MULTIHOMING WITH ILNP IN FREEBSD

Bruce Simpson

A Thesis Submitted for the Degree of PhD at the University of St Andrews

2016

Full metadata for this item is available in St Andrews Research Repository at:
http://research-repository.st-andrews.ac.uk/

Please use this identifier to cite or link to this item:
http://hdl.handle.net/10023/8681

This item is protected by original copyright

This item is licensed under a Creative Commons Licence
Multihoming with ILNP in FreeBSD

Thesis by
Bruce Simpson

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

University of St Andrews
School of Computer Science

2016
(Defended 17th December 2015)
Declaration

I, Bruce Simpson, certify that this thesis, which is approximately 40,000 words in length, has been written by me, that it is the record of work carried out by me, or principally by myself in collaboration with others as acknowledged, and that it has not been submitted in any previous application for a higher degree.

I was admitted as a research student in October, 2011 and as a candidate for the degree of Doctor of Philosophy in November, 2014; the higher study for which this is a record was carried out in the University of St Andrews between 2011 and 2015.

Date ................ Signature of candidate .................................

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of Doctor of Philosophy in the University of St Andrews and that the candidate is qualified to submit this thesis in application for that degree.

Date ................ Signature of supervisor .................................
In submitting this thesis to the University of St Andrews, I understand that I am giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. I also understand that the title and the abstract will be published, and that a copy of the work may be made and supplied to any bona fide library or research worker, that my thesis will be electronically accessible for personal or research use unless exempt by award of an embargo as requested below, and that the library has the right to migrate my thesis into new electronic forms as required to ensure continued access to the thesis. I have obtained any third-party copyright permissions that may be required in order to allow such access and migration, or have requested the appropriate embargo below.

The following is an agreed request by candidate and supervisor regarding the publication of this thesis:

- No embargo on print copy.
- No embargo on electronic copy.

Date ............... Signature of candidate .................................

Date ............... Signature of supervisor .................................
Acknowledgements

I owe my deepest gratitude to my supervisor Prof. Saleem Bhatti for his support and encouragement throughout. Thanks also to: Alexei Blinov, Graeme Burnett, Daniel Loizos & Tom McHugh. Finally, I must thank Cisco Systems, Inc. for their generous support of this work.
Published Research

The reader should note that while the major contributions in the design, analysis and implementation of these works were my own, I acknowledge the contribution and guidance of my fellow authors.

Abstract

Multihoming allows nodes to be multiply connected to the network. It forms the basis of features which can improve network responsiveness and robustness; e.g. load balancing and fail-over, which can be considered as a choice between network locations. However, IP today assumes that IP addresses specify both network location and node identity. Therefore, these features must be implemented at routers.

This dissertation considers an alternative based on the multihoming approach of the Identifier Locator Network Protocol (ILNP). ILNP is one of many proposals for a split between network location and node identity. However, unlike other proposals, ILNP removes the use of IP addresses as they are used today. To date, ILNP has not been implemented within an operating system stack.

I produce the first implementation of ILNP in FreeBSD, based on a superset of IPv6 – ILNPv6 – and demonstrate a key feature of ILNP: multihoming as a first class function of the operating system, rather than being implemented as a routing function as it is today.

To evaluate the multihoming capability, I demonstrate one important application of multihoming – load distribution – at three levels of network hierarchy including individual hosts, a singleton Site Border Router (SBR), and a novel, dynamically instantiated, distributed SBR (dSBR). For each level, I present empirical results from a hardware testbed; metrics include latency, throughput, loss and reordering. I compare performance with unmodified IPv6 and NPTv6. Finally, I evaluate the feasibility of dSBR-ILNPv6 as an alternative to existing multihoming approaches, based on measurements of the dSBR’s responsiveness to changes in site connectivity.
We find that multihoming can be implemented by individual hosts and/or SBRs, without requiring additional routing state as is the case today, and without any significant additional load or overhead compared to unicast IPv6.
Contents

List of Figures xix

List of Tables xxiii

Chapter 1. Introduction 1

1.1. What is multihoming? 1
1.1.1. Limitations of host-based approaches 1
1.1.2. Alternative approaches 2
1.1.3. Identifier-Locator Split Architectures 2
1.2. Dissertation structure 3
1.2.1. Terminology 3
1.3. Contribution 3
1.4. On networking stacks as models 4

Chapter 2. Problem background 7

2.1. Introduction 7
2.2. Multihoming 7
2.2.1. Names, Addresses & Routes 9
2.2.1.1. Origins 9
2.2.1.2. Addresses in IPv4 10
2.2.1.3. Changes in IPv6 10
2.2.2. IP address entanglement 11
2.2.3. The Identifier-Locator Split 12
2.2.4. Summary 12
2.3. The problem of routing table bloat 12
2.3.1. How multihoming introduces bloat 13
2.3.2. Provider-independent addresses 14
2.3.3. Backbone router constraints 15
2.3.4. Effects on the global IP routing table 16
2.3.5. The role of NAT 17
2.3.6. Summary 18
2.4. Solution space 18
2.4.1. General approaches 18
2.4.2. ILSAs: Identifier-Locator Split Architectures 19
2.4.3. How IPv6 changes routing & addressing 20
2.4.3.1. The non-deprecation of NAT 22
2.4.3.2. The non-abolition of PI addresses 22
2.4.3.3. How IPv6 affects ILSAs 23
2.4.4. Summary 23
2.5. Chapter summary 24

Chapter 3. Approaches to IP multihoming 27
3.1. Introduction 27
3.1.1. Chapter structure 27
3.1.2. Terminology 27
3.2. Overview of proposals 28
3.2.1. Deployment & impact 28
3.2.2. Security 29
3.2.3. Privacy 30
3.3. Proposals: Part I—Historical & related work 30
3.3.1. GSE/8+8: Global, Site, and End-system Addresses 30
3.3.2. Nimrod 31
3.3.3. MHMP: Multi-Homed with Multi-Prefix 31
3.4. Proposals: Part II—Site multihoming 31
3.4.1. NPTv6: IPv6-to-IPv6 Network Prefix Translation 32
3.4.1.1. Method 32
3.4.1.2. Caveats 32
3.4.1.3. Summary 32
3.4.2. LISP: Locator/Identifier Separation Protocol

3.4.2.1. Method

3.4.2.2. Caveats

3.4.2.3. Summary

3.5. Proposals: Part III—Host multihoming

3.5.1. SHIM6: Level 3 Multihoming Shim Protocol for IPv6

3.5.1.1. Method

3.5.1.2. Caveats

3.5.1.3. Summary

3.5.2. HIP: Host Identity Protocol

3.5.2.1. Method

3.5.2.2. Caveats

3.5.2.3. Summary

3.5.3. MP-TCP: Multipath TCP

3.5.3.1. Method

3.5.3.2. Caveats

3.5.3.3. Summary

3.5.4. SCTP: Stream Control Transmission Protocol

3.5.4.1. Method

3.5.4.2. Caveats

3.5.4.3. Summary

3.6. Proposals: Part IV—ILNP

3.6.1. Method

3.6.1.1. Flows & transport layer binding

3.6.1.2. Site-controlled multihoming

3.6.1.3. ILCC: ILNP Communication Cache

3.6.1.4. Security

3.6.2. Caveats

3.6.3. Summary

3.7. Summary of Proposals: Parts I—IV

3.8. Methodology
3.8.1. Metrics  
3.8.1.1. Routing state allocation  
3.8.2. Factors  
3.8.2.1. On Ethernet frame size  
3.8.3. Controls  
3.8.3.1. Limitations of iperf for multipath measurements  
3.8.3.2. On iperf performance limits  
3.8.3.3. On packet pacing error in bwping  
3.8.4. Summary  
3.9. Chapter summary  

Chapter 4. Host multihoming with ILNP  
4.1. Introduction  
4.1.1. Contribution  
4.1.2. Chapter structure  
4.1.3. Problem space  
4.1.4. Approach  
4.2. Method: Part I—Naming and Locators  
4.2.1. Identifier-Locator Vectors  
4.2.2. Local identifiers  
4.2.2.1. Implementation  
4.2.3. Routing prefixes for locator values  
4.2.3.1. Locator precedence  
4.2.4. Peer I-LV discovery  
4.2.4.1. Multiple remote L64 values  
4.2.5. Extended /etc/hosts syntax  
4.3. Method: Part II—Network Layer  
4.3.1. Flow initiation  
4.3.2. On ILNP flows & upper layer protocols  
4.3.3. Locator selection & rewriting  
4.3.3.1. Implementation  
4.3.4. Locator updates
4.3.5. Re-use of IPv6 unicast FIB 65
4.3.5.1. Implementation 65
4.3.6. Concurrency control 66
4.3.7. Security 66
4.4. Method: Part III—Sockets 67
4.4.1. Location-independent sockets 67
4.4.1.1. Binding dynamicity 67
4.4.2. ICMPv6 and UDP – in ILNPv6 68
4.4.3. Compatibility with IPv6 applications 70
4.4.3.1. Naming 70
4.4.3.2. Implementation 71
4.4.4. On IP bits used within applications 72
4.4.4.1. Implementation 72
4.4.5. Summary of Method: Parts I—III 72
4.5. Evaluation 74
4.5.1. Scope 74
4.5.2. Scalability analysis 74
4.5.3. Experiment design 74
4.5.4. Configuration 75
4.5.5. Metrics 75
4.6. Results 77
4.6.1. Data presentation 77
4.6.2. Stream measurements 77
4.6.3. Reordering distribution 86
4.6.4. Memory overhead 87
4.7. Chapter summary 91

Chapter 5. Site-controlled multihoming 93
5.1. Introduction 93
5.1.1. Contribution 93
5.1.2. Chapter structure 93
5.1.3. Problem space 94
5.1.4. Approach 94
5.2. Method 95
5.2.1. Marking ILNPv6 with IPv6 Flow Labels 95
5.2.2. ILNPv6 Forwarding 96
5.2.3. ILCC flow state at SBRs 97
5.2.3.1. Implementation 97
5.2.4. Locator Rewriting 98
5.2.4.1. Policies 98
5.2.4.2. Reversibility 98
5.2.4.3. Implementation 99
5.2.5. Locator Update snooping 99
5.2.5.1. Implementation 100
5.2.6. Re-use of IPv6 & Host-based ILNPv6 code 100
5.2.7. Concurrency control 100
5.2.8. Summary 100
5.3. Evaluation 101
5.3.1. Scope 101
5.3.2. Scalability analysis 101
5.3.3. Experiment design 102
5.3.4. Configuration 102
5.3.4.1. NPTv6 implementation 103
5.3.4.2. Static vs. dynamic site multihoming 103
5.3.5. Summary 104
5.4. Results 104
5.4.1. Stream measurements 104
5.4.1.1. Loss 105
5.4.2. Reordering distribution 115
5.4.3. Memory overhead 116
5.5. Chapter summary 120

Chapter 6. Distributed site multihoming 121
6.1. Introduction 121
6.1.1. Contribution 121
6.1.2. Chapter structure 121
6.1.3. Problem space 122
6.1.4. Approach 122
6.2.1. ILNPSync protocol 123
6.2.2. IIP elections 124
6.2.3. Peer locators 124
6.2.4. ILNPSync syntax 125
6.2.5. Security 126
6.2.6. Management interface 126
6.3. Method: Part II—Distributed Forwarding 128
6.3.1. dSBR-ILNPv6 forwarding 128
6.3.2. Multicast hand-over 128
6.3.3. Concurrency & optimisation 129
6.3.4. Summary of Method: Parts I—II 129
6.4. Evaluation 130
6.4.1. Scope 130
6.4.2. Scalability analysis 130
6.4.3. Experiment design 130
6.4.3.1. Induced link failures 131
6.4.3.2. On IIP failures 132
6.4.4. Time line 132
6.4.5. Metrics 133
6.4.6. Configuration 133
6.5. Results 134
6.5.1. Aggregate stream results 134
6.5.2. Time-series stream results 134
6.5.3. Reordering distribution 142
6.5.4. dSBR responsiveness 145
6.5.5. Memory overhead 146
Chapter 7. Conclusions

7.1. Summary 153
7.2. Limitations of this work 155
7.3. On reordering 156
7.4. Further work 158
7.4.1. Edge deployment 158
7.4.2. On the legacy use of IP addresses by applications 159
7.4.3. Lightweight Network Function Virtualisation 160
7.5. Closing remarks 161

Appendix A. Detailed results 163
List of Additional Figures 163

Appendix B. Experimental Apparatus 193
B.1. Development environment 193
B.2. Data collection & analysis 194
B.2.1. Data collation 195
B.2.2. iperf measurement details 197
B.2.3. On bwping measurements & error 198
B.2.4. Other differences between bwping & iperf 198
B.3. Evaluation environment 199
B.3.1. Hardware configuration 199
B.3.2. Network configuration 200
B.3.3. Local timekeeping 201
B.3.4. Network timekeeping 203
B.4. FreeBSD IPv6 RA behaviour 204
B.5. ILNP Experiment configuration 204
B.5.1. ILNP Policy Configuration 204
B.5.2. Example pf configuration 206
B.5.3. ILNP sysctl MIB variables 206
List of Figures

2.2.1 Diagram of common case for IP site multihoming 8
2.2.2 Diagram of host multihoming – with IPv6 MHMP from RFC7157 8
2.3.1 Diagram of a typical site multihoming scenario 13
4.1.1 Diagram of host multihoming scenario 54
4.2.1 Diagram comparing I-LVs with IPv6 addresses 56
4.2.2 Syntax of ILNPv6 /etc/hosts entries 60
4.3.1 Flowchart for the locator selection procedure 64
4.3.2 Relationship of ILCC to IPv6 unicast FIB 65
4.3.3 Protocol diagram of ILNPv6 Nonce Option 66
4.4.1 Call graph, two-way ICMPv6-in-ILNPv6 at hosts 68
4.4.2 Call graph, UDP-in-ILNPv6 transmission at hosts 70
4.5.1 Network diagram: Host locator selection experiment 75
4.6.1 Host Locator Selection, % ICMPv6 Goodput 79
4.6.2 Host Locator Selection, % UDPv6 Goodput 80
4.6.3 Host Locator Selection, ICMPv6 Median RTT 81
4.6.4 Host Locator Selection, ICMPv6 Mean RTT 82
4.6.5 Host Locator Selection, % UDP Reordering 83
4.6.6 Host Locator Selection, % UDP Errors 84
4.6.7 Host Locator Selection, UDP Jitter 85
4.6.8 Locator Selection, Frequency of sequence displacement 87
4.6.9 Locator Selection, buffer overhead at ICMPv6 source 89
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6.10</td>
<td>Locator Selection, buffer overhead at ICMPv6 destination</td>
<td>90</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Conceptual diagram: site-controlled multihoming</td>
<td>94</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Call graph, Native forwarding of ILNPv6</td>
<td>96</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Call graph, ILNPv6 Locator Update Snooping</td>
<td>99</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Network diagram: Site multihoming experiment</td>
<td>102</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Example pf configuration for NPTv6 translation</td>
<td>103</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Locator Rewriting, % ICMPv6 Goodput</td>
<td>107</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Locator Rewriting, % UDPv6 Goodput</td>
<td>108</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Locator Rewriting, ICMPv6 Median RTT</td>
<td>109</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Locator Rewriting, ICMPv6 Mean RTT</td>
<td>110</td>
</tr>
<tr>
<td>5.4.5</td>
<td>Locator Rewriting, % UDP Reordering</td>
<td>111</td>
</tr>
<tr>
<td>5.4.6</td>
<td>Locator Rewriting, % UDP sequence errors</td>
<td>112</td>
</tr>
<tr>
<td>5.4.7</td>
<td>Locator Rewriting, % UDPv6 payload loss</td>
<td>113</td>
</tr>
<tr>
<td>5.4.8</td>
<td>Locator Rewriting, UDP Jitter</td>
<td>114</td>
</tr>
<tr>
<td>5.4.9</td>
<td>Locator Rewriting, Frequency of sequence displacement</td>
<td>116</td>
</tr>
<tr>
<td>5.4.10</td>
<td>Locator Rewriting, buffer overhead at first hop SBR</td>
<td>119</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Diagram of distributed site multihoming with ILNP</td>
<td>122</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Protocol diagram, ILNPSync common header</td>
<td>125</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Protocol diagram, ILNPSync BEAT payload</td>
<td>126</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Protocol diagram, ILNPSync state descriptor payload</td>
<td>127</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Network diagram: dSBR load sharing experiment</td>
<td>131</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Distributed rewriting, 3-way split, % UDPv6 Goodput (aggregate)</td>
<td>135</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Distributed rewriting, 3-way split, % UDPv6 Reordering (aggregate)</td>
<td>135</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Distributed rewriting, 3-way split, % UDPv6 Errors (aggregate)</td>
<td>136</td>
</tr>
<tr>
<td>6.5.4</td>
<td>Distributed rewriting, 3-way split, % UDPv6 payload loss (aggregate)</td>
<td>136</td>
</tr>
<tr>
<td>6.5.5</td>
<td>Distributed rewriting, 3-way split, % Load sharing (time series)</td>
<td>138</td>
</tr>
<tr>
<td>6.5.6</td>
<td>Distributed rewriting, 3-way split, % UDP Goodput (time series)</td>
<td>139</td>
</tr>
</tbody>
</table>
6.5.7 Distributed rewriting, 3-way split, % ICMPv6 packet loss (time series) 140
6.5.8 Distributed rewriting, 3-way split, % UDP payload loss (time series) 141
6.5.9 Distributed rewriting, 3-way split, Sequence displacement 143
6.5.10 Distributed rewriting, 3-way split, % UDP reordering (time series) 144
6.5.11 Distributed rewriting, peer activation: activation delay 145
6.5.12 Distributed rewriting, peer failover: internal failover delay 146
6.5.13 Distributed rewriting, peer failover: external signalling delay 146
6.5.14 Distributed rewriting, peer failover: mean internal & external delay 147
6.5.15 Distributed rewriting, buffer overhead at dSBR IIP 149
6.5.16 Distributed rewriting, buffer overhead at dSBR peer 1 150
6.5.17 Distributed rewriting, buffer overhead at dSBR peer 2 151

A.1.1 Locator Selection, Detailed sequence displacement 164
A.1.2 Locator rewriting, Detailed sequence displacement 166
A.1.3 Distributed rewriting, Detailed sequence displacement 168
A.1.4 Distributed rewriting, 3-way split, % Load sharing (time series) 169
A.1.5 Distributed rewriting, 3-way split, % UDP Goodput (time series) 175
A.1.6 Distributed rewriting, 3-way split, % UDP payload loss (time series) 181
A.1.7 Distributed rewriting, 3-way split, % ICMPv6 packet loss (time series) 187

B.2.1 Pseudo-code for iperf reordering and error totals 198
B.3.1 Photographs of testbed systems 200
B.5.1 Example configuration for distributed Locator Rewriting 205
B.5.2 Example pf configuration from Chapter 5 206
List of Tables

2.1 Names & addresses, as used in IP (and ILNP) ................................. 11
3.1 Multihoming proposals examined in Chapter 3 ............................... 28
3.2 Experiment load rates used throughout Chapters 4-6 ...................... 46
4.1 Host multihoming, summary loss statistics .................................. 78
4.2 Locator selection, memory overhead .......................................... 88
5.1 Locator rewriting, ILNPv6 memory overhead ................................ 118
5.2 NPTv6 prefix translation, pf memory overhead .............................. 118
6.1 Time line of a performance evaluation trial. .................................. 132
6.2 Dynamic rewriting, ILNPv6 peer memory overhead ......................... 148
B.1 Evaluation environment, system hardware specifications ................. 200
B.2 Evaluation environment, Intel I350 network adapter settings .......... 201
B.3 Evaluation environment, FreeBSD networking stack configuration .... 201
B.4 Evaluation environment, FreeBSD timing configuration .................. 202
B.5 Evaluation environment, clock_gettime() precision measurements ... 202
Chapter 1

Introduction

The growth of the Internet routing table is a significant problem. It is known that load balancing and multihoming are the two principal factors involved. Whilst much of this growth is at the edge of the network, it impacts upon the core network: the state required for these functions must be propagated throughout the routing infrastructure. In this dissertation, I shall show that it is possible to multi-home sites without introducing additional routing protocol state (or adding encapsulation).

1.1. What is multihoming?

A multihomed host (or site) is connected to more than one IP network – e.g. to provide network functions such as robustness through diverse paths (failover), or to extend network capacity (load sharing). However, with IP today, site multihoming (and its associated network functions) must be implemented within the routing infrastructure – either by sites themselves, or by Internet Service Providers (ISPs). This approach introduces additional overhead\(^1\) to the global IP routing system in the form of “routing bloat” [2]. It also disallows such functions from being directly implemented (or controlled) at hosts, as they do not generally participate in the routing system.

1.1.1. Limitations of host-based approaches. Given the problems inherent to implementing network functions in the routing system, it is reasonable to investigate how these functions might be engineered at hosts; several existing approaches are discussed at length in Chapter 3. However, in this solution space another problem is

\(^1\)Whilst this problem can be mitigated to some extent by the use of Network Address Translation (NAT), this introduces some problems of its own – also discussed in Sec. 2.3.
encountered: *IP entanglement*. Whilst IP applications continue to use the well-known TCP & UDP protocols, these protocols do not support multiple addresses.

So, applications using these protocols become “bound” to one specific link for the entire duration of the session, preventing the dynamic use of additional (or backup) connectivity. This problem occurs due to the significant overlap between node identity and network location, as represented by the way IP addresses have been used for over 35 years [3, 4], and is discussed at length in Section 2.2.2 on page 11.

1.1.2. Alternative approaches. Despite the problems inherent to host multihoming in IP, several protocols now exist which permit the use of multiple endpoint addresses (and, by extension, network paths) at the transport layer. Such concurrent multipath transport protocols are described at length in Chapter 3; however, such approaches do not address the issues posed by site multihoming & network policy.

Moreover, whilst the introduction of version 6 of the Internet Protocol – IPv6 – has expanded the size of available address space, the underlying systems of routing & addressing have not changed\(^2\) to reflect location independence. This creates issues related to the growth of the Internet; these issues are difficult to solve without changing the present system.

1.1.3. Identifier-Locator Split Architectures. The difficulties inherent to re-engineering the addressing & routing system – and the present IPv4 address shortage – have impeded efforts by the Internet research & engineering communities to resolve the problem of “routing bloat”. So, there has been renewed interest in Identifier-Locator Split Architectures (ILSAs), where network location and node identity are treated as logically separate; this is often referred to as the “ID/Loc” split.

Within an ILSA, multihoming – and its associated network functions – may be implemented as a choice between network locations. By simplifying path choice in this way, ILSAs may mitigate the “routing bloat” which otherwise arises due to multihoming [2]; this is discussed at length in Chapter 2. The ILSA studied in this dissertation – the Identifier-Locator Networking Protocol (ILNP) – has specific properties

---
\(^2\)IPv6 does, however, offer richer support for multihoming features; refer to Chapter 2.
which distinguish it from other proposals discussed in [2]; specifically, its ability to re-use existing IP routing & address systems.

1.2. Dissertation structure

I begin in Chapter 2 by describing how an “ID/Loc” split affects routing state for multihomed sites (and hosts). Routing protocol state is defined as: the contents of a common routing table within a given network topology, and the protocol exchanges used to maintain it. This includes the use of static routes and dynamic routing protocols; for the purposes of Internet routing, the protocol used is usually BGP.

After presenting some related work in Chapter 3, I show how ILNP may be used to realise host multihoming in Chapter 4. A practical demonstration is given using the example of a load sharing algorithm, and its performance (in comparison to IPv6) is empirically studied. In Chapter 5, I show how locator selection and locator rewriting may be extended to provide a site-wide networking function, and evaluate its performance as compared with a related proposal, NPTv6. I further refine this approach in Chapter 6, where I demonstrate a novel, dynamically instantiated, distributed routing process based upon ILNP locator rewriting. Finally, I conclude in Chapter 7 with a summary of my conclusions and directions for future work.

1.2.1. Terminology. I use the terminology of Locator and Identifier as defined [5] throughout this work. Additionally, to avoid confusion between the ILNP network layer and transport layers, I use the terminology of “ILNP flow”—and not “ILNP session” as used in [6].

1.3. Contribution

While ILNP is a radically different architecture from IP, the claim in this work is that it can be implemented within current code bases, rather than requiring a “clean-slate” approach. Its approach does not require support from network routers, “middleboxes”, or other network elements; it is, effectively, a policy applied to the IP forwarding plane, which can provide multihoming capabilities similar to those achieved through “tweaking the route exchange” in IP today [2, Sec. 3.1.3]. So, my work has yielded seven key research contributions:
(I) Throughout this work, I show that both load sharing and failover can be realised in terms of a simple choice between multiple network locators within ILNP;

(II) In addition, I demonstrate its implementation as a set of extensions to the well-known FreeBSD networking stack (for both hosts & routers), and without introducing additional state – i.e. routing bloat – to the IP routing system;

(III) Following from the assertion that on-path network elements need not be modified to use ILNP, I show that this is also possible at the top of the networking stack, by demonstrating how API translation can allow applications to use ILNP without requiring changes (with caveats: refer to Section 4.4.4 on page 72);

(IV) Although ILNP has been specified as a host-based protocol (refer to Section 3.6.1.2 on page 41), the work in Chapters 5—6 demonstrates how its approach may be extended to sites, and how the network functions it can provide may be conveniently managed at site level using policies;

(V) In Chapter 6, I demonstrate a novel approach to the dynamic provisioning of site network capacity through distributed locator rewriting. This provides an example of how it may be directly applied in the problem domain described in Chapter 2, whilst providing robustness comparable to existing approaches;

(VI) Moreover, I show how ILNP may be implemented efficiently in terms of the existing FreeBSD code base, by way of relative performance studies in Chapters 4—5, and measurements of runtime overhead in Chapter 6;

(VII) Finally, as a direct outcome of this work, I have also contributed to the experimental RFCs [6, 7, 8, 9] describing ILNP, by providing feedback in two key areas: its coupling to IP transport protocols (e.g. TCP & UDP), and its interworking with IPSEC encryption; discussed further in Chapters 3 & 4.

1.4. On networking stacks as models

Throughout this work, ILNP – and its feasibility as a solution for some of the issues discussed in Chapter 2 – has been studied as an implementation within the well-known
FreeBSD networking stack. The reader may reasonably ask why this approach was taken in preference to other approaches (e.g. overlay simulation), given the additional development work required. This can be considered in terms of two main points.

(A) Firstly, it satisfies the expectations of the existing research & engineering communities regarding rigour and applicability. Consider that ILNP has been proposed as an alternative to the routing and multihoming architectures discussed in Chapter 2. Given the wide impact on existing engineering practice that its adoption would entail (also discussed further in Chapter 7), it is important that research in this area follows the expectations of these communities, particularly those of the Internet Engineering Task Force (IETF) as this is an important forum for ILNP technical publications.

The IETF’s view on new protocols is perhaps best summarised in the quote “We believe in rough consensus and running code” [10, Slide 19]. Rough consensus already exists in the form of the recommendation made to it by the Internet Research Task Force (IRTF) in [5, Sec. 17.2] that ILNP be pursued as an evolution of the present IP architecture. So, one goal of this thesis is to present the running code for ILNP in FreeBSD, as the findings of this work support that recommendation.

(B) Secondly, it provides an empirical demonstration of certain claims made regarding the ILNP architecture – e.g. that it does not require the use of tunnelling\(^3\) (encapsulation) – a technique which introduces additional network overhead, or that it requires no changes to other on-path network elements – by studying its effects within an existing, widely used system.

This is important because ILNP interacts with several other entities beyond the host forwarding plane by design, including the sockets API and IPv6 Neighbour Discovery subsystems. The behaviour of these subsystems is often deliberately simplified (or even omitted) in simulations. Moreover, simulating ILNP within an overlay topology would – by definition – require the use of tunnelling.

\(^3\)This claim forms an important part of the IRTF recommendation in [5, Sec. 17].
In addition to supporting the central theory behind this dissertation, this approach also yields data and analyses which are directly relevant to its future deployment in FreeBSD & other operating systems – as discussed within my conclusions in Chapter 7. A pragmatic example of this involves the introduction of ILNP within existing IP socket-layer interactions – i.e. the API surface presented to network applications. The underlying concerns behind this are first identified in Chapter 2, and expounded on in Chapter 4 with a practical demonstration in the form of load sharing.
Chapter 2

Problem background

2.1. Introduction

In this chapter, I review the problem space for IP multihoming. I begin in Sec. 2.2 by reviewing how IP addresses are used today, and their role in multihoming. This is followed in Sec. 2.3 by reviewing how IP routing functions today, and the inefficiency which multihoming can introduce – i.e. in the form of routing table bloat. Together, these constitute the background theory of this dissertation. Continuing in Subsec. 2.4, I review possible solution approaches, and describe how the introduction of IPv6 influences the proposals described in Chapter 3. I conclude in Sec. 2.5 with a summary.

2.2. Multihoming

A multihomed host (or site) is connected to more than one IP network. It offers additional network functions beyond those possible with a single link: e.g. traffic engineering (in the form of load sharing) & robustness (in the form of failover).

With IP today, site multihoming can be implemented in the routing system. Fig. 2.2.1 illustrates a common case where a single IP site has two diverse paths to the Internet, i.e. upstream links to Internet Service Providers (ISPs), denoted by ISP_1 & ISP_2 respectively. Regardless of how these paths are used, the site border router (SBR) must advertise its site prefix^1 P_S to both ISPs. This contributes to the problem of routing table bloat, discussed in Sec. 2.3.

^1This is usually a provider-independent routing prefix, as discussed in Subsec. 2.3.2.
2. PROBLEM BACKGROUND

Figure 2.2.1. Diagram of site multihoming – as is typical with IP today. A host resides on a single site network, with a site border router (SBR) providing connectivity to two separate ISPs with physically segregated links. The SBR participates in the Internet routing system; the arrowed lines on the right represent the site prefix $P_S$ being advertised in the global Internet routing table.

However, host multihoming is not directly supported in the Internet architecture at the network layer (and needs special treatment for IP [3]); additionally, hosts do not generally participate directly in the routing system. For the scenario illustrated in Fig. 2.2.2, the host would also need advertise both $P_1$ & $P_2$ to each ISP (as in Fig. 2.2.1). Moreover, even if the host participates in the routing system, the problem of IP entanglement arises due to the behaviour of upper-layer protocols; this is explained in Section 2.2.2 on page 11.

Figure 2.2.2. Diagram of host multihoming – using the multihomed, multi-prefix (MHMP) scheme for IPv6 [11]. A single host has connectivity to two separate ISPs with physically segregated links, with routing prefixes $P_1$ & $P_2$, respectively; both prefixes must also be advertised globally within the Internet routing table.
2.2. MULTIHOMING

So, whilst multihoming offers great flexibility in how network connectivity may be used, it is subject to the constraints of how addressing & routing are implemented in IP, discussed in the following section.

2.2.1. Names, Addresses & Routes. In order to understand why the use of multihoming\(^2\) creates problems, it is helpful to consider first how the addressing system functions. An early Internet Engineering Note (IEN) defines network names, addresses, and routes – as follows: “The name tells what the process is; the address tells where the process is; the route tells how to get there.” [12, Pg. 2] Whilst names generally provide a human-readable description of a network element – e.g. as manifest in the Domain Name System (DNS) used in the Internet today, addresses are used to provide a machine-usuable means by which such network elements may be referenced. Following this, routes express the topological information required to reach an address through the network.

2.2.1.1. Origins. The scalability of the IP address system has been a concern since the late 1970s. In 1977, [13] represented a milestone in understanding the problem of hierarchical routing, whereby the routing of data traffic could scale efficiently with the growth of the wider network. Whilst IP has now employed this since 1981 – first introduced in the form of “classful” IP subnetting, based on octet boundaries of the IP address [14, Sec. 2.3] – the practice of variable-length subnet masking (VLSM), whereby addresses were further divided at arbitrary (yet contiguous) bit indices, was not formally proposed until 1993 in the form of Classless Inter-Domain Routing (CIDR) [15].

The further division of IP address space – in the form of CIDR – was driven by the growth of the network; at that time, networking based on the ISO protocol families was widespread, and envisioned as the future standard. So, Berkeley UNIX\(^3\) (BSD) introduced one of the first widely-available implementations of a generic trie-based routing scheme [16], supporting both ISO & IP protocol families. However,

\(^2\)Here, I am discussing the role of unicast addressing in multihoming, as is normally used for 1:1 communication between hosts; and not multicast, broadcast or the special case of anycasting.

\(^3\)FreeBSD – its direct descendant, in terms of code base – is used as the basis of the empirical studies in Chapters 4-6.
as concerns over the long-term scalability of the IPv4 addressing system grew, the Internet research community began to advocate a transition to a new version of IP: IPv6. Whilst both IPv4 and IPv6 are incompatible at wire level, IPv6 makes several improvements and changes beyond IPv4. Here, I confine myself to discussing those details of the addressing system which are relevant to multihoming; refer to Subsec. 2.4.3 for a discussion of relevant points.

2.2.1.2. Addresses in IPv4. The first widely-used version of the Internet Protocol, IPv4, uses an address format which is 32 bits wide. A common human-readable representation of the IP address is the “dot-quad”, which many people are familiar with: e.g. the form 192.168.0.1, where each four octets are separated by a period. However, the bits contained within this address play an essential role in hierarchical routing [14, Pg. 6, Para. 4]: each IP address also has a prefix length, which specifies the number of significant bits forming the network part of the address; in CIDR notation, this is usually written after the address, e.g. 192.168.0.1/24 [17, Pg. 5, Para. 1]. The remainder of the IP address (i.e. the least significant bits after the prefix length) forms the host part, intended to uniquely identify the host within its local subnetwork. It is this overloaded use of the IP address which gives rise to the problems described at length in Subsec. 2.2.2; in addition, the current shortage of available, contiguous IPv4 address space creates additional problems for hierarchical routing in IPv4, which I discuss in Sec. 2.3.

2.2.1.3. Changes in IPv6. Whilst IPv6 uses a larger address representation (128 bits) [18], it largely follows existing IPv4 practices for hierarchical routing [19] (with some key exceptions, which I discuss in Subsec. 2.4.3): the IPv6 address is also divided into network and host parts, using the same CIDR semantics as for IP addresses. However, as IPv6 has inherited many of IPv4’s operational practices and architectural features – including how addresses are allocated and used in the routing

\[\text{http://www.internetsociety.org/news/ietf-statement-ipv4-depletion}\]

\[\text{Alternative addressing schemes based on IPv6 (e.g. GSE/8+8, discussed in Sec. 3.3) propose further changes to its logical structure.}\]
2.2. MULTIHOMING

system – it also inherits the problems described in Subsec. 2.2.2, and these are discussed further in [3, Sec. 5] (although IPv6 has richer support for multihoming; refer to Subsec. 2.4.3).

2.2.2. IP address entanglement. The overloaded use of IP addresses – i.e. to represent both network location and node identity – creates a set of well-known problems [3, Sec. 2]. From an implementation viewpoint, this may be further understood by considering the bindings of an IP address within the protocol stack, as shown in Table 2.1.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Protocol</th>
<th>IP (v4 and v6)</th>
<th>ILNP (ILNPv6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>FQDN(^a), IP address</td>
<td>FQDN or app-specific</td>
<td></td>
</tr>
<tr>
<td>Transport</td>
<td>IP address</td>
<td>[Node] Identifier (NID), NID</td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>IP address</td>
<td>Locator (L64), L64</td>
<td></td>
</tr>
<tr>
<td>Link</td>
<td>IP address</td>
<td>Dynamic binding</td>
<td></td>
</tr>
</tbody>
</table>

\(^a\) Fully Qualified Domain Name

TABLE 2.1. Use of names and addresses in IP by layer (derived from [20, Tab. II]) – as compared with the ILSA studied in this dissertation, ILNP. IP sessions become “bound” to physical links, as the same IP address is used by each protocol layer to both identify individual nodes and for topological routing. By contrast, an ILSA – e.g. ILNP – treats both roles as logically separate; nodes may be reached by several paths, represented by multiple locator values.

Transport layer protocols (e.g. TCP & UDP) use the IP address to identify nodes. In conjunction with port numbers, these are used to uniquely identify sessions between two hosts. However, as shown in Table 2.1, the overloaded use of the IP address creates an implicit binding. So, transport layer sessions become bound to specific links, further complicating the separation of location from identity within the current IP architecture. [3] further describes the shortcomings of IPv4 addresses in the light of operational experience: specifically, their lack of utility as a unique node identifier.

Additionally, the IP network layer and routing functions use the IP address to identify IP sub-networks, i.e. network location. Such use of IP addresses as locators requires that routers exchange topological information regarding how they may be
reached. Multihomed sites add complexity to this exchange, by requiring additional prefixes to be advertised; refer to Sec. 2.3. Whilst this can be mitigated to some extent by the use of Network Address Translation (NAT), this introduces other problems; refer to Subsec. 2.3.5.

2.2.3. The Identifier-Locator Split. Given the problems inherent to how IP addresses are used today (and their role in multihoming, described in Subsec. 2.2.2), there has been renewed interest in networking architectures where location and identity are logically separate, often abbreviated as the “ID/Loc” split: i.e. Identifier-Locator Split Architectures (ILSAs), which I discuss further in Subsec. 2.4 & Chapter 3. In ILSAs, multihoming may be realised as a choice between locators used to reach nodes. However, changes may be required at hosts if they participate directly in such an ILSA.

In addition, concurrent multi-path transport protocols have recently emerged as a mechanism for host multihoming at the transport layer. I review two such protocols in Chapter 3 as part of the solution space: SCTP & MP-TCP. By contrast, ILSAs provide multihoming capabilities at the network layer. However, multi-path protocols cannot solve the problems of how IP addresses may be used by applications on their own; and, moreover, such protocols do not address the problem space of site multihoming.

2.2.4. Summary. In this section, I have reviewed how the IP addressing & routing system used today has developed, followed by a description of how the overloaded use of IP addresses for multihoming affects several layers of the networking stack. Their co-mingled use as identifiers and locators (in hierarchical routing) creates problems for both routing and applications; and, whilst IPv6 proposes a larger address space, this – on its own – cannot solve these problems. In the following section, I examine how these problems manifest within the IP routing system.

2.3. The problem of routing table bloat

The Internet – in its current IPv4 realisation – is reaching its growth limits, as routing tables have now exceeded the capacity limits available in current routers. The problem
is well known and has been widely discussed; [2] contains an excellent summary of the issues. In this section, I review some of its fundamental concepts of operation, using terminology from [21]. I continue by discussing how the problem is now global in nature, and difficult to resolve due to the limitations of current hardware.

2.3.1. How multihoming introduces bloat. Multihoming directly impacts IP routing table growth, as it requires routers to advertise additional prefixes. The entanglement of location and node identity in IP addresses – as discussed in Sec. 2.2 further complicates the issue. In the example shown in Fig. 2.3.1, a site network uses two prefixes: $P_1$ and $P_2$. For the site to be reachable, both prefixes must be visible upstream from the two ISPs: ISP1 and ISP2. I assume here that these prefixes are de-aggregated within the global routing table, and that connectivity to each ISP is topologically diverse; a common practice where BGP is used to multihome sites.

However, ISP1 and ISP2 – and all upstream routers beyond (not shown in Fig. 2.3.1) – must advertise these prefixes separately. This is because they form part of the IP address, which is used in end-to-end state (identity) for hosts (as in Table 2.1 on page 11). It follows that the additional upstream routing state required for $N_P$ site prefixes with $N_I$ upstream ISPs is $O(N_P \cdot N_I)$. Whilst I have used site multihoming in this example (as that is the more common case for IP today), a site would need to support multihoming to allow individual hosts to be multihomed, and the scalability analysis is the same. I revisit this analysis in my discussion of ILNP-based multi-homing in Chapters 4-6.

Figure 2.3.1. Illustrative scenario for multihoming: a site network, with a site border router (SBR), connecting to two separate ISPs. We assume that, in this case, the site network has two routing prefixes, $P_1$ and $P_2$, rather than a single prefix in the simpler example illustrated in Fig. 2.2.1 on page 8.
2.3.2. Provider-independent addresses. The desire for sites to multihome – in the Internet as a whole – is often accompanied by a desire to control how multiple network paths are used. However, as the operation of hierarchical routing in IP is strongly tied to its bitwise representation, some co-ordination of these globally used values is necessary. So – today – the Internet address space is administered by a confederation of Regional Internet Registries (RIRs). Each RIR is responsible for administering IP address allocation in a defined geographical area (e.g. European address space is administered by RIPE⁶, whilst North American allocations are handled by ARIN⁷; several other RIRs exist to administer IP allocations in other geographic regions). Since the introduction of the CIDR hierarchical routing scheme, IP addresses are globally allocated in blocks⁸. So, for the purposes of global IP routing, prefixes may be divided into two main categories:

**Provider-aggregate (PA):** The majority of IPv4 & IPv6 address space falls into this category. The advantage of PA space is that it is centrally administered, and is often bitwise-contiguous; so, the use of PA space does not normally contribute directly towards routing table bloat.

**Provider-independent (PI):** This category of address is assigned by RIRs directly to independent entities and organisations. PI address space assigned to a site may be used – independently of their upstream ISPs – *without renumbering* when migrating to a new ISP. Additionally, the site is able to control how multiple paths are used through *policy routing*. However, for this reason, such prefixes are normally *de-aggregated* in the global routing table.

Whilst the distinction between PA and PI prefixes is an administrative one (i.e. it does not directly affect IP network stack implementations), it affects how routing is implemented, and so it contributes to the problem of routing table bloat through *de-aggregation*. Both IPv4 and IPv6 address systems now contain reservations for

⁶https://www.ripe.net/
⁷https://www.arin.net/
⁸Whilst reservations of individual IP host addresses are sometimes made, these are special cases: e.g. the use of the IPv4 anycast address 192.88.99.1 in IPv4-to-IPv6 transitioning.[22]
PI space – although initially, its use was deprecated in IPv6; discussed further in Subsec. 2.4.3.2. However, the fact that IP addresses are now administered globally means that problems affecting the addressing system – e.g. routing table bloat – are now global in their scope.

2.3.3. Backbone router constraints. The Internet today consists of a number of confederated networks, known as Autonomous Systems (ASs). Internet backbone routers reside in an area termed the Default Free Zone (DFZ) [21, Sec. 4.1.4] and exchange topology information using the Border Gateway Protocol (BGP) [23]. Normally, a router participating in the DFZ must carry the Full Default-Free [Routing] Table (defined in [21, Sec. 4.1.3]); i.e. there are no “default” (catch-all) routes in the DFZ. In August 2014, the size of this table\(^9\) exceeded 512,000 routes for the first time [24]. So, for backbone routers, routing state scalability is a key implementation concern.

Over time, IP routing implementations have evolved to separate control plane concerns from those of the data plane, both to simplify their management and as an engineering optimisation. This separation is realised within two sets of data structures: the Routing Information Base (RIB) [21, Sec. 3] and the Forwarding Information Base (FIB) [25, Pg. 146].

- The RIB provides a logical control plane view of the routing table, and is typically also used to exchange routes with other routers, e.g. through BGP. Additionally, the RIB plays a central role in policy-based routing; refer to Subsec. 2.3.4.
- The FIB represents the “hot path” for packet forwarding: it is used directly to perform the next-hop lookup for each packet at the data plane. In IP today, this lookup is normally performed using only the destination address of a packet. However, multihoming functions may require changes to this process; refer to Chapter 5.

Additionally, hardware-based routers are often designed as discrete appliances in their own right, i.e. with fixed capacity limits and limited upgrade possibilities. The RIB

\(^9\)Current statistics are available at: \url{http://bgp.potaroo.net/}
is often stored in general purpose DRAM, however – due to the requirement for “wire-speed” forwarding – the FIB is often constructed using specialist hardware designs. Two common approaches involve the use of Ternary Content-Addressable Memory (TCAM), and/or trie structures stored in SRAM [26, Sec. 11.3]. Whilst TCAM-based lookups are comparatively fast, the hardware is expensive and has relatively high power requirements; so, production routers often employ a combination of such techniques. Moreover, the growth in capacity of TCAM & SRAM devices does not follow Moore’s Law [2, Sec. 4]; by 1999-2000, it was clear how this difficult situation was developing [27].

Whilst the failure of hardware capacity to keep pace with routing table bloat can be mitigated to some extent – e.g. by using only a subset of the RIB in the FIB, or by aggressively filtering routing prefix updates – such approaches can only either be employed at the edge of the Internet, or have high management overhead. So, there is a compelling case to limit the growth in Internet routing state. General approaches towards solving this problem are discussed in Subsec. 2.4; refer to Chapter 3 for a discussion of specific proposals.

2.3.4. Effects on the global IP routing table. In Subsec. 2.3.1, I have described how multihoming drives routing table growth at the level of a single site; here, I examine its global effects, as this is the subject of the proposals discussed in Chapter 3. Normally, one would expect that the growth of the Internet routing table would correspond directly to Internet growth – as IP prefixes are normally assigned to new networks for topological routing. The bitwise representation of IP prefixes allows contiguous prefixes to be aggregated [17]; this can streamline the size and number of prefix update messages – i.e. BGP churn – through hierarchical summarisation. So, there are two main factors which lead to the deaggregation of IP prefixes in the Internet routing table:

(1) **Address space fragmentation**—driven by the shortage of available contiguous space in the IPv4 addressing system, especially PI prefixes;

(2) **Multihoming**—where IP prefixes with differing path semantics (e.g. due to traffic engineering) must be advertised separately in the DFZ.
Both of these factors are described at length in [2, Sec. 3]. Some authors emphasise that deaggregation does not account for all causes of BGP churn: e.g. [28, Sec. IV/C] describes how \( \approx 10\% \) of deaggregated prefix advertisements are originated by as little as \( \approx 1\% \) of Internet ASs overall, whereas [29, Fig. 2] shows how the majority of BGP updates are due to duplicate advertisements. These are often caused by misconfiguration, and [30] contains a good analysis of how this can lead to instability in the wider network. Additionally, de-aggregation can sometimes be mitigated by using other techniques, e.g. AS path prepending and/or route summarisation; these are applied at the RIB, and [2, Sec. 3.1.3] describes them as “tweaking” the route exchange. However, these approaches cannot be used in the DFZ more widely; this is because they often depend on local topology information, or cannot be implemented without losing site multihoming capability in the process [28, Sec. II/B]. In summary, when BGP must perform additional roles in network control beyond topological routing, the possibility of erroneous or undesirable network behaviour increases.

2.3.5. The role of NAT. [31] describes Network Address Translation (NAT), a technique which is used to translate from one IP address space to another. It is widely used where sites have chosen to allocate private IP address space internally (defined in [32], and distinct from globally routable IPv4 space); e.g. mobile telecommunications providers, where a large and fluid subscriber base would quickly consume available global IPv4 space. So, today, NAT is often employed as a stop-gap measure for connectivity, in the face of the present IPv4 address shortage. It is also widely used for site traffic engineering; its main advantage is that it does not require that additional IP prefixes be advertised in the global routing system.

However, the use of NAT has been widely criticised within the Internet Engineering Task Force (IETF) community – as it breaks the end-to-end principle outlined in [33]. Moreover, the use of NAT introduces indirection problems to features which are otherwise tied to IP addresses (refer to Table 2.1 & [31, Pg. 7]). The discussion in [34, Sec. 2, “Referrals”] describes how this can occur where IP address bits are used by applications as identifiers, and [31, Pg. 6] provides a specific example in the form of the File Transfer Protocol (FTP), which requires changes to operate in the presence of NAT.
Additionally, IP SECurity Extensions (IPSEC) – which provide end-to-end encryption of IP network & transport sessions – are affected in a similar way; NAT “breaks” IPSEC by obscuring the site (or host) endpoint IP addresses upon which IPSEC trust relationships are based. Finally, NAT complicates node mobility (e.g. within the mobile provider example described above) – creating the requirement for middleboxes, e.g. in the form of Carrier Grade NAT (CGN) [35]. Whilst the IETF have attempted to deprecate its use in IPv6, its ongoing utility has led to proposals to reintroduce it in a simplified form; i.e. in the form of NPTv6, as discussed in Subsec. 2.4.3.1 & Section 3.4.1 on page 32.

2.3.6. Summary. Routing table bloat is a problem inherent to how multihoming is implemented in IP today; it manifests as deaggregation of routing prefixes. The problem is well known, yet both the global nature of the Internet – and its hardware limitations – make attempts at resolving the problem difficult. Whilst NAT has been widely deployed in IPv4 as a stop-gap measure, it imposes certain limitations on how the network can be used, or requires middleboxes to perform invasive translation within packet payloads. Regardless of any measures taken to mitigate routing table bloat or BGP churn, each and every prefix introduced to the Full Default-Free Table in the DFZ can be regarded – in computational and economic terms – as an externalised cost, i.e. one which affects all participants. So, for these reasons, a solution which deprecates the role of BGP in traffic engineering (a common application of multihoming, as it is used today) – is highly desirable.

2.4. Solution space

Having described how IP address entanglement contributes to the problems of routing table bloat & update churn, I now describe general approaches to a solution; refer to Chapter 3 for a detailed discussion of proposals. Here, I also discuss how the introduction of IPv6 attempts to address these problems, and how its mechanisms may support such proposals.

2.4.1. General approaches. It follows that many of the proposals discussed in Chapter 3 advocate either an evolutionary move to IPv6, for reasons discussed
throughout this section – or a revolutionary, clean-slate approach to routing and site addressing (e.g. Nimrod & LISP, discussed in Subsec. 3.3.2 & Subsec. 3.4.2).

From a logistical viewpoint, the evolutionary approach may be more desirable, as the infrastructure supporting the IP routing system is now globally diverse. So, these proposals adopt one (or more) of the following general approaches:

**New addressing schemes:** Whilst IPv6 is the most obvious example of a new addressing scheme (as it offers a larger address space, discussed below), several proposals advocate for a new addressing scheme, based on names (e.g. how the Host Identity Protocol (HIP) places cryptographic identifiers in the DNS; refer to Subsec. 3.5.2) or other tokens (e.g. the Routing Locator (RLOC) scheme employed by LISP, discussed in Subsec. 3.4.2).

**Renumbering:** Given that much of the present IPv4 address system resists aggregation – i.e. due to the fragmentation described in Subsec. 2.3.4 – many proposals would require site renumbering, e.g. the LISP proposal described in Subsec. 3.4.2. However, the process of renumbering is disruptive and potentially has high management costs; in Subsec. 2.4.3 I briefly discuss how IPv6 attempts to address this issue.

**Clean-slate architecture:** Approaches which advocate an entirely new approach. The main advantage of such proposals is that they are not encumbered by how routing & addressing systems are implemented today. However, their deployment may require engineering effort.

**Evolutionary architecture:** These approaches generally aim to preserve the use of existing systems – e.g. DNS, routing & addressing – rather than attempting to replace them. The Identifier-Locator Networking Protocol (ILNP) studied in this dissertation resides in this category; refer to Sec. 3.6.

### 2.4.2. ILSAs: Identifier-Locator Split Architectures

There has been renewed interest in networking architectures based on the “ID/Loc” split, as discussed in Subsec. 2.2.3. Such *Identifier-Locator Split Architectures* (ILSAs, as defined in [36]) often support multihoming without the problems described in Sec. 2.3. Whilst these may collectively employ many of the approaches discussed above in Subsec. 2.4.1, they form a discrete research area in their own right.
Moreover, ILSAs may also be broadly be divided into two categories: namespace based – i.e. their approach introduces a new namespace or modifies an existing one (e.g. the Host Identity Protocol (HIP), which uses public key cryptography within the existing DNS), or mapping-and-encapsulation (map-and-encap) based – i.e. an approach which uses tunnelling to construct a new overlay topology, discussed further in Subsec. 3.2.1.

Map-and-encap approaches have the advantage that convergence times (i.e. the time taken to propagate topology changes throughout the network) are potentially lower; however, encapsulation (tunnelling) adds additional overhead throughout the network due to the additional bits (and processing) required. This distinction is an important one, as it describes why the subject of this dissertation – ILNP – is unique within the ILSA solution space, i.e. it does not use tunnelling, and its specific evolutionary approach does not require changes to the existing network; refer to Subsec. 2.4.3.3 & Sec. 3.6.

2.4.3. How IPv6 changes routing & addressing. The problem of address space fragmentation may be attributed to limitations of the IPv4 routing & addressing scheme used today. Moreover, the shortage of available contiguous IPv4 address space resists its solution through aggressive aggregation – and the management-intensive renumbering required by such an approach. So, effort has been underway at the IETF for several years to “sunset” IPv4 – in favour of IPv6. Whilst such an approach is non-trivial from an operational viewpoint, it offers several benefits; [37, Sec. 1.2] provides a good summary. Those most relevant to the present discussion are:

Larger & simpler addressing: This is the most widely understood benefit of IPv6 as compared with IPv4; IPv6 addresses are 128 bits wide, whereas IPv4 addresses are only 32 bits wide. Whilst IPv6 also uses CIDR [18, Sec. 2.5] (and so inherits many IPv4 practices and behaviours [3, Sec. 5]), its larger address space permits aggressive aggregation in the routing system. This

---

10Whilst IPv6 offers other features – e.g. the removal of subnet broadcasts, multicast improvements, mobility & Quality of Service (QoS) support, these are not relevant to this discussion.
streamlines its corresponding routing tables, and provides much-needed flexibility required by certain ILSA proposals; e.g. GSE/8+8 & ILNP, discussed in Subsec. 3.3.1 & Sec. 3.6.

**StateLess Address Auto-Configuration (SLAAC):** A further advantage of IPv6 over IPv4 is that its larger address space permits automated configuration in the form of SLAAC, a part of IPv6 Neighbour Discovery (ND) [38]. This potentially reduces the management overhead required to administer IPv6 sites, as host addresses need no longer be managed on an individual basis (e.g. how the Dynamic Host Configuration Protocol (DHCP) is often used today in IPv4 networks). However, the use of SLAAC assumes that sites have not aggressively sub-netted beyond the 64-bit boundary used to form auto-configured addresses. Additionally, the boundary lends itself to re-use by ILSAs such as GSE/8+8 & ILNP; this is discussed further in Chapter 3.

**Simplified network renumbering:** In IPv6, network renumbering is a relatively simple process (as compared to IPv4); [39] provides a detailed walkthrough of the process of renumbering an IPv6 site; automated mechanisms for site renumbering were added to IPv6 ND for this purpose [38, Sec. 4.1]. Recalling the discussion in Subsec. 2.3.2, one of the main reasons sites apply for PI address space is to avoid renumbering when migrating to a new ISP. Additionally, for ILSAs which require tunnelling (e.g. LISP, discussed in Section 3.4.2 on page 32), renumbering may be required to defragment the address space used by backbone routers. So, the use of IPv6 considerably simplifies this process.

Whilst there are other new features within IPv6 – e.g. quality of service, IPSEC encryption, and support for node mobility – these are not within the scope of this dissertation. The remainder of this section discusses those features of IPv6 which are relevant to multihoming, and its associated problem of routing table bloat.
2.4.3.1. The non-deprecation of NAT. I have briefly discussed NAT\textsuperscript{11} – and the role it plays in the Internet today – in Subsec. 2.3.5. Whilst the IETF has encouraged the deprecation of NAT in IPv6, this has been met with some criticism; and so, [40] proposes NPTv6 (discussed further in Sec. 3.4) as a solution for site multihoming. Formerly known as NAT66, [41, Pg. 19–20] emphasises its similarities with ILNP, and goes on to discuss how such translation may simplify site multihoming in IPv6: “The complexities of the network are ... transferred to the application itself, but not to its transport”, recalling the situation of IP address entanglement discussed in Subsec. 2.2.2; the situation described here is discussed further in Subsec. 4.4.4, i.e. whilst ILNP can disentangle location at the network layer in a similar way, some applications may still be affected – as they use IP address bits directly, being unaware of the “ID/Loc” split or translation performed there. So, being the closest analogue to ILNP in the solution space, NPTv6 is used for comparative evaluation with ILNP at the site border in Sec. 5.3.

2.4.3.2. The non-abolition of PI addresses. Whilst PI addresses (described in Subsec. 2.3.2) give sites much flexibility – e.g. by allowing them to implement multihoming functions, independently from their upstream ISPs – their use has been contentious, given the problems described in Sec. 2.3. Historically, the position of the IETF has been to deprecate their use in IPv6; however, they were never entirely removed from the IPv6 addressing system. Given their importance to large sites (and the deprecation of NAT in IPv6 by the IETF), there has been pressure\textsuperscript{12} to reintroduce their use more generally – despite the fact that their use can lead directly to further de-aggregation [2, Sec. 2.1.1, Para. 2]. Moreover, the use of PI prefixes (and their place in CIDR-based hierarchical routing) is a practice which the IP engineering community are already familiar with [2, Sec. 8.4], despite the provisions made in IPv6 for renumbering (discussed in Subsec. 2.4.3).

\textsuperscript{11}Here, I am referring to NAT in its 1:1, 1:M and N:M forms – i.e. where transport protocol port numbers may also be translated, in order to share addresses – and not the IPv4-IPv6 transition mechanism known as NAT-PT, which has been deprecated by the IETF for operational reasons.

\textsuperscript{12}http://etherealmind.com/importance-provider-independent-ipv6-addresses/
So, in 2009, the RIR responsible for administering IPv6 address space throughout Europe – RIPE – removed\(^\text{13}\) its imposed restrictions on IPv6 PI allocations. However, these restrictions\(^\text{14}\) were originally introduced by RIPE merely to avoid a situation where the limited supply of PI space was prematurely depleted, not to rule out their use entirely. So, whilst the use of PI addresses has been discouraged in IPv6 (for reasons discussed in Sec. 2.3), they have been reintroduced as a “stop-gap” engineering measure, where a compelling alternative is otherwise unavailable – e.g. in the form of a widely deployable ILSA.

2.4.3.3. *How IPv6 affects ILSAs.* With the enhancements described in Subsec. 2.4.3, IPv6 is well positioned to address the requirements of multihoming (and the problems which have, so far, accompanied it); [37, Sec. 3.8, Pg. 33] provides a brief summary (although, as discussed above, PI addresses are still very much part of IPv6), and [42] expands upon the proposed solutions in [5]; discussed further in Chapter 3.

However, both the new address syntax & SLAAC features of IPv6 may enable the widespread adoption of ILSAs without tunnelling. ILNP is an example of an ILSA which fits this definition, where the meaning of an IP network prefix – as used in topological routing – does not change. Within such an architecture, both sites and hosts are able to use locator rewriting to implement traffic engineering (and other networking functions) according to their own policies – without introducing state to the global routing system through the use of de-aggregated PI prefixes.

2.4.4. **Summary.** The routing bloat manifest in the Internet today – i.e. routing state introduced in the form of deaggregated prefixes – might be eliminated or greatly reduced by the adoption of a new approach to network routing. ILSAs are an ongoing research topic to this end, discussed further in Chapter 3; the focal theory in this dissertation is that ILNP can be employed to reduce or eliminate the routing state otherwise introduced to support multihoming and its use cases. Moreover, the adoption of ILSAs may allow BGP – as used today – to be relegated to the role of

\(^{13}\text{https://www.ripe.net/participate/policies/proposals/2009-08}\)

\(^{14}\)Whilst the use of PI prefixes could be tied by the RIRs to a responsibility to provide transit access – in essence, to force organisations holding PI address space to provide connectivity for *other* organisations – this would be difficult (if not impossible) to enforce.
topological route exchange only. Whilst this may not entirely eliminate the problems described in Subsec. 2.3.1, this would constrain where such problems might occur in the network overall – and so improve the stability of the Internet routing system, also.

2.5. Chapter summary

In this chapter, I have described why multihoming is a desirable networking feature and how it operates, followed by how it contributes to the problem of routing table bloat, and its root cause: the overloaded use of IP addresses, which are used today to represent both network location and node identity. The solution space consists of revolutionary approaches which start anew (clean slate), and evolutionary approaches which preserve how the existing address system is used in network routing.

Whilst the Internet research & engineering communities have been aware of these problems for over 35 years, location independence is a design principle which has regrettably not been realised in the current iteration of the Internet. The problem of routing table bloat affects the whole Internet, also; whilst NAT is often employed to limit the full impact of this problem, its use breaks the end-to-end principle central to IP and its original design. The cost is ultimately borne by users indirectly, i.e. in the form of imposed limits on how they may use and interact with the rest of the network – e.g. NAT, and how it is used to circumvent – but not solve – the shortage of globally routable IPv4 address space.

So, the adoption of ILSAs may offer a solution; in the following chapter, I discuss several related proposals in this area. However, as the problem space is Internet wide, the solution space must take account of the logistical difficulties involved with deployment; an incrementally deployable solution may be preferable. Whilst network overlays are a valid engineering approach – and are often used today for other reasons, e.g. to bridge IPv6 domains across an IPv4-only Internet – their use comes with a performance (and management) penalty. Finally, this dissertation considers one candidate from the ILSA solution space in depth: ILNP, and presents its first implementation in a production operating system. Its position in that space is unique, as
it re-uses the existing IPv6 addressing scheme (and \textit{does not} require tunnelling) – so, it is well suited to \textit{incremental} deployment.
Chapter 3

Approaches to IP multihoming

3.1. Introduction

In this chapter, I examine several proposals to enhance multihoming as implemented in the Internet today.

3.1.1. Chapter structure. This chapter begins with an overview in Sec. 3.2 of multihoming proposals and the overlapping concerns which frame their solution space, followed by four (4) sections discussing each proposal in more depth:

(I) Sec. 3.3 discusses historical proposals (e.g. Nimrod, GSE/8+8) & related work, providing background for the following two sections;

(II) Section 3.4 on page 31 describes network-based multihoming proposals: LISP – a complete routing architecture, and NPTv6 (NAT for IPv6),

(III) Section 3.5 on page 34 discusses the host-based multihoming proposals HIP & SHIM6, and the concurrent multipath transports SCTP & MP-TCP,

(IV) Section 3.6 on page 39 contains a high-level discussion of ILNP, providing background for Chapters 4–6, followed by a summary of Sec. 3.3–3.6.

Section 3.8 on page 43 discusses the evaluation methodology adopted herein, including measurement tools and their application. Finally, the chapter concludes with a summary in Sec. 3.9.

3.1.2. Terminology. The historical note IEN-31 [12] discussed how the terms Locator and Identifier were derived, as they appear in [5]. Layering terms are used as they appear in the ISO-OSI reference model [43].
3.2. Overview of proposals

Table 3.1 lists the proposals examined in this dissertation. These may broadly be divided into two categories: *host-based* approaches – which make the “ID/Loc” split a property of host address management (possibly introducing new namespaces, addresses or transport protocols), and *network-based* approaches – which make the split a property of the routing infrastructure (i.e. leaving IP addresses unchanged at hosts). A subset of the proposals listed in Table 3.1 are summarised in [5], which also restates the problem of routing table growth discussed in [44].

<table>
<thead>
<tr>
<th>Proposal</th>
<th>Property</th>
<th>Host</th>
<th>Site</th>
<th>IPv4</th>
<th>Approach</th>
<th>Refer to</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSE/8+8*</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Addresses</td>
<td>Subsec. 3.3.1</td>
</tr>
<tr>
<td>HIP</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Public keys in DNS</td>
<td>Subsec. 3.5.2</td>
<td></td>
</tr>
<tr>
<td>ILNP</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Names</td>
<td>Sec. 3.6</td>
<td></td>
</tr>
<tr>
<td>LISP</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Map-encap*</td>
<td>Subsec. 3.4.2</td>
<td></td>
</tr>
<tr>
<td>MP-TCPd</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Transport</td>
<td>Subsec. 3.5.3</td>
<td></td>
</tr>
<tr>
<td>Nimroda</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Map-encap</td>
<td>Subsec. 3.3.2</td>
<td></td>
</tr>
<tr>
<td>NPTv6</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Prefix rewriting*</td>
<td>Subsec. 3.4.1</td>
<td></td>
</tr>
<tr>
<td>SCTPd</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Transport</td>
<td>Subsec. 3.5.4</td>
<td></td>
</tr>
<tr>
<td>SHIM6</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Addresses</td>
<td>Subsec. 3.5.1</td>
<td></td>
</tr>
</tbody>
</table>

* Historical proposal (not listed in [5])
* Host-based architecture (with site engineering properties; refer to Chapter 5–6)
* Clean slate architecture, with non-IP namespace
* Concurrent multipath transport protocol
* Simplified realisation of NAT (in IPv6)

**TABLE 3.1. Multihoming proposals examined in this chapter – in alphabetical order.**

### 3.2.1. Deployment & impact.

Some proposals (e.g. Nimrod & LISP) use “map-and-encapsulate” overlay networks (tunnelling) – abbreviated as “map-encap” – rather than native IP forwarding. Whilst overlays may reduce address space deaggregation (and the need for middleboxes), they add significant complexity to deployment: e.g. [45, Sec. 4] describes the three logical steps involved in “map-encap”, and their dependencies in terms of physical routing/switching hardware.

By contrast, some proposals (e.g. HIP, ILNP) use the existing DNS as their rendezvous mechanism; whilst this has the advantage of adding neither routing state nor
additional infrastructure, this choice also inherits the performance characteristics of the DNS, and so the resulting end-user experience may be affected by DNS caching and request round-trip times. Additionally, incremental deployment is a shared concern: it would be desirable to avoid a “flag day” situation where a vastly distributed deployment effort must be coordinated. This is a problem which DNS-based mapping approaches may avoid, as DNS records may be “seeded” well in advance of a deployment date.

Finally, whilst some proposals do actively introduce additional routing state (e.g. LISP’s re-use of BGP), the function of congestion control remains the responsibility of the transport protocol used (e.g. TCP & SCTP).

3.2.2. Security. The widespread use of multihoming affects security for both (1) its control plane – requiring additional security for its signalling mechanisms, and (2) the data plane, where the use of multiple addresses may affect current security approaches. [46] contains an excellent summary of these issues. For (1), most proposals examined in this chapter use some form of cryptographic handshake to secure such signalling, as described in [46, Pg. 6, Para. 4]. For (2), the problem space extends to the normal flow of data traffic; e.g. [47, Sec. 6] describes how the use of multiple addresses may invalidate assumptions made by current intrusion detection schemes.

Moreover, [46, Sec. 4.4] describes a problem inherent to hosts in an “ID/Loc” split architecture: how to deal with unknown locators. Historically, [48, Pg. 61–62] has defined a “Strong End-System” (StrongES) model for IPv4 hosts. whereby the binding between links and addresses (shown in Table 2.1 on page 11) is treated – according to [49, Pg. 4] – as “separate logical hosts” within the same multi-homed host. However, as [50, Sec. 3.2.1.2] states, “IPv6 has always implemented the strong host model”. So, [51] describes a scheme for first-come, first-served validation of source addresses – suitable for use with multi-homed hosts – and its approach is amenable to integration with IPv6 Neighbour Discovery (ND) in existing IPv6 host stacks.

Finally, [52] discusses host firewalls – as opposed to site or router-based firewalls – more generally; it is reasonable to suggest that the adoption of a host-based multihoming scheme might require widening the scope of service-based firewall policies –
e.g. those described in [52, Sec. 4.1] – to adapt to the use of multiple prefixes or locators, i.e. in order to mitigate against the new attacks described in [46, Sec. 4].

3.2.3. Privacy. User privacy is a compelling issue in the Internet today which often overlaps with security. This overlap extends to its solution space also, as security mechanisms should be chosen appropriately to meet the threat model. So, [53, 54] describe pervasive monitoring as an attack; it must be assumed from the outset of communication that any data exchanged may potentially be intercepted by a third party, and this requirement drives the use of other mechanisms to obscure identifiers. These include the use of Cryptographically Generated Addresses (CGAs) [55] & Hash-based Addresses (HBAs) [56] to protect host identity [57, Section IV/A]. Both ILNP and SHIM6 may use them; refer to Sec. 3.6 & Section 4.3.7 on page 66.

3.3. Proposals: Part I—Historical & related work

This section discusses the historical multihoming proposals: GSE/8+8 & Nimrod, and RFC 7157 – which describes IPv6 multihoming without translation.

3.3.1. GSE/8+8: Global, Site, and End-system Addresses. The historical proposal 8+8 [58] – and its subsequent revision, GSE [59] – attempted to solve the problem of routing table bloat, by proposing aggressive aggregation of the address space used for topological routing. GSE proposed an “ID/Loc” split by dividing existing addresses\(^1\) into three components; i.e. “end-system designators” (ESDs) [58, Sec. 5], “site topology partitions” (STPs) & “routing goop” (RG) [58, Pg. 11]. GSE re-uses the DNS for discovery of the RG, avoiding the problems associated with “map-encap”. Site-controlled multihoming is a first class function of GSE/8+8 [59, Sec. 11]. Moreover, the use of RG simplifies site renumbering, and neither GSE or 8+8 use tunnelling. Whilst neither proposal reached RFC status (or deployment), some of their concepts – e.g. integration with the DNS [59, Sec. 10.2], & symmetric locator rewriting [59, Sec. 15] – appear in other proposals (e.g. LISP, ILNP). GSE applicability is discussed further in [60].

\(^1\)8+8 codifies the split in the 128-bit IPv6 address syntax, by placing RG into the most significant 64 bits, and ESDs & STPs into the lower 64 bits – similar to ILNP, as shown by Fig. 4.2.1 on page 56.
3.3.2. Nimrod. [61] proposes a complete “map-encap” routing architecture based on the “ID/Loc” split, with optional extensions for multicasting and mobility [62, 63]. It is designed to scale arbitrarily, yet provide application-specific functionality\(^2\) throughout. Addressing is part of its specification [64]: its “endpoint identifiers” [61, Sec. 3.1] correspond to identifiers in [5] terminology. Nimrod was designed to support policy-driven load sharing and failover functions using maps [61, Sec. 3.5] to control locator selection [61, Sec. 3.3–3.4]. Whilst it has not been deployed\(^3\) in an operational network, the proposal contains a clear and straightforward description of how to realise the “ID/Loc” split architecturally.

3.3.3. MHMP: Multi-Homed with Multi-Prefix. [11] is an informational RFC describing how IPv6 multihoming functions today in the absence of NAT. Moreover, [11, Sec. 6] defines IPv6 multi-homed with multi-prefix (MHMP) hosts: i.e. IPv6 hosts which use a combination of “source address selection, next-hop resolution and (optionally) DNS resolution” to utilise diverse links, without resorting to NAT – e.g. in the form of NPTv6. Many of the features required by MHMP form the basis of features within the ILNP stack studied in Chapters 4—6. However, [11, Sec. 7] explains a corner case for IPv6 multihoming requiring NPTv6, i.e. where a single IPv6 address must be assigned to a host which otherwise does not follow the MHMP requirements. So, following this, the evaluation in Chapter 5 of this dissertation is based on a comparative study with NPTv6, which – according to [11] – would otherwise be required as an intermediate solution.

3.4. Proposals: Part II—Site multihoming

This section discusses two multihoming proposals for sites (and not individual hosts): NPTv6 – which provides a form of NAT adapted to IPv6, and LISP – which defines a complete routing architecture.

\(^2\)Nimrod also specifies flow-based traffic engineering with some properties of the Resource ReserVation Protocol (RSVP) [61, Sec. 5.4].

\(^3\)Source code was historically available from Bolt, Beranek & Newman (BBN), before their acquisition by Raytheon in 2009.
3. APPROACHES TO IP MULTIHOMING

3.4.1. NPTv6: IPv6-to-IPv6 Network Prefix Translation. [40] specifies NPTv6\(^4\), a stateless 1:1 prefix translation scheme for IPv6 which facilitates site renumbering and address independence. Load sharing is supported through the use of multiple translators [40, Sec. 2.3].

3.4.1.1. Method. NPTv6 is simple to implement: it translates the routing prefix part of the IPv6 address, leaving the interface identifier unchanged. Transport checksums are rewritten (according to [40, Sec. 2.6] & [65]) and payloads are unchanged. Additionally, the stateless mapping has low overhead, as no dynamic state need be exchanged between translators.

3.4.1.2. Caveats. The IETF does not recommend [40, Sec. 1] the use of NPTv6. [66] contains an excellent summary of reasons why NAT is harmful, and [67] expounds on how it is specifically harmful\(^5\) to IPv6. Additionally, load sharing is not transparent: it requires multiple NPTv6 translators [40, Sec. 2.4] and explicit next-hop choice amongst them. So, its use creates a single point of failure – even where multiple translators are used.

3.4.1.3. Summary. NPTv6 provides prefix translation – and not locator rewriting. It may be used to reduce global address space usage (and concomitant routing state), [67, Sec. 2.2] just as with IPv4 based NATs [66, Sec. 5]. However, it inherits all of the existing problems inherent to the use of NAT, and does not resolve IP entanglement (shown in Table 2.1 on page 11).

3.4.2. LISP: Locator/Identifier Separation Protocol. [68] proposes a complete routing architecture: LISP\(^6\), with RFC status [69]. It supports traffic engineering (e.g. failover and load sharing) through site-controlled multihoming; i.e. it does not require changes at hosts. LISP has been implemented within Cisco IOS, and as a separate open-source code base, OpenLISP\(^7\). However, the scope of its changes

---

\(^4\)Formerly known as NAT66.

\(^5\)Additionally, the “security through obscurity” effect of deeper translation is absent [67, Sec. 2.4].

\(^6\)http://www.lisp4.net/

\(^7\)http://www.openlisp.org/
is sufficiently wide to mandate new router deployment. A trial network is in place between several Tier 1 ISPs and organisations, and there is current work in progress to further standardise LISP [70, 71].

3.4.2.1. Method. LISP defines its own “ID/Loc” namespace in the form of Routing Locators (RLOCs) and Endpoint Identifiers (EIDs); these correspond to the locator and identifier terms defined in [5]. It is a “map-encap” architecture: to cross a LISP routing domain, IP packets are encapsulated by Ingress Tunnel Routers (ITRs) at its edge. So, LISP ITRs must implement both packet classification and encapsulation functions. LISP forwarding only encapsulates traffic if it cannot be forwarded natively; i.e. the encapsulation is normally visible only to LISP-enabled routers.

The majority of LISP’s complexity resides in the mapping layer: whilst EIDs may re-use IP address values, RLOCs reside in a separate namespace from IP addresses. RLOC mappings provide a flexible, policy-driven framework, which can support traffic engineering between RLOCs. There is ongoing work on other LISP mappings: one such scheme is the LISP Canonical Address Format (LCAF) [72, Sec. 4.4]. This scheme enables policy-driven routing using arbitrary criteria, e.g. geodetic routing using GPS coordinates. However, the scheme most relevant to this dissertation is the AL ternate Topology (ALT) scheme described in [73].

To exchange routing prefix information between the edge(s) of a LISP domain, ALT normally requires an additional BGP full mesh between ITRs, [73, Sec. 4.2–4.3] transported within a GRE-encapsulated [74, 75] overlay network. However, [76] proposes to distribute ALT topology information in existing BGP sessions without an overlay. By contrast, the core LISP network uses dedicated LISP mapping servers (MS) to relay RLOC routing information. This hybrid approach is intended to enable scalability: it represents a compromise between batch distribution of routing information by flooding (e.g. as used by the Open Shortest Path First (OSPF) protocol, commonly used in intra-domain routing) and making explicit requests (as used by an on-demand routing protocol; e.g. the Ad-hoc On-demand Distance Vector Protocol (AODV), used in mobile wireless applications).
3.4.2.2. Caveats. LISP is a large and complex specification with significant development effort behind it. Whilst the use of ALT may reduce global routing state, efficient operation requires strict hierarchical aggregation of EID address space, creating additional management overhead (and possibly requiring the renumbering of Internet core routers, also). However, [73, Sec 7.1] suggests that the phenomenon of route flap\(^9\) – as sometimes seen in the Internet today – is less likely to be caused by changes in the ALT, than by BGP misconfiguration.

Additionally, the “map-encap” scheme used by LISP may affect existing applications, e.g. due to mismatched Maximum Transmission Unit (MTU) sizes; an issue which arises today with some ADSL access networks. Encapsulation may require upgrades to existing routers re-purposed as LISP ITRs (and, complicate diagnostics, also), as its intrusive approach adds headers inside existing IPv4 packets [68, Fig. 2]. Finally, incremental deployment of LISP is problematic, as ALT explicitly requires BGP. If LISP functionality is segregated from current BGP deployment – e.g. by using logically separate instances of BGP – routers may also require potentially expensive memory upgrades.

3.4.2.3. Summary. LISP provides a full “ID/Loc” routing architecture – for routers (and not hosts). The use of LISP with ALT potentially reduces global routing state, by allowing the EID address space to be aggregated. Whilst it can limit “prefix churn” in the global routing table (discussed in Chapter 2), it does not eliminate it. However, it does not attempt to resolve the issues of IP entanglement at hosts. Moreover, it may be difficult to deploy incrementally, as it requires significant (and potentially expensive) changes – both to existing routers, and to the current practice used to manage them.

3.5. Proposals: Part III—Host multihoming

In this section I discuss four proposals for multihoming at hosts: SHIM6 – a “shim” layer for IPv6 multipath, HIP – a complete architecture, and two concurrent multipath transports: Multipath TCP & SCTP.

\(^9\)Repeated BGP UPDATE messages for a set of IP prefixes – in a short time window – which each contain different information.
3.5.1. SHIM6: Level 3 Multihoming Shim Protocol for IPv6. [77] defines SHIM6, a host-based multihoming scheme for IPv6 (and not IPv4). IPv6 applications may use SHIM6 and its capabilities – e.g. failover – without changes.

3.5.1.1. *Method.* As its name implies, SHIM6 is implemented\(^{10}\) as a “shim” at the network layer; it requires minimal modification to existing transport protocols. When a transport session is established, the IPv6 address first chosen by the networking stack is used as an “Upper Layer IDentifier” (ULID) [77, Sec. 1.3]. Independently of the transport, a four-way handshake is used to establish a SHIM6 context (i.e. exchange of *identifier* and *locator* values). Its companion protocol REAP [78] provides locator pair enumeration and failure detection (including “soft handover”\(^{11}\)) functions; their use is managed using extensions to the sockets API [79]. Other features (e.g. load sharing) are scoped as optional extensions [80, Sec. 5.2–5.3]. For security against off-path attacks, either CGAs or HBAs may be used [55, 56].

3.5.1.2. *Caveats.* Whilst SHIM6 can be used transparently by applications, all participating hosts must implement its extensions to IPv6. Moreover, whilst presented as a site multihoming solution, [77, Sec. 1.2] site-controlled multihoming is problematic to implement using its approach, as intermediate hops (e.g. routers, middleboxes) do not participate directly in locator choice. Additionally, it assumes that there are no on-path IPv6 NATs [77, Sec. 3, Par. 5].

3.5.1.3. *Summary.* SHIM6 is an extension to IPv6 providing transparent failover for IPv6 applications. It does not address scalability of the routing system, and its deployment may be problematic due to limitations in locator selection & NAT traversal.

3.5.2. HIP: Host Identity Protocol. [81] defines HIP (v2): a host-based architecture for mobility, multi-homing and address renumbering. An architectural overview of HIP is given in [82]; much of its design aims to address issues posed by the IRON/RANGER proposals [83, 84] (also referenced in [5]), particularly, the

\(^{10}\)At least two implementations are known to exist: OpenHIP and LinShim6.

\(^{11}\)i.e. where \(N \geq 2\) locators are available to a session, permitting dynamic path migration with minimal packet loss.
avoidance of “map-encap” overlays. Whilst it supports failover, there is limited support for traffic engineering & load sharing. However, its proposal requires widespread changes to IP network stacks.

3.5.2.1. **Method.** HIP uses an alternative identifier namespace, separate from IP addresses. The “ID/Loc” split is realised using an asymmetric key crypto-system\(^\text{12}\) (e.g. RSA and/or Diffie-Hellman): Host Identifiers (HI) are represented as public keys, whereas locators are IP addresses. HIs are never directly exposed to the routing system, nor are they directly represented in the addressing schema. Additionally, HIs may reside in public and private namespaces.

Hosts resolve names to HIs using HIP Resource Records in the DNS. Whilst a host can have many HIs, each HI must identify a single host. Additionally, as HIs are large as compared to addresses (e.g. 2048 bit RSA public keys vs 128 bit IPv6 addresses), HIs are hashed to Host Identity Tags (HITs) to provide a compact protocol encoding [81, Sec. 3]. Sessions are established by a four-way cryptographic handshake, i.e. the HIP Base Exchange [81, Sec. 1.2]. Rendezvous servers provide an indirect mapping for mobility & other purposes, e.g. mapping between HIs & HITs to IP addresses [85].

With HIP, IP protocol bindings (shown in Table 2.1 on page 11) must also change: whilst routing prefixes continue to be treated as locators (as used in topological routing), transport protocol sessions are now bound to HITs. Additionally, applications must use HIs in place of network addresses, and control of HIP functions requires extensions to the sockets API [86]. Multiple locators are used primarily for fail-over [87, Pg. 12]; support for load sharing with HITs is the subject of ongoing work [88, Sec. 3.1.3].

3.5.2.2. **Caveats.** Whilst HIP is presented as a set of incremental extensions to the existing IP architecture (mostly for hosts), the scope of the changes required is wide and far-reaching, such that its implementation may be as difficult\(^\text{13}\) as a “clean slate” approach.

\(^{12}\)Most of the changes between HIP versions 1 & 2 are concerned with its use of such cryptographic primitives.

\(^{13}\)Despite this, [42] identifies five existing implementations of HIP.
Although intended to be transparent to existing IP applications, [89, Sec. 3] its proposal requires changes to hosts, applications, services and protocols; the use of interposed DNS agents may be problematic [90, Sec. 3]. Moreover, the dependence upon HITs at the transport layer may be unsuitable for many use cases, and has indirection issues (e.g. wildcard bound UDP sockets used by DNS servers [90, Sec. 5]). However, whilst many of these issues are not specific to HIP, its use of cryptographically derived host identities may complicate their resolution.

So, incremental deployment is an area of concern for HIP; many of the issues in this area are summarised in [87]. Additionally, site-controlled multihoming would be problematic to implement, as routers do not normally participate in HIP.

3.5.2.3. Summary. HIP proposes a complete “ID/Loc” architecture for hosts. It does not make specific provision for site-controlled multihoming – so, if employed to address the concerns discussed in Chapter 2 regarding routing prefix de-aggregation, it would require widespread deployment. Its approach of “retro-fitting” cryptographic identities to the existing IP stack is complex – and, whilst it has been supported by a significant amount of work, it has not been widely deployed as of writing.

3.5.3. MP-TCP: Multipath TCP. [47] defines a set of extensions\(^{14}\) to the TCP protocol enabling concurrent multi-path transfer: Multipath TCP (MP-TCP). Its main advantage is that it does not require a new protocol. Additionally, it may be used transparently by existing applications.

3.5.3.1. Method. When initiating a TCP session, MP-TCP capable hosts include the MP\_CAPABLE option in their initial 3-way handshake. The option includes a cryptographic nonce to protect MP-TCP control information from off-path attacks. Concurrent multipath operation is achieved by grafting sub-flows into an established MP-TCP session. Additionally, congestion control is normally performed discretely for each sub-flow. However, the node may manage congestion control across sub-flows as a group, [47, Sec. 3.3.7] i.e. to achieve efficient resource pooling. Explicit address management messages are also defined in [47, Sec. 3.4.1].

\(^{14}\)As of writing, these are available in production TCP implementations (e.g. Apple Mac OS X and Linux).
3.5.3.2. **Caveats.** MP-TCP requires that all participating end hosts are upgraded to support MP-TCP. Moreover, MP-TCP is also affected by the overloaded use of IP addresses as identifiers (shown in Table 2.1 on page 11) during NAT traversal. Whilst MP-TCP specifies some mechanisms for continued operation across middleboxes (including NATs) – e.g. explicit signalling of new addresses – these often result in falling back to unmodified (i.e. single path) TCP [47, Sec. 6, Pg. 52]. Finally, whilst providing concurrent path choice at the transport layer (i.e. sockets) offers great flexibility, it may be difficult to manage due to its fine granularity. However, this does not preclude the use of site-wide policies to control MP-TCP socket behaviour.

3.5.3.3. **Summary.** MP-TCP provides host (and not site) multihoming by extending the widely used TCP transport protocol. Whilst this does not require deployment of a new protocol, the new multipath semantics are sufficiently different from TCP today to cause deployment issues. Moreover, MP-TCP does not attempt to solve issues of scale within the routing system (discussed in Chapter 2), as its solution space is confined to hosts.

3.5.4. **SCTP: Stream Control Transmission Protocol.** [91] defines a new transport protocol: SCTP, which provides host multihoming capabilities, including failover and (optionally) load sharing. SCTP does not assume (or require) that a host participates in the routing system.

3.5.4.1. **Method.** SCTP combines stream and message oriented communication (as in TCP and UDP, respectively). It does this through the use of associations – rather than sessions – which enable flexible use of multiple IP addresses by applications. Associations are established by 4-way handshake, with a cryptographically generated nonce used to protect against off-path attacks. Information about multiple addresses is exchanged in both INIT and HEARTBEAT chunks [91, Sec. 6.4]. Whilst failure detection (i.e. failover) is provided by the use of HEARTBEAT chunks, load sharing is the subject of ongoing work [92]. Like MP-TCP, congestion control must be extended to support concurrent multipath usage. Additionally, control of associations requires extensions to the sockets API [93].
3.5.4.2. Caveats. As SCTP is not used as widely as TCP & UDP, its use requires co-ordinated deployment between hosts. Its use is not transparent to applications\textsuperscript{15} which must be rewritten. Moreover, whilst associations enable policy-driven use of addresses, applications inherit the shortcomings of their use – as shown in Table 2.1 on page 11. Whilst its problems with NAT traversal have been studied in depth (and are exacerbated by its complexity as compared to TCP & UDP) \textsuperscript{94}, it is not transparent to middleboxes (e.g. firewalls and NATs) \textsuperscript{[91, Sec. 3.3.2.1]}. For this reason, it is often transported using an outer UDP encapsulation \textsuperscript{95}. However, this does not solve the issues raised by the overloaded use of IP addresses as identifiers within SCTP applications.

3.5.4.3. Summary. SCTP provides host (and not site) multihoming at the transport layer. So, it does not address the routing system scalability issues discussed in Chapter 2. Additionally, it is less widely used than the more familiar TCP & UDP transport protocols, and so applications must be rewritten to support its use.

3.6. Proposals: Part IV—ILNP

This section discusses the Identifier-Locator Network Protocol (ILNP): an end-to-end “ID/Loc” architecture which supports both host and site-controlled multihoming (and mobility, also). Moreover, ILNP does not use “map-and-encap” (i.e. tunnelling). The IETF Routing Research Group recommends in \textsuperscript{[5, Sec. 17.2]} that ILNP be pursued further. My empirical work on ILNP within the FreeBSD networking stack has contributed to its standards process in two key areas: firstly, resolving the ambiguity between ILNP flows and 1:M UDP communication as described in Section 4.3.2 on page 62; and secondly, its interoperability with IPSEC encryption as described in Section 5.2.1 on page 95. Whilst this dissertation studies an implementation of ILNP – in the widely used FreeBSD networking stack – it has previously been studied in simulation \textsuperscript{[57, Sec. V]}. Other efforts include an incomplete implementation \textsuperscript{96} based on the Linux Netfilter\textsuperscript{16} framework.

\textsuperscript{15}Whilst “one-to-one” associations \textsuperscript{93, Sec. 4} expedite porting of existing code, they are not a compatibility layer.

\textsuperscript{16}http://www.netfilter.org/documentation/
3. APPROACHES TO IP MULTIHOMING

3.6.1. Method. ILNP is radically different from IP because it has no addresses; instead, it uses node identifiers and topologically-significant locators. [6, 20] expound on its architecture and describe how it may be applied to address the problem of routing table growth (discussed in Section 2.3 on page 12). As with the historical GSE/8+8 proposals described in Sec.3.3, (1) the existing DNS is used to resolve names, and (2) the existing address syntax is re-interpreted – i.e. IPv6 addresses are split into locators and identifiers at the level of each packet (shown by Fig. 4.2.1 on page 56) [20, Sec. II]. Additionally, the precedence\(^\text{17}\) of locators may be used to implement policy-driven network functions (e.g. load sharing & failover); refer to Section 4.2.3.1 on page 58.

ILNP can be realised as a set of extensions to the existing IPv6 architecture – ILNPv6 – as described in [6, Sec. 4–6] & [7, 8, 9]. As the existing DNS is re-used, the resource records\(^\text{18}\) (RR) ID, L32, L64 and LP are defined [97] to publish locators and identifiers (discussed further in [98]). These enhancements to naming were first discussed in [99], following from the historical discussion in [12]. Whilst Dynamic DNS (DDNS) [100] is required for the remote update of ILNP RRs, DNS caching should be disabled\(^\text{19}\) (or tuned to a low expiry interval) [102] to avoid stale lookup results.

3.6.1.1. Flows & transport layer binding. Sessions\(^\text{20}\) are established using a nonce-based handshake ([6, Sec. 8, Par. 5], [103] & Subsec.3.6.1.4). The DNS is used for session initiation; information regarding active locators – including changes in precedence (if required) – is exchanged between hosts dynamically, by using ICMPv6 Locator Update (LU) messages ([6, Sec. 3.2.2] & [104, Sec. III(A)]—and illustrated by [104, Fig. 7(c)])). LU message syntax is defined in [8].

ILNP requires a small set of changes to transport protocols (e.g. TCP, UDP) [7, Sec. 4] to accommodate the use of multiple locator values. Locator Selection & Rewriting are used to implement the “ID/Loc” split; [57, Sec. III] recalls the problem

\(^\text{17}\)Defined as “Locator Preference Indication” (LPI) in [6, Sec. 3.2, Par. 2].

\(^\text{18}\)The RR names in [7] are canonical; older publications use deprecated names.

\(^\text{19}\)A separate study in [101] has shown no significant impact on existing services when DNS Time-To-Live (TTL) was set to zero for A RRs.

\(^\text{20}\)The term “flow” is used throughout this work to disambiguate network and transport layers.
of *binding dynamicity* described in Chapter 2, and [20, Sec. II.D and Sec. II.G] expresses this in formal “tagged tuple” notation, illustrating how locator rewriting operates conceptually. Whilst the correspondence between ILNP sessions and transport layer sessions has not been formally defined, [6, Sec. 8, Par. 6] states that a many-to-one (M:1) mapping is expected. The implementation studied in Chapters 4–6 assumes a 1:1 mapping to simplify the treatment of Locator Update (LU) message; refer to Subsec. 4.3.2 for details underlying this choice.

3.6.1.2. *Site-controlled multihoming.* Whilst ILNP is – architecturally – a host-based proposal (and is thus described in survey articles [42, 105]), Table 3.1 shows that it also has site capabilities. Locator Rewriting & Updates may be implemented at Site Border Router (SBRs), e.g. to provide robust site connectivity using multiple locators. [9, Sec. 3] describes this engineering approach. Chapters 5–6 discuss how it has been implemented, emphasising Locator Update Snooping in Section 5.2.5 on page 99. Additionally, whilst this work is focused on host & site-controlled multihoming, mobility may be supported using the same mechanisms: e.g. [106] explains how mobile ILNP nodes may achieve “soft handover” using multiple concurrent locator values.

3.6.1.3. *ILCC: ILNP Communication Cache.* An ILNP node must track state for flows where it is an active participant. This state is held locally in a data structure called the ILCC [7, Sec. 5]. End hosts maintain state only for flows where they are endpoints as shown in Chapter 4 on page 53. However, ILNP-aware transit SBRs must track flows upon which they perform Locator Rewriting [9, Sec. 3.2], extending the ILCC beyond its description in [7, Sec. 5]. This is required in order to support link changes at the SBR (refer to Chapter 5 on page 93) and is also required to support distributed ILNP site forwarding (refer to Chapter 6 on page 121).

3.6.1.4. *Security.* The first few packets of a transport layer session must include the Nonce Option; its syntax is described in [103]. The Nonce Option is intended to protect against off-path attacks only. Nonce values may (optionally) be generated using a cryptographically secure random number generator (e.g. the *arc4random*
algorithm\textsuperscript{21} used by FreeBSD). Additionally, the use of DNSSEC \cite{107} is highly desirable – i.e. to authenticate DDNS updates – when ILNP is in use across the Internet. However, DNSSEC is not required for the deployment or operation of ILNP.

3.6.2. Caveats. ILNP cannot – on its own – solve the problem of IP entanglement shown in Table 2.1 on page 11; this problem is discussed further in Section 4.4 on page 67. Following the security discussion in Section 3.2.2 on page 29 & \cite{46}: Whilst ILNP does not attempt (nor intends) to solve the problems described in \cite{46}, there may be a false perception that ILNP adds security risks to IP. For example, whilst an attacker could inject ILNP packets containing faked identifiers, this is no different from the situation with IPv6 addresses today. Moreover, CGA and HBA addressing schemes may also be used with ILNP to enhance user privacy. So, ILNP adds very little to the existing IPv6 attack surface – other than its own limited signalling, i.e. in the form of the Nonce Option and Locator Update messages discussed further in Chapter 4.

Finally, whilst my implementation of ILNP has benefited from the flexibility of the BSD networking code, one work \cite[Pg. 11]{96} – describing a similar implementation effort for Linux – found that collisions arose between multiple identifier and nonce values (i.e. without address scope to disambiguate them). By contrast, scope forms part of FreeBSD’s \texttt{sockaddr\_in6{}() } representation of IPv6 addresses, and is also used for next-hop resolution. So, the issue of flow collisions is likely an implementation artefact, and has not been observed in this work.

3.6.3. Summary. ILNP is a host-based, end-to-end “ID/Loc” architecture, with optional engineering extensions providing site-controlled multihoming. It has two advantages over other proposals described in \cite{5}: it may be deployed incrementally without requiring any centralised co-ordination (i.e. a “flag day”) (as it may be implemented by extending IPv6), and it does not have the indirection problems described in \cite[Sec. 17.3]{5}. In addition, as it may co-exist with existing IPv6 mechanisms (e.g. SLAAC and DHCPv6), site renumbering continues to be supported through those mechanisms.

\textsuperscript{21}\url{https://www.freebsd.org/cgi/man.cgi?query=arc4random&sektion=9}
3.7. Summary of Proposals: Parts I—IV
The previous four sections have reviewed several proposals to implement or enhance multihoming in the Internet today – with a specific focus on on “ID/Loc” split architectures. Widespread interest has converged on the three ILNP, LISP and HIP proposals – each of which are at current RFC status and which have implementations available. However, HIP does not address the “routing bloat” issues described in Chapter 2, as its main focus is host identity. Additionally, a proposal to extend the use of NAT in a simplified way – NPTv6, described in Section 3.4.1 on page 32 – was reviewed: as summarised in Section 3.3.3 on page 31, several authors have identified NPTv6 as a useful transitional technology, despite the IETF recommending against its use. So, NPTv6 is used for comparative evaluation with ILNP in Chapter 5. The following section discusses the methodology of evaluations across Chapters 4-6.

3.8. Methodology
In this section I discuss how the evaluation scenarios in Chapters 4-6 were devised, i.e. in terms of network metrics, experiment controls, and the factors used to partition the data thus collected. In particular, frame size has a significant effect on system response; refer to Subsec. 3.8.2 for a fuller discussion. For background, Appendix B.2 on page 194 contains a detailed discussion of the measurement & analysis tools used in this work.

3.8.1. Metrics. The following metrics were collected at each node. I have used this data for three purposes: to contrast ILNP-in-IPv6 (ILNPv6) with IPv6, to evaluate the ILNP implementation, and to validate the focal theory at the core of this dissertation. This final point is discussed further below in Subsec. 3.8.1.1.

**Goodput:** This metric evaluates both per-packet processing time and relative protocol overhead: whilst similar to throughput, it is usually measured at the destination endpoint. It is particularly relevant to the comparative IPv6 evaluation in Chapters 4 and 5. A significant drop in goodput may indicate a design or implementation problem, or a configuration issue in experimental apparatus.
**Round Trip Time (RTT):** This provides a direct indication of relative per-packet processing time, although it includes transit time, serialisation, and receive/transmit overhead between two nodes.

**Packet loss:** A small amount of packet loss is to be expected on any network. However, loss may be an indication of implementation problems or misconfiguration. It may also be observed when one or more network elements on a path are experiencing congestion, or when approaching other system performance limits.

**Jitter:** This is also known as *packet delay variation* and is computed using a rolling window based estimator. High jitter can indicate that an on-path network element is experiencing congestion or performance issues.

**Byte (payload) loss:** This is a companion metric to packet loss, which is used where the underlying measurement tool either does not support explicit loss detection, or its loss estimates may be unreliable, e.g. *iperf*; refer to Subsec. 3.8.3.1.

**Network buffer utilisation:** This metric provides an indication of the memory overhead per packet. The networking stack *must* buffer received packets, and *may* buffer transmitted packets or fields. In addition, some networking stack elements may use additional network buffers – e.g. the *pf* NAT/firewall implementation, and ILNP itself which uses *mbuf* “tags” to hold nonce values.

**Memory allocation:** This provides an indication of relative memory overhead for data structures which are not part of each packet, e.g. routing state (see below), flow state, and/or identifiers. In FreeBSD these are tracked for each specific allocation arena in the system, i.e. by *malloc()* pool tag, or by zone ID.

**Link counters:** The FreeBSD networking stack counts packets received on each configured link. These are used to derive a distribution of received packets on each node by locator. This may be used to confirm that load sharing is taking place. However, link counters may only be cleared by rebooting the system.
In addition, various other metrics were collated from two categories: firstly, those which are specific to each evaluation scenario (e.g. peer activation delay measurements in Section 6.5.4 on page 145), and secondly, those which are specific to the measurement tool used (e.g. `iperf` sequence errors, discussed below in Subsec. 3.8.3.1).

3.8.1.1. Routing state allocation. Memory allocation is directly relevant to the focal theory in this dissertation: i.e. that multihoming is possible without introducing routing state. The FreeBSD kernel allocates routing state from a `malloc()` pool; these allocations are tracked for diagnostic purposes, and cumulative totals may be retrieved from the system at run time. So, before (and after) every trial, I have collected the pool counter for `rtentry` data structures. If the focal theory is correct, we would expect no change in this counter over the lifetime of each trial. Refer to Sec. 4.6, Sec. 5.4 and Sec. 6.5 for detailed discussion.

3.8.2. Factors. The data was partitioned by three categorical variables: Ethernet frame size, load rate, and load sharing configuration. The levels of the first two factors are listed in Table 3.2. The aim is to show that the relative per-packet performance of ILNP is similar across different load sharing configurations, and that it does not incur significant overhead as compared to unmodified IPv6. However, we might reasonably expect that where multiple source/destination locator pairs exist – e.g. the 50/50, 80/20 load sharing configurations – packet processing time may be increased.

**Ethernet frame size:** This factor has two levels: 128 and 1514 bytes. These levels were chosen to be representative of two common IPv6 workloads: small message dispatch (e.g. control messages, multimedia streams), and data transport (e.g. file downloads). The 1514 byte level was chosen as this reflects the default Maximum Transmission Unit (MTU) of 1500 bytes for IP on Ethernet, i.e. after the 14 byte Ethernet header is prepended. The 128 byte level was chosen to reflect the smallest\(^22\) useful message size which includes ILNP and IPv6 headers.

\(^22\)Networking equipment vendors often use a minimum payload size of 64 bytes when evaluating IPv4 performance.
**Load rate:** This factor depends upon Ethernet frame size. For each frame size, the system under study is driven at a discrete range of five load rates. The two ranges listed in the leftmost column of Table 3.2 correspond to uniformly spaced points on a 1-2-5 logarithmic scale and overlaps with common traffic rates (e.g. 10Mbit/s & 100Mbit/s).

**Load sharing configuration:** This factor is relevant to ILNP trials in Chapter 4 and Chapter 5 only. In Chapter 6, a single time-varying configuration is used instead. There are three factor levels: 100/0, 50/50 and 80/20. These are chosen to exercise ILNP with and without load sharing; the 80/20 case represents a “worst case” for packet reordering due to one path (and set of transit queues) being favoured more than others.

<table>
<thead>
<tr>
<th>Size (bytes)</th>
<th>Load (Mbps)</th>
<th>Mean pps</th>
<th>O(pps) scale</th>
<th>≈Kpps</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>1</td>
<td>976.563</td>
<td>$10^3$</td>
<td>1</td>
</tr>
<tr>
<td>128</td>
<td>10</td>
<td>9,765.625</td>
<td>$10^4$</td>
<td>10</td>
</tr>
<tr>
<td>128</td>
<td>20</td>
<td>19,531.25</td>
<td>$10^5$</td>
<td>20</td>
</tr>
<tr>
<td>128</td>
<td>50</td>
<td>48,828.125</td>
<td>$10^5$</td>
<td>50</td>
</tr>
<tr>
<td>128</td>
<td>100</td>
<td>97,656.25</td>
<td>$10^5$</td>
<td>100</td>
</tr>
<tr>
<td>1514</td>
<td>10</td>
<td>825.627</td>
<td>$10^3$</td>
<td>0.8</td>
</tr>
<tr>
<td>1514</td>
<td>100</td>
<td>8,256.275</td>
<td>$10^4$</td>
<td>8</td>
</tr>
<tr>
<td>1514</td>
<td>200</td>
<td>16,512.55</td>
<td>$10^5$</td>
<td>16</td>
</tr>
<tr>
<td>1514</td>
<td>500</td>
<td>41,281.374</td>
<td>$10^5$</td>
<td>40</td>
</tr>
<tr>
<td>1514</td>
<td>1000</td>
<td>82,562.748</td>
<td>$10^5$</td>
<td>80</td>
</tr>
</tbody>
</table>

Table 3.2. Load rate factors common to all experiments in Chapters 4-6.

3.8.2.1. *On Ethernet frame size.* The tools discussed in Subsec. B.2.1 use two sizes of measurement payloads, yielding 1514 byte and 128 byte Ethernet frame sizes respectively. To aid visual comparison, certain plots containing both frame size levels have been presented on identical X-axes within Chapters 4—6. However, as shown by Table 3.2, certain load rate levels exceed the performance envelope of the system, and may not be exercised for 128 byte frames. So, plots where the X axis has been rebased will contain some expected gaps (i.e. empty space).

Payload sizes were adjusted for two reasons: firstly, to keep the Ethernet frame size constant across trials using different tools; and secondly, to avoid the use of IPv6
packet fragmentation at hosts. This also simplifies comparisons between host and SBR scenarios, as IPv6 routers are forbidden to fragment packets [108, Sec. 4.5]. However, the system response can be quite different for varying packet sizes.

For 1514 byte frames, the load rate ceiling of 1000Mbit/s is chosen to exercise ILNP at the full line rate of a 1Gbit/s Ethernet link, as this is commonly used for networking today. For 128 byte frames, the ceiling of 100Mbit/s was chosen as this is close to the packet-per-second performance threshold of the system: i.e. the single CPU core measurements discussed in Subsec. 3.8.3 obtain lower goodput as the effective rate approaches 100Kpps. This is discussed further in Subsec. 3.8.3 and Subsec. 3.8.3.2.

3.8.3. Controls. Each evaluation scenario was controlled for several salient system level variables; these affected IP and ILNP equally. So, measurements reflect the relative performance of each protocol’s software implementation – and not the network hardware. These were – in descending order from the application to the network – as follows:

- the number of application threads – one, as this provided a consistent measurement basis for iperf and bwping\textsuperscript{23} which are described below in Subsec. B.2.1;
- the number of socket-layer queues – implicitly one, as FreeBSD does not implement intermediate queues for UDP or raw IPv6 sockets;
- the number of kernel threads – one interrupt thread for each input link, [109] i.e. input packets were immediately serviced using the next available CPU context;
- the number of link-layer queues (one for direction, at each node; refer to Appendix B.3.2 on page 200), i.e. to constrain the set of network elements where reordering may occur;
- and the length of the hardware transmit and receive queues at each node – which were set to their maximum upper limits (also in Subsec. B.3.2).

\textsuperscript{23}This tool supports only single threaded operation.
limited where packets may be dropped due to hardware queue congestion, by allowing queues to grow – although at the expense of increased latency.

Other factors limited the maximum throughput of user-space tools, e.g. `iperf` and `bwping` (described further in Appendix B.2.1 on page 195). These included processor speed, memory latency, context switch overhead, and system call overhead. For comparison, the highest packet rate achieved by `bwping` in my evaluation is ≈78Kpps, whereas `ng_source` – a single threaded kernel-space sender – can achieve ≈100Kpps; both figures are for 128 byte Ethernet packets at 100Mbit/s load with an ICMPv6 payload. The same experiment run in VMWare Fusion achieved ≈10Kpps [1, Sec. VI].

3.8.3.1. Limitations of `iperf` for multipath measurements. `iperf` can be used to measure the performance of protocols which may be forwarded over multiple paths – but with the caveat that it may overestimate packet loss. This is because of its simple error-counting algorithm; refer to Appendix B.2.2 on page 197. The issue affects `iperf` measurements generally, i.e. it is not limited to either IP or ILNP and it affects both protocols equally. In multipath scenarios, reordering is inherent to the network – and so sequence displacement will be seen in normal operation. `iperf` calculates error and reordering based on a rolling sum of displaced packets. This can be seen in Fig. 4.6.6 on page 84 and Fig. 4.6.5 on page 83. So, to provide an indication of loss observed by `iperf` at the application layer, I have also collated payload (byte) loss statistics, which in these cases is typical for a LAN (<1.5%). So, `iperf` error is not the same as real packet loss, which it can only estimate (by sequence). Whilst this behaviour has been observed in this work with unidirectional flows, two-way `iperf` flows were also affected.

3.8.3.2. On `iperf` performance limits. In all evaluation scenarios studied in this dissertation, `iperf` achieved between ≈62%—≈72% goodput at 1Gbit/s – the highest

\[24\] Whilst tools such as `netmap` may reduce queueing delay – by enabling network adapter queues to be mapped directly into an application address space – its use bypasses the ILNP state associated with the socket layer, making it unsuitable for the evaluation herein.

\[25\] https://www.freebsd.org/cgi/man.cgi?query=ng_source
load rate used in these studies. These results are for a single-threaded\textsuperscript{26} \texttt{iperf} client. It can be seen from the results in Sec. 4.6, Sec. 5.4 and Sec. 6.5 that \texttt{bwping} consistently outperforms \texttt{iperf} at this rate, achieving $\approx 91\%$ goodput. The reason for this performance gap was twofold, and it is also influenced by the system variables discussed above in Subsec. 3.8.3.

Firstly, the timing loop in \texttt{bwping} is simpler; it required fewer \texttt{gettimeofday()} system calls. The relative overhead of these calls is discussed in Appendix B.3.3 on page 201. Secondly, both the \texttt{iperf} client and server ran as user-space processes, whereas the response to \texttt{bwping} probe packets was generated by the kernel-resident ICMPv6 reflection code (discussed further in Section 4.4.2 on page 68). In addition, concurrency control is simpler for ICMPv6 – as, unlike UDP, it has no port numbers. By contrast, both UDP transmission and reception require an additional mutex lock acquisition to prevent race conditions. So, the performance gap between UDP and ICMPv6 results was expected, as \texttt{bwping} is normally able to transmit – and the kernel may process received packets – at a faster rate than \texttt{iperf}, which runs as a user-space process at both endpoints.

3.8.3.3. \textit{On packet pacing error in bwping}. I have identified that \texttt{bwping} may underfill the transmission link by up to $\approx 9\%$ at a requested load rate of 10Mbit/s, although this issue does not affect all load rates. In order to meet a requested load rate, both \texttt{iperf} & \texttt{bwping} must implement some form of packet pacing to regulate their packet transmission rates. These typically operate at a millisecond time scale. However, there are considerable differences between their respective implementations, which are discussed further within Appendix B.2.3 on page 198.

To summarise, there are two principal factors involved in this specific case: firstly, \texttt{bwping} uses integer arithmetic to calculate transmission rates, and its treatment of quotient division can introduce cumulative error. Secondly, whilst it adjusts its packet transmission interval, it does not adjust the size of packet bursts at affected rates. The resultant effects of such poorly aligned or overlapping timer intervals are described well by [110, IV.F], which considers how they arise in cases where the Linux kernel

\textsuperscript{26}The use of multiple threads is problematic as each source thread is effectively treated as an independent flow, complicating interoperability with the ILNP code.
must operate at the same millisecond time scale – although for the more complex case where multiple packet priorities must be supported (not examined in this work).

3.8.4. Summary. In order to study the relative performance of ILNP in comparison with existing protocols – i.e. IPv6 & NPTv6 – I have implemented it within the FreeBSD networking stack. Whilst the final set of changes required for this are limited (refer to Chapters 4-6), the networking stack has a high degree of coupling in software engineering terms; so, the development process has benefited from the use of specific software tools (e.g. KScope, as described in Appendix B.1 on page 193). The factors and controls used in experiments have been chosen to enable a hardware-independent comparison of how each protocol performs, as the aim is to compare software performance, and not hardware; more specifically, this was achieved by permitting queue growth at each network element. Existing measurement tools – e.g. iperf, bwping & fping – have been re-used, with some modifications to enable their use in multipath networking scenarios with ILNP; however, the measurement process used by iperf has certain limitations in such scenarios. The number of trials required to obtain a statistically valid sample size mandated automation through shell scripting; additionally, similar automation was required for post-processing & analysis. Finally, I have re-used several well known tools from the SciPy ecosystem to perform this analysis.

3.9. Chapter summary

In Chapter 2, I have discussed general approaches to address the problem of routing table bloat (refer to Subsec. 2.4). In this chapter, I have described specific proposals which follow these approaches. These consist of host multihoming proposals (e.g. ILNP, HIP & SHIM6) and proposals for site multihoming (e.g. LISP & Nimrod). Additionally, I have examined two concurrent multipath transport protocols: SCTP & MP-TCP, and an approach to NAT tailored to IPv6: NPTv6.

Host-based approaches have the advantage that they do not generally require changes to routers; however, their deployment may affect site networks, e.g. by requiring firewall configuration (or support) for new extension headers. Moreover, they do not address the issues of site network renumbering, or the routing table bloat.
introduced by advertising multiple prefixes to reach a multihomed site. Additionally, whilst multipath transport protocols permit the use of multiple network paths by applications, SCTP requires that applications are rewritten. In addition, both SCTP & MP-TCP inherit most of the disadvantages of host-based “ID-Loc” proposals described in this chapter.

By contrast, site-based approaches do not generally require changes to applications or hosts. In this space, LISP is not an entirely “clean-slate” approach: it proposes to re-use some networking technologies which exist today (e.g. BGP & GRE). The proposal is complex and requires many changes to existing routers (perhaps the deployment of new routers); despite these obstacles, a trial network now exists.

However, ILNP is unique amongst the host-based proposals described here. It re-uses the existing IPv6 naming, routing & addressing systems, following the work of the GSE/8+8 proposals. The goal of this dissertation is to show that it can be implemented efficiently, and with few changes to the existing IPv6 networking stack; the following three chapters demonstrate its implementation and use across several common end-site scenarios. Whilst it requires changes to the end-host stack, no shim layers or tunnels are required; and, unlike SHIM6, it does not require additional IPv6 addresses for signalling.

Finally, in this chapter I have described the methodology used within the evaluations in Chapters 4—6. This work required the application of measurement tools tailored to the area of evaluation – i.e. the IPv6 network layer – and highly automated data collection. In addition, the factors & controls used were chosen to permit hardware-independent comparison of the ILNPv6 & IPv6 network layers. The collection of allocation statistics from the FreeBSD kernel allows the focal theory to be validated: i.e. that ILNP does not introduce routing state to perform multihoming functions.

\[27\text{The requirement for an additional sub-protocol – i.e. REAP, as mentioned in Subsec. 3.5.1 – may be a side-effect of SHIM6 relying upon a separate identifier namespace. By contrast, ILNP does not require such an addition, as the IPv6 address syntax remains unchanged.}\]
Chapter 4

Host multihoming with ILNP

4.1. Introduction

In this chapter, I present an evaluation\(^1\) of host multihoming as a prototype implementation of the Identifier Locator Network Protocol (ILNP) on FreeBSD, as a super-set of IPv6 – called ILNPv6. I demonstrate load sharing using ILNPv6 multihoming, and compare performance with IPv6 forwarding at the end host. The main finding is that ILNPv6 may be implemented at hosts with relatively low overhead, i.e. without introducing additional routing state, and transparently to existing IPv6 applications.

4.1.1. Contribution. The main contributions of this chapter are: (A) to demonstrate how host-based ILNP may be realised as a set of extensions to an existing IPv6 networking stack; (B) to show how ILNP Locator Selection may be used to implement load sharing – a common multihoming application, as described in Chapter 2; (C) to demonstrate the use of ILNP by unmodified IPv6 applications; and (D) to show that the ILNP extensions have very low overhead, as compared with unmodified IPv6.

4.1.2. Chapter structure. This chapter begins with a description of the problem space in Subsec. 4.1.3, followed by a discussion of what is specifically required to implement ILNP at hosts—divided into three sections:

(I) Section 4.2 on page 56 introduces how key ILNP concepts – e.g. network locators, identifiers and names – correspond to existing IPv6 structures;

\(^1\)This chapter is based on a previously published study conducted in a VM environment [1].
(II) Section 4.3 on page 61 describes host ILNPv6 forwarding and what changes to support it in the IPv6 network layer;

(III) Section 4.4 on page 67 discusses differences between IPv6 and ILNPv6 in the socket layer, introduced to support its semantics.

My evaluation in Sec. 4.5 is framed as a comparative study in relative performance with IPv6, i.e. to explore the lightweight approach to the “ID/Loc” split proposed by ILNP. The results are presented and discussed in Section 4.6 on page 77, and the chapter concludes with a summary in Sec. 4.7.

4.1.3. **Problem space.** Host multihoming in IP allows individual hosts to be multiply connected to the network, e.g. by concurrent use of two (or more) network prefixes, each network prefix tied to a separate network interface – as shown in Fig. 4.1.1. Such multihoming capability improves the host’s ability to implement various network functions – e.g. load-balancing, fail-over and multi-path transport.

However, IP does not directly support host multi-homing today, and additional routing state must be added to support it. By contrast, ILNP is able to support such network functions natively in its architecture – by forwarding based on locator selection, rather than addresses alone. Moreover, ILNP may be engineered as a set of extensions to IPv6.

![Figure 4.1.1. Scenario demonstrating host multi-homing within a site.](image)

4.1.4. **Approach.** ILNP defines an “ID/Loc” split architecture, and may be implemented as a set of extensions to IPv6. The claim is that ILNP does not add routing state – i.e. beyond that which is already required for normal routing in IPv6. As, architecturally, ILNP is radically different from IP – it does not use addresses – my implementation considers (1) what needs to change in current stack engineering in order to enable ILNPv6; and (2) what impact this could have on performance.
My implementation is based on FreeBSD 8.4, and mostly resides in a separate ilnp6 kernel module. Although some modifications must be made to the existing IPv6 stack, the additional features of a policy applied to the forwarding plane (i.e. locator selection, described in Section 4.3.3 on page 62) may be used transparently by IPv6 applications. Whilst I have taken a dual-stack approach to implementation – as described in [6, 7] – I do not claim that the implementation is fully dual-stack, for reasons discussed in Section 4.4.3 on page 70.
4. Method: Part I—Naming and Locators

This section discusses architectural components of ILNP, particularly those relevant to its implementation as a super-set of IPv6.

4.2.1. Identifier-Locator Vectors. The term *Identifier-Locator Vector (I-LV)* [6, Sec. 3.3] refers to an ID/Loc pairing encoded into an IPv6 packet, as shown in Fig. 4.2.1. This is key to understanding how existing IPv6 functionality may be reused to implement ILNPv6, including the address assignment mechanisms. The L64 value is an existing IPv6 routing prefix and has the same semantics. *So, no changes to core network routers are required to forward ILNPv6 packets.* The NID value is generated as for IPv6, but interpreted differently in the end-system stack.

One motivation for the re-use of IPv6 address bits is that no changes are necessary to any IPv6 network elements, other than those end-hosts which participate directly in the ILNPv6 protocol. For IPv6 hosts, the scope of my changes is limited to a subset of the networking stack. Therefore, IPv6 and ILNPv6 packets ‘on-the-wire’ are largely identical – excepting the use of the Nonce Option, discussed in Subsec. 4.3.7.

4.2.2. Local identifiers. My implementation re-uses the existing IPv6 interface identifier (*IID*) mechanism of RFC4291 [18, Sec. 2.5.1]. This allocates a unique *IID* for each interface based on hardware address; refer to [111, Sec. 2.2] for details of the IEEE EUI64 address format used to represent them.

ILNPv6 re-uses the *IID* values as node identifier (*NID*) values [6, Sec. 3.1], and re-uses their existing representation. A *NID* value need only be unique within the scope of its Locator, but using existing IPv6 mechanisms, we can achieve the same

```
in6_addr{}

<table>
<thead>
<tr>
<th>IPv6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001:DB8:D00D::0001:0203:0405:0607</td>
</tr>
</tbody>
</table>
```

```
in6_addr{}

<table>
<thead>
<tr>
<th>L64</th>
<th>NID</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001:DB8:D00D::</td>
<td>00-01-02-03-04-05-06-07</td>
</tr>
</tbody>
</table>
```

Figure 4.2.1. IPv6 addresses compared with Identifier-Locator Vectors (I-LVs) [7, 18]
level of uniqueness for \textit{NID} values as for IPv6 \textit{IID} values. Where such an identifier is not available, the behaviour is implementation specific, although the IPv6 address (or I-LV for ILNPv6) must be unique within a subnet prefix [18, Appendix A].

4.2.2.1. \textit{Implementation}. For convenience (to aid development and debugging), I have modified the FreeBSD kernel to use some bits\textsuperscript{2} from an MD5 digest [112] of the host name as a \textit{NID} value. The same value may be used across all interfaces in ILNPv6.

\textsuperscript{2}FreeBSD uses an MD5-based \textit{IID} if a hardware-derived \textit{IID} is unavailable.
4.2.3. Routing prefixes for locator values. In order to discover local $L64$ values, ILNPv6 re-uses the existing IPv6 Router Advertisement (RA) mechanism. The format of RA messages and their handling by the IPv6 stack are unchanged. However, I have added notifications of prefix-related events (e.g. advertisements, expiry of existing prefixes, or explicit withdrawal of a prefix by an on-link IPv6 router). These notifications were implemented using FreeBSD's EVENTHANDLER(9) mechanism [113]. When the prefix list changes, ilnp6 updates the local list of $L64$ values.

4.2.3.1. Locator precedence. Locators have an additional property: precedence. This allows multiple $L64$ values to be used simultaneously, and interpreted as local policy dictates. This property is not native to IPv6, therefore it must be added to the network stack. It is represented by a 16-bit unsigned integer value, with lower values being preferred.

So, during system initialisation, default precedence values are loaded from a local policy table; refer to Subsec. B.5.1 for syntax. The table is managed using a new system command, ilnp6locctl. The implementation was derived from FreeBSD's ip6addrctl tool, which manages IPv6 default address selection policies [114]. This simple scheme was sufficient for the evaluation in Sec. 4.5; a full implementation might distribute local policy in DHCPv6 options [115].

4.2.4. Peer I-LV discovery. ILNP applications are intended to use names, whereas IP applications must use addresses. However, a node initiating an ILNPv6 flow must first learn a remote ILNPv6 node’s $L64$ and $NID$ values [7, Sec. 6]. Moreover, in ILNP, multiple remote $L64$ values may be used simultaneously. So, it would be advantageous – for reasons of both performance and robustness – to initiate flows with knowledge of all locators where a node can be reached. Therefore, both $L64$ and $NID$ values may be advertised in the existing Domain Name System (DNS);
new Resource Records\(^3\) (RRs) – e.g. the \(L64\) and \(NID\) record types, respectively \([97]\) – have been defined\(^4\) to represent them.

4.2.4.1. **Multiple remote \(L64\) values.** The DNS supports multiple matches for a single DNS query, which are suitable for representing multiple remote \(L64\) values for a node. To resolve names to addresses, IPv6 applications typically\(^5\) use the portable `getaddrinfo()` API \([116]\) ; it supports *multiple match results* by design – i.e. as a linked-list of `struct addrinfo` records \([116, Pg. 26, 3rd Para.]\) – and so is suitable for retrieving multiple \(L64\) values in application code.

However, IPv6 applications might discard records beyond the first result, depending on how they are written. Additionally, the implementation of this API *may* sort these records using the prefix selection criteria in \([114]\). So, I have modified the API to return all matching values from `/etc/hosts` in order of *locator precedence* (i.e. *not* using the criteria specified in \([114]\)). The DNS is not used in experiments, as the subject of study is the networking stack itself; so, whilst these modifications reside in the DNS stub resolver code (part of the C runtime in FreeBSD), the same code is shared between several naming service back-ends. Backwards compatibility with IPv6 applications is discussed in Subsec. 4.4.3.

---

\(^3\) ILNP’s \(NID\) and \(L64\) DNS RRs are analogous to the \(AAAA\) resource record (RR) used for IPv6 addresses.

\(^4\) Commercial DNS implementations now support these RRs. At the time of writing, NLNetLabs `NSD v3.2.15`, ISC BIND 9.9.3/9.8.5/9.6-ESV-R9, and Knot DNS 1.3 were the first versions of recursive DNS server software known to support RFC6742.

\(^5\) The `getaddrinfo()` API was originally specified as a an IP-version-agnostic API – suitable for rapid transition to IPv6. Other naming APIs offer only a subset of its functionality, and so I do not examine them further here.
4.2.5. **Extended /etc/hosts syntax.** A host’s DNS resolver may be configured to use the /etc/hosts file to resolve names. This may be used for bootstrapping, or as a fallback measure when DNS is unavailable – or by explicit configuration. For IP, this file contains a static mapping of names to network addresses. I have extended the /etc/hosts syntax to include a 1:M mapping of names to I-LVs, as shown in Fig. 4.2.2. My evaluation in Sec. 4.5 uses this file to simulate the content of DNS-based resolver results.

```
#
# /etc/hosts file extended syntax for ILNPv6
#
# An entry -- an I-LV record -- has the structure:
# L64|lprec,NID hostname
#
# with the following fields:
#
# L64  64-bit Locator value (in IPv6 address format)
# lprec the Locator’s precedence value
# NID  64-bit Node Identifier value (in Canonical EUI64 format)
# hostname a valid hostname value
#
# Example entries are:
2001:0db8:d00d:0:0000:10,02-1f-5b-ff-fe-ff-13-74  foo.yoyodyne.com
2001:0db8:cafe:0:0000:20,2a-37-37-ff-fe-1c-cf-fe  bar.yoyodyne.com
```

**Figure 4.2.2.** The extended syntax in /etc/hosts for ILNPv6
4.3. Method: Part II—Network Layer

In this section, I describe the changes required at the network layer to implement ILNP for hosts. Whilst the discussion is general to IPv6, these are followed with FreeBSD-specific implementation detail where relevant. To avoid confusion between the ILNP network layer and upper transport layers, I use the terminology of “ILNP flow” and not “ILNP session” as used in [6] – throughout this chapter. This naming convention was adopted as part of my contribution to the relevant ILNP RFCs described in Section 3.6 on page 39.

4.3.1. Flow initiation. Having discovered a peer’s identifier (NID value) and locator set (L64 values), the node may then initiate an ILNPv6 flow with that peer. However, three steps are required to establish an ILNPv6 flow, as follows (and, after flow initiation, steps 2 and 3 are repeated for each packet):

(1) Matching and/or allocation: ILNP is interstitial in structural terms. Throughout this work, I assume that there is a 1:1 mapping between transport layer sessions and ILNP flows – discussed further in Subsec. 4.3.2 – to simplify the behaviour of Locator Update (LU) messages within each evaluation scenario. My modifications to the sockets API – discussed in Sec. 4.4 – mostly affect existing operations per-packet. So, ILNP flows are allocated on demand if an existing flow cannot be found for a transport layer session.

(2) Locator selection: This stage is responsible for choosing both locators (and, for hosts, identifiers also – in place of IPv6 Source Selection [114]) for new and existing flows. [6, Sec. 3] describes the relation between both locators and identifiers and transport sessions, using “tagged tuple” notation in [6, Subsec. 3.4].

(3) Locator rewriting: here, the salient fields of an ILNPv6 packet are set, in IPv6 syntax: e.g. source and destination L64 values, the ILNPv6 Nonce Option (prepended on demand where required, e.g. initial session packets [7, Subsec. 10.6]); and (optionally, for hosts), the IPv6 Flow Label (refer

---

6In term of the ISO-OSI [43] model, it would reside at “Layer 3.5” – i.e. above the network layer, yet below transport layers.
4.3.2. **On ILNP flows & upper layer protocols.** The relevant ILNP RFCs [6, 7, 9] do not specify an explicit mapping between between IP transport protocol sessions and ILNP flows. A 1:1 mapping is adequate for the majority of unicast traffic, and was originally specified in draft versions of these documents – prior to my contributions described in Section 3.6 on page 39. However, there are several 1:M communication scenarios which arise in normal UDP/IP network operation: e.g. DNS servers, BitTorrent, and broadcast/multicast Remote Procedure Calls (RPCs). These require careful adaption of the existing scheme, in order to avoid generating redundant bursts of LU messages containing identical information. This caveat is otherwise unrelated to the re-use of IP addresses by applications described in Section 7.4.2 on page 159.

4.3.3. **Locator selection & rewriting.** ILNP nodes may have many \( L64 \) values. The protocol does not impose any defined behaviour on **locator selection**, i.e. how a specific I-LV is resolved to its final destination. This flexibility permits several network layer functions (e.g. multihoming and load-balancing) to be implemented at the host – without requiring “middleboxes”, as are often used today. Site administrators may define **locator selection policies** to achieve the desired behaviour, using the precedence table described in Subsec. 4.2.3.1.

Locator selection is performed for every packet transmission individually, although some essential state is cached to achieve reasonable performance. To demonstrate its capabilities and operation, I have implemented a simple load sharing algorithm; the flow of control is shown by Fig. 4.3.1 on page 64. Time-contingent properties have been omitted for clarity – e.g. where a locator is temporarily unavailable, as its state has not yet been transmitted to the peer.

4.3.3.1. **Implementation.** The \( L64 \) values for the packet’s source and destination \((s_{ilv}, d_{ilv})\) are rewritten [20, Sec. II-G] according to the locator set(s) of the flow \( S \). Next-hop information for each locator is provided by the IPv6 FIB, as discussed in Subsec. 4.3.5. Unreachable locators are rejected and are not used for load sharing.
4.3. METHOD: PART II—NETWORK LAYER

The remote locator set is stored in an individual tail queue for each active flow, i.e. as part of the ILCC discussed in Section 3.6.1.3 on page 41 and [7, Sec. 5.4]. This choice was made for ease of implementation, rather than performance. Transport protocol checksums are rewritten using a checksum-neutral mapping [65]. Whilst host-based ILNP re-uses the IPv6 transmission code in `ip6_output();` locator selection is never performed for IPv6 packets.

4.3.4. Locator updates. Once an ILNPv6 flow has been established, no further DNS lookups are necessary. Flows use the ICMPv6 Locator Update (LU) [117] message to communicate further changes between peers (e.g. new connectivity, explicit withdrawals, and changes in locator precedence). Local events which modify the Locator set – e.g. as described in Subsec. 4.2.3 – will trigger the transmission of LU messages. My implementation defers transmission within 500ms to prevent bursts of redundant LU messages.

LU messages must include the Nonce Option – described in Section 4.3.7 on page 66 – to both authenticate their source, and to identify the ILNP flow for which they provide locator state information. Outgoing LU messages are also subject to locator selection as discussed in Subsec. 4.3.3. However, the implementation described here may allow the IPv6 stack to override these choices, due to its concurrency control requirements.
Figure 4.3.1. Flowchart for the locator selection procedure (the reverse-type box in Fig. 4.4.1). The plain (white) boxes steps are common to all flows. The shaded (orange) boxes are specific to load sharing, where the boolean VLB will be set to TRUE.

Parameters \( s \) and \( d \) denote source and destination I-LVs. Each flow has a locator set \( \text{rls} \) advertised by the remote peer. The set of candidate local locators \( \text{clocs} \) is ordered by \( \text{prec} \); each has a count of transmitted packets \( \text{tx} \), and is associated with an outgoing link \( \text{oif} \). \( \text{loc}() \) and \( \text{nid}() \) refer to the upper and lower 64 bits of an I-LV. Concatenation of L64 and NID values to form I-LVs is also denoted by the \( + \) symbol.
4.3.5. Re-use of IPv6 unicast FIB. The locator selection procedure described in Subsec. 4.3.3 requires knowledge of locator state, including reachability and next-hop information. Much of this information already exists in the IPv6 FIB. So, to re-use existing code, the ILCC contains a locator-to-FIB mapping – shown in Fig. 4.3.2 – where remote locators are associated with existing IPv6 FIB entries. Additionally, the ILNP code will make on-demand calls to refresh this mapping when the state of locator(s) used by an active flow changes, e.g. upon the events described in Subsec. 4.2.3.

The technique of FIB caching is in common use, e.g. in FreeBSD (and other implementations, also), transport protocol sockets may cache the FIB entry used to reach the (most recent) destination. Whilst using a FIB cache is an engineering optimisation, this approach permits a high degree of code re-use, and facilitates integration with the existing IPv6 stack.

4.3.5.1. Implementation. The ilnp6_loc_rtstale() function – shown in Fig. 4.3.1 on page 64 – looks up the next-hop corresponding to a remote locator, and caches

![Diagram](image)
its result. Stale entries may be refreshed by an indirect call to the next-hop lookup\(^7\) function \texttt{rtalloc()}, [118] which is a general part of the FreeBSD network stack. This function is also referenced in the simplified call graphs in Fig. 4.4.1—4.4.2.

### 4.3.6. Concurrency control

For my implementation of host-based ILNPv6, most operations are serialised by a single mutex lock [119]. This choice was made for simplicity; the aim is to show that it is possible to modify and extend the existing IPv6 source code to provide ILNP-based multihoming at hosts, rather than to provide an optimised implementation. FreeBSD may have specific requirements for concurrency control in certain contexts, e.g. ICMPv6 reflection as discussed in Subsec. 4.4.2.

### 4.3.7. Security

ILNPv6 defines a new IPv6 destination\(^8\) option: the \textit{Nonce Option} shown in Fig. 4.3.3 [103]. It is defined for use only by ILNPv6.

```
<table>
<thead>
<tr>
<th>NextHdr</th>
<th>DLen=0</th>
<th>Type=0x8B</th>
<th>OptLen=4</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonce value</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Figure 4.3.3.** An ILNPv6 Nonce Option value – encapsulated in an IPv6 Destination Option header [103]

The Nonce Option contains a 32-bit (or, optionally, 96-bit) value unique to each flow (for a peer), derived using the MD5 hash algorithm [112]. The first packet of an ILNPv6 flow must include this option in its headers. However, it is not required\(^9\) in all flow packets. This option has two roles: (1) it flags a packet as being an ILNPv6 packet and (2) it enables a handshake for ILNP flow initiation. It also protects against off-path attacks on ILNPv6 flows. If such basic security is insufficient, then the use of IPsec is recommended [120, Sec. 4.4], with Identifier values used as part of the IPsec Security Association in place of IP addresses; refer to Section 3.2.2 on page 29 for a discussion of security issues.

---

\(^7\)Historical note: [16] describes 4.4BSD-era forwarding APIs, and the lookup strategy used a general-purpose PATRICIA trie. As of writing, FreeBSD stores FIB information in discrete radix tries for each protocol family.

\(^8\)Whilst the Nonce Option is transmitted as an IPv6 destination option, it may be examined by SBRs; refer to Section 5.2.1 on page 95.

\(^9\)To ease implementation, my evaluation in Sec. 4.5 includes the Nonce Option in all packets.
4.4. Method: Part III—Sockets

This section describes changes necessary within the sockets API – as used by IPv6 applications – to support ILNPv6. Some of these changes are required for the evaluation in Sec. 4.5.

4.4.1. Location-independent sockets. ILNP sockets are independent of network location, and transport protocols use only identifiers in the binding of their endpoints. However, in IP today, sockets may become bound to specific network locations as a side effect of how IP addresses are used; e.g. Table 2.1 on page 11 shows that the IP address is used as an identifier. This use is pervasive and historical [121]. Transport protocols (e.g. TCP, UDP) differentiate between sockets, by combining addresses and port numbers (for both local and remote endpoints) to form the 4-tuple. When the `bind()` or `connect()` APIs are invoked, a next-hop lookup is performed and an appropriate source address is chosen.

4.4.1.1. Binding dynamicity. Binding can be expressed formally using “tagged tuple” notation, as in [20, Sec. II-G]. For both IP and ILNP, the values of the port numbers $p$ and $q$ are chosen by the UDP implementation – and not by the network layer. However, IP treats binding according to Tuple (4.4.1), where the local address $A_L$ and the remote (peer) address $A_R$ cannot change. This implies that IP sockets are bound to a specific link (i.e. the link used to reach the next-hop to the destination), as neither the local nor the peer address may be changed – without disrupting existing connection state.

\[
(UDP : A_L, p, A_R, q) \langle IP : A_L, A_R \rangle
\]

The structure of the 4-tuple does not change for ILNP sockets; both addresses are interpreted as I-LV values. Locator selection (discussed in Subsec. 4.3.3) treats both the local locator $L_L$ and the remote locator $L_R$ in Tuple (4.4.2) as mutable; so, $L64$ values may be rewritten.

\[\text{10\textsuperscript{th}}\text{This can happen implicitly as an engineering optimisation; e.g. when a packet is first sent on a UDP socket.}\]
4.4.2. ICMPv6 and UDP – in ILNPv6. My implementation does not support the use of arbitrary IPv6 transport protocols with ILNPv6. Only raw IPv6, ICMPv6 reflection, and UDPv6 are supported – i.e. only what is needed to evaluate the ILNP network layer using ping6, bwping and iperf (in Sec. 4.5). The call graphs shown in Fig. 4.4.1-4.4.2 illustrate my changes to control flow in the FreeBSD networking stack. In all cases, ILNP operation is indicated by setting a dedicated flag in the socket’s protocol control block (PCB), discussed further in Subsec. 4.4.3.

Fig. 4.4.1 shows a combined call graph for transmit/receive of ICMPv6 ECHO messages. The socket layer functions are omitted for clarity. The raw IPv6 output path – rip6_output() – creates ILNPv6 flow state on demand at the sender; this
is sufficient to support ping6 transmission. The diagram has been simplified by showing by locator selection as a direct call to the ilnp6_mls_select()\textsuperscript{11} function. The absence of dashed lines below this function shows that ILNPv6 does not use IPv6 source selection.

However, the networking stack itself – and not an application process – is responsible for transmitting the ECHO Response. To use ILNPv6 here, the ICMPv6 reflection function – icmp6_reflect() – must also be extended to look up (or create) the flow associated with each input packet, and perform locator selection\textsuperscript{12} for the response packet transmitted by the kernel.

\textsuperscript{11}Transport layer protocols (e.g. ICMPv6, UDP) invoke a wrapper function ilnp6_select_locs(); this is omitted here for concise comparison with the behaviour at ILNP forwarders, described in Chapters 5-6.

\textsuperscript{12}The diagram in Figure 4.4.1 also shows that the calling function icmp6_input() – and not icmp6_reflect() – invokes locator selection. This choice was made to simplify concurrency control, by putting ILNPv6 state in the caller’s scope.
### 4.4.3. Compatibility with IPv6 applications

It would be desirable to allow IPv6 applications\(^{13}\) to use ILNPv6 without changes. Here, I do not attempt to address the issues raised by the 1:M or broadcast uses of UDP sockets (e.g. DNS and broadcast RPC), or peer-to-peer protocols (e.g. BitTorrent). So, I confine my discussion to 1:1 unicast flows, where – even in this constrained use case – a subset of these issues is encountered, discussed in Subsec. 4.4.4.

4.4.3.1. Naming. The naming APIs and sockets APIs are orthogonal to each other in terms of how they are used (and often their implementation, also). Consider that a typical IPv6 application (e.g. `ssh` or a World Wide Web browser) will resolve

---

\(^{13}\)Backwards compatibility between IPv6 and ILNPv6 hosts is described in [6, Sec. 8].
a name to an IPv6 address using `getaddrinfo()`, and establish a connection to a remote site by using the `connect()` function from the sockets API. This function accepts network addresses, not names, i.e. sockets are not visible to `getaddrinfo()`, and names are not visible to `connect()`. Other approaches (e.g. system call wrappers or `setsockopt()` calls) would require changes to the API (and, by extension, applications).

4.4.3.2. Implementation. I use a look-aside cache in my implementation to enable the transparent use of ILNPv6 by unmodified IPv6 applications. The cache works in conjunction with the DNS stub resolver at the host; i.e. an I-LV set is obtained as the result of looking up the name of an ILNP node using a naming service – e.g. the DNS, or the extended `/etc/hosts` file discussed in Subsec. 4.2.4 as used in my evaluation in Sec. 4.5. The cache is used to translate the application’s use of an I-LV to this I-LV set, just as if an IP address had been provided to the sockets API.

In FreeBSD the socket API is part of the kernel, whereas the naming API is part of the C runtime library, and cannot directly manipulate sockets. So, a down-call to the kernel is required. I have used the `sysctl` API [122] to populate the look-aside cache when an I-LV is resolved from a name within the DNS stub resolver. So, when a socket API is invoked, the look-aside cache is examined for an I-LV value corresponding to the IPv6 address passed by the application. If there is a positive match, then the first socket API call which matches the I-LV will use an ILNPv6 flow (allocated on demand), and the socket will transition to ILNPv6 operation. Entries persist for 1000ms after the last socket API call, and IPv6 binaries are unchanged. Finally, the calling thread and process IDs are recorded with each I-LV, to prevent inadvertent use of ILNPv6 by other applications.
4.4.4. On IP bits used within applications. The look-aside cache described here is sufficient for an ICMPv6 application (e.g. ping6, or bwping) to use ILNP without source or binary modification. However, this approach is not sufficient for applications which use the IP address internally as an identifier – or, indeed, any of the IP address bits – in application code; specifically, [34, Sec. 2, “Referrals”] is describing the same problem. Both the HIP and SHIM6 proposals (in Chapter 3) advocate API changes to address this problem, and might be examined for ILNP in future work; refer to Section 7.4 on page 158.

One specific example is iperf – used for the evaluation in Sec. 4.5 – which uses the IP address to discriminate between logical streams of measurement packets. This usage is an engineering convenience; it enables iperf to share code for TCP and UDP flows. It does this by internally mimicking the accept() socket API for UDP sockets, i.e. a new socket is created for each distinct UDP 4-tuple entry.

4.4.4.1. Implementation. Normally, ILNP will deliver each UDP packet with its received I-LV values. It would be problematic to modify iperf to behave differently, as its design depends upon the accept()-like behaviour. Moreover, as IP address bits are used directly, the DNS-based look-aside cache described in Subsec. 4.4.3 cannot be used to perform this translation; it must be performed within the sockets API. So, before the UDP packet is delivered on the socket, ILNP will rewrite the destination I-LV to the first I-LV seen in the flow – denoted by $L_1$ in Tuple (4.4.3). Only packets destined to port 5001 – i.e. the default iperf port – are rewritten by ILNP, to support the use of iperf in the evaluation.

$$\langle UDP : I_L, 5001, I_R, q \rangle |\langle ILNP : L_L, L_1 \rangle$$

4.4.5. Summary of Method: Parts I—III. In Sec. 4.2, I have described some similarities and differences between IPv6 and ILNPv6 which are key to understanding the latter: e.g. the re-use of the existing IPv6 address syntax, and how IPv6 routing prefixes may often be treated as locators, also. Following this, in Sec. 4.3 I explain how the ILNPv6 forwarding plane differs from that of IPv6, yet – as ILNPv6 is intended to be transparent to existing IPv6 network elements – much of the existing
architecture may be re-used – facilitating an efficient, dual-stack implementation. Finally, in Sec. 4.4 I show how these architectural features have been integrated with the FreeBSD IPv6 stack, enabling its use by unmodified IPv6 applications. Whilst the implementation is constrained, it is sufficient for the comparative evaluation between IPv6 and ILNPv6 in Section 4.5 on the following page.
4.5. Evaluation

Given that ILNP adds state to the node, we would expect a modest increase in memory usage by the networking stack, and a slight increase in round-trip time, accounted for by the processing involved with multi-homing functions. The potential routing state saved may be evaluated analytically, i.e. by considering the number of routing prefixes which must be advertised to support host multihoming; refer to Subsec. 4.5.2.

4.5.1. Scope. The aim here is to demonstrate that host multihoming may be implemented without introducing additional routing state, and to evaluate its relative protocol overhead. We would expect a modest increase in memory usage and packet processing overhead for these multi-homing functions, and that this would scale upwards with $N_P$ (the number of locators configured on the node). In this scenario, multihomed hosts perform load sharing of network traffic without using routing protocols or other mechanisms, e.g. NAT-based or link-layer approaches. The evaluation against IPv6 is comparative and not an absolute performance evaluation, as my aim is to show that ILNP can be engineered into an existing IPv6 networking stack (also, to validate the implementation). The experiment controls discussed in Subsec. 3.8.3 were in place throughout the evaluation, and all measurement tools were run in a single threaded mode.

4.5.2. Scalability analysis. If we assume that the hosts are topologically discrete (i.e. they do not share network prefix information with other hosts), then we would expect that the scalability analysis is the same as for sites as shown in Fig. 2.3.1, i.e. for $N_P$ host prefixes (locators) with $N_I$ upstream ISPs the required number of advertisements will be $O(N_P \cdot N_I)$. With ILNP, the state is displaced to the Domain Name System (DNS) with $O(1)$ scale for a single topologically discrete host during the initial DNS lookup of its L64 and NID resource records (simulated using the /etc/hosts file). My analysis does not include DNS lookups, as both ILNPv6 and IPv6 applications perform these before flow initiation.

4.5.3. Experiment design. I have constructed an experiment consisting of two dual-booted IPv6/ILNPv6 hosts: horn and reed, both of which are multihomed on
two site networks: $L_L$ and $L_M$. Both hosts participate in ICMPv6 ECHO and UDP unidirectional trials, with horn configured as the traffic source. In ILNP trials, both are configured to balance outgoing traffic by packet volume across each attached LAN segment.

Using the `fping` tool discussed in Subsec. B.2, I measured distribution of received packets at each locator (interface) configured on each multi-homed host. This was compared with the configured ILNP locator precedence to evaluate for correct load sharing behaviour.

4.5.4. Configuration. Whilst `ilnp-ra` was permitted to advertise a default route, neither horn or reed required an on-link router to reach each other, and IPv6 forwarding was disabled. The IPv6 RA parameters $MinRtrAdvInterval$ and $MaxRtrAdvInterval$ [123, Sec. 6.2.4] were set to 10 and 15 seconds respectively. Refer to Appendix B.4 on page 204 for details of RA/RS message timing specific to FreeBSD.

4.5.5. Metrics. The stream measurements listed in Section 3.8.1 on page 43 were performed over 25 trials, for each of the load rate and frame size combinations listed in Section 3.8.2 on page 45. Four configurations were studied: unmodified IPv6, and three combinations of ILNP precedence values assigned to each locator (100/0,
80/20 and 50/50). ICMPv6 measurements were performed at the source node horn. However, UDP measurements were taken at the destination node reed.
4.6. Results

The main finding is that the overall performance gap between ILNPv6 & IPv6 is very small. Most importantly, the use of ILNP did not introduce additional routing state; refer to Section 3.8.1.1 on page 45 for details of how this measurement was performed. So, my hypothesis – i.e. that host multihoming functions may be implemented without introducing routing state – is strongly supported by these results. In addition, the use of multiple locators to perform load sharing was validated by measuring the distribution of received packets for each locator in a separate set of trials. This feature functioned correctly throughout, so these results are not reported here.

4.6.1. Data presentation. All results are presented with a 95% confidence interval over 25 trials. Error bars in most plots may be too small to be easily seen. Moreover, to aid visual comparison between factors, X-axes have been normalised to the same basis for both packet sizes by load rate. As only a subset of rates have been exercised for each packet size, the gaps observed in such plots are expected; refer to Section 3.8.2.1 on page 46 for details of this and the underlying packets-per-second (PPS) measurement basis. In addition, the Y-axes for most plots in this chapter are scaled to a percentage of received packet volume. However, the absolute sample count (packets) increases in direct proportion to the load rate. So, the bar plots for reordering show an upward slope, as more packets are reordered in certain configurations.

4.6.2. Stream measurements. The gap in achieved goodput between ILNPv6 & IPv6 – for ICMPv6 & UDP packet streams – was very small. The trellis plots on pages 79-85 present stream measurements for IPv6 and three ILNP load sharing configurations. Round-trip time (RTT) results are presented for the ICMPv6 protocol only, as all UDP trials were one-way (unidirectional). Reordering is reported in Subsec. 4.6.3. Jitter in all trials was <0.3ms\textsuperscript{14} and has been omitted for brevity.

\textsuperscript{14}This is believed to be beyond measurement capability, as \texttt{iperf} is limited to 1µs precision by \texttt{gettimeofday()}: refer to Appendix B.3.3 on page 201 for system level timing details.
Neither ICMPv6 packet loss or UDP payload loss were significant, and both are summarised in Table 4.1.

ICMPv6 and UDP goodput results for both IPv6 and ILNPv6 are very similar, up to a threshold of \( \approx 50 \text{Kpps} \). Results for disparate frame sizes have been rebased to a common X-axis, as discussed in Section 3.8.2.1 on page 46; refer to Section 3.8.3.2 on page 48 for discussion of the gap between ICMPv6 and UDP results in the same experiment group, and for explanation of the drop-off at 1Gbit/s due to \textit{iperf} context switch overhead. The sequence error results in Fig. 4.6.6 on page 84 have large error bars (i.e. displaced packets wrongly classified as “lost”) at a subset of load rates in the 50/50 and 80/20 scenarios.

Beyond the \( \approx 50 \text{Kpps} \) threshold, I observe a general performance gap in goodput (<1% relative to IPv6), rising to \( \approx 4\% \) at the highest PPS rate of \( \approx 100 \text{Kpps} \) (i.e. for 100Mbit/s, 128 byte frames). Additionally, Fig. 4.6.3 on page 81 shows that there is a very small increase in median RTT up to the same threshold, beyond which there is an increase of 1.6ms (\( \approx 45\% \) relative to IPv6) at the highest PPS rate.

Finally, the UDP jitter results in Fig. 4.6.7 on page 85 also show that the inter-packet delay becomes less predictable as the system approaches its performance limits. The \( \approx 100 \text{Kpps} \) level represents a “worst case” in terms of packet processing for the system under study: i.e. the packet arrival rate has greatly increased, whereas the departure rate remains constant.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Relevant figure</th>
<th>Upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMPv6 packet loss (aggregate)</td>
<td>Fig. 4.6.1</td>
<td>( \approx 0.01% )</td>
</tr>
<tr>
<td>UDP payload loss, 1Gbit/s, 1514 byte frames</td>
<td>Fig. 4.6.2 on page 80</td>
<td>( \approx 1.50% )</td>
</tr>
<tr>
<td>UDP payload loss, all other trials</td>
<td>Fig. 4.6.2 on page 80</td>
<td>( \approx 0.20% )</td>
</tr>
</tbody>
</table>

Table 4.1. Host multihoming, summary loss statistics – listed by associated goodput & loss metric
Figure 4.6.1. ICMPv6 goodput (as percentage of load rate) between two IPv6 and ILNPv6 hosts
Figure 4.6.2. Unidirectional UDPv6 goodput (as % of load rate) between two ILNPv6 hosts using Locator Selection. The drop-off at 1Gbit/s is due to **iperf** context switch overhead; refer to Section 3.8.3.2 on page 48.
Figure 4.6.3. Median RTT for ICMPv6 streams between two IPv6 and ILNPv6 hosts
Figure 4.6.4. Mean RTT for ICMPv6 streams between two IPv6 and ILNPv6 hosts.
Figure 4.6.5. Packet reordering (as a percentage of received packets) for unidirectional UDPv6 flows, between two ILNPv6 hosts using Locator Selection.
Figure 4.6.6. Sequence errors (as a percentage of received packets) for unidirectional UDPv6 flows, between two ILNPv6 hosts using Locator Selection.
Figure 4.6.7. Jitter between packets for Unidirectional UDPv6, between two ILNPv6 hosts using Locator Selection
4. HOST MULTIHOMING WITH ILNP

4.6.3. Reordering distribution. As one might expect, reordering is not present in the IPv6 control group – as only a single IPv6 address (and, therefore, link) is used. However, for the ILNP experiment group, packet reordering is expected as two links are used, and is not the result of a configuration problem. Reordering is commonly observed with multipath protocols – including ILNP. However, if displacement is greater than 2 positions, this may pessimise TCP performance by triggering unnecessary re-transmissions. This is discussed further in Subsec. 7.3.

Fig. 4.6.5 shows aggregate UDP reordering results. In addition, Fig. 4.6.8 shows the distance from expected packet sequence for late UDP arrivals, at the destination host reed. A modified iperf binary collected these results to 30 positions. However, X-axis limits are set to show only relevant data points: e.g. for 128 byte UDP packets, arrivals were no later than 8 positions out of sequence.

In this scenario, there are only two possible paths for an egress packet, i.e. the links corresponding to the locators $L_L$ and $L_M$ shown in Fig. 4.5.1 on page 75. So, there are four queues which each packet may enter, i.e. two transmit and receive queues at horn and reed respectively. The number of reordered packets – for 1Gbit/s, 1514 byte frames – is 0% of the total received when 1 locator is active, approaches $\approx 27\%$ when 2 site locators are active with a 50/50 split, and approaches $\approx 66\%$ when the split is set to 80/20. The latter represents a “worst case” for packet reordering as one path is favoured much more than the other.

However, the effects of queueing delay – partly due to interrupt coalescing – are visible in Fig. 4.6.8 on the next page. The displacement distribution has a long tail for 1514 byte frames due to their longer dwell time. The service times for each of the four queues in the topology are independent of each other. In addition, the modulating effect of interrupt coalescing (IC) presents itself as periodic gaps in the distribution: where the queue cannot be fully serviced within an interrupt time window, IC has the effect of delaying packets into the next window; refer to Subsec. 7.3.
4.6. RESULTS

Figure 4.6.8. Relative frequency of sequence displacement (as percentage of received packets), between two ILNPv6 hosts using Locator Selection (summarised for all load rates – refer to Fig. A.1.1 on page 164 for detailed plots by each load rate).

4.6.4. Memory overhead. ILNPv6 uses a small additional amount of memory, and did not introduce any additional routing state; i.e. the \texttt{rtentry} pool counter remained constant over the lifetime of each trial. Table 4.2 shows allocations for ILNP control structures at both \texttt{horn} and \texttt{reed} for a single active flow (i.e. in the 50/50 load sharing scenario). The table notes refer to the relative change in these figures if only one locator is active. Table 4.2 shows that the additional non-routing state required at both hosts – i.e. in addition to the FIB, ND and RA structures which are part of IPv6 – amounts to a few hundred bytes. Figures 4.6.9 and 4.6.10 show expected results given the factors, controls and implementation in use. Load
rates are specified in a log-1-2-5 pattern, so the relationship between load rate and buffer overhead appears linear. Results for disparate frame sizes have been rebased to a common X-axis, as discussed in Section 3.8.2.1 on page 46.

The gap between IPv6 and ILNPv6 – observed in the central trellis column – is also expected, and may be attributed to protocol overhead. The drop in 2KB buffer usage for ILNPv6 at both rate ceilings (i.e. 100Mbit/s for 128 bytes, & 1000Mbit/s for 1514 bytes respectively) corresponds to the drop in goodput seen in Fig. 4.6.1 on page 79 and Fig. 4.6.2 on page 80. ILNP also makes use of additional mbuf tags (i.e. the rightmost trellis column) to cache the de-multiplexed Nonce Option during packet processing. Additionally, a modest number of 256 byte buffers (shown in the leftmost trellis column) are used for transmitting LU messages. All of the results presented in this subsection were obtained from the output of the `vmstat -m` command, as run before and after each trial. No memory leaks were observed during trials.

### Table 4.2. ILNPv6 memory overhead for hosts with two active locators.

<table>
<thead>
<tr>
<th>Type</th>
<th>Bytes</th>
<th>Count, static</th>
<th>Count, 1 flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>cla_ent&lt;sup&gt;a,b,c&lt;/sup&gt;</td>
<td>48</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>flow&lt;sup&gt;a&lt;/sup&gt;</td>
<td>240</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>loc&lt;sup&gt;a&lt;/sup&gt;</td>
<td>160</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>locpolentry</td>
<td>64</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>nid</td>
<td>40</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>pol&lt;sup&gt;d&lt;/sup&gt;</td>
<td>48</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Overhead, static</td>
<td>584</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic overhead, for 1 active flow</td>
<td>656</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<sup>a</sup> Instances are created when a flow is active.
<sup>b</sup> Lookaside cache entries are created only at horn.
<sup>c</sup> Decremented at horn where one locator is disabled.
<sup>d</sup> Decremented at reed where one locator is disabled.

The `mbuf` allocation contrast plots in Fig. 4.6.9 on the next page and Fig. 4.6.10 on page 90 show the dynamic network buffer usage for both IPv6 and ILNP at each node. These results are presented with a log-1-2-5 scale on the X axis (load rate) and a log-10 scale on the Y axis (absolute requests). The plots are intended as a sanity check on the implementation, and are not intended as an evaluation of absolute performance.
Figure 4.6.9. Cumulative mbuf buffer requests during an ICMPv6 trial – measured over 25 trials at source horn (for all load rates)
Figure 4.6.10. Cumulative mbuf buffer requests during an ICMPv6 trial – measured over 25 trials at destination reed (for all load rates)
4.7. Chapter summary

In this chapter, I have demonstrated an implementation of ILNP for FreeBSD hosts, and evaluated its relative performance in comparison with IPv6. The results in Sec. 4.6 show that the implementation has very low performance impact and memory overhead. Moreover, hosts may now perform multihoming functions (e.g. load sharing & failover) by using ILNP with locator selection & rewriting, rather than participating in the routing system (or requiring the use of a “middlebox”). Whilst the forwarding plane at each ILNP host now contains ILNP state (i.e. in the form of locator precedence, policy, and state associated with locators), no additional state has been introduced to the routing system. This is because ILNP is effectively acting as a policy applied to the IPv6 forwarding plane: the IPv6 network prefixes configured on each host are re-interpreted as ILNPv6 locators, and their selection may be controlled by a site-defined policy.

The small gap in performance between IPv6 and ILNPv6 (discussed in Section 4.6.2 on page 77) may be attributed to protocol overhead, which is largely expected. In addition, these results were obtained using an unoptimised implementation of ILNP; i.e. it has not been optimised beyond basic architectural assumptions, such as those underlying the FIB cache; refer to Section 4.3.5 on page 65. Finally, ILNPv6 re-uses the IPv6 address bits in the existing IPv6 header format (as discussed in Sec. 4.2). So, ILNPv6 packets are able to traverse IPv6 network elements transparently. Moreover, ILNPv6 may be implemented in terms of existing IPv6 code & data structures: e.g. the FIB, ND and RA tables. In addition, a subset of IPv6 applications may use ILNP without changes; refer to Sec. 4.4.
Chapter 5
Site-controlled multihoming

5.1. Introduction
In this chapter, I discuss how site-controlled multihoming may be implemented with ILNP – without adding routing state. The mechanisms used to apply site policy to forwarding are almost identical to those used for host multihoming. However, there are two key architectural changes required: (1) the use of locator updates with locator rewriting, and (2) the introduction of a ILNP-specific forwarding plane. Much of the existing IPv6 network layer can be re-used to provide this functionality.

5.1.1. Contribution. The main contributions of this chapter are: (A) to demonstrate how the host-based implementation of ILNP – i.e. realised as a set of extensions to IPv6 described in Chapter 4 – may be extended to provide site multihoming functions; (B) to show how these functions may be conveniently managed on a site-wide basis, through the use of sbr-ILNP policies; (C) to show that ILNP provides multihoming capabilities similar to those achieved through “tweaking the route exchange” in IP today; and (D) to show that it has very low overhead when compared with NPTv6/IPv6.

5.1.2. Chapter structure. This chapter is structured as follows:

(I) Subsec. 5.1.3 begins by describing how site-controlled network functions are typically implemented within IP networks today.
(II) Sec. 5.2 continues by expounding[^1] on how ILNP operates at singleton Site Border Routers (SBRs), and how site locator policies are applied, also.

(III) Sec. 5.3 contrasts the relative performance of the sbr-ILNPv6 with its statically-bound analogue, NPTv6.

Results are presented and discussed in Section 5.4 on page 104, and the chapter concludes with a summary in Sec. 5.5.

### 5.1.3. Problem space.

Whilst site multihoming is supported in IP today, it is normally implemented as a function of the routing system. So, network functions – e.g. load-balancing, fail-over – are often implemented using a dynamic routing protocol (e.g. BGP, OSPF, IS-IS). And whilst this approach is widely used, it adds bloat to routing tables – in the form of de-aggregated prefixes. The high-level diagram in Fig. 5.1.1 illustrates two site networks ($L_L, L_M$) which are interconnected by two ISP networks ($L_1, L_2$). To implement site-controlled path choice with IP today, both SBR1 and SBR2 must exchange routing prefix information upstream with ISP1 and ISP2 respectively.

![Figure 5.1.1. Scenario demonstrating site-controlled multi-homing.](image)

Each site network has a single site border router (SBR) and contains a single host (H1, H2). With ILNP, each upstream router (SBR1, SBR2) may be configured to rewrite site locators to globally visible locators ($L_1, L_2$), and to volume balance outgoing traffic on their attached ISP networks (ISP1, ISP2).

### 5.1.4. Approach.

The claim is that ILNP can also provide site-controlled multihoming, and without introducing routing state beyond that which is required for topological routing. As ILNP locators have precedence (discussed in Section 4.2.3.1 on

[^1]: This section also emphasises fast packet processing, as this is an important non-functional requirement for network routing.
page 58), site locator policies can – alternatively – be used to implement multihoming-related network functions. So, I have extended the implementation from Chapter 4 to provide this functionality: sbr-ILNP – as described in [9, Sec. 3]. The implementation is dual-stack (refer to [6, 7]) as both the IPv6 and ILNPv6 forwarding planes are able to co-exist; in addition, the IPv6 FIB is re-used to efficiently resolve remote locators to next-hops.

5.2. Method

This section discusses the implementation of sbr-ILNP, which is based upon the implementation studied previously in Chapter 4.

5.2.1. Marking ILNPv6 with IPv6 Flow Labels. As discussed in Section 4.2 on page 56, IPv6 and ILNPv6 packets are mostly identical. Whilst the Nonce Option [103] is end-to-end only (and is normally inspected only by the destination node), an ILNP router implementing site-controlled multihoming must also inspect it. So, with no other indication that an IPv6 packet is (in fact) an ILNPv6 packet, an SBR must look the flow up in the ILCC, and/or enumerate the IPv6 header chain for a Nonce Option (if present). Both operations are potentially expensive in terms of both CPU cycles and memory latency; efficient option processing is a desirable performance goal for routers, and many implementations delegate option processing to a “slow” path [124].

So, I have used the most significant bit (MSB) of the IPv6 Flow Label [125] to indicate that a packet is ILNPv6 (and not IPv6); routers are disallowed from modifying its contents [108, Sec. 6]. This choice was made to reduce protocol overhead: it constitutes an example of “trading packet headers for packet processing” [126]. More specifically, such an optimisation may be necessary to achieve acceptable performance in scenarios where ILNP provides multihoming for packet flows encrypted using the

\[\text{Enumeration is problematic for routers, as the Nonce Option is a destination option. Although a hop-by-hop option type – which must be inspected by every on-path router – is specified for IPv6, [108, Sec. 4.3] no option type with finer granularity has been specified (e.g. all on-path ILNPv6 routers).} \]

\[\text{Both operations are required for Locator Update snooping – discussed in Subsec. 5.2.2 and Subsec. 5.2.5.} \]
IPSEC ESP protocol. This is because transport layer headers are not visible to the networking stack – or indeed to an SBR – when encrypted traffic is being demultiplexed. This important observation is one contribution which I have made to the ILNP RFC documents described in Section 3.6 on page 39.

### 5.2.2. ILNPv6 Forwarding

An ILNPv6 SBR performs locator selection (and optionally, locator rewriting). I have modified the IPv6 forwarding path to conditionally invoke ILNPv6 forwarding, when an IPv6 packet with the ILNPv6 Flow Label bit is received. This is shown in Fig. 5.2.1 as a call to the C function `ilnp6_forward()`. If an ILNPv6 packet cannot be forwarded (for any reason), the SBR will either fall back to IPv6 forwarding, or drop the packet. This behaviour is configurable on a per-node basis.

![Simplified call graph for ILNPv6 forwarding](image)

**Figure 5.2.1.** Simplified call graph for ILNPv6 forwarding. The processing of “snooped” Locator Updates is omitted here for clarity; refer to Fig. 5.2.2 on page 99. The reverse-type (black) box shows that the forwarder invokes the common locator selection procedure (i.e. using the same code used by hosts). The shaded (light blue) boxes are ILNPv6 functions. The plain (white) boxes are IPv6 data-plane functions. ILNPv6 forwarding does not use IPv6 source selection; however, it re-uses the IPv6 FIB.

Flow state is established in the same way as for hosts: i.e. a Nonce Option must be included in the first packet(s) of a flow, [103, Sec. 6] as discussed in Section 4.3.1 on page 61. The Nonce value is cached in a FreeBSD `mbuf` tag\(^4\) as an engineering optimisation, i.e. to simplify concurrency control. However, both flow classification

---

\(^4\)Tags containing the Nonce Option are created by the C function `ilnp6_non_input()`.
and *locator rewriting* require that all IPv6 headers are enumerated; [108] this is shown in Fig. 5.2.1 as a call to the function `ilnp6_pkt_exthdrs()`.

Next, a forwarding decision is reached by looking up the flow in the ILCC – discussed in Subsec. 5.2.3 – and applying Locator Selection and/or Locator Rewriting, as discussed in Subsec. 5.2.4. If the packet to be forwarded is a Locator Update, some additional processing is needed as described in Subsec. 5.2.5. Next-hop lookups are performed using the FIB cache described in Section 4.3.5 on page 65; stale entries may be refreshed by an indirect call to the `rtalloc()` function [118](also shown at the bottom of Fig. 5.2.1 in a white box).

Finally, packets are transmitted by invoking the IPv6 function `nd6_output()`,\(^5\) which resolves the next-hop to a link layer address using IPv6 Neighbour Discovery (ND). If any packet header fields were modified (e.g. the source and/or destination `I-LV` fields), transport layer checksums will be updated using a checksum-neutral mapping before transmission [65].

**5.2.3. ILCC flow state at SBRs.** On receiving an ILNPv6 packet, an SBR will perform a lookup in the ILCC for an existing flow; refer to Subsec. 3.6.1.3 for a description of the ILCC. If no match is found (and the packet includes the Nonce Option), a new flow will be created. Each flow inherits properties (e.g. locator precedence) from applicable policies, discussed in Subsec. 5.2.4.1.

5.2.3.1. *Implementation.* The ILCC has been implemented as a tail queue of flows. Each flow in the ILCC contains both *initiator* and *target* locator sets, also implemented as tail queues. This division exists to support *locator rewriting*, which must preserve the original *locator set* where a flow transits administrative network boundaries. Both sets are independent of a node’s local locator state (discussed in Section 4.2.3 on page 58), i.e. the *initiator* set is used only by SBRs (and not end hosts). The tail queue was chosen for simplicity and expedience, as much of the FreeBSD IP stack uses this data structure. A production implementation might use a *hash table* or *radix trie* as an engineering optimisation.

---

\(^5\)The `ip6_output()` function cannot be used; it will attempt to perform IPv6 source selection for the flow, overriding the next-hop choice made by Locator Selection.
5. SITE-CONTROLLED MULTIHOMING

5.2.4. Locator Rewriting. Locator rewriting is not NAT, as it operates on I-LVs, and not IP addresses; it represents a policy applied to the forwarding plane itself, in terms of the “ID/Loc” split. Locator Selection – discussed in Section 4.3.3 on page 62 – may be extended to provide Locator Rewriting, which I have implemented by extending the policy table to contain additional information about site locators (as described in Section 4.2.3.1 on page 58), and by adding logic to the forwarding plane (as described in Subsec. 5.2.4.1).

5.2.4.1. Policies. When a packet is to be forwarded, locator selection must examine both the ILCC and the policy table to reach a rewriting (or selection) decision. This mechanism enables site-controlled multihoming, i.e. the SBR itself advertises the precedence of each site-local locator. The policy table entry for each locator contains an (optional) target label, which specifies a target group of locator(s) available for rewriting; each locator may also belong to up to eight (8) target rewrite groups; refer to Appendix B.5.1 on page 204 for configuration syntax.

If a locator matches a policy entry, it will be rewritten to a locator in its target group specified by its target label. The locator choice is made by an algorithm specified in the policy entry, and is independent for each packet in the flow; e.g. for the evaluation scenario in Sec. 5.3, locators are chosen by a simple volume splitting algorithm, which must also count the number of packets transmitted using each locator in the target group. Algorithms also re-use the FIB cache – discussed in Section 4.3.5 on page 65 – to exclude unreachable locators from selection.

5.2.4.2. Reversibility. My implementation does not impose any restriction on the direction of matches – i.e. policy entries may be applied to both source and destination locator(s), and for both ingress and egress traffic. So, the cardinality of group matches may be reversed, as flows may be initiated by an external host – in which case return traffic from an internal host in the same flow will also be rewritten, i.e. by using the same mapping in reverse. So, to perform bi-directional rewriting, SBRs must first compare both source and destination locators with the SBR’s local locator set, to determine whether the ILCC initiator or target locator set for the flow should be used for further comparison.
5.2.4.3. *Implementation*. Policy targets were implemented using *bit sets* in the policy table and ILCC. The use of bit sets offers good relative performance as compared to e.g. red-black trees, although bit sets are statically sized at compile time. Their use was also convenient, as the FreeBSD kernel source code provides structured C macros for their manipulation.

5.2.5. **Locator Update snooping.** The use of *locator rewriting* requires that SBRs rewrite Locator Updates (LUs), also. Hosts may advertise internal locators in LUs without knowledge of site locators [9, Sec. 3]. So, SBRs must “snoop” on LUs within the forwarding plane and potentially rewrite their contents; refer to [8, Sec. 2] for the syntax of LU messages. The control flow is shown in Fig. 5.2.2.

![Call graph for ILNPv6 Locator Update “snooping”.](image)

**Figure 5.2.2.** Call graph for ILNPv6 Locator Update “snooping”. The shaded (light blue) boxes are ILNPv6 functions. The plain (white) boxes are IPv6 data-plane functions. SBRs may rewrite Locator Updates originating from internal hosts to apply site-wide locator policies.

When an SBR receives an LU message destined for another node, its contents are compared with the policy table. If the SBR then determines that the LU must be rewritten, it will instead transmit a *rewritten* LU and drop the original LU message, whilst preserving the identifier of the origin in the IPv6 Source Address field.

The rewritten LU will contain the site locators defined in the SBR’s policy table. The SBR must also cache the original Nonce value transmitted by the end-host in the
ILCC. The Nonce is transported in an IPv6 destination option; however, the SBR – an on-path IPv6 router – must inspect it.

5.2.5.1. Implementation. My implementation of locator update snooping has the following restrictions: whilst generating a new LU message – shown in Fig. 5.2.2 by a call to `ilnp6_flow_sched_lu()` – is simple to implement, it adds a small delay to LU messages as they transit SBRs; this is because a new packet buffer must be allocated, and its transmission is deferred to avoid transmitting duplicates. However, if a host advertises a locator unknown to the SBR, it will be omitted from the rewritten LU. In addition, locator precedence advertised by the host will be overwritten by the SBR. This choice was made to ensure correct behaviour in my evaluation in Sec. 5.3, where the aim is to demonstrate site – not host – multihoming with ILNP.

5.2.6. Re-use of IPv6 & Host-based ILNPv6 code. In addition to the FIB cache described in Section 4.3.5 on page 65, the sbr-ILNP code re-uses code and data structures – from both IPv6, and the ILNPv6 end-host implementation described in Chapter 4. The ILNP end-host code monitors several salient IPv6 data structures for updates, as discussed in Section 4.2.3 on page 58; these include the Prefix List, the Address List, and the FIB. An SBR must monitor these for changes, also; if the state of a locator (link) associated with active flow(s) in the ILCC changes, it must transmit LUs containing updated locator state. Additionally, SBRs use the same mechanism as hosts for locator discovery – as discussed in Section 4.2.3 on page 58. However, they do so by listening to their own RAs, whereby IPv6 site prefixes will appear as site locators to sbr-ILNP.

5.2.7. Concurrency control. In my implementation of sbr-ILNP, most operations are serialised by a single mutex lock. This choice was made for simplicity; the aim is to show that it is possible to modify and extend the existing IPv6 source code to provide ILNP functions, rather than to provide an optimised implementation.

5.2.8. Summary. In this section, I have described how the host-based ILNPv6 implementation has been extended to provide site-controlled multihoming at SBRs. Although the issues faced by a host implementation – e.g. in the socket layer –

---

6Shown in Fig. 5.2.2 as a call to the function `ilnp6_non_find()`. 
do not normally arise within a router-based implementation, fast packet processing is an important non-functional requirement for routers – one which mandates implementation strategies such as those described in Subsec. 5.2.1. Moreover, ILNPv6 forwarding may co-exist with IPv6, and is conditionally invoked from the existing IPv6 forwarding plane; existing IPv6 structures (e.g. FIB, ND & RA) are efficiently re-used. Finally, to perform locator rewriting ILNPv6 routers must also snoop and rewrite Locator Updates – as discussed in Subsec. 5.2.5. Whilst the implementation has some limitations, it is sufficient to perform the evaluation in Sec. 5.3.

5.3. Evaluation

In this section I aim to show that site-controlled multi-homing (with a single SBR) may be implemented without adding routing state.

5.3.1. Scope. The evaluation is focused on the performance of site multi-homing at SBRs, and not end-hosts, which have been discussed in Chapter 4. The relative (not absolute) performance was compared with NPTv6 [40] – an existing scheme for static multi-homing – to validate the implementation. During ILNPv6 trials, both end hosts dove and kite (discussed in Subsec. 5.3.3) used a configuration similar to that of horn and reed (described in Section 4.5 on page 74).

So, in this evaluation, the overhead of ILNP at the end-host is unchanged from the 100/0 configuration discussed in Section 4.6 on page 77; i.e. end-hosts have only a single default route. The same experiment controls discussed in Subsec. 3.8.3 were in place throughout the evaluation, and all measurement tools were run in a single threaded mode. Moreover, all evaluation traffic is unicast.

5.3.2. Scalability analysis. The scalability analysis of saved routing state is almost identical to that for host multi-homing (as described in Sec. 4.5), i.e. for \( N_P \) site prefixes (locators) with \( N_I \) upstream ISPs, we would expect that the required number of advertisements will be \( O(N_P \cdot N_I) \). With ILNP, the state will be displaced
5.3.3. Experiment design. I constructed a test topology of two sites, each comprised of a single SBR and host. The east and west sites were interconnected by a single IPv6 router, which ran an unmodified FreeBSD 8.4 kernel. IPv6 measurements were performed in the same topology, i.e. all nodes ran an unmodified FreeBSD 8.4 kernel during an IPv6 (NPTv6) trial. All nodes were capable of dual-booting between unmodified IPv6 and ILNPv6 code bases.

5.3.4. Configuration. During ILNP trials, both SBRs `emu` and `jay` were configured to allow ILNPv6 forwarding with locator rewriting. This scenario is similar to the NPTv6 multiple translator [40, Sec. 2.4] with two key differences:

1. A single ILNPv6 SBR is connected to multiple external networks;
2. Load sharing with NPTv6 requires additional SBRs [40, Sec. 2.3].

\[^7\]In my evaluation the DNS exchange is simulated by extending the `/etc/hosts` file syntax as in Section 4.2.4 on page 58.
Both sbr-ILNP and NPTv6 rewrite 64-bit IPv6 prefixes (locators). The presence of the IPv6 router *grus* allowed IPv6 NDP to operate correctly: the Ethernet links $L_C—L_F$ in Fig. 5.3.1 were used as point-to-point interfaces, and were configured with different prefix lengths (/64 vs /126) at either end of the link.

Additionally, ILNPv6 packets leaving the forwarding plane are transmitted using the *nd6_output()* function, as discussed in Subsec. 5.2.2. However, the network stack records these packets in read-only FreeBSD link layer counters (and not IPv6 counters – as for packets originated by the SBR itself, and for all transit packets in NPTv6 trials). So, reboots were required between each trial to reset both sets of counters.

5.3.4.1. **NPTv6 implementation.** NPTv6 specifies a stateless mapping between IPv6 network prefixes. A native implementation was unavailable – so, the *pf* stateful firewall was used in its place. The *binat* rule stanza shown in Fig. 5.3.2 provides a bi-directional translation between IPv6 network prefixes. The *pf* configuration [127] used (refer to Appendix B.5.2 on page 206) enables only those features relevant to providing NPTv6. However, its stateful firewall cannot be entirely disabled; i.e. state entries are created for all packet flows, as the NAT implementation in *pf* requires them by design [128, Sec. 2.3]. *pf* also requires exclusions for NDP traffic; unlike IPv4 ARP packets which use a discrete Ethernet encapsulation, IPv6 NDP is encapsulated in ICMPv6 – and so may be affected by firewall rules.

5.3.4.2. **Static vs. dynamic site multihoming.** Unlike ILNP, *pf* does not support Locator Rewriting; each configured *binat* rule must be bound to a single link. State entries contain IP addresses, and are therefore *statically bound* – as shown in Table 2.1 on page 11. *pf* is otherwise unaware of the difference between I-LVs – which have *dynamic binding* – and IPv6 addresses, which do not. Whilst it can perform dynamic site multihoming, this requires special NAT configuration and is not a general component of the IP stack. By contrast, ILNP provides site multihoming as part of its

---

5.3. EV ALUATION 103

**Figure 5.3.2.** Example *pf* configuration for NPTv6 translation: an internal prefix is translated to a site prefix.

```
binat on net3 inet6 from fec0:bad3::/64 to any -> 2001:db8:cafe:c6::/64
```
architecture. Finally, whilst state may be migrated – to a limited\(^9\) degree – between pf SBRs by using extensions introduced for this purpose (e.g. pfsync [129]), pf does \textit{not} support\(^{10}\) the migration of states between links.

5.3.5. \textbf{Summary.} In this section, I have framed the evaluation scenario – i.e. a comparative study between ILNPv6 and NPTv6, and described its implementation requirements. It is expected that ILNPv6 may have additional protocol overhead as compared to NPTv6, due to the additional processing overhead of using multiple locators. The use of pf clearly demonstrates the “impedance mismatch” between IPv6 network elements and the “ID/Loc” split – in the form of ILNP. Although pf is able to support some of the use cases that may be realised with ILNP, in this scenario it is only used to provide NPTv6 functionality.

5.4. Results

The main finding here is that sbr-ILNPv6 has very low performance impact and memory overhead, as compared to NPTv6 (pf). In order to enable site multi-homing functions, one would expect a modest increase in memory usage and packet processing overhead; and that these quantities would scale upwards with the number of site locators (i.e. \(N_P\) as described in Subsec. 5.3.2). To aid visual comparison, the Y-axes for most plots in this section are scaled to a percentage of received packet volume. So, there is a downward slope present as load increases: i.e. the number of samples (packets) increases linearly with load rate, and does not indicate any unidentified tendency in the data. All results were measured over 25 trials for each combination of factor levels described in Section 3.8.2 on page 45.

5.4.1. \textbf{Stream measurements.} The performance characteristics of sbr-ILNPv6 and NPTv6 (pf) are very similar. The trellis plots on pages 107-114 present stream measurements from each scenario as discussed in Section 3.8.1 on page 43; results

\(^9\)Rules may be migrated between pf nodes using pfsync \textit{interface groups}; state entries cannot be so migrated.

\(^{10}\)A singular exception involves the use of “floating” states, which may match packets arriving on multiple configured links.
for disparate frame sizes have been rebased to a common X-axis, as discussed in Section 3.8.2.1 on page 46.

Fig. 5.4.1 and Fig. 5.4.2 show that for ICMPv6 and UDPv6 respectively, the goodput achieved for both NPTv6 and ILNPv6\textsuperscript{11} is very similar – up to \(\approx 50\) Kpps load rate (i.e. for 50Mbit/s, 128 byte frames; refer to Table 3.2 on page 46). However, in Fig. 5.4.1, it can also be seen that \texttt{bweeping} did not generate the requested ICMPv6 load at 10Mbit/s. Recalling Section 3.8.3.3 on page 49, the underflow in transmitted load rate is \(\approx 9\%\) at 10Mbit/s. Refer to Appendix B.2.3 on page 198 for a detailed explanation of this measurement artefact. Beyond \(\approx 50\) Kpps, I observe a drop in goodput for 128 byte frames: 4\% for ICMPv6, and 30\% for UDPv6. This drop is believed to be due to the additional per-packet processing time required by ILNPv6 load sharing. Refer to Section 3.8.3.2 on page 48 for discussion of the gap between ICMPv6 and UDP results in the same experiment group.

ICMPv6 loss was < 0.01\% in all cases, so the corresponding plots have been omitted for brevity; UDPv6 payload loss is shown in Fig. 5.4.7 on page 113. Reordering results are reported in Subsec. 5.4.2. Jitter above 50ms was observed for NPTv6 in some trials, which were discarded. The cause is believed to be specific to \texttt{pf} and its stateful design: its periodic state expiry mechanism requires exclusive access to the \texttt{pf} state table (i.e. the forwarding plane code cannot be executed concurrently when this timer is being serviced), adding uncertainty to packet processing time.

Additionally, Fig. 5.4.3 on page 109 shows that median RTT for both NPTv6 and ILNPv6 is very similar up to the same \(\approx 50\) Kpps threshold. At the highest packet rate of 100Kpps, the RTT of the ILNPv6 flow is \(\approx 0.5\) ms greater (i.e. \(\approx 31\%\) relative to NPTv6). Finally, the UDP jitter results in Fig. 5.4.8 on page 114 show that the inter-packet delay becomes less predictable above \(\approx 50\) Kpps. So, whilst there appear to be differences in packet processing time between NPTv6 (\texttt{pf}) and ILNPv6 (i.e. \texttt{pf} achieves lower mean RTT under load), both achieve similar goodput under load.

\subsection*{5.4.1.1. Loss} The loss observed in trials of \texttt{sbr-ILNPv6} was very low. However, the majority of loss and late packet arrivals observed for NPTv6 (\texttt{pf}) may be attributed to the additional packet processing time required by a stateful firewall. Whilst

\textsuperscript{11}There was very little variation in achieved goodput between ILNP load sharing configurations.
ILNP must inspect IPv6 headers also, there is less code complexity. However, *pf* must inspect transport-layer headers, and all extended IPv6 headers (including options).

In Fig. 5.4.7 I observe loss of up to 2.8% across all NPTv6 trials (i.e. in the leftmost trellis column). The *iperf* error results shown in Fig. 5.4.6 follow largely the same pattern, with a spike at 1Mbit/s due to the small number of samples at that load rate (i.e. the plot uses a percentage scaled Y-axis); refer to Section 3.8.3.1 on page 48 and Subsec. 5.4.2 for interpretation of these results. By contrast, loss is <0.3% for ILNP (i.e. all other trellis columns), except at the highest goodput rate of 1Gbit/s and for PPS rates >50Kpps. The downward trend by load rate is an artefact of the percentage scaled Y-axis.

Throughout the experiment, network adapter queues were configured to allow *elastic* growth, i.e. up to the maximum length supported by the hardware as discussed in Subsec. 3.8.3 and Appendix B.3.2 on page 200. It can be seen from Fig. 5.4.7 that the ILNP forwarding plane has lower loss under load, even when receive and transmit queues are allowed to grow *elastically.*
FIGURE 5.4.1. ICMPv6 goodput (as percentage of load rate) between two ILNPv6 sites using Locator Rewriting
Figure 5.4.2. Unidirectional UDPv6 goodput (as percentage of load rate) between two ILNPv6 sites using Locator Rewriting.
Figure 5.4.3. Median RTT for a single ICMPv6 ECHO flow between two ILNPv6 sites using Locator Rewriting
Figure 5.4.4. Mean RTT for a single ICMPv6 ECHO flow between two ILNPv6 sites using Locator Rewriting.

(A) 1514 byte Ethernet frames

(B) 128 byte Ethernet frames

IPv6 (NPTv6) | ILNPv6 100/0 | ILNPv6 80/20 | ILNPv6 50/50
---|---|---|---

(a) 1514 byte Ethernet frames

(b) 128 byte Ethernet frames

5. SITE-CONTROLLED MULTIHOMING
Figure 5.4.5. Packet reordering (as a percentage of received packets) for unidirectional UDPv6 flows, between two ILNPv6 sites using Locator Rewriting.
Figure 5.4.6. Sequence errors (as a percentage of received packets) for unidirectional UDPv6 flows, between two ILNPv6 sites using Locator Rewriting.
Figure 5.4.7. Unidirectional UDPv6 payload loss (as percentage of transmitted volume) between two ILNPv6 sites using Locator Rewriting.
Figure 5.4.8. Jitter between packets for Unidirectional UDPv6, between two ILNPv6 sites using Locator Rewriting
5.4.2. Reordering distribution. Packet reordering was observed in a limited group of trials, and is expected within a multihomed network topology. Whilst not indicative of pathological behaviour, it may impact TCP behaviour; refer to Subsec. 7.3. As expected, reordering is not present in the IPv6 control group or when only one ILNP locator is used at the SBR. Fig. 5.4.5 on page 111 shows aggregate packet reordering measurements taken at the destination host kite. The number of reordered packets approaches $\approx 27\%$ (for 1GBit/s, 1514 byte frames) when 2 site locators are active with a 50/50 split, and approaches $\approx 66\%$ (for 200MBit/s, 1514 byte frames) when the split is set to 80/20. The Y-axis is scaled to a percentage of received packet volume, so there is an apparent upward slope to each bar.

Fig. 5.4.9 expands on these results, by showing the mean distance from expected packet sequence,\(^{12}\) for late UDP packets across all load rates. The results for each load rate individually may be found in Fig. A.1.2 on page 166. X-axis limits are set to show only relevant data points: e.g. for 128 byte UDP packets, arrivals were no later than 8 positions out of sequence, whereas 1514 byte UDP packets were observed to arrive up to 30 positions late – which may not be visible here due to scale.

Fig. 5.4.9 on the following page shows gaps in the displacement distribution at a small set of intervals. These may be explained as a signature effect of Interrupt Coalescing (IC), which is also discussed further in Subsec. 7.3. However, the 80/20 load sharing configuration represents a “worst case” scenario for packet reordering, where one set of paths is favoured much more than the other; the long distribution tail seen in Fig. 5.4.9b (80/20 split, 1514 byte frames) can be attributed to both increased dwell time (due to the larger frame size), and to the cumulative effects of end-to-end queuing.

This may be explained by referring back to the topology: Fig. 5.3.1 on page 102 shows that there are 3 intermediate hops on the path for each outgoing packet. So, as each node has a single receive and transmit queue, a packet originating at dove must cross four receive and four transmit queues to reach kite. Moreover, the service times for each of the total eight (8) queues vary independently, so the distribution

\(^{12}\)iperf was modified to collect these results to 30 positions.
Figure 5.4.9. Relative frequency of sequence displacement (as percentage of received packets), between two ILNPv6 sites using Locator Rewriting (summarised for all load rates – refer to Fig. A.1.2 on page 166 for detailed plots by each load rate).

seen in Fig. 5.4.9b shows the cumulative effects of the 80/20 path choice decision, as discussed in Section 3.8.2 on page 45 and Section 3.8.3 on page 47.

5.4.3. Memory overhead. The sbr-ILNP implementation uses very little memory. As discussed in Section 3.8.1.1 on page 45, the rtentry pool counter remained constant over the lifetime of each trial. This result provides strong support for my hypothesis, i.e. that site-controlled multihoming may be implemented without introducing routing state. Table 5.1 on page 118 contains a detailed overview of runtime memory usage by the sbr-ILNP code, for a single active flow in the 50/50 load sharing
5.4. RESULTS

scenario. For comparison, Table 5.2 shows the structures allocated by pf to provide \textit{statically multihomed} NPTv6 prefix translation in the same topology.

Table 5.1 shows that the additional non-routing state required at each SBR – i.e. in addition to the FIB, ND and RA structures which are part of IPv6 – is <1KB in total. By contrast, Table 5.2 shows that pf requires \approx9KB\textsuperscript{13} to store rule sets, state entries, and state associated with IPv6 addresses. However, ILNPv6 uses 80\% more memory for each active flow: i.e. 720B for ILNPv6, and 400B for pf. This difference is due to protocol overhead, as ILNP must track locators used by the flow for Locator Rewriting. No memory leaks were observed during the experiment.

Additionally, Fig. 5.4.10 on page 119 shows that whilst pf and ILNPv6 use identical amounts of buffers for transit packets as they arrive (shown in the central trellis column), there are two main differences in their packet buffer usage: (1) the leftmost trellis column of Fig. 5.4.10 shows that pf uses additional 256B buffers. This result is due to how pf “pulls up” ICMPv6 headers to lie in contiguous memory, e.g. when examining the state table for a match with a received packet.

(2) The rightmost trellis column of Fig. 5.4.10 shows that pf uses \approx4x as many tags compared to ILNP. It uses a single tag to cache the demultiplexed \textit{Nonce Option} during packet processing, whereas pf is known to mark packets with mbuf tags in at least four different locations within the forwarding pipeline. Finally, ILNP uses a very small number of clusters (omitted from Fig. 5.4.10 for brevity) to buffer rewritten LU messages, whereas pf will treat them as ordinary IPv6 packets (and does not perform Locator Rewriting).

The mbuf allocation contrast plots in Fig. 5.4.10 on page 119 show the dynamic network buffer usage at SBR \texttt{emu}, for both pf and ILNP during ICMPv6 trials. These results are derived from high-watermark allocation statistics. The plots are intended as a sanity check on the implementation, and are not intended as an evaluation of absolute performance. These are presented with a logarithmic 1-2-5 scale on the X axis (load rate) and a log-10 scale on the Y axis (absolute requests). Load rates

\textsuperscript{13}This figure excludes \approx94KB for OS fingerprints. pf loads these by default even if this feature is disabled in configuration.


<table>
<thead>
<tr>
<th>Type</th>
<th>Bytes</th>
<th>Count, static</th>
<th>Count, 1 flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow&lt;sup&gt;a&lt;/sup&gt;</td>
<td>240</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>loc&lt;sup&gt;a&lt;/sup&gt;</td>
<td>160</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>locpolentry</td>
<td>64</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>nid</td>
<td>40</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>pol</td>
<td>48</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Overhead, static 856
Dynamic overhead, for 1 active flow 720

<sup>a</sup>Instances are created when a flow is active.

**Table 5.1.** ILNPv6 memory overhead for a single site SBR with two active site locators and a single internal LAN. Each type is part of the ILCC, excepting nid. Sizes are for the x86-64 machine architecture. C structure names omit the `ilnp6_` prefix for brevity.

<table>
<thead>
<tr>
<th>Type</th>
<th>Bytes</th>
<th>Count, static</th>
<th>Count, 1 flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>pf_os_fingerprint&lt;sup&gt;a&lt;/sup&gt;</td>
<td>40</td>
<td>420</td>
<td>420</td>
</tr>
<tr>
<td>pf_osfp_entry&lt;sup&gt;a&lt;/sup&gt;</td>
<td>112</td>
<td>710</td>
<td>710</td>
</tr>
<tr>
<td>pf_pooladdr</td>
<td>88</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>pf_rule</td>
<td>912</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>pf_state&lt;sup&gt;b&lt;/sup&gt;</td>
<td>400</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Overhead, static 105616
Overhead, for OS fingerprints<sup>a</sup> 96320
Dynamic overhead, for 1 active flow 400

<sup>a</sup>OS fingerprints are loaded by default.
<sup>b</sup>Instances are created when a flow is active.

**Table 5.2.** NPTv6-in-pf memory overhead for a singleton SBR with two statically multihomed IPv6 site prefixes and a single internal LAN. Sizes are for the x86-64 machine architecture.

are specified in a log-1-2-5 pattern, so the relationship between load rate and buffer overhead appears linear.
Figure 5.4.10. Cumulative mbuf buffer requests during an ICMPv6 trial – measured over 25 trials at first hop SBR emu (for all load rates). NPTv6 is contrasted with ILNPv6 configured for 50/50 load splitting. Results for disparate frame sizes have been rebased to a common X-axis; refer to Section 3.8.2.1 on page 46.
5.5. Chapter summary

This chapter has described how site-controlled multihoming may be implemented using sbr-ILNP, i.e. as a set of extensions to host-based ILNP described in Section 4.2 on page 56. The results here show that it is possible to multi-home sites without introducing routing state, i.e. by using Locator Rewriting in sbr-ILNP. Moreover, ILNPv6 achieves very similar (relative) performance; the gap observed is small and expected, and may be attributed to protocol overhead. It also has low memory overhead, requiring significantly less memory than pf needs to perform NPTv6 functions.

The implementation has not been optimised beyond what is necessary to implement it, e.g. using the FIB cache (discussed in Section 4.3.5 on page 65) to perform next-hop and/or link-layer lookups. In addition, the unoptimized ILNP forwarding plane exhibits lower loss than pf under load – even when receive and transmit queues are allowed to grow elastically, as discussed in Section 3.8.3 on page 47. Whilst a router-based implementation of ILNP need not address socket-layer issues, it must keep packet processing time to a minimum. Existing IPv6 code & data structures (e.g. the FIB, ND and RA tables) were re-used wherever possible, and the implementation is dual-stack; i.e. it co-exists with IPv6 in the same code base at runtime. Future work might compare ILNP with a stateless NPTv6 implementation rather than a stateful firewall/NAT code base, i.e. to investigate relative performance with stateless packet translation architectures in widespread use today, e.g. Carrier Grade NAT (CGN) as mentioned in Section 2.3.5 on page 17.
Chapter 6

Distributed site multihoming

6.1. Introduction

In this chapter, I describe the current approach to distributed site multi-homing, and how ILNP can provide similar levels of resilience – without introducing additional routing state, as is the case today.

6.1.1. Contribution. The contributions in this chapter are as follows: (A) To show that the SBR functionality demonstrated in Chapter 5 for a singleton router may be distributed across multiple SBRs, by using a simple synchronisation & control protocol following the soft state principle (refer to 6.2.1 & [130]); (B) to demonstrate a novel approach to the dynamic provisioning of site network capacity, based on the exchange of ILNP flow state between peer SBRs; (C) to verify that this approach – in common with the previous studies in Chapters 4—5 – does not introduce state to the IP routing system; and (D) to demonstrate that this approach adds minimal memory & packet processing overhead to the existing IP forwarding plane, by way of being directly based upon the work described in previous chapters.

6.1.2. Chapter structure. This chapter begins with a brief overview of how routing is distributed at a site border today, followed by a discussion of distributed ILNP operation at a site border [9, Sec. 3] – i.e. dSBR-ILNP:

(I) Section 6.2 on page 123 introduces the ILNPsync protocol and discusses coordination within dSBR-ILNP;

(II) Section 6.3 on page 128 describes changes to the ILNP forwarding plane required for distributed operation, including load sharing.
The evaluation in Sec. 6.4 studies dSBR-ILNP resilience to induced network failure. Results are presented and discussed in Sec. 6.5, followed by the chapter conclusion in Sec. 6.6.

6.1.3. Problem space. Today, robustness at site borders is usually achieved by deploying multiple SBRs, where each exchanges state – e.g. routing prefixes, reachability, and path preference – using a routing protocol (e.g. BGP). Each SBR shares a redundant view of routing state, i.e. to reduce the risk to connectivity posed by the failure of an individual SBR or link. This circuitous approach is necessary, as the IP network layer does not natively support network functions such as load sharing and/or fail-over. The routing prefixes used for this purpose are often de-aggregated, also – contributing to the routing table bloat discussed in Chapter 3, and subsequent “ID/Loc” proposals to address the problem.

![Figure 6.1.1. Scenario demonstrating robust site multi-homing with distributed SBRs (dSBRs)](image)

6.1.4. Approach. The claim is that ILNP can provide similar levels of resilience, yet without introducing routing state – as is the case with BGP today. Additionally, whilst it does not require encapsulation (tunnelling), it may be used optionally. Locator selection may be distributed, although this requires an additional level of indirection at each SBR: in this scenario, an SBR may now select non-local locators, also – discussed in Subsec. 6.2.3.

I have extended the implementation from Chapter 5 to provide this functionality: dSBR-ILNP – as described in [9, Sec. 3] – where several ILNP SBRs collaborate to
form a logically distributed router, as shown in the high-level diagram in Fig. 6.1.1. Peer SBRs are interconnected by an inter-router link (IRL) – indicated by the green arrow – and use IP multicast for mutual discovery. The IRL is a logical link; whilst the evaluation in this chapter used a single physical link, tunnelling might optionally be used to extend the distributed router between data centres; refer to Section 7.4.3 on page 160. However, the state exchange used in my evaluation (refer to Sec. 6.4) is limited to the ILNP dSBRs and does not introduce routing state.


This section describes co-ordination between distributed ILNP routers at the border of a single site.

6.2.1. ILNPSync protocol. I have devised a very simple\textsuperscript{1} protocol to co-ordinate multiple ILNP SBRs at a single site: ILNPSync. It performs three roles: (1) peer SBR discovery, (2) ILNP flow state exchange, and (3) failure detection – albeit with certain limitations. As the aim is to show that ILNP locator rewriting may easily be distributed, its features are deliberately constrained to those required for the evaluation in Sec. 6.4 only.

ILNPSync uses periodic heartbeat messages to signal node health. BFD\textsuperscript{2} [131, 132, 133] – a companion protocol used with BGP today – also provides failure detection using heartbeats. However, both ILNPSync and BFD are “soft-state” [130] protocols: state may be inserted explicitly, yet state withdrawal is implied – by missing heartbeat messages. So, both protocols will only detect unreachable peers after a number of expected heartbeats fail to arrive, i.e. within some protocol-specified time period. Additionally, BFD does not provide load sharing features, and does not provide discovery; it must be explicitly configured between peers.

\textsuperscript{1}Caveat: ILNPSync is not intended to represent a general approach to router synchronisation.

\textsuperscript{2}Whilst no absolute performance claims are made for BFD, it is typically able to achieve sub-second failover times in commercial implementations.
The implementation\(^3\) of ILNPSync is specific to dSBR-ILNP and is simple, requiring only \(\approx 250\) additional lines of code – although it assumes that IPv6 multicast is available on the IRL. By comparison, one existing state exchange protocol – SCSP\(^4\) [134] – requires \(\approx 6000\) lines of code, although it supports several roles related to IP-over-ATM, and is primarily intended for use in such an environment. Much of the functionality in ILNPSync might be provided by SCSP instead, however significant work would be required to implement it for use by ILNP.

6.2.2. IIP elections. The initial ingress peer (IIP) – i.e. the router\(^5\) where a packet was first received from an internal host – has a special role in the dSBR topology, which is discussed further in Sec. 6.3. An IIP may fail whilst it has ownership of one or more active flows; refer to Section 5.2 on page 95 for a discussion of ILNP flow state at a singleton SBR. So, an election mechanism is required to recover from the loss of an IIP – modulo the “soft-state” protocol behaviour discussed above. When the IIP is determined to be unreachable by the ILNPSync BEAT mechanism, other dSBR peers may claim IIP status (and ownership) for the flow. The IIP election algorithm has been adapted from OSPFv3 [135, Sec. C.3 Router Priority]: i.e. the winning peer is chosen according to the highest bitwise value of the 24 least significant bits of its IFID (as configured on its IRL link). The winning peer indicates this by issuing an ILNPSync NEW message to other peers over the IRL.

6.2.3. Peer locators. The ILCC – first described in Section 3.6.1.3 on page 41 – may be extended to include a hint\(^6\) that a source locator may be local only to another peer SBR. E.g. in Fig. 6.1.1, the site locator \(L_3\) is configured on a link connected to SBR2, but may be configured in the policy table at SBR1 – associated with its ILCC – as a peer locator. So, the additional indirection required to distribute locator selection also extends to next-hop selection. Where locator selection chooses a peer

\(^3\)The implementation studied here supports only a single physical IRL link – shown by the green arrow in Fig. 6.1.1. The IRL is also used to forward (hand-over) packets between dSBR nodes.

\(^4\)As used in the FreeBSD 6.0 Host ATM Research Project stack.

\(^5\)This is usually the SBR with the highest Default Router Priority in IPv6 RAs received by the host on the internal LAN.

\(^6\)Refer to Appendix B.5.1 on page 204 for configuration syntax.
Peer locators must either be advertised by a peer SBR in a claim or be locally available to the node; otherwise they will be ignored during locator selection. ILCC entries which are subsequently created from a matching policy entry inherit the peer locator behaviour. The hint is examined during flow creation and on the receipt of a state packet on the inter-router link. If the locator is configured locally on the node then the hint will be ignored. In addition, the peer locator mechanism does not use additional encapsulation or tunnelling.

6.2.4. **ILNPSync syntax.** The ILNPSync header format – shown in Fig. 6.2.1 – is intentionally similar to ICMPv6. This choice was made in order to streamline code and monitoring; e.g. the Checksum field is computed over the entire message payload. The HMAC field is discussed in Subsec. 6.2.5.

The Type field denotes the type of the message (**NEW**: 1, **EXPIRE**: 2, **BEAT**: 3). The Code field is currently ignored. All fields named Reserved must be set to 0. The NRecs field is reserved for future use by a batched update scheme; it must be set to 1 for **NEW** and **EXPIRE** messages, whereas the **BEAT** message uses this field to indicate how many locator claims are present. The contents of these sets are appended at the end of the payload – in the order of their counter fields. However, the order in which locators appear within each set is left to the implementation.
The syntax of **NEW** and **EXPIRE** messages is identical, as illustrated by Fig. 6.2.3. **NEW** messages are issued by the initial ingress SBR when a flow is created or updated. Peer SBRs should process an **EXPIRE** messages when received from the owning SBR (usually the initial ingress SBR), although ILCC state entries are treated as “soft state” and may expire independently.

### 6.2.5. Security

**ILNPSync** messages are normally exchanged only between ILNP SBRs over a dedicated (trusted) link. However, there is a basic mechanism available to protect against on-path attacks, in the form of a 256-bit HMAC-SHA1 digest. This is computed over each message payload, and the HMAC is also derived from a 160-bit pre-shared key [136, 137]. Additionally, system timers (e.g. the system uptime counter, and a reboot counter) may be used to further protect against timing-based replay attacks, much as has been done for SNMPv3 USM [138, Sec. 1.5.2 (3)].

### 6.2.6. Management interface

Each SBR exposes a local pseudo-interface named `ilnpsync` for configuration and monitoring. The protocol must be bound to the IRL by using the `sysctl` MIB variables described in Appendix B.5.3 on page 206. On startup, the SBR will join a dedicated multicast group on the IRL named by the `syncif` MIB variable.

Upon joining the group the SBR will begin to send **ILNPSync BEAT** messages periodically – shown in Fig. 6.2.2. These indicate peer availability – independently of the link layer – and may contain a list of site locators claimed by the originating SBR.

---

**Figure 6.2.2. ILNPSync BEAT payload:** This is used to indicate peer availability, and, optionally, locators claimed by the peer.

<table>
<thead>
<tr>
<th>Seqno</th>
<th>Period</th>
<th>Claim</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 6.2.3. ILNPSync state descriptor payload; both NEW and EXPIRE messages share this format. The NILocs and NTLocs counter fields specify the size of initiator and target locator sets present. To preserve the 64 bit alignment of IPv6 packets, appropriate padding is added at the end of the payload.

SBR. Each SBR must periodically advertise its claims as described in Subsec. 6.2.3. The default interval between BEAT messages is 10 milliseconds.
6.3. Method: Part II—Distributed Forwarding

This section describes how the ILNPv6 forwarding plane – described in Section 5.2 on page 95 – may be extended to distributed operation.

6.3.1. dSBR-ILNPv6 forwarding. Distribution of the forwarding process must be carefully controlled to prevent forwarding loops, i.e. roles and responsibilities must be well defined and enforced within the forwarding plane. This is especially important where multicast forwarding is used, as discussed in the next subsection. So, the initial ingress peer (IIP) is responsible for choosing the source locator of an egress packet. The IIP is also responsible for snooping – and rewriting – Locator Updates transmitted by an end host. However, LUs may also be relayed by a peer SBR as they are also subject to locator selection.

The IIP indicates its choice of source locator to the other peer SBRs by using locator rewriting on the packet itself. So, the packet does not require further encapsulation (tunnelling) when forwarded over the IRL. This demonstrates how a generic network function (e.g. distributed load sharing) may be realised in terms of ID/Loc without additional encapsulation. However, SBRs must not forward packets received on the IRL, unless they have a matching ILCC entry and were not previously addressed to the IRL.

6.3.2. Multicast hand-over. ILNPSync supports the use of a multicast IRL for both peer discovery and internal hand-over. When the locator selection algorithm at an ingress SBR chooses a peer locator, the packet will be forwarded on the IRL. An appropriate IPv6 multicast group may be chosen as its next-hop.

The use of a multicast next hop may seem counter-intuitive. However, IPv6 forwarders will normally accept packets regardless of link layer information, and base their forwarding decisions only on the unmodified IPv6 destination. Additionally, the use of multicast groups enables logical channelisation, as groups of ILNP flows enable efficient data delivery.

\[ ^8 \text{Implementations (e.g. packet filtering firewalls) may have more stringent checks. However, IPv6 forwarders must normally accept such packets e.g. to support point-to-point links.} \]
may then be forwarded on discrete link-layer multicast groups, potentially improving scalability\(^9\) for large numbers of flows.

Another advantage of this approach is that the next-hop need not be resolved during hand-over, which potentially reduces ILNP forwarding latency. However, it must be carefully implemented to avoid introducing undesired packet amplification. In FreeBSD, the `M_MCAST` flag — located in the `mbuf` chain header [139] — is an indication that a packet has been received (or is to be transmitted) using link-layer multicast. To ensure correct next-hop selection when a packet is leaving the dSBR, the egress peer must clear `M_MCAST` before the packet is forwarded.

6.3.3. Concurrency & optimisation. To keep its implementation simple, both ILNPSync and the dSBR-ILNP forwarding plane share a single mutex lock protecting both components. However, concurrency control in FreeBSD requires that a mutex lock be held on a unicast FIB entry [140] before transmission. When a multicast next-hop is in use on an Ethernet link — e.g. the IRL — this additional mutex lock is not required, as IPv6 resolves multicast addresses to an Ethernet group MAC address with a static mapping [141, Sec. 7]. Moreover, whilst the use of a multicast next-hop may reduce forwarding latency (i.e. as compared to a conventional forwarding path, which resolves the next-hop to a unicast address), this has not been measured — as the aim is to demonstrate distributed forwarding using locator rewriting, rather than evaluate dSBR-ILNP in terms of absolute performance.

6.3.4. Summary of Method: Parts I—II. In Sec. 6.2, I describe the ILNPSync “soft-state” protocol, which is used to co-ordinate multiple SBRs at a single site border. Additionally, in Sec. 6.3 I describe how locator rewriting is re-used within the dSBR — i.e to indicate the choice of egress router efficiently, and without requiring encapsulation (tunnelling). Whilst the ILNPSync protocol is quite simplistic, it is sufficient to demonstrate robustness to link failures within the evaluation in Sec. 6.4.

\(^9\)Certain switch fabrics — e.g. the Benes network — perform efficient replication of multicast packets using hardware techniques [26, Sec. 13.9].
6.4. Evaluation

In this section I aim to show that site multi-homing may be implemented \textit{resiliently}, without introducing additional routing protocol state – i.e. that both the load balancing and fail-over functions of ILNP may continue to operate between sites in the presence of node and/or link failures. All evaluation traffic is unicast.

6.4.1. Scope. The same experiment controls discussed in Subsec. 3.8.3 were in place throughout this scenario. As the evaluation is focused on load sharing (and fail-over) in the dSBR itself – and not the end-hosts\textsuperscript{10} – the scenario of a \textit{failed IIP} is not evaluated here; refer to Subsec. 6.4.3.1. I also assume that common best practice \cite{142} has been followed: e.g. nodes are co-located separately with independent power supplies.

6.4.2. Scalability analysis. The scalability analysis of saved routing state is almost identical to that for site multi-homing (as described in Section 5.3.2 on page 101); i.e. in IPv6, for a local site with $N_P$ site prefixes (locators), $N_I$ upstream ISPs, and $N_L$ SBRs, we would expect that the required number of advertisements will be $O(N_P \cdot N_I \cdot N_L)$ – assuming that, in the worst case, each SBR announces each de-aggregated site prefix individually in BGP. With ILNP, the state will be displaced to the DNS with $O(1)$ scale, i.e. a site’s locators will be retrieved from the DNS\textsuperscript{11} during the set-up of an ILNP flow. Whilst the policy configuration used by each peer SBR is static – yet the packet flow is dynamic – SBRs can mutually signal load sharing capacity to each other, and begin using it, \textit{without} changes to the FIB at each node. By contrast, a distributed load sharing scheme based on a routing protocol – e.g. OSPF, BGP – \textit{would} require such changes.

6.4.3. Experiment design. I have constructed an experiment topology of two site networks ($L_L$, $L_M$) interconnected by two ISP networks (i.e. by \texttt{ark} and \texttt{bay}) – shown in Fig. 6.4.1. Each site network contains a single host. The west site has three

\textsuperscript{10}End-hosts have only a single default route in this scenario. So, the overhead of ILNP at the end-host is unchanged from the 100/0 configuration, as discussed in Section 4.6 on page 77.

\textsuperscript{11}In my evaluation the DNS exchange is simulated by extending the \texttt{/etc/hosts} file syntax as in Section 4.2.4 on page 58.
SBRs (floor, hall and roof) and three global locators (L₁–L₃), whereas the east site has a single SBR porch with two global locators (L_A, L_B). Connectivity to each ISP is physically separate, and each global locator is associated with an independent link. Both ark and bay are configured as plain IPv6 routers and run an unmodified FreeBSD 8.4 kernel.

Hosts dais and spot exchange ICMPv6 ECHO flows (using bwping), and unidirectional UDPv6 flows originate from dais (using iperf). At the west site, the SBRs are configured to rewrite the internal site-wide locator L_L to globally visible locators (L₁–L₃) and to load share outgoing traffic on the link(s) corresponding to each global locator.

6.4.3.1. Induced link failures. As discussed in Subsec.6.4.1, I consider only link failure at peer SBRs in my analysis – and, by extension, the catastrophic loss of one or more peer SBRs. Two of the participating SBRs – hall and roof – are dynamically added and removed from the configuration by induced link failures: their links to the
6. DISTRIBUTED SITE MULTIHOMING

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th>Marker</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Trial start: floor active only</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>hall added to dSBR</td>
<td>2</td>
</tr>
<tr>
<td>60</td>
<td>roof added to dSBR</td>
<td>3</td>
</tr>
<tr>
<td>90</td>
<td>floor, hall, roof active</td>
<td>4</td>
</tr>
<tr>
<td>120</td>
<td>hall removed from dSBR</td>
<td>5</td>
</tr>
<tr>
<td>150</td>
<td>roof removed from dSBR</td>
<td>6</td>
</tr>
<tr>
<td>180</td>
<td>floor active only</td>
<td>7</td>
</tr>
<tr>
<td>210</td>
<td>“Cool-down” period</td>
<td>8</td>
</tr>
<tr>
<td>240</td>
<td>Trial end</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 6.1. Time line of each performance evaluation trial. Relative time periods are denoted in seconds since the start of the trial. Markers used to annotate time series plots in Sec. 6.5 are in the rightmost column.

IRL (shaded light grey in Fig. 6.4.1, which corresponds to the green arrow in Fig. 6.1.1 on page 122) and upstream ISP links L1- L3 (corresponding to the bold black arrowed lines in Fig. 6.1.1) are administratively disabled using the `ifconfig` command. They are later re-added to the dSBR confederation by re-enabling the same links.

6.4.3.2. On IIP failures. Handling catastrophic IIP failure gracefully in FreeBSD would require changes in its IPv6 ND and RA implementation, i.e. following recommendations in [143, Sec. 3] regarding the introduction of an unreachable state for ND cache entries associated with an IIP. However, the time-contingent load sharing behaviour would be obscured due to the “soft-state” nature of the ILNPv6 protocol – i.e. the delay inherent in detecting that an IIP is unreachable, as discussed in Sec. 6.2.

6.4.4. Time line. There are 8 phases lasting 30 seconds within each trial, listed by chronological order in Table 6.1. Each phase is induced by a scripted event trigger; the IIP remains active and reachable at all times as discussed in Subsec. 6.4.1. The locator distribution statistics shown in Fig. 6.5.5 are based on link-layer traffic counters. This is an artefact of `nd6_output()` being used to transmit forwarded ILNPv6 packets, which updates the link-layer counters and not the IPv6 protocol-level counters for the transit link. Moreover – in FreeBSD these counters can only be cleared on system boot. So, nodes were rebooted before each trial.
6.4.5. Metrics. A single load sharing configuration was used to ensure that each of three upstream links were used: 3/3/3. However, this configuration varied in time, as the number of active SBRs varied at particular phases in the experiment according to Table 6.1. So, in addition to the metrics discussed in Section 3.8.1 on page 43 – presented as both aggregate and time-series results – I have collected the distribution of received packets at each member SBR by locator.

Performance data was also collected at each node – to both validate the focal theory, and to evaluate the implementation. This included mbuf network buffer allocation and kernel malloc() statistics, as in Chapter 4 and Chapter 5. Most importantly, the malloc() pool counter for rtentry data structures – which tracks the allocation of routing state by the FreeBSD kernel – was not expected to change during any ILNP trials.

Finally, hand-over delay data was obtained – i.e. by using time-stamped tcpdump packet captures at floor and the IRL. PCS (described in Appendix B.2 on page 194) was used for post-hoc analysis of the packets captured from each measurement point. Simple arithmetic was used to calculate the change in wall-clock time as LU and data packets traversed the network; refer to Appendix B.3.4 on page 203 for details of network time synchronisation.

6.4.6. Configuration. The IRL was implemented as a discrete port-based VLAN. I assume at the start of the experiment that the ILNPSync protocol has already requested membership of the group$^{12}$ using the appropriate kernel APIs. A multicast listener report will be transmitted when the group$^{13}$ is first joined, although this may be delayed up to the Unsolicited Report Interval [144, Sec. 6.1] (default: 1 second). Moreover, I assume that multicast frames will be flooded to all connected ports – in the absence of specific port configuration – as required by the Ethernet bridging standard [145, Subsec. 7.9.4]. In a production deployment of ILNPSync, a network administrator might choose to segregate its traffic – either by using a discrete VLAN

---

$^{12}$Some network controllers require specific configuration for multicast.

$^{13}$An arbitrary link-scope multicast IPv6 address – ff02::80 – was used for ILNPSync control traffic in the experiment.
(as was used in this experiment), or by configuring MLD snooping on ports where the protocol is used [146].

Finally, tcpdump captures on the IRL were performed using a node which did not directly participate in the experiment. The use of multicast forwarding on the IRL facilitated capture without requiring invasive traffic replication techniques, e.g. switch-based port mirroring; whilst this capability was available, it is known to introduce bias in the form of additional switching latency.

6.5. Results

The main findings here are that dSBR-ILNP has low memory and packet processing overhead. Loss across trials was generally very low, although burst loss was observed during internal topology changes; refer to Subsec. 6.5.2. Jitter in all trials was <0.3ms\(^1\) and has been omitted for brevity. All measurements were performed at the destination node spot unless stated otherwise.

6.5.1. Aggregate stream results. The dSBR-ILNP implementation achieved close to 100% UDP goodput for each input load rate, up to \(\approx 80\)Kpps; aggregate ICMPv6 results have been omitted for brevity, as they were almost identical to those obtained for UDP. The results in this sub-section represent stream-based measurements taken over each trial as a whole, rather than as a time series. Fig. 6.5.1 shows the aggregate goodput measured over 25 trials for each load rate. Results for disparate frame sizes have been rebased to a common X-axis, as discussed in Section 3.8.2.1 on page 46.

6.5.2. Time-series stream results. Load sharing performed as expected; reordering is reported separately in Subsec. 6.5.3. Fig. 6.5.5 shows the distribution of received\(^1\) packets at 4 measurement points (links) whilst iperf was active: spot (the target end host), ark (on the link facing the IIP, floor), and bay (on the two

---

\(^1\)This is believed to be beyond measurement capability, as iperf is limited to 1\(\mu\)s precision by gettimeofday(): refer to Appendix B.3.3 on page 201 for system level timing details.

\(^1\)These measurements were obtained from link receive counters using the netstat command. Whilst dSBR-ILNP counts packets transmitted for each locator, these are unsuitable as they may be reset during configuration changes.
Figure 6.5.1. Aggregate goodput (as percentage of load rate) for unidirectional UDPv6 flows, between distributed and non-distributed ILNPv6 sites.

Figure 6.5.2. Aggregate packet reordering (as percentage of received packets) for unidirectional UDPv6 flows, between distributed and non-distributed ILNPv6 sites.
Figure 6.5.3. Aggregate sequence errors (as percentage of received packets) for Unidirectional UDPv6, between distributed and nondistributed ILNPv6 sites.

Figure 6.5.4. Aggregate payload loss (as percentage of transmitted volume) for Unidirectional UDPv6, between distributed and nondistributed ILNPv6 sites.
downstream links facing hall and roof respectively). In addition, there was a relatively small (≈2%) drop in throughput when dSBR peers other than the IIP were introduced or removed (i.e. during the events listed in Table 6.1). Burst loss was observed during these events. This may be attributed to the forwarding plane being blocked from running during ILNPSync state processing, as a single mutex lock is used for concurrency control; refer to Subsec. 6.3.3.

Figures 6.5.6–6.5.8 present UDP goodput, ICMPv6 packet loss, and UDPv6 byte (payload) loss, respectively, during the trial period; refer to Section 3.8.3.2 on page 48 for discussion of the gap between ICMPv6 and UDP results in the same experiment group. The periodic ICMPv6 goodput is not reported here, as these results are mostly identical to those observed in Fig. 6.5.6. For clarity, these results omit measurements reported during the “cool-down” period – i.e. from 210s-240s at the end of the trial. The results also exhibit a lead-lag effect; this is an artefact of the periodic reports used to obtain them, i.e. the effect of a topology change (refer to Table 6.1) may not be reported until the next 10-second reporting period.
Figure 6.5.5. Time series of load distribution by hop, for an ICMPv6-in-ILNPv6 flow transiting an ILNPv6 dSBR – as percentage of packets received at destination spot (summarised across all load rates – refer to Fig. A.1.4 on page 169 for detailed plots by each load rate). Refer to Table 6.1 for event time-line.
Figure 6.5.6. Time series of unidirectional UDP goodput for a flow transiting an ILNPv6 dSBR – as percentage of load rate, measured at destination spot (summarised across all load rates – refer to Fig. A.1.5 on page 175 for detailed plots by each load rate). Refer to Table 6.1 for event time-line.
Figure 6.5.7. Time series of ICMPv6 packet loss for a flow transiting an ILNPv6 dSBR – as percentage of total packets sent during trial, measured at source dais (summarised across all load rates – refer to Fig. A.1.7 on page 187 for detailed plots by each load rate). Refer to Table 6.1 for event time-line.
Figure 6.5.8. Time series of unidirectional UDP payload loss for a flow transiting an ILNPv6 dSBR – as percentage of total bytes sent during trial, measured at destination spot (summarised across all load rates – refer to Fig. A.1.6 on page 181 for detailed plots by each load rate). Refer to Table 6.1 for event time-line.
6.5.3. Reordering distribution. Packet reordering was observed when two (or more) dSBR peers were active; refer to Table 6.1. Fig. 6.5.10 presents a time series of reordering, as measured at the destination host spot. Reordering is expected in multipath networking scenarios, and is not indicative of pathological behaviour; refer to Section 7.3 on page 156 for a fuller discussion of its implications. So, to examine reordering behaviour in more detail, Fig. 6.5.9 shows the distance from expected packet sequence for late UDP packets. iperf was modified to collect these results to 30 positions, although X-axis limits are set to show only relevant data points, and packet sizes are expressed in terms of Ethernet encapsulation to ease comparison. For 128 byte frames, arrivals were no later than 8 positions out of sequence. However, 1514 byte frames demonstrate a greater degree of reordering.

The effects of queueing delay are visible in Fig. 6.5.9. The displacement distribution is not strictly logarithmic, and has a longer tail for 1514 byte frames due to their longer dwell time. Additionally, the wide distribution may be attributed to the cumulative effect of each transit network hop, as the service time for each link-layer queue varies independently. Fig. 6.4.1 on page 131 shows that there are three intermediate hops on the path for each egress packet; this count increases to four when all dSBR peers are active, i.e. between events 2 and 3 in Table 6.1. Moreover, the final link on the path to spot (i.e. indicated by \( L_M \) in Fig. 6.4.1) acts as a bottleneck. So, the level of reordered packets is 0% when 1 site locator is active, \( \approx 50\% \) when 2 site locators are active, and \( \approx 66\% \) when 3 site locators are active. Finally, the effects of interrupt coalescing are visible as periodic gaps in the distribution. It acts as a notch or band-stop filter – with a 96KHz frequency\(^\text{16}\) – applied to queue service time at each link, i.e. the cumulative dispersion of a packet stream corresponds to a step function of sequence [147, Fig. 2].

\(^{16}\)Refer to Appendix B.3.2 on page 200 for configuration details.
Figure 6.5.9. Relative frequency of sequence displacement (as percentage of received packets) for Unidirectional UDPv6, between distributed and non-distributed ILNPv6 sites (summarised for all load rates – refer to Fig. A.1.3 on page 168 for detailed plots by each load rate).

(A) 1514 byte Ethernet frames

(B) 128 byte Ethernet frames
Figure 6.5.10. Time series of unidirectional UDP packet reordering for a flow transiting an ILNPv6 dSBR – as percentage of total packets received, measured at destination spot (summarised across all load rates). Refer to Table 6.1 for event time-line.
6.5. RESULTS

Figure 6.5.11. Peer activation delay: Difference in time between transmission of the first locator claim of newly active peer hall, and the first transit packet containing its site locator (both measured at the IRL)

6.5.4. dSBR responsiveness. The main finding here is that whilst the dSBR is able to use new network capacity very quickly, the deferred LU mechanism used by the implementation\(^\text{17}\) (refer to Section 4.3.4 on page 63) introduces some unnecessary delay when responding to a loss of capacity. The response time of the dSBR during internal topology changes (refer to Table 6.1) is shown in Figures 6.5.11 - 6.5.14.

Firstly, Fig. 6.5.11 shows how long the dSBR takes to use new network capacity when it becomes available. Fig. 6.5.12 shows how quickly the dSBR forwarding decisions change to account for failed nodes (i.e. lost capacity), whereas Fig. 6.5.13 shows the time taken to notify an external ILNP endpoint of such failures (i.e. by issuing an updated LU for the flow). However, Fig. 6.5.13 does not include the time taken for the updated LU to reach – and be processed by – the external endpoint, as this evaluation was constrained to the dSBR itself.

Finally, Fig. 6.5.14 presents results from Figures 6.5.12 - 6.5.13 on the same plot for comparison: whilst the existing LU output path defers LUs by a fixed 500ms interval to avoid state implosion, the error bars show that transmission of the required LU is otherwise timely – i.e. it is subject only to jitter and delay inherent to how FreeBSD schedules the execution of time-contingent callout routines.

\(^{17}\)With the exception of changes required to support peer locators, the LU transmission code is unmodified from that described in Sec. 5.2.
6. DISTRIBUTED SITE MULTIHOMING

Figure 6.5.12. Internal failover delay: Difference in time between transmission of the last locator claim of failed peer SBR hall, and the final transit packet containing its site locator (both measured at the IRL)

Figure 6.5.13. External signalling delay after failover: Difference in time between transmission of the last locator claim of failing peer SBR hall (measured at the IRL), and the transmission of a Locator Update withdrawing its site locator (measured at the ingress SBR, floor)

6.5.5. Memory overhead. The dSBR-ILNP implementation uses very little memory, and does not introduce routing state. Table 6.2 shows that the additional non-routing state required at each peer SBR – beyond the requisite IPv6 FIB, ND and RA structures – is less than a single 4KB memory page on the x86-64 machine architecture. The rtentry pool counter – first discussed in Section 3.8.1.1 on page 45

---

18The results presented in this subsection come from the output of the vmstat -m command before and after each trial.
6.5. RESULTS

Figure 6.5.14. Mean internal failover and external signalling delay, showing effect of deferred Locator Update transmission at the ingress SBR – remained constant over the lifetime of each trial. So, this result provides strong support for my hypothesis: i.e. that site multihoming may be resiliently implemented without introducing routing state (or unnecessary encapsulation), by using dSBR-ILNP.

The mbuf allocation plots in Figures 6.5.15 - 6.5.17 show the dynamic network buffer usage at each of the three dSBR peers – shown by orange rounded rectangles in Fig. 6.4.1 on page 131. These results are presented with a log-1-2-5 scale on the X axis (load rate) and a log-10 scale on the Y axis (absolute requests). The plots are intended as a sanity check on the implementation, and are not intended as an evaluation of absolute performance. Figures 6.5.15 - 6.5.17 show expected results for the controls, factors, and implementation used. Load rates are specified in a log-1-2-5 pattern, so the relationship between load rate and buffer overhead appears linear. There are no memory leaks, a modest number of clusters is used for transmitting LU messages, and the tags used to demultiplex the ILNP Nonce Option are quickly recycled by the system.
<table>
<thead>
<tr>
<th>Type</th>
<th>Bytes</th>
<th>Count, static</th>
<th>Count, 1 flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td>240</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>loc</td>
<td>160</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>locpolentry</td>
<td>64</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>nid</td>
<td>40</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>pol</td>
<td>48</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>sypeer</td>
<td>72</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>syclaim</td>
<td>32</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Overhead, static, 3 peers 1448
Overhead, for each peer detected on IRL 376
Dynamic overhead, 1 flow, 3 peers 720

\( ^a \) Instances created when a flow is active.

\( ^b \) Instances created when a dSBR peer is detected on the IRL.

\( ^c \) One instance is created for each locator claimed by a peer.

\( ^d \) Peer locators are associated with a discrete policy entry.

Table 6.2. ILNPv6 memory overhead at each dSBR peer with a single site locator and a single LAN. Sizes are for the x86-64 machine architecture. C structure names omit the `ilnp6_` prefix for brevity.
Figure 6.5.15. Cumulative mbuf buffer requests during an ICMPv6 trial – measured over 25 trials at dSBR IIP floor (for all load rates)
Figure 6.5.16. Cumulative mbuf buffer requests during an ICMPv6 trial – measured over 25 trials at dSBR peer hall (for all load rates)
Figure 6.5.17. Cumulative mbuf buffer requests during an ICMPv6 trial – measured over 25 trials at dSBR peer roof (for all load rates)
6.6. Chapter summary

In this chapter, I have demonstrated a novel, distributed site router based upon ILNP: dSBR-ILNP. The system provides an example of resilient site-controlled multihoming – without introducing state to the routing system, as is the case with IP today; refer to Section 2.3 on page 12. In addition, dSBR-ILNP has very little overhead in terms of memory & packet processing time – and does not require the use of tunnelling. The ILNPSync protocol was used to coordinate member SBRs. This protocol also provides some additional functions: e.g. the failure detection required by a resilient forwarding scheme (refer to Sec. 6.2), and peer discovery, which partially automates the provisioning of new network capacity.

Moreover, the implementation has not been optimised beyond the re-use of the FIB cache discussed in Section 4.3.5 on page 65. The ILNP forwarding plane at each SBR now also contains a relatively small amount of ILNPSync state for each available peer router. However, no additional state has been introduced to the routing system – beyond that which is required for normal routing in IPv6. So, the role of de-aggregated prefixes and multiple advertisements might be replaced by site-defined ILNP locator policies.

The dSBR-ILNP implementation has a known pessimisation: forwarding plane execution may be blocked by ILNPSync input processing, as discussed in Subsec. 6.3.3. This was observed in Subsec. 6.5.2 as burst loss during internal topology changes. Future work might study the use of an alternative to ILNPSync which can scale beyond a single site (refer to Subsec. 7.4.3), and how to address the pessimisation inherent to the dSBR-ILNP forwarding plane; e.g. using lockless data structures.
Chapter 7

Conclusions

7.1. Summary

I began this dissertation in Chapter 2 by describing multihoming in terms of its applications (e.g. load sharing & failover), and the limitations inherent to its realisation in the IP addressing & routing system used today. In Sec. 2.3, I have described how this leads to the problem of routing table bloat; whilst closely related to the shortage and fragmentation of IP address space (refer to Subsec. 2.2.1), the problem is now global in scope – despite the Internet research & engineering communities having been aware of the problem for over 35 years. Whilst NAT has been pressed into service to maintain connectivity, this regressive approach has many disadvantages; refer to Subsec. 2.3.5.

In many respects, location independence has emerged as IP’s missing feature; IL-SAs (described in Subsec. 2.4.2) represent an alternative interpretation of the ideas discussed in [12, Pg. 2], which embrace it as a core design principle; in such architectures, multihoming enjoys first class support. Each proposed solution discussed in Chapter 3 attempts to solve the problem differently; however, all such proposals follow the general approaches described in Subsec. 2.4. Within this solution space, ILNP is unique: it re-uses the existing IP addressing and routing system, and does not introduce additional routing state to provide multihoming functions. Moreover, it also benefits from the adoption of IPv6 (refer to Subsec. 7.4.1 & Section 2.4.3 on
So, this pragmatic & evolutionary approach allows existing networking infrastructure to be preserved; as discussed in Chapter 2, an incrementally deployable solution is preferred.

In Chapter 4 I have shown how ILNP may be used to implement multihoming functions at end-hosts, using the example of load sharing. As stated in Section 2.2 on page 7, host multihoming does not enjoy direct support in IP today; the ILNP-based approach achieves this without introducing routing state, nor does it require hosts to participate in the routing system. Moreover, my implementation has minimal performance impact, and existing IPv6 code is efficiently re-used – e.g. router and neighbour discovery. So, many of the benefits of IPv6 (discussed in Section 2.4.3 on page 20) are preserved; e.g. automatic address configuration in the form of SLAAC, and site renumbering (not demonstrated in this work).

Following this in Chapter 5, I have described how ILNP provides site-controlled multihoming capabilities, and demonstrated load sharing between two sites; again, without requiring the advertisement of de-aggregated prefixes required with IP today (refer to Sec. 2.3). This approach has little effect on performance as compared with NPTv6, and both ILNP hosts and routers co-exist with IPv6 transit routers. In addition, my sbr-ILNP prototype has similar (in some cases, better) performance than a pf-based NPTv6 implementation. Moreover, the changes required to support ILNP at the SBR are quite limited, and may be implemented in terms of existing IPv6 networking code.

Finally, in Chapter 6, I have demonstrated how ILNP can provide robust site multi-homing – using the example of a novel, distributed router with dynamically provisioned network capacity. The approach does not introduce routing state, and has minimal performance impact. Whilst the scenario was subject to certain limitations of IPv6 ND (refer to Section 6.4.3.2 on page 132), peer SBRs may be added and removed to the distributed router without interruption to active traffic flows. Additionally, the burst loss observed during topology changes might be addressed by employing alternative concurrency control schemes. The distributed ILNP SBR has

\[1\text{Whilst some authors \cite[Sec. 6.3.3]{36} describe ILNP as a host-based ILSA, this work demonstrates its application at sites through additional engineering.}\]
potentially wider applications in the Network Function Virtualisation (NFV) space; refer to Sec. 7.4.

7.2. Limitations of this work

The main focus of this work has been the application of ILNP to *host & site multi-homing* and the elimination of otherwise redundant *routing state*. Whilst enabling network mobility with ILNP overlaps in the area of implementation, it lies outside the scope of this dissertation – although it is the subject of previous [102] & ongoing research. So, whilst I have not studied my implementation in simulated Internet environments (e.g. PlanetLab\(^2\)), experiments were designed to be general in scope; refer to Section 3.8 on page 43. More specifically, comparative evaluations with existing protocols (i.e. IPv6 & NPTv6, as studied in Chapters 4 & 5, respectively), were designed in such a way that direct comparisons can be made, and conclusions drawn based on these scenarios.

Additionally, in this work I have confined myself to a study of ILNP and its application in terms of the *IPv6 network layer*, modulo the changes required to support UDP *iperf* sessions (discussed in Section 4.4.4 on page 72). Whilst a full host implementation of ILNP would require the modification of *all* transport protocols intended to be used with it (minimally, TCP & UDP), this is unlikely to significantly affect the results for ILNP *locator rewriting* presented in this work (as it is implemented at the *network layer*). The problems of interposing network stack functionality beneath the transport layer are common to several other proposals (e.g. SHIM6 & HIP); refer to Chapter 3 & 4, and the discussion of future work in Subsec. 7.4.1. Moreover, the experimental implementation has been exercised only with a single active ILNP flow; the mapping of multiple transport layer sessions to flows [6, Pg. 10] has not been demonstrated.

Regarding the use of ILNP within IPv4: whilst a specification has been published for ILNPv4 [117, 148, 149], its implementation requires some problematic and invasive changes to the IPv4 stack, and also lies outside the scope of this work. The IETF is transitioning IPv4 to “sunset” status, and ILNP deployment potentially benefits

\(^2\)http://www.planet-lab.org/
from changes in IPv6 to support multihoming more generally; refer to Section 2.4.3 on page 20 & Subsec. 7.4.2.

Finally, the claim in this work is that ILNP does not require significant change to existing code bases; the implementation used throughout this work required ≈3500 additional lines of code in FreeBSD 8.4 (excluding comments). I have not attempted to demonstrate these modifications in the context of a commercial routing product or operating system, and such work lies outside the scope of this dissertation. However, as the FreeBSD code base is well-known (and several commercial products\(^3\) are based upon it, including Apple OS X), it is reasonable to suppose that such a demonstration is feasible – as, whilst the ILNPv6 forwarding plane is implemented as a separate set of code paths from IPv6, it re-uses much of the existing IPv6 stack; refer to Chapters 4 & 5.

### 7.3. On reordering

In Chapters 4, 5 & 6, I have presented aggregate results for packet reordering and its sequence displacement distribution. In each scenario, multiple paths are possible for each packet – based on the configured locator precedence. However, this leads to some paths being favoured at the expense of others by the load sharing algorithm; refer to Section 4.3.3 on page 62. This algorithm selects the next-hop for each packet individually. The results I have observed are expected, and may be explained in terms of queueing theory. Whilst the effects may vary depending upon the topology used in each experiment, there are four factors common to each instance of high packet reordering, in descending order of their effects:

**Skewed path ratios:** Where an 80/20 load sharing ratio was configured for the experiments in Chapters 4 & 5, high reordering was observed – as one path is favoured over alternate paths in both topologies. The time-varying 3/3/3 split used in the dSBR-ILNP experiment exhibits the widest (and most skewed) distribution when all three dSBR peer routers are active; refer to Fig. 6.5.9 on page 143. Both the 80/20 and periodic 3/3/3 load sharing ratios represent a worst case amongst the packet reordering results in this work.

\(^3\)https://en.wikipedia.org/wiki/List_of_products_based_on_FreeBSD
“Elastic” queues: In Section 3.8.3 on page 47 I have stated that as the aim is to compare relative protocol performance (and not hardware performance), the queues at each network element (i.e. hosts & routers) are allowed to grow to the maximum supported by the hardware; refer to Appendix B.3.2 on page 200. So, rather than dropping packets when a network element is congested, they will be queued for delayed delivery. The effects of such “elastic” queueing present themselves in the results as higher reordering at higher load rates.

Frame size: Larger packets have a longer dwell time within each link-layer queue. The effects of this are most apparent in Fig. 6.5.9a on page 143, where the tail of the reordering distribution for 1514 byte Ethernet frames is elongated (i.e. greatest sequence displacement) as compared with 128 byte frames.

Interrupt coalescing: To demonstrate that ILNP is capable of operating at 1Gbit/s Ethernet load rates, interrupt coalescing (IC) was enabled\textsuperscript{4} during each experiment (also described in Subsec. B.3.2). Many modern network controllers use interrupt-driven I/O (as opposed to polling); so, as a hardware-based engineering optimisation, IC aims to increase throughput by imposing a time limit on the interrupt routine used to service received (or transmitted) packets. In terms of queueing theory, IC modulates the service time at each link-layer queue according to a step function of packet sequence [147, Fig. 2]. If a link-layer queue still contains packets beyond the IC time window, these packets will not be serviced until the next time window. So, the effects of IC present themselves as periodic gaps in each reordering distribution plot.

Together, these factors explain – in terms of link-layer queues – why high reordering may be observed in apparently simple network topologies. Reordering is to be expected in multipath configurations, and may arise at multihomed sites – regardless of whether IP or ILNP is the network layer in use. However, these results have some implications for transport-layer protocols; specifically, TCP [150, Sec. III-A]. Where

\textsuperscript{4}Refer to Appendix B.3.2 on page 200 for configuration details.
packets are more than 2 positions out of sequence, the Fast Re-transmit [151, Sec. 3] feature of TCP is activated; this may pessimise performance by inducing unnecessary re-transmissions of reordered TCP segments, as TCP treats segments reordered beyond 2 positions as being lost. As discussed in Section 3.5.3 on page 37, MP-TCP addresses this issue by running a separate instance of its congestion control algorithm for each sub-flow (i.e. discrete network path).

7.4. Further work

7.4.1. Edge deployment. I have described the problem of routing table bloat at length in Chapter 2, and have stated that – common to all approaches discussed in Chapter 3 – an incrementally deployable solution is preferable for logistical reasons. Based on measurements of the IPv6 routing system, some authors have observed\(^5\) that the uptake of IPv6 has been limited to the core of the network. However, by using ILNP – in conjunction with the changes which IPv6 makes to routing & addressing (discussed in Subsec. 2.4.3) – edge networks may no longer require direct participation in the routing system (i.e. in the form of BGP peering) to implement multihoming functions. Moreover, ILNP does not require changes to core routers; refer to Section 4.2 on page 56.

So, as deployment of ILNP would be focused at the edge of the network, it would be reasonable to pursue its implementation within such edge devices. Today, Internet Access Devices (IADs) are often constructed using off-the-shelf components to reduce cost, including the use of the Linux\(^6\) operating system. One such device – the Linksys WRT54G – has seen widespread use [153, 154] within networking research. So, following the discussion in Subsec. 7.2, the implementation of ILNP in IADs – and in networking stacks commonly used by end-users (e.g. Microsoft Windows, Linux,


\(^6\)Support for ILNP in Linux is a work in progress [152].
Apple\textsuperscript{7} Mac OS X) – is both feasible, and a desirable goal to enable its widespread deployment. I discuss some of the required work to enable wider uptake in the following Subsec. 7.4.2.

Finally, multihoming has – to date – required direct participation in the routing system. Today, end-users enjoy a choice of heterogeneous access technologies: e.g. 4G, 802.11, UMTS, DSL, and Ethernet. The novel, distributed site router demonstrated in Chapter 6 is an excellent example of how ILNP could – in future – be applied to enable wider consumer choice and flexibility. By refining the techniques employed in dSBR-ILNP – e.g. by standardising the interworking between peer SBRs, and automating their configuration – adding robustness and additional network capacity to a home network might be made as simple as plugging in a new appliance.

7.4.2. On the legacy use of IP addresses by applications. Throughout this work, I have studied the application of ILNP at the unicast IPv6 network layer only. The scheme described in Section 4.4.3 on page 70 aims to preserve backwards compatibility for legacy applications, however it is limited to UDP (and \texttt{iperf}) specifically. Following the general discussion of IP address entanglement in Section 2.2.2 on page 11, I have described in Section 4.4.4 on page 72 how – even if these uses of IP address bits can be disentangled at the network layer – applications may still use them as identifiers, with the expectation that they retain the properties discussed in Subsec. 4.4.3.

These problems are common to other proposals (e.g. HIP & SHIM6, discussed in Sec. 3.5) and would need to be addressed to achieve wider uptake of ILNP, given its nature as a host-based protocol. However, such problems are exacerbated where the use of IP addresses as identifiers are central to the application protocol. Peer-to-peer (P2P) applications – e.g. BitTorrent [155], and Adobe’s proprietary Real Time Media Flow Protocol [156] – are a notable and widespread example. The problem of \textit{binding dynamicity} is – to some degree – inherent to the sockets API in wide use today; so, the API may need to change to reflect the uptake of ILSAs. Such change is problematic, as application developers working with a high-level language (e.g. Java, Python, Go)

\footnote{Apple first adopted the MP-TCP protocol in 2013: \url{https://support.apple.com/en-gb/HT201373}}
rely on a *trickle-down* effect; they need time to adapt to new systems-level facilities, which are often exposed only at the relatively low level of the C runtime environment.

Moreover, several applications in common use today – e.g. Skype\(^8\) & μTorrent\(^9\) – are distributed *only* as precompiled binaries. Whilst these would benefit from the multihoming functions provided by ILNP, it is difficult to change their behaviour without the use of specific techniques: e.g. API-based translation, of which the work described in Subsec. 4.4.3 is a deliberately constrained example. Finally, a finer-grained expression of locator precedence may be required for use by applications; the Application-Layer Traffic Optimization (ALTO) Protocol [157] provides a framework for traffic engineering which is driven by end host requirements, and this might form a suitable starting point for such work.

### 7.4.3. Lightweight Network Function Virtualisation

The lightweight and flexible properties of ILNP lend themselves well to addressing the problems of *data-centre networking*; specifically, *virtualisation*. Today, Network Function Virtualisation (NFV) forms a distinct problem space, often implemented in terms of Software Defined Networking (SDN). Whilst SDN may be employed to achieve NFV – i.e. by decoupling routing from physical location, often employing ILSAs or ILSA-like techniques – this requires widespread deployment of “flow controllers” and SDN-capable routers & switches. However, ILNP is a relatively lightweight approach within the ILSA space. Distributed NFV functions could be implemented in terms of ILNP locator precedence, with a protocol such as SCSP (discussed in Sec. 6.2) used to distribute forwarding state. Tunnelling – e.g. in the form of the Virtual Extensible LAN (VXLAN) protocol [158] – could optionally be applied directly to the model demonstrated in Chapter 6, allowing sites – distributed across disparate topologies & equipment – to reside within the same logical NFV domain.

Moreover, the distribution of compute tasks is a common application in the NFV space. Today, this often requires the use of “middleboxes”, e.g. in the form of “flow directors” – appliances which direct requests to various servers, based on factors such as geographic location and system load. In addition, tasks may be migrated

\(^{8}\)http://www.skype.com/en/

\(^{9}\)https://www.utorrent.com/
between nodes according to the same factors. One recent proposal [159] describes a
task distribution scheme which does not require “middleboxes”. It is loosely based
on ILNP; by using a modified I-LV syntax (refer to [159, Sec. 2] & Section 4.2 on
page 56), the route used to reach a virtual host may also be decoupled from its physical
location. The proposal also describes how task migration may be achieved through
the re-use of locator rewriting [159, Sec. 5.1]. This approach also has the advantage
that encapsulation is not required, potentially yielding cost and performance benefits.

7.5. Closing remarks

In this dissertation, I have demonstrated ILNP and its application to solving certain
problems which arise with IP today regarding multihoming and network functions.
Moreover, the technique of locator rewriting – demonstrated in this work as a first
class function of the operating system at end-hosts – may enable end-to-end software
defined networking within the IP architecture, as ILNP may be realised as
an extension of the existing IP software stack. In particular, the work in Chapter
6 demonstrates a novel approach to dynamic provisioning of network capacity, and
provides a compelling alternative to the approaches examined at length in Chapter
2. If one considers location independence as IP’s missing feature, then – as shown
by the work presented in Chapters 4—6 – this work might form the foundation of
incremental refinements to IP and the connectivity which it can provide to end-hosts.
Appendix A

Detailed results

This appendix contains additional plots with further detail for the scenarios studied in Chapters 4-6. For the legend used in the time series plots in Fig. A.1.4 on pages 169-174, refer to Fig. 6.5.5 on page 138.

List of Additional Figures

A.1.1 Locator Selection, Detailed sequence displacement 164
A.1.2 Locator rewriting, Detailed sequence displacement 166
A.1.3 Distributed rewriting, Detailed sequence displacement 168
A.1.4 Distributed rewriting, 3-way split, % Load sharing (time series) 169
A.1.5 Distributed rewriting, 3-way split, % UDP Goodput (time series) 175
A.1.6 Distributed rewriting, 3-way split, % UDP payload loss (time series) 181
A.1.7 Distributed rewriting, 3-way split, % ICMPv6 packet loss (time series) 187
Figure A.1.1. Relative frequency of sequence displacement (as percentage of received packets), between two ILNPv6 hosts using Locator Selection (across all load rates). These results expand upon those in Fig. 4.6.8 on page 87. (cont. pg. 165)
(c) 128 byte Ethernet frames, 50/50 split

(d) 128 byte Ethernet frames, 80/20 split
Figure A.1.2. Relative frequency of sequence displacement (as percentage of received packets), between two ILNPv6 sites using Locator Rewriting (across all load rates). These results expand upon those in Fig. 5.4.9 on page 116. (cont. pg. 167)
(c) 128 byte Ethernet frames, 50/50 split

(d) 128 byte Ethernet frames, 80/20 split
Figure A.1.3. Relative frequency of sequence displacement (as percentage of received packets) for Unidirectional UDPv6, between distributed and non-distributed ILNPv6 sites (across all load rates). These results expand upon those in Fig. 6.5.9 on page 143.
Figure A.1.4. Time series of load distribution by hop, for an ICMPv6-in-ILNPv6 flow transiting an ILNPv6 dSBR (as percentage of packets received at destination spot). These results expand upon those in Fig. 6.5.5 on page 138 and use the same legend. Refer to Table 6.1 on page 132 for event timeline. (cont. pg. 170-174)
(c) 1514 byte Ethernet frames, 200 Mbps load rate ($\approx 20\text{Kpps}$)

(d) 1514 byte Ethernet frames, 500 Mbps load rate ($\approx 50\text{Kpps}$)
(E) 1514 byte Ethernet frames, 1000 Mbps load rate ($\approx$100Kpps)
(f) 128 byte Ethernet frames, 1 Mbps load rate (≈1Kpps)

(g) 128 byte Ethernet frames, 10 Mbps load rate (≈10Kpps)
(H) 128 byte Ethernet frames, 20 Mbps load rate ($\approx 20$ Kpps)

(i) 128 byte Ethernet frames, 50 Mbps load rate ($\approx 50$ Kpps)
(1) 128 byte Ethernet frames, 100 Mbps load rate (≈100Kpps)
Figure A.1.5. Time series of unidirectional UDP goodput for a flow transiting an ILNPv6 dSBR – as percentage of load rate, measured at destination spot. These results expand upon those in Fig. 6.5.6 on page 139. Refer to Table 6.1 on page 132 for event timeline. (cont. pg. 176-180)
(c) 1514 byte Ethernet frames, 200 Mbps load rate (≈20Kpps)

(d) 1514 byte Ethernet frames, 500 Mbps load rate (≈50Kpps)
A. DETAIL RESULTS

Time within trial (s)

% UDP target goodput

(E) 1514 byte Ethernet frames, 1000 Mbps load rate (~100Kpps)
(f) 128 byte Ethernet frames, 1 Mbps load rate (≈1Kpps)

(g) 128 byte Ethernet frames, 10 Mbps load rate (≈10Kpps)
(H) 128 byte Ethernet frames, 20 Mbps load rate (≈20Kpps)

(i) 128 byte Ethernet frames, 50 Mbps load rate (≈50Kpps)
(1) 128 byte Ethernet frames, 100 Mbps load rate (≈100Kpps)
Figure A.1.6. Time series of unidirectional UDP payload loss for a flow transiting an ILNPv6 dSBR – as percentage of total bytes sent during trial, measured at destination spot. These results expand upon those in Fig. 6.5.8 on page 141. Refer to Table 6.1 on page 132 for event timeline. (cont. pg. 182-186)
A. DETAILED RESULTS

(c) 1514 byte Ethernet frames, 200 Mbps load rate (≈20Kpps)

(d) 1514 byte Ethernet frames, 500 Mbps load rate (≈50Kpps)
(E) 1514 byte Ethernet frames, 1000 Mbps load rate (≈100Kpps)
(f) 128 byte Ethernet frames, 1 Mbps load rate (≈1Kpps)

(g) 128 byte Ethernet frames, 10 Mbps load rate (≈10Kpps)
(h) 128 byte Ethernet frames, 20 Mbps load rate (≈20Kpps)

(i) 128 byte Ethernet frames, 50 Mbps load rate (≈50Kpps)
(1) 128 byte Ethernet frames, 100 Mbps load rate ($\approx$100Kpps)
Figure A.1.7. Time series of ICMPv6 packet loss for a flow transiting an ILNPv6 dSBR – as percentage of total packets sent during trial, measured at source dais. These results expand upon those in Fig. 6.5.7 on page 140. Refer to Table 6.1 on page 132 for event timeline. (cont. pg. 188-192)
(c) 1514 byte Ethernet frames, 200 Mbps load rate (≈20Kpps)

(d) 1514 byte Ethernet frames, 500 Mbps load rate (≈50Kpps)
A. Detailed Results

(e) 1514 byte Ethernet frames, 1000 Mbps load rate (≈100Kpps)
(f) 128 byte Ethernet frames, 1 Mbps load rate (≈1Kpps)

(g) 128 byte Ethernet frames, 10 Mbps load rate (≈10Kpps)
(h) 128 byte Ethernet frames, 20 Mbps load rate (≈20Kpps)

(i) 128 byte Ethernet frames, 50 Mbps load rate (≈50Kpps)
(1) 128 byte Ethernet frames, 100 Mbps load rate (≈100Kpps)
Appendix B

Experimental Apparatus

This appendix describes the tools used to develop FreeBSD/ILNP, the evaluation environment used to conduct ILNP performance & validation studies, and the measurement tools used to perform these studies. Refer to Section 3.8 on page 43 for a discussion of the evaluation methodology adopted by this work.

B.1. Development environment

In this work, the operating system is the model, as opposed to using a simulation. The changes to the FreeBSD networking stack described throughout Chapters 4—6 require a specific set of tools which are quite different from those used for application development.

**KScope:** This application\(^1\) combines a text editor with a cross-referencing environment, based on the \texttt{cscope}\(^2\) source code indexing engine. It is particularly suitable for working with large, existing code bases (e.g. operating systems kernels, and multi-process applications). It also allows call graphs to be extracted and saved in the \texttt{.dot} format used by the GraphViz\(^3\) suite of visualisation tools. Here it has been used for development of the ILNP code base, and for producing the call graph diagrams in each experiment chapter. The development of Locator Selection involved the review of 15 separate (related) C source files, which was greatly assisted by its use.

\(^{1}\text{http://sourceforge.net/projects/kscope4/}\)

\(^{2}\text{http://cscope.sourceforge.net/}\)

\(^{3}\text{http://graphviz.org/}\)
VMware Fusion: This is a hosted (i.e. Type 2) hypervisor for the Mac OS X operating system. It was used as a test environment for the FreeBSD network stack changes described in Chapters 4—6, and as the evaluation environment for the initial work upon which Chapter 4 is based [1].

Each experiment was performed using a dedicated hardware testbed. Its configuration is described below, with relevant details provided within the evaluation section of each chapter. Additionally, in the early stages of development, the ping command was modified to enable ILNP operation on its socket descriptor, and to interpret received ILNP option data.

B.2. Data collection & analysis

The re-use of common tools from the SciPy ecosystem [160] allowed for rapid analysis of the data from each experiment as it was collated. Python was a convenient environment for analysis, as its native I/O facilities are able to accommodate the .tar format used for experiment logging with minimal work. However, each tool has a learning curve which researchers must master to make effective use of these tools.

NumPy: This library is widely used for numerical work in Python [161]. Here, it has been used primarily for post-processing the raw data collated from each experiment; it is also used by other libraries listed below. Whilst NumPy is efficient for large array processing in Python, it lacks the data-driven analysis capabilities found in Pandas.

Pandas: This is a high-level data analysis library [162] which provides facilities similar to those offered by more familiar environments such as R and Matlab. It relies upon NumPy for low-level array manipulation. It has a rich set of data-driven analysis tools, and has been used for most of the analysis throughout this work.

IPython: This is an interactive environment which provides a notebook-style graphical interface to Python [163]. Here it was used for exploratory analysis of experiments in progress, and was also used for prototyping the final analysis used to derive results throughout this work.
**B.2. DATA COLLECTION & ANALYSIS**

**tarfile**: This module is included with the Python language interpreter. Whilst `.tar` provided a convenient container for multiple results logged by each node (i.e. at the end of each trial), this library enabled analysis of the raw data without extracting it to disk. Moreover, the Python `gzip` module allows transparent decompression of `.tar.gz` files.

**multiprocessing**: This module is also included with the Python interpreter. It enables concurrent multiprocessing by encapsulating both *inter-process communication* and *synchronisation* primitives as a set of Python classes. Here it was used for co-ordinated batch processing of results logged by multiple nodes, described below.

**pcs**: The Packet Construction Set (pcs) is an object-oriented Python framework which provides a high-level interface to `pcap`. I have extended it to perform *post-hoc* analysis of ILNP control and data messages: i.e. pcs was used to derive the activation and signalling measurements reported in Chapter 6.

Each node logged its results individually, and certain results – e.g. the peer activation measurements in Fig. 6.5.11 on page 145 – required that data from each node was processed according to the path(s) followed by the ILNP flow under study. So, identifying this order enabled the peer activation analysis work-flow to be *parallelised*, i.e. by using one process to analyse each trial. Approximately 400GB of raw data was collated from the experiments described in Chapters 4-6. Most of this data was collated during the dSBR experiment in Chapter 6; the use of Python `multiprocessing` allowed $\approx 9000$ compressed `.tar` files – each containing `pcap` packet captures – to be analysed in $\approx 12$ hours on an 8-way compute node.

**B.2.1. Data collation.** Throughout all experiments, trial control was scripted using the FreeBSD `/bin/sh` interpreter. This choice was made for both expedience – as it was readily available and did not require additional software packages – and for

---

4https://docs.python.org/2/library/tarfile.html
5https://github.com/gvnn3/PCS
6Due to API limitations, the retrieval of `pcap` data by PCS from `.tar.gz` files requires the use of a discrete Python thread with its own handle to the underlying file.
simplicity, as data collation tasks could be refined on the command line manually, and then automated to derive results. Each node was connected to a testbed controller by an IPv4 management network, which was otherwise segregated from the experiment topology. The controller script was responsible for initiating trials and managing their progress; a separate script was copied to each testbed node which co-ordinated data collation from the tools discussed below.

**iperf**: This tool\(^7\) is widely used in computer networking research. Its main role is to perform TCP/UDP goodput, jitter and sequence error measurements. However, it omits support for ICMPv6. It has been used throughout this work to profile the ILNP code base. Additionally, *iperf* was modified to perform a frequency count of late packets, yielding the displacement results reported in Chapters 4-6.

**bwping**: Although less widely known, [165, Table 2] this tool supports ICMPv6-based measurements. It performs ICMPv6 RTT, goodput and loss measurements using *packet pairs*. Here, it has been modified to report median RTT also, and to output results in Tab Separated Values (TSV) format for convenient collation.

**fping**: This obscure tool probes multiple hosts using ICMPv6 in parallel. Here it has been used to validate ILNP load sharing behaviour in Chapter 4; its existing report capabilities were expedient for this use. It was modified to count ICMPv6 responses with differing source addresses (i.e. locators), and to output results as a histogram – by IPv6 address – in TSV format.

**netstat**: The FreeBSD version of this command was used to collate link counters in each evaluation, and per-protocol statistics as a check on implementation.

**vmstat**: This FreeBSD diagnostic command reports system-level statistics, specifically I/O counters, interrupts and memory allocation statistics. Here it has been used to profile the memory overhead of ILNP and IPv6.

\(^7\)https://iperf.fr/
tcpdump: This is a command line interface to the widely used `pcap`\textsuperscript{8} packet capture library. It was extended to capture ILNP data and control traffic, in order to obtain the peer activation and failover results in Chapter 6.

The per-node trial script was also responsible for logging trial results back to the controller. The `.tar` file format was chosen for the same reasons as `/bin/sh` – it is readily available in FreeBSD. Whilst this was appropriate for most of the evaluations in this work, the use of a scheme such as Hierarchical Data Format\textsuperscript{9} (HDF) may be more appropriate for metrics requiring millions of discrete samples, e.g. post-hoc analysis of one-way packet delay (not performed in this work).

**B.2.2. iperf measurement details.** Packet reordering behaviour can potentially bias `iperf` trials; periodic reports are also affected. As this behaviour is not well documented, this section summarises some details of its algorithm. The error count includes late arrivals (which are possible – and likely – in multipath configurations) and duplicates\textsuperscript{10} (which have not been observed in this work). It is known that if the final report packet from the source arrives early (i.e. the packet containing the sender’s totals), then the error count will include packets which may have arrived after the final report.

In `iperf-2.0.5`, the roll-over of the error counter to the out-of-order counter takes place at line 792 of file `Reporter.C`. The C structure `ReporterData` contains rolling sums for the flow, whereas `TransferInfo` contains both the periodic and final report statistics. The C function `reporter_condprintstats()` is responsible for rolling over the `TransferInfo` counters between periodic reports. However, this roll-over takes place before the RTP jitter algorithm is applied to packet sequence numbers [166]. So, a packet considered late may affect jitter in the next reporting period. In practice, this has little effect due to the smoothing parameters applied by the jitter algorithm.

\textsuperscript{8}http://www.tcpdump.org/
\textsuperscript{9}http://www.hdfgroup.org/
\textsuperscript{10}In Chapter 6, packets are not duplicated outside of the distributed SBR, because the hand-off process over the multicast inter-router link is strictly controlled to prevent amplification.
if this_seqno != prev_seqno + 1:
    if this_seqno < prev_seqno + 1:
        # late arrival
        out_of_order := out_of_order + 1
    else:
        # possible early arrival
        error := this_seqno - prev_seqno - 1
else
    accept packet

Figure B.2.1. Pseudo-code for iperf reordering and error totals, as implemented at line 750 of file Reporter.C in iperf-2.0.5.

B.2.3. On bwping measurements & error. As explained in Section 3.8.3.3 on page 49, the implementation of packet pacing in bwping has a subtle error – due to its use of integer arithmetic – which causes it to underfill the transmission link by up to $\approx 9\%$ at load rates of 10Mbit/s. By contrast, iperf uses floating point arithmetic in its implementation of packet pacing; it does not appear to be affected by this issue.

In the source code\textsuperscript{11} for bwping, the variable integral_error\textsuperscript{12} contains a cumulative sum of the error due to integer underflow, and the pseudo-error due to integer divisions which have produced non-zero remainders. The transmission rate is adjusted based on the results of these division operations. However, the number of packets in a burst – denoted by pktburst – is never adjusted to account for rounding errors in the calculation of the transmission rate. This leads to the undesired side-effect of bwping sending less packets than are required to meet the requested load rate in affected cases.

B.2.4. Other differences between bwping & iperf. The key difference between these tools is that they differ in how trial duration is specified. For instance, whereas both tools accept a target load rate parameter, bwping will only accept a total volume

\textsuperscript{11}Lines 364–455 of: https://sourceforge.net/p/bwping/code/HEAD/tree/trunk/bwping/bwping6.c

\textsuperscript{12}This name does not imply the use of advanced flow control logic, e.g. in the form of a proportional–integral–derivative (PID) control loop. Packets are transmitted by bwping in linear bursts only.
parameter – and not a trial duration parameter, unlike iperf. So, during each measurement trial, the experiment control script must calculate the total payload volume from the factors for each trial group before bwping can be correctly invoked.

Moreover, both tools have significant differences in their I/O & timing behaviour: iperf uses multiple POSIX threads, whereas bwping is implemented as a single threaded process, using synchronous I/O multiplexing – in the form of the select() system call – to process receives, sends and timing control within the same thread. This structural difference also accounts for differences in their respective implementations of packet pacing described above.

B.3. Evaluation environment

The use of a hardware testbed is desirable to provide a controlled environment for experiments. Measurements performed with VMs may be biased or otherwise limited due to causes beyond the researcher’s direct control (e.g. scheduler jitter, memory pressure and other systems-level performance constraints). Additionally, whilst paging to disk was disabled throughout the testbed, processes were not assigned real-time scheduling priority – in order to reflect scheduler behaviour in typical use. The testbed environment was also constructed to be operating system agnostic – e.g. to enable future studies involving Linux and other networking stacks. So, Cobbler\textsuperscript{13} was used to install the FreeBSD system images over the network. By using pc-sysinstall\textsuperscript{14} in MFSbsd\textsuperscript{14} loaded from the SYSLINUX memdisk\textsuperscript{15} driver, one can eliminate the requirement for FreeBSD-specific boot infrastructure\textsuperscript{[168, 169]}. The tunables in Table B.3 were also needed here to prevent boot failure due to network buffer over-allocation, as the networking hardware used defaults to multi-queue operation.

B.3.1. Hardware configuration. Table B.1 contains details of the systems used to perform the evaluations in Chapters 4-6. All nodes used hardware class A, with the exception of nodes ark and bay, which used hardware class B; and ilnp-ra, which was implemented within a virtual machine hosted on hardware class

\textsuperscript{13}http://www.cobblerd.org/
\textsuperscript{14}http://mfsbsd.vx.sk/
\textsuperscript{15}http://www.syslinux.org/wiki/index.php/MEMDISK
200

B. EXPERIMENTAL APPARATUS

<table>
<thead>
<tr>
<th>Hardware class</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server model</td>
<td>Cisco UCS C220-M3</td>
<td>Gateway GR380-F1</td>
</tr>
<tr>
<td>CPU model</td>
<td>Intel Xeon E5-2630</td>
<td>Intel Xeon E5520</td>
</tr>
<tr>
<td>Microarchitecture</td>
<td>Intel “Sandy Bridge”</td>
<td>Intel “Nehalem”</td>
</tr>
<tr>
<td>CPU clock</td>
<td>2.3 GHz</td>
<td>2.26 GHz</td>
</tr>
<tr>
<td>Processors installed</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Physical cores</td>
<td>6 per package</td>
<td>4 per package</td>
</tr>
<tr>
<td>Logical threads</td>
<td>2 per core</td>
<td>2 per core</td>
</tr>
<tr>
<td>Memory installed</td>
<td>8 GiB DDR3</td>
<td>24 GiB DDR3</td>
</tr>
<tr>
<td>Memory clock</td>
<td>1600 MHz</td>
<td>1066 MHz</td>
</tr>
<tr>
<td>I/O chip family</td>
<td>Intel C600</td>
<td>Intel 5520 &amp; ICH10R</td>
</tr>
</tbody>
</table>

TABLE B.1. Evaluation environment, system hardware specifications

Figure B.3.1. Photographs of testbed systems. Top right: Extreme Summit X450a-48t switch.

B. Photographs of these systems are in Fig. B.3.1. All trials were logged directly to RAM; so, storage technology is not described here.

B.3.2. Network configuration. Each node used Intel I350 network adapters connected by PCI-Express, with a minimum of 6 ports on each node; this includes ports integrated on the system board. Nodes were interconnected by an Extreme Networks Summit X450a-48t switch. Discrete port-based VLANs were used to segregate control (i.e. IPv4, and IEEE 1588 Annex F packets) and data (i.e. IPv6 and ILNPv6 packets) network traffic respectively. This switch supports only store-and-forward (as opposed to cut-through) switching.

Whilst the I350 is capable of multi-queue operation, these adapters were configured for “elastic” single queue operation with interrupt coalescing. Here, I use the
“elastic” to emphasise that one cannot simulate an unbounded queue with hardware. So, system tunables are set to allow the length of each transmit and receive queue to grow to the limits which are supported by the hardware. The igb(4) driver settings used to achieve this are listed in Table B.2, although some are not documented in the manual page [170].

The network stack settings listed in Table B.3 remove the limits on ICMPv6 responses, and increase the maximum permissible socket buffer size. These changes were necessary for the RTT studies in Chapters 4-6. Moreover, interrupt coalescing acts as a notch or band-stop filter applied to queue service time [147, Fig. 2]. In the testbed configuration below, its effective frequency is 96KHz.

<table>
<thead>
<tr>
<th>Tunable name</th>
<th>Default</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>hw.igb.buf_ring_size</td>
<td>4096</td>
<td>16384</td>
</tr>
<tr>
<td>hw.igb.enable_aim</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>hw.igb.max_interrupt_rate</td>
<td>8000</td>
<td>96000</td>
</tr>
<tr>
<td>hw.igb.num_queues</td>
<td>&lt;dynamic&gt;</td>
<td>1</td>
</tr>
<tr>
<td>hw.igb.rx_process_limit</td>
<td>100</td>
<td>4096</td>
</tr>
<tr>
<td>hw.igb.rxd</td>
<td>256</td>
<td>4096</td>
</tr>
<tr>
<td>hw.igb.txd</td>
<td>256</td>
<td>4096</td>
</tr>
</tbody>
</table>

Table B.2. Evaluation environment, Intel I350 network adapter settings for “elastic” single queue operation, with interrupt coalescing

<table>
<thead>
<tr>
<th>Tunable name</th>
<th>Default</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>kern.ipc.maxsockbuf</td>
<td>262144</td>
<td>2097152</td>
</tr>
<tr>
<td>kern.ipc.nmbclusters</td>
<td>1024 + maxusers * 64</td>
<td>262144</td>
</tr>
<tr>
<td>net.inet6.icmp6.errppslimit</td>
<td>100</td>
<td>-1</td>
</tr>
<tr>
<td>net.link.ifqmaxlen</td>
<td>50</td>
<td>8192</td>
</tr>
</tbody>
</table>

Table B.3. Evaluation environment, FreeBSD networking stack configuration

B.3.3. Local timekeeping. Each node was configured to use a kernel timing frequency of 1KHz using the CPU timestamp counter (TSC) as its local time source, by applying the settings in Table B.4. The choice of the TSC was made to obtain
high precision\textsuperscript{16} – although this comes at the expense of more timer interrupts for the operating system. However, this is mitigated by the lower read latency which the TSC offers over the traditional clock source on x86 systems – i.e. the Programmable Interrupt Controller – as the TSC register is local to the CPU.

Table B.5 shows the estimated clock precision for each hardware type using the \texttt{benchtime}\textsuperscript{17} program. These results indicate the measurement overhead incurred by \texttt{iperf} and \texttt{bwping}, which measure elapsed time using the \texttt{gettimeofday()} API \cite{171}. The \texttt{benchtime} program computes the mean difference over 2 million samples where the POSIX standard \texttt{clock_gettime()} function has returned an updated value \cite{172, 173}. In FreeBSD, \texttt{gettimeofday()} uses the same underlying timing code – so, whilst these results are not absolute, the additional cost of using a different timing API is likely to be $O(k)$ constant.

<table>
<thead>
<tr>
<th>Tunable name</th>
<th>Default</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>kern.hz</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>kern.timecounter.hardware</td>
<td>n/a</td>
<td>TSC</td>
</tr>
<tr>
<td>kern.timecounter.smp_tsc</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td>kern.timecounter.tick &lt;hardware-dependent&gt;</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

\textbf{Table B.4.} Evaluation environment, FreeBSD timing configuration

<table>
<thead>
<tr>
<th>Hardware type</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base units (ns)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Resolution (ns)</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Precision, mean (ns)</td>
<td>156</td>
<td>172</td>
</tr>
<tr>
<td>Precision, minimum (ns)</td>
<td>151</td>
<td>167</td>
</tr>
<tr>
<td>Precision, maximum (ns)</td>
<td>1500</td>
<td>1231</td>
</tr>
<tr>
<td>Precision, standard deviation (ns)</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

\textbf{Table B.5.} Evaluation environment, measured precision of \texttt{clock_gettime()} over 2 million samples. The resolution (in seconds) is computed from \texttt{kern.timecounter.tick / kern.hz} (in B.4). All values in this table are reported in nanoseconds.

\textsuperscript{16}The hardware in the evaluation environment is contemporary Intel-based, and is therefore unaffected by issues historically seen with the TSC; e.g. the lack of synchronisation between CPU cores, or between changes in system power state.

\textsuperscript{17}http://freebsd.markmail.org/message/thajlfkzyzp2rmj4d
B.3.4. Network timekeeping. The peer activation & timeout delay studies in Chapter 6 are based on measurements of unidirectional packet delay. For accurate measurement, wall clock time at each node must be synchronised to a central time source. The system clock on each node was accurate to within $\pm 20\mu s$ of the (virtualised) master clock, which was hosted in an NTP-synchronised hypervisor [174]. This was achieved by using a software\textsuperscript{18} implementation\textsuperscript{19} of IEEE 1588 in P2P delay mode [175]. The choice of IEEE 1588 over NTP was made for the following reasons:

- It permits strict segregation of synchronisation from data, by using the Ethernet multicast encapsulation described in [175, Annex F] in preference to UDP/IP.
- A high precision clock source does not need to be directly connected to each node (e.g. using a serial interface), as is often necessary with NTP to achieve highly accurate synchronisation.

However, there is an important difference between ntpd and ptpd\textsuperscript{2} where nodes are frequently rebooted, as is the case within a testbed environment. The frequency offset, as measured against an external reference clock, is unique to each node and its operating characteristics; ambient temperature and manufacturing quality of the system oscillator will influence its measured value. To maintain accuracy, ntpd will measure the frequency offset of the local clock [174, Sec. 9] and store its value persistently between each system boot. By contrast, whilst the clock servo algorithm in ptpd\textsuperscript{2} uses the same kernel time discipline [176] to steer the frequency offset (perhaps more aggressively than ntpd), it does not persist the offset when a node is rebooted. So, to maintain accurate timekeeping, the frequency offset was measured by ntpd for 24 hours upon node installation, and loaded by an rc script before each experiment trial.

\textsuperscript{18}Whilst the use of hardware-based IEEE 1588 timestamping support would have provided better accuracy, the specific hardware driver support required is unavailable in FreeBSD 8.x.

\textsuperscript{19}https://code.google.com/p/ptpv2d/
B.4. FreeBSD IPv6 RA behaviour

In FreeBSD, IPv6 Router Advertisement (RA) service is provided by the `rtadvd` system process. It follows existing recommendations regarding how to respond to unicast Router Solicitations (RS) [123, Sec. 6.2.6]. Response to unicast RS sent by end-hosts is controlled by two hard-coded parameters: `MAX_RA_DELAY_TIME` and `MIN_DELAY_BETWEEN_RAS`. The first parameter is used to delay the router’s unicast RA response by a uniformly random interval up to 500000μs. The second parameter controls the slewing of RS response as follows: if a response to a previous unicast RS message is already scheduled to be transmitted within 3 seconds, the periodic timer which is used to schedule unsolicited multicast RA transmission will not be reset. So, FreeBSD will preferentially respond to unicast RS with a multicast RA in the first instance, and will respond with unicast RA opportunistically.

B.5. ILNP Experiment configuration

Locator precedence is not native to the IPv6 stack, and is retrieved from the policy table when the ILNP module is notified of new IPv6 prefixes. Local precedence values are configured within the `ilnp6locctl` policy table described in Subsec. B.5.1; the `/etc/hosts` database is used only to retrieve remote locator precedence values during flow initiation, i.e. before Locator Update (LU) messages have been received from a remote endpoint. At each end host, two interfaces are active simultaneously and have Locator values with associated preference values. These values are initially learned from the `ilnp6locctl` policy table (and the `/etc/hosts` database), and can be updated by subsequent LU messages.

B.5.1. ILNP Policy Configuration. The ILNP policy table is loaded from `/etc/ilnp6locctl.conf` during system boot. The syntax of this file is shown in Fig.B.5.1; comments begin with a hash (#) character and are ignored. An additional system command `/usr/sbin/ilnp6locctl` command is provided to manage the policy table. The boot-time\textsuperscript{20} script [177] which invokes this command has a dependency on the `network_ipv6` configuration phase. This ensures that the ILNP

\textsuperscript{20} The ILNP code base described in this dissertation requires that a node is rebooted when policy configuration is changed.
Figure B.5.1. Example configuration for distributed Locator Rewriting at an ILNP Site Border Router (SBR). The policy entry for 2001:db8:1000:: is configured on the site internal LAN, and specifies a rewrite target (group 2) for packet volume balancing. Locator 2001:db8:cede:: is part of this group and contains a hint (flags 1) that it is a peer locator – i.e. that it is directly configured on a peer SBR.

policy table has been loaded before any learned IPv6 address configuration (e.g. from IPv6 Router Advertisements) has been applied to the networking stack.

**scope:** This field is reserved and must be set to 0.

**advice:** This field specifies which locator selection policy is associated with a given locator. It contains a text string whose value may be either none (i.e. provide failover between available locators), or v1b (i.e. perform volume balancing and failover between locators in a rewrite group). All other values are reserved. The default value is none.

**label:** This field specifies the rewrite group to which a locator belongs. It contains an integer value (-1 to 8). The default value is -1 which specifies that a locator does not belong to any group.

**rewrite:** This field specifies a target rewrite group for the locator. It contains an integer value (-1 to 8) which must correspond to an existing label value. The presence of this field is optional; the default value is -1 which specifies that the locator will not be rewritten.

**flags:** This field indicates that certain flags apply to the locator. It contains an unsigned decimal integer value (0 to 255). Only one flag value is defined: 1 (peer locator), which provides a hint to the locator selection algorithm that the given locator may not be available to the local host (i.e. it may physically reside elsewhere on a peer SBR). The presence of this field is optional; the default value is 0 which specifies an empty value.
loopback = "lo0"
mgmt = "net0"
local = "net1"
localnet = "fd00:bead::/64"
link_a = "net2"
peer_a = "2001:db8:cafe::2"
global_a = "2001:db8:cafe:b0de::/64"
link_b = "net3"
peer_b = "2001:db8:d00d::2"
global_b = "2001:db8:d00d:b0de::/64"
set limit { states 10, frags 10, src-nodes 10, tables 10, table-entries 10 }
set timeout interval 5
set ruleset-optimization none
set loginterface net1
set skip on { $loopback, $mgmt }
no scrub in all
no scrub out all
binat pass on $link_a inet6 from $localnet to any -> $global_a
binat pass on $link_b inet6 from $localnet to any -> $global_b
pass in quick all no state
pass out quick all no state

Figure B.5.2. Example pf configuration used in Chapter 5 to demonstrate NPTV6 operation. This listing is abridged from /etc/pf.conf.

B.5.2. Example pf configuration. Fig. B.5.2 contains an abridged listing of the /etc/pf.conf configuration file used at node emu in Fig. 5.3.1 on page 102. The configuration used by node jay followed the same syntax, modulo changes to its site (locator) prefixes.

B.5.3. ILNP sysctl MIB variables. The configuration of ILNP at each node introduces additional system variables. These may be set or retrieved using the standard sysctl system command [178]. Each sysctl MIB variable resides under the net.inet6.ilnp6 OID tree. Locator rewriting and forwarding must be explicitly enabled at each node by setting the relevant sysctl MIB variables.

attach_syncif: This variable specifies that the ilnpsync pseudo-interface should be attached on boot. It contains an integer value (0 or 1). (unused)

beat: This variable sets the period between dSBR heartbeat transmissions from the local node in microseconds. It contains an integer value (0 or 500 to
5000000), i.e. between 500 microseconds and 5 seconds. If set to 0, all dSBR heartbeat processing is halted. The default value is 10000 (10 milliseconds).

**rebalance**: This variable controls the behaviour of the volume balancer when a dSBR locator claim is advertised or withdrawn (e.g. when a peer dSBR router is detected as coming up or going down). It contains an integer value (0, 1 or 2). If set to 0, rebalancing is disabled. If set to 1, rebalancing takes place only when a dSBR locator claim is withdrawn. If set to 2, rebalancing will take place both when dSBR locator claims are advertised and withdrawn. The default value is 2. *(See note below)*

**syncif**: This variable specifies the name of a network interface (as a UTF-8 encoded string) where the dSBR synchronization protocol will be bound. Changes are applied immediately when the variable is set (i.e. any previous interface binding will be overwritten). The default value is "" (the empty string).

**synckey**: This variable contains the value of a 160-bit pre-shared key for HMAC-SHA1 authentication between dSBR nodes (as a 42 character UTF-8 encoded string of C-style hexadecimal digits, i.e. prefixed with 0x). Changes are applied immediately when the variable is set. The default value is "" (the empty string).

*Note: Rebalancing of load sharing at dSBR nodes is implemented by setting the output packet count for each locator within a target rewrite group to 0. The volume balancing algorithm will act on the relative weights (inverse of precedence) assigned to each locator within policy.*
# Acronyms & Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABI</td>
<td>Application Binary Interface</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ARIN</td>
<td>American Registry for Internet Numbers</td>
</tr>
<tr>
<td>ARP</td>
<td>Address Resolution Protocol</td>
</tr>
<tr>
<td>ARPA</td>
<td>Advanced Research Projects Agency</td>
</tr>
<tr>
<td>AS</td>
<td>Autonomous System</td>
</tr>
<tr>
<td>ATM</td>
<td>Asynchronous Transfer Mode</td>
</tr>
<tr>
<td>BFD</td>
<td>Bi-directional Forwarding Detection</td>
</tr>
<tr>
<td>BGP</td>
<td>Border Gateway Protocol</td>
</tr>
<tr>
<td>BSD</td>
<td>Berkeley Software Distribution</td>
</tr>
<tr>
<td>CASE</td>
<td>Computer Aided Software Engineering</td>
</tr>
<tr>
<td>CGA</td>
<td>Cryptographically Generated Address</td>
</tr>
<tr>
<td>CGN</td>
<td>Carrier Grade NAT</td>
</tr>
<tr>
<td>CIDR</td>
<td>Classless Inter-Domain Routing</td>
</tr>
<tr>
<td>CLA</td>
<td>Cached Legacy Addresses</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DAD</td>
<td>Duplicate Address Detection</td>
</tr>
<tr>
<td>DDNS</td>
<td>Dynamic DNS</td>
</tr>
<tr>
<td>DFZ</td>
<td>Default-Free Zone</td>
</tr>
<tr>
<td>DHCP</td>
<td>Dynamic Host Configuration Protocol</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>DSBR</td>
<td>Distributed SBR</td>
</tr>
<tr>
<td>DSL</td>
<td>Digital Subscriber Line</td>
</tr>
<tr>
<td>FIB</td>
<td>Forwarding Information Base</td>
</tr>
<tr>
<td>Acronym</td>
<td>Full Form</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>FQDN</td>
<td>Fully Qualified Domain Name</td>
</tr>
<tr>
<td>GSE</td>
<td>Global, Site &amp; End-system Addressing</td>
</tr>
<tr>
<td>HBA</td>
<td>Hash Based Address</td>
</tr>
<tr>
<td>HIP</td>
<td>Host Identity Protocol</td>
</tr>
<tr>
<td>HMAC</td>
<td>Hashed Message Authentication Code</td>
</tr>
<tr>
<td>I-D</td>
<td>Internet Draft</td>
</tr>
<tr>
<td>I-LV</td>
<td>Identifier-Locator Vector</td>
</tr>
<tr>
<td>IAD</td>
<td>Internet Access Device</td>
</tr>
<tr>
<td>IC</td>
<td>Interrupt Coalescing</td>
</tr>
<tr>
<td>ICMP</td>
<td>Internet Control Message Protocol</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical &amp; Electronic Engineers</td>
</tr>
<tr>
<td>IEN</td>
<td>Internet Engineering Note</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IFID</td>
<td>IPv6 Interface Identifier</td>
</tr>
<tr>
<td>IID</td>
<td>Interface Identifier</td>
</tr>
<tr>
<td>IIP</td>
<td>Initial Ingress Peer</td>
</tr>
<tr>
<td>ILCC</td>
<td>Identifier-Locator Correspondent Cache</td>
</tr>
<tr>
<td>ILNP</td>
<td>Identifier-Locator Networking Protocol</td>
</tr>
<tr>
<td>ILSA</td>
<td>Identifier-Locator Split Architecture</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IPSEC</td>
<td>IP Security Extensions</td>
</tr>
<tr>
<td>IRL</td>
<td>Inter-Router Link</td>
</tr>
<tr>
<td>IRTF</td>
<td>Internet Research Task Force</td>
</tr>
<tr>
<td>IS-IS</td>
<td>Intermediate System/Intermediate System</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standards Organisation</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunications Union</td>
</tr>
<tr>
<td>L64</td>
<td>Locator Value (64-bits)</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>LIR</td>
<td>Local Internet Registrar</td>
</tr>
<tr>
<td>LISP</td>
<td>Locator-Identifier Separation Protocol</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>LR</td>
<td>Locator Rewriting</td>
</tr>
<tr>
<td>LS</td>
<td>Locator Selection</td>
</tr>
<tr>
<td>LSB</td>
<td>Least Significant Bit</td>
</tr>
<tr>
<td>LU</td>
<td>Locator Update</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Control</td>
</tr>
<tr>
<td>MD5</td>
<td>Message Digest Algorithm 5</td>
</tr>
<tr>
<td>MHMP</td>
<td>Multihomed/Multi-Prefix</td>
</tr>
<tr>
<td>MIB</td>
<td>Management Information Base</td>
</tr>
<tr>
<td>MP-TCP</td>
<td>Multipath TCP</td>
</tr>
<tr>
<td>MSB</td>
<td>Most Significant Bit</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>ND</td>
<td>Neighbour Discovery</td>
</tr>
<tr>
<td>NFV</td>
<td>Network Function Virtualisation</td>
</tr>
<tr>
<td>NID</td>
<td>Node Identifier</td>
</tr>
<tr>
<td>NPTv6</td>
<td>Network Prefix Translation for IPv6</td>
</tr>
<tr>
<td>NTP</td>
<td>Network Time Protocol</td>
</tr>
<tr>
<td>OO</td>
<td>Object Oriented</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnect</td>
</tr>
<tr>
<td>OSPF</td>
<td>Open Shortest Path First</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>PA</td>
<td>Provider Aggregate</td>
</tr>
<tr>
<td>PCB</td>
<td>Protocol Control Block</td>
</tr>
<tr>
<td>PCI</td>
<td>Peripheral Component Interconnect</td>
</tr>
<tr>
<td>PCS</td>
<td>Packet Construction Set</td>
</tr>
<tr>
<td>PI</td>
<td>Provider Independent</td>
</tr>
<tr>
<td>PPS</td>
<td>Packets per Second</td>
</tr>
<tr>
<td>PTP</td>
<td>Precision Time Protocol</td>
</tr>
<tr>
<td>RA</td>
<td>Router Advertisement</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RFC</td>
<td>Request For Comments</td>
</tr>
<tr>
<td>RIB</td>
<td>Routing Information Base</td>
</tr>
<tr>
<td>Acronym</td>
<td>Abbreviation</td>
</tr>
<tr>
<td>---------</td>
<td>--------------</td>
</tr>
<tr>
<td>RIPE</td>
<td>Réseaux IP Européens</td>
</tr>
<tr>
<td>RIR</td>
<td>Regional Internet Registrar</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>RS</td>
<td>Router Solicitation</td>
</tr>
<tr>
<td>RSVP</td>
<td>Resource reSerVation Protocol</td>
</tr>
<tr>
<td>RTP</td>
<td>Real Time Protocol</td>
</tr>
<tr>
<td>RTT</td>
<td>Round Trip Time</td>
</tr>
<tr>
<td>SBR</td>
<td>Site Border Router</td>
</tr>
<tr>
<td>SCSP</td>
<td>Server Cache Synchronisation Protocol</td>
</tr>
<tr>
<td>SCTP</td>
<td>Stream Control Transmission Protocol</td>
</tr>
<tr>
<td>SDN</td>
<td>Software Defined Networking</td>
</tr>
<tr>
<td>SHA-1</td>
<td>Secure Hash Algorithm 1</td>
</tr>
<tr>
<td>SHIM6</td>
<td>Level 3 Multihoming Shim Protocol for IPv6</td>
</tr>
<tr>
<td>SLAAC</td>
<td>StateLess Address Auto-Configuration</td>
</tr>
<tr>
<td>SNMP</td>
<td>Simple Network Management Protocol</td>
</tr>
<tr>
<td>SRAM</td>
<td>Static RAM</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
</tr>
<tr>
<td>StrongES</td>
<td>Strong End Station</td>
</tr>
<tr>
<td>TCAM</td>
<td>Ternary Content Addressable Memory</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TE</td>
<td>Traffic Engineering</td>
</tr>
<tr>
<td>TSC</td>
<td>Timestamp Counter</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>ULA</td>
<td>Unique Local Address</td>
</tr>
<tr>
<td>UMTS</td>
<td>Universal Mobile Telecommunications System</td>
</tr>
<tr>
<td>USM</td>
<td>User-based Security Model</td>
</tr>
<tr>
<td>UTF-8</td>
<td>Unicode Translation Format (8-bits)</td>
</tr>
<tr>
<td>VLAN</td>
<td>Virtual LAN</td>
</tr>
<tr>
<td>VLSM</td>
<td>Variable Length Subnet Masking</td>
</tr>
</tbody>
</table>
Bibliography


man.cgi?query=rtalloc&sektion=9&manpath=FreeBSD+8.4-RELEASE.


[175] IEEE. IEEE Standard for a Precision Clock Synchronization Protocol for Net-
2008.


[177] The FreeBSD Project. rc(8). In FreeBSD System Manager’s Manual. The
FreeBSD Project, November 2009. URL http://www.freebsd.org/cgi/
man.cgi?query=rc&apropos=0&sektion=8&manpath=FreeBSD+8.4-RELEASE&
arch=default&format=html.

[178] The FreeBSD Project. sysctl(8). In FreeBSD System Manager’s Manual. The
cgi?query=sysctl&apropos=0&sektion=9&manpath=FreeBSD+8.4-RELEASE&
arch=default&format=html.