of forwarding a packet in kernel mode Click router. First, Click polls the NIC and moves a packet from NIC into the main memory (the rx\_fill function)\(^1\). After that, all read and write to the packet are conducted using a pointer to that packet. There is no more copying of that packet in the main memory. After the Click router decides how to forward that packet, the tx\_queue function is called to move the packet to the outgoing NIC.

\(^1\)We assume Click uses device polling to receive and send packets. Using device polling can prevent receive livelock and improve the packet forwarding performance [119].
To understand the performance baseline of kernel Click router, we conduct a set of experiments to test its forwarding speed. We use an inexpensive desktop workstation PC in our experiments. Our machine has a 2.66GHz Intel Core2 Duo CPU, 4G memory, and dual-port Intel PRO/1000 Gbit PCIe Ethernet adapter. The OS kernel is Linux-2.6.19.2 with the Click kernel patch. We run a simple Click configuration that merely moves packets from one NIC to another NIC without performing any packet processing tasks such as IP address lookup and TTL decreasing. In our tested machine, the maximum forwarding speed is about 1050K packet per second (pps) for 64-byte small packets, corresponding to 540 Mbps bandwidth for 64-byte packets, or more than 12 Gbps bandwidth for 1500-byte packets (if other components in the system, like the memory and bus bandwidth, are not the bottleneck.).

5.2.2 Conventional User Mode Forwarding

In Linux platform, user mode Click router uses the so-called PF_PACKET socket to read packets from kernel and write packets to kernel. When a packet is ready in NIC, the kernel moves it into the main memory and attaches that packet to the kernel buffer associated with the PF_PACKET socket. When the packet-receiving task is scheduled, user mode Click calls the recvfrom() system call to copy the entire

![Figure 5.1. Packet forwarding in kernel mode Click software router.](image-url)
packet, including the MAC header, into a user space buffer. After Click finishes processing the packet, the `send()` system call is invoked to copy the packet back into the kernel socket buffer and the kernel sends it out. Figure 5.2 depicts the procedure of forwarding packets in conventional user mode Click.

![Figure 5.2. The conventional packet forwarding in user mode Click.](image)

We repeat the experiments described in section 5.2.1, with the Click router running in user mode. In our experiments, the maximum forwarding speed is about 230 Kpps, which is less than one quarter of the forwarding speed of kernel mode Click running in the same machine.

5.2.3 Identify the Causes of Slow User Mode Forwarding

Given the performance measurement results described above, a nature question to ask is: what are the reasons for the degraded performance of the conventional user mode packet forwarding? From the above discussions, we can see that the differences between packet forwarding in kernel mode Click and user mode Click are two-fold. First, kernel mode Click uses its own device handling mechanism to receive and send packets; second, user mode Click has two more steps in packet forwarding, i.e., invoking system calls and copying the packet between user space and kernel.
**Device Handling:** The Linux kernel supports the NAPI device driver packet processing framework [120], which can poll the device in case of heavy packet load. According to the measurement in [119], Click working in polling mode is still 15% faster than native Linux kernel with NAPI driver.

**System Call:** User mode Click uses system calls to send and receive packets. Invoking system calls has certain overhead, such as changing the status of CPU, saving and restoring CPU registers. We measure the overhead of invoking the `recvfrom()` and the `send()` system calls used in user mode Click. The overhead is measured in terms of CPU cycles, using the “time stamp counter” of Intel CPU [121].

<table>
<thead>
<tr>
<th>System Call</th>
<th>send()</th>
<th>recvfrom()</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU cycles</td>
<td>3,000 cycles</td>
<td>3,400 cycles</td>
</tr>
</tbody>
</table>

**Table 5.1.** CPU cycles consumed in invoking system calls.

Our measurement shows that the overhead of calling the `send()` system call is about 3,000 cycles. The 3,000 cycles include two parts. The first one is from calling `send()` in user program to the kernel starting to execute `packet_sendmsg()` in the af_packet kernel module; the second one is from kernel finishing executing `packet_sendmsg()` to the returning of `send()` in user program. For the `recvfrom()` system call, the overhead is about 3,400 cycles. Therefore, for the user mode Click to forward one packet, the overhead of using system calls is about 6,400 CPU cycles.

**Memory Copying:** In additional to the overhead of invoking system calls, the conventional user mode packet forwarding also has two extra packet copying operations, one moves a packet from kernel to user space buffer, and the other pushes the packet in user space buffer back into kernel. We measure the CPU cycles consumed in copying data between user space and kernel. For 64-byte packets, it takes about 140 cycles to copy a packet from user space to kernel and 160 cycles to copy a packet from ker-

---

2 `packet_sendmsg` is the actual sending function of PF_PACKET socket.
nel to user space. The CPU cycles consumed in copying packets with different sizes
between user space and kernel are presented in Table 5.2.

<table>
<thead>
<tr>
<th>packet size (byte)</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1,024</th>
<th>1,500</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy_to_user (cycle)</td>
<td>162</td>
<td>188</td>
<td>239</td>
<td>302</td>
<td>442</td>
<td>575</td>
</tr>
<tr>
<td>copy_from_user (cycle)</td>
<td>140</td>
<td>157</td>
<td>200</td>
<td>259</td>
<td>388</td>
<td>507</td>
</tr>
</tbody>
</table>

Table 5.2. Overhead of copying packets between user space and kernel.

Overall Cost: From the results shown in Table 5.1 and Table 5.2, we see that to
forward one packet in user mode Click, there is a total of 6,700-cycle overhead in
using system calls and copying data between user space and kernel. In our tested
machine with a 2.66 GHz CPU, the extra 6,700 CPU cycles would limit the speed of
the conventional user mode packet forwarding to no more than 400 Kpps for 64-byte
packets.

Our analysis indicates the directions of achieving faster forwarding speed in user
mode software router. First, although the native Linux kernel supports device polling
by NAPI, using Click device handling still has certain performance advantage. Sec-
ond, we should avoid any additional packet copying operations between user space
and kernel. It is desirable that a packet is always processed in an “in-place” manner
once it is in the main memory. Finally, the user mode software router should avoid
using expensive system calls to interact with kernel. These insights help us to design
an efficient user mode forwarding scheme, which is the topic of section 5.3. How to
use that scheme to build a flexible and high-performance virtual network platform is
discussed in section 5.4.

5.3 Efficient User Mode Packet Forwarding

Having identified the directions in improving the conventional user mode software
router. In this section, we present the design of an efficient user mode packet forwarding
scheme. One thing worthy of highlighting is that the basic idea of the scheme
presented here can be applied to any user mode packet processing systems. Building virtual network platform is one of the many interesting applications of this efficient user mode packet forwarding scheme.

5.3.1 Overview

Our efficient user mode packet forwarding scheme is modularized and consists of a kernel module and a user module, referred to as KM and UM, respectively. The KM interacts with the network devices and performs only basic packet sending/receiving tasks. Most of the packet processing tasks are performed by the UM. From the perspective of layered network architecture, the KM works purely on MAC layer. For KM, each packet is a complete layer-2 frame. The UM, however, performs the network layer tasks. For example, in case of IP, UM does the IP lookup, TTL modification, etc.

Next we discuss how the KM and UM work together to achieve fast user mode packet forwarding.

5.3.1.1 Directly controlling the devices

The KM uses the device handling mechanism of Click to interact with the NICs, such as moving packets between NICs and main memory via DMA. The Click mechanism achieves better performance by putting NICs into polling mode to eliminate all programmed I/O with the NICs and avoid executing the networking code of the OS kernel [119].

5.3.1.2 Sharing memory between KM and UM

To avoid any additional packet copying operations, we create a buffer in the main memory to store all the packets and that buffer is shared by KM and UM. When a packet is ready in NIC, the KM moves the packet from NIC to the shared buffer. The UM then directly accesses that packet using a pointer or index and processes that
packet “in-place” in the shared buffer. After the UM finishes processing the packet and the outgoing interface is decided, the KM can send the packet out accordingly.

Most modern OSes provide the mechanism to share memory between kernel and user space programs. Using the mmap mechanism [122], which is a method of memory-mapped file I/O, we can allocate a piece of physical memory in kernel space and a user space program can map that memory to its own virtual memory address space. Hence, both the kernel and the user mode program can use their own virtual addresses to access that physical memory. The changes in the shared memory are visible to both OS kernel and the user mode program.

### 5.3.1.3 Avoiding expensive system calls

To avoid using the expensive system calls, the UM and the KM adopt an asynchronous model in accessing their shared buffer. That is, both of them independently monitor the “state” of a packet stored in the shared buffer, e.g., they poll the state of a packet. If either UM or KM notices a packet is ready to be taken over, the UM or KM starts to process the packet. For example, after the KM moves a packet from NIC to the shared buffer, it marks the packet as “ready for UM to handle”. Once the UM finds that a new packet is available, it processes that packet and changes the state of that packet to be “ready for KM to send out” when the processing is done. When the KM notices a processed packet is ready, it moves the packet from the shared buffer to the outgoing NIC and sends it out. By adopting this asynchronous buffer accessing model, there is no need to use any system call-based explicit notifications between UM and KM.

### 5.3.2 The Design of Efficient User Mode Packet Forwarding

To efficiently use the shared buffer, we design a two-level addressing mechanism for the KM and UM to access packets stored in the shared buffer.
A memory buffer, called the *packet pool*, is allocated in the kernel. All packets are stored in the packet pool. A packet in the packet pool is addressed by an *index*. For each NIC, we also create two other memory buffers, the *rxRing* and the *txRing*, which are ring buffers storing only the indexes of packets in the packet pool. The rxRing has the indexes of packets received from a NIC; the txRing stores the indexes of packet to be sent out via that NIC.

The packet pool, rxRing, and txRing are mapped to the virtual memory space of *UM* and these three shared buffers should never be swapped out from the main memory. Using the indexes in the rxRings and txRings, the UM can directly access the packets stored in the packet pool. When the KM receives a packet from NIC A and stores the packet in the packet pool, KM writes the index of that packet into the rxRing of NIC A. The UM receiving a packet from NIC A is essentially reading the rxRing of NIC A to get the index of a packet in the packet pool. To forward a packet received at NIC A out via NIC B, the UM removes the index of that packet from of NIC A’s rxRing and writes that index into the txRing of NIC B. The KM monitors the txRings of all NICs. Once an index appears in the txRing of NIC B, it sends the packet out via NIC B and removes that index from the txRing.

### 5.3.2.1 The packet pool

The *packet pool* is a piece of consecutive memory organized as an array of equal size *slots*. Each slot is identified by its index in the array. The size of a slot should be large enough to store one entire packet, i.e., the slot size is larger than the MTU of the NIC. Given $Addr_{pool}$, the address of the first byte in the packet pool, and $S_{slot}$, the size of a slot, the address of the slot at index $idx$ is $Addr_{slot} = Addr_{pool} + S_{slot} \times idx$.

Each slot in the packet pool has a “stat” flag to record the state of that slot. The “stat” flag can be one of the five states shown in Table 5.3. Clearly, only when the state of a slot is “FILLED”, the UM can process the packet stored in that slot; only
<table>
<thead>
<tr>
<th>State</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPTY</td>
<td>This slot is empty and can be used to store a packet.</td>
</tr>
<tr>
<td>MOVEIN</td>
<td>This slot is in the middle of loading data from NIC.</td>
</tr>
<tr>
<td>MOVEOUT</td>
<td>This slot is in the middle of loading data to NIC.</td>
</tr>
<tr>
<td>FILLED</td>
<td>The packet in this slot is ready to be processed.</td>
</tr>
<tr>
<td>PROCESSED</td>
<td>The packet in this slot is ready to be sent out</td>
</tr>
</tbody>
</table>

Table 5.3. A slot in the packet pool can have five states. This table shows the meaning of each state.

when the state of a slot is “PROCESSED”, the packet in that slot can be moved to NIC by the KM. The transition of the “stat” flag of a slot is depicted in Figure 5.3.

![Slot state transition diagram](image)

Figure 5.3. Slot state transition diagram.

5.3.2.2 The rxRing and txRing

A rxRing or txRing is basically an array of packet indexes. In Figure 5.4, we show the example of a ring of size 8. Each index represents a slot in the packet pool. The initial values of all indexes in rxRing and txRing are set to $-1$, which means they do not represent any slots in the packet pool.

For each ring buffer, the KM uses a pointer $ptr_k$ to record which index in the ring buffer is the next one to access; the UM also has a pointer $ptr_u$ serving for the same purpose. For example, in Figure 5.4, the $ptr_u$ pointer is 0 and the index at position 0 in the ring is 20. That means the UM should process packet at index 20 in the packet pool. The initial value of $ptr_k$ and $ptr_u$ of all rxRings and txRings should be zero.
Figure 5.4. rxRing and txRing structure.

For the rxRing of NIC A, if the index at position $ptr_k$ is $-1$ and a packet is ready at NIC A, the KM writes the packet index into position $ptr_k$ in NIC A’s rxRing after moving the packet from NIC A to the packet pool. Then the $ptr_k$ pointer of NIC A is updated to $ptr_k = (ptr_k + 1) \% S_{ring}$, where $S_{ring}$ is the size of the ring buffer. The UM reads the index at position $ptr_u$ and processes the packet if the index is not $-1$. After the UM finishes processing the packet, it resets the index at $ptr_u$ to $-1$ and moves forward the $ptr_u$ pointer.

For the txRing of NIC A, KM and UM play the opposite roles. That is, after finishing processing a packet, the UM writes packet index into position $ptr_u$ and moves $ptr_u$ forward; the KM reads the packet index at position $ptr_k$, resets the index at $ptr_k$ to be $-1$ after sending out the packet via NIC A, and moves forward pointer $ptr_k$.

5.3.3 A Complete Example

Now we show a complete example regarding how a packet is forwarded by the efficient user mode processing scheme. This example is plotted in Figure 5.5. If a packet is ready in NIC A and the index at position $ptr_k$ of NIC A’s rxRing is $-1$, the KM finds a slot ($slot_2$) in the packet pool whose state is “EMPTY” and changes the state of $slot_2$ to “MOVEIN”. Then the KM updates the index at position $ptr_k$ in
NIC A’s rxRing (step (1)) and moves the packet from NIC A to slot₂ (step (2)). At this point, the state of slot₂ should be changed to “FILLED” and the ptr_k pointer of the rxRing of NIC A is moved forward by the KM. The UM monitors the rxRing of NIC A and notices that the index at position ptr_u indicates a packet is ready (step (3)). The UM reads the rxRing and gets the index of that packet in the packet pool. Then the UM can directly access the packet and performs necessary operations, such as determining how to forward the packet (step (4)). Suppose the packet should be forwarded out via NIC B. The UM writes the index of that packet to position ptr_u in NIC B’s txRing (step (5)) and changes the state of slot₂ to be “PROCESSED”. After that, the ptr_u pointer of the txRing is moved forward by the UM. The KM now notices that the ptr_k position of NIC B’s txRing has a valid packet index and that packet is ready to be sent out (step (6)). Hence, the KM changes the state of slot₂ to be “MOVEOUT” and moves the packet in slot₂ to NIC B (step (7)). After the packet is sent out, the state of slot₂ is changed to “EMPTY” and the ptr_k pointer of the txRing is moved forward by the KM.

Figure 5.5. Steps of forwarding a packet.
5.4 EUROPA: Flexible and High-performance Network Virtualization

Having discussed the scheme achieving efficient user mode forwarding, in this section, we use this scheme to build EUROPA, a flexible and high performance virtual network platform.

5.4.1 EUROPA Architecture

An EUROPA server is sliced into virtual routers (VRs). Similar to the user mode forwarding scheme discussed in previous section, the data plane of an EUROPA virtual network consists of a kernel module and a user module, denoted as EUROPAKM and EUROPAUM, respectively. The EUROPAKM runs in kernel mode of a server and it is shared by all virtual routers in that server. The EUROPAUM runs inside the virtual router and it is free to be customized by each virtual network. The EUROPAKM uses shared memory to interact with the EUROPAUMs of different virtual networks. Figure 5.6 depicts the basic architecture of EUROPA.

![Figure 5.6. The architecture of EUROPA.](image)
5.4.2 Virtual Routers

When a virtual router is created, a new packet pool is allocated in kernel space. For each virtual NIC of that VR, a rxRing and a txRing are also allocated in kernel space. The packet pool, the rxRings and the txRings created for a VR can only be mapped to the EUROPAUM running inside that VR. This prevents the interference among different virtual networks hosted in the same server.

One VR in EUROPA can have multiple virtual NICs. The virtual NICs are virtual Ethernet devices. Each virtual NIC has a unique MAC address and a virtual NIC is bound to a physical NIC. Packet sending and receiving via a virtual NIC are actually conducted by its bound physical NIC. The binding of virtual NICs and physical NICs should be setup when VR is created.

In EUROPA, we use an OS-level virtualization scheme [101], OpenVZ [112], to slice a server into VRs. OpenVZ is a lightweight virtualization scheme used in several network virtualization systems [108,123]. Compared with other virtualization approaches, such as full-virtualization [105] and paravirtualization [124], virtualization on the OS level provides the best performance and scalability [125]. The performance difference between a virtual machine in OpenVZ and a standalone server is almost ignorable [101].

5.4.3 Packet Forwarding in Virtual Networks Hosted by EUROPA

We use Click running in kernel mode to implement EUROPAKM. The EUROPAUM running inside a virtual router closely resembles a Click user mode software router (except how EUROPAUM sends and receives packets), so that one can quickly implement desired packet processing functions, e.g., using the existing rich collection of Click elements or writing his own Click elements. In the following, we present the basic design of EUROPAKM and EUROPAUM.
5.4.3.1 Design of EuropaKM

The EuropaKM interacts with the physical NICs and performs only layer-2 packet sending/receiving tasks. Figure 5.7 depicts the diagram of EuropaKM. For clarity, we show only one virtual router $VR_A$ with two virtual NICs. At first, a packet is polled from NIC to the main memory by the “poll device” function. Then the “classifier” function decides which VR owns that packet according to the packet destination MAC address. If the packet belongs to $VR_A$, it is copied into the packet pool of $VR_A$ and the rxRing of the virtual NIC is updated (the “to pool” function in Figure 5.7). For a broadcast packet, the “classifier” duplicates the packet and copies a clone to the packet pool of each VR. The “from pool” function monitors the txRings and pushes a packet to the “to device” function once the packet is ready to be sent out, i.e., the state of the packet is “PROCESSED”. The “to device” function expects the MAC layer header of the packet has already been properly set. Hence, the “to device” function expects the MAC layer header of the packet has already been properly set. Hence, the “to device” function expects the MAC layer header of the packet has already been properly set. Hence, the “to device” function expects the MAC layer header of the packet has already been properly set.

---

3Only a pointer of the packet is passed to the “to device” function.
device” function just moves the packet from the packet pool to the NIC and the NIC sends it out.

5.4.3.2 Design of EUROPAUM

The EUROPAUM implements the network layer of a virtual network hosted in EUROPA. Figure 5.8 shows the diagram of EUROPAUM. The "from pool” function reads a packet from the packet pool according to the index stored in the rxRing of a virtual NIC (in Figure 5.8, veth0 is the packet incoming NIC). The “from pool” function does not copy the packet but reads the index of the packet from the rxRing. The actual packet is still stored in the packet pool. The packet is processed by the “processing” function. What the “processing” function does is up to each virtual network, e.g., a virtual network can develop its own Click elements to process the packets. The indexes of the processed packets are written to the txRing of the outgoing virtual NIC (veth1 in Figure 5.8) by the “to pool” function.

![Diagram of EUROPAUM](image)

**Figure 5.8. Diagram of EUROPAUM.**

**Address resolution:** Setting proper MAC layer header is the duty of EUROPAUM. The “to pool” function in Figure 5.8 should change the MAC layer header of a packet before updating the txRing of the outgoing virtual NIC. If a virtual network uses IP, ARP protocol should be used to resolve IP addresses into MAC addresses. In case of
a virtual network using a network layer protocol other than IP, the virtual network should implement its own address resolution protocol in EUROPAUM.

**Packets generated by EuropaUM:** The EUROPAUM not only forwards packets, in some cases it also generates packets. For example, the EUROPAUM needs to generate the ARP response message to assist neighbors in resolving its IP address. When the EUROPAUM generates a packet, it should reserve a slot in the packet pool by itself. After filling the packet into the reserved slot, the EUROPAUM writes the index of that slot into the txRing of the outgoing virtual NIC and that packet will be sent out by the EUROPAKM.

5.4.4 Discussion

5.4.4.1 Security and Isolation

Security and isolation are important issues in building any shared platforms. In EUROPA, although we use memory mapping to share memory between user programs and OS kernel, a user program cannot access memory beyond the shared memory space. Accessing the memory beyond the shared memory space will only cause the user program to terminate abnormally without hurting the OS kernel. More importantly, one VR can access only its own packet pool, rxRings and txRings. A VR doing something wrong, such as illegally moving the pointers in rxRings or txRings, or writing a corrupted packet into the packet pool, only affects the VR itself.

Further enhanced security and isolation mechanism can be implemented by adding some sanity check into EUROPAKM. For example, when the “from pool” in Figure 5.7 accesses a packet in the packet pool of some VR, it should first check whether the packet index in txRing is smaller than the number of slots in the packet pool. Since the EUROPAKM is not exposed to VRs and it is more “controllable”, by carefully implementing and testing the EUROPAKM, we can avoid dangerous actions leading to kernel instability.
5.4.4.2 Overhead of packet classification

Virtualizing a physical NIC into multiple virtual NICs introduces some additional overhead in terms of packet copying. Comparing Figure 5.7 with Figure 5.5, we see that when a physical NIC is shared by multiple VRs, the EuropaUM cannot directly move packets from NIC to the packet pools of different VRs. A packet has to be moved to the main memory and be classified. Then the packet is copied to the packet pool of a VR, which results in one undesirable extra packet copying operation.

One potential solution to eliminate that extra packet copying is to allocate one single packet pool in kernel space and share it to all VRs. This solution, although avoids the extra copying, destroys the isolation among different VRs because one VR can directly access the packets belonging to other VRs.

It is better to let the NIC hardware do the packet classification and determine which VR owns a packet. Some high-end server-class NICs in the market already provide hardware-based packet classification [126]. However, NICs with packet classification capability are significantly more expensive than other commodity NICs. More importantly, those NICs can only classify packets according to MAC address or VLAN tag [126], but other fields in a packet head can be used to indicate which virtual router owns a packet [46,47,50]. We believe the scheme used in Europa, although has one more copying operation for each packet, is a better tradeoff between performance and cost in building virtual network substrate at this time.

5.4.4.3 Overhead of polling packet state

Both the virtual router and the EuropaKM adopt polling to check the state of a packet. Polling inevitably introduces the CPU usage overhead. Even there is no packet to process, a virtual router still polls for packets and consumes CPU cycles.

---

4The cost of a NIC with packet classification capability ranges from several hundred dollars to more than two thousand dollars. Other commodity NICs usually cost about 50~100 dollars.
We believe this CPU overhead should not be an issue for a network virtualization platform. When certain amount of CPU resource is allocated to a virtual router, the virtual router should be entitled to use all its resource. The server virtualization mechanism ensures that a virtual machine does not exceed its CPU resource quota. Hence, even a virtual router uses all its CPU resource, it should not affect other virtual routers hosted in the same Europa server. Besides, polling as fast as possible ensures the next packet to be promptly processed. One possible compromise to reduce the polling overhead is dynamically changing the polling frequency according to the packet incoming rate.

5.5 Experimental Evaluation

This section evaluates the data plane performance of Europa. Our experiments show that the packet forwarding speed of Europa is much better than conventional user mode packet forwarding schemes and can match the best known forwarding speed of software router running in commodity hardware.

5.5.1 Experiment Setting

Figure 5.9 shows the testbed used in our experiments. The middle machine in Figure 5.9 runs software routers to forward packet between the sender and receiver machines. All machines are identical commodity desktop PCs. Each one has a 2.66GHz Intel Core2Duo CPU, 4G memory, and two Intel PRO/1000 Gbit NICs. The software router machine runs a customized 2.6.18 Linux kernel. We first apply the OpenVZ patch to a vanilla Linux 2.6.18 kernel and then manually change the source code to include the Click kernel patch. Hence, our kernel supports both OpenVZ and kernel mode Click.

When testing the performance of Europa, we create one or multiple virtual routers in an Europa server. Each virtual router has two virtual NICs, mapped
to the two physical NICs of the server, and the EUROPA virtual router forwards packets between those two virtual NICs. The default packet pool size of each EUROPA virtual router is 128 packets. We compare the performance of EUROPA virtual router with two other software routers, i.e., kernel mode Click and user mode Click software routers. Kernel mode Click provides a baseline of the best known packet forwarding speed of software routers running in commodity hardware. User mode Click, on the other hand, presents the forwarding performance of conventional software routers running in user mode, which can be safely customized.

5.5.2 UDP Experiments

We first use UDP traffic to test the forwarding speed of virtual routers hosted in EUROPA. The packet forwarding speed is measured in terms of packets per second (pps). Minimal length packets (64-byte) are used to stress the virtual routers.

5.5.2.1 Single virtual router

To understand the raw packet forwarding speed of EUROPA, we configure the EUROPA server to host only one virtual router, which is loaded with an IP router configuration with only two entries in its forwarding table. One is to the sender and the other on is to the receiver. We also use similar configuration to test the forwarding speed of kernel mode Click and user mode Click running in the same machine. Figure 5.10(a) plots the forwarding speed results in our experiments. We see that as the packet input speed increases, user mode Click quick reaches a saturation forwarding speed of about 200 Kpps. Kernel mode Click achieves close to 1000 Kpps.
peak forwarding speed. **Europa** virtual router can forward packets at about 820 Kpps, which is more than four times the speed of user mode Click router.

![Graph showing UDP packet forwarding speed](image)

**Figure 5.10.** UDP packet forwarding speed.

Next, we test **Europa** in a more realistic configuration. We extract about 170K IP prefixes from a RouteViews [90] BGP table and install them in the forwarding table of an **Europa** virtual router. The sender machine generates 64-byte UDP packets with random class-C destination IP addresses. The **Europa** virtual router forwards all incoming packets to the receiver machine. Kernel mode Click and user mode Click routers are also evaluated in similar setting with the large forwarding table. The forwarding speed results are shown in Figure 5.10(b). We see that **Europa** still matches the speed of kernel mode Click and is much faster than user mode Click. Figure 5.10(b) also shows that the forwarding speed gap between kernel mode Click and **Europa** is smaller than that shown in Figure 5.10(a). As more CPU cycles are consumed by computational tasks such as IP address lookup, the advantage of kernel mode Click becomes less noticeable, because running those computational tasks in kernel space or user space does not make too much difference in terms of CPU cycle consumption.
5.5.2.2 Multiple virtual routers

We also evaluate the scalability of EUROPA in terms of hosting multiple virtual routers in one server. We vary the number of concurrent EUROPA virtual routers from 1 to 10 and measure the speed of forwarding 64-byte UDP packets. Because section 5.5.2.1 shows that the performance trends of EUROPA, kernel mode Click, and user mode Click are similar in small forwarding table configuration and large forwarding table configuration, here we present only the small forwarding table experiment results.

![Graph](image)

**Figure 5.11.** UDP packet forwarding speed vs. number of virtual routers.

Figure 5.11 plots the average forwarding speed of each virtual router and the aggregate forwarding speed of all virtual routers hosted in an EUROPA server. We see that the forwarding speed of a virtual router is inversely proportional to the number of current virtual routers hosted in the EUROPA server, because multiple virtual routers are competing for CPU and bandwidth resources. As there are more concurrent virtual routers, the aggregate forwarding speed becomes smaller. The reason is that the CPU needs to more frequently switch between different virtual routers to run their data plane processes when there are more virtual routers. The CPU context switching overhead lowers the aggregate forwarding speed of multiple EUROPA virtual routers. However, we can expect that the context switching overhead can be alleviated with the increasing popularity of CPUs with more cores.
5.5.3 TCP Experiments

Next, we evaluate the TCP performance of EUROPA. The iperf tool is used to generate TCP traffic between the sender and receiver machines in Figure 5.9. We do not change any TCP-related parameters of iperf but use the default values. Again, only the experiment results of small forwarding table are presented here.

5.5.3.1 Single virtual router

We test the TCP throughput of a single EUROPA virtual router, kernel mode Click, and user mode Click. Figure 5.12 plots the experiment results. As we can see, EUROPA virtual router achieves almost the same TCP throughput as kernel mode Click. Compared with user mode Click, the throughput of EUROPA virtual router is about 22% higher. Because TCP always tries to use large packets, the number of packets forwarded per second is small even the throughput is small to one Gbps line speed. Hence, the advantage of EUROPA virtual router as compared with user mode Click is not as significant as the UDP experiment results shown in section 5.5.2. However, we can expect that if faster NICs are used in our experiments, e.g., 10 Gbps NICs, EUROPA virtual router will show more significant advantage as compared with user mode Click.

![Figure 5.12. TCP throughput.](image)
5.5.3.2 Multiple virtual routers

We also evaluate the TCP throughput when multiple concurrent virtual routers are hosted in one EUROPA server. Figure 5.13 shows the average throughput of one virtual router and the aggregate throughput of all virtual routers, when the number of concurrent virtual routers varies from 1 to 10. Not unexpectedly, the average TCP throughput of each EUROPA virtual router shows inversely proportional property to the number of virtual routers; and the aggregate TCP throughput lowers as more virtual routers are hosted in an EUROPA server. However, because TCP uses large packets, the aggregate throughput gets about 13% lower only as the number of virtual routers increases from 1 to 10.

![Figure 5.13. TCP throughput vs. the number of virtual routers.](image)

5.5.4 Forwarding Performance and Packet Pool Size

EUROPA uses packet pool to share packets between EUROPAKM and a user mode virtual router. The packet pool implicitly works as a buffer to cache packets. To study how the size of packet pool affects the forwarding performance of EUROPA, we run two virtual routers in one EUROPA server and change the packet pool size of these virtual routers from 2 slots to 256 slots. We measure aggregate 64-byte UDP packets forwarding speed and aggregate TCP throughput for each packet pool size. Figure 5.14 presents the experiment results. We see that for both UDP and
TCP experiments, the forwarding performance of EUROPA shows little sensitivity to packet pool size larger than 4. The reason is that the input traffic in our experiments is close to constant rate. As more burst shows in the traffic, we expect larger packet pool size can be more helpful in achieving better forwarding performance.

![Graphs](image)

**Figure 5.14.** UDP and TCP performance vs. packet pool size.

### 5.6 Conclusion

This paper presents EUROPA, a customizable and high-speed virtual network platform. EUROPA is built from cost-efficient commodity hardware. Most of the data plane functions of a virtual network hosted by EUROPA run in virtual machine. Only minimal data plane functions, such as those handling network devices, dwell in the kernel mode. Hence, virtual networks hosted by EUROPA can safely and almost fully customize their data planes. We design an efficient user mode packet forwarding scheme to achieve high-speed packet forwarding in EUROPA. This scheme avoids using expensive system calls to interact with the operating system kernel and introduces minimal extra packet copying operations during forwarding packets in virtual networks. Our experimental results show that EUROPA performs much better than
other virtual network platforms adopting user mode packet forwarding and matches the best known software router forwarding speed.
6.1 Introduction

Data centers with a cluster of commodity servers become common places for data storage, data analysis, and large-scale network services [60,62]. Such a data center infrastructure is driven by the demand of petabyte of data storage and high computation power required for processing the data. More importantly, it is projected that the demand for data storage and processing will grow rapidly as more data are available for applications such as web searching, medical image processing, social network mining, and scientific computing. To meet the demand of the growth, one of the essential requirements for data center infrastructure is that it must scale to hundreds of thousands or millions of servers.

While inexpensive commodity PCs make it possible to expand a data center to millions of servers, interconnecting these servers in a scalable and cost-efficient fashion can be challenging. With a data center of increasing server number and storage size, the communication bandwidth has to scale more than linearly (or squarely) to meet the bandwidth demand of frequent data accessing and shuffling in distributed data processing and storage [61,63,64]. In order to keep the interconnection cost low, one natural choice for interconnecting these servers is to leverage commodity hardware such as inexpensive Ethernet switches and the existing network cards in commodity PC servers. So far, there are two approaches for interconnecting servers with commodity switches. The first approach is switch centric where the switch functionality is extended to accommodate the need of the interconnection, while requiring
no modification to the servers (including network interface, operating system, and applications) [58,59,79]. The second approach is server centric where each server acts as both data processing/storage and data relay node while requiring no change to the switches [54,55,57].

In this chapter, we take a server centric approach and propose a server interconnection structure called DPillar. Each server in a DPillar network is a computation workstation as well as an intermediate node relaying the data between other servers. This server centric design offers many unbeatable advantages. First, it avoids the configuration effort required by a separate switching fabric and simplifies the management of the data center network. Second, shifting the networking functionalities from a separate switching fabric to the servers provides much higher degree of programming capability, which facilitates the design of efficient fault-tolerant routing scheme and traffic-aware routing scheme. Third, the server-based design is much more cost-efficient because it uses only low-end layer-2 dummy switches.

More specifically, DPillar aims to leverage plug-and-play commodity Ethernet switches with only layer-2 switching capability. Ethernet switches with moderate number of ports (e.g., 24 or 48 ports) and with the ability to switch layer-2 frames at line speed are widely available and relatively inexpensive. Layer-2 Ethernet switches also have the advantage of requiring minimal configuration effort as they are basically plug-and-play devices. Further, DPillar requires only two network interfaces for each server. As most off-the-shelf PC servers offer two high-speed Gbit Ethernet ports, one primary port and one backup port, there is no need to physically upgrade the servers. More importantly, when expanding an existing DPillar network with additional servers, it is not required to upgrade existing servers in the data center. Therefore, such an interconnection structure can scale to any number of servers with minimal deployment overhead.
Despite the fact that each server has only two network interfaces, the DPillar interconnection offers rich connections between servers, so the aggregate bandwidth can facilitate data-intensive applications. The structure of the DPillar network is totally symmetric so that it removes any network bottleneck at the architecture level. We have designed a simple yet highly efficient routing scheme for DPillar network. One salient feature of the proposed routing scheme is that it eliminates the need of doing table lookup when servers are relaying packets. Our prototyping implementation using commodity PC shows that the PC servers can perform data forwarding in line speed without consuming significant resources at the servers. Therefore, such an interconnection structure is feasible for the server centric approach. Furthermore, we propose routing schemes that can efficiently handle a wide range of failures and perform load balancing in DPillar network.

The rest of this chapter is organized as follows. Section 6.2 provides the background of interconnection networks and related work on data center networks. Section 6.3 presents the network topological structure of the DPillar network. Section 6.4 and section 6.5 are devoted to the discussion of routing in DPillar network. Section 6.6 presents the prototyping implementation and performance evaluation of DPillar. Section 6.7 concludes this chapter.

6.2 Background and Related Work

6.2.1 Interconnection Networks in Data Centers

There are two categories of interconnection networks used in building data centers. The first one has a clear boundary between the network and the end hosts. Usually, multiple levels of switches are interconnected into a switching fabric and the servers are attached as the “leafs” of the switching fabric [79]. The servers are pure end-hosts, which perform computation task only. Having one interface is enough for each server to be connected with other servers.
In the second category of interconnection network, the servers are not only the computation workstations but also intermediate nodes relaying traffic for other servers. Servers are connected with each other by point-to-point links or hubs to construct certain topologies. Classic interconnection topologies include full mesh, hypercube, butterfly, de Bruijn, etc [127–129]. Compared with the interconnection network with a switching fabric, using servers as relay nodes and placing more intelligence on servers is usually more flexible, because the servers are much easier to program than the switching devices in a switching fabric.

![Table 6.1. Comparison between different data center interconnection networks. Parameter \( n \) is the number of ports each switch has; \( N \) represents the total number of servers; \( U_s \) is the unit price of a \( n \)-port switch; \( D \) is the network diameter. For DCell, FiConn, and BCube, \( l \) is the number of recursively construction level. For DPillar, \( k \) is the number of server columns.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>DCell</th>
<th>FiConn</th>
<th>BCube</th>
<th>FatTree</th>
<th>DPillar</th>
</tr>
</thead>
<tbody>
<tr>
<td>server degree</td>
<td>( l + 1 )</td>
<td>2</td>
<td>( l + 1 )</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>diameter ( D )</td>
<td>( 2^l )</td>
<td>( \approx 2^{l+1} )</td>
<td>( l + 1 )</td>
<td>constant</td>
<td>( \leq \frac{2k}{N} )</td>
</tr>
<tr>
<td>bisection width</td>
<td>( \frac{N}{4 \cdot \log N} )</td>
<td>( \frac{N}{4 \cdot 2^l} )</td>
<td>( \frac{N}{2} )</td>
<td>( \frac{N}{2} )</td>
<td>( \frac{N}{k} )</td>
</tr>
<tr>
<td>number of servers ( N )</td>
<td>( (n + 1)^2 )</td>
<td>( 2^{l+2} \left( \frac{n}{3} \right)^2 )</td>
<td>( n^{l+1} )</td>
<td>( \frac{n^{l+1}}{4} )</td>
<td>( k \left( \frac{n}{2} \right)^k )</td>
</tr>
<tr>
<td>number of servers as a function of diameter ( D )</td>
<td>( (n + 1)^D )</td>
<td>( 2D \left( \frac{n}{4} \right)^{D/2} )</td>
<td>( n^{D} )</td>
<td>( N/A )</td>
<td>( 2^{\frac{3}{2}D} \left( \frac{n}{2} \right)^{\frac{5}{2}D} )</td>
</tr>
<tr>
<td>number of switches</td>
<td>( \frac{N}{n} )</td>
<td>( \frac{N}{n} )</td>
<td>( (l + 1) \frac{N}{n} )</td>
<td>( 6 \frac{N}{n} )</td>
<td>( k \frac{N}{n} )</td>
</tr>
<tr>
<td>cost of connecting one server†</td>
<td>( \frac{U_s}{n} )</td>
<td>( \frac{U_s}{n} )</td>
<td>( (l + 1) \frac{U_s}{n} )</td>
<td>( 5 \frac{U_s}{n} )</td>
<td>( 2 \frac{U_s}{n} )</td>
</tr>
<tr>
<td>switch upgrade</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>traffic balance</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

†Not including the cost of NICs and cables.

6.2.2 Related Work in Data Center Networks

A thread of recent research activities on data center networks have proposed several interconnection architectures. The Monsoon network presented in [53] uses a hierarchical switching fabric where the top-of-rack switches are connected to a high-bandwidth core switch. There is a Directory Service in Monsoon network to provide
the mapping between network addresses and MAC addresses. The fat-tree network presented in [58] is also a switching fabric-based interconnection network. The fat-tree network uses identical switches to build the switching fabric. Therefore, there is no need to install the high-bandwidth and expensive core switches. The switches used in [58] should have layer-3 switching capability and need to be slightly upgraded in order to make full use of the underlying topology. The PortLand network is proposed in [59], which is also based on fat-tree. PortLand uses hierarchical pseudo MAC addresses, so as to support efficient layer-2 routing and forwarding, as well as virtual machine migration. A centralized fabric manager is used to maintain soft state about the network topology and assist ARP resolution.

The DCell [54] interconnection network is a server-centric network where the servers are not only the computation workstations but also the intermediate interconnection nodes. A higher level DCell network can be recursively constructed from lower level DCell networks, so that the number of servers in a DCell network grows double exponentially as the level increases. As the levels in a DCell network increases, the servers need to install more interfaces to do interconnection. The links in DCell network are not evenly loaded. Those links connecting lower level DCells are usually more loaded than the links connecting higher level DCells. The FiConn network proposed in [55] uses similar recursive construction scheme as DCell. However, each server in FiConn can have only two interfaces. FiConn also has the issue of unevenly loaded links. The BCube [57] is another server-based network. Servers in BCube have multiple interfaces and multiple layers of commodity switches are used to connect the servers. BCube has rich connections so as to support bandwidth intensive applications running in data centers. Table 6.1 summarizes some key features of different data center interconnection networks.
6.3 Interconnection of DPillar

In this section, we first present the interconnection structure of DPillar. Then we discuss some of the topological properties of DPillar and the cost of building such a network.

6.3.1 Network Structure of DPillar

A DPillar network is built of two kinds of devices, dual-port servers and \( n \)-port switches. The servers are arranged into \( k \) columns; the switches are arranged into \( k \) columns too. We use \( H_0 \sim H_{k-1} \) to represent the \( k \) server columns and \( S_0 \sim S_{k-1} \) to represent the \( k \) switch columns. The \( k \) server columns and \( k \) switch columns are alternately placed along a cycle, as shown in Figure 6.1. Visually, it looks like the \( 2k \) columns of servers and switches are attached to the cylindrical surface of a pillar. Using its two ports, a server in each server column is connected to the two switches in its two neighboring switch columns. In other words, for a server in column \( H_i \), one of its ports is connected to a switch in column \( S_i \) and the other port of the server is connected to a switch in column \( S_{(i+k-1)\%k} \). For a switch in column \( S_i \), half of its \( n \) ports are connected to \( n/2 \) servers in \( H_i \) and the other half are connected to \( n/2 \) servers in \( H_{(i+1)\%k} \). Deciding which \( n \) servers are connected to the same \( n \)-port switch is important and we will discuss it soon later. For easy description, in the rest of this chapter we call server column \( H_{(i+1)\%k} \) a \textit{clockwise neighboring column} of \( H_i \) and \( H_{(i+k-1)\%k} \) a \textit{counter-clockwise neighboring column} of \( H_i \).

In a DPillar network with \( k \) columns of servers, each server column has \( (n/2)^k \) servers; each switch column has \( (n/2)^{k-1} \) switches, where \( n \) is the number of ports of the switches. For the \( (n/2)^k \) servers in any server column \( H_i \), each of them is assigned with a unique \( k \)-symbol label \((\nu^{k-1}...\nu^0)\), where a symbol \( \nu^i \) (\( 0 \leq i \leq k-1 \)) is an integer number between 0 and \( (n/2 - 1) \). Under this naming scheme, one server in DPillar can be uniquely identified as \((C, \nu^{k-1}...\nu^0)\), which means a server with label
Figure 6.1. The vertical view of a DPillar network.

(...ν \(k-1\) ...ν0) in server column \(H_C\). We call \((C, ν^{k-1}...ν^0)\) the ID or the address of the server.

Given the IDs of the servers in a DPillar network, the interconnection between the servers and the switches is as follows. For all the \(2(n/2)^k\) servers in any server column \(H_C\) and its clockwise neighboring server column \(H_{(C+1)\%k}\), they can be divided into \((n/2)^{k-1}\) groups, with each group having \(n\) servers. The labels of the \(n\) servers in the same group have the following property. That is, their labels are the same if the \(C\)th symbol (i.e., symbol \(ν^C\)) is removed. It is easy to see that among the \(n\) servers within the same group, half of them are from \(H_C\) and the other half are from \(H_{(C+1)\%k}\). The \(n\) servers in the same group are connected to the same switch in switch column \(S_C\). In other words, given any label \((ν^{k-1}...ν^C...ν^0)\), there are \(n/2\) servers in \(H_C\) whose labels are \((ν^{k-1}...ν^C...ν^0)\) where \(0 ≤ ν^C ≤ n/2 − 1\); there are \(n/2\) such servers in \(H_{(C+1)\%k}\) too. Those \(n\) servers are connected to the same \(n\)-port switch in \(S_C\).

Figure 6.2 shows a DPillar network built from 8-port switches. There are two server columns in this network. We duplicate server column \(H_0\), cut the cylindrical surface of the pillar along column \(H_0\), and spread that cylindrical surface into a two-dimension area. As each switch has eight ports and there are two columns of servers
(n = 8, k = 2), a server column has \((\frac{8}{2})^2 = 16\) servers. The label of each server has two symbols. The first row of servers in Figure 6.2 have label (00) and the last row of servers have label (33). If we select a label \((\nu^1 \nu^0) = (00)\), there are four servers in \(H_1\) whose labels are \((\nu^*_1 0)\) with \(0 \leq \nu^*_1 \leq 3\), i.e., (00), (10), (20), and (30). There are four servers in \(H_0\) whose labels are (00), (10), (20), and (30) too. Those eight servers are connected to the same switch in switch column \(S_1\).

### 6.3.2 Topological Properties of DPillar

After presenting the interconnection of DPillar, we proceed to study the basic topological properties of the DPillar network. As we can see from section 6.3.1, a DPillar network is uniquely defined by two parameters, \(k\), the number of server columns, and \(n\), the number of ports of a switch. We call such a DPillar network \((n, k)\) DPillar network for short.
6.3.2.1 Number of servers DPillar accommodates

Clearly, since each server column has \( (n/2)^k \) servers, there are \( k(n/2)^k \) servers in a \((n, k)\) DPillar network. In the rest of this chapter, we use \( N \) to represent the total number of servers in a \((n, k)\) DPillar network, i.e., \( N = k(n/2)^k \).

Proposition 6.3.1 A \((n, k)\) DPillar network can accommodate \( k(n/2)^k \) servers.

Given Proposition 6.3.1, let’s provide some examples on how many servers a \((n, k)\) DPillar network can support. Considering that 48-port Gbit Ethernet switches are widely available now and relatively inexpensive, a \((48, 3)\) DPillar network has 41472 servers. The number of servers will be about 1.3 million for a \((48, 4)\) DPillar network. If we build a \((48, 5)\) DPillar network, it has about 40 million servers.

6.3.2.2 Number of switches used by DPillar

Next we consider the number of \( n \)-port switches used in a DPillar network. This number is important because the switches are the major “networking devices” we will invest in building a DPillar network. As we have mentioned, each switch column has \( (n/2)^{k-1} \) switches in a \((n, k)\) DPillar network. As there are \( k \) switch columns, the total number of switches is \( k(n/2)^{k-1} \). Actually, there is an explanation why there are \( (n/2)^{k-1} \) switches in each switch column. If we change all other symbols in label \((\nu^{k-1}...\nu^C...\nu^0)\) except symbol \( \nu^C \), there are \( (n/2)^{k-1} \) different combinations. Each of those \( (n/2)^{k-1} \) combinations requires one switch to connect \( n \) servers whose labels are \((\nu^{k-1}...\nu^C^*...\nu^0)\) where \( 0 \leq \nu^C^* \leq n/2 - 1 \). Therefore, the number of switches in each switch column is \( (n/2)^{k-1} \) and the total number of switches in a \((n, k)\) DPillar network is \( k(n/2)^{k-1} \).

Proposition 6.3.2 A \((n, k)\) DPillar network uses \( k(n/2)^{k-1} \) switches.
6.3.2.3 Bisection width of DPillar

Bisection width is an important factor to quantify the performance of an interconnection network. It is defined as the smallest number of edges removal of which divides the nodes in the network into two parts of equal size. Larger bisection width means the network can sustain more communications between nodes in the network. Because servers in a data center usually have lots of interactions among them, such as running MapReduce [61] applications, it is desirable that a data center network has large bisection width. The bisection width of a \((n, k)\) DPillar network is \((n/2)^k\), as stated in Proposition 6.3.3.

**Proposition 6.3.3** The bisection width of a \((n, k)\) DPillar network is \((n/2)^k\).

**Proof** Our proof is inspired by previous work [130] in studying the bisection width of butterfly networks.

Clearly, if we cut a \((n, k)\) DPillar network horizontally, i.e., each server column is cut into halves, we can always cut a DPillar network into a top half and a bottom half by cutting the connections among \(H_{k-1}, S_{k-1}\), and \(H_0\). For example, we can divide the DPillar network shown in Figure 6.2 into top and bottom halves by cutting the links cross a “virtual” line between row (13) and row (20). Only some of the connections among \(H_{k-1}, S_{k-1}\), and \(H_0\) will cross that virtual line. Because each switch in \(S_{k-1}\) has \(n/2\) links crossing the virtual line, the total number of links is \((n/2)^{k-1} \times (n/2) = (n/2)^k\). Hence, we have an upper bound, \((n/2)^k\), for the bisection width of a \((n, k)\) DPillar network.

Next we find the lower bound of the bisection width. Let \(G\) denote the number of servers in each column of a \((n, k)\) DPillar network. We consider bisecting the \(2G\) servers in server columns \(S_0\) and \(S_{k-1}\) by embedding a complete bipartite graph \(K_{G,G}\) into a \((n, k)\) DPillar network, so that the left side nodes and right side nodes of \(K_{G,G}\) are mapped to the servers in \(S_0\) and servers in \(S_{k-1}\) of the \((n, k)\) DPillar network,
respectively. If each of the $G$ servers in $S_0$ has a path to every server in $S_{k-1}$, because DPillar network is symmetry, there are at most $G/2$ paths uses the same server-to-switch link. Also because the bisection width of a complete bipartite graph $K_{G,G}$ is $G^2/2$, the size of the cut that bisects the $2G$ servers in $S_0$ and $S_{k-1}$ should be at least $G$.

Now we consider a minimal cut $C$ that bisects all servers in a $(n, k)$ DPillar network into $Set_1$ and $Set_2$. If there exist two neighboring server columns, e.g., $S_i$ and $S_{(i+1)\%k}$, where the $2G$ servers are bisected by cut $C$, we know that the size of cut $C$ is at least $G$. Otherwise, we find two neighbors server columns $S_j$ and $S_{(j+1)\%k}$ so that among the $2G$ servers in those two server columns, more are in $Set_1$ than in $Set_2$. Then we move some servers (among those $2G$ servers in server columns $S_j$ and $S_{(j+1)\%k}$) from $Set_1$ to $Set_2$ so that half of those $2G$ servers are in $Set_1$. Note that moving the servers from $Set_1$ to $Set_2$ does not increase the size of cut $C$. We already know that bisecting the $2G$ servers in $S_j$ and $S_{(j+1)\%k}$ requires cutting at least $G$ links. Hence, the size of cut $C$ has lower bound $G = (n/2)^k$. Also because the upper bound is $(n/2)^k$, the bisection width of a $(n, k)$ DPillar network is $(n/2)^k$.

We can see that the bisection width of a $(n, k)$ DPillar network is equal to the number of servers in each server column. The bisection width of wrapped butterfly network [127] is also equal to the number of nodes in each level\(^1\). However, wrapped butterfly network requires each node to have degree 4 to achieve that bisection width. DPillar network requires each node (the PC server) to have degree 2 (two NICs) and achieves the same bisection width.

\(^1\)For a wrapped butterfly network with $k$ levels, each level has $2^k$ nodes.
Table 6.2. The cost of the networking devices, including switches and cables, when using switches with different number of ports to build DPillar networks with four columns of servers.

<table>
<thead>
<tr>
<th>switch type</th>
<th>8-port</th>
<th>16-port</th>
<th>24-port</th>
<th>48-port</th>
</tr>
</thead>
<tbody>
<tr>
<td>switch unit price</td>
<td>$50</td>
<td>$150</td>
<td>$180</td>
<td>$600</td>
</tr>
<tr>
<td>number of servers</td>
<td>1,024</td>
<td>16,384</td>
<td>82,944</td>
<td>1,327,104</td>
</tr>
<tr>
<td>total networking cost</td>
<td>$14,848</td>
<td>$339,968</td>
<td>$1,410,048</td>
<td>$35,831,808</td>
</tr>
<tr>
<td>per-server cost</td>
<td>$14.5</td>
<td>$20.75</td>
<td>$17</td>
<td>$27</td>
</tr>
</tbody>
</table>

6.3.3 Cost of Building DPillar

DPillar network is cost-efficient as it uses only inexpensive commodity hardware. Here we provide some “example budgets” of building DPillar networks. We ignore the cost of servers and focus on the networking devices of DPillar, including the switches and the Ethernet cables. As most off-the-shelf servers already integrate dual-port interfaces, there is no need to invest on NICs.

The unit prices we get from an online retailing store ([www.newegg.com](http://www.newegg.com)) are $150 for a 16-port Gbit Ethernet switch (TRENDnet TEG-S16R) and $1 for an Ethernet cable. We expect the wholesale price of the switches and cables would be even lower.

For a (16, 4) DPillar network, there are 16,384 servers. The cost of the switches is $307,200. The cost of the cables is $32,768 as we need two cables for each server. The total for the networking devices is about 0.34 million dollars, which means on average it costs about $20 to connect one server in this (16, 4) DPillar network. Table 6.2 shows the total cost and the per-server cost of building DPillar networks with four columns of servers, when different types of switches are used. In general, letting $U_s$ be the unit price of a $n$-port switch and $U_c$ be the unit price of an Ethernet cable, the average cost of connecting one server in the DPillar network is $2(U_s/n + U_c)$. If we ignore the cost of the cables, the average cost of connecting a server in a DPillar network is two times the per-port cost of the switches used in this DPillar network.
6.4 Routing in DPillar Network

Because of the symmetric structure of the DPillar network, routing and packet forwarding in DPillar network can be simple and efficient. In this section, we present a routing scheme which has constant running time in computing routes in DPillar.

6.4.1 Routing Algorithm

The packet routing and forwarding process in DPillar can be divided into two phases. In the first phase, the packet is forwarded from the source server to an intermediate server whose label is the same as the destination server’s label. In the second phase, the packet is sent from that intermediate server to the destination.

We consider a scenario where a server $A$ sends a packet to destination server $B$. The addresses of those two servers are $(C_A, L_A)$ and $(C_B, L_B)$, where $L_A$ and $L_B$ are the $k$-symbol labels of server $A$ and $B$. Let the labels of those two servers be $L_A = (\nu_A^{k-1} ... \nu_A^0)$, $L_B = (\nu_B^{k-1} ... \nu_B^0)$, and $L_A \neq L_B$.

From server $A$ in column $H_{C_A}$, the packet can be sent to a server $A_1$ in column $H_{(C_A+1)\%k}$. The label of $A_1$ is the same as the label of $A$ except the $C_A$th symbol in $A_1$’s label can be any number from 0 to $(n/2 - 1)$. If server $A_1$ sends the packet to server $A_2$ in column $H_{(C_A+2)\%k}$, $A_2$’s label is the same as $A_1$’s except that the $((C_A + 1)\%k)$th symbol of $A_2$’s label can be any number from 0 to $(n/2 - 1)$. We see that when a packet is forwarded for one hop, we can “change” one symbol in the label of the server which receives that packet. When a packet is always forwarded from one server column to its clockwise neighboring server column (see Figure 6.1), within $k$ hops, the packet can reach a server with any given label. For example, in a $(n, k)$ DPillar network, the trace of a packet forwarded from $(0, 0...0)$ to $(k - 1, 1...1)$ can be $(0, 0...0) \rightarrow (1, 0...1) \rightarrow (2, 0...11) \rightarrow (k - 1, 1...1)$. As this path resembles a helix, we call the first phase of the packet forwarding process the helix phase.
Note that in the helix phase, we can always send the packet to either a server in the clockwise neighboring column or a server in the counter-clockwise neighboring column. However, the direction of forwarding a packet should not be changed back and forth in order to avoid loops. Some field in the packet header can be used to record the forwarding direction information of this packet. In DPillar routing, the default direction of forwarding a packet in the helix phase is the clockwise direction.

After the packet is forwarded to an intermediate server $B^*$ whose label is the same as the label of destination server $B$, one can forward the packet to $B$ by always sending it to the server in the clockwise neighboring column whose label is $L_B$ too, or sending along the counter-clockwise direction. We select the shorter one among those two paths in our DPillar routing. Obviously, the shorter one is also the shortest path between $B$ and $B^*$. In other words, suppose server $B^*$ is in column $H_{C_{B^*}}$, $B^*$ sends the packet to a server in column $H_{(C_{B^*}+1)\%k}$ if $(C_B + k - C_{B^*})\%k \leq \left\lfloor \frac{k}{2} \right\rfloor$; otherwise $B^*$ sends the packet to a server in column $H_{(C_{B^*}+k-1)\%k}$. In either case, the label of the next-hop server should be $L_B$. We see that in this phase of packet forwarding, the trace of the packet is like a segment in a ring. We call the second phase the ring phase in packet forwarding.

Algorithm 2 shows the pseudocode of the routing algorithm each server in DPillar runs to decide the next-hop server of forwarding a packet. This algorithm takes the address of the server running this algorithm $(C_A, L_A)$, the destination server’s address $(C_B, L_B)$, and the forwarding direction $D$ as input parameters. $D=1$ means the direction is clockwise; $D=-1$ indicates the counter-clockwise direction. The default value of $D$ should be 1. The output is the address of the next-hop server $(C_P, L_P)$.

In Algorithm 2, first server $A$ checks whether it can directly reach server $B$. There are two cases. (i) If $L_A$ and $L_B$ are the same after removing the $C_A$th symbol, and $B$ is in the same column of $A$ or $B$ is in the clockwise neighboring column of $A$, $A$ can directly reach $B$. (ii) If $L_A$ and $L_B$ are the same after removing the $C_B$th symbol, and
B is in the same column of A or B is in the counter-clockwise neighboring column of A, A can directly reach B as well.

If server A cannot directly reach server B, server A first checks whether its label $L_A$ is the same as $L_B$. (i) If $L_A \neq L_B$, the forwarding should be in the helix phase. Server A always forwards the packet to a server in its clockwise neighboring server column, i.e., column $H_{(C_A+1)\%k}$. The label of the next-hop server is the same as $L_A$ except that the $C_A$th symbol of $L_A$ is changed to the $C_A$th symbol in $L_B$. (ii) If $L_A = L_B$, the label of the next-hop server is fixed to $L_A$ and the packet forwarding is in the ring phase. We choose the one which is closer to column $H_{C_B}$, among the two neighboring server column of $H_{C_A}$, as the column of the next-hop server.

Algorithm 2: SRoute($C_A, L_A, C_B, L_B, D$)

input : ($C_A, L_A$) is the address of the server running this algorithm. ($C_B, L_B$) is the destination. $D$ is the forwarding direction recorded in the packet header (either 1 or -1). $L_A=(\nu^{k-1}_{A}...\nu_{A}^{0})$, $L_B=(\nu^{k-1}_{B}...\nu_{B}^{0})$.

output: Address of the next-hop server ($C_P, L_P$).

/* $L_A-\nu^C_A$ means removing the $C_A$th symbol from label $L_A$ */
1 if {$(C_B+k\cdot C_A)\%k \leq 1$ and $L_A-\nu^C_A==L_B-\nu^C_B$} or {$(C_A+k\cdot C_B)\%k \leq 1$ and $L_B-\nu^C_B==L_A-\nu^C_A$} then /* A can directly reach B */
2 $C_P \leftarrow C_B$;
3 $L_P \leftarrow L_B$;
4 else /* A cannot directly reach B */
5 if $L_A == L_B$ then /* the ring phase */
6 $C_P \leftarrow L_B$;
7 if $(C_B + k - C_A)\%k \leq \lfloor \frac{k}{2} \rfloor$ then $C_P \leftarrow (C_A + 1)\%k$;
8 else $C_P \leftarrow (C_A + k - 1)\%k$;
9 else /* the helix phase */
10 $L_P \leftarrow (\nu^{k-1}_{A}...\nu_{B}^{A}...\nu_{A}^{0})$;
11 $C_P \leftarrow (C_A + D + k)\%k$;
12 return ($C_P, L_P$);
6.4.2 Overhead of Routing and Forwarding

With the aforementioned DPillar routing, there is no need to maintain any routing table in the servers. Each server can determine the next hop of forwarding a packet in constant time, no matter how many servers are there in a DPillar network.

The only table each server needs to maintain is an ARP table which maps the network layer address (such as IP address) to Ethernet MAC address. As each server connects with two \( n \)-port switches, a server is directly connected with \( 2(n - 1) \) servers. The number of entries in the ARP table is \( 2(n - 1) \), which is a constant too. Therefore, once we have selected the type of switches to build a DPillar, the overhead of mapping network layer address to Ethernet MAC address is also a constant which does not depend on the number of servers in a DPillar network. We will present a prototyping implementation of the routing algorithm and the performance evaluation of our implementation in section 6.6.1.

6.4.3 Longest Path in DPillar

The DPillar routing presented in Algorithm 2 does not compute the shortest path between two servers. However, the paths computed by Algorithm 2 have bounded length. Here we study what is the length of the longest path in a DPillar network.

Because we use Ethernet switches as dummy layer-2 connection media, the switches should switch packets at line speed. In counting the path length in a DPillar network, we treat the distance between two servers connecting to the same switch as one hop, although a server needs to go through two server-to-switch links to reach another server connecting to the same switch.

We consider the worst case scenario where the labels of two servers have no common symbols. If the label of server \( A \) and the label of server \( B \) have no common symbols, a packet needs to be forwarded \( k \) times in order to reach a server \( B^* \) whose label is the same as server \( B \)'s label. After that, from server \( B^* \), the packet still needs
to be forwarded \(\lceil k/2 \rceil\) hops in order to reach server \(B\) in the worst case. The longest path computed by Algorithm 2 in a \((n, k)\) DPillar network is \(k + \lceil k/2 \rceil\).

**Proposition 6.4.1** Using Algorithm 2, any two servers in a \((n, k)\) DPillar network can reach each other in at most \(k + \lceil k/2 \rceil\) hops.

**Proof** For a \((n, k)\) DPillar network, the server label has \(k\) symbols. In the worst case, a packet needs to be forwarded \(k\) hops in order to reach a server whose label is the same as the destination’s label. At this point, the server having the packet and the destination server are located along a ring with \(k\) nodes (all those nodes have the same label). In the worst case, the packet needs to be forwarded \(\lceil k/2 \rceil\) more hops to reach the destination server. Therefore, the longest path in a \((n, k)\) DPillar network is \(k + \lceil k/2 \rceil\).

One thing worthy of mention is that in a \((n, k)\) DPillar network, the shortest path from servers \((0, 000)\) to server \((\lfloor \frac{k}{2} \rfloor, 1...1)\) is also \(k + \lceil k/2 \rceil\). Therefore, the physical diameter of a \((n, k)\) DPillar network is also \(k + \lceil k/2 \rceil\).

### 6.4.4 Traffic Distribution in All-to-All Communication

For all-to-all communication in a \((n, k)\) DPillar network, each server has \(N - 1\) flows to other \(N - 1\) servers, where \(N = k(n/2)^k\) is the total number of servers in the network. The total number of flows is \(N(N - 1)\). As each flow traverses at most \(3k/2\) hops\(^2\) and each hop consists of 2 server-to-switch links, the total number of links those \(N(N - 1)\) flows traverse is at most \(3kN(N - 1)\). The total number of server-to-switch links is \(2k(n/2)^k = 2N\) and all links in a DPillar network are identical. Therefore, each link carries about \(3k(N - 1)/2\) flows.

\(^2\)We ignore the floor operator for simplicity.
Proposition 6.4.2 In all-to-all communication where each server in a \((n, k)\) DPillar network has one flow to each of the other servers, a server-to-switch link carries at most \(3k(N - 1)/2\) flows.

6.5 Handling Failure and Balancing Load in DPillar

The rich connections in DPillar facilitate the design of simple yet efficient fault-tolerant routing scheme. In this section, we present the design of a routing scheme which can bypass a wide range of failures in DPillar. We also discuss how to use the insights gotten from designing the fault-tolerant routing scheme to balance the traffic load in DPillar.

6.5.1 Discovering Failures

To bypass failures in DPillar network, the first step would be detecting the failures. Each server in DPillar runs a lightweight Hello protocol to report the reachability to other servers connected to the same switches. If a server \(A\) does not hear Hello message from server \(B\) for a certain period of time, server \(A\) assumes \(B\) is not directly reachable. Servers should not forward any Hello messages.

We can set the period of sending Hello messages to be slightly smaller than the time for ARP cache to timeout. Therefore, the server can always have “fresh” ARP cache so it does not need to query for the MAC address of a neighbor connecting to the same switch before sending out any data packets to that neighbor.

6.5.2 Bypassing Failures

We consider a scenario where a server \(A\) needs to use \(P\) as the next hop server to reach destination server \(B\), according to the DPillar routing presented in Algorithm 2. However, server \(P\) is not reachable from \(A\), so \(A\) should try to bypass this failure by sending packets to another next hop \(P'\). Suppose the IDs of those four servers are
(C_A, L_A), (C_B, L_B), (C_P, L_P), and (C_{P'}, L_{P'}). We discuss how to bypass the failure in the ring phase and the helix phase separately.

6.5.2.1 The ring phase

Bypassing a failure in the ring phase is relatively straightforward. As there are two ways to forward a packet to the destination in the ring phase, either along the clockwise direction or the counter-clockwise direction, a packet should be forwarded along the other direction, if it cannot be forwarded along one direction. However, to avoid forwarding loops, the forwarding direction of a packet should not be changed more than once in the ring phase. If a packet has changed its forwarding direction in the ring phase and it encounters a failed server again, that packet should be dropped.

6.5.2.2 The helix phase

When a failure is detected in the helix phase, i.e., nexthop P is unreachable, A first tries to bypass the failed server by sending the packet to a reachable server in the clockwise neighboring column. If none of the servers in the clockwise neighboring column is reachable, the packet will be forwarded to a server in the counter-clockwise neighboring column of server A.

![Diagram](image)

**Figure 6.3.** A (4, 3) DPillar network example.
Select a clockwise nexthop: How to select the label of the new nexthop server is important for avoiding any forwarding loops. We use the example shown in Figure 6.3 to explain the reason. Let’s consider the scenario that server A, whose address is (0,000), sends a packet to server B, whose address is (2,001). According to Algorithm 2, the ID of the nexthop server $P$ should be (1,001) but server (1,001) has failed. Suppose server A tries to detour the failed server (1,001) by sending the packet to a new nexthop server (1,000), the path taken by the packet will be (0,000) → (1,000) → (2,000) → (0,000), which is a loop.

We see that the reason for the aforementioned loop to occur is that, although A detours the packet and sends it to server (1,000), the packet will reach A again because A is an intermediate node from server (1,000) to destination B, according to Algorithm 2. To avoid this loop, we can let server (1,000) not forward the packet to (2,000). For example, if (1,000) forwards the packet to (2,010), the path taken by the packet will be (0,000) → (1,000) → (2,010) → (0,010) → (1,011) → (2,001).

The insights here are two-fold: (i) Forwarding from (0,000) to (1,000) will detour the failed server (1,001); (ii) Forwarding from (1,000) to (2,010) will ensure (0,000) is not in the path from (2,010) to (2,001), because when the packet reaches a server in column 0 again, the address of that server must be $(0,x1y)$, where $x, y \in \{0, 1\}$.

In the above example, to make sure the packet is forwarded from (0,000) to (2,010), we use tunneling. That is, server (0,000) encapsulate the packet with another outer header, whose destination is set to (2,010). Tunneling the packet between (0,000) and (2,010) will make sure the path of the encapsulated packet is (0,000) → (1,000) → (2,010). Therefore, the failed server (1,001) is bypassed.

We conclude the discussion into the following Proposition 6.5.1 regarding how to bypass a failed servers in the helix phase. Note that we omit the $\%k$ in presenting the indexes for clarity.
Proposition 6.5.1 When sending a packet to destination \((C_B, \nu_B^{k-1}\ldots\nu_B^0)\), server \((C_A, \nu_A^{k-1}\ldots\nu_A^0)\) can bypass server \((C_A + 1, \nu_A^{k-1}\ldots\nu_A^0)\) by tunneling the packet to \((C_A + 2, \nu_A^{k-1}\nu_{P'}^{C_A+1}\nu_A^0)\), if \(\nu_{P'}^{C_A} \neq \nu_B^0\) and \(\nu_{P'}^{C_A+1} \neq \nu_A^0\).

Proof Because of the symmetric structure of DPillar, without losing generality, we consider server \(A\), with address \((0, \nu_A^{k-1}\ldots\nu_A^0)\), bypasses server \((1, \nu_A^{k-1}\ldots\nu_B^0)\) in reaching destination \((C_B, \nu_B^{k-1}\ldots\nu_B^0)\). Server \(A\) tunnels the packet to server \(P'\), whose address is \((2, \nu_A^{k-1}\nu_{P'}^1\nu_B^0)\), and we have \(\nu_{P'}^1 \neq \nu_A^1, \nu_{P'}^0 \neq \nu_B^0\).

After the packet is forwarded from \(P'\) to some server \(A_0\) in column \(H_0\), the address of \(A_0\) is \((0, \nu_B^{k-1}\nu_{P'}^1\nu_B^0)\). Because \(\nu_{P'}^1 \neq \nu_A^1, A_0\) is not \(A\). \(A_0\) forwards the packet to a server \(A_1\) in column \(H_1\) whose address is \((0, \nu_B^{k-1}\nu_{P'}^1\nu_B^0)\). Because \(\nu_{P'}^0 \neq \nu_B^0, A_1\) will not be the failed server \((1, \nu_A^{k-1}\ldots\nu_B^0)\). After \(A_1\) forwards the packet, it reaches some server \(A_2\) in column \(H_2\) whose label is the same as the destination. After that, it is the ring phase and we can always bypass a failure in ring phase by changing the packet forwarding direction.

Make a “packet u-turn”: Proposition 6.5.1 assumes a server can always forward a packet to another server in its clockwise neighboring column in the helix phase. However, it is possible that a server \(A\) cannot send the packet to any servers in its clockwise neighboring column, e.g., the server \((0,0,0)\) in Figure 6.3 has a link failure so it is disconnected from the top switch between \(H_0\) and \(H_1\), or the top switch between \(H_0\) and \(H_1\) has failed. If that is the case, server \(A\) needs to change the packet forwarding direction (make a “u-turn”) to bypass a failure in the helix phase, i.e., if a packet cannot be forwarded to a server in the clockwise neighboring column, it can be forwarded to a server in the counter-clockwise neighboring column. The forwarding direction information should be recorded in the packet header, so that other servers will forward this packet along the new direction. Similar to the rationale
behind Proposition 6.5.1, we can prove the following Proposition 6.5.2. Again, the \(k\) is omitted in presenting the indexes for clarity.

**Proposition 6.5.2** When sending a packet to destination \((C_B, \nu_B^{k-1}...\nu_B^0)\), if server \((C_A, \nu_A^{k-1}...\nu_A^0)\) cannot reach any server in \(H_{C_A+1}\), it can bypass the failure by changing the packet forwarding direction and sending the packet to server \((C_A-1, \nu_A^{k-1}...\nu_{P'}^{C_A-1}...\nu_A^0)\), where \(\nu_{P'}^{C_A-1} \neq \nu_A^{C_A-1}\).

**Proof** We consider the scenario where server \(A\), whose address is \((0, \nu_A^{k-1}...\nu_A^0)\), sends a packet to destination \(B\) \((C_B, \nu_B^{k-1}...\nu_B^0)\) but \(A\) cannot send the packet to any server in \(H_1\) during the helix phase. To bypass the failure, server \(A\) changes the forwarding direction of the packet and sends it to nexthop \(P'\) whose address is \((k-1, \nu_A^{k-1}...\nu_{A'}^0)\), where \(\nu_{P'}^{k-1} \neq \nu_A^{k-1}\).

After the packet is forwarded along the counter-clockwise direction from \(P'\) to some server \(A_1\) in \(H_1\), the address of \(A_1\) should be \((1, \nu_{P'}^{k-1}...\nu_B^{1}...\nu_A^0)\). Because \(\nu_{P'}^{k-1} \neq \nu_A^{k-1}\), \(A_1\) is not a server connected to server \(A\) by the same switch and \(A_1\) can directly reach some servers in \(H_0\). After \(A_1\) forwards the packet to some server \(A_0\) in \(H_0\), \(A_0\)'s address is \((0, \nu_{P'}^{k-1}...\nu_B^0)\). Because \(\nu_{P'}^{k-1} \neq \nu_A^{k-1}\), \(A_0\) is not \(A\) and there will be no loops. After \(A_0\) forwards the packet to a server \(A_{k-1}\) in column \(H_{k-1}\), the packet reaches a server whose label is same as the label of \(B\).

**Selecting a counter-clockwise nexthop:** After the forwarding direction of a packet is changed, it should be recorded in the packet header so that all DPillar servers will follow the new direction in forwarding that packet. To avoid potential forwarding loops, when the forwarding direction of a packet was changed in the helix phase, that packet should not change direction again. In other words, for a packet whose direction was changed before from clockwise to counter-clockwise and it cannot be forwarded along the new direction (no reachable servers in the counter-clockwise neighboring column), the packet should be dropped. If server \(A\) needs to forward
packet to server $P$ in the counter-clockwise neighboring column but $P$ is unreachable, $A$ can bypass the failure according to the following Proposition 6.5.3, when there are reachable servers in the counter-clockwise neighboring column. The proof of Proposition 6.5.3 is similar to the proof of Proposition 6.5.1. We omit it here to save space.

**Proposition 6.5.3** When sending a packet to destination $(C_B, \nu^{k-1}_B \ldots \nu^0_B)$, server $(C_A, \nu^{k-1}_A \ldots \nu^0_A)$ can bypass server $(C_A - 1, \nu^{k-1}_A \ldots \nu^{C_A-1}_B \ldots \nu^0_A)$ by tunneling the packet to $(C_A - 2, \nu^{k-1}_A \ldots \nu^{C_A-1}_{P'} \nu^{P'-2}_A \ldots \nu^0_A)$, if $\nu^{(C_A-1)}_{P'} \neq \nu^{C_A-1}_B$ and $\nu^{(C_A-2)}_{P'} \neq \nu^{(C_A-2)}_A$.

**6.5.3 The Fault-tolerant Routing Algorithm**

Based on the discussion in section 6.5.2, we design the DPillar fault tolerant routing algorithm as shown in Algorithm 3. This fault-tolerant algorithm first calls Algorithm 2 to compute a nexthop $(C_P, L_P)$. Then Algorithm 3 tests whether $(C_P, L_P)$ is reachable. If it is, Algorithm 3 does nothing. Otherwise, Algorithm 3 tries to find a new nexthop according to the basic ideas specified in Proposition 6.5.1 $\sim$ Proposition 6.5.3 and returns a new nexthop. If the new nexthop is null, it means the packet should be dropped to prevent forwarding loops.

**6.5.4 Traffic-aware Routing in DPillar Networks**

It is worthy to highlight that because DPillar has totally symmetrical structure, traffic in a DPillar network should be evenly distributed in the long term, if communications happen evenly between any source and destination servers, and the volume of each communication is also evenly distributed. However, in the short term, traffic bursts may overload some servers and we focus on how to alleviate that situation.

We make the traffic balancing decision in a local manner to avoid complicating the DPillar routing too much. Basically, each server monitors the load of its directly connected neighboring servers (servers one hop away). The forwarding engine of each
Algorithm 3: FTRoute\((C_A, L_A, C_B, L_B, D)\)

**input**: \((C_A, L_A)\) is the address of the server running this algorithm. \((C_B, L_B)\) is the address of the destination. \(D\) is the forwarding direction, either 1 or -1.

**output**: Address of the nexthop server \((C_P, L_P)\).

/* call SRoute to compute a nexthop */
\[(C_P, L_P) = SRoute(C_A, L_A, C_B, L_B, D);\]
1 if \((C_P, L_P)\) is reachable then /* no failure */
2 return \(((C_P, L_P))\);
/* try to bypass the failure */
3 if \(L_A == L_B\) then /* the ring phase */
4 if the packet direction was changed then
5 \(C_P \leftarrow\) null; \(L_P \leftarrow\) null;
6 else
7 if \((C_A + k - C_B) \% k \leq \lfloor k/2 \rfloor\) then \(C_P \leftarrow (C_A + 1) \% k;\)
8 else \(C_P \leftarrow (C_A + k - 1) \% k;\)
9 else /* the helix phase */
10 if the packet direction was changed then
11 apply Proposition 6.5.3 to get new nexthop;
12 if nexthop exists then set \(C_P\) and \(L_P\);
13 else \(C_P \leftarrow\) null; \(L_P \leftarrow\) null;
14 else
15 apply Proposition 6.5.1 to get new nexthop;
16 if nexthop exists then set \(C_P\) and \(L_P\);
17 else
18 change packet direction according to Proposition 6.5.2;
19 if nexthop exists then set \(C_P\) and \(L_P\);
20 else \(C_P \leftarrow\) null; \(L_P \leftarrow\) null;
21 return \((C_P, L_P)\);

server maintains some statistics regarding the history of packet forwarding during the last few seconds or minutes, such as the packet dropping rate during the last few minutes, the CPU share used in packet forwarding. We can extend the Hello protocol discussed in section 6.5.1 to also report the server load information.

When a server \(A\) forwards packets, it should take into account the load of those neighbors in selecting the nexthop. If a nexthop server \(P\) is overloaded, \(A\) should avoid using \(P\) and select another nexthop. The exact same rationales we have discussed in
presenting the fault-tolerant routing scheme can be adopted to bypass the overload server $P$.

6.6 Evaluations

We evaluate DPillar from two different aspects. First, using a small testbed, we study the microscopic behavior of DPillar by measuring the overhead of one DPillar server in computing nexthop and in forwarding traffic for other servers. Second, we study the macroscopic behavior of DPillar by simulating the packet routing and forwarding in a large scale DPillar network, using a simulation tool we developed.

6.6.1 Implementing DPillar Routing & Measuring the Performance

6.6.1.1 Implementation

We have implemented the static routing algorithm presented in section 6.4 as an element in the Click software router [113]. In our implementation, we use IP address to encode the column and label information of each server, so as to provide backward comparability to the upper layer applications running in the data center and those applications can still use TCP/IP. We use the most significant 8 bits in the IP header to represent the column number of a server. As each host has two NICs, the least significant bit in the IP header is used to represent the NICs. The $k$-symbol label consumes $k\lceil\log_2(n) − 1\rceil$ bits. In our implementation, the 32 bits in the IPv4 address is allocated as shown in Figure 6.4.

Figure 6.4. Configure the IP addresses of each host according to the host column and label information.
Suppose we use 48-port switches to build a DPillar network with 4 columns of hosts. The total number of hosts in this DPillar network is about 1.3 million. We need to use 2 bits in the most significant 8 bits in the IP header to represent the column. In the rest 24 bits, we use $4\lceil\log_2(48) - 1\rceil + 1 = 21$ bits to represent the label of each server and the interface.

6.6.1.2 Performance measurement

The testbed network. We use a PC with 2.4GHz dual-core CPU and 1GB memory to run the DPillar routing and forwarding element implemented in Click. A testbed network consisting of three servers, as shown in Figure 6.5, are used in our experiment. In Figure 6.5, server $P$ is a DPillar server which forwards packets between server $A$ and server $B$.

Overhead of processing one packet. For each packet forwarded by server $P$, we record the time it takes to compute the nexthop server. Our measurement results show that the static routing algorithm of DPillar consumes about 90 CPU cycles to compute the nexthop server for one packet. The fault-tolerant routing algorithm of DPillar consumes about 250 CPU cycles.

![Figure 6.5. The testbed network. Server $P$ forwards packets between $A$ and $B$.](image)

For comparison purpose, we also measure the time it takes to do table lookup in the conventional IP forwarding implemented in Click (the RadixIPLookup element), when the table size varies. Figure 6.6 plots the results of our experiments. One table lookup takes more than 600 CPU cycles when the table has 128 entries. As the number of entries in the table increases, it takes longer time to do lookup.
Figure 6.6. The number of CPU cycles one table lookup operation consumes in conventional IP forwarding.

**Overhead of forwarding traffic.** We also test the overhead in terms of CPU usage when a DPillar server forwards traffic for other servers. In this experiment, server $A$ and $B$ in Figure 6.5 use `iperf` to send out UDP traffic to each other at various speed and server $P$ forwards the traffic between $A$ and $B$. We record the CPU usage of the Click kernel thread and plot the result in Figure 6.7.

Figure 6.7. The server CPU usage under various traffic load. The packet size is 1024 bytes.

The results in Figure 6.7 show that even the DPillar server forwards traffic at full load, i.e., 1 Gbps each direction, 2 Gbps in total, the CPU usage of the DPillar server is less than 50%. Our server is a dual-core machine and the less than 50% CPU
usage is for one CPU core. The other core is almost 100% idle. As commodity PCs with multi-core CPUs (dual-core or quad-core) are becoming common, we expect the traffic forwarding overhead of the DPillar server can be amortized.

Another way to decrease the CPU load but still maintain the throughput is to use larger packets [57]. As *Jumbo Frame packet transfer* is commonly supported in commodity Ethernet switches³, we can take advantage of this feature to reduce the CPU load of DPillar servers in relaying traffic for other servers.

### 6.6.2 Simulation Evaluations

We also developed a simulation tool to simulate the packet routing and forwarding in a large scale DPillar network. Given the number of server columns $k$ and the switch port number $n$, our simulation tool builds a $(n, k)$ DPillar network topology and simulates how each server routes packets.

#### 6.6.2.1 Average Path Length

In our simulations, we focus on a scenario where the DPillar network is built from 12-port switches, i.e., $n = 12$, and the number of server columns $k$ varies. For each network, we randomly select 100,000 source-destination pairs and simulate packet routing and forwarding in the network. We also simulate the shortest paths between those source-destination pairs. Figure 6.8 plots the results of the average path lengths.

The results show that the path lengths are proportional to the number of server columns in a DPillar network. Using the simple routing algorithm does not inflate the path length too much, especially when the number of server columns $k$ is relatively small.

³The $150 16$-port Gbit switch TRENDnet TEG-S16R mentioned in section 6.3.3 supports up to 12.2k bytes jumbo frame.
6.6.2.2 Average Path Length in Failure-tolerant Routing

We also study the fault-tolerant behavior of DPillar routing. In this simulation, we use a (12,4) DPillar network to conduct our experiments. In each simulation instance, we first randomly fail a certain number of servers in the network. Then we randomly select 100,000 source-destination pairs and simulate the fault-tolerant routing scheme presented in Algorithm 3. We range the number of failed servers from 1 to 300. The average path length, as a function of the number of failed hosts, is plotted in Figure 6.9.
Our simulation results show that the average path length increases as there are more failed servers in the DPillar network. However, no packet drop occurs in our simulations, even when 300 servers in the DPillar network have failed. As a (12, 4) DPillar network has about 5,000 servers, 300 server failures mean 6% of the servers are failed.

6.7 Conclusion

This chapter presents DPillar, a scalable data center network architecture which uses only commodity off-the-shelf hardware. DPillar can easily scale to huge number of servers without imposing any additional requirements to the devices, such as installing additional NICs in the servers. The topology of DPillar is totally symmetric and a DPillar network has balanced network capacity. DPillar is a server centric and the networking intelligence is placed in the servers, so that the switches used in DPillar are merely dummy layer-2 devices connecting the servers. We designed simple yet efficient routing schemes for DPillar. Our prototyping implementation and simulation studies show that the routing schemes are lightweight, high-performance, and efficient in bypassing failures in the network.
CHAPTER 7
CONCLUSION

Internet is a dynamic system that evolves over time. Although Internet has been a huge success since it was created more than four decades ago, there are still many new problems needed to be addressed in the current Internet. As Internet evolves, we can expect more challenges will appear in the future. This dissertation presents the research work in diversifying the Internet, with the motivations of providing more reliable, robust, and diverse services in Internet. Four interesting problems are studied in this dissertation. First, the policy diversity in Internet inter-domain routing is explored and a series of practical policy guidelines are presented. Those policy guidelines safely accommodate the diverse mutual transit commercial agreements and provide more path diversity in inter-domain routing. The second part of this dissertation studies adopting multiple BGP routing processes to exploit the Internet AS-level path diversity. Those multiple routing processes proactively compute multiple complementary paths to the same destination, so that in case of failure and outage, one of the routing processes always produces a working path to the destination. Third, providing diverse data plane functions via network virtualization is studied and two virtual network platforms are presented. These platforms achieve both high degree of flexibility for a virtual network to customize its data plane functions, and high packet forwarding speed for each virtual network. The last part of this dissertation presents a scalable server-centric data center network structure. This new data center network uses off-the-shelf commodity PC servers and low-end layer-2 Ethernet switches. Be-
sides, this network structure expands to any number of servers without requiring to physically upgrading the existing servers.

There are several interesting directions for future work that one could pursue based on the work presented in this dissertation.

**Accommodating more general commercial agreements:** Although the mutual transit agreements studied in this dissertation cover a range of diverse commercial agreements in Internet, we still lack a systematic and practical way to accommodate an exhausted list commercial agreement in inter-domain routing. Accommodating more general and diverse agreements is both challenging and important, because more those commercial agreements will appear as Internet grows and evolves.

**Inter-domain routing system for future Internet:** The multiple process routing scheme proposed in this dissertation adopts the philosophy of reusing the existing components as much as possible. However, a “clean slate design” sometimes could be more straightforward to address the problems we have encountered in the inter-domain routing system. Designing future inter-domain routing system is an interesting and challenging problem. To better support inter-domain routing in future Internet, the routing system must be not only scalable and flexible, which are critical fast convergence and minimal transient routing problems during convergence should are also required. The incentives to deploy such a new routing system and how partial deployment will benefit or impact the whole routing system are also important questions one have to address.

**Hardware-based network virtualization:** The work on network virtualization presented in this dissertation also open several new directions for future research. Although software-based virtual routers are more flexible for customization, the packet forwarding speed of software virtual routers cannot match the hardware counterparts. To virtualizing the backbone networks, we must resort to hardware-based solutions. However, providing high degree of flexibility in hardware virtual routers is a chal-
lenging problem one has to address. Solving this problem may require the design of novel router architectures, which can inherently support efficient virtualizing both the control plane and the data plane of a router.

**Operation cost-efficient data centers:** The data center network structure presented in this dissertation emphasizes the building cost of data centers. However, in reality the operation cost of data centers with huge number of servers is also considerable and energy efficiency is an increasingly important issue for data centers. Addressing this issue requires efforts in the design of both the server side and the interconnection network side. There are many interesting practical problems to study, such as server virtualization, virtual machine scheduling and migration, and power-aware routing protocols supporting efficient server virtualization.
BIBLIOGRAPHY


based operating system virtualization: a scalable, high-performance alternative
to hypervisors,” in EuroSys ’07: Proceedings of the 2nd ACM SIGOPS/EuroSys
European Conference on Computer Systems 2007. New York, NY, USA: ACM,


[103] S. Bhatia, M. Motiwala, W. Muhlauer, Y. Mundada, V. Valancius, A. Bavier,
N. Feamster, L. Peterson, and J. Rexford, “Hosting virtual networks on com-


[105] VMware White Paper, “Understanding Full virtualization, Paravirtualization,
paravirtualization.pdf.


[108] M. B. Anwer and N. Feamster, “Building a fast, virtualized data plane with
programmable hardware,” in Proceedings of VISA 2009 - The First ACM SIG-
COMM Workshop on Virtualized Infrastructure Systems and Architectures, Aug.
2009.


M. Manesh, S. Nedevschi, and S. Ratnasamy, “Can software routers scale?” in
PRESTO ’08: Proceedings of the ACM workshop on Programmable routers for
extensible services of tomorrow, 2008.

[111] P. Crescenzi and V. Kann, “A compendium of NP optimization problems,”
online document: http://www.nada.kth.se/~viggo/problemlist/.


179


