Copyright Statement

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand).

This thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- Any use you make of these documents or images must be for research or private study purposes only, and you may not make them available to any other person.
- Authors control the copyright of their thesis. You will recognise the author's right to be identified as the author of this thesis, and due acknowledgement will be made to the author where appropriate.
- You will obtain the author's permission before publishing any material from their thesis.

To request permissions please use the Feedback form on our webpage.
http://researchspace.auckland.ac.nz/feedback

General copyright and disclaimer

In addition to the above conditions, authors give their consent for the digital copy of their work to be used subject to the conditions specified on the Library Thesis Consent Form and Deposit Licence.
Evaluating and Improving SHIM6 and MPTCP: Two Solutions for IPv6 Multihoming

Habib Naderi

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science, The University of Auckland, July 14, 2014
Abstract

Site multihoming refers to a situation when a site has access to the Internet through more than one Internet service provider. In the current Internet, multihoming is achieved by using Border Gateway Protocol. A multihomed site acquires its provider independent or provider aggregatable address block and then announces this address block through all its service providers to the Internet’s global routers. This solution works for IPv4, but scalability is an issue for IPv6. High demand for multihoming and the huge address space provided by IPv6 requires a solution which is able to scale well.

A wide range of solutions have been proposed for IPv6 multihoming during past years. These solutions can be put into three categories: Routing approaches, Middle-Box approaches and Host-centric approaches. The first part of our research focuses on an analysis, from a deployability viewpoint, of seven proposed solutions which are still under investigation by researchers and Internet experts. The analysis shows that there is no perfect solution yet and all proposed solutions still need to be improved. However, host-centric solutions are more attractive from the viewpoint of deployability since they do not need any change in the Internet routing system. SHIM6 and MultiPath TCP (MPTCP) are two incrementally deployable solutions belonging to this category. Failure detection and recovery without breaking active communications is one of the main requirements for every multihoming solution. MPTCP and SHIM6 are both able to provide this functionality. SHIM6 is equipped with a specific failure detection and recovery protocol (REAChability Protocol) and MPTCP employs a central congestion control mechanism (SEMICOUPLED Congestion Control Algorithm) for this purpose. These protocols are not yet widely deployed on the Internet, so the efficiency of these protocols is still an open research area. The second part of the work presented in this thesis focuses on this research area by conducting a series of real-world and simulation experiments for SHIM6 and simulation experiments for MPTCP which have been designed to investigate the efficiency of REAP and SEMICOUPLED.

Traffic Engineering is one of the major weak points of SHIM6. Since the solution is implemented in end-hosts, having a site-aware traffic engineering mechanism is challenging. To address this issue, we propose a dynamic traffic engineering mechanism which offers a rich set of traffic engineering features for SHIM6-enabled sites.
Acknowledgements

Firstly, I would like to thank my supervisors, Associate Professor Nevil Brownlee, Professor Brian Carpenter and Dr Ulrich Speidel, for their support and guidance during the time of this research. They always provided interesting ideas and useful advice. I always learned something interesting from our meetings. I am very grateful for their help, support and encouragement throughout this research.

I would like to thank John Ronan from Telecommunications Software & Systems Group, Waterford Institute of Technology, and the University of Auckland Information Technology Services (ITS) who helped me run SHIM6 experiments over the Internet between Auckland and Dublin. I really appreciate their cooperation in that part of my research.

I would like to thank Pita Jarupunphol, Yu-Cheng Tu, Moon-Ting Su, Steven Hu, Se-Young Yu, Maziar Janbeglou, Marc Jeanmougin, and many friends from the Department of Computer Science for their friendship and support. We have always had interesting discussions about various aspects of our research. Moreover, thank you to Robyn Young and Sithra Sukumaar from the Department of Computer Science for providing me with all the necessary help during the research.

Finally, I would like to thank my wife, Manizheh, and my family for providing me support and for coping with me throughout the time of this research. I am very grateful for their support and encouragement.

I would also like to mention that section 4.10 has been derived from [89] and Chapter 2 has been derived from [90].
Abbreviations

ACK  ACKnowledgement
API  Application Programming Interface
BEX  Base EXchange
BGP  Border Gateway Protocol
CGA  Cryptographically Generated Addresses
DCCP  Datagram Congestion Control Protocol
DFZ  Default Free Zone
DHT  Distributed Hash Table
DNS  Domain Name Service
Dos  Denial of Service
DSN  Data Sequence Number
ECN  Explicit Congestion Notification
EID  End-point Identifier
ETR  Egress Tunnel Router
FQDN  Fully Qualified Domain Name
GSPN  Generalized Stochastic Petri Net
HBA  Hash Based Addresses
HIP  Host Identity Protocol
ICMP  Internet Control Message Protocol
ID  IDentifier
IETF  Internet Engineering Task Force
ILNP  Identifier/Locator Network Protocol
IP  Internet Protocol
IPv4  Internet Protocol version 4
IPv6  Internet Protocol version 6
IRTF  Internet Research Task Force
ISP  Internet Service Provider
ITR  Ingress Tunnel Router
LISP  Locator/ID Separation Protocol
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>MITM</td>
<td>Man In The Middle</td>
</tr>
<tr>
<td>MPTCP</td>
<td>MultiPath TCP</td>
</tr>
<tr>
<td>MTU</td>
<td>Maximum Transmission Unit</td>
</tr>
<tr>
<td>NAROS</td>
<td>Name, Address and ROute System</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>NAT44</td>
<td>IPv4-to-IPv4 NAT</td>
</tr>
<tr>
<td>NBS</td>
<td>Name Based Socket</td>
</tr>
<tr>
<td>NTPv6</td>
<td>IPv6-to-IPv6 Network Prefix Translation</td>
</tr>
<tr>
<td>PA</td>
<td>Provider Aggregatable</td>
</tr>
<tr>
<td>PI</td>
<td>Provider Independent</td>
</tr>
<tr>
<td>PKI</td>
<td>Public Key Infrastructure</td>
</tr>
<tr>
<td>REAP</td>
<td>REAchability Protocol</td>
</tr>
<tr>
<td>RIR</td>
<td>Regional Internet Registry</td>
</tr>
<tr>
<td>RLOC</td>
<td>Routing LOCator</td>
</tr>
<tr>
<td>RRG</td>
<td>Routing Research Group</td>
</tr>
<tr>
<td>RTT</td>
<td>Round Trip Time</td>
</tr>
<tr>
<td>SAN</td>
<td>Stochastic Activity Network</td>
</tr>
<tr>
<td>SADR</td>
<td>Source Address Dependent Routing</td>
</tr>
<tr>
<td>SCTP</td>
<td>Stream Control Transmission Protocol</td>
</tr>
<tr>
<td>SPN</td>
<td>Stochastic Petri Net</td>
</tr>
<tr>
<td>TE</td>
<td>Traffic Engineering</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TFRC</td>
<td>TCP Friendly Rate Control</td>
</tr>
<tr>
<td>TPN</td>
<td>Timed Petri Net</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>ULA</td>
<td>Unique Local Address</td>
</tr>
<tr>
<td>ULID</td>
<td>Upper Layer IDentifier</td>
</tr>
<tr>
<td>ULP</td>
<td>Upper Layer Protocol</td>
</tr>
</tbody>
</table>
# Contents

1 Introduction 1
   1.1 Problem Statement and Motivation 4
   1.2 Thesis Overview 9

2 Multihoming Solutions for IPv6 11
   2.1 Introduction 12
   2.2 Proposed Solutions in the Area 15
      2.2.1 LISP 16
      2.2.2 ILNP 17
      2.2.3 NPTv6 18
      2.2.4 HIP 19
      2.2.5 Name-based Sockets 19
      2.2.6 SHIM6 20
      2.2.7 MPTCP 23
   2.3 Analysis of the proposed solutions 24
      2.3.1 Analysis of LISP 25
      2.3.2 Analysis of ILNP 27
      2.3.3 Analysis of NPTv6 28
      2.3.4 Analysis of HIP 29
      2.3.5 Analysis of Name-based Sockets 30
      2.3.6 Analysis of SHIM6 31
      2.3.7 Analysis of MPTCP 32
      2.3.8 Summary of the Analysis 33
   2.4 Conclusion 34

3 Stochastic Activity Networks 37
   3.1 Introduction 38
   3.2 SAN Elements 40
   3.3 Dynamic Behaviour of the SAN 41
      3.3.1 Example 1 42
## CONTENTS

3.3.2 Example 2 ........................................... 43
3.4 Möbius Modeling Tool .................................... 45

4 SHIM6 ............................................. 49

4.1 Introduction .......................................... 51
4.2 Related Work .......................................... 52
4.3 REAchability Protocol .................................. 52
4.4 Congestion Control Algorithms ......................... 54
    4.4.1 TCP Congestion Control Algorithm ............... 55
    4.4.2 TCP Friendly Rate Control ....................... 55
4.5 Internet Experiments .................................. 57
    4.5.1 Experiment Setup ................................ 57
    4.5.2 Results from Internet Experiments ............... 60
4.6 Lab Experiments ...................................... 65
    4.6.1 Experiment Setup ................................ 66
    4.6.2 Results from Lab Experiments .................... 67
4.7 Operational Issues .................................... 70
4.8 Model Description ..................................... 71
    4.8.1 REAP Sub-model ................................ 71
    4.8.2 Application Sub-model .......................... 72
        4.8.2.1 DCCP Application ..................... 74
        4.8.2.2 TCP Application ...................... 74
        4.8.2.3 Failure/Recovery Detection ............. 76
    4.8.3 SHIM6 Sub-model ............................... 76
    4.8.4 Limitations .................................... 76
4.9 Validating the Model .................................. 78
4.10 REAP Performance in Large Scale Networks .......... 80
    4.10.1 REAP Recovery Time ........................... 81
    4.10.2 REAP Traffic .................................. 82
    4.10.3 Effect of the REAP Parameters on Performance 84
        4.10.3.1 Send Timer .............................. 85
        4.10.3.2 Initial Probes ......................... 85
4.11 TFRC Improvement ................................... 89
4.12 Transport to REAP Notification ....................... 91
4.13 Conclusion ......................................... 92

5 MPTCP ............................................. 95

5.1 Introduction .......................................... 96
### CONTENTS

5.2 Related Work ................................ 97  
5.3 SEMICOUPLED ($q=1$) Congestion Control Algorithm .......... 98  
5.4 Model Description ................................ 99  
5.5 Validating the Model .............................. 104  
5.6 Increasing Number of Subflows ...................... 106  
5.7 Path Failure ................................ 110  
5.7.1 Permanent Failure ........................... 110  
5.7.2 Temporary Failure ........................... 114  
5.8 Shared Bottleneck .............................. 117  
5.9 Variant Loss Rate ............................. 121  
5.10 Conclusion .................................. 123  

6 Traffic Engineering in SHIM6 125  
6.1 Introduction ................................ 126  
6.2 Traffic Engineering Solutions for SHIM6 ......................... 129  
6.2.1 Proxy SHIM6 ............................ 130  
6.2.2 Extended SHIM6 .......................... 131  
6.3 NAROS: A Traffic Engineering Solution for Host-centric Multihoming 132  
6.4 Adopting NAROS in SHIM6-enabled Networks (SHIM6-enabled NAROS) 134  
6.4.1 Reporting Locator Preferences to NAROS ............ 135  
6.4.2 Enforcing Dynamic TE Policies .................. 137  
6.5 SHIM6-enabled NAROS vs Proxy SHIM6 and Extended SHIM6 .... 140  
6.6 Conclusion .................................. 142  

7 Conclusion ........................................ 145  
7.1 Results and Contributions ........................... 146  
7.1.1 Which solution can be the winner from a deployability viewpoint? 146  
7.1.2 How efficient is REAP in handling communication path failures and how can it be improved? 146  
7.1.3 How efficient is SEMICOUPLED in handling communication path failures? 148  
7.1.4 How does the number of subflows and the characteristics of in-use communication paths affect the throughput of MPTCP? 148  
7.1.5 How can SHIM6 be improved or integrated with another solution to provide a rich set of traffic engineering features for site administrators? 149  
7.2 Future Work .................................. 149  

Bibliography ........................................... 150
# List of Figures

1.1 Address prefix announcement in a multihomed site with PI addressing. Both providers need to advertise the site’s PI address prefix in addition to their own address prefix. ................................................. 5

1.2 Address prefix announcement in a multihomed site with PA addressing. Provider1/Provider2 needs to advertise the PA address prefix that the site has acquired from Provider2/Provider1 in addition to its own address prefix. .......................................................... 5

2.1 Location of the shim layer in the TCP/IP protocol stack. ................. 21

2.2 SHIM6 operation phases. .................................................. 22

3.1 Graphical representation of the SAN model for example 1. .............. 42

3.2 Graphical representation of the SAN model for example 2. .............. 44

4.1 How REAP reacts when a failure affects communication between two SHIM6-enabled hosts. .................................................. 54

4.2 Testbed for experiments with SHIM6 over the Internet between Auckland and Dublin. SADR is not enabled on the university edge router, so only two paths are available. .................................................. 59

4.3 Results from the experiments with DCCP and TCP over the Internet with four address pairs. The x-axis shows the experiment number and the y-axis shows the value for the measured parameter. Times are in milliseconds. .................................................. 61

4.4 Packets traveling between Dublin and Auckland in the TCP experiment. ACKs are sent for every two packets. ............................ 62

4.5 Packets traveling between Dublin and Auckland in the DCCP experiment. ACKs are sent every RTT. .................................... 63

4.6 Trace of packets around a failure event in the DCCP experiment over the Internet. .................................................. 64
LIST OF FIGURES

4.7 Trace of packets around a failure event in the TCP experiment over the Internet. ................................... 64
4.8 Testbed for experiments with SHIM6 in the Lab with two SHIM6-enabled hosts each equipped with four network cards. .......................... 66
4.9 Results from lab experiments for TCP and DCCP with four address pairs. The x-axis shows the experiment number and the y-axis shows the value for the measured parameter. Times are in milliseconds. ........................ 68
4.10 Results from lab experiments for TCP and DCCP with nine address pairs. The x-axis shows the experiment number and the y-axis shows the value for the measured parameter. Times are in milliseconds. ........................ 69
4.11 Graphical representation of the REAP sub-model. .......................... 73
4.12 Graphical representation of the application sub-model. .......................... 75
4.13 Graphical representation of the SHIM6 sub-model. .......................... 77
4.14 Results from [38] (Fig. 3) and a similar experiment with our model. .......................... 80
4.15 Average and total REAP recovery time for the experiments with different number of address pairs for 10000 instances of REAP. REAP shows a poor performance when the number of address pairs exceeds 9. .......................... 82
4.16 Total number of sent probes during recovery for different numbers of address pairs for 10000 instances of REAP. It shows a linear correlation between number of address pairs and number of sent probes. .......................... 83
4.17 Number of probes sent in the first 10 minutes of the recovery phase for different number of address pairs. A large amount of probes are sent at the start of exploration. .......................... 84
4.18 Number of failed connections during recovery phase for different number of address pairs. .......................... 84
4.19 Effect of send timer on recovery time and traffic in case of 16 address pairs. The results show that the effect of send timer on recovery time and traffic is not significant. .......................... 85
4.20 Effect of increasing the number of initial probes on recovery time and traffic in the case of nine address pairs. Increasing the number of initial probes has a significant effect on recovery time. There is a minor change in total number of sent probes but the traffic at the start of the recovery phase is increased. .......................... 86
4.21 Effect of sending initial probes concurrently on recovery time and traffic. This shows improvement in recovery time. .......................... 88
4.22 Effect of clearing loss history, after recovery, on TFRC sending rate. The sender increases its sending rate quickly after recovery. .......................... 90
5.1 Graphical Representation of the application submodel. ............... 99
5.2 Graphical representation of the subflow submodel. .................. 100
5.3 Part of the output function for gate \texttt{rcv\_ack} which calculates the new congestion window size upon receiving an acknowledgement. ........... 101
5.4 Graphical representation of the failure submodel. .................... 102
5.5 Graphical representation of the measurement submodel. ............. 103
5.6 How submodels of the MPTCP SAN model are connected to each other. 103
5.7 Composed model of a MPTCP-enabled host using the Rep/Join feature of Möbius. ......................................................... 104
5.8 Changes in congestion window $\alpha$ and throughput for two subflows for the case of equal and unequal loss rates. ...................... 105
5.9 Changes in congestion window for the 1st and 8th congestion window. In a stable state, the congestion window size for both subflows converges to a similar value since they have the same loss rates. ............. 106
5.10 Changes in total throughput in case of 1, 2, 4 and 8 subflows. Increasing the number of subflows increases total throughput. .................. 107
5.11 A closer look at Fig. 5.10. Using eight subflows shows more stable throughput. ................................................................. 108
5.12 Changes in average throughput for experiments 1, 2, 3 and 4. Adding a faster path shows a significant improvement in throughput. ............. 109
5.13 Changes in total congestion window for experiments 2, 3 and 4. Total congestion window for experiment 4 reaches to a bigger size than other experiments in stable state because the loss rate is smaller than the two other experiments. ................................. 109
5.14 Total number of sent packets for experiments 2, 3 and 4. In the long run, lower RTT, and in the short run, lower loss rate lead to a higher throughput. ......................................................... 110
5.15 Changes in total throughput in case of a permanent path failure at time 45000. Throughput is gradually recovered by moving the load to other available paths. ......................................................... 111
5.16 Changes in $\alpha$ in the case of a permanent path failure at time 45000. $\alpha$ is increased to move the load to other paths. ..................... 112
5.17 Changes in total congestion window in the case of a permanent path failure at time 45000. Other subflows increase their congestion window to accommodate the load. ................................. 112
5.18 Recovery time in the case of 2, 4 and 8 subflows in the case of permanent path failure. Increasing the number of paths, and therefore, subflows, decreases the recovery time. The difference in recovery time between 2 and 4 subflows is noticeable. ................................. 113
5.19 Changes in $\alpha$ for 2, 4 and 8 subflows in case of a permanent failure at time 45000. .................................................. 113
5.20 Changes in congestion window for an experiment with two subflows in case of a temporary failure. Failure occurs at time 45,000 on the first path. The failed path recovers and goes back to normal at time 75000. .......................... 115
5.21 Changes in $\alpha$ for an experiment with two subflows in case of a temporary failure. Failure occurs at time 45,000 on the first path. The failed path recovers and goes back to normal at time 75000. .......................... 115
5.22 Changes in total throughput. ........................................ 116
5.23 Changes in total congestion window, $\alpha$ and throughput for 2, 4 and 8 subflows in case of a temporary failure. Failure occurs at time 45,000 on the first path. The failed path recovers and goes back to normal at time 75000. ........................................ 116
5.24 Changes in total congestion window for two experiments with 1 and 2 subflows when there is a shared channel between subflows. ................. 118
5.25 Loss rates for two experiments with 1 and 2 subflows when there is a shared channel between subflows. ........................................ 118
5.26 How loss rate affects $\alpha$, in an experiment with 2 subflows, when there is a shared channel between subflows. ........................................ 119
5.27 Changes in lose rate in the case of 4 and 8 subflows when there is a shared channel between subflows. Increasing the number of subflows leads to lower loss rates. ........................................ 120
5.28 Changes in throughput in the case of 2 and 4 subflows when there is a shared channel between subflows. The throughput is increased more slowly and more smoothly in case of 4 subflows. ................. 120
5.29 Total number of sent packets in the case of 1, 4 and 8 subflows when there is a shared channel between subflows. Having more subflows will lead to a better performance in the long run .......................... 121
5.30 Average throughput for different number of subflows when there is a shared channel between subflows. Employing more subflows will have a positive impact on the throughput. .......................... 121
5.31 Changes in total throughput for three experiments with fixed and vari-
ant loss rates. SEMICOUPLED could handle the changes in loss rate
properly and total throughput for all experiments converges to the same
value in a stable state. ............................................. 122
5.32 Changes in loss rate for a path with a variant loss rate. Loss rate changes
randomly between 0% and 1% every 5 seconds. ....................... 123

6.1 An example of ingress filtering. P2 and P3 are dropped because they are
forwarded to the wrong service provider. ............................ 127
6.2 The NAROS_SHIM6_REQUEST message format. .................... 136
6.3 The NAROS_SHIM6_RESPONSE message format. ................... 136
6.4 Message sequence when a SHIM6 host in a network which is equipped
with a SHIM6-enabled NAROS server initiates a communication with a
remote SHIM6 host. .................................................. 138
6.5 The NAROS_SHIM6_POLICY_CHANGE message format. ........ 139
6.6 Message sequence when SHIM6-enabled NAROS server broadcasts a pol-
icy change. ............................................................ 140
List of Tables

2.1 Summary of characteristics of the seven active solutions for IPv6 multi-homing. .................................. 34

4.1 Results from experiments over the Internet and simulation with four address pairs. ........................................... 79

4.2 Results from experiments in the Lab and simulation with four address pairs. .................................................. 79

4.3 Results from experiments in the Lab and simulation with nine address pairs. .................................................. 79

4.4 Effect of transport to REAP notification on application recovery time. 3 consecutive expiration of retransmission/nofeedback timer is considered as a failure. .................................................. 91

6.1 Summary of characteristics of SHIM6-enabled NAROS and two other existing TE solutions for SHIM6. ................. 142
This form is to accompany the submission of any PhD that contains research reported in published or unpublished co-authored work. Please include one copy of this form for each co-authored work. Completed forms should be included in all copies of your thesis submitted for examination and library deposit (including digital deposit), following your thesis Acknowledgements.

Please indicate the chapter/section/pages of this thesis that are extracted from a co-authored work and give the title and publication details or details of submission of the co-authored work.

| Title: A Performance study on REAchability Protocol in Large Scale IPv6 networks, published in Second International Conference on Computer and Network Technology (ICCNT), 2010. Section 4.10. | REAP Performance in Large Scale Networks, is mainly derived from this paper. |

| Nature of contribution by PhD candidate | The PhD candidate designed and built the simulation model, ran the experiments, analyzed the results and wrote the text. |
| Extent of contribution by PhD candidate (%) | 90% |

### CO-AUTHORS

<table>
<thead>
<tr>
<th>Name</th>
<th>Nature of Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brian Carpenter</td>
<td>Provided suggestions to improve the work and helped in improving the paper’s presentation and English.</td>
</tr>
</tbody>
</table>

**Certification by Co-Authors**

The undersigned hereby certify that:

- the above statement correctly reflects the nature and extent of the PhD candidate’s contribution to this work, and the nature of the contribution of each of the co-authors; and
- in cases where the PhD candidate was the lead author of the work that the candidate wrote the text.

<table>
<thead>
<tr>
<th>Name</th>
<th>Signature</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brian Carpenter</td>
<td></td>
<td>20/11/2013</td>
</tr>
</tbody>
</table>

Last updated: 25 March 2013
This form is to accompany the submission of any PhD that contains research reported in published or unpublished co-authored work. **Please include one copy of this form for each co-authored work.** Completed forms should be included in all copies of your thesis submitted for examination and library deposit (including digital deposit), following your thesis Acknowledgements.

Please indicate the chapter/section/pages of this thesis that are extracted from a co-authored work and give the title and publication details or details of submission of the co-authored work.

**Title:** Evolving status of IPv6 multihoming. A short version of this paper (A review of IPv6 multihoming solutions) was published in ICN 2011, The Tenth International Conference on Networks. Chapter 2 is derived from this paper.

<table>
<thead>
<tr>
<th>Nature of contribution by PhD candidate</th>
<th>The PhD candidate investigated and wrote the review and analysis for the multihoming solutions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extent of contribution by PhD candidate (%)</td>
<td>75%</td>
</tr>
</tbody>
</table>

**CO-AUTHORS**

<table>
<thead>
<tr>
<th>Name</th>
<th>Nature of Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brian Carpenter</td>
<td>Wrote the introduction, chose solutions and provided suggestions to improve analysis and helped in improving the paper’s English.</td>
</tr>
</tbody>
</table>

**Certification by Co-Authors**

The undersigned hereby certify that:

- the above statement correctly reflects the nature and extent of the PhD candidate’s contribution to this work, and the nature of the contribution of each of the co-authors; and
- in cases where the PhD candidate was the lead author of the work that the candidate wrote the text.

<table>
<thead>
<tr>
<th>Name</th>
<th>Signature</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brian Carpenter</td>
<td>[Signature]</td>
<td>20/11/2013</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Contents

1.1 Problem Statement and Motivation .................. 4
1.2 Thesis Overview ........................................ 9
CHAPTER 1. INTRODUCTION

The rapid growth of the Internet during past years and known limitations in its native protocol, Internet Protocol version 4, raised a concern among experts about the future. IPv4 addresses will run out in the near future. It is a big obstacle to the development of the Internet. One proposed solution, and the most promising one, was to replace IPv4 with a new protocol that was able to resolve IPv4 issues. Internet Protocol version 6 (IPv6) was proposed in 1995 as a replacement [43].

IPv6 increases the size of an IP address from 32 bits to 128 bits. With such a huge address space, IPv6 is able to connect $3.4 \times 10^{38}$ different nodes to the Internet. Three types of addresses are provided by IPv6: unicast, anycast and multicast [64]. A unicast address is an identifier for a single network interface. This type of address is used for a one-to-one communication between two nodes. An anycast address identifies a set of interfaces. Anycast addresses are used for communicating with the nearest node in a group. A multicast address is an identifier for a set of interfaces. Multicast addresses are used for communicating with all nodes belong to the same group. Multicast is more efficient than broadcast. Broadcast packets interrupt all nodes on the local network, even the nodes that are not interested in them; instead multicast packets are delivered only to the nodes which are members of the multicast group.

IPv6 address are represented in the form of a string of hexadecimal numbers separated by ‘:’. Each IPv6 address contains eight pieces of 16-bit hexadecimal numbers. An IPv6 address may contain a group of zeros. To simplify the writing of such addresses, group of zeros can be compressed and represented by a ‘::’. A ‘::’ indicates one or more groups of zeros. To represent IPv6 address prefixes, a method similar to CIDR (Classless Inter-Domain Routing) [58] is used; i.e. an IPv6 address followed by a decimal number. The decimal number specifies the length of the prefix. For example, 2001:0DB8:0:CD30::/64 represents a 64-bit prefix, which assigns 8 leftmost bytes of the address to the prefix. The remaining bytes, 8 rightmost bytes, can be used to represent node addresses; e.g. 2001:0DB8:0:CD30:123:4567:89AB:CDEF/64 is an IPv6 unicast address within that prefix.

Early deployments and experiments have shown that IPv6 is stable and reliable enough to replace IPv4, but two areas of concern remain. One is the problem of interworking between pure IPv6 and pure IPv4 regions of the Internet, a problem not originally anticipated to be serious since general dual-stack deployment was intended. The other is the problem of multihoming, which is the focus of this thesis.

Multihoming has been an issue ever since the concept of the Internet was invented, under the name “catenet” [101]. The original simple drawing of a catenet clearly shows a host computer connected to more than one network. It is, perhaps, surprising that the problem is still open after 35 years, and this suggests that it is, in fact, a genuinely hard problem.
A formal definition of multihoming, which is presented by the IETF, is “a site that has more than one connection to the public Internet with those connections through either the same or different ISPs”\(^1\). Based on this definition, the simplest case of multihoming is a host with more than one IP address. It actually presents some sort of replication so that if one address becomes unreachable for any reason (Denial of service attacks, ISP failure, connection failure, \ldots), the other address provides accessibility for the host. This approach can be extended and used in a site to provide its accessibility to the Internet. A site may have two connection links to the same or different ISPs. In the case that one of these links fails, the site still would be reachable via the other link. In addition to fault tolerance, a multihoming solution for IPv6 should also support capabilities like redundancy, load sharing, performance, simplicity and security [5]. In the current Internet, this function is in high demand. A significant number of stub networks are now multihomed and this number is growing [37].

There are similarities between multihoming and mobility. A multihomed host may have multiple IP addresses, that enable it to switch to another working path in the case of failure of the current path. Alternatively, the host may have a single address where the routing system handles multiple paths to it. The process is similar to a handover process in mobile communications when a mobile node moves from one area to another one, but the aim is different. In multihoming, this switching is mainly performed to increase reliability while in mobility it is required to preserve the communication.

To connect a site to the Internet, at least one globally routable IP address is needed. A globally routable IP address is a known IP address for core routers so that packet delivery between this address and other globally routable addresses is possible. A site can acquire one or more globally routable IP address(es) directly from its Regional Internet Registry (RIR)\(^2\) or from its ISP. The former is called Provider Independent (PI) addressing and the later is called Provider Aggregatable (PA).

Most sites would like to have PI addresses because it gives them freedom to change their providers without renumbering. In other words, they can switch to a new ISP without needing to reconfigure their hosts or internal routers. Managing PI addresses imposes a significant overhead on Internet core routers [81]. They have to keep more addresses in their routing table, which leads to bigger routing tables and greater costs for routing update and lookup.

An alternative for a site is to acquire its globally routable addresses from its ISP. In this case, the ISP assigns a part of its own address block to the site. This approach does not have the negative impacts that PI addresses have on the Internet routing system; although, in this case, renumbering may be a difficult but unavoidable task [34].

\(^1\)http://datatracker.ietf.org/wg/multi6/charter/

\(^2\)Each region in the world has an RIR which manages allocation of IP addresses in that region.
the sake of routing scalability, sites are encouraged to use PA addresses. Unique Local Address (ULA) [66] is a solution for IPv6 enabled sites which need PI-like addresses for local use. A ULA is a globally unique address that is not globally routable but can be treated like a global scoped address by applications.

Internet routing is, of course, based on binary addresses. In an idealised world, the hierarchy of these binary addresses would map perfectly onto a tree structure of routers, which, in turn, maps onto the topology of links leading to end-users. In such a network, optimised routing would be trivial. In fact, it would be a perfect example of PA addressing, where all customers of a single ISP would be represented by exactly one aggregated entry in the global routing system. However, if an end-user is connected twice to such a network, there are only two choices: either the user has two different addresses (one for each connection) or the same address is used for each connection, in which case the perfect mapping between addresses and topology is broken. In the real network, which deviates dramatically from the ideal, we face a worse version of the problem. The topology is a mesh, not a tree, and many end-users have PI address space assigned directly to the user. The routing system has to compute optimised paths across the mesh. When end-users are connected twice, their address blocks must be handled twice by the routing system. Far from having one routing entry per ISP, we would have at least one for every multihomed site.

In the next section, we describe the issue related to the current multihoming solution and explain our motivations for doing the research presented in this thesis.

1.1 Problem Statement and Motivation

In the current Internet, multihoming is achieved by using BGP (Border Gateway Protocol). A multihomed site acquires its PI or PA address block and then announces this address block through all its service providers to the Internet global routers. Using PI addresses is more popular in the current Internet. Fig. 1.1 shows an example of a multihomed site with PI addressing. In this case, the multihomed site needs to announce its address block prefix (130.216.187.0/24) to all the service providers. The service providers also need to announce the prefix to the upper layer service providers.

In case of PA addressing, the site needs to announce the address prefix received from each service provider through all other service providers. Fig. 1.2 shows an example of a multihomed site with PA addressing. The site receives two address blocks from its service providers: 20.0.0.0/24 from the service provider 1 and 30.0.0.0/24 from the service provider 2. To make the site accessible through both providers, it should announce 20.0.0.0/24 through service provider 2 and 30.0.0.0/24 through service provider 1.

Using the current solution, each multihomed site inserts at least one new entry into
CHAPTER 1. INTRODUCTION

Figure 1.1: Address prefix announcement in a multihomed site with PI addressing. Both providers need to advertise the site’s PI address prefix in addition to their own address prefix.

Figure 1.2: Address prefix announcement in a multihomed site with PA addressing. Provider1/Provider2 needs to advertise the PA address prefix that the site has acquired from Provider2/Provider1 in addition to its own address prefix.
the global routing tables. This increases the size of routing tables in the core routers. It means that the core routers need a bigger main memory to hold the routing table and a faster processor to perform operations like “lookup” in a reasonable time. As the demand for multihoming is growing, this solution may lead to routing table explosion in Default Free Zone (DFZ) routers [84]. In other words, this solution would not scale well if a few million multihomed sites could obtain PI prefixes. Despite all the worries about exponential growth for BGP tables, the experimental fact is that the growth is approximately linear\(^3\). It is around 8%-10% per annum (down from 33% in 2010) for IPv4 and 20-40% per annum for IPv6 (down from 90% in 2011). No one could really explain why the growth is apparently so controlled but it is certain that if a few million sites request PI prefixes and request them to be announced in BGP, the growth will explode. That cannot happen in IPv4 since there are no prefixes available, but it could happen in IPv6 where there is an effectively infinite supply of prefixes. IPv6 addresses are 128 bits wide which expands to \(3.4 \times 10^{38}\) different addresses. With such a huge address space, IPv6 is capable of providing unique addresses to every node attached to the Internet. Since we do not understand why growth is currently constrained, we cannot safely predict that it will continue to be constrained. As the demand for multihoming grows, scalability becomes a major concern for multihoming in the future Internet.

Requirements for multihoming solutions were defined in [5]. One in particular should be noted: a requirement that transport layer sessions, such as TCP connections and UDP streams, should be able to survive a re-homing event. This increases the difficulty in solutions where multiple addresses are used for each host, since TCP and UDP assume fixed addresses.

Similar goals and considerations for multihoming are applied to both IPv4 and IPv6 [6] [5], but there are some features and attributes in IPv6 which affect the design architecture of multihoming:

1. IPv6 provides a huge address space, so the scalability issue seems to be more serious.

2. Advertising multiple prefixes on the same link is a possibility in IPv6, so source and destination address selection is a complex issue for hosts.

3. IPv6 tries to avoid Network Address Translation (NAT). Network Address Translation (NAT) [46] is a technique that helps sites to hide their non-routable, internal, or PI addresses behind a set of PA addresses. A NAT box at the border of the site rewrites incoming and outgoing packet headers and modifies source

\(^3\)http://www.potaroo.net/presentations/2014-05-12-bgp2013.pdf
and destination addresses to hide host addresses in outgoing packets. NAT is a popular technique in the current Internet, but, because of its major drawbacks, designers are not keen to use it in the future IPv6 Internet. Therefore NAT-based solutions, which are popular in IPv4, may not be applicable to IPv6.

4. IPv6 is not universally deployed yet. Although the growth is rapid\(^4\), it is still possible to make some changes at a reasonable cost.

IPv6 offers a larger address space compared to IPv4. The future Internet is supposed to use IPv6 and the routing table size seems to be a serious issue. Thus, a new solution is required for multihoming in IPv6. A wide range of solutions has been proposed during past years. These solutions can be put into three categories based on the component in the system that provides multihoming functionality [78][111]:

1. **Routing approaches**: Routers are responsible for providing multihoming service. These solutions try not to affect end hosts. To be deployed in the Internet, the routing system should change.

2. **Middle-Box approaches**: An intermediary box between the site and the Internet offers multihoming service. Hosts are hidden behind the middle box and let the middle box do required operations on behalf of them.

3. **Host-centric approaches**: Hosts are multi-addressed and implement multihoming functionality. The Internet routing system is unaware of the existence of this capability in hosts. These solutions might need some support from the site’s edge routers to make sure the packets are forwarded to an appropriate service provider. Mechanisms like SADR (Source Address Dependent Routing) [67] may be used for this purpose.

Although a wide range of solutions has been proposed, there is no broadly held consensus about the “correct” approach to multihoming for IPv6. The first part of our research focuses on analyzing a set of proposed solutions, which are still under investigation by researchers and Internet experts. The aim of our analysis is to find an answer for the following question:

1. **Which solution can be the winner from a deployability viewpoint?**

We try to find out which of those solutions has a bigger chance to be the final solution for IPv6 multihoming. Two important factors are considered in our analysis: 1) is the solution able to provide the main functionalities of the current solution, which is based on BGP? 2) How difficult would be the deployment?

\(^4\)https://www.vyncke.org/ipv6status
The analysis showed that all reviewed solutions have some weak points and still need improvement, but from the deployability viewpoint host-centric solutions look more attractive. Making changes in a huge and complicated system, like the Internet, is difficult; so, host-centric solutions which only modify end hosts and do not need any change in the Internet routing system are easier to deploy. SHIM6 and MPTCP are two proposed solutions from this category (a review of active solutions for IPv6 multihoming is presented in chapter 2). They need minimum change, even among host-centric solutions. They only need the host’s protocol stack to be updated. Both are backward compatible and can be deployed incrementally. Detecting path failures and being able to recover from them without breaking active communications is one of the main requirements for every multihoming solution. MPTCP and SHIM6 are both able to provide this functionality. SHIM6 is equipped with a specific failure detection and recovery protocol (REAP\(^5\) [10]) and MPTCP employs a central congestion control mechanism (SEMICOUPLED Congestion Control Algorithm [103]) for this purpose. These protocols are not yet widely deployed on the Internet, so the efficiency of the these protocols, especially in the event of failure, is still an open research area. The second part of the work presented in this thesis focuses on this research area by conducting a series of real-world and simulation experiments on SHIM6 and MPTCP. The aim of the experiments is to address the following questions:

2. How efficient is REAP in handling communication path failures and how can it be improved?

3. How efficient is SEMICOUPLED in handling communication path failures?

The flexibility of our model of MPTCP allowed us to increase the scope of our study and investigate the behaviour of SEMICOUPLED in some other situations as well. By conducting more experiments, we try to throw some light on the following question:

4. How does the number of subflows and the characteristics of in-use communication paths affect the throughput of MPTCP?

SHIM6 was the focus of IETF activity for several years but it could not obtain the support of the ISP community because it removes their control of the choice of paths for customer traffic (known as “traffic engineering”). Since the solution is implemented in end-hosts, having a site-aware traffic engineering mechanism is challenging. In a SHIM6-enabled site, hosts decide what path, and, therefore, which service provider, to use to communicate to a remote host. The path might change later without any notice to the application or routing system. Although this behaviour is one of the favourite features of SHIM6 in terms of failure detection and recovery, it makes network administrators

\(^5\)REAchability Protocol
unhappy since they lose control over the incoming/outgoing traffic. The last part of the work presented in this thesis focuses on this issue. We try to answer the following question:

5. How can SHIM6 be improved or integrated with another solution to provide a rich set of traffic engineering features for site administrators?

1.2 Thesis Overview

This thesis is organized as follows:

In chapter 2, we present a history of multihoming and proposed solutions for IPv6. We then briefly review seven solutions which have been proposed in the area. We then analyze the proposed solutions from a deployability viewpoint. We try to answer research question 1 in this chapter.

Chapter 3 briefly describes Stochastic Activity Networks (SAN). SAN is a general-purpose modeling tool used for the simulation studies in chapters 4 and 5.

Chapter 4 presents the results of our experiments with SHIM6 in the lab and over the Internet. Using a SAN-based simulation model, we investigate the behaviour of REAP in the case of path failure in a large-scale SHIM6-enabled site with 10,000 hosts. We also propose and evaluate a number of improvements to REAP which can reduce recovery time and traffic, as well as a signaling mechanism which improves the efficiency of TFRC in SHIM6-enabled networks. This chapter focuses on research question 2.

Chapter 5 presents the results of our experiments with a SAN-based simulation model of MPTCP. The model, which fully implements the SEMICOUPLED congestion control algorithm, is described and validated. Then, we investigate the behaviour of SEMICOUPLED in case of 1) permanent and temporary path failure, 2) increasing the number of available paths 3) existence of a shared bottleneck between paths and 4) paths with variable loss rates. Research questions 3 and 4 are the focus of this chapter.

In Chapter 6, we focus on traffic engineering, which is one of the major weak points of SHIM6. To address research question 5, a mechanism that provides a dynamic site-aware traffic engineering solution for SHIM6-enabled networks is proposed in this chapter. The solution is also compared with other existing TE solutions for SHIM6.

We summarize the results and contributions of our work in chapter 7 and suggest areas for future work.

---

6 The model can be downloaded from www.cs.auckland.ac.nz/~hnad002

7 TCP Friendly Rate Control
Chapter 2

Multihoming Solutions for IPv6

Contents

2.1 Introduction ..................................... 12
2.2 Proposed Solutions in the Area ...................... 15
  2.2.1 LISP ..................................... 16
  2.2.2 ILNP ..................................... 17
  2.2.3 NPTv6 ..................................... 18
  2.2.4 HIP ...................................... 19
  2.2.5 Name-based Sockets ........................... 19
  2.2.6 SHIM6 ..................................... 20
  2.2.7 MPTCP ..................................... 23
2.3 Analysis of the proposed solutions ................... 24
  2.3.1 Analysis of LISP ............................ 25
  2.3.2 Analysis of ILNP ............................ 27
  2.3.3 Analysis of NPTv6 ........................... 28
  2.3.4 Analysis of HIP ............................ 29
  2.3.5 Analysis of Name-based Sockets ................ 30
  2.3.6 Analysis of SHIM6 .......................... 31
  2.3.7 Analysis of MPTCP .......................... 32
  2.3.8 Summary of the Analysis ...................... 33
2.4 Conclusion ........................................ 34
This chapter presents a history of multihoming and proposed solutions for IPv6\(^1\), then briefly reviews seven solutions which are currently under investigation in the area. We also analyze the solutions from a deployability viewpoint in this chapter.

### 2.1 Introduction

There has never been a broadly held consensus about the “correct” approach to multihoming for IPv6. The problem was mentioned for IPv4 as early as 1981 [100] and was identified for IPv6 at an early stage (e.g., [28] in 1998). However, the original assumption was that one particular property of IPv6, its ability to support multiple simultaneous prefixes for a given network and multiple simultaneous addresses per host, would be used to solve the problem. In this view, a customer site with N different ISPs would obtain a different PA prefix from each ISP, and would use all N of them simultaneously. Thus, each ISP could arrange for all its customers’ prefixes to aggregate into a single short prefix announced to BGP4.

This solution can be referred to as multi-PA multihoming, and minimises the effect of multihoming on routing table size, but has four notable disadvantages:

1. It does not provide automatic failover of existing transport sessions when one of the prefixes fails.
2. It requires partial renumbering of a site when an ISP is added or dropped [23].
3. It is unfamiliar to IT managers accustomed to IPv4 practices.
4. Source address selection, next hop selection and DNS server selection can be problematic for multi-addressed hosts [120].

It became clear by about 1999 that these disadvantages meant that other solutions had to be found. However, the goals for IPv6 multihoming were not defined until 2003 [5] and the first architectural analysis appeared in 2005 [68]. This document identified five major classes of solution:

1. Based on standard IP routing
2. Based on Mobile IP
3. New protocol element providing a session identifier
4. Modified protocol element to support dynamic address changes

\(^1\)This chapter is an extended and updated version of [90]
5. Modified relationship between hosts and site exit routers

At that time, draft proposals fitting all these categories were produced. An alternative view is to classify the proposals according to whether they require host modifications, routing system modifications, or both. A third aspect is what layer of the network stack they concern: network layer, transport layer, or session layer. One particular difficulty in the analysis was that of finding a common approach shared by experts in routing operations, network layer protocol designers, and transport or applications layer specialists.

The first class of solution, simply using the standard routing system to provide multihoming, is documented in [6] for the case of IPv4. It requires each multihomed site to have its own globally unique PI address prefix, and there has always been concern that if extended to IPv6, this would exceed the reasonable capacity of the BGP4 routing system within a few years of further growth. It is easy to envisage scenarios where at least ten million unique PI prefixes would be needed, with some estimates being much higher.

Thus, all the proposals specific to IPv6 attempt to escape from this problem. It is generally felt that at some level of abstraction, all such solutions involve separating the two functions of an IP address - its function as a locator (how to find a destination) from its function as an identifier (how to identify one end-point of a communication).

In Mobile IP [71] the role of the identifier is taken by the Home Address and that of the locator is taken by the Care-of Address. However, this has never been felt to be a plausible solution for the multihoming of large sites, simply due to the overhead cost of Mobile IP. Thus, the early proposals focused on other methods of separating the locator and identifier. A suggestion presented in 1998 by Deering [42] but originated as early as 1996 [63] was simply to use the IP address space in two different ways: some addresses would be used as end-system identifiers and others as wide-area locators for the routing system. At the border between a site and the open Internet, a packet containing end-system identifiers as its addresses would be encapsulated in an outer packet using wide-area locators as its addresses. Thus, the locators could be chosen to ensure good route aggregation, and some kind of mapping system is needed between destination identifiers and their corresponding locators. Such schemes are known generically as map-and-encap.

Encapsulation schemes have the disadvantage of expanding the packet size, which, in turn, may lead to exceeding the maximum transmission unit (MTU) size of a link and hence to the inefficiency of fragmentation. Alternative proposals have been made to use some variant of header translation to support multihoming. In the case of IPv6, these proposals are often referred to as “8+8” after the initial proposal by O’Dell [98].
In this class of solution, the 128 bits of the IPv6 address are cut into two parts - most simply, 8 fixed bytes that act as a unique end-system identifier, and 8 mutable bytes that act as a locator for a site and subnet, and can be rewritten by the routing system to allow aggregation and dynamic choice of ISP. The locator bytes are ignored by the source and destination host software.

It should be noted that in both map-and-encap, and effectively in 8+8, the packet finally delivered is identical to the packet sent. Proposals were made, and are in current use for IPv4, in which Network Address Translation (NAT) is employed to translate the PI source address of a packet into a PA address for wide-area routing. Unfortunately this means that the packet finally delivered is different from that sent, producing many of the undesirable side-effects of NAT even if ambiguous private addresses are not used.

Another approach which has been repeatedly advocated, e.g. [22], is a scheme in which user sites are provided with a geographically based address prefix. In such a case wide-area routing can use aggregation based on those prefixes rather than on ISP prefixes, with local Internet Exchange Points handling all prefixes for a given geographical region. Such solutions, however, do not match the current economic structure of the Internet, where aggregation and routing is based almost entirely on highly competitive ISPs with overlapping geographical coverage.

The preceding choices relegate the solution entirely to the IP and routing layers. Other solutions were discussed that consider higher layers. One was to use the intrinsic address agility of SCTP (Stream Control Transmission Protocol [115]) to support multihoming. The disadvantage of this was that it did not resolve the problem for existing TCP and UDP applications. Another was to support multihoming by introducing an additional level of end-system identifier above the IP address, namely HIP (Host Identity Protocol [88]). This remains a research area.

In the period 2001-2003 at least 35 drafts related to IPv6 multihoming were produced in the IETF, covering the entire range of solutions mentioned above. After considerable discussion, one particular approach known as SHIM6 was selected for standards development in the IETF. Although SHIM6 was the focus of IETF activity for several years, it does not yet appear to have the support of the ISP community, because it removes their control of the choice of paths for customer traffic (known as “traffic engineering”). Thus, other solutions are also under consideration. The Regional Internet Registries have relaxed their criteria for allocating PI prefixes, so that some early adopters of IPv6 are able to multihome in the old-fashioned IPv4 style [6], with the resulting threat of routing table growth. This concern eventually led to an Internet Architecture Board workshop and report [84] followed by a new burst of research and development activity. Several technical proposals have been developed, notably LISP, ILNP, NPTv6, MPTCP, continued work on HIP, and name-based transport. These efforts are briefly
summarised in the next section.

This list of current proposals is incomplete. A wider range of proposals, aimed mainly at BGP4 scaling, has been discussed by the Routing Research Group [81] but is not further discussed here. Recently, some mechanisms and solutions have been proposed that implicitly or explicitly are related to multihoming. Happy Eyeballs [124][35], implicit multihoming approaches proposed in the IETF mif WG\(^2\) in the context of provisioning domains [31], IPv6 multihoming without NAT [120] and solutions proposed in the IETF homenet WG [62][119] are samples of such solutions and mechanisms. We do not consider these solutions here since they are mechanisms, not complete solutions, which are helpful in providing multihoming (SADR and Happy Eyeballs) or they are trying to address node multihoming (proposed solutions in the context of provisioning domains) which are out of scope for this thesis, since the focus of our work is site multihoming. IPv6 multihoming without NAT does not propose a single overall solution. It decomposes the multihoming problem into three sub-problems and then proposes solutions for each sub-problem.

The rest of this chapter focuses on reviewing and analyzing the seven proposed solutions for IPv6 multihoming that are currently under investigation.

2.2 Proposed Solutions in the Area

Although a wide variety of solutions for IPv6 multihoming have been proposed during past years, there is no agreement in the research and technical community for choosing one of them as the best solution. Scalability has been the main concern [84] and avoiding huge routing tables has been one of the most important goals in this area. Although current studies do not show exponential growth in the routing table size in recent years,\(^3\) it could happen for IPv6 where there is an effectively infinite supply of prefixes. We do not understand why growth is currently constrained, so we cannot safely predict that it will continue to be constrained. As the demand for multihoming grows, scalability is still a concern for IPv6 multihoming. The identifier-locator separation technique is considered as fundamental for this problem and has been employed by a majority of the solutions. In this section, we present a brief review of the seven proposed solutions in this field.

\(^2\)Multiple Interfaces Working Group

\(^3\)http://www.potaroo.net/presentations/2014-05-12-bgp2013.pdf
2.2.1 LISP

LISP (Locator/ID Separation Protocol) [47] is a fully developed map-and-encap solution. It defines an EID (End-point IDentifier), with the same format as an IP address, used to uniquely identify a host, and an RLOC (Routing LOCator) as an IP address used to identify a connection point to the wide-area Internet. Many EIDs are “behind” a given RLOC, and RLOCs are assumed to be PA addresses so that they can be aggregated in the BGP4 system. As in any map-and-encap solution, a packet with EIDs as its source and destination addresses is encapsulated in an outer packet with RLOCs as its addresses. The encapsulating/decapsulating router serves as a border router between one or more sites and the Internet. All hosts behind it share the same RLOC, but there is no need for any relationship between their EIDs. (In practice, the EIDs will resemble IP addresses derived from each site’s PI prefix.)

This is relatively simple, but two major issues arise. Firstly, LISP requires a reliable and high performance mapping system to find the correct RLOC for a given EID, and secondly, LISP must coexist and interoperate with the existing unmapped Internet during a transition period of unknown duration. The mapping problem could be solved in various ways, and is mainly an engineering problem. The interoperation problem is quite tricky [70][80]. Although EIDs resemble PI addresses, they will not be accessible by normal BGP4 routing, since the very existence of LISP is intended to keep down the size of the BGP4 routing table. Thus, the existing Internet will contain no routes between existing IP prefixes and EID prefixes. One proposed solution is for the LISP system to include proxy tunnel routers, which will announce highly aggregated prefixes for very large ranges of EIDs into the existing Internet; these routers will then become concentrators for traffic between LISP and non-LISP hosts, with no pretense of finding the shortest path. An alternative is to use a special NAT which, simplifying somewhat, translates EIDs into RLOCs and vice versa. This resembles the NPTv6 solution discussed below.

LISP has two major components [47][57][70][80][48][56][69][49]: data plane which performs the map-n-encap operation, and control plane, which is the EID-to-RLOC mapping the system. The map-n-encap process is performed by LISP routers: ETR (Egress Tunnel Router) and ITR (Ingress Tunnel Router). ETRs perform decapsulation and ITRs are responsible for encapsulation. A fast and reliable control plane should provide assistance for ITRs so that they can encapsulate outgoing packets in an outer header which contains RLOCs. LISP+ALT [57] is the mapping database, proposed by the LISP working group, to be used as the LISP control plane.

LISP defines three types of packet for supporting its EID-to-RLOC mapping system:

1. Map-Request: which is used by ITR to query the mapping system for a particular
the EID-to-RLOC mapping. The response would be a Map-Reply message.

2. **Map-Reply**: ETR emits Map-Reply in response to Data-Probe or Map-Request.

3. **Data-Probe**: which is a LISP-encapsulated data packet with the same inner and outer destination addresses. It is used in some mapping systems to trigger a Map-Reply from a decapsulating ETR.

The mapping system should support *weight* and *priority* which are useful features for traffic engineering and load sharing without using BGP. Priority tells the ITR which ETRs to use and in which order, and weight tells the ITR how to split load across ETRs of a given priority. This allows users to have redundancy and load sharing without the complexity of running BGP.

There are two performance concerns related to LISP: 1) Encapsulation overheads which includes processing time and also exceeding MTU by adding the LISP header. 2) EID-to-RLOC lookup latency and packet loss.

### 2.2.2 ILNP

ILNP (Identifier Locator Network Protocol) [11][15][17][12][13][30][14][16] is a direct descendant of 8+8 [98] and was developed in the Internet Research Task Force (IRTF). It largely matches the short description of 8+8. The main difference is that the firm requirement for the 64-bit end-system identifier to be globally unique has been relaxed to “unique within the context of a given [64-bit] Locator”. As in 8+8, the locator is ignored in transport-layer checksum calculations (i.e., it is treated as zero). Host IPv6 stacks, and optionally site border routers, are aware of ILNP in order to select a working locator after a failure occurs. However, ILNP uses ICMP to signal locator updates, rather than using an IPv6 extension header.

In ILNP, the application layer should only use FQDNs. Mapping an FQDN to an identifier and locator is performed by DNS. ILNP proposes adding two new resource records to DNS: *I* and *L* records [17]. *I* records are used for mapping an FQDN to an identifier and *L* records are used for mapping an FQDN to a locator(s). Two more resource records are also added for reverse lookup: *PTRI* and *PTRL*. *PTRI* is used for mapping an identifier to FQDN and *PTRL* is used for mapping a locator to an authoritative DNS server. Secure DNS Update protocol is used for updating locators in case of mobility. The current identifier-locator mapping is cached in each end system and updated using a new ICMP *Locator Update* message [13]. The application layer uses FQDNs and transport layer uses identifiers; locators are only used in the network layer.
Multihoming is better achieved by ILNP having one single identifier and one locator per provider and by using the described mechanisms for changing locators and announcing the change through ICMP and DNS. NAT also can benefit from ILNP, because IP addresses are not used above the network layer in ILNP.

2.2.3 NPTv6

NPTv6 [122] is a proposed stateless version of network address translation. A site would run stable IPv6 addressing internally (it could be PI, PA or ULA). The NAT connecting the site to a particular ISP has an equivalent amount of PA addresses from that ISP, and simply translates the source address of each outgoing packet, and the destination address of each incoming packet, 1:1 between the internal stable address and its equivalent external ISP-dependent address. Port translation is not required, since there is an external address for every internal address. In this model, the site will never need to change its addressing plan, however often it adds or removes ISPs. From the viewpoint of wide-area routing, this solution is identical to the original multi-PA multihoming model. From the site’s viewpoint, it is equivalent to PI multihoming, except for some of the usual disadvantages of NAT.

The mapping algorithm works in the following manner:

1. A one’s complement checksum of both internal and external /48 bit prefixes are calculated.

2. The internal checksum is subtracted from the external checksum using one’s complement arithmetic.

3. The subnet mask in the original packet is added to the result of step 2. The result would be the subnet area of the mapped address.

The mapping algorithm is two-way which means that there is no need for an NPTv6 box to maintain any per-node or per-connection state. It also allows internal nodes to participate in peer-to-peer communications. Although that is an advantage, it raises some security concerns. To address the security issues, using a firewall along with an NPTv6 device is recommended to control incoming traffic. The algorithm is checksum neutral which means that the changes in the header are performed in a way that keep the checksum in the transport layer unchanged. As a result, there is no need to modify transport layer headers. However, modifying IP headers in transit makes NPTv6 incompatible with security mechanisms, like IPsec.

To use NPTv6, both internal and external prefixes need to be /48 or shorter so as to have at least 16 bits available for the subnet. For longer prefixes, a slightly more
complex algorithm should be used and a subset of addresses might be untranslatable. Subnet masks with all ones (0xFFFF) cannot be used behind an NPTv6 box. Such masks would cause a problem with ones-complement arithmetic.

### 2.2.4 HIP

HIP [87][88][95][72][93][94][116][77][76][75][75] is a host-centric solution for secure end-to-end mobility and multihoming that uses the identifier/locator split approach. It is still regarded as experimental, but it does, like SHIM6, allow upper layers to see a constant identifier (the HIP Identity Token) regardless of the locator-addresses in use. A common socket API extension is proposed in [74] for SHIM6 and HIP since they look similar to the upper layer protocols.

In HIP, IP addresses are used as locators, but a host identifier is the public key component of a private-public key pair. Host identity is a long-term identity, so it can be used for looking up locators. Host identity is created by the host itself and can be stored in DNS or a PKI (Public Key Infrastructure) to make it accessible for other hosts. Each host has one host identity, but can have more than one host identifier. HIP is a general solution that is able to work for any transport protocol.

To use HIP, an HIP association should be established first through an HIP Base EXchange (BEX) process. When this process is successfully completed, both hosts will have the other side’s private key.

Locator (IP) changes are signalled to a peer using a three-way UPDATE signaling mechanism. HIP multihoming [94] uses the same mechanisms as HIP mobility for updating its set of locators.

Opportunistic HIP [88] can be used in a situation where using or accessing DNS or other mechanisms, like Distributed Hash Tables (DHT), is not possible. It works in the same way as SSH, which adds the public key of the server to the known host file after the first connection.

### 2.2.5 Name-based Sockets

Name-based transport [121] is a very attractive approach, which could hide multihoming, mobility and renumbering from applications. The idea is to make it unnecessary for application software to have any knowledge of IP addresses or any other form of network layer identifiers. That should make it possible to choose any convenient combination of the above-mentioned solutions without impact on applications.

Name-based Sockets (NBS) enable application developers to use names instead of network-layer addresses (locators) for establishing communications. Managing locators is pushed to the operating system. Mapping names to locators is still performed using
DNS. No change in DNS is required for using NBS. At the beginning of the communication, names are exchanged, so the responding host does not need to use DNS reverse lookup to do the reverse mapping (IP to name mapping). The exchange process is asynchronous and backward compatible, so NBS-enabled hosts are still able to communicate with legacy hosts. The NBS features are not available until the exchange process has been completed successfully.

NBS is not transparent to applications. It provides a socket interface that is implemented around a new address family (AF\_NAME). If an application needs to use NBS features it should explicitly use the AF\_NAME address family.

2.2.6 SHIM6

This is a “shim” inserted into the network layer of each host. It assumes that a site has a PA prefix from each ISP, as in the original IPv6 model. When a host has multiple addresses, one of them is chosen as an identifier for use by the transport layer. SHIM6 then dynamically changes the addresses used in the actual packets according to reachability, but always presents packets containing the original identifier-address to the transport layer.

SHIM6 can therefore be classified as a network layer solution that simulates locator-identifier separation, but otherwise is strictly compatible with standard IPv6 and its routing mechanisms. To succeed, it requires all hosts to implement it and it requires sites to use multiple address prefixes (most likely PA prefixes).

The main goals of SHIM6 design are:

- Session survivability
- Minimal impact on ULP (upper layer protocol)
- Address some security threats
- Defer the setup of shim state to minimize overhead on short-lived sessions
- Spreading (not balancing) the load on the different locators of a host

SHIM6 uses the identifier/locator scheme, but does not define a new name space. IPv6 addresses are used as identifiers and also as locators. Initial connection is established using one of the locators, which would play the role of ULID (Upper Layer IDentifier) for the whole life of the communication. This ULID is used by the transport layer and locators are managed by the shim layer inside the network layer. ULID selection is performed using the mechanism for default address selection proposed in [117]. In case of failure the locator changes, but the ULID remains the same.
The shim layer is placed on top of the IP routing sub-layer and under the IP endpoint sub-layer (Fig. 2.1). It does a mapping between ULID and locator(s) bi-directionally; ULID to locator mapping at the sender side and locator to ULID mapping at the receiver side. This process is transparent to the ULP. A state called ULID-pair context is maintained for each ULID pair by SHIM6 to help sender and receiver to perform mapping consistently. That state is independent from the transport layer. We refer to this as SHIM6 context in the rest of the thesis. \(<\text{peer ULID, local ULID, context tag}>\) uniquely identifies a context. A context tag is a 47-bit number, the largest number that can be fitted in an 8-octet extension header, which should be unique among all contexts. One bit is reserved to differentiate SHIM6 messages from data packets with the shim header. They both use the same protocol number.

Figure 2.1: Location of the shim layer in the TCP/IP protocol stack.

SHIM6 operates in the following phases (Fig. 2.2):

- Initial contact between ULPs. No action by the shim at this stage.

- After a while (some packet exchanges or a specific time), SHIM6 initiates 4-way context establishment. As a result both ends obtain a list of locators for each other. If context establishment fails, it means that the other end does not support SHIM6.

- Communication continues without any change for the ULP packets. SHIM6 does not do anything since the ULID pair is the same as the locator pair; there maybe some exchanges between shim layers regarding unreachability detection.

- An outage occurs. It might be signalled by the ULP or by an unreachability detection mechanism [10]. One or both ends try to find a new working address pair.

- After finding the alternative pair, SHIM6 rewrites addresses in the transmitted packets and tags them with a payload extension header, which contains the re-
receiver’s context tag. A context tag helps the receiver to find the associated context and apply reverse changes so that, from the ULP viewpoint, everything is the same as before.

- The new locator pair is monitored by the unreachable mechanism for future failures.

The protocol also enables peers to change locator pairs in the case of events that affect the communication, like interface down or change in locator preferences. SHIM6 contexts are garbage collected when they are not in use anymore. The mechanism for determining unused contexts is implementation dependent (existence of ULP state, expiry timer, etc.).

$I1$, $R1$, $I2$ and $R2$ are four messages used for the context establishment process. One end (initiator) starts the process by sending an $I1$ message to the other end (responder). Upon receiving the $I1$ message, the other end responds with $R1$. Then the initiator sends an $I2$ message and the responder completes the process by sending an $R2$ message.

![SHIM6 operation phases](image-url)
This 4-way exchange allows the responder to protect itself against flooding attacks by avoiding creating context on the first packet. If both ends try to establish context at the same time and \( I_1 \) messages are issued by both, it can end up with an immediate issue of \( R_2 \). \( R_{1bis} \) and \( I_{2bis} \) are used in the case of lost context. If an end receives a message that contains a SHIM6 payload extension header but it cannot find a matching context, it will respond with \( R_{1bis} \). It initiates a re-creation of the context using \( I_{2bis} \) and \( R_2 \).

Update Request and Update acknowledgement messages are used to change the list of locators exchanged during context establishment. The Locator preference option can be used in the Update Request message. The locator preference option enables hosts to express priority and weight values for each locator in the locator list.

To protect the binding between an ULID and the associated locator set, SHIM6 uses CGA [18] or HBA [20] addresses. HBA and CGA are two mechanisms for binding a public signature key to an IPv6 address. Using these mechanisms, SHIM6 secures binding between the multiple addresses, with different prefixes available to a multi-addressed SHIM6-enabled host.

### 2.2.7 MPTCP

Multipath TCP (MPTCP) [55][54] is an extension to traditional TCP which enables it to use multiple paths between multihomed/multi-addressed communication peers simultaneously. The aim of MPTCP is to improve resource utilisation and failure tolerance. MPTCP provides a set of features on top of TCP which is supposed to be backward compatible to be able to work with middle boxes (e.g. NAT, firewall, proxy) and legacy applications and systems without involvement of the users.

MPTCP differs from SCTP [115] in two important ways. Firstly, it aims to preserve the existing TCP socket interface, so that application software currently using TCP will not require any changes. Secondly, rather than using one address pair at a time, it will use all working address pairs simultaneously, using TCP-like mechanisms to automatically adjust the load on each path. It does require a multi-PA addressing model for the user site, and it does nothing for UDP traffic.

An MPTCP connection is started like a regular TCP connection. Then, if there are extra paths available, additional TCP connections (subflows) will be created. MPTCP operates in such a way that all these connections behave like a single TCP connection. Each connection is assigned a token which uniquely identifies that connection. Connection-level sequence numbers are used to reassemble the data which have been sent through different subflows. Connection-level FIN is used for terminating all subflows.
Some new options have been defined to provide MPTCP specific features:

- *Multipath capable* option in SYN/ACK packets is used to announce a host’s support for MPTCP.

- *Join* option is used to associate a new subflow to an existing connection.

- *Add address* option is used by a host to notify the peer of its other addresses. Each address has an ID and hosts must keep the mapping between addresses and IDs.

- *Remove address* option can be used by a host to notify the peer about the unavailability of one of the already announced addresses. Affected subflows must be closed.

- *Data sequence mapping* option, which defines the mapping from data sequence number (DSN) to subflow sequence number to ensure in-order delivery to the application at receiver side. A DSN is a 64-bit integer which numbers all data sent over the MPTCP connection. Each subflow has its own 32-bit sequence number space and an MPTCP option maps the subflow sequence space to the data sequence space.

A receive window exists at the connection level. Subflows share the same receive window but have different congestion windows. In other words, receive window is per connection but congestion window is per subflow. To achieve resource pooling, congestion window is coupled for each subflow. A coupled congestion control algorithm is proposed [103] for this purpose. It is ‘safe’ which means that it promises to be fair when there is a shared bottleneck with single-path TCP flows.

Hosts are free to use their own policy for spreading data over different paths. There is a mechanism in MPTCP which enables a receiver to notify the sender of which paths are preferred to be used as backup. MPTCP does not offer any mechanism for fine-grained control over preferences of paths. It is proposed that solutions, like using ECN or fake congestion signals [104], be used for this purpose.

Retransmission policy in the event of failure is not mandated by MPTCP and hosts are free to use their own policies. Resending data on different paths is possible; the receiver can handle this by using data sequence numbers.

### 2.3 Analysis of the proposed solutions

Deployability is a key attribute for new Internet protocols. In this section we analyze the proposed solutions, reviewed in 2.2, from a deployability viewpoint. Two important
questions are the basis of our analysis: 1) is the solution able to provide main functionalities of the current solution, which is based on BGP? 2) How difficult would be the deployment? Based on these two questions, we have considered seven aspects in our analysis, which we believe are important and affect deployability of a multihoming solution in the IPv6 Internet.

1. Scalability: Is the solution able to address the scalability issue related to the current solution?

2. Amount of modifications: How much modification is required for deploying the solution?

3. Security: Does the solution improve security of the current solution?

4. Traffic engineering: Does the solution provide more strong features for traffic engineering compared to the current solution?

5. Deployment cost: How much effort is required for deploying and maintaining the solution?

6. Renumbering: How difficult is renumbering a site which employs the solution?

7. Code availability: Is there any implementation available?

2.3.1 Analysis of LISP

- **Scalability:** LISP is a map-and-encap protocol. A packet, which contains EIDs as source and destination addresses, is encapsulated by a border router and its destination address is set to RLOC by adding a new header. Decapsulation is performed by the destination site’s border router. This mechanism makes topological address aggregation possible and allows LISP to scale well. The routing system in the Internet core remains the same.

- **Amount of modification:** LISP defines three new network elements: Egress Tunnel Router (ETR), Ingress Tunnel Router (ITR) and mapping system. An ETR receives LISP-encrypted IP packets from the Internet and sends decapsulated IP packets to the site end-systems. An ITR performs the reverse operation. EID-to-RLOC mapping is also performed by an ITR using a mapping system such as LISP+ALT. Modifications are limited to edge routers (xTRs). No change is required in hosts or the Internet core.
• **Security:** LISP provides a basic protection for communication between xTRs using a 32-bit nonce. This technique does not protect the communications against Man-In-The-Middle attacks. LISP design assumes that mapping systems employ proper technology to protect themselves against attacks. In case of LISP+ALT, it uses BGP and shares its security mechanisms. In other words, LISP+ALT is as secure as BGP.

• **Traffic Engineering:** The mapping system supports weights and priorities in the mapping database, which are useful features for traffic engineering and load sharing. Priorities are used by an ITR to select suitable ETRs for a given prefix, and weights help an ITR to decide how to split load across ETRs of a given priority. This mechanism does not add any state to the global routing system.

• **Deployment cost:** A new software component, like LISP+ALT, is required for EID to RLOC mapping. LISP works for both IPv4 and IPv6. By using LISP, address aggregation is possible and sites are able to benefit from LISP right after the deployment. Encapsulation increases the size of the packets, which may exceed MTU and cause fragmentation and inefficiency. Incremental deployment is possible if the interoperability issue, mentioned in 2.2.1, can be resolved in a reasonable and efficient manner. The mapping process also imposes a processing overhead on the communications. Depending on the type and location of the mapping system, some extra traffic may be imposed on the Internet.

• **Renumbering:** Changing locators does not affect transport and application layer protocols and applications. EIDs will be the same but RLOCs should be updated. Therefore, the border routers and mapping system should be updated to reflect the change. The update mechanism should be real-time otherwise active sessions can not survive in the event of renumbering.

• **Code Availability:** There are four implementations for LISP: OpenLISP [109], implemented for Linux by the Université Catholique de Louvain. Cisco LISP4, an implementation of LISP for Integrated Operating System (IOS) by Cisco. LISP-mob [106] which is an open-source LISP and LISP Mobile Node implementation for Linux, Android and OpenWRT. LISPmob was initially developed by Cisco Systems Inc, and it is currently maintained by the Barcelona Tech University. LISP-Click[110] is another implementation of LISP by the Université Catholique de Louvain.

2.3.2 Analysis of ILNP

- **Scalability:** ILNP employs the concept of identity/locator separation. ILNP Mechanisms are implemented in end-systems and do not affect the core routing system. Each host can have more than one IP address and address aggregation is possible. As a result, service providers can only advertise the aggregated IP prefixes and there is no need for advertising specific prefixes.

- **Amount of modification:** Hosts should be upgraded to support ILNP. DNS service should provide support for new resource records (I, L, PTRI and PTRL). Support for the new ICMP message (Locator Update) should be added to the ICMP protocol. Using FQDN in applications, instead of an IP address, is encouraged although it is not mandated. Legacy applications would be able to work on ILNP-capable hosts with no change.

- **Security:** ILNP employs IPsec to improve the security of communications. It does not include source and destination locators in the IPsec authentication header. Only identifiers are included in the authentication headers so modifying locators does not have any negative impact on the security of important information. ILNP also proposes a nonce destination option as a lightweight security alternative where IPsec is not suitable or unavailable [14].

- **Traffic Engineering:** TE policies can be enforced through border routers [16]. ILNP allows site border routers to rewrite source and destination locators in the packets. It does not affect the transport or application layer, because they use identifiers which are permanent during the session’s life time.

- **Deployment cost:** Host networking software should be modified. DNS service also should be modified to support new resource records. Although ILNP is a new network protocol, it is backward compatible with “pure” IPv6. Routers and IPv6 services, except for DNS and ICMP, do not need any change. An ILNP-capable node can also support ordinary IPv6 at the same time. Thus, an approach like dual-stack looks possible for the transition period. Changing locators is announced to other end-systems using a new ICMP message. That may impose some extra traffic on the Internet.

- **Renumbering:** When a renumbering occurs, all related DNS records should be updated since the mapping between identifiers and locators is stored in the DNS. ILNP hosts are notified of the change by the ICMP locator update message. However, changing the address of special servers, like DNS, is an issue.
• **Code Availability:** A research demonstration implementation of ILNP has been developed in the University of St Andrews\(^5\). Another attempt has been made at Tsinghua University [30].

### 2.3.3 Analysis of NPTv6

- **Scalability:** NPTv6 offers using PI addresses to sites and at the same time provides a mechanism to minimize their negative impact on the routing table size. Each host has its own address, which is accessible from outside, but these addresses do not need to appear in the global routing system. The address-mapping mechanism in NPTv6 is algorithmic and checksum-neutral so there is no need to maintain any per-node or per-connection state in the NPTv6 box.

- **Amount of modification:** No modification is needed in hosts and routing system. Only one or more NPTv6 boxes need to be installed in the site.

- **Security:** With its two-way mapping algorithm, peer-to-peer communication for nodes behind an NPTv6 device is possible, which opens some security concerns compared to NAT44. But it should be noted that the security is not worse than regular IPv6 communications (without NAT). A normal firewall can be used in either case to improve the security.

- **Traffic Engineering:** There is no specific feature in NPTv6 for traffic engineering although traffic engineering rules can be enforced by an NPTv6 boxes. In a multi-addressed site, an NPTv6 box can make the decision based on TE policies, about the source address that should be used.

- **Deployment cost:** A new software component, an NPTv6 box, is required to be installed in the site. Address translation imposes a processing overhead on the communication. The NPTv6 box is a single point of failure, so if an NPTv6 box fails, all connections made through the box will also fail. It is possible to have more than one NPTv6 box in a site to address this issue although the failed connections need to be re-established.

- **Renumbering:** With NPTv6, using PI or ULA addresses internally is possible which makes renumbering easy. Renumbering does not affect hosts. Only NPTv6 box(es) and router(s) should be updated.

- **Code Availability:** There are two implementations of NPTv6 available. An implementation of both stateless and stateful approaches by Université de Liège\(^6\).

---

\(^5\)http://ilnp.cs.st-andrews.ac.uk/

\(^6\)http://nfnat66.sourceforge.net/
Another attempt has been made at Beijing Univ. of Posts & Telecommunication [128].

### 2.3.4 Analysis of HIP

- **Scalability:** HIP is a host-centric solution for secure end-to-end mobility and multihoming which uses the identifierlocator split approach. A set of locators can be associated with each identifier, so hosts can be multi-addressed and address aggregation is possible without any change in the routing system.

- **Amount of modification:** A new protocol layer should be added to the host’s networking software between transport and network layers. Host Identifiers (HIs) should be stored in the DNS or a PKI. A new resource record should be defined and supported if the DNS is chosen to store HIs. Applications should use the extended socket interface if they want to benefit from HIP’s new features. Opportunistic HIP can be used as an option in the situations that using or accessing DNS or PKI is not possible. It works similarly to SSH which adds the public key of the server to the known host file after the first connection.

- **Security:** Data traffic between two HIP hosts is typically protected with IPsec Encapsulating Security Payload (ESP). A pair of Security Associations (SA) is created during HIP Base EXchange process (BEX) by using cryptographic host identifiers. Using these mechanisms, HIP tries to protect communications against DoS and MITM (Man-In-The-Middle) attacks. Protecting against MITM attacks in opportunistic HIP is not easy when unpublished HIs are used because verifying such HIs is almost impossible.

- **Traffic Engineering:** There is no specific feature for traffic engineering in HIP.

- **Deployment cost:** If using DNS for storing HIs is not possible or desirable, a PKI is required. Operating system and networking software vendors should include support for HIP in their products. HIP can be used in both IPv4 and IPv6 environments. There are some issues related to running legacy applications which use IP addresses in the transport layer over a HIP-aware system. Some solutions have been proposed to resolve these issues to facilitate incremental deployment [93]. Both ends of the communication should support HIP to be able to benefit from its features.

- **Renumbering:** Transport layer protocols use HIs so hosts can change their IP addresses without breaking transport layer sessions. Locator changes are signalled
to the communication peer using a three-way UPDATE signaling mechanism. For new communications, published HIs should be updated in DNS/PKI.

- **Code Availability:** OpenHIP, implemented by Boeing for Linux, Windows and Mac OS X is an open-source implementation of HIP\(^7\). HIP for Linux (HIPL)\(^8\) is another implementation for HIP which runs on Linux and Symbian. HIP for inter.net is an implementation of HIP for FreeBSD by Ericsson but seems inactive now. InfraHIP\(^9\) has been developed by Helsinki Institute for Information Technology (HIIT) to study application-related aspects of HIP. cuteHIP is a Java-based implementation of HIP\(^10\).

### 2.3.5 Analysis of Name-based Sockets

- **Scalability:** Name-based sockets enable applications to communicate based on domain names. IP addresses are fully managed by the network layer and completely transparent to transport and application layers. Name-based sockets are equivalent to identifier/locator separation and, therefore, are able to provide a scalable mechanism for routing in the Internet. No change is required in the Internet’s core routing system.

- **Amount of modification:** The host networking stack should be modified to support name-based sockets. Applications also should be modified to use domain names instead of IP addresses.

- **Security:** Using domain names instead of IP addresses makes name-based sockets vulnerable to domain name spoofing. Name-based sockets prevent this attack through a forward lookup in DNS and comparing IP addresses in the DNS response and received packet. It is a weak protection technique especially when DNS is insecure. It also imposes an extra load on DNS and the communication path. Name-based sockets are also vulnerable to redirection and flooding attacks. Techniques, like exchanging random numbers before redirecting the connection, are proposed as protection mechanisms, although they only work for off-path attacks. In fact, name-based sockets do not improve the security of Internet communications. They just try to keep the level of security at the same level as today’s Internet.

\(^{7}\)http://www.openhip.org/
\(^{8}\)https://launchpad.net/hipl
\(^{9}\)http://infrahip.hiit.fi/
\(^{10}\)http://www.nlnet.nl/project/cutehip/
• **Traffic Engineering:** Name-based sockets do not provide any specific feature for traffic engineering.

• **Deployment cost:** No new hardware or software component is required except the new socket layer. Name-based sockets are backwards compatible so incremental deployment is possible for both IPv4 and IPv6. Networking software and OS vendors should add support for name-based sockets to their products. Socket re-use is not possible, even for UDP, because the network layer has to keep a one-to-one coupling between sockets and communication sessions, which has a negative impact on performance.

• **Renumbering:** The design of NBS permits hosts to notify the other end of the communication about locator changes. Therefore, when a renumbering occurs active sessions can survive. For new sessions, DNS records should be updated.

• **Code Availability:** none known

#### 2.3.6 Analysis of SHIM6

• **Scalability:** SHIM6 is a host-centric solution, so it has no impact on the routing system and provides the possibility of using PA addresses for sites and address aggregation for service providers, which makes it a scalable solution.

• **Amount of modification:** Host networking software should be modified to support SHIM6. Applications can be modified to use SHIM6 specific features via the proposed API [74] although it is not mandated. Even without using the proposed API, applications still benefit from the SHIM6 failure-detection and recovery mechanism. No modification is required for the routing system.

• **Security:** SHIM6 does not improve the security of Internet communications. It just tries to keep the security at the same level as regular IPv6 communications. HBA/CGA, context tag and 4-way handshake for context establishment are protection mechanisms for SHIM6-specific features in communications [4].

• **Traffic Engineering:** SHIM6 provides some light-weight and restricted features for traffic engineering. It enable hosts to announce their preferences for their available locators to the other end of the communication. This occurs at host level; network operators still need to have their own traffic engineering mechanisms in place.

• **Deployment cost:** No new hardware or software component is required. Operating system vendors should add support for SHIM6 to their products. It can be
deployed incrementally but it will not be effective until both ends have support for SHIM6. Although SHIM6 has been proposed for IPv6, there is no reason why a similar solution cannot be designed for IPv4 networks.

- **Renumbering:** SHIM6 is able to handle locator changes without breaking the communication. Therefore, keeping active contexts is possible in case of renumbering, although such contexts may create confusion and security issues.

- **Code Availability:** There are four implementations of SHIM6: LinShim6\(^ {11}\) [26], implemented for Linux in the Université Catholique de Louvain. OpenHIP, implemented by Boeing for Linux, Windows and Mac OS X\(^ {12}\). Two other implementations exist for Linux by Tsinghua University [29] and Seoul National University [99].

### 2.3.7 Analysis of MPTCP

- **Scalability:** MPTCP is an extension to TCP and compatible with that. MPTCP enables hosts to create parallel flows through different possible paths to a given host to improve network utilization and fault resiliency. Hosts can be multi-addressed and address aggregation is feasible. There is no need to change the routing system. On the other hand, applications which use other transport protocols are not helped.

- **Amount of modification:** Only hosts should be modified to include MPTCP in their network protocol stack. MPTCP features are transparent to application layer protocols and applications.

- **Security:** MPTCP includes a hash-based handshake algorithm to make it as secure as traditional TCP. The mechanism ensures three key requirements: verifying parties at subflow handshake, verifying new addresses and replay protection. This mechanism provides a medium-level security. New mechanisms are required to achieve a high level of security [21].

- **Traffic Engineering:** Hosts are free to use their own policy for spreading data over different paths. There is no mechanism in MPTCP to enable the receiver to notify the sender of its preferences. Solutions, like using ECN and fake congestion signals, are proposed for this purpose. Network operators still need to have their own traffic-engineering mechanisms in place.

\(^{11}\)http://inl.info.ucl.ac.be/LinShim6

\(^{12}\)http://www.openhip.org/docs/shim6.html
• **Deployment cost:** No new software or hardware component is required for deploying MPTCP, but the operating system and networking software vendors should include support for MPTCP in their products. MPTCP is backwards compatible with regular TCP so that MPTCP-capable hosts are able to communicate with regular TCP hosts, although to benefit from MPTCP specific features, both ends of the communication should have support for that [112]. This makes incremental deployment of this solution completely possible. MPTCP works for both IPv4 and IPv6 and does not require application software to be touched.

• **Renumbering:** MPTCP is able to create/terminate subflows on the fly without affecting the communication. Therefore, although one of the host’s IP addresses is used to establish a connection, in case of renumbering, keeping such connections is possible.

• **Code Availability:** Two versions of MPTCP have been implemented in the Université Catholique de Louvain\(^1\): One in the Linux kernel space based on the LinShim6 code base\(^2\) and another one in user space based on a user-level TCP Stack called Daytona. There is also a NetScaler Firmware implementation from Citrix Systems, Inc\(^3\) and an implementation for iOS7 from Apple, Inc.

### 2.3.8 Summary of the Analysis

Table 2.1 summarizes characteristics of the described solutions\(^4\). Our analysis can be summarized as follows: SHIM6, HIP, MPTCP, ILNP and name-based sockets are solutions with the host-centric approach to multihoming. Although router changes in ILNP are optional, providing important features, like traffic engineering, may not be possible without those modifications. LISP can be considered as an example of the routing approach and NPTv6 is an example of the middlebox approach. The amount of modification which is required for deploying a solution in the Internet is an important factor. Solutions which need fewer modifications would be more desirable because they offer less deployment cost. Location of the modifications is also important. For example, local modifications, e.g. modification in hosts, look much easier to make compared to global modifications; e.g. modifications in the Internet routing system. Only LISP offers a strong set of features for traffic engineering. Other solutions are similar or even weaker than BGP4. Traffic Engineering is an important feature from the

\(^1\)http://nrg.cs.ucl.ac.uk/mptcp/implementation.html

\(^2\)http://mptcp.info.ucl.ac.be/

\(^3\)http://support.citrix.com/proddocs/topic/ns-optimization-10-1-map/ns-mptcp-gen-wrapper-con.html

\(^4\)The table was borrowed from [90]
administrator’s viewpoint since it enables them to control a site’s incoming and outgoing traffic. LISP and HIP have some issues with incremental deployment. As the Internet is a widespread network, incrementally deployable solutions have a higher chance of being adopted. Only NPTv6 is not able to preserve communications in case of failure and renumbering, although SHIM6 and MPTCP also have some issues with renumbering in special cases. Solutions which make renumbering simple are more desirable from a site administrator’s viewpoint because they offer more flexibility in changing service providers.

The co-chairs of the IRTF RRG have recommended the work on ILNP be pursued toward a routing architecture in which multihoming will be one of the main features [81].

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Solution</th>
<th>LISP</th>
<th>ILNP</th>
<th>NPTv6</th>
<th>MPTCP</th>
<th>HIP</th>
<th>NBS</th>
<th>SHIM6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Modifications</td>
<td></td>
<td>ER</td>
<td>H, SP, A*, ER*</td>
<td>N/A</td>
<td>H</td>
<td>H, A, SP*</td>
<td>H, A</td>
<td>H, A*</td>
</tr>
<tr>
<td>Security (Compare to BGP4)</td>
<td></td>
<td>Similar</td>
<td>Stronger</td>
<td>Similar</td>
<td>Similar</td>
<td>Stronger</td>
<td>Similar</td>
<td>Similar</td>
</tr>
<tr>
<td>TE (Compare to BGP4)</td>
<td></td>
<td>Stronger</td>
<td>Similar</td>
<td>Weaker</td>
<td>Weaker</td>
<td>Weaker</td>
<td>Weaker</td>
<td>Weaker</td>
</tr>
<tr>
<td>Incremental Deployment</td>
<td></td>
<td>Possible with Conditions</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible with Conditions</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Renumbering without breaking established communications</td>
<td></td>
<td>Possible</td>
<td>Possible</td>
<td>Possible with Conditions</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible with Conditions</td>
</tr>
<tr>
<td>New Component</td>
<td></td>
<td>Mapping System</td>
<td>None</td>
<td>NAT Device</td>
<td>None</td>
<td>PKI*</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Code Availability</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

*: optional  A: Application  ER: Edge Router  H: Host  SP: Internet Services and Protocols

Table 2.1: Summary of characteristics of the seven active solutions for IPv6 multihoming.

## 2.4 Conclusion

This chapter presents a review of seven multihoming solutions for IPv6 which are currently under investigation. Although a large number of solutions have been proposed for this problem, only a few of them satisfy necessary technical requirements and therefore have a chance of being chosen by the technical community as the standard solution. We summarized and analyzed seven important solutions which are active in this area. Results of our analysis show that each solution has its own drawbacks and weak points so that it is difficult to choose one of them as “the perfect solution”. More research and effort is needed for achieving a scalable, deployable, manageable and secure solu-
tion for IPv6 multihoming and all active solutions still have a chance of improving and becoming the standard solution.

Making change in a huge and complicated system, like the Internet, is difficult; so host-centric solutions which only modify end hosts and do not need any change in the Internet routing system are easier to deploy. SHIM6 and MPTCP are two host-centric solutions which need minimum changes, even among host-centric solutions, for deployment. Both are incrementally deployable and both are equipped with a specific mechanism for handling communication path failures. Because of these features, this thesis focuses mainly on the properties of SHIM6 and MPTCP, but all solutions deserve further attention.
Chapter 3

Stochastic Activity Networks

Contents

3.1 Introduction ........................................... 38
3.2 SAN Elements .......................................... 40
3.3 Dynamic Behaviour of the SAN ....................... 41
  3.3.1 Example 1 ........................................ 42
  3.3.2 Example 2 ........................................ 43
3.4 Möbius Modeling Tool ................................. 45
In this chapter, we describe Stochastic Activity Networks (SAN), which is a suitable formalism for modeling concurrent and distributed systems. We have used this formalism to build models for our simulation studies in chapters 4 and 5.

3.1 Introduction

Rapid evolution of technology has led to a world of complicated systems. Designing such systems takes a lot of time and money. Speed, optimized usage of resources and fault tolerance are three major necessary characteristics of most new systems. Performance evaluation techniques allow researchers and designers to describe these characteristics in the form of measurable parameters.

Modeling is one of the major techniques for performance evaluation. In this technique, first a model of the system is constructed and then different experiments are conducted. Modeling can be used before (in design phase) and also after constructing the systems. Such flexibility has turned modeling into a desirable and popular technique. To build a model, we need a formal method which allows us to include important details of the real system in the model. Because of the difference in the nature of real systems, a wide variety of modeling formalisms have been introduced.

One, and perhaps the most important, of those formal methods is queuing networks [27]. A queuing network is able to model complicated systems which share independent resources but is weak at describing some details like concurrency, blocking and splitting. Because of these issues, it is not considered to be a suitable tool for modeling distributed and concurrent systems.

Petri nets are another formalism for modeling. A Petri net has suitable facilities for describing concurrency and synchronization, which make it suitable for modeling distributed and parallel systems. There is no concept of time in standard Petri net. In a Timed Petri Net [129], time is defined as a number and is assigned to each transition. This number shows the execution time of the transitions. Molloy and Natkin, independently and nearly at the same time, introduced the Stochastic Petri Net (SPN) [86] and showed that it was isomorphic to a continuous-time Markov chain. After them, Marsan introduced the Generalized Stochastic Petri Net (GSPN) [82]. GSPN is equipped with a new type of transition called instantaneous transition, which eliminates some disadvantages of SPN.

The Petri net is a simple formalism. This simplicity makes the process of modeling difficult. A lot of effort has been made to resolve this issue and facilitate the modeling process. Stochastic Activity Network (SAN) [85] is the result of one of those efforts. SAN adds instantaneous activities and input and output gates to SPN to facilitate modeling. SAN was introduced at the same time as GSPN, and the concept of
instantaneous activity was presented by both of them independently.

The computing power of a Petri net with inhibitor arcs is equal to a Turing machine. If time is ignored and its gate’s functions are computable, the computing power of SAN and GSPN is equal. A function or predicate, which is assigned to each gate, can be replaced by a network of places and instantaneous activities because it is supposed that executing these functions and predicates takes no time. Execution time of each transition in a SAN can be described by a random variable with any desired distribution. Queuing networks exhibit the uncertain behaviour of concurrent systems in the form of temporal uncertainty. But, there is also another type of uncertainty in complicated systems, spatial uncertainty. This means that the next state of the system after finishing an activity is not definite. In a queuing network model, this uncertainty is quantified by an assignment of branching probabilities to the output connections of each node. SAN is able to exhibit both types of uncertainty.

SAN-based models can be analyzed with both analytical and simulation techniques, but it is too hard to be done manually. Instead, having tools that are able to perform such analysis automatically is necessary. Software tools, like Möbius [41], are able to provide this functionality.

SAN models are simple and flexible. A system which can be described by a finite state machine can also simply be modelled by a SAN. It makes SAN a good choice for modeling network protocols. SAN has been used by other researchers as a modeling tool in different areas, including network protocols and communication systems [7][92][113][108][60][1]. Using SAN and Möbius, a modeller can model different parts of a network protocol in different levels of detail. Tools like ns-2/ns-3 and OPNET are more popular than SAN for simulating network protocols but, at the time we started creating the models, there were concerns about scalability [91][51], complexity and bugs¹ in ns-2/ns-3. OPNET was not freely available at that time. We needed a reliable and flexible tool for large scale simulation, so we decided to use SAN and Möbius.

We use SAN in chapters 4 and 5 to create simulation models of SHIM6 and MPTCP. Flexibility of the SAN helps us to focus only on the parts of the protocol that are important in our study. For example, in the MPTCP model, we are interested in focusing on the SEMICOUPLED congestion control algorithm. We model the congestion control algorithm in detail while other parts of the protocol are modelled in less detail and at a higher level of abstraction. This chapter describes the SAN and gives examples which show its strength and flexibility in modeling networking protocols and systems. It also describes Möbius and its useful features for our study. Möbius is the modeling tool that we use for simulating SAN models of REAP and SEMICOUPLED in chapters 4 and 5.

¹http://en.wikipedia.org/wiki/Ns_simulator#Criticism


3.2 SAN Elements

A stochastic activity Network (SAN) consists of the following primitive elements:

- **Activities**, which are of two kinds: timed activities and instantaneous activities. Timed activities are depicted with thick lines (\[\text{\textcolor{black}{\textbf{ }}}\]). A function, called an *activity time distribution function*, is associated with each timed activity. This function generates the required time for the process which is modelled by the activity. Instantaneous activities are depicted with thin lines (\[\text{\textcolor{black}{|}}\]) and can model quick processes which do not need any time. For example, a timer can be modelled by a timed activity while processing of a packet in a host, if the processing time is not important for the modeller, can be modelled by an instantaneous activity.

- **Places**, which are depicted with a circle (\[\text{\textcolor{black}{\textbullet{}}}\]). Each place should contain 0 or more tokens. Tokens are objects which are able to move between places. A socket buffer, for example, can be modelled by a place. A token is a good representation for a packet.

- **Input gates**, which have a finite set of inputs and just one output. An input gate connects one or more places to an activity. An input gate with n inputs is depicted as \[\text{\textcolor{black}{\textbullet{}}}\]. For each input gate, a computable function called *input function* and a predicate called *input predicate* are defined. Input predicate describes a condition based on the number of tokens in input places. An input gate is a good object for modeling preconditions of a process. For example, an input gate can be used to control the size of a buffer. The buffer can be modelled by a place and the number of tokens in that place can be considered as the number of data packets in the buffer. A timed activity can model processing of the packets. An input gate, which connects the place and the activity, can delay the processing if the buffer is not full, for example.

- **Output gates**, which have a finite set of outputs and just one input. An output gate connects an activity to one or more places. An output gate with n outputs is depicted as \[\text{\textcolor{black}{\textbullet{}}}\]. For each output gate, a computable function called *output function* is defined. An output gate is a good object for modelling the actions that should be taken when a process is finished. For example, an output gate which connects an activity, that models processing a packet, to a place, which models a buffer, can discard the packet if the buffer is full.

- **Cases**, which are depicted with small circles (\[\text{\textcolor{black}{\textcircled{}}}\]) on the activities. A probability is associated with each case which identifies the probability of choosing that case when the activity fires. Cases can be used to model uncertainties in the system.
For example, when a system sends out a packet it expects to receive an acknowledgement in response but there are other possibilities, e.g., receiving an ICMP error message, as well. These possibilities can be modelled using cases which are associated to the activity that models the sending process.

To simplify drawing, an input gate with one input, enabling predicate \( e(x) : x \geq 1 \) and input function \( f(x) = x - 1 \) and an output gate with one output and output function \( g(x) = x + 1 \), are shown as simple arrowheads. \( x \) specifies the number of tokens in an input/output place and is called marking of the place. Marking is also defined for the whole SAN model. The set of markings of places is called the marking of the model and, in fact, specifies the state of the model. Changing marking of a place will change the marking of the model which shows a change in the state of the system.

An input gate in a marking is on hold if the value of its enabling predicate is true. An activity is called enabled in a marking if all input gates which are connected to that are on hold. In other words, an input gate allows an activity to start its processing if all its preconditions are satisfied. A marking/state is called stable if no instantaneous activity is enabled; otherwise it is called unstable. A stable marking/state shows that the system is processing something while an unstable marking/state shows a quick transition which takes no time.

A reactivation predicate can be associated to each activity. Enabling time for an enabled activity resets when the reactivation predicate for that activity is evaluated as true. For example, when TCP sends out a packet, it starts a timer called retransmission timer. The timer can be modelled by a timed activity. When an acknowledgement is received, it stops the timer if there is no unacknowledged packet in the congestion window; otherwise it resets the timer. A reactivation predicate can be used to model resetting the timer when there are still some unacknowledged packets in the window.

### 3.3 Dynamic Behaviour of the SAN

A SAN is a dynamic system and is executed as time goes on. Instantaneous activities fire as soon as they get enabled. A time duration is defined in the form of a constant number, or random variable, for each timed activity. This duration, which is called enabling time, specifies the time required for the process modelled by the activity to be completed. When a timed activity becomes enabled, it waits for its enabling time to pass and then fires. Firing an activity models completion of a process. Timed activities are able to fire only in stable markings. Firing two or more activities at the same time is impossible. Firing is performed in three steps in the following order:
1. If the activity has cases, a case is chosen according to the defined probabilities.

2. Input functions of all the connected input gates are executed.

3. Output functions of all the output gates connected to the chosen case are executed.

Firing an activity may change the number of tokens in its input/output places and create a new marking. Such a new marking may change the status of activities in the model from enabled to disabled and vice versa. Each stable marking describes performing a time consuming process in the modelled system. Instantaneous activities help the modeller to model the transitions which are important but their processing time is negligible. It is assumed that the system does not spend any time in unstable states.

The system starts in an initial marking. At least one activity should be enabled in this marking. When the activity fires, its input and output gate(s) functions are executed which may change the number of tokens in their input/output places and create a new marking. In the new marking, new activities might be enabled. Then the first one eligible to fire will fire and move tokens along places and create a new marking. This process continues until the system reaches the point that there is no enabled activity in the model. That would be the end state for the model and shows that the system has finished.

The following two examples show how SAN can be used for modeling network protocols and concurrent systems.

### 3.3.1 Example 1

Consider a simple example which models a host that sends 1000 packets to a remote host. The sending host sends a packet and waits for its acknowledgement from the receiving host. When it receives the acknowledgement, it sends the next packet. Fig. 3.1 shows the graphical representation of this system.

![Figure 3.1: Graphical representation of the SAN model for example 1.](image-url)
Place *Sender* models the sending host. The initial marking for this place is set to 1000. Every token in this place models a packet which should be sent to the receiver. Activity *Send_packet* models the sending process. It is a timed activity and the time required for a packet to be sent by the sender and received by the receiver should be associated to this activity. Place *Receiver* models the receiving host. Activity *Send_ack* models the process of sending an acknowledgement back to the sender. If there is a packet in place *Ack*, it shows that an acknowledgement has been received by the sender, so a new packet may be sent. The initial marking of place *Receiver* is set to zero which means that no packet has been received yet. The initial marking of place *Ack* is set to one which means that a packet can be sent. Therefore, in the initial marking, *Send_packet* is enabled because both *Sender* and *Ack* contain tokens. *Send_packet* fires when its enabling time is passed. As a result, one token is added to place *Receiver* and one token is removed from both *Sender* and *Ack*. In the new marking, *Send_packet* is not enabled because *Ack* is empty but activity *Send_ack* is enabled because there is a token in *Receiver*. This state models a situation where the receiver has received a packet and should send an acknowledgement and the sender is waiting for the acknowledgement. When *Send_ack* fires, it removes the token from *Receiver* which makes *Send_ack* disabled and adds a token to *Ack* which makes *Send_packet* enabled. Now, the sender can send the next packet. This process continues until *Sender* and *Receiver* become empty. When both *Sender* and *Receiver* are empty, no activity is enabled. This is the end state and shows that the sender has already sent 1000 packets to the receiver.

### 3.3.2 Example 2

Consider a system with two servers. The first server is twice as fast as the second one. Jobs enter the system and exit after getting service. Each Job is processed by one of the servers. The faster server has higher priority so when both servers are available the job is assigned to the faster one. Jobs have to wait in a queue in the case that both servers are busy. This queue has a limited capacity. Jobs that encounter a full queue exit the system immediately. Service times for different jobs are different.

Figure 3.2 shows a graphical representation of a SAN which models this system. The initial marking for Places *Fast*, *Slow* and *P* is 1. Place *Jobs* represents the queue and gate *OG2* controls its capacity. Activity *Entrance* models entering jobs to the system. If the first server (the faster one) is free, there would be one token in place *Fast*. Place *Slow* plays the similar role for the second server. Place *WhichSel* shows which server has been chosen for running the job. Gate *IG1* is open if one of the servers is free. Assigning incoming jobs to servers is done by *SelProc* and *OG1* based
on the number of tokens in WhichSel. FastProc and SlowProc model the processing which is done by the servers. Variant inter-arrival time and processing time for jobs is modelled by assigning proper activity time distribution functions to Entrance, FastProc and SlowProc. These functions generate random values, as enabling times, whenever these activities get enabled.

![Graphical representation of the SAN model for example 2.](image)

**IG1:**

Predicate: MARK(Fast)\(!=0 \ || \ MARK(Slow)\(!=0

Function:

\[
\text{if (MARK(FAST)\(!=0 \}) \{ \\
\text{MARK(Fast)} = 0; \\
\text{MARK(WhichSel)} = 0;
\} \]

\[
\text{else \{ \\
\text{MARK(Slow)} = 0; \\
\text{MARK(WhichSel)} = 1;
\}}
\]

**OG1:**

\[
\text{if (MARK(WhichSel) == 0)}
\]

\[
\text{MARK(Fast Alloc)} = 1
\]

\[
\text{else}
\]

\[
\text{MARK(Slow Alloc)} = 1
\]

**OG2:**

\[
\text{if (MARK(Jobs) < 10)}
\]

\[
\text{MARK(Jobs)} = \text{MARK(Jobs)} + 1;
\]

Figure 3.2: Graphical representation of the SAN model for example 2.

At the initial state, the marking for Fast, Slow and P is 1. So, only Entrance is enabled. In fact, the system is waiting for the first job to arrive. When Entrance fires, the token is removed from P. Then output function of the gate OG2 is executed and adds one token to the place Jobs. The simple arrow from Entrance to P returns the token removed right after firing the activity to P. The marking of the system changes and the system makes transition to a new state.
At the new state, two activities are enabled: \textit{Entrance} and \textit{SelProc}. \textit{SelProc} is enabled because \textit{IG1} is on hold and there is a token in the place \textit{Jobs}. \textit{Entrance} is enabled because there is a token in \textit{P}. \textit{SelProc} fires immediately since it is an instantaneous activity. Input function of \textit{IG1} is executed and selects fast server to run the job. Then output function of \textit{OG1} is executed and assigns the job to the fast server by putting a token in place \textit{FastAlloc}.

At the new state, \textit{Entrance} and \textit{FastProc} are enabled. If \textit{FastProc} fires first, it removes the token from \textit{FastAlloc} and puts it in the place \textit{Fast} which means that the job was processed and the fast server is ready to run a new job. If \textit{Entrance} fires first, it enables \textit{SelProc} and the slow server is selected, this time for the new job since the fast server is still busy. Then \textit{IG1} removes the token from \textit{Slow} and \textit{OG1} puts a token in \textit{SlowAlloc} which makes the \textit{SlowProc} enabled. If a new job arrives at this stage, it has to wait in the queue until one of the servers becomes available (until \textit{FastProc} or \textit{SlowProc} fires and puts a token in \textit{Slow} or \textit{Fast} and makes the \textit{SelProc} enabled).

The size of the queue, modelled by \textit{Jobs}, has been limited to 10. This restriction is applied by \textit{OG2}. A new job is added to the queue, only if the number of tokens in \textit{Jobs} is less then 10. Each token in \textit{Jobs} represents a pending job in the queue. Jobs that face a full queue are ignored.

### 3.4 Möbius Modeling Tool

Analyzing SAN models is a complicated task and cannot be done manually. A modeling tool is essential for this purpose. We picked Möbius as the modeling tool for our research. At the time that we started modeling, there were other SAN tools like SharifSAN [3] and SANBuilder [2] available but we could not find any stable and reliable version of them. Möbius is stable and is supported by the developers. It is also free for academic use.

Möbius [41] is a powerful tool for modelling static and dynamic behaviour of complex systems. It enables the modeller to build a model of a large system by composing small submodels. Möbius supports different modeling formalisms. SAN is one of them. Möbius does not limit the user to its predefined components. New components can be developed, using the C++ programming language, and used in models. Möbius provides an integrated development environment, including graphical editor, database for results and a drawing tool for drawing graphs. This environment makes the process of modelling and analysis simple.

If a SAN model satisfies some conditions, it can be solved using analytical methods. Simulation is a general technique which is always applicable to the SAN models. Möbius provides support for both types of analysis. In addition to the standard SAN elements
and features, Möbius provides some more features which make the modeling process easier.

Möbius supports another type of places called extended places. An extended place can represent a structure or array of primitive types. That enables the modeller to store different types of information in the model.

Another useful feature of Möbius is its Rep/Join feature. Using this feature, a modeller is able to design the model in a modular manner. Each component of the system can be modelled as a separate SAN sub-model. Sub-models can be replicated, using the Rep feature, in order to model different instances of the same component in the system. There is no limitation on the number of replicas, except for the available memory of the system, so by having a large main memory and using a 64-bit hardware and operating system, Möbius is able to run a model of a very large system with thousands of replicas. Using the Join feature, SAN sub-models can be joined together to build a composed model of a complex system. Möbius enables sub-models in a composed model to share places. By sharing places, sub-models are able to access each other’s state variables, which makes it easy to synchronize different tasks in different sub-models. These features make Möbius a suitable tool for modeling parallel and concurrent systems. We use these features in chapters 4 and 5 to create simulation models of SHIM6 and MPTCP.

Each SAN model can have a set of global variables that enables the modeller to design different experiments with the same model. Model parameters can be defined as global variables and passed to the model before executing experiments. Möbius provides a study editor for this purpose. Users can define studies. Each study is a set of values for the model’s global variables and represents an experiment.

Möbius provides a SAN editor to facilitate creating SAN models. Users can easily choose SAN elements from a toolbar and put them on the drawing area to create a graphical representation of the model. Element attributes can be set by clicking on the element and choosing associated properties.

Möbius relaxes some of the SAN restrictions. Functions, i.e. gate functions, activity time distribution functions and case distributions, have access to all local and shared places. To use the marking of a place in an output gate function, for example, there is no need to connect them in the graphical representation; so the modeller can include only important connections in the graphical representation which makes it easier to manage and understand the model.

Möbius is able to compile SAN models for different target platforms and create stand-alone executables. The executable can run on any machine which runs the target platform and there is no need to have Möbius installed. The results can be stored in binary files and then moved, for processing, to a system which has the Möbius IDE.
Möbius is also able to run simulation over a network and use a set of computers for this purpose. It is very useful for large scale simulation since the simulation process can be distributed over a set of computers in a network. This effectively reduces simulation time.

SAN is used for creating simulation models and Möbius is used for running simulation experiments in chapters 4 and 5. Most graphs in those two chapters are also generated by Möbius.
Chapter 4

SHIM6

Contents

4.1 Introduction ..................................... 51
4.2 Related Work .................................... 52
4.3 REAchability Protocol .......................... 52
4.4 Congestion Control Algorithms ............... 54
  4.4.1 TCP Congestion Control Algorithm ....... 55
  4.4.2 TCP Friendly Rate Control ................ 55
4.5 Internet Experiments .......................... 57
  4.5.1 Experiment Setup .......................... 57
  4.5.2 Results from Internet Experiments ....... 60
4.6 Lab Experiments ................................ 65
  4.6.1 Experiment Setup .......................... 66
  4.6.2 Results from Lab Experiments .......... 67
4.7 Operational Issues ............................ 70
4.8 Model Description .............................. 71
  4.8.1 REAP Sub-model ............................ 71
  4.8.2 Application Sub-model ..................... 72
  4.8.3 SHIM6 Sub-model ......................... 76
  4.8.4 Limitations ............................... 76
4.9 Validating the Model ............................ 78
4.10 REAP Performance in Large Scale Networks .. 80
  4.10.1 REAP Recovery Time ....................... 81
  4.10.2 REAP Traffic ............................... 82
  4.10.3 Effect of the REAP Parameters on Performance ...... 84
4.11 TFRC Improvement .............................. 89
4.12 Transport to REAP Notification ............... 91
This chapter presents the results of a number of experiments with SHIM6 in the lab and over the IPv6 Internet. The aim of the experiments is to investigate how fast SHIM6 is able to recover from a path failure and how much traffic is generated during the recovery process. Using a SAN-based simulation model, we also investigate the behaviour of REAP in case of path failure in a large-scale SHIM6-enabled site with 10,000 REAP instances and evaluate a number of improvements to REAP which can reduce recovery time and traffic. A signaling mechanism, which improves the efficiency of TFRC in SHIM6-enabled networks, is also proposed and evaluated.

4.1 Introduction

SHIM6 [97] is one of the proposed solutions that was chosen by IETF as an engineering solution for multihoming in IPv6. SHIM6 is a host-centric solution, which means that it is fully implemented in hosts and does not affect other components of the Internet. SHIM6 allows hosts to be multi-addressed. That means a multihomed SHIM6-enabled site does not need to have its own provider independent address block, instead it can use address blocks which are assigned by its providers. Address aggregation is possible, which resolves the scalability problem that was discussed in section 1.1.

SHIM6 inserts a layer between IP and transport layers to monitor long-lived communications. Important information about each communication is stored in a data structure called a context. A context is created when a communication is established between two hosts and SHIM6, using some heuristics, recognizes that it will be a long-lived communication. All upper layer communications between two hosts may share the same context. SHIM6 employs an identifier/locator separation scheme. The IP address used for establishing a communication is considered as an identifier and is called Upper Layer ID (ULID). The host’s other IP addresses play the role of locators. If a path failure occurs, SHIM6 activates REAchability Protocol (REAP) [10] to explore a new path using both hosts’ locators. When the new path is found, the shim layer inside the IP layer is activated. It moves the communication to the new path by rewriting source and destination addresses in the IP header. All incoming and outgoing packets are inspected and their addresses are rewritten if required. ULID remains unchanged which makes the address switching process transparent to the upper layers.

In this chapter, we present the results of a study on the behaviour of two SHIM6-enabled hosts by conducting a number of experiments in the lab and on the Internet. The aim of the experiments is to study the behaviour of REAP in case of link failure. We are specifically interested in measuring how long it takes for REAP to detect and recover a link failure and how much traffic is generated by REAP during the recovery process. We have also built a simulation model of a SHIM6-enabled host and
conducted experiments to investigate the behaviour of REAP in a large scale IPv6 network. Based on our findings from real experiments and simulation, we also propose some enhancements to improve performance in SHIM6-enabled networks.

4.2 Related Work

Sebastien Barre et al. present the results of their evaluation of the performance of reachability protocol (REAP) in a lab environment in [25]. Antonio de la Oliva et al. [38] simulate a SHIM6-enabled environment and perform a performance study on REAP. They study the effect of different values for send timer on TCP and UDP applications recovery time. They also propose an improvement for TCP which reduces the recovery time. Lizhong Xie et al. [127] propose an improved address exploration method which maintains address switching time at a constant value. Antonio de la Oliva et al. present the results of an analytical study on REAP performance in [39]. J. Ronan et al. made an empirical evaluation of LinShim6 [107].

The differences between the work presented in this chapter and the above mentioned research can be summarized as follows:

- We run our experiments in the lab and also over the Internet. In the lab, we run experiments with more than four address pairs as well. To the best of our knowledge, our experiments over the IPv6 Internet and lab experiments with more than four address pairs are the first attempted using SHIM6.

- We study the behaviour of REAP and evaluate the effect of two improvements by simulating a large scale SHIM6-enabled network.

- We propose a signaling mechanism, similar to what is proposed in [38], to be used by TCP Friendly Rate Control (TFRC). We describe how the mechanism should be adopted. We also show the effect of this improvement by using our simulation model.

4.3 REAchability Protocol

SHIM6 employs a protocol for failure detection and recovery called REAchability Protocol (REAP) [10]. REAP is responsible for two main jobs: detecting failure and finding a new operational address pair, which is called address pair exploration, to recover from the failure. REAP uses FBD (Force Bidirectional Detection) for verifying reachability and detecting failure. The idea is that if there is traffic in one direction there also should be traffic in the reverse direction. REAP sends keep-alive messages in the case
that there is incoming traffic but there is no data to be sent in return. It would be a
sign of failure if there was outgoing traffic but no traffic in return.

To manage this process, REAP employs two timers: \textit{send timer} and \textit{keep-alive
timer}. When a payload packet is sent, REAP starts send timer and stops keep-alive
timer. When a packet is received from the peer, REAP stops send timer and starts
keep-alive timer. When keep-alive timer expires, it means that it is time to send a
keep-alive message to the peer in response to the already received packet. When send
timer expires, it means that there has been no incoming traffic for a while which is
a sign of failure. Thus, when send timer expires, REAP considers it as a failure and
starts an address exploration process to find a new working pair of addresses. First,
using known local and peer addresses, a list of address pairs is created. Each address
pair consists of two addresses: one local IP address and one remote IP address. So,
the list contains number of local IP addresses $\times$ number of remote IP addresses entries.
For example, if each host has three IP addresses, the list will have nine entries. This
list is pruned and ordered using address selection rules described in [117]. Then REAP
starts to check reachability of the entries in the list by sending probe messages to the
peer using the list entries. The list is traversed sequentially and one probe is sent to the
remote host for each address pair. The first four probes, which are called \textit{initial probes},
are sent with a time interval equal to 0.5 seconds. For the next probes, an exponential
backoff procedure is employed to avoid signaling storms. This procedure increases
the time between probes until it reaches 60 seconds. Later probes will be sent with
this time interval. Most implementations double the time between probes although
this type of increase is not mandated by [10]. Probes are sent with different source
addresses; to make sure that they are forwarded to the right provider, a mechanism
like Source Address Dependent Routing (SADR) should be enabled on the site’s edge
router. Service providers usually enable ingress [50] filtering to protect themselves
against source address spoofing attacks. As a result, probes which are forwarded to the
wrong provider will be discarded and will not have a chance to reach the remote host.

This process is finished when a working address pair is found. When the peer
receives a REAP probe, it starts a similar process to find a working address pair. As
a result, two ends of the communication may use different address pairs after recovery.
SHIM6 conceals the whole process from the transport layer. The transport layer only
experiences a delay which might be long depending on the time required for the address
exploration process. Fig. 4.1 shows an example of how REAP reacts to a failure.
4.4 Congestion Control Algorithms

Network congestion occurs when a link has to carry so much data that it is not able to operate in its normal way. The typical consequences of this event are queuing delays and packet loss, which directly affect throughput. To control the congestion and minimize its negative effects, network protocols usually employ algorithms. These algorithms are called congestion control algorithms. Link failures are treated like severe congestion by the transport layer. Since the focus of this chapter is on how SHIM6 reacts to a link failure and how its reaction affects its transport layer and therefore applications, behaviour of the congestion control algorithm employed by the transport layer is important for our study.

In this section, we briefly describe two congestion control algorithms: TCP congestion control algorithm and TCP Friendly Rate Control (TFRC) algorithm. We have chosen TCP congestion control algorithm because it is used by TCP which is a widely used transport protocol on the Internet\(^1\). To extend the scope of our study, we have also chosen TFRC which is a rate-based mechanism, suitable for audio/video streaming applications, and works differently from the TCP congestion control algorithm.

\(^1\)http://www.caida.org/data/passive/trace_stats/
4.4.1 TCP Congestion Control Algorithm

The standard TCP congestion control algorithm is described in [8]. TCP, in fact, employs four different algorithms to control congestion in different situations:

1. **Slow Start**: This algorithm is used by a sender at the beginning of the transmission or after repairing a loss detected by the retransmission timer. In this mode, the congestion window is increased by one SMSS (Sender Maximum Segment Size) for each received acknowledgement. TCP stays in slow start until the size of the congestion window exceeds the ssthresh (Slow Start Threshold). The initial value of ssthresh is set arbitrary high but it is reduced when a congestion is detected. When congestion window size exceeds ssthresh, TCP switches to congestion avoidance mode.

2. **Congestion Avoidance**: In this mode, the congestion window is increased by one full-size segment every RTT. TCP continues in this mode until a loss is detected. Receiving three consecutive duplicate ACKs or expiration of the retransmission timer notifies the TCP sender about a loss. The retransmission timer is started when a segment is sent out. Expiration of this timer means that TCP did not receive an ACK for that segment in a reasonable time interval. The value of this timer is calculated based on the RTT of the communication link. A duplicate ACK is a notification, sent from receiver to sender, to inform the sender about receiving an out-of-order segment. The missing segment might be lost or might arrive later. If three duplicate ACKs are received, TCP switches to Fast Retransmit. If the retransmission timer expires, it switches to Slow Start.

3. **Fast Retransmit**: TCP switches to this mode when it receives three duplicate ACKs. Receiving three duplicate ACKs is an indication of segment loss, so the sender should retransmit the missing segment without waiting for the retransmission timer to expire.

4. **Fast Recovery**: After sending a missing segment in Fast Retransmit mode, TCP switches to this mode and continues sending new data, as it would in congestion avoidance mode, until a non-duplicate ACK is received.

4.4.2 TCP Friendly Rate Control

TCP Friendly Rate Control (TFRC) [52] is a rate-based congestion control mechanism which minimizes quick changes in transfer rate. The goal of TFRC is to be fair when competing for bandwidth with TCP flows while it reacts more smoothly to bandwidth changes than the TCP congestion control algorithm. It is suitable for unicast flows
which need a smooth throughput. Applications, like media streaming applications with a buffering mechanism, are able to benefit from TFRC. In contrast to TCP-like congestion control mechanisms, TFRC gradually reduces sending rate when congestion is detected in the communication path.

The implementation of Datagram Congestion Control Protocol (DCCP) [73] for Linux can be configured to use TFRC as its congestion control mechanism. Congestion Control ID 3 (CCID3) [53] has been assigned to TFRC. We have used DCCP with CCID3, implemented in the Linux kernel, in our experiments. Most of the media streaming applications follow a DCCP half-connection (HC) scenario. An HC-sender sends application data and an HC-receiver responds with acknowledgements. The receiver sends acknowledgements at least once per round-trip time. Acknowledgements also contain some feedback information which helps the sender to adjust its sending rate. Although DCCP is a connectionless protocol, its header includes a sequence number which enables the receiver to detect lost packets. In addition to that, the receiver also notifies the sender of recent loss intervals. Using this feedback information, the sender tries to keep its sending rate at a suitable level for the receiver. A timer, called nofeedback timer, is employed by the sender for detecting congestion. The value of this timer is calculated based on round-trip times between sender and receiver. If this timer expires, it means that nofeedback has exceeded four round-trip times, so the sender concludes that congestion has occurred in the path. Therefore, the sending rate is halved to reduce traffic and risk of packet loss. When a loss occurs, the receiver reports it to the sender in the first feedback which is sent after loss. The amount of loss is reported as “loss event rate”, denoted by \( p \) in [53]. \( p \) describes the rate that loss events have occurred so far. It is reported as a real number between 0 and 1. 0 means no loss has been detected so far, which gives the sender an opportunity to increase its maximum allowed sending rate to at most twice the reported receive rate. The receiver’s receive rate is always considered as an upper limit for the sender’s sending rate.

When the receiver reports a loss (\( p > 0 \)), the sender uses \( p \) in the following formula (TFRC’s throughput equation) to calculate a new sending rate which fits the current conditions of communication link and receiver:

\[
X = \frac{s}{R \sqrt{2bp/3} + (t_{\text{RTO}}(3\sqrt{3bp/8})p(1 + 32p^2))}
\] (4.1)

where:

\( X \) is the transmit rate in bytes/second.
\( s \) is the packet size in bytes.
\( R \) is the round trip time in seconds.

\( p \) is the loss event rate, between 0 and 1.0, of the number of loss events as a fraction of the number of packets transmitted.

\( t_{RTO} \) is the TCP retransmission timeout value in seconds.

\( b \) is the number of packets acknowledged by a single TCP acknowledgement.

The sending rate directly affects packet transmission scheduling. When TFRC receives a packet from an application, it calculates a nominated time (\( t_{nom} \)) for sending it according to the current sending rate. When the application sending rate is greater than TFRC’s sending rate, it is possible that some packets are buffered and scheduled to be sent later to respect the TFRC sending rate.

### 4.5 Internet Experiments

We set up a test environment which enabled us to run a set of experiments with LinShim6 over the Internet. LinShim6 is an implementation of the SHIM6 protocol, developed in the Université Catholique de Louvain, for the Linux operating system. This section describes the hardware and software configurations for these experiments. It also presents the results. We start with the Internet experiments because results from these experiments show important details more clearly than lab experiments. This makes it easier to describe the effect of those details in the results from lab experiments. The rest of this section describes the environment for the experiments and discusses the results.

#### 4.5.1 Experiment Setup

Fig. 4.2 shows the configuration of the test environment. There are two SHIM6-enabled multi-addressed hosts, located in Auckland and Dublin\(^2\). Each host is equipped with two network interface cards and configured with two prefixes from two different providers. The SHIM6 host in Auckland is connected to a router (Lab router). Lab router is a Linux machine which has been configured as an IPv6 router. This router simulates link failures for the experiments. The university edge router has been configured to forward packets to different providers according to their destination address. This configuration enabled us to run experiments with four address pairs over the Internet.

To simulate a failure, all paths except one are blocked on the lab router to push SHIM6 to switch the active communication to the open path. Utility `ip6tables` is used

---

\(^2\)Thanks to John Ronan from Telecommunications Software & Systems Group, Waterford Institute of Technology, who helped us in running these experiments between Auckland and Dublin. He setup a multihomed SHIM6 host in Dublin for these experiments.
for blocking paths. We have developed a client/server program to schedule failures on the lab router. We call it failure scheduler. The server side of this program runs on the lab router and the client side runs on the SHIM6 host in Auckland. The failure scheduler decides when a failure should happen and which paths should be blocked.

LinShim6 [24][26] version 0.9.1 is installed to provide SHIM6 functionality for hosts. LinShim6 is an implementation of SHIM6 protocol for the Linux operating system, developed in the Université Catholique de Louvain. As this version of LinShim6 has been integrated in Linux kernel 2.6.27, hosts have this kernel installed. Lab Router runs kernel 2.6.31. We have developed a client/server program which runs on SHIM6 hosts and is able to generate a stream of 1000, 250-byte packets per second (~250 kB/s) uni-directional traffic using TCP and DCCP protocols. We used this program in our experiments as the application layer program.

Source Address Dependent Routing (SADR) is a prerequisite for SHIM6. SADR is a mechanism for forwarding packets according to their source addresses. When PA addresses are used, providers expect the packets received from their customers to carry their assigned address prefixes as source addresses. Without SADR, some packets might be forwarded to a wrong service provider and dropped [50]. Unfortunately, we could not convince the university network administrators to enable SADR on our university edge router. To run the experiments, they agreed to add static routes to the university edge router’s routing table to forward packets destined to the host in Dublin through different providers according to their destination addresses. Fig. 4.2 shows the two paths which were actually used in the experiments. Since the University edge router forwards the packets according to their destination address, only two address pairs, out of four possible address pairs could work. Packets which use the other address pairs will be dropped by the Internet providers because of ingress filtering [50]. To resolve this issue, we have changed LinShim6 to shuffle address pairs before starting the exploration process and put the working address pair in a random location in the list of address pairs. Therefore, the working address pair could appear in any location in the list and create different recovery cases. The SHIM6 host in Dublin was directly connected to two IPv6 service providers and had SADR enabled.
RTT for the red path: 307 ms  RTT for the blue path: 300 ms

Figure 4.2: Testbed for experiments with SHIM6 over the Internet between Auckland and Dublin. SADR is not enabled on the university edge router, so only two paths are available.

Each experiment starts with running the server side of the application on one SHIM6 host and running the client side on another one. The client connects to a server on one of the available paths and sends 256-byte packets to the server in an infinite loop. The server receives the packets and simply discards them. When communication reaches a stable level, the failure scheduler blocks the current path to simulate a failure. REAP detects the failure and starts its exploration process to find a new working address pair. When the process is complete and a new path is found, the application is notified to continue the data transfer. The failure scheduler waits for a while to let the application recover and restore the throughput. In our experiments, 120 seconds delay at the beginning of the experiment and also between a recovery and its next failure was sufficient for the application to bring the throughput back to 250 kB/s. Then it blocks the current path which pushes the REAP to move the communication back to the first path. After each recovery REAP exploration time, the number of sent and received probes and the application recovery time are stored in a log file using logging facilities in LinShim6. Application recovery time is defined as

\[
\text{Arrival time of the first data packet on the new path} - \text{Arrival time of the last data packet on the old path.}
\]

Since our application generates uni-directional traffic from client to server, REAP exploration time and number of sent and received probes are measured at the client side (sender) and application recovery time is measured at the server side (receiver). The
client sends packets at a fixed rate, so it always has something to send to the server. As a result, the client first starts using the new path when a recovery is completed. According to our definition of application recovery time, we get more precise values when we measure this parameter at the server side. We measure other parameters at client side, because the client starts the exploration sooner than or almost at the same time as the receiver. We will explain more about this in 4.5.2.

The client side runs in Ireland and the server side runs in Auckland. We have decided to run the server side in Auckland because the lab router, which generates the failures, is the last hop before the SHIM6 host in Auckland. Therefore, when a failure is generated, we can be sure that the server does not receive any packet after the failure, which makes measurement of the application recovery time more precise.

4.5.2 Results from Internet Experiments

Fig. 4.3 shows the results from two experiments, one with TCP and another one with DCCP. Each experiment has run the scenario, described in 4.5.1, for 250 failures. Each experiment took ~9 hours to finish. In all graphs, x-axis shows the experiment number. REAP exploration time and application recovery time are shown in milliseconds. The title of each graph shows the name of the transport protocol and the measured parameter. Mean and standard error are shown on the top right corner of each chart.

Comparing results shows that when DCCP is employed, REAP exploration time, number of sent probes and application recovery time are bigger than when TCP is used. The number of received probes is always 1 in case of DCCP, while it changes between 1 and 2 when TCP is used as transport layer protocol. To explain this, we need to look into REAP in more detail.
**EXPL**: REAP Exploration Time  **SP**: Sent Probes  **RP**: Received Probes  
**ART**: Application Recovery Time (Arrival time of the first data packet on the new path - Arrival time of the last data packet on the old path)

Figure 4.3: Results from the experiments with DCCP and TCP over the Internet with four address pairs. The x-axis shows the experiment number and the y-axis shows the value for the measured parameter. Times are in milliseconds.

When send timer expires, REAP sends an exploration probe and changes its state from operational to exploring. In this state, it keeps sending probe exploration packets until it receives something from the other side. If it receives a probe operational or probe inbound-OK, it means that a new operational address pair has been found and therefore the recovery is complete. Then it returns to state operational. Therefore if the number
of received probes for one side of the experiment is one, it means that the exploration process has been started by that side. In our results, the number of received probes for the client side is always one, which means that in the DCCP experiments, the SHIM6 host, which runs the client side of the application, always starts the exploration process. In TCP experiments, the value of RP (number of received probes) changes between 1 and 2 which means that in some runs the other side has started the exploration process. This is the result of a difference in the behaviour of TCP and DCCP.

DCCP employs delayed acknowledgements and sends an acknowledgement every RTT, while TCP sends acknowledgements faster; for every two packets (<12 ms) in our experiments (see Fig. 4.4 and Fig. 4.5). As mentioned before, our application generates unidirectional traffic which goes from client to the server in both sets of experiments for TCP and DCCP. TCP sends an acknowledgement for every two packets, so from REAP’s viewpoint the communication is bidirectional. DCCP suspends sending acknowledgements, so from REAP’s viewpoint it looks like a unidirectional communication. In the case of unidirectional traffic, send timer is activated only at the sender’s side (the application’s client side in our experiments). As a result, REAP at the client side always detects the failure and starts the exploration process. The receiver detects failure when it receives an exploration probe from the sender and then starts the exploration process.

Figure 4.4: Packets traveling between Dublin and Auckland in the TCP experiment. ACKs are sent for every two packets.
Figure 4.5: Packets traveling between Dublin and Auckland in the DCCP experiment. ACKs are sent every RTT.

Fig. 4.6 shows a trace of packets around a failure event in the DCCP experiment. The SHIM6 host in Dublin is the client and plays the role of sender in the experiment. Packets are captured on the SHIM6 host in Auckland which is running the server and plays the role of receiver. Packets which are unrelated to the experiment are blurred in the figure. Three instances of SHIM6 keep-alive messages are observed in the trace. SHIM6 sends keep-alive messages when it receives something from the other side but the receiving application/transport layer does not send anything in response. DCCP employs delayed acknowledgement, so SHIM6 has to send keep-alive messages to tell the other side of the communication that this side is alive. First a SHIM6 probe is received from Ireland host (packet 22278) which means that the Ireland host has detected the failure and started the exploration process. This packet triggers the exploration process at Auckland. A second SHIM6 probe (packet 22279) is issued by the Auckland host right after receiving the probe from Ireland. In other words, expiration of the send-timer starts the exploration process at sender. The receiver starts exploration process when it receives the first probe from the sender.

Fig. 4.7 shows a trace of packets around a failure event in the TCP experiment. Similar to the previous trace, the SHIM6 host in Dublin is the sender. Packets which are unrelated to the experiment are blurred in the figure. In this trace, no instance of keep-alive is observed. TCP sends acknowledgements faster which eliminates the need to sending keep-alive messages. The first SHIM6 probe (packet 3884) is issued by the Auckland host. This is the result of send timer expiration since no probe has been received from Ireland before that. The second probe is received from the Ireland host (packet 3885) about 323 ms after the first probe. The probe reports that it was the second probe which was sent by the Ireland host, which means that the Ireland
host had started its exploration process at least 500 ms (time interval between initial probes), before the Auckland host. In other words, expiration of the send timer at both sides is the trigger for the exploration process.

Figure 4.6: Trace of packets around a failure event in the DCCP experiment over the Internet.

Figure 4.7: Trace of packets around a failure event in the TCP experiment over the Internet.
As a result, in case of DCCP, the time for the exploration process is the sum of the time to reach the working address pair at the sender side plus the time to reach the working address pair at the receiver side. For TCP, sending acknowledgements causes the send timer to be activated at the receiver side as well, so the failure might be detected by either side. However the send timer at the other side will also expire soon, so the exploration process at both sides is started with a small time difference. In other words, sender and receiver perform the exploration process in parallel. In this case, the exploration time would be equal to the maximum of exploration time at sender and receiver sides. As a result, the exploration time, and therefore the number of sent probes and application recovery time for DCCP, is greater than for TCP. That means that the same application can make REAP behave differently by using different transport layer protocols. In other words, not only the behaviour of the application but also the behaviour of the transport layer protocol can affect the behaviour of the REAP, recovery time and generated traffic.

By looking carefully in the results, we can observe some signs of probe loss. Each host has two IP addresses from two different providers, so there are four address pairs in the REAP candidate list for each run. The first four probes are sent with a time interval equal to 500 ms, so scanning the whole list will take 1500 ms for the sender. In the worst case, when the working address is the last one in the list and communication is unidirectional, receiver receives the fourth probe 1500+RTT/2 ms after exploration starts by the sender. Then the receiver starts exploration. In the worst case, the working pair is the last one on the receiver’s candidate list, which causes the sender to receive the fourth probe 1500+RTT/2 ms after sending the fourth probe. The sender starts doubling the time interval between sending probes for the second scan of the candidate list. RTT for the paths in the experiments are 300 and 307 ms, so the time between sending the fourth probe and receiving the fourth probe from the receiver, which is the worst case, is around 1800 ms. This time is sufficient for sending only one more probe from the sender. Taking the probe operational which is sent after receiving the fourth probe from the receiver into account, the maximum number of sending probes would be six. Therefore, any run with more than six sent probes can potentially be a sign of probe loss. Our DCCP results show only one instance of this situation, but more instances can be observed in our TCP results.

4.6 Lab Experiments

We have set up a test environment in the lab to do similar experiments using LinShim6 with more than 4 address pairs. This section describes the hardware and software configuration for these experiments. It also presents and discusses the results. In the
Lab, we could run experiments with 4, 9 and 16 address pairs to study the behaviour of SHIM6 when the number of address pairs are increased. The environment is also different from the Internet experiments; i.e. RTT and loss rate are smaller. We use the results of these experiments and also Internet experiments for validating our simulation model as well (see section 4.9). The rest of this section describes the environment for the experiments and discusses the results.

### 4.6.1 Experiment Setup

Fig. 4.8 shows the hardware configuration for the test environment in the lab. Host 1 and Host 2 are configured as two SHIM6-enabled multi-addressed hosts. Each host is equipped with four network interface cards. Two IPv6 routers, similar to the lab router in the Internet experiments, connect these hosts together. To simulate a failure, one instance of *failure scheduler* server runs on each router. The client side runs on one of the SHIM6 hosts and schedules failures, then asks the server instances to simulate failures by dropping packets on the specified paths. There is always only one path open and all the other paths are blocked. This configuration enabled us to run experiments with 4, 9 and 16 address pairs to get a clearer view of the behaviour of SHIM6 in the real environment.

We have used the same application program as in our Internet experiments to generate the required traffic. Our experiments also follow the same scenario as for Internet experiments. All hosts and routers use Ubuntu Linux version 9.10. LinShim6 version 0.9.1 has been installed to provide SHIM6 functionality for hosts. Since this version of LinShim6 has been integrated in Linux kernel 2.6.27, hosts have this kernel installed.
Routers run Linux kernel 2.6.31.

4.6.2 Results from Lab Experiments

Fig. 4.9 and 4.10 show the results from four experiments in the lab, two with TCP and two with DCCP, with four and nine address pairs. Each experiment has run the scenario described in 4.6.1, for 250 failures. Experiments with four address pairs took ~9 hours and experiments with nine address pairs took ~11 hours to finish. In all graphs, the x-axis shows the experiment number. REAP exploration time and application recovery time are shown in milliseconds. The title of each graph is a triple in the format: <transport-protocol> - <measured-parameter> - <number-of-address-pairs>. Mean and standard error are shown on the top right corner of each chart. Experiments with 16 address pairs fail when the working address pair is located at or close to the end of the REAP candidate list. REAP employs exponential backoff after sending initial probes to avoid generating large bursts of traffic during exploration which leads to long exploration time when there is a long candidate list. For 16 address pairs, this delay causes the connection to time out and stop the experiment. In some cases, SHIM6 removes the context without finding the new address pair. We believe that packet losses cause the exploration process to go to the second round of exploration which causes much longer delays and makes the SHIM6 decide to stop exploration and remove the context. We only include results from the experiments with four and nine address pairs in this section.
**EXPL:** REAP Exploration Time  **SP:** Sent Probes  **RP:** Received Probes  **AP:** Address Pairs  
**ART:** Application Recovery Time (Arrival time of the first data packet on the new path - Arrival time of the last data packet on the old path)

Figure 4.9: Results from lab experiments for TCP and DCCP with four address pairs. The x-axis shows the experiment number and the y-axis shows the value for the measured parameter. Times are in milliseconds.
EXPL: REAP Exploration Time  SP: Sent Probes  RP: Received Probes  AP: Address Pairs
ART: Application Recovery Time (Arrival time of the first data packet on the new path - Arrival time of the last data packet on the old path)

Figure 4.10: Results from lab experiments for TCP and DCCP with nine address pairs. The x-axis shows the experiment number and the y-axis shows the value for the measured parameter. Times are in milliseconds.

DCCP results show a different behaviour from Internet experiments. In the lab, TCP and DCCP show a similar behaviour. This is because RTT in the lab experiments is small (0.3 ms), so DCCP sends acknowledgements faster than in the Internet
experiments. In this case, from REAP’s viewpoint, both TCP and DCCP communications are bi-directional. The results show that RTT is important and can affect the behaviour of REAP. Unfortunately, we could not find a research partner in a country closer than Ireland so that we could run experiments on paths with an RTT lower than Ireland and higher than lab experiments. In fact, we tried two extreme situations in our experiments. Trying something in between might lead to other interesting results.

Probe losses are observable in the lab experiments too. Calculations, similar to those in section 4.5.2, show that it can be considered as a sign of probe loss if the number of sent probes exceeds six, in case of four address pairs, or exceeds 10, in case of nine address pairs. Probe loss causes REAP to go to the second round for scanning the list of address pairs, which leads to sending more probes and also longer exploration time.

Probe losses are the main reason for difference between average values for DCCP and TCP experiments. By removing the runs with probe loss from the results and calculating the average values again, we get similar values for TCP and DCCP experiments in the lab (see 4.9).

4.7 Operational Issues

Running experiments in the Lab and over the Internet uncovered some issues for the practical use of SHIM6:

- Firewall: SHIM6 adds its own extension header to the packets. Some firewalls drop IPv6 packets if they do not recognize their extension headers. We experienced this problem in the Internet experiments. This is a general issue and is not specific to SHIM6. RFC 7045 confirms existence of this issue: “Unfortunately, it is an established fact that several widely used firewalls do not recognise some or all of the extension headers standardised since RFC 2460 was published” [32]. A solution is also proposed in the RFC to address the issue.

- Filter precedence: This is a LinShim6 specific issue. *iptables* filters process packets before Linshim6. It means that *iptables* filters can only see ULIDs and not real source and destination addresses. So, if SHIM6 changes source/destination addresses in the packets, iptables filters are not able to recognize them.

- New path employment: When a new path is discovered, although the transport layer is notified, in some cases it does not start using the new path; so the application stays in wait state. We found that producing some traffic on the new
path, e.g. by sending some ICMP packets, can resolve this issue, but it still has its negative impact on the application recovery time.

- Source Address Dependent Routing (SADR): SADR [67] is necessary for effective use of SHIM6. SADR needs routers to keep one routing table per source address prefix in their memory. It also imposes more processing by the routers to make forwarding decisions. Because of these issues, we found it difficult to convince Auckland university network administrators to enable this functionality on the edge routers. Without SADR, packets are forwarded based on their destination address only. As a result, some REAP probes might be forwarded to a wrong service provider and dropped because of ingress filtering [50]. This issue also is not specific to SHIM6. Every multihomed site with multi-addressed hosts will need a mechanism to deal with ingress filtering.

4.8 Model Description

We have built a model of a SHIM6-enabled host using Stochastic Activity Networks (SAN) [85]. The Möbius modeling tool [41] has been used for simulating this model. We use this model to evaluate the performance of REAP in large scale networks. The model is also used for evaluating our proposed improvements to REAP and TFRC.

Five components are included in the model: TCP congestion control mechanism, TFRC congestion control mechanism, application, SHIM6 and REAP. They are organised as three sub-models: REAP, SHIM6 and application. The sub-models are described, in detail, in the rest of this section.

4.8.1 REAP Sub-model

Fig. 4.8.1 shows graphical representations of the REAP sub-model. Activity Failure generates a failure at a specific time. Initial marking of place $p1$ is 1, so only one failure is generated in each run. Gate Failure_out_gate records the current time, as the failure time, and sets the marking of place path_fail to 1. path_fail is shared between REAP and application sub-models. Changing the marking of path_fail informs the application sub-model about the failure. Three different states of REAP are represented by three places in the model: operational, exploring, InBoundOK. Initial marking of operational is 1 which means that the system is in operational state at the start of simulation.

Send timer is modelled by activity send_timer. Place ST_start is shared between REAP and application sub-model. Whenever a packet is sent by application sub-model, it resets the send timer if it is already running; otherwise it starts the timer by setting the
marking of $ST_{\text{start}}$ to 1. To enable the application sub-model to reset the send timer, a reactivation predicate has been defined for $send_{\text{timer}}$. $counters$ is an extended place which contains a set of variables which are used in the model. To avoid defining a set of places for this purpose, we have defined a structure containing all required variables for this extended place. One of those variables ($restart_{\text{send}}$) is shared between REAP and application sub-models and used as an indicator for reactivating (resetting) $send_{\text{timer}}$. When send timer expires, gate $send_{\text{out}}$ sets the marking of $ST_{\text{timeout}}$ to 1.

Failure_Detect gets enabled in the state operational when send timer expires. Gate $send_{\text{timeout}}$ changes the state to exploring by removing the token from operational and adding it to place exploring, and then activating another part of the model to send exploration probes by setting the marking of $probe_{\text{start}}$ to 1.

Activity $probe_{\text{timer}}$ and its associated gates and places simulate sending probes during the exploration process. Initial marking of $probe_{\text{start}}$ is 0. Gate $probe_{\text{in}}$ enables $probe_{\text{timer}}$ if there is a token in $probe_{\text{start}}$ and the system is in state exploring or InBoundOK. The enabling time of $probe_{\text{timer}}$ is a variable with initial value 500 ms. Gate $probe_{\text{out}}$ calculates a new enabling time for $probe_{\text{timer}}$ when a probe is sent. REAP employs exponential backoff for sending probes to avoid a signaling storm. The number of tokens in $probe_{\text{in}}$ shows the number of sent probes.

Activity $ex_{\text{recv}}$ and its associated places and gates model the REAP exploration process in state exploring. Output gates connected to $ex_{\text{recv}}$ perform required processing upon receiving different types of packets from the peer. Output cases are randomly selected to simulate different events. Activity $ibok_{\text{recv}}$ and its associated places and gates model the REAP exploration process in state IBOK.

REAP’s keep-alive timer is modelled by activities $KA_{\text{timer}}$ and $KAI_{\text{timer}}$.

### 4.8.2 Application Sub-model

Fig. 4.8.2 shows graphical representation of the application sub-model. This sub-model consists of three main parts:

1. Sender application which runs over DCCP. This part models an application which uses DCCP to communicate to another SHIM6-enabled host. This application generates 250-byte packets which are sent via DCCP. Congestion is controlled by the TFRC algorithm.

2. Sender application which runs over TCP. This part models an application which uses TCP to communicate to another SHIM6-enabled host. This application generates 250-byte packets which are sent via TCP. Congestion is controlled using TCP’s congestion control algorithm.
Figure 4.11: Graphical representation of the REAP sub-model.
3. The part that connects this sub-model to the REAP sub-model and receives and applies failure/recovery notifications.

Type of application is one of the model parameters. The initial marking of place start is 1, so start_act is enabled and fires right after starting simulation. Gate start_actions initializes the model variables according to the model parameters. Depending on the user’s choice, one of the applications will be activated. Extended place tcp_counter contains a set of variables which are used in this sub-model. There are some other places, on the left side of the model representation, which are not connected to any activity or gate. They are shared places which connect this sub-model to other sub-models. As described in section 3.4, local and shared places are accessible by all components in the sub-model and there is no need to have a connection in the graphical representation.

4.8.2.1 DCCP Application

Activity app_send is enabled if the model parameters are set to use the DCCP application. app_send simulates an application that generates packets at the specified rate. TFRC is a rate-based mechanism, so when a packet is generated by the application it might have to wait in a queue. Gate tfrc_send_now enables activity send_now if the current sending rate allows sending this packet. send_now is an instantaneous activity, so it fires and sends the packet immediately. Otherwise send_later is activated and its enabling time is set to the time that the next packet is allowed to be sent. In any case, when send_now or send_later is fired, the sent packet is added to the place sent_udp as a token. The REAP send timer is also restarted.

Activity feedback models receiving feedback packets from the peer. fb_in_gate checks whether a failure has already occurred in the path. If a failure has occurred, then it stops receiving feedback packets by disabling feedback. Gate fb_in_gate_2 checks for the recovery to become complete. It enables feedback when it receives recovery complete notification from REAP. Gate udp_fb_rcv calculates a new sending rate upon receiving a new feedback packet using the formula described in section 4.4.2.

Activity nofb_timer models TFRC’s no-feedback timer. The enabling time of this activity is calculated in gate udp_fb_rcv whenever a new feedback is received. When a failure occurs, nofb_timer fires and gate nofb_timer_timeout calculates a new sending rate and a new value for the no_feedback timer. This timer and REAP’s keep-alive timer are reset by udp_fb_rcv whenever a feedback is received.

4.8.2.2 TCP Application

This part of the model simulates an application which sends packets to a remote host through TCP. It assumes that a TCP connection has already been established, so it
Figure 4.12: Graphical representation of the application sub-model.
only models data transfer over an established connection. Activity *Send* generates the application load. Every firing of this activity simulates sending a packet from an application to the TCP transport layer. A new packet is sent out if there is space in TCP’s congestion window. *sent_segment* plays the role of the send buffer and gate *send_ok* checks to make sure the marking of *sent_segment* does not exceed the current congestion window size. Activity *wait* models receiving acknowledgements from the peer. Gate *rcv_ack* calculates new congestion window size and retransmission timer values when an acknowledgement is received. Similar to the DCCP application, activity *wait_enabled* and its associated gates disable/enable *wait* in case of failure/recovery.

Activity *ret_timer* and its associated gates and places model TCP’s retransmission timer. When this activity fires, congestion window drops to one and retransmission time is doubled.

### 4.8.2.3 Failure/Recovery Detection

This part of the model receives failure/recovery notifications from REAP and applies them to the application. *path_fail* is a shared place with the REAP sub-model. REAP sets the marking of this place to one when a failure occurs. Gate *failure_in_gate* disables *wait* and *feedback* and records the failure time. *in_explore* is another shared place with REAP. REAP sets the marking of this place to one when the exploration process starts. Activities *failure* and *failure_detect* capture the occurrence of failure, and detect REAP failure, respectively. Activity *recovered* models receiving recovery notification from REAP. When REAP returns to the state *operational*, it means that the failure has been recovered. This time is recorded as the recovery completion time. Then gate *recovered_out_gate* enables *wait* and *feedback*.

### 4.8.3 SHIM6 Sub-model

Fig. 4.13 shows a graphical representation of the SHIM6 sub-model. It models the SHIM6 context establishment process. We do not describe this sub-model in detail, since it does not have any major effect on the experiments in the rest of this chapter. However, it will be useful for future studies.

### 4.8.4 Limitations

Communication links are assumed to be reliable and packets are lost only when a failure occurs. If a probe is lost, it might cause the exploration process to start the second round of reachability checks for address pairs, which leads to a longer exploration time. To simplify the model, we do not consider probe losses and assume that all SHIM6
Figure 4.13: Graphical representation of the SHIM6 sub-model.
contexts are recoverable and that all of them are recovered in the first round of the exploration process. Results from our experiments in the lab and over the Internet show that probe loss occurs but the frequency is low.

The model does not consider receiver buffer overflow. We assume that there is always enough space to accommodate receiving data. Buffer overflows are handled by flow control mechanisms. The focus of our study is link failure and flow control has no direct effect on that.

We are interested only in the events that occur during link failure and recovery. Link failures are detected by expiration of the retransmission timer, not by duplicate ACKs. Duplicate ACKs can not be received on a failed link, therefore we have only included slow start and congestion avoidance modes in the TCP model.

We do not consider the congestion which might be caused by the REAP traffic itself. Results in section 4.10 show that the traffic generated by REAP is small compared to normal traffic, even for a large site.

4.9 Validating the Model

To verify the accuracy of our model, we have run some experiments with the same parameters as experiments described in sections 4.5 and 4.6. We have modelled an application which generates 1000, 250-byte packets per second. The SHIM6 context is generated right after a communication starts. In other words, the first packet exchanged between hosts triggers SHIM6 to create the context. Failure time is a model parameter and can be set to different values. In these experiments, we set it to 120 seconds, so a failure is generated 120 seconds after the simulation starts. It is assumed that DCCP employs TCP Friendly Rate Control (TFRC) for congestion control and TCP uses the TCP congestion control algorithm. The REAP send timer is set to 10 seconds.

Tables 4.1, 4.2 and 4.3 show the results. Each column shows the value of the measured parameter and also its error interval. Our model does not consider packet losses, so we have not considered the runs from lab and Internet experiments, which were affected by packet loss for calculating average values.
Table 4.1: Results from experiments over the Internet and simulation with four address pairs.

<table>
<thead>
<tr>
<th></th>
<th>EXPL (ms)</th>
<th>SP</th>
<th>RP</th>
<th>ART (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td>1264 ± 33</td>
<td>3.54 ± 0.06</td>
<td>1.54 ± 0.03</td>
<td>12099 ± 52</td>
</tr>
<tr>
<td>Simulation</td>
<td>1397 ± 53</td>
<td>3.8 ± 0.13</td>
<td>1.352 ± 0.05</td>
<td>12554 ± 54</td>
</tr>
</tbody>
</table>

**EXPL**: REAP exploration time  
**SP**: Number of sent probes  
**RP**: Number of received probes  
**ART**: Application recovery time

Table 4.2: Results from experiments in the Lab and simulation with four address pairs.

<table>
<thead>
<tr>
<th></th>
<th>EXPL (ms)</th>
<th>SP</th>
<th>RP</th>
<th>ART (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab</td>
<td>780.72 ± 29</td>
<td>3.484 ± 0.05</td>
<td>1.448 ± 0.03</td>
<td>10894 ± 29</td>
</tr>
<tr>
<td>Simulation</td>
<td>992 ± 50</td>
<td>3.46 ± 0.18</td>
<td>1.36 ± 0.05</td>
<td>11479 ± 61</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>EXPL (ms)</th>
<th>SP</th>
<th>RP</th>
<th>ART (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab</td>
<td>798 ± 28</td>
<td>3.188 ± 0.06</td>
<td>1.504 ± 0.03</td>
<td>13829 ± 27</td>
</tr>
<tr>
<td>Simulation</td>
<td>992 ± 50</td>
<td>3.48 ± 0.11</td>
<td>1.34 ± 0.06</td>
<td>12817 ± 27</td>
</tr>
</tbody>
</table>

Table 4.3: Results from experiments in the Lab and simulation with nine address pairs.

<table>
<thead>
<tr>
<th></th>
<th>EXPL (ms)</th>
<th>SP</th>
<th>RP</th>
<th>ART (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab</td>
<td>8391 ± 544</td>
<td>6.448 ± 0.13</td>
<td>1.452 ± 0.03</td>
<td>18803 ± 546</td>
</tr>
<tr>
<td>Simulation</td>
<td>9996 ± 1215</td>
<td>6.76 ± 0.24</td>
<td>1.33 ± 0.06</td>
<td>21003 ± 1215</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>EXPL (ms)</th>
<th>SP</th>
<th>RP</th>
<th>ART (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab</td>
<td>8302 ± 549</td>
<td>6.252 ± 0.135</td>
<td>1.516 ± 0.03</td>
<td>23452 ± 893</td>
</tr>
<tr>
<td>Simulation</td>
<td>9996 ± 1215</td>
<td>6.736 ± 0.24</td>
<td>1.27 ± 0.05</td>
<td>24688 ± 1757</td>
</tr>
</tbody>
</table>

As shown in the tables, the results from the model are close to the results from real experiments. We consider this as a strong indication that our simulation model is consistent with the experimental results.

Results of a performance study on REAP have been presented in [38]. The authors built their own simulation model and ran an experiment to measure recovery time for a TCP application with and without applying their proposed signaling mechanism. The mechanism proposes a signal to be sent to TCP when REAP recovers the failure,
so TCP can start using the new path right away and does not need to wait for its retransmission timer to expire. We implemented the mechanism in our model to be able to use the results from their work as another reference to validate our model. We set the model parameters to the same values and ran a similar experiment with our model. Fig. 4.14 shows the results from [38] (Fig. 3) and also from our model. These results are close too.

![Figure 4.14: Results from [38] (Fig. 3) and a similar experiment with our model.](image)

**4.10 REAP Performance in Large Scale Networks**

In this section\(^3\) we present and analyze the results of a number of simulation experiments with a large number of REAP instances. The aim of the experiments is to see how REAP reacts to path failures in a large SHIM6-enabled multihomed network. These experiments are too big to run in the real world, so we have used our simulation model of REAP, which was described and validated in sections 4.8 and 4.9. We have focused on REAP recovery time and traffic as two important performance parameters. REAP recovery time is the time that REAP takes to detect the failure and find a new working address pair. REAP traffic is the traffic which is generated by REAP during its exploration process. Results from experiments in this section give us an idea about the effectiveness of REAP if it were to be in a large SHIM6-enabled computer site. We also propose some enhancements and show their effectiveness by using our simulation model.

We have executed the model with 10,000 instances of its REAP sub-model. It can be considered, for example, as a SHIM6-enabled site with 1000 hosts where each host has 10 connections to 10 different remote hosts that should be recovered when a failure occurs.

\(^3\)This section has been derived from [89].
affects the site.

We assume that SHIM6 contexts detect the failure between 0 and 10 seconds (value of the send timer) after it occurs. This is a realistic assumption, because different contexts in a site do not detect failure at the same time - it depends on the time that a packet has been sent out through a SHIM6 context. It is also possible that the failure is detected by the remote SHIM6 host. The maximum time for detecting a failure is equal to the value of send timer. A new operational address pair is selected randomly and different address pairs have an equal chance to be selected.

We do not consider the congestion which might be caused by the REAP traffic itself, because our results show that the traffic generated by REAP is small compared to normal traffic, even for a large site, and can be ignored. For example, in the case of 25 address pairs, about 4800 probes per second are sent during the first 10 seconds of the exploration process which is the peak of the traffic (more results are presented in section 4.10.2). Every probe in the first 10 seconds carries at most seven address pairs; four initial address pairs and three more after employing exponential backoff. Thus, the average probe size in the first 10 seconds is 232 bytes; each probe needs 72 bytes for the fixed part and 40 bytes for each address pair. As a result, a load of 4800 probes per second does not occupy more than one MB/s of the site’s available link capacity. Large sites usually have high bandwidth links to the Internet and this amount of traffic does not cause a major problem for them. Our model assumes that all contexts are recoverable, which means at least one operational address pair exists in the list. If there is no operational address pair available, then the recovery is not possible and the results do not matter. We also assume that all contexts will be recovered in the first iteration when REAP scans the list of address pairs.

4.10.1 REAP Recovery Time

We have measured average and total REAP recovery time for different numbers of address pairs for 10000 instances of REAP. We define total REAP recovery time as the recovery time for the whole site, i.e., the time between failure occurrence and recovering the last context. In other words, it shows the recovery time for the last context that is recovered. The average recovery time is calculated by dividing the sum of recovery times for REAP instances by the number of REAP instances. Fig. 4.15 shows the average and total recovery time for the experiments with 4, 6, 9, 12, 16, 20 and 25 address pairs.
CHAPTER 4. SHIM6

The average and total recovery time increases when the number of address pairs is increased. The correlation is not linear because REAP uses an exponential backoff algorithm for increasing the time interval between probes. The graph demonstrates that REAP will show poor performance when the number of address pairs exceeds 9. It should be noted that recovery time includes failure detection and address exploration times. In our model of the Internet, nine address pairs seems to be the worst case but we have included larger numbers in our experiments to obtain a clearer view of REAP’s behaviour.

4.10.2 REAP Traffic

We have measured the average and total number of probes sent during the address exploration process in the experiments. Fig. 4.16 shows the total number of probes sent during the recovery process. Fig. 4.17 shows the number of probes sent in the first ten minutes of the exploration process. Fig. 4.16 shows that there is a linear correlation between number of address pairs and number of sent probes. Fig. 4.17 shows that a large amount of probes are sent at the start of exploration. For example, in the case of four address pairs, 93% of the probes, and in the case of 25 address pairs 34% of probes, are sent during the first 10 seconds. There are some intervals when very few probes were sent. This can be seen more clearly for the case of 16 and 25 address pairs. It means that for some SHIM6 contexts the time interval between probes is large because of the exponential backoff, so REAP instances have to wait for a long time before probing the next address pair. Some connections might be dropped by the transport or application layer before REAP can recover them. Fig. 4.18 shows the number of
such connections. We used 300 seconds as a typical value for upper layer connection time-out in our experiments. Different applications use different time-out values; e.g. 22 seconds for IE 6.x on WinXP, 224 seconds for FireFox 1.5 on Solaris 1, 149 seconds for Safari 2.0.3 on MacOS X Tiger\(^4\). We considered 300 seconds as an upper limit and assumed that longer recovery time is unacceptable for most applications.

![Graph showing correlation between number of address pairs and number of sent probes.](http://www.nanog.org/meetings/nanog39/presentations/ipv6_katsuyasu.pdf)

Figure 4.16: Total number of sent probes during recovery for different numbers of address pairs for 10000 instances of REAP. It shows a linear correlation between number of address pairs and number of sent probes.

As Fig. 4.18 shows, the number of such connections grows when the number of address pairs is increased. Our experiments show that REAP can probe nine address pairs safely, but when the number of address pairs is more than that the site may experience some failures. As can be seen in the graph, in the case of 25 address pairs, more than 50% of contexts might not be able to recover before the upper layer protocol time-out.

\(^4\)http://www.nanog.org/meetings/nanog39/presentations/ipv6_katsuyasu.pdf
4.10.3 Effect of the REAP Parameters on Performance

In this section, we investigate the effect of the send timer and initial probes on recovery time and traffic in a large scale network.
4.10.3.1 Send Timer

We also conduct some experiments with different values for the send timer to see the effect of small and large values on the performance parameters. Fig. 4.19 shows the results of executing the model with 16 address pairs for different values of send timer (2, 4, 6, 8, 10, 12, 14 s). The results show that the effect of this timer on total recovery time is linear, but the effect is not significant. Although choosing small values for this timer leads to a better performance, it increases the chance of wrong behaviour in failure detection.

![Figure 4.19: Effect of send timer on recovery time and traffic in case of 16 address pairs. The results show that the effect of send timer on recovery time and traffic is not significant.](image)

4.10.3.2 Initial Probes

REAP sends four initial probes to check for the first four address pairs in the list of address pairs and then continues with an exponential backoff algorithm. We showed in the previous section that this algorithm may cause long delays during the exploration process leading to long recovery times and, therefore, some connections will be lost before REAP can recover them. We tried two different scenarios to resolve this issue:

- **Increasing the number of initial probes:** The default value for number of initial probes is 4. REAP sends four probes for the first four address pairs in the list and then starts exponential backoff. We tried different values for this parameter starting from 4 in case of 9 address pairs. Fig. 4.20 shows the effect
of this modification on recovery time and traffic. Increasing the number of initial probes has a significant effect on recovery time. There is a minor change in total number of sent probes, but the traffic at the start of the recovery phase is increased. For example, increasing number of initial probes from four to five will cause about 6.5% increase in traffic in the first 10 seconds of the recovery process, 22% decrease in average recovery time and 34% decrease in total recovery time. That suggests that it can reduce the number of failed connections too. We tried the same experiment with 16 address pairs and a similar modification in number of initial probes. The results showed a 46% decrease in number of failed connections.

Figure 4.20: Effect of increasing the number of initial probes on recovery time and traffic in the case of nine address pairs. Increasing the number of initial probes has a significant effect on recovery time. There is a minor change in total number of sent probes but the traffic at the start of the recovery phase is increased.

- **Sending initial probes concurrently**: Initial probes are sent sequentially with
a time interval of 0.5 second. Removing this delay and sending initial probes at the same time decreases recovery time [127], but it definitely increases the traffic. To see the effect of this modification on performance, we have added this feature to the model and run some experiments. Fig. 4.21 shows recovery time and traffic for different number of address pairs after applying this modification to the model. The results show that in the case of nine address pairs, this modification will cause an 11% decrease in average recovery time, 4.5% decrease in total recovery time, and 8.2% increase in traffic compared to the default behaviour of the protocol (Fig. 4.15-4.18).
Figure 4.21: Effect of sending initial probes concurrently on recovery time and traffic. This shows improvement in recovery time.
4.11 TFRC Improvement

This section proposes and evaluates a mechanism which improves the performance of TFRC in SHIM6-enabled networks.

Receiving a marked packet in an ECN-capable connection, or expiration of its nofeedback timer notifies TFRC of congestion. To react to congestion, TFRC halves its sending rate, updates the nominated time for sending next packet and restarts the nofeedback timer. Subsequent expiration of this timer will reduce sending rate and increase inter-packet interval \( t_{ipi} \) and nominated time for sending the next packet \( t_{nom} \) using the following formula:

\[
\begin{align*}
    t_{ipi} &= \text{average packet size} / \text{sending rate} \\
    t_{nom} &= t_{nom} + t_{ipi}
\end{align*}
\]

Employing this policy, TFRC tries to adapt itself to the communication media conditions and avoid making the congestion worse. When a failure occurs, the current communication path cuts off and no packet is able to reach the other end of the communication. While the recovery process is in progress, the nofeedback timer will be expired several times depending on the time required for the process to be completed. When TFRC’s sending rate becomes less than the application’s sending rate, packets are scheduled to be sent at nominated times in the future. Therefore, it is unlikely that TFRC will start to send packets right after recovery. In other words, it takes some time for TFRC to detect completion of the recovery.

The effect of this behaviour can be seen in a SHIM6-enabled network in tables 4.1, 4.2 and 4.3 by comparing REAP exploration time and application recovery time. For example, the difference between exploration time and recovery time in the case of four address pairs in table 4.1, is 10.209 seconds. REAP’s send timer was set to 10 seconds in our experiments, so it takes at most 10 seconds for REAP to detect failure. The remainder, 0.209 seconds, is the time it takes for TFRC to detect recovery. As the REAP recovery time is increased, the recovery detection time for TFRC is also increased (5.150 seconds in case of nine address pairs).

We propose a signaling mechanism which notifies TFRC of completion of the recovery. Then TFRC can reset \( t_{nom} \), update the nominated sending time of the buffered packets according to that, and start using the new path immediately after receiving the notification. This will eliminate recovery detection time and reduce application recovery time. Applying this modification to the simulation model and running the experiment with nine address pairs showed an 18% improvement in ART.

When a loss occurs, the receiver reports it to the sender in the first feedback packet it sends after the loss by setting its loss event rate to a value greater than zero. That
causes the sender to use TFRC’s throughput equation (described in section 4.4.2) to calculate the sending rate. When SHIM6 is employed in the network, the recovery process usually takes a few seconds to be completed, e.g. more than 10 seconds in our experiments. Loss-event rate is calculated according to the amount of packet loss in the eight most recent loss event intervals. Since the length of each interval is equal to one RTT, the TFRC receiver reports the maximum value for this parameter, which is 1, to the sender in the first feedback packet after recovery. This value is interpreted by the sender as a serious congestion in the path. Setting $p$ to 1 in TFRC’s throughput equation leads to a small value for its sending rate. If we assume that the communication links are reliable, no packet will be lost after recovery and the next feedbacks will also report the same loss-event rate. As a result, the sender will keep its sending low for a while. Although this approach looks appropriate for a congested path, it does not seem like a suitable policy for a new path. In other words, it is not reasonable to use information about the old path to decide about the new path.

To resolve this issue, we propose that the receiver should clear its loss history when a notification signal about a new path is received from SHIM6. That will allow the sender to increase its sending rate quickly after recovery. Fig 4.22 shows the result of applying this improvement to our simulation model. Failure occurs at time 120,000. To avoid long delays in our experiments, described in sections 4.5 and 4.6, we used this technique and it worked well.

![TFRC Sending Rate (4 Address Pairs)](image)

Figure 4.22: Effect of clearing loss history, after recovery, on TFRC sending rate. The sender increases its sending rate quickly after recovery.
4.12 Transport to REAP Notification

This section proposes and evaluates a signaling mechanism which improves application recovery time in SHIM6-enabled networks.

Both TFRC and TCP employ timers to detect congestion in the communication path. TCP uses a retransmission timer and TFRC uses a nofeedback timer. Time-out value for these timers is calculated based on the round-trip time between sender and receiver. On the other hand, REAP also monitors the path and sets up send-timer to detect failures. When a failure occurs, it triggers both REAP and the transport layer’s congestion control mechanism, since the link cuts off and no packet is able to reach the other end of the communication.

Time-out values for TCP and TFRC timers are calculated based on round-trip time and usually set to a value less than or equal to one second while the default value for REAP’s send timer is 10 seconds. Therefore, in most cases, TCP and TFRC are able to detect failures faster than REAP.

We propose a notification mechanism which enables TCP and TFRC to notify REAP when they detect a failure. When REAP receives this notification, it can start exploring a new working address pair immediately, which will reduce recovery time. To avoid incorrect notifications and distinguish between congestion and failure, we propose that \( n \) consecutive expirations of retransmission/nofeedback timer is considered as a failure and used as a basis for sending notification. \( n \) can be chosen by the transport layer based on attributes of the communication line. Table 4.4 shows the effect of this improvement, with \( n = 3 \), on the results of simulation.

<table>
<thead>
<tr>
<th>4 Address Pairs</th>
<th>TCP</th>
<th>DCCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original (ms)</td>
<td>11479</td>
<td>12817</td>
</tr>
<tr>
<td>Improved (ms)</td>
<td>2997</td>
<td>1109</td>
</tr>
<tr>
<td>9 Address Pairs</td>
<td>TCP</td>
<td>DCCP</td>
</tr>
<tr>
<td>Original (ms)</td>
<td>21003</td>
<td>24688</td>
</tr>
<tr>
<td>Improved (ms)</td>
<td>12002</td>
<td>12245</td>
</tr>
</tbody>
</table>

Table 4.4: Effect of transport to REAP notification on application recovery time. 3 consecutive expiration of retransmission/nofeedback timer is considered as a failure.

It should be noted that this improvement does not eliminate the need for the send timer. Transport layer timers are only active when there is traffic on the line. REAP still needs send-timer to be able to detect failures when applications are idle and no packet is exchanged through the line or transport layer, e.g. UDP, does not employ a congestion control mechanism.
4.13 Conclusion

This chapter presents the results of a study on SHIM6. We have conducted some experiments in the lab and also on the Internet with LinShim6. These experiments show that not only behaviour of the application but also behaviour of the transport layer protocol can affect behaviour of the REAP and, therefore, recovery time and generated traffic. They also uncovered some issues for practical use of SHIM6: 1) some firewalls do not recognize and, therefore, drop SHIM6 packets and 2) it is hard to convince network administrators to enable SADR on the site’s edge routers, due to its performance penalties. None of these issues are specific to SHIM6. SHIM6 extension header is not the only header which is not recognized by firewalls. IPv6 packets which carry standard headers, like fragmentation headers, are also dropped by those firewalls. SADR is required to deal with ingress filtering. Every multihomed site with multi-addressed hosts need such a mechanism in place; otherwise packets might be dropped if they are forwarded to a wrong provider.

We have also built and used a simulation model of a SHIM6-enabled host using SAN to study the behaviour and performance of this protocol. We have conducted two sets of experiments for this purpose.

Our first set of experiments has focused on REAP. They have simulated the situation in which a failure occurs in a large IPv6 site. Two important performance parameters have been measured in these experiments: REAP recovery time and traffic. The simulation results show a reduction in performance when number of address pairs is increased, especially when it exceeds 9. It also shows that the traffic generated in the start of the recovery process can be large, but it should not cause a major problem for a large-scale network with a high bandwidth link. We have also investigated the effect of send timer and initial probes on the performance. The results show that the value of send timer does not have a significant effect on the performance, but deferring the exponential backoff algorithm (e.g. by increasing the number of initial probes) and sending initial probes in a burst can improve recovery time. These modifications increase traffic but their positive effect on the recovery time is more than their negative impact.

Our second set of experiments focused on the interaction between SHIM6 and transport layer protocols. We have chosen TCP and DCCP, which employ two different congestion control mechanisms. The focus of our study in these experiments has also been on traffic and recovery time, which directly affect performance. Results from DCCP experiments show that it takes some time for TFRC to detect recovery and start using a newly discovered path, which can increase application recovery time. To resolve this issue, we have proposed a signaling mechanism which enables REAP to notify TFRC when it discovers a new path. This mechanism eliminates the recovery detection time
and improves application recovery time. It also enables a receiver to reset its loss history to allow the sender to increase its sending rate quickly, over the new path. We have also proposed a mechanism which enables the transport layer to notify REAP about failures. That enables REAP to start its exploration process before send timer expires. Employing this mechanism can reduce failure detection time and improve application recovery time.

We evaluated the effect of above mentioned improvements by simulation. Implementing those improvements in LinShim6 and evaluating their effect in real experiments are left as areas for future work.
# Chapter 5

## MPTCP

### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>96</td>
</tr>
<tr>
<td>5.2</td>
<td>Related Work</td>
<td>97</td>
</tr>
<tr>
<td>5.3</td>
<td>SEMICOUPLED ($\phi=1$) Congestion Control Algorithm</td>
<td>98</td>
</tr>
<tr>
<td>5.4</td>
<td>Model Description</td>
<td>99</td>
</tr>
<tr>
<td>5.5</td>
<td>Validating the Model</td>
<td>104</td>
</tr>
<tr>
<td>5.6</td>
<td>Increasing Number of Subflows</td>
<td>106</td>
</tr>
<tr>
<td>5.7</td>
<td>Path Failure</td>
<td>110</td>
</tr>
<tr>
<td>5.7.1</td>
<td>Permanent Failure</td>
<td>110</td>
</tr>
<tr>
<td>5.7.2</td>
<td>Temporary Failure</td>
<td>114</td>
</tr>
<tr>
<td>5.8</td>
<td>Shared Bottleneck</td>
<td>117</td>
</tr>
<tr>
<td>5.9</td>
<td>Variant Loss Rate</td>
<td>121</td>
</tr>
<tr>
<td>5.10</td>
<td>Conclusion</td>
<td>123</td>
</tr>
</tbody>
</table>
CHAPTER 5. MPTCP

This chapter presents the results of a number of experiments with a SAN-based simulation model of MPTCP. The model, which fully implements the SEMICOUPLED congestion control algorithm, is described and validated first. Then, we evaluate the behaviour of SEMICOUPLED in case of 1) increasing number of available paths 2) temporary and permanent path failure 3) existence of a shared bottleneck between paths and 4) paths with variable loss rate.

5.1 Introduction

Multipath TCP (MPTCP) [55] is an extension to traditional TCP which enables it to use multiple paths between multihomed/multi-addressed communication peers simultaneously. The aim of MPTCP is to improve resource utilization and fault tolerance. MPTCP is a set of features on top of the TCP which is supposed to be backward compatible so that it can work with middle boxes (e.g. NAT, firewall, proxy) and legacy applications and systems without any change to applications software or to the working habits of the users [54].

An MPTCP connection is started like a regular TCP connection. Then, if there are extra paths available, additional TCP connections (subflows) will be created. MPTCP operates in such a way that all these connections look like a single TCP connection. Subflows share a single receive window, but different congestion windows. In other words, only one receive window is kept per connection, but one congestion window is kept per subflow.

Hosts are free to use their own policy for spreading data over different paths. A multipath flow should behave in a way that improves throughput, does not harm other flows and balances congestion. In other words, a multipath transport should be fair at shared bottlenecks, be able to balance the load on available paths, present a stable behaviour in the long run and adapt itself to changes in load and other dynamics.

Using separate congestion control on different paths, known as UNCOUPLED, does not consider common bottlenecks and, therefore, does not provide load balancing and resource pooling. By coupling congestion windows, using an algorithm like COUPLED, load balancing, fairness and stability would be achievable, but inability to adapt properly to fluctuating congestion, which is a common event in the Internet, is a major weak point of this family of algorithms [126]. The SEMICOUPLED work [126, 125] proposes a set of algorithms combining COUPLED and UNCOUPLED to achieve desired features of a congestion control algorithm for a multipath transport. SEMICOUPLED ($\varphi=1$) is a member of this family, which is easy to implement in the operating system, and has been proposed for MPTCP [103]. We study the behaviour of this algorithm in this chapter. The aim of this study is to see how SEMICOUPLED reacts to dif-
ferent levels of congestion and how this reaction affects throughput. As a candidate solution for multihoming, MPTCP should be able to handle provider failures, which can be considered as a severe level of congestion, effectively. It also should be able to utilise available communication links to the Internet. In a multihomed site, different communication links provide connectivity to the Internet through different providers. As a result, different communication links are likely to have different characteristics and a multihoming solution should be able to utilise them efficiently.

For the experiments in this chapter, we use Stochastic Activity Networks (SAN) [85] to build a simulation model of MPTCP and the Möbius modeling tool [41] to simulate and analyze the model.

5.2 Related Work

Wischik et al. introduce the SEMICOUPLED congestion control algorithm in [126][125]. They also present results of a number of simulation experiments using their own simulator to show that the proposed algorithm meets the requirements for a multipath congestion control algorithm. We use results from [126] to validate our simulation model (see section 5.5). They use a two-path scenario for their experiments. They propose the algorithm proposed to IETF be employed by MPTCP [103]. Raiciu et al. [102] propose using MPTCP in data centers to improve performance and robustness. They show that MPTCP outperforms standard TCP when path diversity is available in a data center network. Chen et al. [36] investigate the effectiveness of MPTCP over cellular and WiFi channels by conducting some experiments with a simple two-path scenario.

The work presented in this chapter is a behavioural study on SEMICOUPLED, from multihoming viewpoint, and different from the above-mentioned research in some aspects. In section 5.7, we investigate the effect of permanent link failure and temporary link failure and recovery. Provider failure, and therefore link failure, is a possible event in a multihomed site, expected to be handled effectively by the deployed multihoming solution. In section 5.6, we extend the scope of experiments in [126] and increase the number of subflows up to eight subflows. Our results show the behaviour of SEMICOUPLED, in detail, when the number of subflows are increased. The results also show which characteristics of the path have more effect on throughput. In section 5.8, we again extend the scope of experiments in [126], increase the number of subflows up to 64 and specifically investigate the effect of endogenous losses. Our results show the effect of increasing the number of subflows on short-lived and long-lived communications. In section 5.9, since we have a controllable environment for our experiments, we run a number of experiments similar to [36] and then compare results with equivalent
scenarios with a mixture of paths with fixed and variable loss rates.

5.3 SEMICOUPLED (\(\varphi=1\)) Congestion Control Algorithm

SEMICOUPLED (\(\varphi\)) is a family of algorithms which tries to balance resource pooling and equipoise. Resource pooling refers to a situation where a set of disjoint bottleneck links behave as if they are a single pooled resource. Equipoise, or equal balance, refers to a situation where the traffic is balanced across available paths to some suitable extent. By choosing a proper value for \(\varphi\), it is possible to relax the tension between resource pooling and equipoise. With \(\varphi=0\), SEMICOUPLED behaves similarly to COUPLED algorithms which achieve resource pooling but not equipoise. COUPLED algorithms are so sensitive to congestion that even a short-term congestion in a path moves too much traffic off the path. With \(\varphi=2\), SEMICOUPLED behaves similar to UNCOUPLED where each subflow controls the congestion separately. UNCOUPLED algorithms maintain equipoise but are not able to achieve resource pooling. SEMICOUPLED (\(\varphi=1\)) is a good choice since the required calculations become simple and can be computed using basic arithmetic, while it provides a good level of balance between resource pooling and equipoise.

SEMICOUPLED (\(\varphi=1\)), with RTT compensation to achieve fairness, is the algorithm which has been proposed to be employed in MPTCP as the congestion control algorithm [126][125][103]. It combines COUPLED and UNCOUPLED and tries to be fair at common bottlenecks and also provide resource pooling to meet the desired properties of a multipath-aware congestion control algorithm.

The algorithm considers the effect of round trip time (RTT) and proposes coupled increases in TCP congestion avoidance state:

- For each ACK on path \(r\), increase \(w_r\) by \(\min(\alpha/w_{total}, 1/w_r)\)

where \(w_r\) is the congestion window size for the subflow on path \(r\) and \(w_{total}\) is the sum of congestion window sizes of all subflows. \(\alpha\) shows the amount of coupling between subflows and determines how aggressively the congestion window size should be increased. It is calculated using the following formula:

\[
\alpha = W_{total} \frac{\max_r(W_r/RTT_r^2)}{\left(\sum_r W_r/RTT_r\right)^2} \tag{5.1}
\]

where \(RTT_r\) is the RTT for path \(r\).

In case of loss, it follows the traditional New Reno behaviour:
• For each loss on path \( r \), decrease \( w_r \) by \( w_r / 2 \)

In the rest of this thesis, we refer to SEMICOUPLED (\( \varphi = 1 \)) as SEMICOUPLED.

5.4 Model Description

We have built a model of MPTCP with the SEMICOUPLED congestion control algorithm using stochastic activity networks. The model has four components: application, subflow, failure and measurement. The application submodel models the behaviour of a heavy load application which generates 10 MByte/s load in form of 1 kB packets. The subflow submodel simulates the behaviour of a TCP subflow with the SEMICOUPLED congestion control algorithm. The failure submodel models failure/recovery events in one of the paths. The measurement submodel models a timer which expires every second. It collects some information and also applies changes to the model parameters, if required. We have used the Möbius simulation engine to run the experiments for this chapter.

Fig. 5.1 shows the graphical representation of the application submodel. The first part of the submodel, containing place app_start, activity app_init and gate app_initialize, initializes the submodel. This submodel is also able to model the behaviour of a light load application, but that part has not been used in the experiments for this chapter. Activity app_initialize decides the type of application and also the number of packets which should be generated during simulation. The initial marking of app_start is 1, so this part runs only once at the start of simulation. Activity fast_send fires every (1.0/application-sending-rate) second and adds one packet to the socket buffer if the buffer is not full. Place app_data represents the socket buffer. application-sending-rate and buffer size are two model parameters which have been set to 10,000. In other words, the application submodel generates a fixed and heavy load equivalent to 10,000 packets per second for the experiments in this chapter. Packet size has been set to 1 kB.

![Figure 5.1: Graphical Representation of the application submodel.](image)

Fig. 5.2 shows the graphical representation of the subflow submodel. It models the behaviour of an established TCP flow between two MPCTP-enabled hosts. Sending and
receiving data and acknowledgements and calculations for the retransmission timer and congestion window are included in this submodel. Gate `start_actions` initializes subflow variables, like congestion window and retransmission timer, to their default initial values at the start of simulation. Place `app_data` is a variable shared between this submodel and the application submodel, and models the socket buffer. Activity `tcp_send` models packet processing time in the TCP stack. This parameter is configurable in the model and has been set to 0.02 ms in the experiments. One packet is sent every 0.02 ms if there is a packet available in `app_data` for sending and also an empty space is available in the send buffer. These conditions are checked by input gate `tcp_send_ok`. Then `post_send` increments the number of packets in the send buffer and decrements number of packets in the socket buffer (`app_data`) when `fast_send` or `slow_send` fires. Marking of the place `sent_segment` always shows number of packets in the send buffer which are waiting for acknowledgement.

Figure 5.2: Graphical representation of the subflow submodel.

Receiving an acknowledgement is modelled by activity `wait4ack`. Output gate `rcv_ack` implements the calculations which are performed by the TCP receiver upon receiving an acknowledgement. Calculating retransmission timer value and congestion window size are the most important parts of the `rcv_ack` output function. It also models packet losses by considering some acknowledgements as loss reports (duplicate ack). Fig 5.3 shows how `rcv_ack` calculates a new size for a subflow’s congestion window when an acknowledgement is received. In fact, it is an implementation of the algorithm described in section 5.3. `tcp_counters` is an extended place which contains internal variables. `cong_wnd_array` is an array which contains congestion window sizes for subflows. `rtt_array` is another array which stores RTTs for subflows. To access the array elements, Möbius provides its own syntax. For example, `tcp_counters->cong_wnd_array->Index(2)->Mark()` provides access to the congestion window of the third subflow. The
algorithm calculates the total congestion window first. Then it calculates $\alpha$ and finally calculates how much the congestion window should grow for the associated subflow.

```c
// calculate total congestion window
for (i = 0; i < num_subflow; i++)
    total_cwnd += tcp_counters->congWndArray->Index(i)->Mark();

// coupling is done in congestion avoidance
if (tcp_counters->congWnd->Mark() >= tcp_counters->tcpSsthresh->Mark()) {
  // calculate Max(\frac{Wr}{RTT_\text{avg}}^2) and \text{SUM}(\frac{Wr}{RTT_\text{avg}})
  for (i = 0; i < num_subflow; i++) {
    temp = (tcp_counters->congWndArray->Index(i)->Mark() * mss * mss) /
            (tcp_counters->rttArray->Index(i)->Mark() * tcp_counters->rttArray->
             Index(i)->Mark());
    if (temp > max)
      max = temp;
    sum += tcp_counters->congWndArray->Index(i)->Mark() * mss /
           tcp_counters->rttArray->Index(i)->Mark();
  }

  // calculate $\alpha$ and how much congestion window for this subflow should grow
  alpha = total_cwnd * max / (sum * sum);
  grow = min(round(alpha*(packet_acked*tcp_counters->packetSize->Mark())*mss/
              total_cwnd),
             (packet_acked*tcp_counters->packetSize->Mark())*mss/tcp_counters->
             congWnd->Mark());
  tcp_counters->alpha->Mark() = alpha;
}
```

Figure 5.3: Part of the output function for gate `rcv_ack` which calculates the new congestion window size upon receiving an acknowledgement.

The send buffer is implemented as a circular queue to make this part of the model as close as possible to a packet level model. Activity `post_send` inserts the sent packet at the rear of the queue. Sending time and RTT sample are also stored for each packet in the queue. RTT is a model parameter and can be set for each path. Enabling time of the activity `wait4ack` is adjusted using the RTT of the packet located at the front of the queue. When `wait4ack` fires, all packets that have `sending-time+RTT-sample` equal or greater than current time are considered as acknowledged and removed from the queue.

The TCP retransmission timer is modelled using activity `ret_timer`. When a packet is sent, this timer starts by setting `ret_timer_in` marking to `max_retransmit` which is a global variable and can be set as a model parameter. `max_retransmit` specifies the number of retransmissions before the connection gets terminated. The activity restarts when an acknowledgement is received. It becomes disabled when the send buffer is empty. Gate `ret_timer_timeout` implements the actions which should be taken when the retransmission timer expires. It doubles the retransmission timer value and drops the congestion window to 1.

Places `failure_out` and `recovered` are shared between subflow and failure submodels to provide a communication medium for exchanging failure and recovery notifications.
When a failure event is generated by the failure submodel, the \textit{failure\_out} marking changes to 1 which enables \textit{failure\_detect}. It fires immediately and blocks the gate \textit{wait\_in\_gate} which disables \textit{wait4ack}. That means that no acknowledgement will be received after that. So, when the congestion window becomes full, sending data will also stop. When the failure is resolved, the marking of place \textit{recovered} is set to 1 by the failure submodel. It enables \textit{recovery\_done} in the subflow submodel. Firing \textit{recovery\_done} unblocks \textit{wait\_in\_gate} so that \textit{wait4ack} will be able to fire again. That means that acknowledgements can be received and the subflow is able to go back to its normal state.

For each subflow, loss rate can also be specified as a model parameter. We have implemented loss events in output gate \textit{rcv\_ack}. When \textit{wait4ack} fires, a random number between 0 and 100 is generated. If the generated number is less than subflow’s loss rate percentage, the acknowledgement is considered as a loss report and the actions, specified by the SEMICOUPLED algorithm for this case, are performed.

A graphical representation of the failure submodel is shown in Fig. 5.4. It is a simple submodel which generates a failure event. In case of temporary failure, then it waits for a specified time and generates a recovery event after that. Initial marking of place \textit{failure\_in} is 1, so activity failure is enabled when simulation starts. When it fires, gate \textit{failure\_occurred} modifies the marking of failure\_out to notify the subflow submodel of the failure. It also modifies the marking of \textit{wait4recovery} to enable activity \textit{recovery}. The enabling times of both activities are model parameters which can be set to arbitrary values. When \textit{recovery} fires, it modifies the marking of the place \textit{recovered} to notify the affected subflow about recovery completion.

![Figure 5.4: Graphical representation of the failure submodel.](image)

Fig. 5.5 shows the graphical representation of the measurement submodel. This submodel is used for measuring some parameters during simulation. Activity \textit{one\_sec\_timer} implements a timer which fires every second. Gate \textit{one\_sec\_timer\_process} implements a function which calculates the average throughput and checks to see whether it shows a stable state. It is mainly used for experiments with temporary or permanent failures. It is also used to modify model parameters to simulate dynamic and variant behaviours of the system.
Fig. 5.6 shows how the submodels are connected to each other. The Rep/Join feature of Möbius allowed us to replicate the subflow submodel and implement the multipath nature of MPTCP in our model. Since the SEMICOUPL ED congestion-control algorithm needs to access the congestion window and RTT for all paths, this information has been shared between replicas. The replicated component is joined with the application, failure and measurement submodels to build a complete model of an MPTCP-enabled host. Fig. 5.7 shows the composed model in the Möbius environment.

Figure 5.6: How submodels of the MPTCP SAN model are connected to each other.
5.5 Validating the Model

In this section we present the results from our model for the case of two subflows. Wischik et al. [126] did a performance study on MPTCP using their own simulator. To check the accuracy of our model, we compared our results with [126]. Fig. 9 in [126] shows the results from two experiments with one MPTCP connection, which runs two subflows on two paths with equal and unequal loss rates under constant levels of congestion. In the first experiment, RTT and loss rate for one path is set to 86 ms and 0.57%, respectively. For the other path, they are set to 431 ms and 0.56%. In the second experiment, RTT and loss rate are set to 93 ms and 1.79% for the first path and to 414 ms and 0.26% for the second path. Fig. 5.8 shows results from our model. The left hand plots show the results from the experiment with equal loss rates and the right hand plots show the results from the experiment with unequal loss rates. All model and simulation parameters have been set to the same values as specified in [126] (section 5.2, Fig. 9). The first graph (top), shows the changes in congestion window for each subflow. The second graph shows the throughput for each subflow. The third graph, shows the changes in $\alpha$ and the last graph, at the bottom, shows the changes in total throughput. The results are from one run of simulation, similar to [126], although we believe that one run does not provide sufficient accuracy for results. The samples, in our result, have been collected every one second, which explains why our results do not show fluctuations like those in [126] since they have collected the samples every RTT. Considering the above-mentioned points, the results show that the behaviour of our
SAN model is similar to the model in [126], allowing us to be confident of our model.

Figure 5.8: Changes in congestion window $\alpha$ and throughput for two subflows for the case of equal and unequal loss rates.
5.6 Increasing Number of Subflows

Section 5.5 discussed our simulation results for two subflows. In this section we investigate the effect of increasing the number of subflows on the behaviour of the SEMICOUPLED algorithm and throughput.

Each subflow runs on a separate path. The first subflow employs a path with RTT=80 ms and the eighth subflow employs another path with RTT=150 ms. Other subflows run on separate paths which have RTTs in between with 10 ms steps, i.e. 90 ms, 100 ms, 110 ms, 120 ms, 130 ms and 140 ms. In these experiments, we have assumed that there is no bottleneck link shared between paths and each path is used by a large number of other flows. So the losses are exogenous and the loss rate can be approximated by a fixed rate. The loss rate on all paths has been set to 0.1%. We will look into endogenous losses in section 5.8. To get more stable and reliable results, we have run the simulation for 100 times.

Fig. 5.9 shows how the congestion window for the first and last subflows, on the fastest and slowest paths, changes in the case of eight subflows. Both subflows increase their congestion window rapidly at the start of simulation, but packet loss and SEMICOUPLED algorithm forces them to decrease their congestion window. The eighth path has the largest RTT, so it takes more time for its associated subflow to decrease its congestion window. When the system becomes stable, the congestion window size for all subflows converges to a similar value, as the loss rates for all paths have been set to the same value. That is an expected behaviour of the SEMICOUPLED algorithm which tries to provide fairness and resource pooling.

![Figure 5.9: Changes in congestion window for the 1st and 8th congestion window. In a stable state, the congestion window size for both subflows converges to a similar value since they have the same loss rates.](image)

Fig 5.10 shows the changes in total throughput. When the SEMICOUPLED algo-
Algorithm is used, increasing the number of subflows increases total throughput. This is another expected behaviour of MPTCP, which tries to utilize available paths to increase the performance.

Our results also show that increasing the number of subflows does not necessarily increase the throughput with the same ratio. Our experiments, with a set of paths with similar attributes (RTT=80 ms, Loss Rate=0.1%), show that by increasing the number of subflows from one to two, gain for the throughput is about 12%; from two to four it is about 17% and from four to eight, the gain is about 21%. The loss rate is the main reason for this. Packet losses force subflows to decrease their congestion window sizes and SEMICOUPLED controls them to be more conservative about increasing them. In an ideal environment, with no packet loss, increasing the number of subflows will increase the throughput with the same ratio but in the real world, and especially on the Internet, errors and packet losses cannot be ignored.

Fig 5.11 shows a closer look at the throughput graphs. Vertical lines show error intervals. The graph shows that, in the case of using paths with similar attributes, increasing the number of subflows will lead to more stable throughput.

To get a more detailed view of how path attributes affect performance, we conducted another experiment. In this experiment, we started with one subflow which ran over a path with RTT=80 ms and loss rate=0.1% (experiment 1). Then we added a second subflow which ran on a new path in three cases:

2. Second path has similar attributes.
3. Second path is 50% faster (RTT=40 ms) with the same loss rate.
4. Second path has the same RTT and 50% less loss rate (0.05%).

Figure 5.10: Changes in total throughput in case of 1, 2, 4 and 8 subflows. Increasing the number of subflows increases total throughput.
Figure 5.11: A closer look at Fig. 5.10. Using eight subflows shows more stable throughput.

In the rest of this section, we will refer to these experiments with their numbers. Fig. 5.12 shows the changes in average throughput. Adding a faster path (lower RTT) shows a significant improvement in throughput. In two other cases, the improvement is less and the average throughput is close. That means that the RTT of the added path has more effect on throughput than its loss rate. Fig. 5.13 shows the changes in total congestion window for experiments 2, 3 and 4. Total congestion window for experiment 4 reaches to a bigger size than the other experiments in a stable state because its loss rate is smaller than that for the two other experiments. For experiments 2 and 3, total congestion window size converges to the same value since they have similar loss rates. Although subflows in experiment 4 are able to have bigger congestion windows, their higher RTT does not allow them to deliver more packets than subflows in experiments 2 and 3.
CHAPTER 5. MPTCP

Figure 5.12: Changes in average throughput for experiments 1, 2, 3 and 4. Adding a faster path shows a significant improvement in throughput.

Figure 5.13: Changes in total congestion window for experiments 2, 3 and 4. Total congestion window for experiment 4 reaches to a bigger size than other experiments in stable state because the loss rate is smaller than the two other experiments.

Fig. 5.14 shows the total number of sent packets for the experiments during the simulation. Experiment 4 shows a better performance for a limited time at the beginning of the simulation. But as the time goes on, experiment 3 increases its throughput faster than 2 and 4 and makes a big difference at the end. Having a lower loss rate gives the second subflow in experiment 4 a chance to increase its congestion window faster at the start of the simulation and send more packets than the other experiments. But it is just a temporary advantage. After a while, packet losses start to happen and force the subflow to decrease its congestion window. In the long run, lower RTT gives
more opportunity to the added subflow in experiment 3 to send more packets than the other experiments with the same or even bigger congestion windows. In other words, smaller loss rate leads to larger congestion windows but it does not necessarily lead to higher throughput.

Figure 5.14: Total number of sent packets for experiments 2, 3 and 4. In the long run, lower RTT, and in the short run, lower loss rate lead to a higher throughput.

5.7 Path Failure

In this section, we investigate how SEMICOUpled reacts to a path failure. Path failure is an event which causes a path not to be able to send and receive data. Our model is able to simulate two types of path failures: permanent and temporary. Permanent path failure is when the path fails and never comes back to the normal state. Temporary failure is a failure for a limited period of time. Our failure submodel is able to model both events. Time and duration of failure can be set in the model.

In the experiments in this section, we aim to investigate the effect of path failure on the behaviour of the SEMICOUpled algorithm. We are specifically interested to see how the algorithm reacts to the failure and how effectively it is able to recover from it. We also measure the time it takes for the system to bring the throughput back to a stable state. We define stable state as a condition where the changes in throughput in a 5-second time window are less than 10%. We refer to the difference between end point of the window and failure time as recovery time in the rest of this section.

5.7.1 Permanent Failure

In this section, we have started with a model of a connection with four subflows on four different paths. RTT and loss rate for all paths have been set to 100 ms and
0.1% respectively. Failure occurs at time 45,000 on the first path. Fig. 5.15 shows the changes in throughput around the time 45,000, when the failure occurs. Throughput drops when path 1 fails and the associated subflow drops its congestion window to one and stops sending data. When SEMICOUPLED detects this situation, it starts moving the load to the other subflows by increasing the value of $\alpha$. Increasing $\alpha$ allows the other subflows to increase their congestion window to be able to carry more load. Since all paths have the same attributes, the extra load is divided equally among them. Fig. 5.16 shows how $\alpha$ changes after failure and Fig. 5.17 shows how changes in $\alpha$ reflect the total congestion window. As can be seen in the Fig. 5.15, throughput is gradually increased to recover the failure. Recovery time for this experiment is 6120 ms.

![Throughput](image)

Figure 5.15: Changes in total throughput in case of a permanent path failure at time 45000. Throughput is gradually recovered by moving the load to other available paths.
CHAPTER 5. MPTCP

Figure 5.16: Changes in $\alpha$ in the case of a permanent path failure at time 45000. $\alpha$ is increased to move the load to other paths.

Figure 5.17: Changes in total congestion window in the case of a permanent path failure at time 45000. Other subflows increase their congestion window to accommodate the load.

To see how the number of paths can affect recovery time, we have conducted a set of experiments with 2, 4 and 8 subflows. RTT and loss rate for all paths have been set to 100 ms and 0.1% respectively. Failure occurs at time 45,000 on the first path. Fig. 5.18 shows the recovery time for each experiment. It shows that increasing the number of paths decreases the recovery time. The difference in recovery time between 2 and 4
subflows is noticeable while there is a small difference between 4 and 8 subflows. That means that having more subflows not only makes the throughput more stable, it also enables MPTCP to stabilize throughput faster when a path failure occurs. The reason is that changes in $a$ have an inverse relationship with the number of paths in this case. So, when there are more subflows, smaller changes in $a$ can take care of the drop in throughput caused by failure. That makes the throughput less unstable and, therefore, easier to stabilize. Fig. 5.19 shows the changes in $a$ in case of 2, 4 and 8 subflows.

![Recovery Time (ms)](image)

Figure 5.18: Recovery time in the case of 2, 4 and 8 subflows in the case of permanent path failure. Increasing the number of paths, and therefore, subflows, decreases the recovery time. The difference in recovery time between 2 and 4 subflows is noticeable.

![Changes in $a$](image)

Figure 5.19: Changes in $a$ for 2, 4 and 8 subflows in case of a permanent failure at time 45000.
5.7.2 Temporary Failure

Temporary failure refers to the condition that a path failure occurs, but only for a limited time interval. After that period the failed path recovers and goes back to normal. A severe congestion or a temporary error in the path can cause this kind of failure. In 5.7.1 we investigated the behaviour of SEMICOUPLED against the failure. In this section we conducted some experiments to investigate how SEMICOUPLED reacts to path recovery.

We started with an experiment with two subflows which run on two different paths. RTT and loss rate for both paths have been set to 100 ms and 0.1% respectively. Failure occurs at time 45,000 on the first path. The failed path recovers and goes back to normal at time 75000. Fig. 5.20 shows the changes in subflow congestion windows during the time interval which includes failure and recovery events. When subflow 1 detects failure, it drops the congestion window size. SEMICOUPLED detects the failure and tries to move the load to the second path. The details of this process were already described in section 5.7.1. At time 75000, the failed path recovers and goes back to normal. The associated subflow starts increasing its congestion window. Increasing the congestion window causes SEMICOUPLED to adjust $\alpha$ so that some load can be transferred back to the recovered path. Fig. 5.21 shows how $\alpha$ changes in reaction to failure and recovery events. Fig. 5.22 shows the changes in total throughput. It shows that SEMICOUPLED reacts to the recovery and restores the throughput quickly.
Figure 5.20: Changes in congestion window for an experiment with two subflows in case of a temporary failure. Failure occurs at time 45,000 on the first path. The failed path recovers and goes back to normal at time 75000.

Figure 5.21: Changes in $\alpha$ for an experiment with two subflows in case of a temporary failure. Failure occurs at time 45,000 on the first path. The failed path recovers and goes back to normal at time 75000.
We have repeated the same experiment with 4 and 8 subflows. Fig. 5.23 shows the results. Increasing the number of subflows does not show any special difference in the behaviour of SEMICOPLED in this case.

Figure 5.23: Changes in total congestion window, $\alpha$ and throughput for 2, 4 and 8 subflows in case of a temporary failure. Failure occurs at time 45,000 on the first path. The failed path recovers and goes back to normal at time 75000.
5.8 Shared Bottleneck

In the experiments in previous sections, we assumed that subflows employ separate paths and there is no shared channel between them. In other words, we assumed that each subflow runs on a totally different path and only exogenous errors affect the subflows. In this section, we aim to investigate the behaviour of SEMICOUPLED where there is a shared communication channel. We assume that there is a communication channel which is shared by all available paths for an MPTCP connection. As a result, the loss rate is not fixed and is driven by the subflows. Paths still can have different RTT.

We start with a set of experiments with 1 and 2 subflows. RTT for all paths has been set to 100 ms. Paths share a channel with capacity 8000 pkt/s. Application sending rate has been set to 10000 pkt/s. Results are from one simulation run and show a typical scenario. Fig. 5.24 shows the changes in total congestion window for the experiments. Changes in congestion window for the case of one subflow shows a fixed pattern. It increases the congestion window until it reaches the shared path capacity. Then packets are lost and cause the flow to decrease its congestion window. Changes in congestion window in the case of two subflows show that SEMICOUPLED is less aggressive and more conservative in increasing congestion window. It increases congestion windows slower to avoid losses. Fig 5.25 shows loss rates for the experiments. As the figure shows, loss rate in the case of a single subflow is much higher than the case of two subflows. SEMICOUPLED tries to decrease loss rate by adjusting $a$. Fig 5.26 shows how and when subflows are affected by packet losses and how $a$ is adjusted accordingly. The figure shows that changes in $a$ reflects changes in loss rate. Higher loss rate causes more variation in $a$. Fig. 5.8 is a sample. In the case of unequal loss rate (higher loss rate), more variation is observed in $a$ compared to the equal loss rate (lower loss rate). In other words, $a$ can be used as an indicator for loss rate.
Figure 5.24: Changes in total congestion window for two experiments with 1 and 2 subflows when there is a shared channel between subflows.

Figure 5.25: Loss rates for two experiments with 1 and 2 subflows when there is a shared channel between subflows.
Figure 5.26: How loss rate affects $\alpha$, in an experiment with 2 subflows, when there is a shared channel between subflows.

To get a more general picture of the behaviour of SEMICOUPLED when a shared bottleneck exists, we conducted another set of experiments with 1, 2, 4 and 8 subflows and ran the simulation 100 times. Fig. 5.27 shows the changes in loss rate in the case of 4 and 8 subflows. It confirms that increasing the number of subflows leads to lower loss rates. Fig 5.28 shows the changes in throughput in the case of 2 and 4 subflows. The throughput is increased more slowly and more smoothly in the case of 4 subflows. Fig. 5.29 shows the total number of packets which have been sent in the case of 1, 4 and 8 subflows. It shows that having more subflows will lead to a better performance in the long run. Some research in the past shows that running standard TCP flows in parallel improves performance, but it will have a negative impact on the throughput if the number of flows exceeds a certain maximum. For example, [114] shows that maximum performance is achieved by using 12 parallel TCP flows. To investigate whether SEMICOUPLED will also show such behaviour, we gradually increased the number of subflows to 64, but we did not observe any reduction in throughput. Fig. 5.30 shows the results and confirms that employing more subflows will have a positive impact on the throughput until it exceeds the link capacity. The effect is almost linear.
Figure 5.27: Changes in lose rate in the case of 4 and 8 subflows when there is a shared channel between subflows. Increasing the number of subflows leads to lower loss rates.

Figure 5.28: Changes in throughput in the case of 2 and 4 subflows when there is a shared channel between subflows. The throughput is increased more slowly and more smoothly in case of 4 subflows.
Figure 5.29: Total number of sent packets in the case of 1, 4 and 8 subflows when there is a shared channel between subflows. Having more subflows will lead to a better performance in the long run.

Figure 5.30: Average throughput for different number of subflows when there is a shared channel between subflows. Employing more subflows will have a positive impact on the throughput.

5.9 Variant Loss Rate

In 5.6, we investigated the behaviour of the SEMICOUPLED algorithm in high statistical multiplexing cases and considered a fixed loss rate for communication paths. In this section we aim to investigate how SEMICOUPLED reacts to the paths with variant loss rate. Communication channels which employ wireless technologies, like WiFi, 3G or satellite links, show such behaviour in practice. In this section we conduct some
experiments to investigate the behaviour of the SEMICOUPLED in presence of such paths.

We run three experiments with an MPTCP connection which runs over two paths, one subflow over each path, with RTT=100 ms:

1. Both paths have a fixed loss rate (0.5%). This experiment is used as a reference.

2. Path 1 has a fixed loss rate (0.5%). The second path has a variant loss rate which randomly, with a uniform distribution, changes between 0% and 1% every 5 seconds. So, the average loss rate for the second path is 0.5%, equal to the fixed loss rate for the first path.

3. Both paths have a variant loss rate which randomly, with a uniform distribution, changes between 0% and 1% every 5 seconds. So, the average loss rate for both paths is 0.5%

Fig 5.31 shows the changes in total throughput for above experiments. The total throughput for all experiments converges to the same value in a stable state. That means that although the loss rate has changed every 5 seconds in experiments 2 and 3, SEMICOUPLED could handle the changes properly and provide a throughput similar to experiment 1. Fig 5.32 shows the loss rate on path 2 in experiment 2.

![Throughput Graph](image)

Figure 5.31: Changes in total throughput for three experiments with fixed and variant loss rates. SEMICOUPLED could handle the changes in loss rate properly and total throughput for all experiments converges to the same value in a stable state.
CHAPTER 5. MPTCP

Figure 5.32: Changes in loss rate for a path with a variant loss rate. Loss rate changes randomly between 0% and 1% every 5 seconds.

5.10 Conclusion

In this chapter, we present the results of a study, based on simulation, of the SEMICOUPLED congestion-control algorithm. This algorithm has been proposed to be employed by MPTCP. We built a simulation model of SEMICOUPLED using SANs. Using the model, we studied the effect of increasing number of subflows, path failure, shared bottleneck and variant loss rate on the behaviour of SEMICOUPLED.

Increasing the number of paths, and therefore subflows, improves throughput but not necessarily in the same ratio. In other words, if we double the number of subflows the throughput is not necessarily increased by a factor of two. Packet loss is the main reason for that. It forces subflows to decrease their congestion window sizes and SEMICOUPLED controls them to be more conservative about increasing them again. We have also conducted some experiments to study how adding a new path can affect the throughput. The results show that the RTT of the new path has more effect on throughput than its loss rate. Therefore, if there is a choice between two paths, using the path with lower RTT may provide higher throughput. Smaller loss rate leads to a larger congestion window but it does not necessarily lead to a higher throughput.

Another set of experiments in this chapter aims to investigate how SEMICOUPLED reacts to a path failure and how the number of paths can affect recovery time and throughput in the case of temporary and permanent failures. The results show that increasing the number of paths reduces the recovery time and makes the throughput
more stable. It also enables MPTCP to stabilize throughput faster when a permanent path failure occurs. In the case of temporary failures, we conducted a set of experiments to show how SEMICOUPLED reacts to the path recovery. The results show that SEMICOUPLED reacts to the recovery and restores the throughput quickly. They also show that increasing the number of subflows does not have a noticeable effect on the behaviour of SEMICOUPLED in this case.

Investigating the behaviour of SEMICOUPLED in the case of the existence of a shared channel between communication paths is another area of the work presented in this chapter. In this case, our model simulates a communication channel which is shared by all available paths for a MPTCP connection, so the loss rate is not fixed and is driven by the subflows. The results in this case show that SEMICOUPLED is conservative in increasing congestion windows which leads to lower loss rate and better performance in the long run. Short-lived connections do not benefit from increasing the number of subflows, they only experience a smoother throughput.

We also investigated how SEMICOUPLED reacts to paths with variant loss rate. The results show that SEMICOUPLED is able to handle frequent changes in loss rate properly and provide a stable throughput. The results in this case do not show a noticeable difference compared to the case of using paths with a fixed loss rate or mixture of paths with fixed and variant loss rates. That means that MPTCP is a good choice for sites with varying loss rate links like satellite links.

Measuring loss rate is a challenging task. We found that $\alpha$ is a good indicator for loss rate. $\alpha$ is a parameter in SEMICOUPLED algorithm that controls how aggressively to increase congestion window size. Higher loss rate causes more variation in $\alpha$. Monitoring changes in $\alpha$ can help SEMICOUPLED to evaluate loss rate of in-use paths.
Chapter 6

Traffic Engineering in SHIM6

Contents

6.1 Introduction .............................................. 126
6.2 Traffic Engineering Solutions for SHIM6 .............. 129
  6.2.1 Proxy SHIM6 ....................................... 130
  6.2.2 Extended SHIM6 ............................... 131
6.3 NAROS: A Traffic Engineering Solution for Host-centric Multihoming .............. 132
6.4 Adopting NAROS in SHIM6-enabled Networks (SHIM6-enabled NAROS) .............. 134
  6.4.1 Reporting Locator Preferences to NAROS ............ 135
  6.4.2 Enforcing Dynamic TE Policies ...................... 137
6.5 SHIM6-enabled NAROS vs Proxy SHIM6 and Extended SHIM6 ...................... 140
6.6 Conclusion .............................................. 142
This chapter focuses on traffic engineering which is one of the major weak points of SHIM6. We propose a mechanism which provides a dynamic site-aware traffic engineering solution for SHIM6-enabled networks. This solution is based on the Name, Address and ROute System (NAROS) [79] and we call it SHIM6-enabled NAROS.

6.1 Introduction

The IETF traffic engineering group\textsuperscript{1} defined Internet Traffic Engineering (TE) as:

\begin{quote}
That aspect of Internet network engineering concerned with the performance optimisation of traffic handling in operational networks, with the focus of the optimisation being minimised over-utilisation of capacity when other capacity is available in the network.
\end{quote}

According to this definition, TE is a process which affects routing decisions in order to reach a goal or meet a requirement. The goal or requirement can be anything; e.g. minimising cost, balancing load, maximizing performance, avoiding insecure routes etc. Every site, based on its characteristics, has its own goals for TE. Site administrators usually define some policies in order to help achieve their goals. ISPs have their own goals for TE, which are not discussed here.

A multihomed site is connected to the Internet through more than one service provider. One of the main TE requirements for a multihomed site administrator is to be able to control how much inbound/outbound traffic is exchanged with the site’s providers. The IPv6 addressing architecture [65] allows IPv6 hosts to have multiple unicast global IP addresses. This means that every host can be reachable via multiple IP addresses. Therefore, when two multiaddressed hosts decide to communicate with each other there is a set of source-destination address pairs which can be used for this purpose. In a multihomed site, different global unicast addresses are usually assigned to the hosts by Internet service providers. This is usually a two-stage process. An address block is assigned to the site by each ISP and then addresses within those address blocks are assigned to the hosts by using a mechanism like DHCPv6 [45] or address autoconfiguration [118]. So choosing the source address for the communication indirectly specifies the provider that should be used. Ingress filtering [50] is a security mechanism which is used by providers to prevent source address spoofing. A packet is discarded by a provider if its source address is not within the address block which has been assigned to the sender’s site. Fig. 6.1 shows an example of how ingress filtering can affect a multihomed site with multi-addressed hosts. A1 and A2 are assigned by \textit{provider}_A and B1 and B2 are assigned by \textit{provider}_B to the hosts. P2 and P3 are two

\textsuperscript{1}www.ietf.org/html.charters/tewg-charter.html
packets which are forwarded to the wrong provider and dropped because of ingress filtering. P2 carries B1 as source address but it is forwarded to provider\_A and P3 carries A2 as source address but it is forwarded to provider\_B. When a host picks an IP address as source address, the packet should be forwarded to the provider which owns the address block; otherwise the packet is likely to be discarded. The current solution for TE in IPv4 Internet uses BGP4 [105] features. In a multi-addressed site, this solution is not applicable because BGP is not able to correlate different address blocks assigned to the same site. That implies that in an IPv6 multi-addressed network, a new solution for TE is required.

RFC 3484 [44] and its successor [117] propose two algorithms in the form of two sets of rules. These algorithms can be used by multi-addressed IPv6 hosts when they need to choose source and destination addresses for communicating with remote hosts.
The algorithms try to order the list of available addresses based on their attributes. Although proposed rules and heuristics are helpful for a host to make a better decision, they are fixed and do not give the site administrators the ability to define and enforce their desired TE policies. In fact, those rules try to make the best choice for the host, but, since they only consider the host’s local information, the choice would not necessarily be the best choice for the site. In other words, each host decides independently, so there is no guarantee that the result would satisfy the TE requirements for the whole site. Therefore, for a multihomed site which employs a host-centric solution, like SHIM6, these algorithms do not provide an acceptable level of control over the site’s incoming/outgoing traffic.

Name, Address and ROute System (NAROS) [79] aims to help hosts in a multihomed-multi-addressed site to choose the best source address for communicating to remote hosts. It proposes a new service in the site for this purpose. It takes the responsibility of address selection off the hosts and moves it to a central service which is aware of host and network TE requirements. When a host needs to start a communication with a remote host, it asks the service which source address should be used for the communication. Therefore, hosts do not decide which source address to use. They let the service, which is aware of the site’s TE policies, decide and tell them which address is the best choice according to the TE policies. NAROS was proposed in 2003 and to the best of our knowledge no work has been done on it since then.

RFC 7078 [83] proposes using the DHCPv6 service, by adding a new option for distributing address selection policies. The idea of having a central service in the site which knows about TE policies is shared between NAROS and RFC 7078 but they propose different ways for implementing that. Integrating this service with DHCP eliminates the requirement for having a new service in the site but it has some drawbacks: 1) Hosts receive the whole policy table and should keep it for the future use. The table might be large and some entries might never be used. 2) Sites are forced to use DHCPv6. IPv6 offers other mechanisms like stateless address autoconfiguration [118] for dynamic configuration of IPv6 hosts as well. 3) Maximum DHCP message size restricts the policy table size.

SHIM6, as a host-centric multihoming solution, does not provide sufficient TE capabilities for administrators. There are some options in the SHIM6 protocol which enable hosts to exchange their preferences for choosing locators with the other end of the communication, but SHIM6 does not provide any mechanism for site administrators to use these options for controlling the traffic. The main focus of this chapter is to propose a solution which addresses this issue in a SHIM6-enabled site.

In this chapter, a new mechanism for traffic engineering, based on NAROS, is proposed. This mechanism adds some new requests to the NAROS protocol to enable
SHIM6 hosts and the NAROS server to exchange information with each other. It helps the NAROS server make a decision based on a richer set of information. In the proposed solution, SHIM6 hosts act as an enforcement point for the NAROS server’s decisions. In other words, SHIM6 hosts provide the NAROS server with extra information to help it to make better decisions. When a decision is made, it is enforced by SHIM6 hosts. To do this, an extension to NAROS and SHIM6 protocols is proposed.

What is described in this chapter is a proposal. To the best of our knowledge, there is no implementation for NAROS. Implementing this proposal needs the NAROS protocol to be implemented first. For testing, a couple of SHIM6 enabled sites that have deployed our version of NAROS and SHIM6 is required. Unavailability of these resources did not allow us to implement and test this proposal.

Existing proposals for traffic engineering solutions for SHIM6 are briefly described in the next section.

### 6.2 Traffic Engineering Solutions for SHIM6

There are three categories of solutions for enforcing TE policies in sites, in the case that hosts are multi-addressed:

1. **Exit routers decide and enforce TE policies:** This category of solutions assumes that exit routers are aware of TE policies. There are two options for enforcing policies in this case. One option is that the exit router rewrites source addresses if they are in conflict with TE policies. Therefore, it does not matter what source address is selected by the host. In the end, it is the exit router which decides which source address should be used. The second option is that the exit router checks the source address and sends back an ICMP error message to force the host to choose another address if the source address should be changed to comply with the TE policies.

2. **Hosts decide and enforce TE policies:** In this category, hosts decide which source address should be used. There are two options in this case: There is a service available in site to help hosts in this regard or they decide themselves by maintaining copies of BGP tables or algorithms, like the ones proposed in [117].

3. **A middle box decides and enforces TE policies:** In this category, neither hosts nor exit routers are aware of the TE policies. There is another component in the system which is aware of TE policies, and sits between host and exit routers. Hosts send packets to this middle box and it manipulates them, if required, before sending them to the exit router.
To the best of our knowledge, two solutions have been proposed to address TE issues in SHIM6 so far: Proxy SHIM6 [19] and Extended SHIM6 [96]. Proxy SHIM6 belongs to category 3 and Extended SHIM6 belongs to category 1. The rest of this section briefly describes these two solutions.

6.2.1 Proxy SHIM6

Proxy SHIM6 (P-SHIM6) is an extension to the SHIM6 architecture which aims to improve SHIM6 capabilities. It tries to add the following features to SHIM6:

- ULID (Upper Layer IDentifier) portability
- Traffic Engineering
- No SHIM6 context management at hosts
- Providing multihoming support for legacy hosts

In this approach, each multihomed site receives a PA prefix from each of its providers, but hosts are configured with a CMULA (Centrally Managed Unique Local Address) prefix from a central registry. CMULA is a non-routable PI prefix. CMULA should be globally unique, location independent and permanently allocatable. Hosts do not need to be SHIM6-enabled. Therefore, legacy hosts are also able to benefit from this approach. Hosts are configured with a single address containing the CMULA prefix which is used as their ULID. This means that host addresses are not dependent on ISPs, which makes the ULID portable and renumbering easy. The site is served by one or more proxy SHIM6 boxes. The SHIM6 protocol is executed by P-SHIM6 boxes on behalf of hosts; peer nodes should be SHIM6 enabled or behind a P-SHIM6 box.

In this approach, all address information is stored in DNS. Locators (host’s global IP addresses) are stored in AAAA resource records (RR). To store CMULAs, a new resource record is proposed to be added to DNS: ULID RR. A DNS Application Layer Gateway (ALG) located in P-SHIM6 box manages the host requests to DNS. DNS requests are processed by the DNS-ALG and both AAAA and ULID records are queried. The response is also processed by the DNS-ALG. CMULA is returned to the host in the form of an AAAA record and PA addresses are stored for future use. Adding DNS-ALG will make P-SHIM6 incompatible with DNSSEC [9] unless the DNS-ALG is security aware. If the DNS-ALG is not security-aware then it would be difficult, or even impossible, for a security-aware resolver to obtain or validate signed DNS data.

When a host sends out a packet, the P-SHIM6 box retains it and establishes a context with the remote SHIM6-enabled host/P-SHIM6 box. When the context is
established, it is used for processing outgoing/incoming packets. SHIM6 contexts are maintained and managed by P-SHIM6 and hosts are not aware of them.

The reverse DNS tree of CMULAs should be populated properly because reverse DNS tree lookup is used for obtaining the locator set related to a CMULA. When a reverse lookup is performed on a CMULA, an FQDN is returned along with the locator set in an additional information field.

The SHIM6 uses CGA [18] or HBA [20] addresses to protect the binding between an ULID and its associated locator set. Therefore, the addresses which are assigned to hosts within a SHIM6-enabled site should be CGA or HBA. In the case of P-SHIM6, hosts do not run SHIM6 protocol so another component should generate CGA and HBA addresses and assign them to hosts. P-SHIM6 proposes adding a new component to the DHCP [45] server for this purpose. As a result, this approach is incompatible with stateless address autoconfiguration [118] since the addresses of the hosts within the multihomed site need to be configured using DHCP.

Hosts use CMULAs, and the P-SHIM6 box is the only entity in the site which knows and actually manages locators. Therefore, the P-SHIM6 box is capable of acting as an enforcement point for the site’s TE policies. Administrators can define their desired TE policies and the P-SHIM6 box can enforce those policies when it establishes new contexts or changes working address pairs.

Since all communications will go through the P-SHIM6 box, a failure in that can terminate all communications. To handle such failures, P-SHIM6 proposes a mechanism for preserving active communications and diverting them to a backup box in the case that the primary P-SHIM6 box has failed. To the best of our knowledge, there is no real implementation of P-SHIM6 available.

### 6.2.2 Extended SHIM6

Extended SHIM6 is an extension which provides complete identifier/locator separation as well as a method for packet rewriting in routers for traffic engineering. SHIM6 has an option, \textit{ULID-pair} option, which allows using a ULID, which is different from locators. All SHIM6 control messages support this option. Extended SHIM6 proposes using this option and assigning a provider independent identifier to each host. Such an identifier should not be bound to any particular interface and it is desirable that it has long-term stability so it can survive in the event of renumbering and mobility. Also, for scalability, it is recommended that the identifiers be allocated hierarchically. This simplifies lookups and guarantees uniqueness.

An inevitable requirement for traffic engineering in host-centric multihoming solutions is a mechanism that enables hosts to get some information about a site’s pref-
It is recommended that a site’s preferences be distributed via DHCPv6. Extended SHIM6 proposes a new DHCPv6 option for this purpose. The new option carries the list of a site’s IPv6 locator prefixes along with their preferences.

To enforce TE preferences, exit routers or routers close to them should rewrite source locators in packets. Communication is started using an FQDN. That FQDN is mapped to its corresponding identifier using DNS. There are two ways to implement the mapping mechanism. One way is using a new resource record for identifiers. Another solution is overloading AAAA records. The former would be inefficient unless most hosts have the new resource record. The latter should be backward compatible so that non-routable identifiers do not confuse legacy hosts that are not aware of overloaded records. The mapping also needs to have PTR records in the reverse DNS tree (ip6.arpa) which point to the host’s FQDN. Then looking up locators using the FQDN would be possible. It is recommended that SRV records [61] are used instead of AAAA records to benefit from SRV priority and weight attributes. By setting priority and weight, a site administrator can influence locator selection on the remote hosts if they use SRV records instead of AAAA records.

SHIM6 allows routers to rewrite locators if a payload extension header exists in the SHIM6 packet. Such rewrites are not permitted for SHIM6 control messages. The extensions proposed by extended SHIM6 allow routers to rewrite locators in any IPv6 packet. Two new SHIM6 options, *sent locator pair option* and *receiver locator pair option*, are proposed for this purpose. These options help SHIM6 hosts on both ends of the connection to realize how routers have changed the packets. Since some hosts may have a subset of their site’s prefixes, it is also recommended that I1 and R1 messages also carry a *locator list option* to prevent routers from using the locators which are not supported by the host. Implementing this feature needs all packets to carry SHIM6 headers even in the case that SHIM6 does not rewrite addresses.

### 6.3 NAROS: A Traffic Engineering Solution for Host-centric Multihoming

Name, Address and ROute System (NAROS) [79] is a service which aims to help hosts in a multihomed/multi-addressed site to choose the best source address for communicating to remote hosts. By controlling source address selection, sites will have more control over their interdomain traffic. Before transmitting the packets, hosts contact the NAROS server to determine which address should be used as source address. This approach tries to address load balancing and traffic engineering in multihomed sites which are using a host-centric multihoming solution. Load, status of the communica-
tion links and administrative policies are important factors that influence the source address selection process. The NAROS server takes all these parameters into account to choose the best source address for communicating with a specific remote host.

The NAROS approach proposes a protocol for communication between a NAROS server and hosts inside the site. This protocol runs over UDP and contains two messages:

1. **NAROS_REQUEST**: which goes from hosts to NAROS server when they want to start a communication with a remote host. It contains, at least, destination and host global addresses.

2. **NAROS_RESPONSE**: which is the NAROS server’s response and contains, at least, the best source, the destination prefix and a lifetime. It actually tells the client that the selected source is the best one for contacting all hosts matching the prefix. The response will be valid for the duration of the lifetime.

The result of the default source address selection mechanism [117] is arbitrary when a host has several global-scope IPv6 addresses as in the case of multiaddressed hosts. NAROS proposes that a multi-addressed host asks the NAROS service which source address to use whenever it needs to start a communication with a remote host. The NAROS service moves the source address selection process from hosts to a central service, so that site administrators will have more control over this central service and be able to set it up so that it considers TE policies in the process. NAROS does not enforce TE policies, it just recommends to hosts which source address will make a better match with TE policies. Policies are not pushed to hosts. Hosts should contact the NAROS server whenever they need the NAROS recommendations.

A NAROS server can be integrated with a DNS server to minimize the overhead and traffic imposed on the network. NAROS proposes two messages, **NAROS_DNS_REQUEST** and **NAROS_DNS_RESPONSE**, for this purpose. A host’s DNS client should be modified to wrap each DNS request with the NAROS request and send it, as a **NAROS_DNS_REQUEST**, to the NAROS server. When the NAROS server receives the request, it extracts and forwards the DNS request to the DNS server. When the response from DNS is received, the NAROS server examines the response and selects the best source address for contacting the remote host. The result is sent to the host in the form of a **NAROS_DNS_RESPONSE**. Coupling NAROS and DNS does not cause any problems for DNSSEC and the combination is still compatible with DNSSEC.

Choosing a proper source address will help sites to enforce egress TE policies. In other words, by selecting a suitable source address, TE policies can be enforced on the communications which are initiated by local hosts. For enforcing ingress TE policies,
on the flows which are initiated by remote hosts, NAROS offers limited capabilities. It proposes a coupling mechanism for the NAROS and DNS servers for this purpose. This idea is that the requests which are received by the site’s DNS server are processed by the NAROS server too. The NAROS server orders the list of locators, based on ingress TE policies, before sending it to the requester. This solution assumes that 1) remote hosts do not change the order of the list and 2) remote hosts always use DNS to contact local hosts. Neither of these assumptions can be guaranteed in practice.

When NAROS is used, hosts do not need to keep all of the TE policies and rules locally. They can query the NAROS server whenever and about whatever they need. This approach is dynamic and efficient. Hosts only need to deal with the policies that affect them. They always have access to the latest version of the policies, so they do not need to store and manage policies which might be used in the future or might not be used at all. NAROS is a stateless service, so it is possible to replicate the NAROS server in a site to increase availability of the service.

NAROS allows TE without making any change in the Internet routing system. The changes are limited to the hosts. It makes NAROS a suitable choice for TE when a host-centric multihoming solution, like SHIM6, is employed. C. Delanois [40] presents the results of evaluating performance of the NAROS server on real traffic. The results show that the average load on the NAROS server is about 35 requests per second and the bandwidth overhead is about 0.35%, which is reasonable.

In the next section, we propose SHIM6-enabled NAROS, which is a mechanism for employing NAROS in SHIM6-enabled networks.

### 6.4 Adopting NAROS in SHIM6-enabled Networks (SHIM6-enabled NAROS)

SHIM6 is a host-centric multihoming solution and does not provide a sufficient set of features for a site administrator to manage incoming and outgoing traffic to its site. Site administrators need to be able to define and enforce TE policies based on their limitations and requirements. NAROS is able to help hosts to increase their awareness of TE policies in the site so that they enforce the policies by choosing proper source and destination addresses for communicating to the hosts outside the site. SHIM6 has unique features which can provide a rich set of TE features for administrators in integration with NAROS. Two main features of SHIM6, which are useful in this case, are:

1. SHIM6 hosts can exchange their locator preferences when a SHIM6 context is created or at any time after that using update messages. These preferences are
valuable information and a SHIM6 host can report them to a NAROS server. They help the NAROS server to make more accurate decisions.

2. SHIM6 is able to change the current address pair, at any time, without breaking the communication. Using this feature, dynamic TE policies can be enforced. In other words, whenever a NAROS server is informed about a change in TE policies, it can notify SHIM6 hosts and they will enforce them transparently to the applications.

NAROS and SHIM6 can be extended to be able to communicate with each other and provide a set of efficient TE features for site administrators. The rest of this section describes our proposed mechanism which makes the communication between SHIM6-enabled hosts and NAROS possible. We call the mechanism SHIM6-enabled NAROS.

6.4.1 Reporting Locator Preferences to NAROS

Locator preferences are valuable bits of information that SHIM6 hosts obtain from the other end of their communications. These preferences describe TE policies at one end of the communication that should be considered by the other end. A NAROS server already knows about the local preferences. Having remote preferences helps the NAROS server to recommend local policies that are more compatible with the remote policies.

A SHIM6 host receives locator preferences in two cases: when a SHIM6 context is established and when an update message is received. There are two SHIM6 options, locator list option and locator preference option, which can be used in I2, R2, I2bis and update request messages. Locator list specifies the list of available locators on the host and locator preferences describes their preferences. Locator preferences are defined as a combination of priority and weight, similar to DNS SRV records [61]. Priority specifies the priority of the locator. The host expects the other end of the communication to try locators with higher priority first. In the case of two locators with the same priority, weight specifies which one should be tried first.

To enable hosts to include locator preferences in their requests for the NAROS server, we propose a pair of request/response messages to be added to the NAROS protocol. The format of the NAROS-SHIM6-REQUEST message is illustrated in Fig. 6.2. A new message type is defined in this request (NAROS_SHPM6_REQUEST=6). A SHIM6 host sends a list of locators and also locator preferences received from the remote SHIM6 host to the NAROS server. It also includes its own addresses, at the end of the request, as the list of available source addresses. The NAROS server can keep this information to be used in the future for responding to the requests that come from non-SHIM6 hosts for the same destination.
Upon receiving this request, the NAROS server tries to prioritize available address pairs so that they can satisfy TE policies at both ends. The format of the response message is illustrated in Fig. 6.3. The response contains an ordered list of acceptable address pairs. Address pairs that do not satisfy TE policies are not acceptable and will be removed from the list. This approach is a site-management option. It would be possible for all address pairs to be acceptable but the method allows some to be unacceptable. When the SHIM6 host receives the response from the NAROS server, it can change the current address pair, using the mechanism described in 6.4.2, if the NAROS server suggests using a different address pair for the communication. Communication over unacceptable paths should be stopped as soon as possible.

![Figure 6.2: The NAROS_SHIM6_REQUEST message format.](image)

Figure 6.2: The NAROS_SHIM6_REQUEST message format.

Fig. 6.4 shows a sample scenario when a SHIM6 host in a network, which is equipped with a SHIM6-enabled NAROS server, initiates a communication with a remote SHIM6 host. The diagram shows the messages that are exchanged between the SHIM6 hosts
and also between the SHIM6 host and the NAROS server. H1 is the SHIM6 host that starts the communication with H2. Before sending the first packet, it contacts the NAROS server using a NAROS_REQUEST message and asks about the source address that should be used. The NAROS server responds and proposes a source address for the communication. H1 starts the communication using the proposed address. After a while, assuming that it is a long-lived communication, SHIM6 is activated and creates a SHIM6 context for the communication. Context establishment messages are exchanged and after receiving an R2 message, H1 knows about H2’s locator preferences. H1 sends H2’s locator preferences to the NAROS server and asks for an ordered list of acceptable locator pairs for communicating with H2 using NAROS_SHIM6_REQUEST. The NAROS server replies, using NAROS_SHIM6_RESPONSE, and sends the ordered list back to H1. If the current address pair is not in the list, SHIM6 stops the communication and starts an exploration process using the list of acceptable address pairs. SHIM6 probe messages that are received on unacceptable paths are ignored. That guarantees that the exploration process will find an acceptable address pair, otherwise the communication will be dropped at the end.

6.4.2 Enforcing Dynamic TE Policies

SHIM6 is able to change the current address pair, at any time, without breaking the communication. This is a feature, which can be used by a NAROS server, to efficiently enforce dynamic TE policies. When the NAROS server is informed about a change in TE policies, it can broadcast that change in the site. SHIM6 hosts, which find the change relevant, apply the change according to the new policy. Fig. 6.5 shows the format of the NAROS_SHIM6_POLICY_CHANGE message. This message is broadcast by the NAROS server whenever a new TE policy is defined or an existing policy is changed. Each policy contains a destination prefix and the best source address for communicating with addresses within that prefix.

When a SHIM6 host receives a policy change notification, it starts a process that tries to enforce the change. The first step is to check whether the changes affect any communication on the host. It can be performed by comparing policy’s destination prefix with the destination addresses of all address pairs for all active SHIM6 contexts. The result can be one of the following three cases:

1. There is no SHIM6 context communicating with a remote host that has an IP address that belongs to one of the announced policies.

2. There is a SHIM6 context for which its remote host has an IP address that belongs to one of the announced policies but it is not used at the moment.
3. There is a SHIM6 context using one of the announced destination prefixes, but the current source address is not within the best source prefix for that destination. In case 1, the notification is irrelevant and should be ignored. In case 2, the policy does not affect the communication at the moment, but if the host decides to change the current address pair, due to a failure or receiving an update message later, it should contact the NAROS server to get the ordered list of address pairs according to the latest version of the TE policies. The context should be marked as out-of-date so that SHIM6 remembers that it should be updated before making any change in the current address pair in the future.

In case 3, a change in the current address pair is required. First, the SHIM6 host should contact the NAROS server and send a NAROS_SHIM6_REQUEST to get the new ordered list of acceptable address pairs, according to the latest changes in TE policies. If the current address pair is not in the list, which means it is not acceptable, then it should check reachability of the address pairs on the list, using the reachability
exploration procedure described in [10]. If any of those address pairs are reachable, it should change the current address pair in the SHIM6 context. It should also report the change to the remote SHIM6 host by sending an update message that includes new priorities in its locator preferences option field.

Fig. 6.6 shows a scenario where a policy change notification is broadcast in the network. H1 and H2, two SHIM6 hosts, have already established a communication and are exchanging data. The NAROS server broadcasts a policy change notification using a NAROS_SHIM6_POLICY_CHANGE message. Upon receiving the notification, H1 checks whether it affects the current communication with H2. The scenario assumes that it does. Then H1 contacts the NAROS server using a NAROS_SHIM6_REQUEST and asks for an ordered list of acceptable address pairs. The NAROS server responds with the list. New locator preferences are reported to H2 using a SHIM6 update message. If the current address pair is not in the list, H1 stops the communication and starts its exploration process using the ordered list of acceptable address pairs received from the NAROS server. SHIM6 probe messages that are received on unacceptable paths are ignored. This guarantees that the exploration process will find an acceptable address pair or the communication will be dropped at the end of the process.
6.5 SHIM6-enabled NAROS vs Proxy SHIM6 and Extended SHIM6

Table 6.1 summarizes characteristics of our proposed solution and also two other existing TE solutions for SHIM6.

Proxy SHIM6 and SHIM6-enabled NAROS need to be added to the network as two new services. Proxy SHIM6 and Extended SHIM6 propose some changes in DHCP and DNS. SHIM6-enabled NAROS does not need any change in any other services in the network.

SHIM6-enabled NAROS and Extended SHIM6 propose some changes in SHIM6 protocol. Proxy SHIM6 needs to change the DNS client, which runs on hosts. Therefore, all three solutions need some changes in hosts.

To deploy Extended SHIM6, routers should change to be able to do address rewriting in the packets. SHIM6-enabled NAROS and Proxy SHIM6 do not need any change in the routing system.

All solutions are able to control the traffic when communication is started by a host inside the site (Egress TE).

For communications initiated by remote hosts, only SHIM6-enabled NAROS is
equipped with a mechanism for enforcing TE policies (Ingress TE). Proxy SHIM6 and Extended SHIM6 rely on DNS and remote hosts and do not provide any mechanism for enforcing TE policies in this case.

SHIM6-enabled NAROS is the only solution that proposes a mechanism for enforcing changes in TE policies to already established communications.

Proxy SHIM6 proposes adding a new resource record to DNS and Extended SHIM6 recommends using SRV records instead of AAAA records. They also need the reverse DNS tree to be populated properly. SHIM6-enabled NAROS does not rely on DNS and does not need any change in DNS behaviour or structure.

SHIM6 hosts are hidden behind Proxy SHIM6. As a result, firewalls are not able to apply filters on source addresses unless the firewall is placed behind the Proxy SHIM6 server. Extended SHIM6 needs exit routers, or routers close to them, to rewrite addresses in the packets. If the firewall is located after the router which does the rewriting, then the same problem for Proxy SHIM6 can occur to Extended SHIM6 as well. SHIM6-enabled NAROS is as compatible as original SHIM6 with firewalls (refer to section 4.7 for more information).

Proxy SHIM6 does not need the SHIM6 protocol to be installed on hosts, so all hosts will benefit from SHIM6 features; but it is incompatible with stateless address autoconfiguration and needs to have a DHCP server in the site to generate CGA or HBA addresses. This affects backward compatibility of this approach. Extended SHIM6 needs all packets to carry SHIM6 headers and DNS SRV records be used instead of AAAA records. It can cause some problems for legacy hosts that are not aware of these changes. Legacy hosts are not able to benefit from SHIM6-enabled NAROS, but they still can do their normal operations in the network.

Proxy SHIM6 needs to keep state for each long-lived communication. Also, all long-lived communications should go through the Proxy SHIM6 which makes it a performance bottleneck in the system and affects scalability. SHIM6-enabled NAROS is as scalable as original NAROS. The proposed extensions do not have any negative impact on scalability. A TE policy, at the worst, affects the traffic in a site just like an ISP outage. The results presented in 4.10.2 show that the traffic that is generated by SHIM6 in this case, even in a large scale network, is manageable.
Table 6.1: Summary of characteristics of SHIM6-enabled NAROS and two other existing TE solutions for SHIM6.

6.6 Conclusion

In this chapter we proposed a mechanism that provides TE facilities for a SHIM6-enabled network. The proposed system, SHIM6-enabled NAROS, extends SHIM6 and NAROS protocols to enable them to communicate with each other. The proposed extension adds some useful features to NAROS and SHIM6 while it keeps all their original advantages. The changes are still limited to the hosts inside the multihomed site. It is backward compatible and legacy hosts can still work in the network without any problem. SHIM6-enabled NAROS does not keep any state; therefore, it is as scalable as the original version. In addition to these advantages, SHIM6-enabled NAROS makes ingress TE and dynamic enforcement of the TE policies possible for SHIM6-enabled sites. Administrators are able to control communications that are initiated by remote hosts. Local hosts, which are the destination of such communications, can change the working path in order to enforce TE policies. This change can happen at any time, not only at the start of the communication, which makes the enforcement of dynamic TE policies possible. SHIM6 monitors all communications, therefore TE policies are enforced to all communications no matter how they have been initiated.

The combination of SHIM6-enabled NAROS server and SHIM6-enabled hosts creates a mechanism for enforcing TE policies. A SHIM6-enabled NAROS server proposes the best address pair for a communication and the SHIM6-enabled host involved in the communication enforces the decision. Such an enforcement will be transparent to the application and transport layer except for the case that TE policies at the two ends of communication are in conflict. In that case, the communication gets terminated.

Cedric Launois [40] did an evaluation of NAROS using a 24 hours trace from a
network with around 8000 hosts. His analysis showed that the average load on the NAROS server was 35 requests per second. Our proposed solution, in worst case, doubles the load on the NAROS server which means that in a similar site the average load can reach up to 70 requests per second. In comparison, the average load on one of the root DNS servers in a typical day is 442 requests per second [123]. Thus, an average load of 70 requests per second is still reasonable and can be handled by a SHIM6-enabled NAROS server in a large site.
Chapter 7

Conclusion

Contents

<table>
<thead>
<tr>
<th>7.1</th>
<th>Results and Contributions</th>
<th>146</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1.1</td>
<td>Which solution can be the winner from a deployability viewpoint?</td>
<td>146</td>
</tr>
<tr>
<td>7.1.2</td>
<td>How efficient is REAP in handling communication path failures and how can it be improved?</td>
<td>146</td>
</tr>
<tr>
<td>7.1.3</td>
<td>How efficient is SEMICOUPLED in handling communication path failures?</td>
<td>148</td>
</tr>
<tr>
<td>7.1.4</td>
<td>How does the number of subflows and the characteristics of in-use communication paths affect the throughput of MPTCP?</td>
<td>148</td>
</tr>
<tr>
<td>7.1.5</td>
<td>How can SHIM6 be improved or integrated with another solution to provide a rich set of traffic engineering features for site administrators?</td>
<td>149</td>
</tr>
</tbody>
</table>

| 7.2 | Future Work | 149 |
In this chapter, we summarize our results and contributions and propose some areas for future work.

7.1 Results and Contributions

In this section we summarize our findings and contributions. Research questions are restated and our answers are summarized.

7.1.1 Which solution can be the winner from a deployability viewpoint?

Multihoming is a mechanism which provides computer sites with failure resistance and performance. The current solution of Multihoming, used in the IPv4 Internet, is based on BGP features and has a potential scalability issue. This issue makes it inappropriate for the IPv6 Internet due to the huge address space provided by IPv6 and the growing demand for multihoming. This is a known issue and a wide range of solutions for IPv6 multihoming have been proposed during past years. The first contribution of this thesis is an analysis of seven active proposed solutions in this area. We tried to find out whether any of those solutions can be the winner from a deployability viewpoint. Our analysis showed that none of the solutions satisfy all requirements in order to be adopted as the final solution, but all of them have areas for improvements and still have a chance of being chosen as the final solution or a part of that.

7.1.2 How efficient is REAP in handling communication path failures and how can it be improved?

Host-centric solutions look more attractive from a deployability viewpoint because they do not need any change in the Internet routing system, which makes deployment easier. SHIM6 is one of the proposed host-centric solutions for IPv6 multihoming. It only needs the host’s protocol stack to be updated which is the minimum change, even among other host-centric solutions. It is backward compatible and incrementally deployable and provides a specific mechanism, named REAP, for failure detection and recovery.

The second contribution of this thesis is a performance evaluation of REAP by conducting a set of experiments in real environments (over the Internet with four address pairs and in the lab with more than four address pairs) and also by using a set of large-scale simulation experiments. The aim of the experiments was to identify how effective REAP is in handling communication path failures, and how it can be improved. The results of this work, presented in chapter 4, can be summarized as follows:
• Not only the behaviour of applications but also the transport layer protocol can affect the behaviour of REAP in case of path failure. That means that running the same application over different transport layers could cause different reactions by REAP leading to a different recovery time and traffic.

• Exploration time dramatically increases when the number of address pairs exceeds nine. Experiments with 16 address pairs showed very poor results in the lab, so nine address pairs appears to be an upper limit for REAP.

• The exploration process can generate a large amount of traffic at the beginning of the process but that does not cause a major problem for large-scale networks with high bandwidth links to the Internet.

• Deferring the exponential backoff algorithm (e.g. by increasing the number of initial probes) and sending initial probes in a burst can improve recovery time. These modifications increase traffic but not at the level that can cause a major problem for large sites. Such improvements can be beneficial for large sites especially if they expect to deal with more than nine address pairs.

• A signaling mechanism which enables REAP to notify the transport layer at the end of the exploration process can improve application recovery time. Also a signaling mechanism which enables the transport layer to notify REAP when it detects a path failure can improve failure detection time. Such mechanisms destroy the transparency of SHIM6 to the transport layer but they can significantly improve recovery time and would be beneficial for interactive applications such as web browsers or media streaming applications. Although the idea of such signaling mechanisms is not new, applying them to the TFRC congestion control algorithm and showing how they affect its performance in practice is a contribution of our work.

• The experiments over the Internet revealed some operational issues. One issue is that some firewalls drop SHIM6 packets because they do not know the SHIM6 extension header. This issue is not specific to SHIM6; [33] proposes a solution for this. Another issue is enabling SADR on the site’s edge routers. We found it difficult to convince site administrators to enable SADR because of its performance penalties. This issue also is not specific to SHIM6 and may affect other host-centric solutions as well.
7.1.3 How efficient is SEMICOUPLED in handling communication path failures?

MPTCP is another host-centric solution which needs changes only in the host’s protocol stack. Similar to SHIM6, it is incrementally deployable and equipped with a congestion control mechanism (SEMICOUPLED) which is able to handle communication path failures. Another contribution of this thesis, presented in chapter 5, is a performance evaluation on SEMICOUPLED using a simulation model. The aim of the simulation experiments is to investigate how efficient SEMICOUPLED is in handling communication path failures:

- In case of permanent failure, increasing the number of subflows decreases the recovery time and enables MPTCP to stabilize throughput faster. Permanent path failure refers to a failure that never recovers, at least for the duration of the communication.

- In case of temporary failure, SEMICOUPLED utilizes the recovered path and restores the throughput quickly. Temporary failure refers to a path failure that is recovered after a short period of time. A severe congestion or a temporary error in a path can cause this kind of failure.

We can conclude from the results that MPTCP demonstrates a good performance in detecting and recovering path failures.

7.1.4 How does the number of subflows and the characteristics of in-use communication paths affect the throughput of MPTCP?

By extending our model of MPTCP, we conducted more experiments to throw some light on the effect of number of subflows and characteristics of communication paths on the throughput of MPTCP. Interesting findings can be summarized as follows:

- Increasing the number of subflows does not necessarily increase the throughput in the same ratio unless there is no packet loss, which is an unrealistic assumption for real environments.

- The RTT of a path has more effect on throughput than its loss rate. Smaller loss rate leads to a larger congestion window, but it does not necessarily lead to higher throughput. Thus, if there is a choice between two paths, the path with a lower RTT will be a better choice.
• When there is a shared channel between subflows, SEMICOUPLED is conservative in increasing congestion windows, which leads to better throughput but only in the long run. In the short run, having less subflows may provide higher throughput.

• SEMICOUPLED is able to handle frequent changes in loss rate properly and provide a stable throughput. That means that MPTCP is a good choice for sites with varying loss rate links like satellite links.

• Measuring loss rate is a challenging task. We found that $\alpha$ is a good indicator for loss rate. $\alpha$ is a parameter in the SEMICOUPLED algorithm that controls how aggressively to increase congestion window size. Higher loss rate causes more variation in $\alpha$. Monitoring changes in $\alpha$ can help SEMICOUPLED to evaluate loss rate of in-use paths.

7.1.5 How can SHIM6 be improved or integrated with another solution to provide a rich set of traffic engineering features for site administrators?

Lack of a site-aware TE mechanism is one of the main weak points of SHIM6. To address this issue, we propose a mechanism for traffic engineering in SHIM6-enabled sites in chapter 6. The mechanism is an extension to SHIM6 and NAROS that enables them to communicate with each other. The mechanism is backward compatible, scalable and makes ingress TE and dynamic enforcement of the TE policies possible in a SHIM6-enabled site.

7.2 Future Work

There are some areas in our study that have potential for improvement or further investigation.

The mechanism proposed in chapter 6 targets SHIM6. The mechanism proposes having a service, like NAROS, in a SHIM6-enabled site which is aware of TE policies. That proposal extends NAROS and SHIM6 protocols so that SHIM6-enabled hosts and the NAROS server are able to share useful information about local and remote TE policies. MPTCP, similar to SHIM6, suffers from lack of a rich set of TE features. Since there are similarities between SHIM6 and MPTCP, i.e. both are able to exchange information about available paths and both are able to change communication paths on the fly, it seems that the mechanism has potential to be used for MPTCP-enabled
sites as well. Investigating how the mechanism can be extended or modified to be used for MPTCP is an interesting open research question.

In chapter 4, we evaluated some improvements to REAP and TFRC using simulation. Implementing the improvements and evaluating them in a real environment would be valuable.

In chapter 4, two congestion control algorithm are considered in the experiments and improvements: TCP congestion control algorithm and TRFC. There are other algorithms for congestion control; e.g. DCCP can be configured to use TCP-like congestion control (CCID2) algorithm. Investigating which protocols and applications can benefit from SHIM6 and how the proposed improvements can be adapted for their associated congestion control algorithms is an area for future research.

In chapter 5, we investigated the behaviour of SEMICOUPLED using a simulation model. Conducting real experiments in the lab and over the IPv6 Internet, similar to what we did for SHIM6, could lead to more realistic results. To be able to run such experiments, a test bed should be designed and implemented. The test bed should be able to artificially generate paths with different delay and loss rate and simulate temporary and permanent path failures. A logging mechanism is also required to log important internal MPTCP information. Designing a controllable test environment and running experiments, similar to the simulation experiments, will provide valuable data for research community.

SEMICOUPLED considers two parameters for calculating and adjusting congestion window size: RTT and loss rate. Measuring both of them are challenging. Our simulation results, in chapter 5, showed that using paths with high loss rate had a negative impact on the throughput. We found that $\alpha$ is a good indicator for loss rate. $\alpha$ is a parameter in the SEMICOUPLED algorithm that controls how aggressively to increase congestion window size. Higher loss rate causes more variation in $\alpha$. It seems that monitoring $\alpha$ can provide useful information for tuning SEMICOUPLED. How $\alpha$ should be monitored and how the collected information can be used to tune throughput is an interesting area for research.

We performed a performance evaluation on MPTCP in chapter 5 by focusing on throughput. There is another goal for MPTCP in addition to throughput: MPTCP flows should not be harmful to regular TCP flows. To be able to evaluate this factor, our model of MPTCP should be extended so that it simulates TCP flows and runs them with MPTCP subflows on the same path. By adding this feature to the model, it would be possible to evaluate MPTCP from the ‘do no harm’ aspect as well.
Bibliography


[34] B. Carpenter and Y. Rekhter. Renumbering needs work. *Internet RFC 1900*, 1996.


[107] J. Ronan and J. McLaughlin. An empirical evaluation of a Shim6 implementa-
tion. In Mobile Networks and Management. Second International ICST Confer-
ence, MONAMI 2010, Santander, Spain, September 22-24, 2010, Revised Selected
Papers, 2012.

picture archiving and communication network using stochastic activity networks.

[109] D. Saucez, L. Iannone, and O. Bonaventure. OpenLISP: An open source imple-
mentation of the locator/ID separation protocol. In Demo Session of the IEEE

[110] D. Saucez and V. Nguyen. LISP-Click: A click implementation of the locator/ID
separation protocol. 1st Symposium on Click Modular Router, November 2009.

Conference on, 2005.

[112] M. Scharf and A. Ford. Multipath TCP (MPTCP) application interface consider-

[113] A. Sedaghatbaf and M. Abdollahi Azgomi. Attack modelling and security evalu-

level network striping for data intensive applications using high speed wide area
(CDROM), Supercomputing '00, 2000.


[116] M. Stiemerling, J. Quittek, and L. Eggert. NAT and firewall traversal issues of
host identity protocol (HIP) communication. Internet RFC 5205, 2008.


[119] O. Troan and L. Colitti. IPv6 multihoming with source address dependent routing
(SADR). Internet Draft (work in progress), 2013.


