An Indirection Architecture for the Internet

RICHARD GOLD
Dissertation presented at Uppsala University to be publicly examined in MIC Aula, Pollacksbacken, Lägerhyddsvägen 2, Uppsala, Friday, December 16, 2005 at 13:15 for the degree of Doctor of Philosophy. The examination will be conducted in English.

Abstract


We present an indirection architecture for the Internet called SelNet. SelNet provides a uniform indirection mechanism for controlling the route that packets take through the network and which functions are invoked to process these packets. In the current Internet, at least for the majority of users, there is only one way that a packet can go and that is to the default route. Whilst this is sufficient for many applications, numerous applications have arisen which require alternative routes or processing to be present not only at the application-layer of the protocol stack, but at the network-layer itself. Solutions to such scenarios attempt to place an indirection point between the communicating end-systems either with a middlebox (such as a proxy) or by altering one or more of the Internet's naming systems. However these approaches lead to an application-specific network, which is against the Internet's design goals. We argue for a uniform approach to indirection instead of building multiple, partially overlapping structures as is the current trend. SelNet differs from existing indirection approaches in that it is function-orientated, rather than node-orientated and that it provides an explicit, controllable resolution mechanism for resolving host names and services. The motivation behind our approach is to create efficient indirection structures for supporting new applications which have indirection requirements. We present a detailed design and specification of SelNet. We then go on to describe implementation work with the LUNAR ad-hoc routing protocol and the Janus middleware for accessing sensor networks systems. The purpose of this implementation work is to demonstrate the feasibility of SelNet and its ability to reach its goals.

Keywords: Indirection, Network Architecture, Design, Extensibility

Richard Gold, Department of Information Technology, Box 337, Uppsala University, SE-75105 Uppsala, Sweden

© Richard Gold 2005

ISSN 1651-6214
ISBN 91-554-6419-X
urn:nbn:se:uu:diva-6199 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-6199)
Look, that’s why there’s rules, understand?
So that you think before you break ’em.

Terry Pratchett, “Thief of Time”
Acknowledgments

The PhD process has been a journey for me in many different ways. I have spent the last six years living outside of my home country and have had the privilege of being a guest in both Germany and Sweden. I have learned a lot from this experience both professionally and personally. Along the way towards the PhD I have benefited a great deal from the people around me who I have tried to document below:

Per Gunningberg has been my supervisor during my time at Uppsala University. He has tremendous wisdom and experience, and what’s more: compassion and understanding for the eccentricities of his PhD students! It has been a pleasure to be one of his students. Christian Tschudin has inspired me enormously during the course of my thesis work and has been my main technical collaborator. We have also shared a lot of time outside of work which has always been rewarding for me. Ian Wakeman whose undergraduate course in Distributed Systems convinced me to get involved with computer networking research in 1997 and has since become involved with my research. His understanding of the field has contributed a great deal to my work. Jon Crowcroft hosted me in Cambridge during 2003 and has deeply inspired and informed my work. Our evenings in various pubs in various countries have always been a fascinating experience. Mikael Smirnow who was my boss during my time in Berlin, he always tried to give me the freedom I needed to find my own way.

My time in the Communications Research group has given me not only an excellent working environment but also some great friends. My office-mate, Erik Nordström, has been my partner-in-crime for many of the adventures that took place during this thesis. His unparalleled technical skill and analytical thinking has helped me hugely. Olof Rensfelt, whose skill with a C compiler is matched only by his skill in the kitchen. I have been the lucky beneficiary of these and much more. Lars-Åke Larzon with his encyclopedic knowledge of networking and pragmatic approach has helped me out of many a tight spot. David Lundberg has always been willing to lend me his time and abilities.

My other colleagues in the Communications Research group have provided a stimulating environment during my time there: Henrik Lundgren, Arnold Pears, Thabotharan Kathiravelu, Laura Feeney, Oskar Wibling and Christian Rohner. I have also had the privilege of working with some great students: An-
dreas Westling, Fredrik Bjurefors, Henrik Olsson, Sergio Angel Marti, Mats Uddenfeldt, Björn Ahlström and Per Svensson. I have often visited the Systems Software Engineering & Networking research groups at University College London, especially Cecilia Mascolo and have always had a lot of fun there. The PhD students at Luleå Technical University have been great company: Sara Landström, Pierre Fransson, Johan Nyqvist and Anders Lindgren. My colleagues in the WinterNet project from SICS: Bengt Ahlgren and Adam Dunkels.

Various friends outside of work have helped make my time in Uppsala an enjoyable experience: Agnes Runqvist, Maria & Anna Björnermark, Carola Silverplats, Henrik Nilsson, Karin Pernestål, Virginie Delporte, Björn Uddenfeldt and Sven Westergren.

My friends outside of Sweden: Torsten Ackemann for “serious fun” in Berlin, London and cyberspace. Stefanos Zachariadis for an almost unbelievable amount of hilarious chat sessions that served to lighten the mood when I needed it. Jana Holland who has been my friend for a long time and has always been there when I needed her. Dan Tidhar for unlocking so many of Berlin’s secrets for me and for teaching me German grammar among so many other things. Jan Grote for many adventures and discussions. Mattis Fjällström for being the "loyal opposition". Elisa Sammelselg for showing me real hospitality.

My sister, Andrea, has been a staunch supporter and my biggest fan. She has always helped to pick me up when things got tough. My parents, Vivian and Trevor, have supported me throughout the years and helped me out in so many different ways. They have tried to let me find my own way and make my own mistakes. I thank them deeply for that.

Finally, a special thanks must go to Sandra Hellstrand. Without her love, support, intelligence and understanding this thesis would not have happened.

To all those I have forgotten:

Your name here: x_____________________________x

The work in this thesis has been supported by generous funding from both SSF (Stiftelsen för Strategisk Forskning) through the WinterNet project and Vinnova through the SCANET project. The SSF graduate school PCC++ has partially supported some of the work in this thesis.
List of Included Papers


Reprints for this thesis are made with permission from the publishers.

**Comments on my Participation**

**Paper A** : I was the co-author of this paper. I invented and investigated one of the scenarios. I participated in many discussions with Christian Tschudin concerning all of the scenarios and propositions made in the paper.

**Paper B** : I was the lead author of the paper and wrote most of the text. I had frequent discussions with my co-authors which were of enormous help in clarifying my thoughts.
**Paper C**: I was a co-author of this paper and contributed some of the text. I supervised the implementation work at Uppsala. I was also the co-designed of LUNAR together with Christian Tschudin and performed many tests and evaluations of it.

**Paper D**: I was one of the main authors of this paper, I supervised the SelNet implementation work and wrote the majority of the text.

**Paper E**: I was the main author of this paper, supervised the implementation work and wrote nearly all of the text.

**Other Work**

In addition to papers A through E, I have also authored, co-authored and in some cases presented the following papers, posters and demos:

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Deployment</td>
<td>75</td>
</tr>
<tr>
<td>4.2</td>
<td>Resilience</td>
<td>76</td>
</tr>
<tr>
<td>4.3</td>
<td>Security considerations</td>
<td>77</td>
</tr>
<tr>
<td>5</td>
<td>Conclusions and Outlook</td>
<td>78</td>
</tr>
<tr>
<td>8</td>
<td>References</td>
<td>81</td>
</tr>
<tr>
<td>C</td>
<td>Paper C: LUNAR – A Lightweight Underlay Network Ad-hoc Routing</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>Protocol and Implementation</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Introduction</td>
<td>85</td>
</tr>
<tr>
<td>1.1</td>
<td>The LUNAR Approach</td>
<td>86</td>
</tr>
<tr>
<td>1.2</td>
<td>Ad hoc Horizon</td>
<td>86</td>
</tr>
<tr>
<td>1.3</td>
<td>Underlay Routing and Internet Access</td>
<td>86</td>
</tr>
<tr>
<td>2</td>
<td>Related Work</td>
<td>87</td>
</tr>
<tr>
<td>3</td>
<td>The LUNAR Protocol</td>
<td>88</td>
</tr>
<tr>
<td>3.1</td>
<td>SelNet</td>
<td>88</td>
</tr>
<tr>
<td>3.2</td>
<td>SelNet Control Messages (XRP)</td>
<td>88</td>
</tr>
<tr>
<td>3.3</td>
<td>SelNet Packet Format</td>
<td>90</td>
</tr>
<tr>
<td>3.4</td>
<td>LUNAR Protocol Logic</td>
<td>90</td>
</tr>
<tr>
<td>3.5</td>
<td>Intercepting DHCP for Address Self-Configuration</td>
<td>91</td>
</tr>
<tr>
<td>4</td>
<td>Implementation</td>
<td>91</td>
</tr>
<tr>
<td>4.1</td>
<td>Measurements</td>
<td>92</td>
</tr>
<tr>
<td>4.2</td>
<td>A Tiny Multihop Ad Hoc Access Point</td>
<td>93</td>
</tr>
<tr>
<td>4.3</td>
<td>LUNAR for Embedded Systems</td>
<td>93</td>
</tr>
<tr>
<td>4.4</td>
<td>LUNAR for Bluetooth Scatternets</td>
<td>94</td>
</tr>
<tr>
<td>4.5</td>
<td>LUNAR for Windows</td>
<td>94</td>
</tr>
<tr>
<td>5</td>
<td>Discussion</td>
<td>95</td>
</tr>
<tr>
<td>5.1</td>
<td>Forced Path Re-establishment</td>
<td>95</td>
</tr>
<tr>
<td>5.2</td>
<td>Formal Validation</td>
<td>95</td>
</tr>
<tr>
<td>6</td>
<td>Conclusions</td>
<td>96</td>
</tr>
<tr>
<td>9</td>
<td>References</td>
<td>97</td>
</tr>
<tr>
<td>1</td>
<td>Introduction</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>Architecture</td>
<td>103</td>
</tr>
<tr>
<td>2.1</td>
<td>Supporting Multiple Sensor Network Types</td>
<td>105</td>
</tr>
<tr>
<td>2.2</td>
<td>Supporting Multiple Access Applications</td>
<td>105</td>
</tr>
<tr>
<td>3</td>
<td>Implementation</td>
<td>106</td>
</tr>
<tr>
<td>3.1</td>
<td>Sensor Network</td>
<td>107</td>
</tr>
<tr>
<td>3.2</td>
<td>Gateway</td>
<td>107</td>
</tr>
<tr>
<td>3.3</td>
<td>XRP Engine</td>
<td>107</td>
</tr>
<tr>
<td>3.4</td>
<td>XRP Agent</td>
<td>108</td>
</tr>
<tr>
<td>3.5</td>
<td>Signaling between Agent and Engine</td>
<td>108</td>
</tr>
<tr>
<td>4</td>
<td>Discussion</td>
<td>110</td>
</tr>
</tbody>
</table>
1. Introduction

The Internet of today is a vast network of inter-connected networks whose original design dates back to a US Department of Defense project in the 1960s called ARPANET [44]. In forty years, the Internet has evolved from the research laboratory to a critical piece of the global communications infrastructure. The Internet is a foundation, supporting the communication of many applications such as the World Wide Web (WWW), Email, instant messaging, multimedia and peer-to-peer file-sharing. These applications are used by billions of people all over the world. The Internet supports these applications by giving them the ability to send packets containing application data to any computer or device which is connected to the Internet, irrespective of the physical location of the end-point. In order to achieve this, the end-points of the Internet must be named.

The Internet has two global name-spaces which it uses to name end-points: DNS names and IP numbers. DNS names are hierarchical and human-readable, for example: www.hrw.org. IP numbers are 32 bit integers, e.g., 209.237.248.126, and are used by the Internet for making forwarding decisions. DNS names are typically resolved to IP numbers, in order for them to be understandable by the Internet. The IP number contains two different concepts:

1. **Identity**: The name of the network interface of a host.
2. **Location**: The name of the location of a host.

This tight binding between the concepts of identity and location in a single entity causes problems for scenarios which contain mobility or middleboxes. In these scenarios identity and location are not as strictly correlated as the IP addressing model dictates. For example, the identity of a mobile host does not change, even though its location will as it moves between different networks. We can further envisage scenarios where we wish to have alternative packet processing semantics than standard Internet packet forwarding. For example, media transcoding or compression.

The motivation of this work is to introduce indirection capabilities into the Internet in order to provide the ability for an application or agent outside the core of the Internet to explicitly control:

---

1A middlebox is a device which interposes itself between two communicating end-points. Examples include firewalls and proxies.
1. The route that packets take between end-points.
2. How packets are processed.

The route chosen today in the Internet is determined by the standardized routing protocols and the way that packets are processed is fixed by the IPv4 forwarding process. These cannot be influenced by the end-points. Furthermore the IPv4 routing and forwarding code is implemented in millions of routers and therefore they will be difficult to change. By introducing an indirection step it is possible for an end-point to control a route and how it is processed whilst still keeping the original Internet protocols.

The contribution of this work is the design and specification of an indirection architecture for the Internet called SelNet. SelNet provides a uniform indirection primitive which can be used for controlling not only which route a packet takes through the network, but also which functions are invoked to process the packet en route. SelNet allows for a high degree of extensibility whilst maintaining backwards compatibility with the existing Internet.

The other alternative to providing some kind of indirection support, of which SelNet offers one technique, is to change the core protocols of the Internet itself. However, as the Internet has grown, it has become increasingly hard to change. On the 1st January 1983, the ARPANET changed its core protocol suite from NCP (Network Control Protocol) to TCP/IP thus marking the beginning of the Internet as we know it today. This was called the “Flag Day” [41]. On this day, it was attempted to convert all machines on the ARPANET from NCP to TCP/IP. There were network outages during that day, but the network was basically functional the next day. The number of nodes that needed to be converted at that time was well under 1,000. Now it is not feasible to have a day of downtime since the Internet is globally so important. Also it would be impossible to co-ordinate the 350,000,000 or so machines on the Internet. The growth of the Internet has therefore led to a corresponding decrease in its extensibility.

The obvious question to ask at this stage is: “why do we need extensibility?”. If the Internet serves our purposes well enough, then the additional overhead incurred by introducing more extensibility into the system is useless. One motivation is the increasing amount of architectural stress inside of the Internet [17]. By this we mean that recent additions to the Internet are breaking or bending the fundamental principles of the Internet. We present one example here, but discuss more examples in section 1.2. One of the fundamental principles is symmetric connectivity: Any host on the Internet should be able to communicate with any other host on the Internet. A typical example of this is the File Transfer Protocol (FTP) protocol [43] where a client connects to a FTP server and then the server connects back to the client to continue the

---

2We use the term “architecture” to refer to the design and set of principles of a system.
transaction. In order for this transaction to function correctly symmetric connectivity has to be present. However, in the Internet of today there are many middleboxes deployed which break this principle of symmetric connectivity, typically in the name of security. Examples of such middleboxes are:

**Network Address Translation (NAT)**: NATs [18] translate between two different IPv4 namespaces. Typically one namespace contains only RFC1918 [42] private addresses and the other namespace is the global Internet. NAT was originally designed to cope with the address space exhaustion of the Internet, but has subsequently been used for security and site isolation purposes.

**Firewalls**: These provide security properties by blocking certain types of traffic that do not conform to security policies. These are typically used at the ingress/egress point of an organization’s network to prevent malicious or unwanted traffic from entering or leaving the network.

**Proxies**: A proxy is a device that “stands for” or “represents” another device. Typically proxies are used to control traffic by enforcing that traffic must flow through a proxy where it can be analyzed or they are used to provide performance enhancements through caching or transcoding.

In the scenario where we wish to make a file transfer, in order for this transfer to work with FTP, the middlebox must explicitly be configured to allow through that particular protocol. This leads to the situation where application-specific knowledge is embedded inside the network, rather than exclusively residing on the end-points. These workarounds to get applications to work more or less correctly shows that the introduction of new services (security, for example) conflict with the principles of symmetric connectivity.

Since alternative connectivity requirements are required for security or administrative concerns, network operators have deployed middleboxes such as NATs, Firewalls and Proxies to provide the desired functionality. As these middleboxes necessarily break the assumption of symmetric connectivity to achieve their goals, applications such as Voice over IP (VoIP), online gaming and peer-to-peer filesharing have problems functioning as they require symmetric connectivity. A network architecture with support for indirection could allow new services to be introduced without breaking symmetric connectivity.

We believe that this approach would allow us to better manage the conflicting concerns between the network’s principles and the applications that use them.

---

3The usage of NAT for security purposes is a controversial point. Many believe that NAT does not provide security. We note here that NAT, especially when combined with port forwarding, provides more fine-grained control over connectivity to hosts. This control can be used to limit connectivity, thereby providing some measure of security.
1.1 Background

In order to understand the difficulties in extending the functionality of the Internet, we review some of the existing attempts. During the 1990s, many mechanisms were proposed by the research community to add or replace functionality to the Internet. For example:

Multicast - 1989 [15]: Group communication for the Internet is still not fully deployed. Whilst multicast support is present in some regions of the Internet, there are still unsolved issues, for example, regarding charging for multicast traffic that crosses provider domains [25].

IPv6 - 1998 [16]: The flagship replacement for IP version 4. The deployment of IPv6 is still active and has been ongoing for since 1999.

DiffServ (QoS) - 1998 [9]: DiffServ is an attempt to bring service differentiation to the Internet. It enjoys some success in closed networks, for example TV-on-demand where one company operates and controls a dedicated piece of infrastructure and then TV-on-demand is allocated a fixed part of that infrastructure’s resources. However it has not been widely deployed in inter-domain scenarios.

These three examples have not been widely deployed due to various difficulties:

1. The requirement that every router in the Internet has to be upgraded
2. The complexity of the protocols themselves
3. The lack of understanding of the economic implications
4. The immense difficulty involved with attempting to change wholesale a critical piece of the global communications infrastructure

As a result of these difficulties, a general frustration grew in the research community concerning these barriers to deployment of new Internet protocols and many looked for alternative ways to get their systems deployed. One approach was quite radical: Active networking [59, 22, 24, 14, 53]. This idea in its purest form makes every router in the Internet a Universal Turing Machine and each packet a Turing Machine. In other words, a packet carries its own code which is executed on a router. This approach is illustrated in figure 1.1. Active networking provides unconstrained extensibility. Each packet could potentially carry its own application, transport and network layer protocols. Active networking could not be widely-deployed in the Internet with current technology due to security and resource consumption issues. It is very hard to protect a host from code with malicious intent since AAA (Authentication, Authorization & Accounting) decisions in general have to be made for each

---

4IPv6 was standardized in 1998 but deployment did not begin until 1999.
piece of code. Resource consumption is also an issue, since without any restrictions it would be trivial for an attacker to send a malicious piece of code to a host or route and perform a resource exhaustion attack. It is also very difficult to protect mobile code against a host with malicious intent since it the host will have a great deal of control over the code executing on it. Whilst there have been many security proposals for active networks [2, 38, 52], there are still too many unanswered questions for them to be accepted in real-world situations. However, active networking as a concept is intriguing. If it were deployed it would allow for an unprecedented degree of customization of computer communication and thus remove the barriers to deployment of new technologies.

**Figure 1.1:** Before shows the classic passive model of protocol processing, after shows the new active model where code is carried in “active” packets. These active packets are then processed by an active packet interpreter which runs on each node of the network.

The architectural principles of indirection have its roots in Active networking. However, indirection is much more constrained and avoids many of the problems of security and resource control by not relinquishing so much control. This is because indirection presents a more controllable technique for extending the functionality of a network. Instead of allowing mobile code which is a major shift in control, indirection allows an agreement between the user requesting indirection and the network which provides it. Indirection provides the means whereby a suite of functions in the network can be used. Instead of just a single set of functionality being available e.g., IPv4 packet forwarding, having some indirection functionality would allow packets to be redirected away from the default processing path and towards a processing path which performs different tasks. Examples include IPv6, media transcoding, compression or customized routing. They are currently difficult to introduce to the Internet due to the difficulty of changing the core protocols of the Internet. How to perform indirection and which functions or tasks should be available in the network are questions which are examined in detail in this work.
1.2 Trends in Extending the Internet

We examine four trends in extending the Internet: methods of deployment, the separation of identity and location, middleboxes and the issues involved in inter-connecting different types of networks. The motivation for choosing these trends is that they expose architectural design issues of the Internet. There are many other issues that affect an architecture such as security and resource allocation. However, we would argue that these are issues that are not specifically related to extending the functionality of the Internet. Although security and resource control are pervasive issues with almost everything to do with the Internet, they are tangential to the central issue of how to extend the functionality of the Internet.

1.2.1 Methods of deploying new protocols

The default mechanism present in the Internet to allow extensibility is the IP options field of the IPv4 packet header [10]. However, IP options are seen as extremely inefficient and are liable to be dropped by many routers [21]. IP options are handled outside of the fast-path of a router and therefore require much more processing. Furthermore the IP options approach does not let us replace the existing IP layer functionality, which is something we would like to do to enable new protocols, e.g., IPv6. Overlay networks [3] are an excellent way of quickly deploying protocols. However, they are not native to the existing infrastructure and only use it as a bridge. They build their own network on top of the Internet by constructing tunnels, which are virtual network links, between end-points. Therefore it is difficult to change the Internet itself with this approach. Overlays are indeed useful for adding functionality to the existing Internet but not necessarily useful for changing it. Consequently they also suffer from brittleness by forcing the lower layers of the protocol stack to remain as they are designed. An overlay also loses locality information that is hidden in the network layer.

1.2.2 Separating Identity and Location

The IPv4 address collapses the two notions of identity and location into an elegant, but sometimes troublesome entity. Problems arise for mobility and naming due to interdependencies between the two concepts and also because one cannot be changed without affecting the other. We examine here the trends in attempting to separate identity and location to allow for mobility and redirecting to alternative servers.

Salzer in [46] makes the following distinctions between names, addresses and routes based on Shoch’s original definitions:
A name identifies what you want.
An address identifies where it is.
A route identifies a way to get there.

For the purposes of the following discussion we make the following additional definitions and refinements:

- A name is semantically meaningful e.g., a DNS name. An entity may have many names.
- An identifier represents a flat, opaque identity e.g., a random bit string.
- A locater is the same as an address. We make this additional definition in order to provide clarity in our explanations.

Mobility

There has been an increase of mobile devices in the shape of mobile phones, laptops and PDAs connecting to the Internet. In the Internet, devices are identified by their IP number. This is a crucial point. By using the definitions mentioned above, we can immediately see the tension in using an IP number as an identifier and a locater. If a mobile device’s IP number changes when the device roams to a new network, it has to all intents and purposes a different identity. This becomes a problem for certain applications such a Voice over IP (VoIP) where a mobile user may roam between multiple networks during the course of a conversation. This is solved with an indirection step in a SIP proxy at the application layer. This is a costly and non-reusable solution. Also, certain security applications such as SSH or IPsec [50] identify a host by its IP number, a source of much frustration for the mobile user that is forced to use another IP number when switching networks. Finally, it also becomes difficult if a mobile device is running a service which others wish to use (for example, sharing a printer or Internet connection at a conference) since the identity of the device is determined by whatever IP number the device may currently have.

Mobile IP [39] solves this problem by providing a mobile node with two IP numbers. One which represents the identity of a node and one which represents the location of a node. Mobile IP has issues regarding inefficiency with triangle routing [60] and header inflation due the extra encapsulation it has to perform in order to handle two IP numbers. Unfortunately the indirection mechanism of Mobile IP is not reusable for other applications such as handling middleboxes or performing customized routing.

Finding some way of separating identity from location is a key problem area of network architecture research to allow for mobility and redirection. In order to do this, many proposals suggest some kind of resolution functionality between the identity and the locater [60, 49, 12, 6]. This means that a resolution is needed each time a host wishes to communicate with another, to ensure that
the address information fresh. Indirection is provided by having the identity-
to-location resolution process interposed into the communication between the
two end-points. Resolution is thereby used as an indirection mechanism.

Extensible Naming and the DNS System

Certain applications such as content delivery networks (CDNs) and load bal-
ancing systems need an indirection step since content both may move and
may be replicated. The operators of these services have begun to code their
requirements into the DNS system since there is no other natural way for these
applications to satisfy their indirection requirements. The Internet uses DNS
names and IP numbers to name its end-points. In order not to break the se-
mantics of DNS, the applications force the clients to make multiple requests
so that it is possible for a CDN to work out the location of a client and redirect
it to the preferred server. However this is an overloading of the DNS sys-
tem which is designed to only perform mappings between DNS names and IP
numbers and not to contain application-specific requirements. Additionally,
many resolution steps have to be made in order for the necessary redirection
to be performed\(^5\). As an example assume that we wish to redirect an HTTP
request to the nearest server: a URL (Uniform Resource Locater) specifies a
host which contains the content that a client wishes to retrieve and this DNS
name is bound to an IP number. However it is not always necessary to go to the
host specified in the URL to retrieve the content if there may be a proxy cache
nearby such as one set up by an ISP or by a dedicated CDN operator such as
Akamai. There is currently no architecturally pure way to do this redirection
to a proxy as the IPv4 Internet does not provide any inherent indirection mech-
anism either through naming or through an explicit indirection layer. Instead
other resolution mechanisms like extending DNS in an application-specific
way are used. Ideally we would like to be able to resolve a URI (Uniform Re-
source Identifier) to a content server which is “nearby” to the client machine.
We would prefer to use a URI rather than a URL since a URI provides a nat-
ural point to separate identity and location. This “nearby” resolution would
enable us to more efficiently use the network’s resources.

We would like to use separate identifiers and locaters at the network layer.
This approach provides indirection by identifiers being interpreted or resolved
or translated to another form to provide a locater. This resolution, if it can be
performed in a extensible way, can therefore be customized to return different
locaters depending on a host’s requirements. For example, we could resolve
an identifier to the “closest” or “best” server depending on our specification.
This would would mean that the difficulties in dealing with mobility, content
naming and middleboxes would be greatly reduced as we would be able to

\(^5\) A brief description of a typical Akamai CDN session can be found here: http://www.cs.
washington.edu/homes/ratul/akamai.html
identify an end-point with an identifier whilst being able to resolve that identifier to an appropriate locater. Appropriate in this context means the current location of a mobile host or a middlebox which contains a cached copy of an item of content.

1.2.3 Middleboxes

The Internet is assumed to be a transparent network which sits between various communicating end-points. Since the original vision was developed, various middleboxes have arisen in the network that provide end-point functionality and therefore are not completely transparent. Middleboxes are effectively attempts to introduce indirection into the Internet.

The end-to-end model of the Internet means that a DNS name or an IP number can only be used to identify a destination, not an intermediary. Therefore we cannot use DNS name such as hrw.org or an IP number such as 199.173.149.140 to refer to anything else other than the actual destination. CDNs circumvent this problem by manipulating DNS as described in section 1.2.2. Since the Internet does not provide any resolution mechanism for mapping between the identity role of an IP number and the locater role, it is not possible for middleboxes to be supported natively by the Internet.

In order to support middleboxes various tricks or hacks are used, for example: address translation, DNS manipulation, transparent proxying and TCP connection splicing. These tricks may cause problems for certain applications. The root of these problems is that middleboxes also work at the transport and application layers while the original design of the Internet architecture assumes that only end-points have these layers. This results in a brittle network since changes in the layers above the network layer will not be handled correctly by NATs or other middleboxes. One example is of such a change is the deployment of new transport protocols such as the Datagram Congestion Control Protocol (DCCP) [32] or the Stream Control Transmission Protocol (SCTP) [48] this causes a problem since most middleboxes can only handle TCP, UCP and ICMP packets. It is often the case that IP packets without recognized transport protocols are discarded by middleboxes also middleboxes can filter out certain packets. NAT is a classic example of this since by default a host behind a NAT box is not globally addressable and packets addressed directly to that host are discarded. This means that the mapping of connections between outside and inside as well as the filtering aspects rapidly become complex. NAT traversal mechanisms are necessarily a hack since there are several different types of NATs [45], each with their own set of peculiarities which the potential NAT traverser has to work around. Many applications such as web browsing, email and instant messaging work fine through a NAT without any tweaking. However, there are a more complex set of applications
which have problems with NATs due to the assumptions taken by the implementation of the NAT box and the limitations of only identifying traffic by the tuple of: protocol number, source IP number, destination IP number and source & destination port number. This set includes, Video conferencing, VoIP, online gaming and peer-to-peer file sharing applications. The source of their common problem is that they assume symmetric connectivity which means that end-points outside the NAT should be able to connect to end-points behind the NAT which are not visible due to address translation and filtering.

Another problem with middleboxes is conflicting services. There is no way to order the services or to detect them. For example, content adaptation, security, compression and cache lookup must be performed in the correct order.

1.2.4 Interconnecting multiple types of networks
Since the Internet was conceived, there has been growth in different types of computer networks. Wireless LANs, ad-hoc networks, mesh networks, sensor networks and Personal Area Networks (PANs) have arisen during this time. Since the Internet has become the network of choice, the TCP/IP protocol suite has been ported to run on top of these networks. However, it is far from clear that this is the only way to interconnect different users and types of networks. The TCP/IP suite imposes homogeneity on the different types of networks it is run over. Whilst this approach is elegant and allows for a uniform approach to interacting with different types of networks, it necessarily abstracts away from the details of the underlying networks themselves. This can lead to difficulties in situations where the structure of the underlying networks are completely different to the Internet. A typical example of this are sensor networks where the addressing structures are often built around application data rather than around the relative location of nodes. This means that in order for such a network to be interconnected to the Internet, compromises need to be made in how accurately the Internet model or architecture can reflect the exact nature of these networks. One alternative approach is to use indirection to abstract away from the internals of how different networks are implemented. This would mean that instead of identifying an end-point in a destination network using the addressing semantics of that particular network, a bridge or gateway is used to map between the different types of networks. This requires indirection as the communication taking place is not end-to-end.

1.3 Thesis Research Areas
The Internet architecture uses a end-to-end, uniform approach to connectivity (informally known as The Hourglass Model) which assumes that the layers
above and below the network layer are heterogeneous whereas the network layer itself is homogeneous. The Internet architecture also assumes a direct path between a source and a destination. As noted previously, this architectural assumption of a direct path is not valid in the deployed Internet due to the introduction of middleboxes. As we have noted in section 1.2, extending the functionality of the Internet is often difficult due to the restrictions imposed by this hourglass model.

The main research area of this work is investigating how to extend the functionality of the Internet with the use of indirection. Since indirection does not assume a direct path between a source and a destination, it provides a different connectivity model. We investigate the trade-offs inherent in an indirection approach and compare it to the current Internet approach.

The main differences between the an indirection approach and the Internet approach is where they place packet forwarding, addressing, routing and management. The Internet places this functionality inside the nodes of the network whereas the indirection approach places this functionality in gateways between the different networks. As the IPv4 Internet functionality resides at each node of the network, this provides a uniform way for higher-level protocols to interact with a network. It is reasonable for protocols above the network layer to make assumptions about how the network will behave. However, this can lead to difficulties when an entity is introduced into the network which violates the assumptions that there is no application-specific knowledge embedded in the network. This is most clearly illustrated by the NAT example described in section 1.2.3. The indirection approach requires more complexity in terms of discovering and managing how to communicate with different networks, but has the advantage of being able to provide explicit support for introducing middleboxes into the network.

This thesis is concerned with how to construct a network architecture which uses an indirection approach. How this indirection is created and managed are the core topics of the work presented. The proposal is SelNet.

SelNet consists of two parts: XRP (eXtensible Resolution Protocol) and SAPF (Simple Active Packet Format). XRP is a uniform interface to resolving between names and SelNet’s internal representation of identifiers. SAPF is a minimal packet format providing basic packet demultiplexing functionality through the use of selectors. The key properties of the SelNet architecture are:

1. **Function-orientated approach**: the addressable components of the SelNet architecture are functions rather than hosts or interfaces.
2. **Explicit Resolution**: Typically resolutions in the Internet are implicit e.g., DNS and ARP, in that there is very little an end-point or user can

---

6Simple in this context refers to the fact that there is only one field in the protocol header format.

7Thus giving the name “Selector Network” to SelNet
do to influence these resolutions. SelNet makes resolution a central part of the architecture and places it under the control of the endpoints. Resolutions are made from implementation-specific names to SAPF selectors.

SelNet proposes a uniform approach to managing indirection. It resolves name and/or address resolution activities into its own internal representation which we use for indirection purposes. By requiring explicit resolution, SelNet becomes a natural point to handle mobility, middleboxes and other indirection requirements. We suggest the architectural principle that indirection should be a core part of any network architecture since it is a generalizable principle applicable to many situations. We argue that a uniform approach to performing and managing indirection will help create more efficient indirection structures instead of multiple partially overlapping structures as is the case today. Additionally, contrary to existing approaches, SelNet can address functions in the network rather than hosts or interfaces. Examples of such functions are customized routing, media transcoding and compression.

We now provide an example of name resolution to show how these two approaches of function-orientation and explicit resolution are used in SelNet.

In SelNet name resolution is performed in the following way: First, the source node puts a query on a link which asks about how to reach a given destination. Then the target node will reply with information about how to reach it, e.g., a link layer address and a SAPF selector which identifies the requested target.

More specifically, if a node A wants to communicate with node B, node A sends a resolution request (RREQ) to the “well known” XRP selector which is a selector which all nodes share. This request specifies the name that node A wishes to resolve (e.g., the IP number of B) and how the resolution should be done. This could be an ARP-style resolution, which would return a complete link layer + SAPF selector address pair, or a DNS-style resolution where a translation between name spaces occur (e.g., from logical name to IP number), or a combined resolution (i.e., from logical name to link layer + selector address pair).

Figure 1.2 shows a resolution from an IP number to a link layer + SAPF selector pair. Before sending out a RREQ, Node A installs a new selector which points at a function to handle any resolution reply it may receive. This function is identified by the selector ‘r’ in figure 1.2. We assume that B is the node that A wants to reach:

1. A sends out a RREQ which is received by B (carrying the target address as well as A’s reply details). B decides that it should send back a reply and prepares itself for receiving IP packets from A. To this end, node B creates a local selector entry ‘d’ (either randomly assigned or by other means) for the delivery of packets to its IP stack. The com-
Figure 1.2: SelNet performing a resolution from an IP number to a link layer + SAPF selector pair. Inside each node are function tables which map SAPF selectors to functions.

1. The RREQ (broadcasted) resolution target: IP addr of B resolution style: eth+sel reply to: <ETHa, r>

2. RREP (to <ETHa, r>) result: <ETHb, d>

3. DATA (to <ETHb, d>) payload: IP packet

bination of B’s link layer address ethb and the selector ‘d’ is then sent back in a RREP to A’s link layer address and selector ‘r’

2. Based on the RREP, node A will install a forwarding entry with selector ‘f’ pointing to the pair ethb+d’

3. Once this resolution step has completed, the system is now at the state where forwarding can take place.

The motivation behind using a function-orientated approach and an explicit resolution mechanism is to deal with the lack of extensibility inherent with network architectures that have a fixed semantics for interpreting addresses. The usage of these two techniques allows SelNet to resolve a name to a function. This is unlike the Internet which only resolves names to addresses of hosts or interfaces. We illustrate this below:

The above table shows how resolution is performed in the Internet and how it can be augmented with SelNet. Note that SelNet does not return a locator, but rather a handle for referring to the destination. The purpose of introducing SelNet as an underlay8 to IPv4 is to enable backwards compatibility. By hav-

8An underlay is a network built underneath of the network layer, as opposed to an overlay which
**Architecture** | **Resolution Steps**
--- | ---
Internet | DNS name → IP number → identifier → locater
Internet + SelNet | DNS name → identifier → locater → identifier → function → SelNet
native SelNet | name → identifier → function

*Figure 1.3: Arrows indicate resolution. Either with DNS or with XRP. Arrows inside a brace indicate association.*

By adding an extra resolution step we are able to provide indirection by allowing a name to be resolved to an entity which is not a destination address. In this sense SelNet can be said to be working with *start addresses* (e.g., the link layer + SAPF selector in figure 1.2) rather than destination addresses such as an IP number. In the LUNAR work presented in section 3.3 we use this indirection technique to provide multi-hop routing in an ad-hoc network by obscuring the true location of the destination host. Every host in the network looks like it is one hop away. The native SelNet approach which we used in the Janus work is presented in sections 3.4 and 3.5 mapping between an IP network and a sensor network. With an indirection step, it is not necessary for the external network to know the addressing details of the sensor network and the external network can retain its IPv4 addressing semantics.

### 1.4 Contributions

My scientific contributions are:

- The specification and design of an indirection architecture for the Internet called SelNet that attempts to provide a uniform indirection primitive which can be used for a wide-range of applications. SelNet allows a high degree of extensibility whilst maintaining backwards compatibility with the existing IPv4 Internet.
- A carefully chosen set of implementations and simulations which demonstrate the feasibility of SelNet and its ability to reach its goals.

#### 1.4.1 Implementation Work

During the course of this research, several implementations of SelNet were created which I was directly or indirectly involved with. The implementation work was typically done in the context of the papers included in this thesis.

is built on top of the network layer.
The goal of the implementation work was to investigate how the architectural principles of SelNet could meet the expectations of areas where indirection could be beneficial.

**SelNet**

There have been four implementations of SelNet. The first three were implemented under Linux and the fourth was implemented in the NS2 network simulator:

1. **SelNet v0.1 – 2001** : The very first implementation of SelNet was written by Christian Tschudin and became LUNAR. This version ran only over Ethernet.
2. **MPLS & SelNet – 2002** : A system which is similar to SelNet is MPLS [37]. In order to further compare these two systems we inter-networked the two systems together so that a single path through a network could be constructed which became part MPLS and part SelNet. This path is completely transparent to the applications running over it. This work was done by Andreas Westling as part of a Masters thesis and supervised by me.
3. **SelNet v1.0 – 2003** : This version of SelNet included a UDP Tunneling support for SelNet. The way that SelNet is structured means that implementing SelNet on top of other datagram services other than Ethernet was not too difficult since the principle remains the same. However, tunneling all SelNet traffic over UDP presented some difficulties since the packets must be received on the Linux virtual interface, be read in from the UDP socket and then processed. We successfully transferred a 100MB file using this approach and ran an interactive SSH session.
4. **NS2 SelNet – 2004** : In order to better understand how SelNet behaves in large networks and to investigate route selection as a SelNet application, a version of SelNet was implemented for the NS2 network simulator. Once again, the difficulty in implementing network software which fits between the layers of the protocol stack became apparent. The NS2 model of the TCP/IP protocol stack is very rigid and not easily extensible since it was not designed to be used in that way. The focus of NS2 is on the transport and application layer, and it is therefore relatively straightforward to extend these layers. Extending the network layer or inserting half-layers is considerably more difficult, requiring extensive rewriting of NS2 and subsequently a large amount of layer violations in order to propagate SelNet packets to the desired place.
LUNAR

LUNAR now exists in two main forms: user-space and kernel-space versions which were written between 2001 and 2005. They were written by Christian Tschudin. LUNAR has been implemented under Linux, Windows 2000/XP and the Sharp Zaurus. LUNAR has subsequently been extended in the following ways:

1. **microLUNAR – 2002** This was a stripped-down version of LUNAR which ran on the Lego Mindstorms. It used the microIP stack and Van Jacobson header compression. It was written by the Datakom II microLUNAR project students and was supervised by me.

2. **LUNAR over Bluetooth – 2003** As a follow-on to the microLUNAR project, we ported microLUNAR to run over Bluetooth. The goal of this implementation was to investigate how LUNAR would behave if it was placed in an environment where the link layer is connection orientated. Bluetooth requires the setting up of pair-wise connections between nodes before communication. Instead of using a broadcast packet to discover neighboring nodes, Bluetooth uses an inquiry phase to build up a neighbor set which an application can then query. This inquiry phase must be completed in order for an application to be able to communicate with its neighbors. Whilst this inquiry is not so costly to do when an application is started, it becomes very costly if it has to be performed regularly to keep track of topology changes in the network as is the case for LUNAR and other ad-hoc routing protocols. LUNAR over Bluetooth was written by Olof Rensfelt and supervised by me.

Janus

In order to investigate how different types of networks can be inter-connected, we applied the SelNet principles to interconnecting a IP network with a sensor network. An implementation was created which ran on the Contiki operating system. This implementation allowed us to send SelNet queries into the sensor network and report them to a host running a TCP/IP stack. Additionally, the sensor network was able to react to an event and send a message to the host outside of the sensor network. Indirection was used to mask the addressing details of each network from the other. The networks communicated through a middleware which translated between them. SelNet was used to provide an intermediate representation of the TCP/IP network and the sensor network. The function-orientated aspect of SelNet mapped well to the data-centric properties of the sensor network. An additional implementation iteration allowed us to take this approach one step further and actually inject code into the sensor network. This implementation runs under the Contiki network simulator. The code was written by Mats Uddenfeldt and Björn Ahlström in 2005 and supervised by me.
2. Related Work

2.1 Introduction

We present a survey of relevant work in the area of providing indirection functionality to the Internet. Related work is divided into the following areas:

1. Network architectures: There have been many new network architectures proposed during the last 25 years, we present a representative sample of those which focus on providing an indirection service.

2. Underlay networks: These provide indirection by inserting an indirection layer underneath the network layer. By inserting an extra layer below the network layer, an underlay can control how packets are processed before they reach the IP layer. Forwarding decisions can be made before the destination IP address is looked up by the IP forwarding component of a host or router for example, deciding which flow or quality class a packet belongs to.

3. Overlay networks: Proposals involving overlay networks are concerned with extending the functionality of the Internet by building structures on top of the network layer. Indirection is thereby achieved by introducing extra processing after a packet has exited the network layer. This is a pragmatic approach to extending the Internet as it allows the new functionality to be deployed immediately by only changing the end-systems of the Internet, not all routers and middleboxes as well. Therefore overlay networks are backwards compatible with the existing IPv4 Internet.

2.2 Network Architectures

2.2.1 Yellow Book (1980)

Yellow Book transport services [7] was a transport service for disparate networks. It was developed at University College London in 1980. Yellow Book concatenated different address types together and then used gateways between different types of networks to translate between them. It deals with interconnecting between networks such as X.121 and TCP/IP. SelNet attempts to position itself under transport protocols such as TCP or X.121 and thereby find a common point for uniform addressing (i.e., SAPF selectors). This by-passes
the upper layers of the protocol stack whereas with Yellow Book, the transport level addresses are preserved and concatenated together to perform source routing through the network.

2.2.2 Nimrod (1991)

Nimrod [28] defines itself as “A new IP routing and addressing architecture”. It is notable as being one of the first projects to concentrate on the architectural issues of the Internet rather than the implementation issues of one or more layers. Nimrod is a more ambitious project than SelNet as it attempts to cover all aspects of a network architecture. Nimrod proposes a complete naming, addressing, forwarding and routing architecture for the Internet. The philosophy of the project was that if these goals were met, then the architecture would be sufficiently powerful and flexible. A key point of the Nimrod design space is the separation of identity and location. From the the routing algorithms that should be used to the packet format for forwarding, the Nimrod proposal has a great deal of detail as to how these could be designed and implemented. SelNet, however, provides the ability for different network architectures to be implemented on top of it, but does not specify how this should be done. Additionally, Nimrod does not address how different types of functionality could be addressed inside of the network.

2.2.3 Metanet (1997)

Metanet [58] is an alternative to the Internet architecture. It suggests that instead of attempting to impose a single homogeneous structure on the Internet, regions should be constructed which share some common attributes. The SelNet approach can be used to build a Metanet. XRP can be used as the protocol to traverse regions as described in sections 3.4 and 3.5. The benefits of Metanet are efficiency and deployability. The complexity involved of imposing a single homogeneous structure is moved to translating between different regions in the Metanet. This allows for local optimizations which do not need to interfere with the operation of the rest of the Metanet. As long as a region conforms to an agreed interface with its neighboring regions, it is free to change its internal representation. Additionally this architecture embraces middleboxes such as NATs, proxies and firewalls since they define the boundaries of a region. We note here the foresight of the authors of suggesting such an approach in 1997 which has inspired a number of subsequent work including our own.
2.2.4 Active Names (1999)

Active Names [54] is a network architecture that virtualizes the name resolution process in order to interpose new services. Active Names inserts itself into the name resolution activity (e.g., a DNS lookup) whereas SelNet inserts itself into the link layer address resolution activity (e.g., ARP). Active Names provides indirection through this virtualization since requests can be redirected to any arbitrary service. This system shares many similarities with SelNet, especially in the goals of the project. However, the architectural choices that are made are quite different. Since SelNet is designed to interface to network resolution activities, we need to sit below the network layer. Active Names, on the other hand, is concentrating on introducing extensibility into the name resolution process. Additionally, the Active Names system is invoked once per connection which is acceptable for the applications that they specify, however we wish to be able to cope with more dynamic networks which may change their properties during their lifetime.

2.2.5 TRIAD (2001)

TRIAD [23] is a content layer for the Internet. The rationale behind this project is that since so much of the Internet’s traffic is based on locating and transferring content it makes sense to add additional mechanisms to the Internet architecture to support this. They define an explicit content layer above IP and use content-aware routers and DNS servers to introduce their new resolution functionality into the network. We have also thought about content-based routing in SelNet and it seems to us that TRIAD and SelNet are complementary in their approaches. TRIAD deals with locating content in the network. SelNet’s indirection mechanisms can support the location of content and the subsequent packet forwarding mechanisms by allowing the resolution of a piece of content to a particular server. TRIAD provides a resolution system which provides content-to-name or content-to-number resolution. It would be possible to express queries for content using XRP, then using TRIAD to locate the appropriate content server through its protocols and then forward packets to the server using SAPF.

TRIAD demonstrates one way of how content-based routing could be introduced into the Internet. It emphasizes backwards compatibility as much as possible and advocates changing NATs, BGP routers and DNS instead of attempting to replace the entire Internet infrastructure. Projects such as TRIAD inspired us to work on the original SelNet proposal to see if we could create a network architecture which would provide explicit support for alternative styles of routing such as content-based routing.
2.2.6 IPNL (2001)

Resolution and translation mechanisms have also been investigated by IPNL [20]. In IPNL (IP Next Level), the authors propose an extended version of NAT to provide the benefits of NAT simultaneously with the benefits of IPv4. The authors argue that the reason that NAT is deployed is not only to deal with the address space exhaustion issue, but also because NAT provides site isolation. This has numerous administrative and security-based advantages. The disadvantage of NAT is that it breaks the symmetric connectivity principal of the Internet because some hosts may reside in domains which cannot be directly addressed from the Internet. IPNL therefore attempts to reconcile NAT and IPv4 together in a cohesive architecture. In order to achieve this the authors use Fully Qualified Domain Names as global end-to-end addresses and then use IPNL addresses to perform dynamic translation between different domains. This approach is intriguing as it advocates making routing decisions based on a name rather than an address as in the current Internet routing architecture. Also, by creating a layer above IPv4 i.e., essentially using IPv4 as a link layer, the authors ensure that only NAT boxes and end-systems need to be upgraded in order for IPNL to be feasible in the Internet of today.

SelNet also provides different domains in the network and offer some translation mechanisms in order to map between them. However, IPNL is much more focused on solving the specific problem of how to provide both the benefits of NAT and IPv4. It makes no claims to providing in-network processing or redirection to specific middleboxes, for example. Additionally, instead of tightly integrating SelNet with IP in the way that IPNL functions, we place our translation mechanisms below IP in order to avoid dealing with the complexity of rewriting IP addresses like NAT or IPNL. IPNL is potentially more well-suited than SelNet as a near-term solution, but perhaps SelNet is more suitable for a longer-term solution which could potentially replace IPv4 as it is more flexible.

2.2.7 Role Based Architecture (2002)

The Role Based Architecture (RBA) [11] is a proposal for a network architecture with no layers. SelNet and RBA share similar goals in wishing to extend the functionality of the Internet. SelNet does not specify how the layering of the network architecture it supports should be implemented. Whilst the layered model provides structure and standardization to the Internet architecture, it encounters problems with attempts to evolve the Internet. This is because there is no natural way to extend the functionality of the Internet. The RBA proposal proposes a protocol heap. In the RBA model, the various protocol actions (Roles) are decomposed into functions which are addressed by a heap of
headers carried in a packet. Since the order of protocol actions is determined by the headers carried by a packet, this model explicitly allows for middleboxes. This is because in order to add a new protocol action, it is sufficient to simply add another protocol header to a packet which references a new role. Since the protocol actions on a host or router are applied in the order specified in the packet, adding a new protocol action does not violate any existing architectural assumptions and middlebox redirection could be added on-the-fly. It would be possible to implement a version of RBA using SelNet. A packet could carry selectors which reference various protocol actions which reside on a host. In order to extend the functionality of the network layer, it would be sufficient to add an additional selector to the packet header so that a new protocol action can be invoked. We see RBA more as a way of reasoning about a network architecture than as a competitor to SelNet. The approach and vision of the RBA proposal has informed our own work on SelNet.

2.2.8 Plutarch (2003)

Plutarch [12] is a network architecture proposal for bridging disjunct networking contexts to form a cohesive network. It is comprised of contexts which are groups of network elements (hosts, switches, routers etc.) that are homogeneous in terms of naming, addressing, routing and transport protocol. Contexts are bridged together by the use of interstitial functions (IFs) which are inserted between contexts to map between them. This is very similar to the Metanet proposal discussed in section 2.2.3. Plutarch provides more depth than the original Metanet proposal, but still does not provide specific details about how it could be implemented. In our implementation work with SelNet, we have presented some possible ways that a Metanet could be constructed. Plutarch can express the heterogeneity of the current Internet as well as future networking systems by dividing them into homogeneous contexts. Plutarch and SelNet share a common approach of making the heterogeneity inherent in the Internet explicit and controllable. Indirection is thus achieved by allowing a packet flow to traverse multiple contexts which could potentially contain multiple types of functionality that operate on packet flows. Plutarch does not specify mechanisms to actually perform this task, but rather leaves it to the actual implementation details of each particular context.

2.2.9 Host Identity Protocol (2004)

The Host Identity Protocol [30] (HIP) is a proposal for adding a cryptographic namespace to the Internet in order to solve the problems of the current two global namespaces of DNS and IP addresses. Examples of these problems are lack of anonymity, lack of authentication and lack of dynamic re-
addressing. HIP provides a separation between identity and location which allows the identity of a host to be constant even though the location may change. The approach of HIP of a cryptographic namespace matches well with SelNet’s approach of resolving names. SelNet could use HIP to specify the cryptographically-assured identities of hosts in the network and then resolve these names into SelNet’s internal representation. Our one concern with HIP is the complexity of its protocol specification. We would have hoped that for such a fundamental name service, it would be possible to have a simpler protocol for ease of understanding and implementation.

2.3 Underlay Networks

We discuss Multi Protocol Label Switching (MPLS) [37] in this section. To the best of our knowledge, this is the only widely-deployed underlay network in existence.

2.3.1 MPLS (2001)

Multi Protocol Label Switching (MPLS) [37] is an underlay network which uses label switching in the network for faster and simpler packet forwarding. The motivation behind this is to provide functionality to IP networks that currently do not exist. Both SelNet and MPLS operate underneath IP and both use label switching for forwarding. SelNet, however, provides a more rich range of indirection functionality. MPLS is typically used for Traffic Engineering, Quality of Service (QoS) and Virtual Private Networks (VPNs). MPLS is able to make decisions about how to treat traffic before the packet reaches the IP stack. MPLS adds a label to a packet when it enters an MPLS enabled network. This label identifies a Forwarding Equivalence Class (FEC) which tells MPLS how to forward the labeled packet. All packets belonging to the same FEC are treated in the same way. This allows MPLS to perform QoS amongst other things as it exercising control over a particular flow of packets. The biggest difference between SelNet and MPLS lies in how the control plane, rather than the forwarding plane. In SelNet, the end-system expresses its needs by using the XRP protocol to steer the resolution process in the network. MPLS does not provide such a mechanism to the end system as it was constructed to assist providers in controlling their networks. The different goals clearly delineate their differences. Also, MPLS specifies quite clearly which functions should be present in the network. SelNet does not specify which functions should be present, but how any arbitrary function can be accessed. The indirection of SelNet is therefore more powerful.
2.4 Overlay Networks

2.4.1 Internet Indirection Infrastructure (2002)

The perception that the Internet suffers from being overly direct has also been observed by the Internet Indirection Infrastructure (i3) project [49]. In order to provide indirection they use a “meeting at the middle” approach called Rendezvous. Instead of a host sending a packet directly to a recipient the packet is sent via a rendezvous point. The indirection goals of SelNet and i3 are very similar but the approaches are very different. i3 restricts itself to IP names as the only type of addresses and to IP forwarding as the single supported packet processing function. For looking up the appropriate rendezvous point i3 uses the Chord Distributed Hash Table (DHT) lookup service. A receiver puts a key called a trigger into the lookup service. This trigger is then used by the sender to route a packet through the i3 overlay network to the receiver. By allowing hosts to communicate with each other indirectly, i3 enables effective multicast and anycast support. This is because the sending host does not need to know the identities of the receiving hosts. The i3 approach provides host mobility support since the indirection hides the location of the receiving host. One important difference is that the i3 proposal is positioned architecturally as an overlay network i.e., above IP, whereas SelNet is an underlay network i.e., below IP. This means that i3 is necessarily restricted by the underlying network. The approach trades off ease-of-deployment against richness of expression. We see SelNet more as an architectural approach which goes beyond what can be done with an overlay network as it can change the functionality of the protocol stack itself.

2.4.2 Layered Naming Architecture (2004)

The Layered Naming Architecture [6] position paper advocates adding two additional naming layers to the current Internet architecture. One layer names services in the networks with service identifiers (SIDs) and the other layer names end points in the network with EIDs end host identifiers. The purpose of these two additional layers is to correct a number of issues with the current Internet architecture involving lack of indirection such as mobility and middleboxes. Ensuring a separation between an identifier that represents a service and an identifier that represents an endpoint allows mobility to be easily introduced into the Internet. This is because the collapsing of identity, access and location into the IPv4 address is no longer present. SelNet also adds extra layers to the Internet architecture, but not as many as the resolution that SelNet provides is richer than the individual resolutions provided by the Layered Naming Architecture. The main issue that are targeted are deconstructing the naming elements of the Internet architecture into their component pieces (e.g.,
DNS and URLs) and by that providing explicit architectural support for mobility, middleboxes and modest support against Distributed Denial of Service (DDoS) attacks. The multiple resolution steps provide a mechanism for introducing indirection. SelNet, by comparison, provides its indirection by a single resolution step. We feel that it is not necessary to have so many resolution steps to provide indirection. SelNet resolves SIDs to selectors which are then used for forwarding, thus avoiding the EIDs and IP routing/forwarding of the Layered Naming Architecture. The Layered Naming Architecture proposal also introduces the notions of intermediaries and delegation as replacements for middleboxes. An end-point can delegate packet receiving capabilities to an intermediary somewhere else in the network by inserting a rule into the resolution infrastructure. This is how proxies, NATs and Firewalls can be cleanly re-introduced into the Internet. SelNet shares similar goals, but is comparable to only a certain part.

2.4.3 Resilient Overlay Networks

Resilient Overlay Networks (RON) [3] builds an overlay network on top of IP in order to provide more resilient routing than the Internet currently provides. The general idea is to set up a group of RON nodes at various places in the Internet which form an overlay mesh. When the default route to a particular destination fails, there will exist an alternative route through the RON mesh. As a short term solution, RON is attractive and appears to work well. We agree with the authors of [13] that overlay approaches such as these will not work in the long term due to the complexity of adding control mechanisms on top of the network which also must deal with the capacity mismatch between the overlay and the underlying network. We advocate an underlay approach which strikes a better balance between the needs of the users and the network operators.

2.5 Conclusions

The related work presented here shows a trend in network architecture research of moving from a uniform, homogeneous model of the Internet to a more pluralistic model where the heterogeneity of the different networks are explicit. Different proposals have different motivations for pursuing this approach, but in general they are reacting to how the Internet has evolved over time. More specifically, the challenges brought about by the increase and difference of requirements imposed on the Internet as it has grown. The different actors using the Internet have brought their own sets of requirements which challenge the fundamental principles of the Internet architecture. For example,
security, mobility and new services. New requirements are typically dealt with pragmatically by adding new functionality outside of the existing architecture either with middleboxes or with overlays. Network architecture researchers reacted to this breaking or overloading of the Internet architecture and responded with designs that could accommodate these new requirements in an integrated fashion.
3. Summaries of the Papers

The following papers were written during the course of the thesis work and present the architectural foundation of SelNet (papers A and B) and various application scenarios for SelNet (papers C, D and E).

3.1 Paper A: Network Pointers

The first published paper laid the architectural foundation for the SelNet architecture. It investigated an alternative way of structuring a network based around the concept of Network Pointers. This concept was born as a result of the notion that the Internet is overly direct. This notion arose from observing the growth of overlay networks such as Peer-to-Peer Networks, Virtual Private Networks (VPNs) and route optimization overlays (such as RON [3]). The commonality between these different approaches is that they attempt to provide indirection by building a new network over the existing infrastructure. By redirecting packets into a virtualized overlay network the overlay itself can choose how to route packets. This is a marked difference from being completely dependent on the routing configuration of the networking infrastructure below the application. The main reason for wishing to build a new infrastructure on top of the existing one is that there is missing functionality of the underlying infrastructure. For example, customized routing decisions or media transcoding are common in overlays. We generalize this to say that if indirection was present at the core of the network architecture, then it would not be necessary to implement overlays. The application, instead of building an overlay, would simply request the network to redirect its packets to a certain function which would be controlled by the application. This process can be seen in figure 3.1. Network Pointers are one way of achieving this indirection. In summary a network pointer consists of a pointer value and a packet processing function in order to provide indirection support. Unlike other indirection proposals, we do not use IPv4 addresses as part of our new architecture but rather create a separate namespace under IP which consists of local names which can be dynamically resolved. This allows us to disentangle identity, access and location which are typically conflated in the IPv4 address.

The contribution of this paper is the establishment of the concept of a Network Pointer. This concept is then used in three different scenarios in the paper
to show how they can benefit from the use of Network Pointers. We then de-
scribe in more detail how Network Pointers can be constructed, how they can
be resolved and addressed. Implementation details are then discussed. This
describes the first implementation of SelNet and its functionality.

My contribution to this paper was as co-author. I invented and investigated
with some preliminary implementation work the asymmetric connectivity sce-
nario. I participated in many discussions with Christian Tschudin concerning
all of the scenarios and propositions made in the paper.

3.2 Paper B: SelNet: A Virtualized Link
Layer

Following on from the work detailed in section 3.1, we present SelNet\(^1\) which
is one possible instantiation of the network pointers system. It provides a in-
direction layer which can host IPv4 networks as well as additional infrastruc-
tures. We present in this paper a complete network architecture based on the
network pointer idea which is positioned as a virtual link layer underneath
the existing IPv4 network. We show how forwarding via selectors can be per-

\(^1\)\textit{"The Selector Network"} : so named as it uses selectors at the core of its design
formed and how dynamic resolution of names is achieved.

SelNet is a network architecture which is positioned in the current context of the Internet architecture as a virtualized link layer. It provides explicit indirection support to the upper layers of the protocol stack which can be used for not only network layer indirection, such as loose source routing, but also application layer indirection mechanisms such as distributed proxy networks. The contribution of the SelNet network architecture is twofold: one is the architectural approach of placing indirection at the bottom of the protocol stack and the second is how to maintain flexibility in the face of changing requirements. SelNet aims to achieve this by the usage of explicit resolution mechanisms and addressing packet processing functions.

Another contribution of this paper was the establishment of SelNet as an implementation of the Network Pointers concept. It describes in detail how SelNet actually functions and goes through, step-by-step, a number of scenarios. We show how static forwarding is accomplished in SelNet by the demultiplexing of SAPF selectors and how these selectors can be dynamically allocated by the usage of the XRP protocol. With these SelNet primitives, we illustrate how SelNet can be used to implement NAT-style site isolation scenarios without the brittleness of a typical NAT solution. We also show how route or path selection can be achieved with SelNet and how this allows SelNet to scale. We close with some discussion of SelNet mechanisms for soft-state and protection from Distributed Denial of Service (DDoS) attacks.

I was the lead author of the paper and wrote most of the text. I had frequent discussions with Per Gunningberg which were of enormous help in clarifying my thoughts. Christian Tschudin helped me with discussions and some vital re-drafting of the paper. Ian Wakeman also contributed general feedback on the paper.

3.3 Paper C: LUNAR: Lightweight Underlay Network Ad-hoc Routing Protocol and Implementation

The first implementation of SelNet was the LUNAR² ad-hoc routing protocol. LUNAR targets the common-case of network clouds with approximately a dozen nodes and a diameter of up to three hops [51]. We believe that such settings will be the most popular ones where ad hoc networks can and will be put into operation. The aim with LUNAR was to explore novel ad hoc routing strategies and to constrain ad hoc routing protocol design to pragmatic boundaries. Low protocol complexity helps to easily implement LUNAR in

²Lightweight Underlay Network Ad-hoc Routing Protocol
other environments, as we demonstrate with μLUNAR and also form a good starting point for adding the important self-configuration elements. The main difference between LUNAR and existing ad-hoc routing protocols like AODV, DSR and OLSR is that it is based on SelNet. LUNAR therefore positions itself underneath IP and uses the XRP mechanisms for dynamic resolution and SAPF for packet forwarding.

This paper describes how SelNet can be used as a foundation for an ad-hoc routing protocol. We show the components of SelNet and how they can be used to build on-demand forwarding paths suitable for an Ad-hoc environment. In particular how the XRP message formats are used for discovering available routes. The various implementations of LUNAR are presented: LUNAR, μLUNAR for embedded systems, Bluetooth LUNAR and a port to Windows XP/2000. Evaluation results are subsequently presented and discussed which show LUNAR to have equivalent or better performance than existing Ad-hoc routing protocols. We also present some preliminary work on formal methods which involves the verification of the LUNAR protocol in the SPIN model checker.

I was a co-author of this paper and contributed some of the text. I supervised the implementation of microLUNAR (which was a student project) and Bluetooth LUNAR (which was a Masters thesis project). I was also the co-designed of LUNAR together with Christian Tschudin and performed many tests and evaluations of it.

3.4 Paper D: Janus An Architecture for Flexible Access to Sensor Networks

In order to investigate how SelNet could be used for interconnecting different types of networks, we attempted to interconnect an Internet-style network with a sensor network. The motivation behind this was to provide a flexible way to access a sensor network. This is desirable as sensor networks are typically deployed in harsh or remote environments where it is hard to change or upgrade the nodes of the network. We designed and built an architecture called Janus which is based on SelNet. Janus was used to interconnect a host in an IP network and a 14 node sensor network. It is challenging to interconnect two such disparate networks since they operate on fundamentally different principles. The clearest illustration of this is the addressing structure of the two networks. The addressable component of an IP network is the IP address, whereas in a sensor network it is often the case that the addressable component is a piece of application data or a function. These two types of addressing can be contrasted as node-centric and data-centric. We believe that a SelNet-style approach is useful in this context since it maps very well to the data-centric model of
sensor networks. Additionally, Janus provides a lightweight approach where primitive functions on a gateway to the sensor network can be combined at an external host to provide service composition. This is in contrast to existing RPC approaches where the server contains comparatively more functionality than the client. We believe that it is desirable to keep as little functionality as possible in the gateway. This is because the gateway will usually be in a location which is hard to access physically and because the gateway may be a sensor network node itself. We present in the paper a description of the Janus architecture, show how it is implemented and present some directions for future work.

I was one of the main authors of this paper, I supervised the SelNet implementation work and wrote the majority of the text.

3.5 Paper E: Janus: An Active Middleware for Accessing Sensor Networks

The previous paper described the initial design and implementation of Janus. This paper has a more complete description of Janus and also presents the code deployment mechanisms. Providing service composition for the sensor network is useful when the services available can be combined. We believe that it will be useful for sensor network deployments if the nodes in the sensor network can be dynamically reprogrammed. This allows for the sensor network to be responsive to future application requirements. The main focus of this paper is to show how code deployment fits naturally into the Janus framework. We have ported the \( \mu \)LUNAR system to the Contiki operating system which is used inside of the sensor network and used it for the basis of our code deployment platform. The reason behind this is that we need to address particular nodes in the network in order to reprogram them, and \( \mu \)LUNAR gives us the ability to communicate with any node in the sensor network by giving us ad-hoc, multi-hop routing. We then extended \( \mu \)LUNAR to carry code in the XRP messages. This code could then be inserted into the SAPF table inside a node in the sensor network. Once this code had been inserted it could be addressed in the same way as existing functions in the sensor network. The paper describes how we have implemented this functionality. We note here that it took one person, who was unfamiliar with the codebase, only two weeks part-time to add loadable module support to \( \mu \)LUNAR.

I was the main author of this paper, supervised the implementation work and wrote nearly all of the text.
4. Conclusions and Future Work

4.1 Conclusions

This thesis is concerned with the description of an indirection architecture designed to solve the problems that the Internet has. Amongst the imminent problems that the Internet currently faces are: address space exhaustion, routing, security and general ossification. These are all areas that border on the alternative architecture presented in this thesis, but it is the area of indirection that is the main focus. Particularly the use of indirection to extend the functionality of the Internet. SelNet (or the “Selector Network”) is a network architecture which provides indirection for extending the functionality of the Internet. Indirection in this context means the ability to control, with relative ease, the paths that packets take through the network and how they are processed.

In the current Internet, at least for the access networks which comprise the majority of users, there is one and only way that a packet can go and that is to the default route. Whilst this is sufficient for many applications, numerous applications have arisen which require indirection to be present not only at the application-layer of the network, but at the network-layer itself. For example, the usage of proxies or middleboxes requires the positioning of a network entity between a pair of communicating endpoints. This is not so difficult if the proxy lies already on the path between these two endpoints, but it rapidly becomes a difficult problem if it does not. Another example is the selection of routes or paths through the network. Currently the user has no influence on the path that his packets will take through the network. This means that if the primary path between a pair of communicating endpoints becomes overloaded, a secondary path which may have marginally less capacity than the primary will not be available. This is a phenomenon that has been documented by the Resilient Overlay Networks (RON) project [3] and leads to both over- and under-utilization of network resources [13]. The difficulty arise because the routers in the core of the Internet are not well-placed to see the big picture of end-to-end paths through the entire network. The endpoints of the Internet are, however, much more well-suited to seeing the quality of the paths between themselves. This has been noted by [34, 19, 47].

SelNet is not the first project to discover that indirection as a primitive is useful for such scenarios as outlined above. The research area that grew
around the advent of overlay networks [3] is one such area. The main motivation behind these overlay networks is to build a completely new network over the existing networking infrastructure. By completely we mean that the overlay builds its own routing, addressing and forwarding infrastructure. This approach is very appealing since it allows the rapid deployment of new networking systems which have a real stumbling block to the advancement of networking research. There are drawbacks to the overlay networks such as inefficient routing, inflated headers and lack of reusability. One overlay solution is seldom used as the foundation another since the complexity would rapidly increase with multiple overlays stacked on top of each other. This would only exacerbate the problems of inefficient routing and header inflation.

There are two main elements to SelNet’s design philosophy. One is that we address functions rather than nodes or interfaces as in IPv4/v6 or Ethernet. As pointed out by Ahlgren et al. [1], the main invariant of the current IPv4 network architecture is the IPv4 address itself. The structure and semantics that are applied to these 32 bits by the various routers and end systems of the Internet are what prevent it from being changed. The first design point of SelNet is that indirection should be a core component of any future network architecture. The second main point of the SelNet design philosophy is that we believe that functions offers enough generality for long-term usage of the architecture. Thirdly, we believe that the control of the network architecture should be as much in the hands of the end users as possible.

The principles that we have examined with SelNet have demonstrated more general applicability through our work on LUNAR and Janus. LUNAR has shown that SelNet can be run in extremely constrained environments and Janus has shown that we can bridge fundamentally different types of networks using SelNet’s principles. Both systems are extremely similar in their codebases, the main differences being the functions that provide the functionality and how packets are received and sent. Other than that, the codebases are fundamentally the same. Our implementation of Janus which provided dynamic code deployment in a sensor network was in fact based on our original $\mu$LUNAR codebase.

From our work with SelNet we can conclude that there is a lot of value in researching network architectures, even if they are not deployed immediately or even at all. It provides a framework for structuring discourse on how networks should be constructed. It is useful to evaluate and consider alternative approaches to existing systems since it helps us understand better how future systems could be structured.
4.2 Future Work

The future of SelNet lies in two different domains. Firstly, examining the implications of SelNet and what can be done with SelNet. Secondly the implementation and deployment aspects of SelNet. In the first domain it would be extremely interesting to see what applications does SelNet enable? For example, having the explicit resolution process before actual data transfer can take place means that service discovery can be integrated directly into SelNet. Recent work has been done in applying techniques from Natural Language Processing (NLP) to service discovery and location [56]. It would be potentially useful to apply these techniques to SelNet to provide better service location for the end user. Another aspect of the implications of SelNet would be the economic aspects of SelNet. Which business models are enabled or threatened by SelNet? Would Service Level Agreements (SLAs) between operators be viewed in a different light if user-controlled route selection was enabled?

The second domain concerns implementation and deployment. Currently there is still lacking a solid implementation of SelNet which permits both the route selection and distributed proxy scenarios to be deployed in a real-world setting. Ideally such an implementation would be in the Operating System kernel in order to minimize the performance aspect of an indirection layer being in userspace. The advantage of having such an implementation would mean that we could deploy it to interested parties and examine the experiences of other users using SelNet in slightly more diverse settings than we have currently been able to do. Another interesting possibility would be to port a stable implementation of SelNet to small devices such as the Linksys home router / access point which runs Linux. This would mean that for certain applications we could use a simple hardware platform for deployment in a similar way to Least Cost Routing boxes in the telephony world. Finally, we considered for a while the prospect of porting SelNet to the Planet Lab [40], however this approach was quickly discarded since currently it is not possible to have full superuser access on a Planet Lab node, so Linux kernel modules can not be loaded and network virtualization tools such as the Linux TUN/TAP device (used in the userspace versions of SelNet and LUNAR) also can not be used. This will hopefully be changed in some future version of Planet Lab when more virtualization infrastructure can be deployed.
5. Summary in Swedish

Internet är idag ett världsomspännande nät bestående av en mängd sammankopplade nätverk med design som härstammar från det amerikanska försvardsdepartementets ARPANET-projekt under 1960-talet. Under de senaste 40 åren har Internet utvecklats från ett idéstadium till en central del i den globala kommunikationsinfrastrukturen. Denna framgång kan förklaras med att Internet i själva verket är en bas som stödjer en uppsjö av applikationer som t.ex. Webben (World Wide Web), e-post, multimedia och fildelning. Dessa applikationer används idag av miljarder människor över hela världen. Internet stödjer dessa applikationer genom att tillhandahålla en grundläggande tjänst där datameddelanden kan skickas till vilken annan dator eller enhet som helst.


I avhandlingen presenteras en arkitektur för att lösa problemen genom att införa ett indirekt mellanstege i Internet. Det ger bättre möjlighet att kontrollera vilken väg datameddelanden tar samt hur de behandlas. Några av de mest aktuella problemen i dagens Internet inkluderar: utmattning av adressystemet, förutbestämda vägvalsbeslut för meddelanden som skickas, säkerhet, samt en allmän stagnering av införandet av ny funktionalitet.

Rutten som ett datapaket tar i dagens Internet bestäms av standardiserade routingprotokoll i kombination med tolkning av adressdelen i paketet längs vägen mellan sändare och mottagare. Denna rutt är fix och bestäms av IPv4:as algoritm för vidareformedling av paketet. Val av rutt kan inte påverkas av ändpunkterna, som i de flesta fall representerar användare. Genom att använda ett indirekt mellanstege blir det möjligt för ändpunkter att få större kontroll över vilken väg paketen tar samt hur de behandlas. Detta kan ske samtidigt som existerande internetprotokoll kan behållas i sin nuvarande form.

Det vetenskapliga bidraget i denna avhandling är en design samt en speci-
fikation av en arkitektur för indirekt pakethantering i Internet kallad SelNet (Selector Network). SelNet tillhandahåller primitiver som inte bara möjliggör vilken väg ett paket tar genom Internet, utan också vilka funktioner som exekveras för att hantera paketet längs dess väg. Fördelarna med SelNet är bättre möjligheter för utveckling och införande av ny funktionalitet i Internet, samtidigt som kompatibiliteten med de nuvarande internetprotokollen bibehålls.

Det finns ett stort värde att forska kring en ny internetarkitektur som kan klara kommande tillämpningar och nya kommunikationstekniker. Även om den i slutändan bara är ett förslag till en ny arkitektur som kan bli implementerad i nästa generations Internet, är det viktigt att flera alternativ jämförs och vägs mot varandra metodiskt. Vi tror att SelNet-arkitekturen kommer att mer eller mindre indirekt påverka nästa generations Internet.
6. References


Acta Universitatis Upsaliensis

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 128

Editor: The Dean of the Faculty of Science and Technology

A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. (Prior to January, 2005, the series was published under the title “Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology”.)

© Copyright ACM 2002
Reprinted with permission.
Network Pointers

Christian Tschudin
Uppsala University and CS Dept, Dept. of Information Technology,
University of Basel, Uppsala University,
Bernoullistrasse 16, Box 337,
CH–4053 Basel, SE-751 05 Uppsala,
Switzerland Sweden.

The Internet architecture can be characterized as having a rather coarse grained and imperative style of network packet handling: confronted with an IP packet and its source and destination addresses, the infrastructure almost blindly and unalterably executes hundreds of resolution, routing and forwarding decisions. There are numerous attempts that try to “extend” the Internet in order to either reduce the immediate impact an arbitrary packet can have (e.g., NAT), or to insert diversions from the normal processing paths in order to better use the existing resources (e.g., content delivery). In this paper we argue that we need a more fine grained control, in the hands of end nodes, over how packets are handled. The basic abstraction presented here is that of networking pointers, which we show to relate to low level concepts like ARP caches, but also high level routing decisions for terminal mobility, content delivery networks, or peer-to-peer overlay forming. We report on first implementation experiences of an “underlay” networking approach which uses pointer tricks underneath IP in order to provide new network layer services.

1 Introduction

The universal connectivity provided by the Internet architecture is the major source of the stress that it is currently experiencing. Because the Internet has one, and only one method to identify a destination (IP address), to deliver datagrams (routing tables managed by third parties) and to address services (ports), there have been many attempts to introduce more flexibility into the Internet:

• Overlays, be it for peer-to-peer, virtual private networks, or route optimization à la the Resilient Overlay Networks project [1], redo what the IP layer is supposed to do: packet forwarding. The difference is that end nodes explicitly want to have a say in how forwarding should be done.
• The IP address merges three different networking concepts in a perhaps elegant but troublesome way: identity, location and access. Changing location whilst preserving identity is the source for the Mobile IP proposal. On
the other hand, the conflict between identity and access is where firewalls and NATs intervene.

Changing the Internet’s packet forwarding behaviour really means changing the way the Internet manages state. We ask the question: based upon which header fields, which routing tables and which lookup processes should forwarding decisions be taken? There is a myth associated with the Internet that it is stateless. However, this only refers to per-connection state and does not apply to the infrastructure itself. On the contrary, a surprisingly large amount of configuration state is present in the network: default routes, address ranges in DHCP, BGP tables, OSPF tables, DNS content, port to service mappings, and more recently the addition of per-connection state in the form of NAT mappings.

1.1 Network Pointers in a Nutshell

Network pointers provide a conceptual and programming framework for packet processing in general. In the first place, a network pointer is an arbitrary packet processing function that can be “addressed” through an opaque pointer value.

A simple example would be the sending of an IP packet to an end point: the destination IP address is mapped to the IP address of the default gateway. This is then mapped to the ethernet address of the gateway’s interface and cached in an ARP table entry. In fact, after the aforementioned mapping procedure, the use of any (local) pointer value identifying the cache entry would suffice to forward data to the destination. What we propose is to make that resolution from names and other entities to pointers an explicit architectural element across the whole network. Network pointers and their remote instantiation play a key role in providing “directable indirection” to the network users and operators.

Before going into more details we now review some items of related work which concern themselves with the manipulation of state (and thus packet processing functionality) in the network. This state is typically used to alter the amount of direction and indirection in the network.

1.2 Related Work

The Resilient Overlay Networks (RON) project [1] is designed to increase the reliability of the routing paths present in today’s Internet routing infrastructure. It works by creating an overlay network of RON nodes which all agree to forward packets on behalf of the other nodes in the overlay network. During typical operation a RON node probes the standard paths provided to it by the network for failure. When such path failure is detected, the node then attempts
to route around the failure. We see RON as re-creating loose source routing at the overlay level. It builds alternate routes (indirection) over an IP routing layer that cannot be steered i.e., which is too direct. Similarly to RON, Mobile IP had to also invent its own routing agents in order to work around the lack of influence an end-user has on routing decisions.

The fact that the Internet suffers from being overly direct has also been picked up by the Internet Indirection Infrastructure (i3) project [9]. They see indirection, as a generic concept, is sorely needed in the current Internet. In order to provide this, they use a Peer-to-Peer lookup service which implements a Rendezvous approach i.e., meeting in the middle. A receiver puts a key called a trigger into the lookup service. This trigger is then used by the sender to route a packet through the i3 overlay network to the receiver. Triggers are much like network pointers, except that i3 restricts itself to IP addresses as the only type of pointer values and to IP forwarding as the single supported packet processing function.

Indirection can also be a problem, as exemplified by NAT boxes and Firewalls which introduce indirection in an implicit way. Many tunneling mechanisms have been devised (e.g., PPP-over-SSH [3]) in order to bring back a direct connection (IP-wise) between two end systems when it is desired. Being unable to control when direction and indirection occurs is a major hindrance in the Internet of today and has resulted in many patches in order to get over these problems. We see NAT and overlay networks as a general symptom of this lack of control.

1.3 Got state?

In the face of this inaccessible state and inability to influence the directness of the current Internet, we propose a program of Internet deconstruction: breaking the packet processing black box into components and allowing its recombination in a user-controlled manner via pointer manipulation. Network Pointers, like their programming language cousins, allow the user to control the amount of in- and re-direction. We then present our own architecture based on the idea of Network Pointers and late binding of address semantics. By shifting the focus to underlaying IP (instead of overlaying IP), much richer and optimizable network recombinations are possible, as we are able to combine the components of the IP stack in different ways. IP (and its associated socket library) then becomes an access mechanism and emulation target, whereas overlays – now supported at the same level as IP – are a way to create partial and transient clouds of emulated directness.
2 Network Deconstruction

In this section we briefly discuss three network usage scenarios that serve as prototypical examples for the deficiencies of the Internet’s one-model-fits-all approach. We will also show how these scenarios would benefit from “network pointers”. A network pointer basically stands for a packet processing function. If a packet is “sent to a pointer’s value”, it will be processed according to that pointer’s function. This function can be, for example, the forwarding of a packet whose header contains a link layer address and the pointer value of a next (remote) network pointer. How these pointers are allocated and managed will be discussed in the following section.

2.1 Case 1: Routing in Ad hoc Networks

Ad hoc networks are supposed to set up Internet connectivity in a completely decentralized way, without any help from preconfigured networking elements, and in very short time.

One class of successful ad hoc routing protocols works “on-demand” i.e., they explore the network topology only when there is traffic to be delivered. In this case, the source node establishes delivery paths that run through the network. These paths funnel all traffic for a destination in the right direction. Expressing this in other terms, on-demand ad hoc routing protocols establish per-target “connection trees” whose root is located at the target node.

![Figure 1: ARP forwarding and pointer selection in wireless ad hoc networking.](image)

We can easily represent that tree with forwarding pointers being the nodes and the connecting edges. Instead of labeling these pointers with the target’s name, we imagine that a pointer (node) has a local pointer value and that all neighbours just point to that address (i.e., “link layer” details plus pointer value at the remote node). This is shown in figure 1.

Our approach differentiates the “access” (pointer value, which is a simple label) from the “location” (full delivery path). Data packets sent to the first access pointer will be correctly forwarded, regardless of the final destination’s location.
2.2 Case 2: Asymmetric Connectivity (NAT + Firewalls)

NAT boxes, either introduced for coping with the scarcity of IP addresses or for controlling network traffic, break the basic universal connectivity model of the Internet. Some nodes or services behind the NAT are not accessible from the “ordinary” Internet, while they can see and connect to the “ordinary” nodes. In order to overcome this asymmetry, people have started to setup tunnels and reverse tunnels to access machines behind NAT boxes. A legitimate use of such tunneling is a traveling user wanting to access some files that he left on his home machine unfortunately located behind a NAT. First, before leaving, the user would have had to create a persistent tunnel from his home machine to some waypoint machine that has full connectivity to the Internet. Secondly, when abroad, the user has to login to the waypoint machine, and from there through the reverse tunnel connect to the home machine. Overall, this exports one port i.e., SSH from behind the NAT to the outside world.

![Diagram](image)

Figure 2: Using an outgoing tunnel to pass through the firewall.

By using network pointers in this scenario, it is possible to concatenate together application-layer tunnels (e.g., SSH) as well as standard IP paths into one seamless construct. Typically the traffic from a node in the traditional Internet to a node behind a NAT box has to be manually directed to a waypoint, and then from the waypoint into the tunnel to traverse the NAT box as shown in figure 2. By allowing user-defined mappings to be established between pointer values and their associated packet processing functions, we can use different layers of the networking stack on a per hop basis. This enables us to present nodes behind NAT boxes as nodes with full Internet connectivity.

2.3 Case 3: Mobile Personal Area Networks (PAN)

An important new challenge for computer networks are small area networks linked to a person, which interconnect the cellular phone with the embedded PC in the belt, the wireless ear plug, and the MP3 player in the pocket etc. Networked devices travel as a clique and rely on special nodes (e.g., the cellular phone) to connect them to the rest of the Internet. Because of the intermittent connectivity and often changing access points to the fixed network, it would
be problematic if all devices in the PAN were to handle routing and addressing on an individual basis.

In this case we use a routing data structure comprised of two trees where one tree (i.e., set of pointers) encapsulates the PAN and its components. In figure 3 this is the tree from the fixed network to the PAN, where the dotted lines represent logical paths to nodes inside the PAN and the unbroken line represents the physical path. Another tree is then used from inside the PAN to point from the current gateway to the PAN components to allow them access to the fixed network. This process is shown in figure 3.1. This pointer from the fixed network to the PAN is addressable i.e., other nodes can then use this pointer as a way of communicating with the PAN.

The pointer inside the fixed network can also be called a “point of presence”, or rendez-vous place in terms of the Internet Indirection Infrastructure proposal. Again, we dissociate identity from location and change the pointer values and names locally and on demand: the data structure’s end points (the PAN members, and the “point of presence” in the fixed network), remain stable, while the intermediate pointers can change without requiring the end nodes to react on this.

Figure 3: Changing forwarding pointers to the PAN as a whole.
2.4 The Deconstruction Plan

Our general approach with network pointers is to facilitate the introduction of new functionality into the network in the following way:

We keep IP and applications running on end nodes as they are. We then go under IP as we wish to have the ability to influence the functionality of the lower layers rather than just treating them as a black box. This allows us to configure the logical layer 2 and the ability to perform redirection at this layer. Addresses at the IP layer are translated to internal access names i.e., pointer labels. Pointers are responsible for either forwarding a datagram to a neighbour, or for modification of packet headers (address translation, tunneling, header compression, etc).

Once we are in the “pointer space”, we can begin to build our own abstractions. For example, we can bring an Ad-Hoc networking cloud into the IP space without IP having to be concerned about the multihop conditions of the underlying network. Simplification is also the order of the day with the Firewall/NAT traversal scenario. Rather than having multiple manual steps from the NAT’d machine to the waypoint and then from the user’s machine to the waypoint, we make pointer allocation methods available that enable a semi-automated setup of NAT tunnels.

An important aspect of a pointered network architecture is the management of pointer state which we require to pass through a mandatory resolution step. Pointers become a natural translation point not only for addresses and tunnel splicing, but also for management domains. By restricting resolution e.g., linking it to authentication, we yield more control over a network’s infrastructure to the operator. Ultimately, the plan is to deconstruct IP forwarding and addressing semantics in order to build new network layer services in our own “Pointer Space” that can be mapped back into the IP space. How resolution and pointer allocation works is explained in more detailed in the following section.

3 Selectors and Resolution

3.1 Terminology

A network pointer is a packet processing function that has a local address, called a selector. To send a packet to a network pointer, you need to know the pointer’s address (selector) and the context in which that selector is valid. Inside a context, which is typically a host, packets can be routed through different pointers based on selectors only. In order to send packets to pointers residing in other contexts, one also needs to specify a transport specific context address (Ethernet, IP, Application-layer tunnels etc.). In this case of a
packet forwarding function, the network pointer itself contains the address of the downstream function i.e., the link layer address and the remote selector.

In case of Ethernet, packets would carry a link layer address and a selector. The link layer address determines the context in which the function can be selected. A forwarding chain of network pointers would rewrite a packet’s link layer address and selector value on each hop i.e., the network pointers contain the address of the downstream node as well as the selector value of the remote pointer.

Selectors are local i.e., they apply to the local context. This enables one to route packets through local pointers in a uniform way: network pointers become the back plane of protocol stack components (see also section 4). At the same time, selectors can point to network pointers that transport packets to other contexts. In this way contexts can be nested, which leads to layers and tunnels. In the tunnel case, the selector would point to a tunnel entry function that pushes its own addressing data in front of the packet header. The tunnel exit function, activated by the tunnel selector on the last forwarding leg, strips out that additional addressing. Protocol layering is obtained by letting each protocol entity of a layer sending packets to higher layers to the selector of the next higher protocol entity. We can use the network pointer abstraction to represent forwarding state (possibly instantiated by end nodes) in the network or in the packet header. If contexts are globally addressable we can use network pointers as remote relay points in order to implement a cached form of loose source routing: packets only need to carry the name of the first pointer, from where they will be passed on from one pointer to the other. Alternatively we can send packets to specific transport pointers that use the packet header as a parameter stack by reading the destination parameters immediately following the packet’s selector field.

3.2 Network Pointers and C-Pointers

The term “network pointers” relates to the standard forwarding behavior of network nodes: Routers “redirect” incoming packets to other directions. This is basically a dereferencing operations (usually involving some lookup in a router table) and matches well the classical concept of pointers in programming languages. The main difference is of course the distributed control: in our network pointer view, each data packet becomes a thread of control that works its way through a chain of dereferencing operations. Whereas a list traversal in the classical programming world would be carried out by a single CPU. The other difference is that we generalize the concept of pointer and include with it the possibility to mangle packets, hence we see network pointers as generic packet processing functions. Note that there is not a one-to-one mapping between packets in and packets out for a network pointer. It may
produce two or more outgoing packets (if multicast semantics are required) for each incoming packet or it may require two or more incoming packets per outgoing packet.

3.3 Selectors as Local Addresses

Network pointers are addressed by a pointer value (selector) that we prepend to each data packet. In order to avoid spanning the name space for all potential pointers (and having to enumerate them at a global level), we require that selectors are purely local names. Only those pointers can be addressed that actually are resident in a specific context. The selector, typically implemented as an opaque number, will be used at the receiving context to find the network pointer that the packet should be processed by. We leave the issue of end-to-end addressing to whichever higher-level protocols are using the network pointer infrastructure.

The local naming scheme means that selectors have no global meaning and that, for example, packets being forwarded from one context to the other have to change their selector on each hop. Selector (or header) translation thus is an intrinsic property of a pointered network architecture. To some extent this also applies to current IP networking, where the destination address is repeatedly rewritten to some local ethernet address that is different for each hop.

3.4 Name Resolution

Due to the fact that local pointer names shall be the generic and sole addressable items, we need a way to map end-to-end destination addresses or server names to local selector values to other kinds of identifiers, including global addresses. To this end we provide a generic resolution function residing on all network nodes to which we assign a well-known selector value. Using this well-known selector, an application can request address resolution and get a local pointer name that “stands for” the item to resolve.

For example, a request for resolving a neighbour’s IPv4 name to the Ethernet address (à la ARP) would result in a local delivery pointer being instantiated on the fly which takes care of delivering any packets sent to it to this neighbour. In fact, such function instantiation on the fly already occurs today inside protocol stacks where an ARP cache keeps that mapping in memory. The selector view makes this state explicit and presents the associated network pointer in the form of an addressable selector. We note here the difference between resolution and translation. First, the IPv4 name has to be resolved. Subsequent packets however will be subject to header translation (which is much cheaper), essentially rewriting the local delivery selector into the associated Ethernet address details.
Extending this view we map routing to selectors too. Finding the next hop is in fact a resolution request for a route. This can be done for each single packet or for a complete packet flow, as for example, in on-demand routing protocols for ad hoc networks. As a result, selectors become the (local) names of delivery path entries.

Explicit name resolution is also the entry point for substituting content descriptors or locators with access paths to the (closest) place from which content can be fetched, which is very similar to the approach proposed in TRIAD [7].

3.5 Start-Addresses, not End-Addresses

Unlike IPv4 addresses, a selector does not define a packet processing semantics in advance and across the whole network. Instead, explicit resolution activities are required to bind a selector to some network pointer (packet handler). We have thus moved from an end-address point of view to a start-address point of view. This enables to repointer parts or all of a packet processing chain in order to cope with mobility without having to change higher layer applications (see e.g., example 3 on moving personal area networks).

3.6 SelNet - An Implementation for Network Pointers

We have implemented a Selector Network for the Linux operating system. From former explorations we already knew that packet forwarding based on simple selector lookup rather than IP routing, can be done with 30% less overhead [13]. What we wanted to understand better was the role of the resolution protocol as a general tool for setting up the network pointers.

Consequently we implemented an “eXtensible Resolution Protocol” (XRP) and combined it with the SAPF [11] packet format which represents our network pointers. As a proof of concept for our underlay approach we applied the pointer networking approach to wireless ad hoc networks. The goal was to fool IP about the multihop nature of an ad hoc network by implementing ARP forwarding: ARP requests would return the name of a full delivery tunnel to use instead of a next hop link layer address.

The resulting LUNAR ad hoc routing protocol (Lightweight Underlay Network Ad hoc Routing [12]) works by trapping ARP requests and translating them into XRP queries that are sent to neighbouring nodes. Based on these queries, the neighbours create new network pointers and propagate the request until the destination is found. To our surprise, we could get a first LUNAR version up and running very fast (in hours). Also, the implementation matched the performance of well-established ad hoc routing protocols, although requiring
only 30 to 60% of their code size. A stripped down version of LUNAR was even ported to the Lego Mindstorm embedded devices [8].

4 Future Directions

We envisage network pointers to become a pivotal element in future protocol architectures, both at the conceptual and the implementation level. We describe in this section some possible future applications of network pointers.

Protocol heaps are proposed in [2] as a middleware concept for letting applications influence the processing of packets through the network. Beside the correspondence at the representation level (heap and pointers), we believe that network pointers provide an essential building block for steering packets through processing components and for identifying processing instances. For example, the NAT example presented in [2] makes use of opaque “cookies” which we would map to selectors in a much more natural way: network pointers would then play the role of a cabling back plane for protocol heap modules.

Related to protocol stack engineering we point to work done a decade ago on speeding up packet processing called active messages: several header parsing steps can be avoided by putting the upcall’s memory address directly into the packet’s header [6]. This works well in the controlled environment of distributed systems: for an open network environment, however, additional protection of handler addresses is required. In our network pointer approach we use local selectors as well as an explicit resolution step for retrieving the dynamic selector values.

The use of selector-like state identifiers has been recently proposed in [4]: Ephemeral state processing (ESP) enables to instantiate short and predefined remote computations in a lightweight fashion. Each packet can carry a single ESP instruction in order to operate on the “ephemeral state store” residing in each node. State entries are identified with a randomly chosen 64-bit tag, which corresponds very well with our selector concept to address state and functions.

Finally we point to our own ongoing work on stored program router [10] where the forwarding table is turned into a general programming area. Individual table entries contain single instructions (e.g., forward, push label, conditional forward) such that packets can be processed by routing them through a chain of table entries. Because packets can also influence the table’s content, we obtain a complete computing model where threads of computations are carried by single packets. The chaining of execution steps is done via the packet’s selector value which becomes a direct instruction pointer. In terms of the network pointer concept, we organize the routing table as a single context.
consisting of an array of tiny network pointers.

5 Summary & Outlook

A “network pointer” basically is a packet processing function similar to the indirection support proposed in the i3 project. However, instead of using IP addresses as the target we propose to use a separate local name space (selectors) for labeling network pointers. This enables a precise separation of the IP address meanings (identity, location, access), although only a single naming concept is used.

More generally, network pointers represent packet processing functions that can range from ad hoc path discovery to multicast forwarding, content access, or VPN encryption. Network pointers bring back part of the Catenet architecture [5] where explicit control of address translation and concatenation was a key concern before the flat Internet model began to rule the world. Now that the Internet has become asymmetric and fragmented for good or for bad reasons, we need to empower the end nodes, enabling them to recreate end–to–end services by rewiring the network’s packet processing functions.

Acknowledgements

The authors wish to gratefully acknowledge the useful discussions and comments of, Per Gunningberg, Erik Nordström, Arnold Pears and David Lundberg. The comments of the anonymous reviewers also helped form the later version of the paper.
7. References


Paper B


© Copyright ACM 2004

Reprinted with permission.
A Virtualized Link Layer with Support for Indirection

Richard Gold and Per Gunningberg
Dept. of Information Technology, Uppsala University, Box 337, SE-751 05 Uppsala, Sweden.

Christian Tschudin
Computer Science Dept., University of Basel, Bernoullistrasse 16, CH–4053 Basel, Switzerland

The current Internet today hosts several extensions for indirection like Mobile IP, NAT, proxies, route selection and various network overlays. At the same time, user-controlled indirection mechanisms foreseen in the Internet architecture (e.g., loose source routing) cannot be used to implement these extensions. This is a consequence of the Internet’s indirection semantics not being rich enough at some places and too rich at others. In order to achieve a more uniform handling of indirection we propose SelNet, a network architecture that is based on a virtualized link layer with explicit indirection support. Indirection in this context refers to user-controlled steering of packet flows through the network. We discuss the architectural implications of such a scheme and report on implementation progress.

1 Introduction

One of the requirements on the Internet today that was not part of the original design is that of indirection. Indirection enables such applications as mobility, proxies and route selection. Current solutions to these problems inside the Internet architecture include Mobile IP for mobility [39], application-specific solutions for proxies [5] and overlay networks for route selection [1]. These systems try to provide indirection by either extending the network layer or by building an application-layer network since the functionality that they require is not present in the current Internet. However each system is separate from the other and each requires deployment of separate infrastructure. Loose-source routing would have been one potential way to implement these services at the network layer but it has been disabled in most ISP networks for performance and security reasons because it was difficult to control and manage.

SelNet is a virtualized link layer (VLL), it sits at layer 2.5 and provides a completely flat topology with no structure to the network layer. It provides explicit indirection hooks to the network layer for control purposes. SelNet’s virtualization service allows the creation of virtual networks using the network layer protocol whilst creating a virtual link layer out of any available datagram protocol. It is based on the network pointers approach detailed in [13]. These indirection hooks occupy a similar space to ARP in IPv4 or Router Discovery
in IPv6 in that they provide a mapping between a network layer address and a link layer address. These hooks allow the user to influence how the virtual link layer responds to queries. The result that the virtual link layer provides when it is asked to resolve a network layer address to a link layer address will be determined by the user’s requirements. Internally SelNet uses a label-switching approach for forwarding packets through the VLL. Due to its virtualization properties SelNet does not influence the topology or technology of the underlying link layer.

SelNet proposes a uniform approach to managing indirection. It resolves all address resolution activities into its own internal VLL representation. By requiring explicit resolution of addresses, SelNet becomes a natural point to handle mobility, route selection, NAT and other indirection requirements. Our contribution is the architectural principle that indirection should be implemented towards the bottom of the network stack since it is a generalizable principle applicable to many systems. We argue that a uniform approach to performing and managing indirection will help create more efficient indirection structures instead of multiple partially overlapping structures. Additionally, contrary to existing approaches, SelNet addresses packet processing functions in the network rather than nodes. Examples of packet processing functions are multicast fan-out, media transcoding and compression. Also we note that there is significant benefit, in terms of flexibility and potential system lifetime, in not specifying how nodes are addressed, but rather leaving that to the protocol stacks which are hosted by SelNet.

1.1 Diagnosis

In the Internet a node is able to communicate with any other node in the Internet without prior invitation i.e., signaling. This unrestricted model of universal connectivity, although desirable for its simplicity and user empowerment, allows unsolicited communication and therefore opens up the way for Denial of Service (DoS) attacks. This connectivity model does not allow for per-application expressions of connectivity requirements. There is much debate about NAT and its site isolation principle* breaking the fundamental principles of the Internet, but it illustrates that there is a real desire for controlling or limiting universal connectivity. We believe that individual expressions of connectivity requirements will allow for a better balance between the users of the network and the network operators. For example, a video conferencing application that needs a certain port range open on a NAT box in order to receive incoming packets should be able to express these requirements. Furthermore, the Internet’s transparency to applications in that they do not have any knowl-

*We note here that although address space reuse was the original goal of NAT, site isolation has become its most compelling feature.
edge about the network in-between them is widely held as favorable, but also limits the influence users can have on the path. In practice a default route entry is used. This causes difficulty for any application that wishes to take an indirect route to its destination.

The main cause of this current architectural stress is how the change in requirements which are being placed on the Internet affect the properties listed above. Mobility makes transparency harder to achieve since the network needs to be consulted about the current location of the end-point. Site isolation defies the principle of universal connectivity as it removes a section of the network from the globally addressable space. Additionally, the need to protect against distributed denial of service (DDoS) attacks make it undesirable to keep unsolicited communication as a fundamental principle. In order for the Internet to keep its phenomenal growth, it will have to change to satisfy these new requirements.

2 Related Work

Related work in this area can be divided into two main areas: one is other underlay techniques, the second is indirection architectures.

2.1 Underlay Networks

Multi Protocol Label Switching (MPLS) is an underlay network which uses label switching into the network for faster and simpler packet forwarding. In order to achieve this goal a label is added to the packet when it enters an MPLS enabled network. This label identifies an action in the next hop, telling it how to forward the labeled packet. When the packet has reached the boundary of the MPLS enabled network the label is removed and regular IP routing is performed. IP forwarding is computationally more expensive than label switching since inexact matching (e.g., longest prefix matching) needs to be performed as opposed to exact matching for label switching.

SelNet, like MPLS, also introduces label switching between the network and link layer. This makes MPLS a natural target for internetworking with SelNet. In [16], we attempted to internetwork MPLS and SelNet so that an underlay path could be built which was partially an MPLS path and partially a SelNet path. IP traffic could then flow over this multi-hop path without any IP routing taking place. We discovered that the crucial difference between MPLS and SelNet is that MPLS labels must be distributed among all MPLS routers in the MPLS network whereas with SelNet the labels are local to each SelNet node, but we can also choose to globally assign some selectors. But this is not mandatory. MPLS labels are used to address paths whereas the SelNet labels
address functions. This allows us greater flexibility and control over packet processing since we can redirect SelNet packets to functions rather than just to nodes. This property is useful for extending the functionality of the network.

2.2 Indirection Architectures

The fact that the Internet suffers from being overly direct has also been observed by the Internet Indirection Infrastructure (i3) project [49]. In order to provide indirection they use a Rendezvous approach i.e., meeting at the middle, at their i3 servers. For looking up the appropriate server they use the Chord Distributed Hash Table (DHT) lookup service. A receiver puts a key called a trigger into the lookup service. This trigger is then used by the sender to route a packet through the i3 overlay network to the receiver. The goals of SelNet and i3 are very similar however the approaches are very different. i3 restricts itself to IP names as the only type of addresses and to IP forwarding as the single supported packet processing function. The i3 proposal is positioned architecturally as an overlay network i.e., above IP, whereas SelNet is an underlay network i.e., below IP. We see these two systems as being essentially complementary in nature since SelNet could be one mechanism used to redirect certain packets to the i3 service for further processing e.g., end-system multicast or a lookup service for tracking end host mobility.

Plutarch [12] is a network architecture proposal for bridging disjunct networking contexts to form a cohesive network. It is comprised of contexts which are groups of network elements (hosts, switches, routers etc.) that are homogeneous in terms of naming, addressing, routing and transport protocol. Contexts are bridged together by the use of interstitial functions (IFs) which are inserted between contexts to map between them. IFs are used to provide indirection by being able to chose which context to map a particular packet flow onto. Plutarch is intended to be a architecture which can express the heterogeneity of the current Internet as well as future networking systems by dividing them into homogeneous contexts. Plutarch and SelNet share a common approach of making the heterogeneity inherent in the Internet explicit and controllable. Plutarch does not specify mechanisms to actually perform this task, but rather leaves it to the actual implementation details of each particular context.

Active Names [54] is a network architecture that virtualizes the name resolution process in order to interpose new services. Active Names provides indirection through this virtualization since requests can be redirected to any arbitrary service. This system shares many similarities with SelNet, especially in the goals of the project. However, the architectural choices that are made are quite different. Active Names inserts itself into the name resolution activity (e.g., a DNS lookup) whereas SelNet inserts itself into the link layer
address resolution activity (e.g., ARP). Since SelNet is designed to interface to network resolution activities, we need to sit below the network layer. Active Names, on the other hand, is concentrating on introducing extensibility into the name resolution process. Additionally, the Active Names system is invoked once per-connection which is acceptable for the applications that they specify, however we wish to be able to cope with more dynamic networks which may change their properties during their lifetime.

The Resilient Overlay Networks (RON) project [1] builds an overlay network on top of IP in order to get around the lack of loose source routing in the current IPv4 Internet. The general approach is to set up a group of RON nodes at various places in the Internet which form an overlay mesh over the Internet topology. The intent behind this is that when the default route through the Internet to a particular destination fails, there will exist an alternative route through the RON mesh since multiple providers will be providing the connectivity of the mesh. As a short term solution, RON is attractive and appears to work well, however, we agree with the authors of [13] that overlay approaches such as these will not work in the long term due to the complexity of adding additional control mechanisms on top of the network and the topology and capacity mismatch between the overlay and the underlying network. We advocate an underlay approach which strikes a balance between the needs of the users and the network operators will be a more appropriate long term solution.

3 SelNet

SelNet consists of two parts: XRP (eXtensible Resolution Protocol) and SAPF (Simple Active Packet Format). XRP is a uniform interface to access and configure SelNet’s resources. SAPF is a minimal packet format providing basic demultiplexing functionality through the use of selectors†. In this section we show how this architecture works and how the associated “virtualized link layer” model permits us to embed different network personalities and to extend and redefine their services. Throughout this section of the paper, we assume that every node in the network is SelNet capable. We then discuss in section 4.1 a more realistic scenario where we show SelNet’s ability to co-exist with the existing network infrastructure through partial deployment.

3.1 Static Forwarding in SelNet

SelNet positions itself as an underlay network, which means that it sits below the network layer and exports an indirection primitive to the upper layers of the protocol stack. As a virtual link layer, it allows us to use any available

†Thus giving the name “Selector Network” to SelNet
datagram service as a link layer to SelNet e.g., ethernet, or IP itself. In the following example we assume that we map the virtual link layer to a real ethernet thus reconstructing the current Internet model.

The simplest example of how SelNet functions is to describe how forwarding is done in a static environment. Packet forwarding inside SelNet’s virtual link layer is based on “selectors”: Each SelNet packet on the wire carries beside its link layer destination address an additional address field called selector. This packet format is defined by SAPF (Simple Active network Packet Format). The selector address, an flat 64-bit value, identifies the function which is to process an incoming packet (similar to a flow or path ID). The payload is some form of arbitrary content which is handled by whatever function is assigned to the selector in question. Selectors have different values depending on how they are assigned.

Figure 1 shows a scenario where selectors are static and pre-allocated. In this example an application on Node A wishes to communicate with an application on Node C. In order to do so, the application on Node A opens a socket to SelNet and writes to it, thus invoking the `selnet_demux` operation. When this operation is called, the forwarding function is invoked since the selector $s$ is used in this example to communicate with the remote application. The forwarding function performs two tasks: it rewrites the selector from $s$ to $t$ since selector $s$ is only valid inside Node A, and then sends the packet with the rewritten selector to Node B. Selector $t$ on Node B corresponds to the forwarding function which will carry the packet over the next hop to Node C. Once again the selector is rewritten, this time from $t$ to $u$.

When the packet reaches the destination i.e., Node C, selector $u$ is demultiplexed and the payload is passed to the function which is associated with selector $u$. In this case, it is a local delivery function which passes the payload of the packet through a socket to the application. In the next section we show how dynamic resolution is used to establish selectors in nodes in the network.

Note that it is not necessary to rewrite applications to use selector sockets instead of IP. Because SelNet positions itself as a (virtual) link layer, an adaption layer can be inserted between the IP layer and SelNet which maps IP
addresses to selectors in the same way as IP addresses are mapped to ethernet addresses. Thus, existing applications can still continue to access networking functionality via the IP sockets API although IP traffic is carried over SelNet.

3.2 Generalized ARP replacement

In the previous example we showed how SelNet works when selectors are statically assigned (e.g., by an authority such as IANA). However, selectors can also be dynamically assigned by communicating nodes since selectors only have validity on the node that assigns them. Static, global selectors function only when all nodes independently agree on the assignment. This is because there is no distribution mechanism in SelNet for distributing labels as there is in MPLS, for example.

In SelNet, dynamic forwarding state can be set up in a way similar to ARP. First, the source node puts a query on a link which asks about how to reach a given target address. Then the target node will reply with its link layer address as well as a selector which identifies the requested target. In SelNet, this ARP style resolution is carried inside an “eXtensible Resolution Protocol” (XRP) packet format, which is the standard signaling means of SelNet.

More specifically, if a node A wants to communicate with node B, node A broadcasts a resolution request (RREQ) to the “well known” XRP selector. This request specifies the address that node A wishes to resolve (e.g., the IP address of B) and how the resolution should be done. This could be a ARP-style resolution, which would return a complete link layer + selector address pair, or a DNS-style resolution where only a translation between name spaces occur (e.g., from logical name to IP address), or a combined resolution (i.e., from logical name to link layer + selector address pair).

Figure 2 shows this process in more detail. Before sending out a RREQ, Node A installs a function at a new selector to handle any resolution reply. This function is identified by the selector ‘r’ in figure 2. Let’s assume that B is the node that A wants to reach. B receives the RREQ (carrying the target address as well as A’s reply details) (1), it decides that it should send back a reply and prepares itself for receiving IP packets from A. To this end, node B creates a local selector entry ‘d’ (either randomly assigned or by other means) for the delivery of packets to its IP stack. The combination of B’s link layer address eth_b and the selector ‘d’ is then sent back in a RREP to A’s link layer address and selector ‘r’ (2). Based on the RREP, node A will install a forwarding entry with selector ‘f’ pointing to the pair eth_b+’d’ (3).

Once this resolution step has completed, the system is now at the state where forwarding can take place as described in section 3.1.
Figure 2: SelNet performing an ARP-style resolution.

3.3 Indirection via Multi-hop Resolution

The examples above show how forwarding is performed in SelNet and how state can be dynamically installed on nodes in the SelNet network. To create indirection, we combine these two processes to perform multi-hop resolution dynamically and thereby forward packets via an intermediate node. This process allows SelNet to setup a custom forwarding path through the network similar to a loose source route enabling potential indirection. With this forwarding path the sender does not know the location of the destination, it is
abstracted away behind the selector that the sender uses to get to the next hop on the path.

We have implemented an ad-hoc routing protocol called LUNAR using SelNet for forwarding and demultiplexing [14] which uses exactly this approach. The corresponding configuration can be seen in figure 3. For the sake of brevity we do not include all the details of the XRP message contents. When the intermediate node between Node A & Node B receives A’s RREQ, the address to resolve is checked if it matches the intermediate node’s address. If this is not the case, then the request is routed towards the destination. How this routing decision is made is not specified by SelNet since routing is not part of the link layer of the protocol stack. It could be made by consulting IP routing tables, a lookup to a Peer-to-Peer infrastructure or, in the case of LUNAR, simply rebroadcasting to reach nodes outside of the source node’s radio range. The resolution process can be viewed as the recursive application of the resolution process detailed section 3.2. Once the forwarded RREQ reaches Node B, the destination is checked to see if it matches Node B’s address. If the match is successful, Node B sends a RREP to the forwarding function on the intermediate node. The RREP contains a selector pointing to Node B. Any packet sent from the intermediate node to that selector will end up at Node B and be demultiplexed up to the appropriate function. This is the same forwarding technique detailed in section 3.1. In this example, as in the previous sections, that function is the IP stack. The intermediate node then sends a RREP to Node A which contains a selector pointing to the intermediate node. When a packet is sent to that selector it is forwarded to the intermediate node and demultiplexed up to the function which will forward the packet to the network selector pointing to Node B.

By allowing state to be dynamically installed by agreement between the client and the intermediate nodes and addressing functions rather than nodes, SelNet is able to cope with a wide range of networking personalities. The resolution request/reply model allows SelNet to perform late binding on which addressing schemes are supported by the network. This late binding can also be used to handle mobility as a resolution step needs to be performed before communication, thus ensuring that the most current version of the address mappings are discovered prior to communication. This removes the need for a separate system such as Mobile IP to handle mobility as the underlying SelNet infrastructure will not be as direct and transparent as the current Internet architecture.

3.4 NAT

NAT is a very difficult entity for the current Internet architecture to cope with as it breaks the fundamental principles of transparency and universal connec-
tivity. However it does provide site isolation which is in high demand and in a very cost-efficient way by not requiring the the network or end systems had to change. SelNet is able to provide NAT equivalent functionality through the XRP resolution process and SAPF selector rewriting. However, due to the indirection hooks that SelNet has, we are able to selectively restore symmetric connectivity for those nodes behind the NAT which are explicitly authorized. We first describe how a node behind a NAT box gets out to the Internet and then how a node on the Internet can access a node behind the NAT.

The main piece of functionality that NAT provides is a mapping between one address space and another. Typically this is between a domain behind the NAT and the Internet. NAT is actually a special case of the forwarding function detailed in section 3.1 where we not only rewrite the selector, but also the IP address. The process works as follows: first, a node sends out a request for resolution for a node outside of the NAT domain which will reach the SelNet NAT. This uses the process described in 3.2 to dynamically install forwarding state on the SelNet NAT. The SelNet NAT can then performs the appropriate translation to the resolution request which can be translating the network layer address or the link layer address as well as rewriting the selector. Which type of translation that is performed is determined by the local environment i.e., the network layer that is being hosted by SelNet. Once the translation has been done, the translated route request can then be forwarded towards the destination located in the Internet. When this forwarding is performed, state is installed on the SelNet which keeps track of the mappings between NAT domain and the Internet. This state can be stored in the SAPF table by inserting a function there which is responsible for performing the translation.

In order for a node inside of the NAT realm to be externally reachable, it must first send an XRP message to the SelNet NAT requesting for a service or a port range to be opened on its behalf. The SelNet NAT has to approve this XRP request before the appropriate port translation functionality can be initialized. When a node outside of the NAT realm wishes to contact a node inside the NAT realm, its resolution request will reach the SelNet NAT which will then check to see if the resolution request is allowed to pass through the NAT. If this is the case, then the SelNet NAT inserts a function into its SAPF table to handle the mapping between the two address spaces. The resolution request is then forwarded to the node in the NAT realm. The destination node then processes the resolution request and sends back a reply to the source via the SelNet NAT. The SelNet NAT then demultiplexes the reply to the associated function which will perform the appropriate translation before forwarding the packet outside of the NAT realm. This process allows symmetric connectivity to be selectively enabled.

We note here that the addressing mechanism used by the network layer in such an “InterNAT” context is not defined by SelNet. Using IP address plus
port number is one way, but there are other proposals such as IPv4+4 or IPNL. SelNet provides the hosting mechanism for such networking personalities.

3.5 Route Selection & Scalability

Since SelNet’s selectors are opaque i.e., it is not known from the selector value itself what semantics are bound to it, we can attach different functions to selectors. In this section we show how selectors can be used to identify long-lived and public forwarding paths e.g., a transatlantic link or autonomous systems, rather than only local, small-scale network paths.

Resolution of delivery paths triggered by endusers will not scale if these requests require discovery and resolution over the full path across the entire network. For scaling reasons, some form of aggregation will be needed. This implies a reduction of information complexity from a large number of intermediate nodes constituting a path to a single identifier representing a path through the network. One of the effects of this is that if a handful of paths exist between a source and a destination exist, it becomes feasible to allow endusers to select one of these paths through the network for their traffic. This is analogous to Least Cost Routing consumer devices in the telecoms world which chose a provider to route over depending on cost.

The SelNet model for core nodes in the network is to make them as simple as possible i.e., to function as layer 2.5 switches which are only forwarding packets. We imagine various routing protocols running on top of SelNet and feeding SelNet with the appropriate information that it needs to set up paths. We agree with the authors of [7] that physically separating the route computation process from the routers themselves is a good strategy for reducing router complexity.

Figure 4: Highway routing with SelNet. Selector Stacks shown with attached payload.

Mapping a Destination to a Selector Stack

A first example involves an equivalent to autonomous systems on which IP routing is based today. Like MPLS, we envisage label stacks (selector stacks) inside packets. Instead of mapping a destination address to the next-hop autonomous system, SelNet can map a destination to a selector stack which
describes a loose source route through the network. The process works as follows: When a source node requests the resolution of a destination, it will obtain a selector that points to a special local forwarding function. This forwarding function will add the selector which points to a routing function on the next-hop node. Additionally each packet sent to this function will be augmented by a selector pointing to the resolved loose source route. In other words, The front-most selector identifies a resolved path to the routing function which, instead of doing label switching, will pop the front-most selector from the selector stack. The second selector will then be demultiplexed into the node’s SAPF table to invoke the function that will handle this packet. Typically in the core of the network this will be a simple forwarding function. The second selector functions as a path selector similar to PathIDs in BA-NANAS [31].

Consider now the case where each selector in this selector stack representing a loose source route maps to an autonomous system. This means that forwarding such a packet consists of taking the next selector from the stack and treat it as an entry point to a path to the named autonomous system. This path will be labelled with a statically assigned selector such as those discussed in section 3.1. Because connectivity at the AS level is long lived, we can proactively install such delivery and share them between many users. The difference to the IP case is that for IP, the mapping from IP destination number to AS number is implicitly done at each ingress point, while here this mapping is exposed to the end nodes and thus becomes “selectable”. This allows the potential for the end-users to chose one of a set of paths through the network.

**Mapping a Destination to a Data Highway**

In the previous example we used selectors to identify trans-AS paths. Another routing abstraction, which extends the loose source routing concept, consists in using selectors to work with multi-exit data highways. Instead of resolving a destination to a path (identified by an entry selector), or a series of paths with mandatory waypoints (identified by a stack of selectors), we resolve a destination to a pair which comprises of two parts a “highway-name+exit-name”. The highway part specifies which route, among several possible ones if several providers exist, should be used to reach the destination. The exit part specifies which exit on that highway to take. The exit roughly corresponds to the waypoints that a packet will have to traverse. The advantage of this approach is that it provides a level of abstraction which the providers can use to dynamically change their network topology without affecting the traffic flowing through their network. As long as the exit-names are kept in the same order, it does not matter how the network is actually structured.

For example, there might be different transatlantic highways starting from New York to Geneva passing through London and having an exit at Grenoble.
A user in Boston who wants to send a packet to Grenoble has several options now. In one case she could compose a packet with a source route “Boston/Sprint-Highway(NY, Geneva)+exit at Grenoble/default Grenoble router” and issue two resolution requests: one “name lookup RREQ” for the selectors to use for the highway and the exit points, and one RREQ to get a path to Boston. The latter selector permits to reach Boston, the former selectors to build a selector stack. In the Boston node, a packet’s front selector would be popped, which diverts the packet to the transatlantic highway; Each intermediate node on this highway will check whether it matches the exit name. The Grenoble node will match and pop again a selector, finding that it has to be forwarded to the default local delivery function. This process can be seen in figure 4.

Let us further assume that the operator of this highway establishes a shortcut from London to Grenoble e.g., to counter some flash crowd problems. In this case, the packet would already be picked out of the data stream in London and reach Grenoble by this temporary bypass. Thus, the use of selectors offers traffic engineering flexibility on both sides.

Note that except resolution requests at the edge (Boston and possibly also at Grenoble), no end user state had to be stored inside the core network. Because highway and exit points are long-lived, their selector would be cached at the edge and corresponding RREqs would not need to go transatlantic. Note also that the only user-specific packet processing inside the core relates to fast selector stack inspections and forwarding decisions.

4 Discussion

Here we discuss some of the architectural consequences and issues that arise from the design choices of SelNet.

4.1 Deployment

During the discussion in this paper, we have assumed a network where all nodes are running SelNet. This was deliberately chosen to show SelNet’s capability as a stand-alone network architecture. Since deployment on the current Internet is an extremely slow process, we discuss here how SelNet’s underlay approach allows for incremental deployment. One typical way of running SelNet is to use it as an underlay to IP. This means that any IP traffic sent from a SelNet node has the SAPF header inserted between the IP header and the Ethernet (or other link layer) header. The SAPF header allows the packet to be demultiplexed to the correct SelNet packet processing context on arrival. This method of deployment clearly positions SelNet as a virtualized link layer.
However, it is very likely that we would like to start SelNet deployment only in a handful of local area networks. These LANs may well be geographically distributed, but connected via the Internet. In which case, tunneling SelNet traffic over the Internet is one way of achieving SelNet connectivity between these LANs. Using an approach similar to Minimal IP Encapsulation we can insert the SAPF header between the transport and network layer headers so that SelNet traffic can be a) effectively routed through the Internet and b) correctly demultiplexed at the receiving node. Our current implementation of SelNet runs over both Ethernet and UDP. We chose UDP for implementation purpose rather than IP for speed of prototyping. Despite the additional overhead incurred from using tunnels, in our experience SelNet is still usable. This mix of both underlay and overlay approaches shows a promising direction for pragmatic future deployment.

4.2 Resilience

Since SelNet introduces some forwarding state on intermediate nodes, it must have some resilience properties to ensure recovery in the event of failure. Consider the example in figure 3, once the multi-hop path has been established SelNet will periodically refresh the selectors using a soft-state approach. Usually applications will do this matching the time cycles of ARP (120 seconds between refreshes). However, this can be configured on a per-application basis.

Referring back to figure 3, if Node A fails then the selectors present at both Node B and the intermediate node will be garbage collected after a certain timeout. Once the Node A comes back online, the selectors will not be present at the intermediate node, so Node A will have to perform the resolution again in order to ensure that a path still exists between it and its destination.

Whilst such an approach works for small networks, if SelNet is to scale up, then there must be some way of performing soft state so that the maintenance load of XRP does not get out of hand. Inspired by the work done in [8] and [29] we propose a mechanism which uses reliable triggers (i.e., the full set of forwarding state) hop-by-hop to install state and then an end-to-end soft state refresh mechanism to ensure that these triggers are kept up-to-date and are re-installed correctly in advent of failure. This means that the full transmission of state happens at the beginning of the connection and in the case of failure. State is kept up-to-date by periodic refreshing. Thus the transmission of the full set of state is infrequent so the load is not too great, but the state is kept consistent by periodic refreshing by more lightweight mechanisms.
4.3 Security considerations

We outline in this section how access control and DDoS protection are performed in SelNet. Traditional measures of privacy, integrity and authentication are dealt with by standard cryptosystem approaches.

Typically, solutions that introduce more flexibility into the network are rightly considered as security risks as they provide a hook into the system which could be used to break in or take down the system. One such system was the original IP loose source routing which was usually switched off by ISP networks due to security, performance and business concerns. There are two reasons why we feel that SelNet is a better solution for network operators than loose source routing: firstly, it requires explicit agreement from the operator that a certain route is allowed to be taken and secondly, the network operator can choose exactly which routes are exposed to the users and which are not. These bilateral agreements are enforced by the XRP access control mechanisms detailed in the next section.

Access control via XRP

Access control is an important part of the SelNet architecture. By providing fine-grained access control to the user and network operator, we hope that more services will be introduced in a secure way. This is in contrast to the unsolicited communication principle that underlies the Internet architecture. We share the view of the authors of [4] who see security as an enabler of new functionality rather than constricting it. Access control is provided in SelNet through XRP. In SelNet no communication can take place until resolution is complete. Therefore it is much easier for a communication instance to be approved by intermediate nodes (typically an XRP policy box). In the Internet architecture, any node can send to any other node without prior agreement. This makes security difficult as legitimate communication is hard to distinguish from a malicious attack. Firewalls are an attempt to introduce some access control into the network, but they often cause problems by inadvertently blocking legitimate communication due to their lack of ability to discriminate between permitted and non-permitted traffic.

In XRP intermediate nodes have to approve both the destination and method of reaching destination. That is, before a selector can be sent back to the destination, the XRP resolution request has to be processed. These approval policies can be complex or trivial. For example, route all IPv4 traffic by default or route only traffic which conforms to a policy. The key point here is: Who is allowed to do what indirection? By allowing the approval policies to be calibrated as required, we can have access control suitable for certain areas of the network. For example, the core transit networks will very likely have very simple policies due to processing constraints whereas the edge or access networks will likely have more complex policies to match their application
Protection against DDoS

Since unsolicited communication is permitted by the Internet architecture, a node cannot express which packets it wishes to receive and which is does not. In [33] the authors also note this problem with the current architecture. We believe that SelNet can help with these DDoS attacks because an XRP resolution request would be sent to the nearby XRP processing box and that is the limit to where the DDoS attack could go without explicit authorization of the XRP box. Even if the local box is being DDoS’d, it is a simpler problem to handle than a DDoS which traverses multiple IP networks which obscures the identity of the original sender (especially if the packets are spoofed).

One proposal to counter DDoS attacks is Pushback. Pushback is a way of installing filters on intermediate nodes in the network in order to block certain address ranges which are known to be DDoS:ing other machines. SelNet, through XRP, provides a mechanism with which an end node can instruct intermediate nodes (for example, firewall or edge router) that certain types of packets should be blocked. Due to the extensible nature of XRP, we can easily deploy additional indirection schemes. We note here that this indirection scheme redirects certain packets into a black hole rather than to another packet processing function.

5 Conclusions and Outlook

The contribution of the SelNet network architecture is twofold: one is the architectural approach of placing indirection at the bottom of the protocol stack and the second is how to maintain flexibility in the face of changing requirements which SelNet aims to achieve by the usage of explicit resolution mechanisms and addressing packet processing functions rather than nodes.

The main advantages of SelNet are flexibility, performance and security: Being an underlay network also means that we are not constrained by the network layer stack like an overlay network. Since the addressable units of our architecture are functions rather than nodes, we allow for greater flexibility and extensibility than architectures which mandate an addressing mechanism for nodes. With label switching, packet forwarding is faster than with IP [57], and we do not incur the packet header overhead of an overlay approach. Since SelNet is underneath the network layer, it is easier to secure SelNet since resolution is an explicit activity that must take place before data forwarding on the network layer can take place.

We have the core infrastructure of SelNet implemented and we have a real-world implementation of a simple distributed proxy scenario which we will
continue to develop. Additionally we have a version of SelNet based on LUNAR implemented for NS-2 and one element of future work is to investigate user-controlled route selection and the general scalability of SelNet.
8. References


Paper C

© Copyright NEW2AN 2004
Reprinted with permission.
In this paper we describe LUNAR, a lean and efficient routing protocol for wireless ad hoc networks. We report on its implementation, on performance comparisons and on a formal validation result. The protocol departs in several ways from standard ad hoc routing protocols: it does not feature route repair, route caching, route maintenance or packet salvation. Nevertheless it closely matches the performance of AODV in settings inside of the “ad hoc horizon”. We discuss the philosophy of LUNAR, report on several ports including versions for Windows and a stripped down μLUNAR capable of running inside a microcontroller and operating over infrared or Bluetooth.

1 Introduction

Ad-hoc networks are typically described as a group of mobile nodes connected by wireless links where every node is both a leaf node and a router. This flat routing environment causes many challenges which have resulted in a slew of routing protocols and research projects designed to solve the problems posed by ad hoc networks [7, 15, 10, 8].

Many ad hoc routing protocols started with a simple concept and then added “essential” features like route maintenance or packet salvation for staying ahead in the competition among protocols. While this happened at the level of protocol design and simulations, the availability of implementations of these refined ad hoc routing protocols is far from satisfying; Stability as well as ease of installation and configuration are other domains that usually are neglected too. This is in contrast to the continuous sophistication of ad hoc routing protocol specifications as is visible in their version numbers: All four top MANET protocol candidates went through 10–13 revision cycles, but for some protocols there is no public implementation available at all, others more or less have one reference implementation, and today none is cross-platform (Windows, Linux, Mac and PDAs).
1.1 The LUNAR Approach

The aim with LUNAR was to explore novel ad hoc routing strategies and to constrain ad hoc routing protocol design to pragmatic boundaries. Low protocol complexity helps to easily implement LUNAR in other environments, as we demonstrate with μLUNAR. Lean protocols also form a good starting point for adding the important self-configuration elements, as we also show in this paper.

LUNAR adopts a hybrid routing style as it combines elements of both reactive and pro-active ad hoc routing approaches. It is reactive in the sense that it discovers paths only when required; It is pro-active in the sense that it rebuilds active paths from scratch every 3 seconds, even if everything is fine with the current path. This removes the need for additional path maintenance procedures and link repair actions and reduces the complexity of the protocol. Another design choice was the separation of delivery paths. Each host pair potentially has its own path and softstate associated with it: It is the duty of the originating node to keep that state alive. This makes implementation easier as, for example, intermediate nodes can now be fully reactive and do not have to start protocol activities by themselves.

1.2 Ad hoc Horizon

LUNAR limits itself to 3 hops. This decision was based on our experience (when experimenting with the standard MANET protocols over IEEE 802.11) that 3 hops is already pushing the limits in many ways. We hypothesize that there is an ad hoc horizon beyond which it becomes ineffective to handle topology changes as they occur in mobile wireless networks. First, when multihop routing is in place, it means that the wireless cards operate close to their limits, resulting in a highly fluctuating connectivity space: Slight position changes or objects getting in the way drastically change the neighbor set. Second, the freshness of routing information decays rapidly with the number of hops – attempts to do local repair potentially mask or at least delay the recognition of trouble spots. Third, the discovery and maintenance of routes by the use of flooding beyond the ad hoc horizon disturbs the communications of remote nodes more than it serves the local needs.

1.3 Underlay Routing and Internet Access

Another major departure from the MANET protocols is that LUNAR positions itself below IP. Instead of making ad hoc routes visible at the IP level, we create a subnet illusion. To the IP stack it looks as if the mobile node was connected to a LAN. Inside LUNAR we use an underlay at “layer 2.5” which emulates the LAN behavior by establishing point-to-point multihop paths or
point-to-multipoint multicast trees to mimic the broadcasts at the LAN level. The underlay approach leads to immediate implementation benefits as LUNAR does not have to interact with the IP routing tables. Moreover, the underlay discovery mechanisms inside LUNAR permits to add self-configuration elements (address assignment, gateway discovery) in a very straightforward way.

Figure 1: LUNAR as an underlay to the IP layer

2 Related Work

Routing below the IP layer for ad hoc networks was independently adapted by [1] using label switching which is equivalent to the “selectors” that we use in LUNAR. A similar project is [2] where the authors also aim at putting L2.5 routing logic inside the (wireless) network interface card.

There are four main routing protocols currently being considered for standardization by the IETF MANET group. Two are reactive (DSR & AODV) and two are pro-active (OLSR & TBRPF). There is a DSR click router implementation available which seems stable, unfortunately it was not available at the time that this work was originally done. For AODV, which was recently raised to full RFC status, there are four full AODV implementations are available. For our comparisons we used AODV-UU [14] which is a stable and mature implementation. OLSR [16] is available in a stable implementation from INRIA and appears to work well, therefore it was also included in comparisons. Unfortunately, there is no publicly available implementation of the TBRPF protocol.

For AODV, formal validations have been carried out by the Verinet group [19] at the University of Pennsylvania. Using a theorem prover and a SPIN model of AODV in a 2 node setup (with an AODV router environment), it could be shown that – with additions – it is in fact a loop free routing protocol.
3 The LUNAR Protocol

LUNAR presents itself towards the IP stack as a network interface card that attaches to a local area network. All nodes participating in the LUNAR network will appear as being one IP-hop away. Internally, this connectivity is implemented by forwarding data packets over multiple hops.

Figure 1 shows the position of the LUNAR network with respect to the traditional IP stack. The design of LUNAR is based upon the SelNet underlay network [17] which is a frugal forwarding abstraction designed to support a wide range of data forwarding and routing styles. Typically, SelNet is implemented at L2.5, but can also be put at L3.5 as an overlay where it can be used for packet redirection.

The main idea behind LUNAR is to link ad hoc path establishment to ARP. Historically, DSR also followed this approach by doing multihop ARP [9]. Typically ARP is confined to broadcasting only to the subnet that the node is currently residing in. For LUNAR, we decided to allow nodes which receive such resolution requests to rebroadcast them to reach nodes which are outside of the original radio range of the requesting node.

3.1 SelNet

The SelNet network basically offers a demultiplexing service. An incoming packet will be dispatched to an appropriate handler routine based on a single packet header field called “selector”. As a packet is forwarded through a SelNet network, it will have a different selector on each leg of its path because the SelNet network will rewrite the combination of Ethernet address and selector on each hop, in the same way as IP forwarding rewrites the Ethernet address for each subnet crossed. A special selector value is defined for XRP packets (eXtensible Resolution Protocol) which enables to create rewriting state in remote nodes and, in the case of LUNAR, to address the LUNAR specific forwarding logic. Delivery paths dynamically obtain their own set of selector values such that XRP control traffic and the multiple ad hoc delivery paths are kept separated.

3.2 SelNet Control Messages (XRP)

XRP (eXtensible Resolution Protocol) is SelNet’s external data format for requests and replies. Its name stems from the philosophy that “resolution” is a mandatory step before using SelNet’s forwarding capabilities. An example would be the resolution of a neighbor’s ethernet address such that one can send data to it, or a LUNAR resolution where we want to resolve an IP address to a multihop path. In both cases SelNet will set up the necessary state inside the local host as well as in the network, and return an opaque “handle” in form of
a selector value. In a second step one can use this selector for sending packets
to the now resolved destination; The corresponding API consists of a simple
\texttt{selnet\_demux(selector, data, length)}; procedure.

XRP messages are mere containers for parameters, its format was inspired
by the CASP proposal [3] and RSVP. Here we describe the generic data struc-
tures and point to some LUNAR specific parameters. XRP messages begin
with a header which is 4-Bytes long:

\begin{verbatim}
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| version | ttl | flags | reserved |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\end{verbatim}

This header details the version, Time To Live (TTL) and the flags associated
with that header. After the header, we have the various parameters that we wish
to send. There can be multiple XRP parameters per packet.

\begin{verbatim}
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| length (bytes) | class | class-type |
| ... contents ... |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\end{verbatim}

A “null” object closes this list so XRP becomes a self-delimiting packet
format.

\begin{verbatim}
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 0 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\end{verbatim}

A typical LUNAR “Route Request” would contain the following parame-
ters:

\begin{itemize}
  \item Request series (Len=12, class=request series, ctype=sel)
  \item Address to resolve (Len=8, class=target, ctype=IPv4)
  \item Requested resolution (Len=4, class=reqstyle, ctype=sel/eth)
  \item Reply address (Len=20, class=reply addr, ctype=sel/eth)
\end{itemize}

The request series (opaque 64 bit field in form of a selector, ctype=sel)
is used to identify each discovery wave for preventing broadcast storms. We
request the resolution of an IP address (address to resolve, ctype=IPv4) and
wish to obtain the result in form of a selector/ethernet pair that encodes the
first hop of the delivery path (requested resolution, ctype=sel/eth). The reply
address is our local address where we wish to receive the result of this request.

The LUNAR “Route Reply” message will have only one parameter contin-
ing a selector and ethernet address value pair:

\begin{itemize}
  \item Forward address (Len=4, class=reqstyle, ctype=sel/eth)
\end{itemize}

This information tells the requesting node where to send data traffic to. All
data packets sent to this port will be forwarded to the stack whose IP address
was given in the original route request.
3.3  SelNet Packet Format

As can be seen in the XRP example above there is no field that specifies the LUNAR context in which the resolution should be carried out. In fact, this information is encoded as a “well known” selector port to which all LUNAR requests have to be sent to, whereas LUNAR replies are sent to a dynamically allocated selector port as indicated in the parameter list. This relates to SelNet’s basic demultiplexing philosophy and its one-header-field packet format:

```
+---------------------------------+-
| selector (64 bits) | data  |
+---------------------------------+-
```

The same selector+data format is used for a) XRP requests, b) XRP replies as well as c) pure data traffic; Differentiation is made by the selector value only.

In order to underlay IP we define a new ethernet frame type such that IP and SelNet frames can be distinguished at arrival. A complete SelNet frame over ethernet thus has the shape:

```
+---------------------------------+-
| dst (48) | src (48) | typ (16) | selector (64) | data  |
+---------------------------------+-
```

3.4  LUNAR Protocol Logic

LUNAR interconnects with IP by trapping all control traffic and translating it into explicit signaling at the SelNet level. When the IP stack issues an ARP request for an IP number’s Ethernet address, LUNAR translates this to a “route request” (RREQ) expressed as an XRP message. This RREQ is broadcast to all neighbor nodes. Re-broadcasting logic is then responsible for checking duplicate RREQs (each request has a unique ID which is stored on each node it arrives to), for propagating the resolution information to the next neighbors and for temporarily storing return information for the reply.

When the target node is reached, it sends back a “route reply” (RREP) – this time using a unicast message – to the node from which it received the broadcast. Using the return information there, this reply message travels back towards the originator while at the same time creating a data delivery path towards the target node. Data traffic travels inside the delivery paths that LUNAR has established.

LUNAR also has to take care of IP broadcasts that the communication stack might be sending (e.g. broadcast pings). To this end we use a similar flooding based discovery mechanism and express such resolution requests as XRP messages too. In order to emulate a L3 broadcast we send a L2 RREQ broadcast: all neighbors match the discovery request and will rebroadcast it before replying. We delay this reply a little in order to fuse it with the replies from
downstream nodes. Depending on how many downstream nodes replied, the
intermediate node will install either a “broadcast forwarding handler” or a
“unicast forwarding handler”. The net effect is the creation of a delivery tree
where edges are using broadcast in case a node has more than 2 child nodes
and using unicast otherwise.

All data forwarding state in intermediate nodes is phased out after approx-
imately 6 seconds. Therefore, the originator of a delivery path would have to
periodically refresh it. Instead of refreshing, we chose a more suitable strategy
for ad hoc networks by building a new path every 3 seconds. This way LU-
NAR is able to react swiftly to topology changes, although it does not use any
hello beacons or other link layer mechanisms to discover broken “links”. Old
state is garbage collected by each node without additional signaling overhead.

3.5 Intercepting DHCP for Address
Self-Configuration

We have already described how LUNAR intercepts ARP messages and turns
them into RREQs for establishing forwarding paths. Similarly, LUNAR in-
tercepts DHCP messages and translates them into its own address allocation
and resource discovery strategy. In fact, each LUNAR node implements a fake
DHCP server. If the DHCP client asks for a specific IP address, LUNAR will
try to resolve this address using the XRP messages. If no delivery path can
be set up to the given IP address, we assume that the address is not in use
and LUNAR grants the corresponding lease to the DHCP client. However, if
a RREP message was received, a new random IP address is picked and tested
several times before handing it to the client.

Gateways are also solicited using XRP resolution messages. In a variation
of LUNAR’s path setup described above, we allow several XRP replies to
travel back instead of a single reply. Back at the originator we collect them
and return the list of available gateways in the DHCP reply message.

4 Implementation

Figure 2 shows the software decomposition of LUNAR. In a first approach we
implemented LUNAR as a Linux user space program that interposes itself be-
tween the IP stack and the wireless card driver. The TUN/TAP device is used
to interface with the IP stack while NETLINK provides all information on
pending ARP requests in an early phase. Alternatively, one can also capture
the ARP packets once the IP stack sends them out and use ARP reply packets
to signal that a delivery path was found. However, because there is no ARP
packet type that could be used to remove an ARP entry, we still need direct
access to the ARP cache. The tight interfacing with ARP is essential for throttling the IP stack as soon as a LUNAR delivery path times out. This avoids that any (TCP) packets are lost because the IP stack will know that it has to redo an ARP before continuing with the data transmission. In order to avoid a stop-and-go behavior at the ARP interface level we let LUNAR create a new delivery path after 3 seconds in parallel to the existing one and switch silently to the new. LUNAR removes an ARP entry only if no traffic is observed for a 3 second period and the path times out.

![Figure 2: LUNAR node architecture (user space implementation).](image)

A new version of LUNAR has been development which is fully kernelized and which does not have the TUN/TAP and NETLINK dependency. This makes the deployment of the LUNAR module simpler as users do not have to recompile their kernel for TUN/TAP and NETLINK support. The kernelized LUNAR version also creates an ethernet device which is used by the IP stack like any other subnet technology. The source code is published under the GPL and can be accessed at [11].

4.1 Measurements

We measured the performance of LUNAR in a controlled setting of a linear network of three stationary nodes and one mobile node that roams along this linear network (see [12] for details). The following table gives the result for UDP traffic (ping and MP3 streaming):

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Ping</th>
<th>MP3</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLSR</td>
<td>89.0%</td>
<td>91.9%</td>
</tr>
<tr>
<td>AODV-UU</td>
<td>91.9%</td>
<td>97.9%</td>
</tr>
<tr>
<td>LUNAR</td>
<td>96.5%</td>
<td>96.8%</td>
</tr>
<tr>
<td>AODV-UU+SNR</td>
<td>99.1%</td>
<td>99.7%</td>
</tr>
</tbody>
</table>
LUNAR had better PING performance than plain AODV which is due to broadcast messages reaching farther than unicast messages: Because AODV trusts broadcast messages for identifying delivery paths, it can have problems in so called “communication gray zones” [12]. LUNAR is not subject to this problem because it does not really on HELLO beacons for link break detection. The patched AODV-UU+SNR version, which eliminates the gray-zone problem, now outperforms LUNAR also for the PING case. However, we consider the LUNAR performance of 96% to be sufficiently good for justifying not adding new protocol features.

4.2 A Tiny Multihop Ad Hoc Access Point

In order to demonstrate the reduced complexity of working below the IP layer, we fitted LUNAR and a Linux system on a single 1.4 MB floppy disk. This “LUNAR-on-a-floppy” features an automatic gateway and contains a NAT module which will map the 192.168.42.x LUNAR subnet addresses to a topologically correct IP number in the fixed Internet. As a result we have implemented a self-configuring access point which enables mobile nodes to get Internet connectivity over multiple ad hoc hops.

4.3 LUNAR for Embedded Systems

LUNAR’s simplicity also permitted us to implement the routing protocol in the very constrained environment of the LEGO Mindstorms RCX. The RCX is equipped with a h8300 Hitachi microcontroller with 32 KBytes of RAM and communicates over an infrared device running at 2400 Baud. The resulting system consists of a stripped down BrickOS (7KB), μLUNAR (3.5KB), Van Jacobsen header compression (4.5KB), μIP[5] (3.5KB), a Web server and the application (1.8KB).

Figure 3: Multihop IR coverage for mobile LEGO Mindstorm robot.

As an application we demoed the steering of an RCX-car over IP and the delivery of a dynamic HTML page displaying the unit’s status. Because the IR
device has a limited reception angle of 60°, we covered a circle shaped area with two additional LUNAR-enabled stationary RCX nodes (see Figure 3). The mobile node in the center will always be able to communicate with the console via either a one-hop or a two-hop path.

To highlight the extremely constrained environment we note here that in μLUNAR we merged the link layer address and the selector values, effectively making layer 2 becoming the SelNet layer and shrinking the Ethernet+selector address fields from 48+64 bits down to 16 bits.

4.4 LUNAR for Bluetooth Scatternets
We have developed a LUNAR version which enables us to route UDP and TCP traffic over multiple Bluetooth piconets. A Bluetooth piconet always has one master node which connects up to seven slave nodes. All communication inside a piconet passes through the master node which regularly polls the slaves. New nodes join a piconet by a lengthy inquiry process and will become slave nodes. In order to interconnect two piconets i.e., forming a so called scatternet, it is necessary that one node switches back and forth between the master role in one piconet and the slave role in the other.

Porting LUNAR to Bluetooth was not a straightforward task. Firstly, there is no broadcast functionality within piconets, all data communication is connection oriented. Secondly, not all connected nodes are able to inquire for new nodes, neither are all connected nodes fully discoverable. Currently we solve this problem by letting idle slaves temporarily disconnect from the master and look for new piconets to bridge to and then feeding this information back as if it was gathered via a broadcast mechanism. This proactive activity is necessary because of performance: A source node, which according to LUNAR is responsible for establishing the delivery tunnel and sending the data, would otherwise have to frequently disconnect and subsequently reconnects only for finding nodes in potential neighbor piconets.

4.5 LUNAR for Windows
In an effort to provide LUNAR for multiple platforms, a Windows XP/2000 version has recently been produced. It is based on the new kernelized Linux version of LUNAR and is implemented as an NDIS intermediate network driver [13]. NDIS* is an interface for network drivers used in the latest Windows operating systems. In this manner, Ethernet packets going out to or coming in from the network card can be intercepted and new packets injected in both directions. This is the main functionality that is required by LUNAR since it operates at a layer 2.5.

*Network Driver Interface Specification
The porting process was greatly simplified due to the strict modularization of kernelized LUNAR which separates the core logic from the networking and other platform specific details. Hence, the core logic is unmodified in the Windows version and the only parts that have been changed are the LUNAR netbox and platform modules. These modules respectively contain networking and various other platform specific functionality needed by LUNAR. To this end we wrote an NDIS wrapper which offers the same (Linux kernel) functionality that the LUNAR kernel module relies on.

5 Discussion

5.1 Forced Path Re-establishment

At first sight, the forced re-establishment of delivery paths after 3 seconds seems unusual and inefficient. However, it keeps up well with the actual behavior of other ad hoc routing protocols. It takes approximately 2 seconds for AODV to conclude that two consecutive HELLO messages from a neighbor were lost and that the topology changed, before AODV will start its route repair or rediscovery procedure. For LUNAR, the change in topology can occur anytime inside the LUNAR interval thus on average at 1.5 seconds. We were able to observe this faster reaction time in a qualitative way when listening to streaming of MP3 over AODV and LUNAR enabled ad hoc networks.

5.2 Formal Validation

LUNAR relies on the efficient forwarding of data packets using a simple packet format which features only a single field called the “selector”. Hence, there is no TTL field for data packets which can prevent packets from looping, should the routing protocol configure such a looping forwarding path. One safety net consists in that all forwarding entries in a node keep a “forwarding credit”, which is decremented by every packet passing through as well as a soft state garbage collection mechanism. Nevertheless, it is important that the routing protocol does not lead to forwarding loops.

In order to verify the correct operation of LUNAR we have carried out formal verifications using SPIN [6]. A LUNAR version has been implemented in the modeling language PROMELA in about 450 lines of code including topology generation. We have used this model to successfully verify the correctness of the established paths by varying over the topology.

Due to the state space explosion of this exhaustive search approach, we are so far limited to verify configurations with up to 5 nodes and belonging to selected topology classes. Figure 4 shows various topology changes that
were considered (single nodes coming and going away, and connectivity being simultaneously gained and lost at different places in the network).

6 Conclusions
We have introduced the LUNAR ad hoc routing protocol which targets the common-case of network clouds with 10-15 nodes and a diameter of up to three hops. We believe that such settings will be the most popular ones where ad hoc networks can and will be put into operation. More specifically, in larger settings and for IEEE 802.11 there are such severe degradations occurring under any ad hoc routing scheme that we do not consider this to be a relevant use case that a routing protocol should try to address.

Although the LUNAR protocol does not include any route repair or packet salvation optimizations, it offers comparable performance at less than half of the code size of other MANET protocols (despite LUNAR containing self-configuration and Internet connectivity logic). LUNAR successfully exploits the constraints assumed concerning the network size, which justifies the extremely lightweight approach.

Acknowledgements
The authors would wish to thank Henrik Lundgren and Erik Nordström for their performance comparison. We also thank David Rosén for his work in the μLUNAR project.
9. References


[11] LUNAR home pages: 


DIN'05, Köln, Germany, September 2005.

© Copyright ACM 2005

Reprinted with permission.
Janus: An Architecture for Flexible Access to Sensor Networks

Adam Dunkels, Richard Gold, Sergio Angel Marti, Arnold Pears, Mats Uddenfeldt
CNA Lab, Dept. of Information Technology,
SICS, Uppsala University, Box 1263,
SE-16429 Kista, SE-751 05 Uppsala,
Swede

We present the design and implementation of the Janus architecture for providing flexible and lightweight access to sensor network resources from Internet-type networks. Janus provides flexibility by focusing on functions of the sensor network rather than the data it contains. This allows us to perform service composition by dynamically combining functions together. In contrast to existing application-specific access techniques, Janus separates the access technique from the sensor network itself by inserting itself between the sensor network and the access network. This attribute allows for application-specific access techniques to be deployed dynamically without requiring the sensor network itself to be upgraded. Although Janus provides RPC-like semantics, unlike typical RPC systems, most of the functionality of Janus is present at the RPC client, which is located outside of the sensor network, in order to keep the RPC server inside the sensor network as lightweight as possible. We have implemented a prototype of Janus to interconnect a network of sixteen sensors in a typical environmental monitoring scenario to an Internet host.

Categories and Subject Descriptors: C.2.1 [Computer System Organization]: Network Architecture and Design

General Terms: Design

Keywords: Architecture, Sensor Networks

1 Introduction

Many wireless sensor network applications do not operate in isolation: there is a need to monitor and configure the sensor network, as well as to collect data...
sensed by the network. In some cases sensor networks are deployed in harsh or remote locations, where it is hard to physically access the sensors [10]. Being able to remotely access the sensor network not only reduces the complexity of the deployment, but can also be crucial to the long term viability of the network. One example of a sensor network deployment requiring remote access is the Great Duck Island habitat monitoring network [9]. In the words of Mainwaring et al.[9]:

""Although personnel may be on site for a few months each summer, the goal is zero on-site presence for maintenance and administration during the field season, except for installation and removal of nodes."

In this paper we present Janus, a flexible architecture for remote access to sensor networks. Janus is designed both to support existing sensor network paradigms, and to be a flexible platform for future development both in terms of new functionality and new combinations of existing functionality.

Existing approaches to remote access to sensor networks are typically application-specific [2, 36, 35, 27]. Whilst these approaches each perform well in their own right, they necessarily limit the range of interactions permissible with a sensor network. This is because these access applications have to be embedded in the sensor network itself, typically at the gateway node of the sensor network. The goal of Janus is to remove these restrictions with the following techniques:

1. Provision of a flexible signaling mechanism which supports both passive and active access approaches
2. A generic RPC-like mechanism which matches the data-centric nature of sensor nets by providing an interface to invoke functions rather than forwarding packets to destination nodes. We chose not to follow existing RPC approaches which exchange abstract data representations between a client and a server. Instead our approach uses named function invocation to remove the need for API synchronization between Janus and the sensor network.
3. On-demand setup of access to sensor network resources of varying types. Unlike current RPC approaches which require synchronization of an API before usage, Janus allows for dynamic negotiation of an API for accessing the sensor network.
4. A lightweight approach where primitive functions on a gateway to the sensor network can be combined at an external host to provide service composition. This is in contrast to existing RPC approaches where the server contains comparatively more functionality than the client. We believe that it is critical for as little functionality as possible to be present in the gateway. This is because the gateway will usually be in a location which is hard
to access physically and additionally since the gateway may be a sensor network node itself.

We have implemented a prototype version of Janus. To demonstrate the feasibility of the approach we used Janus to connect a typical sensor network to Internet hosts on a LAN. Our sensor network implements both active querying and passive event notification. A client application can request data from specific nodes, such as the sound sensor history of a particular node. Motion detectors on the sensors have been programmed to propagate an alert message to the gateway when the detected motion exceeds a given threshold. These basic functions are then combined to implement reactive data operations for regions of the network. We can also ask the network for active notification if there is a high level of activity in a particular area.

2 Architecture

Janus comprises two entities: an engine, which runs on the sink node in the sensor network, and an agent that communicates with the engine. The engine and the agent communicate using the extensible Resolution Protocol (XRP) [6], as shown in figure 1.

![Architecture overview](image)

*Figure 1: Architecture overview: (1) XRP requests sent by the agent (2) XRP replies sent by the engine.*

**Gateway**: Sits between the sensor network and the access network and runs the XRP engine.

**XRP Engine**: Provides an RPC-style interface to sensor network functionality which the XRP agent uses to discover network resources and operations provided in the sensor network.
**XRP Agent**: Exchanges XRP messages with the XRP engine to query the sensor network and to receive events

Janus uses XRP as a basis upon which to construct an RPC-style interface to the sensor network. A more detailed discussion of how XRP is used is provided in section 3.5. Participating in each Janus XRP transaction are an XRP Agent and an XRP Engine. In the remainder of the paper we refer to the XRP Agent as the *agent* and the XRP Engine as the *engine*.

The agent uses a typical RPC client semantics and sends requests for function invocation to the engine. The engine executes tasks on behalf of the agent and returns the result to the agent from which the request originated. An additional mode of operation is when the agent registers at the engine interest for a particular event from the sensor network. When this event occurs, the engine then sends an XRP message to the agent informing it of this event.

XRP permits us to specify a extensible number of parameter fields for each packet. This makes XRP expressive enough to meet our needs. The XRP packet begins with an XRP command, which in our case implementation is either a data request (DREQ) or a data reply (DREP). The XRP command parameters are of variable length and their semantics is given by a *class* field, which names the parameter and a *class-type* which defines the parameter’s data type. XRP parameters can appear anywhere inside the parameter block, thus are not bound to a specific position. We take advantage of this when we export the API of the gateway. See [6] for a more detailed description of the XRP packet format and its possible uses.

The agent and the engine exchange XRP messages with each other in order to:

1. Dynamically discover available sensor network resources. This enables the agent to ascertain which resources are available to it via the engine. This is useful in situations where the agent does not have explicit knowledge of the exact nature of the sensor network resources.
2. Send queries from the agent to the gateway concerning the state of sensor network. Examples of these involve querying the current temperature, light levels and motion sensing.
3. Send information from the gateway to the agent about the state of the sensor network. An agent can be notified by the sensor network when a specific event occurs. Examples of these include when motion is detected or when there is a significant change in temperature.

The gateway uses XRP to export the sensor network resources as functions which the agent can access. The agent can then invoke these functions by sending XRP messages to the gateway. The functions are named by *selectors*, which are opaque and uniquely map to functions at the gateway.
2.1 Supporting Multiple Sensor Network Types

Real-world sensor network deployments typically provided one of the following interfaces to either access collected data or to query the sensor network: directed data diffusion, database abstractions or passive monitoring. We discuss how each interface can be implemented with our architecture:

Directed diffusion [7] is a data-centric routing protocol for sensor networks. The protocol is based on a model in which a sink node first registers a data interest with the network. When the interest has been propagated through the network, the sensor network begins sending data in the reverse direction of the interest propagation. Directed diffusion fits well with the Janus architecture. The first DREQ message sent from the agent to the gateway causes a data interest to be registered with the network. Data from the sensor network will then be reported back from the gateway to the proxy with an DREP message.

TinyDB [8] is perhaps the most well-known example of a sensor network database abstraction. TinyDB makes the sensor network appear as a database that can be queried for data. Data queries are made using an SQL-like syntax, and data is transported through the sensor network back to a gateway node. Data aggregation is used to reduce the amount of communication in the network. The database abstraction model also works well with Janus. When the XRP gateway receives a DREQ message from the agent, a TinyDB query to be inserted into the sensor network. The results of the TinyDB query is then placed in an DREP message and sent back to the agent.

Many deployed wireless sensor networks have implemented a passive monitoring network (e.g., [2, 36]), where data is transported through a sensor network to a base station. The base station either logs the sensed data for later processing, or transmits the data to an external entity. Our architecture can be seen as a generalization of the gateway approach from such deployments.

2.2 Supporting Multiple Access Applications

In addition to supporting multiple types of sensor networks, Janus also provides support for multiple access applications. This scenario is depicted in figure 2.

The goal of using application-specific access techniques is to enable access services such as HTTP, SQL or an SMS gateway to process the information gathered by the sensor network. As this information is quite simple due to the constrained nature of the sensor network itself we do not envisage this translation to be an impossible task. The advantage of this approach is to enable a whole class of deployed applications to interface with the sensor network. This would dramatically increase the ease-of-use of interpreting the sensor network data. In our scenario described in section 1, if the beach, the field and the forest of the island are different sensor networks monitoring different as-
pects of the environment we may wish to access the sensor network resources in different ways. We may care about specific events happening on the beach and therefore wish to be notified when a specific event occurs. One such notification mechanism could be via an SMS gateway. Subsequently we may care about the average of rainfall in the forest and wish to have this information presented to us on a regular basis either via a web page or by updating a database. As depicted in figure 2, these differing sensor networks can be connected through various gateways to one agent. This agent can then provide application-specific proxies for access applications to connect to. If we take this approach, we can keep the application-specific code on an entity outside of the sensor network. This makes it much easier to upgrade or add new functionality to support other access applications than if this application-specific code is located in or near the sensor network.

3 Implementation

We have implemented a prototype which comprises of a subsection of the architecture described in the previous section. Our prototype was written in C++ under Linux and we connected it to a sample sensor network using a USB connector attached to a sink node. We have successfully run tests with the agent and the gateway + engine on separate machines and with a sensor network of up to 16 sensor nodes.
3.1 Sensor Network

Our sensor network uses a standard equipment sensor board developed at Freie Universität, Berlin. These devices have an MSP 430 microcontroller, 2KB of RAM, 8KB of EEPROM, and a wide variety of sensors. The sensors can detect motion, light, temperature, vibration, and sound.

The sensor nodes run Contiki, a small operating system for tiny devices that allows us to implement the desired functionality. Our code has been written in C.

As an example of a real sensor network, we have implemented a scenario where nodes store sensed sound data, aggregate it and send it to a sink node in reply to a query. For our purpose, we have implemented the following functions in the sensor network:

**Query current**: When the sink node asks a specific node about the current sound level in one of its sensors, the node checks the current sensor state and sends back the result.

**Query log**: Nodes consult their sound sensor periodically, compute a mean of the sound data every 5 seconds and store the result in their memory to form a set of measurements over the last minute. When a node receives a `query_log` message from the sink, it will reply with the information that it has stored.

**Event notification**: Critical and/or interesting events in the network are reported to the sink for notification outside the network.

3.2 Gateway

The gateway sits between the sensor network and the access network and contains an engine. It handles incoming connections from agents and listens to incoming events from the sensor network. When an agent connects to the gateway it can issue XRP commands, which are interpreted by the engine at the gateway. All knowledge about the sensor network is situated at the gateway and all functions used to query the sensor network are implemented here. An API at the gateway can be exported as a list of the available functions and their corresponding selectors.

3.3 XRP Engine

The engine is in charge of the agent interaction on the gateway. This interaction takes the form of XRP messages sent back and forth as seen in figure 3. When an agent has connected it begins to issue XRP commands. These commands are interpreted by the engine. The agent can discover the available
sensor network resources on this gateway by sending a DREQ for the API. The engine processes this request and composes an XRP packet with the API. This XRP packet is then sent back to the agent as a DREP message. A function is invoked on the gateway when the engine processes a QUERY style DREQ for a selector bound to a local function. The local function communicates with the sensor network and returns a result. The DREP is then built with the returned result as the payload.

3.4 XRP Agent

The role of the agent is to exchange XRP messages with an engine to query the resources of the sensor network and to receive events generated inside the sensor network. The agent needs to request the API of the gateway to be able to actively query the sensor network. After the agent has received the API it can proceed to invoke functions at the gateway in a way similar to RPC. In an extended architecture the agent could be invoked by separate access servers to provide user interaction and present data. In our implementation we attach a simple text-based client to the agent.

3.5 Signaling between Agent and Engine

All XRP messages in our implementation are transmitted via UDP between the agent and the gateway. All the functions to interact with the sensor network are implemented at the gateway. We provide both low-level functions to query individual nodes and high-level functions to allow for more advanced queries, like the mean value inside a given area.

The gateway listens to events that originate inside the sensor network and waits on incoming connections to reach its engine. Once an agent connects, it negotiates with the gateway to retrieve an API of functions to reach the sensor network. After the API has been exported to the agent, it can issue requests by invoking functions at the gateway.

The signaling is achieved using a combination of XRP DREQ and DREP messages as shown in figure 3. The figure shows how XRP packets are sent between the agent and the engine at the gateway. A selector to handle the reply is always installed prior to sending a request and we will refer to this selector as the reply selector. Each DREQ contains the reply selector along with the address of the sending node. This information is used by the gateway to target its reply.

When an agent wishes to retrieve the API from a gateway, it installs a reply selector $r$ and issue (1) a DREQ for the API to the gateway. The engine running at the gateway processes the request and responds (2) with a DREP to the $r$ selector on the agent. This reply contains the API represented as XRP
Figure 3: Signaling: The left column in the table represents function addresses and the right table column represents function names. (1,2) Exporting the API of the sensor network (3,4) Issuing an RPC by addressing a selector bound to a function together with the packet payload as an argument.
parameters inside the packet. In figure 3 we show the \textit{query\_max()} function along with its selector. This function takes an area as the argument and returns the node with the maximum sensor value. After the API has been exported, the agent can invoke functions at the gateway by addressing selectors on the gateway with a QUERY style message. The agent installs a reply selector $s$ to issue (3) a DREQ with the function argument as the payload, in this case the requested area inside the network. The engine running at the gateway processes this request by invoking the corresponding local function. The gateway issues individual requests for the sensors within the given area to find the node with the highest sensor value. The engine proceeds by building a DREP containing the returned value as an XRP parameter and sends (4) this to the reply selector at the agent.

4 Discussion

We now discuss some of the issues that arose whilst implementing Janus. We discuss some of the features that XRP has, show how Janus can support existing sensor network approaches and present some ideas for future work.

4.1 XRP Properties

The XRP protocol has the following properties which are crucial for providing flexible access to sensor networks: \textbf{Expressiveness}: XRP messages need to be interpreted in order for the correct functionality to be executed. By using the XRP protocol, a query is made for some information or data from the sensor network. This XRP query is then processed by the engine in order to return the appropriate result. This process of interpreting the XRP query is how XRP provides expressiveness. Rather than using a self-describing packet format, XRP performs signaling to set up functions on the engine that perform the tasks that the agent has requested. The reason for this design choice is that it is easier for a signaling protocol to express a new piece of functionality through its instruction set than a packet header format to do likewise. \textbf{Modularity}: By inserting a flexible architecture between the two different networks, it is possible for both networks to fundamentally change as long as they adhere to the API that Janus provides. Alternative access applications may be deployed on-demand to meet new system requirements. For example a sensor network which provides, via a gateway, an SMTP access method may wish to additionally deploy an SMS gateway to provide event notification. Different types of sensor networks are useful for different scenarios and it is our intention to not restrict how the sensor network is implemented, but rather to use Janus to allow them to be used in a consistent manner.
The usefulness of these properties can be explored through the discussion of invariants [1] in network architectures. Invariants are pieces of an architecture that cannot be changed without stopping an architecture from functioning correctly. The authors of [1] claim that all network architectures contain invariants, a claim that we agree with. Examples are the IPv4 address in the Internet and the SIM card in mobile phones. Since the access application and sensor network in our scenario are potentially quite disparate and variable, we believe that we should not chose either one to be the invariant for our system. We propose that Janus which is inserted between the access applications and the sensor network should be the explicit invariant of our system. We note here that this does not mean that the functionality remains static, but rather that the XRP protocol format is adhered to.

4.2 Future Work

A promising direction for future work is to completely integrate Janus with the Contiki Operating System to provide an active networking platform for sensor networks. Our goal is to use the Janus function table to store programs which have been shipped in XRP packets. The function table presented in figure 3 shows pre-loaded programs being accessed by XRP queries. We are currently implementing an XRP interpreter and a function table for the sensor nodes used for the implementation presented in this paper. We will then build the functionality to allow programs to be sent inside XRP packets and subsequently loaded dynamically into the Janus function table running inside a sensor network node. Through our work with microLUNAR [11] we know that it is feasible for an XRP interpreter and a function table to be implemented in less than 3.5KB.

5 Related Work

Existing solutions for querying networks such as SNMP provide a standard interface for accessing the resources of a network. However the expressiveness of SNMP is a concern. It provides a get/set operation for a Management Information Base (MIB). This provides the ability for both event-driven and on-demand types of query/responses from the sensor network. However, it is not straightforward to send specialized queries into the sensor network from outside of the sensor network. With SNMP, we are limited by what information and events that are defined in our MIB at the time of deployment. Furthermore, we cannot see a clear migration path from SNMP to an active network platform as discussed in section 4.2. SNMP could potentially be useful for querying a sensor network if we could make the assumption that the the gate-
way node to the sensor network will never be a sensor network node itself. Since SNMP uses ASN.1 for representation, it is not suitable for implementation in severely memory-constrained devices. From our work on LUNAR for embedded systems [11] we know that it is feasible to implement an XRP parser in an embedded environment.

Similar concerns about implementation in constrained environments apply to existing RPC-style systems such as SUNRPC, Java RMI, CORBA and WSDL. The representation systems used such as XDR or XML were not designed for devices which are extremely memory-constrained such as sensor network nodes. For the Janus approach where we wish in the future to support the gateway to the sensor network being a sensor network node and also to migrate the Janus function table into the sensor network itself, we must be mindful of the resource consumption of the tools that we choose.

Delay Tolerant Networking (DTN) [5] is a network architecture that is designed for environments with intermittent connectivity, scheduled transmissions, and possibly long propagation delays. DTN provides a convergence layer abstraction that allows running DTN both on top of TCP/IP and a sensor network network protocol. DTN can be used for connecting sensor networks and the Internet [4] by inserting a DTN gateway at the interconnection point, but can also be used within the sensor network.

Buonadonna et al. [3] use a special purpose gateway called the Sensor Network Appliance (SNA) running Linux. The SNA has a radio module that can communicate with the wireless sensors, an Ethernet connection, as well as support for satellite or cellular connectivity. The SNA runs an ODBC-compliant database management system and a HTTP server that both can be used for remote access of the sensor network. This is different from our approach, in that the Janus places the user interface—the HTTP server and database management system—at a proxy rather than on the gateway. This makes modifications to the interface easier after deployment of the sensor network. Janus also enables new services to be added after deployment simply by setting up additional agents and/or access applications.

6 Conclusions

This paper presents Janus, an approach to providing flexible access to sensor networks. Janus allows clients to access sensor network resources without requiring detailed knowledge of the sensor network implementation structure. The approach is lightweight, flexible and extensible. We have demonstrated its ability to support Internet client interaction with a prototype sensor network implementation for both simple and complex operations on the target sensor network. We have found our initial implementation to be promising and plan
to integrate several different types of sensor networks as well as several access mechanisms in the near future.

Acknowledgment

This work was partly sponsored by SSF, the Swedish Strategic Research Foundation, VINNOVA, the Swedish Agency for Innovation Systems, and the European Commission under contract IST-004536-RUNES.
10. References


Paper E
Janus: An Active Middleware for Accessing Sensor Networks

Richard Gold                     Björn Ahlström                     Mats Uddenfeldt
rmg@it.uu.se                     bjorn.ahlstrom@gmail.com           mats@uddenfeldt.se

Dept. of Information Technology, Uppsala University
Box 337, SE-751 05 Uppsala, Sweden.

One of the research challenges in sensor networking is how to access the resources of the sensor network from a remote location i.e., collecting information and reacting to events from the sensor network. Although access mechanisms already exist, they are typically application-specific and thus inflexible. This reduces the potential life-time of the sensor network by limiting how we can change it after deployment. We may wish to upgrade the sensor network after deployment due to changing application requirements and other unforeseen circumstances. Sensor network deployments in harsh or remote environments make upgrading very difficult. We present an architecture called Janus which comprises of an extensible middleware for interfacing with a sensor network and a code deployment platform which can be run inside of a sensor network. We model the sensor network as a collection of functions which provides us with a natural way to extend its functionality by adding more functions. We have implemented the middleware and tested it against a sample real-world sensor network and have implemented the code deployment platform in the Contiki network simulator.

1 Introduction

Sensor networks are an attractive solution to many remote monitoring applications such as environmental, habitat and intrusion monitoring scenarios. In order for a sensor network to be a viable solution to monitoring a harsh or isolated environment where it is hard to physically access the sensors, there must be some technique to access the resources of the sensor network remotely. How a sensor network is accessed depends a great deal on how the sensor network is represented i.e., how it appears to the outside world. A typical approach is to provide a database abstraction of the sensor network [1]. However, we chose to model the sensor network as a collection of functions. We call this approach function-orientated. This provides us with a natural way to extend the capabilities of the sensor network as a new piece of functionality can be represented as one more function in the sensor network.

The motivation for extending a sensor network’s functionality through a remote mechanism is that once a sensor network is deployed, changing it man-
ually through physical access is often very difficult. This is because there may be many sensors, we may not actually know where these sensors are located and physical access to the location where the sensors are may be infeasible. We may wish to upgrade the sensor network to patch bugs in the deployed system, the existing software or even add new functionality to the system to take into account changing application requirements. One sensor network deployment which is used for habitat monitoring is the “Great Duck Island” system [2] which is located on an island off the coast of Maine, USA. The bird colonies on this island are extremely sensitive to disturbance and researchers are typically only on-site during the summer months. This means that upgrading a sensor node or network without a remote mechanism is extremely difficult. How to achieve remote upgrading is therefore the focus of this paper.

We present an architecture called Janus∗ which comprises of an extensible middleware for providing access to the sensor network resources from an external network (previously presented in a workshop paper from the authors) and a code deployment platform which runs inside the sensor network. We acknowledge that security is a major issue with such a code deployment approach. We imagine that existing security tools for active networks could be applied to our architecture as well. For example: Proof Carrying Code (PCC) [3], programming language support [4] or a more holistic approach [5]. The contribution of this paper is the demonstration of how a sensor network can be represented as a collection of functions and how this approach allows us to deploy code in the sensor network and access it externally in a uniform way. Uniformity comes from Janus using the same protocols inside the sensor network as it does outside the sensor network. Since we wish to use the same protocol both inside and outside of the sensor network, it is a requirement that the protocol is lightweight, this makes it difficult to reuse existing signaling protocols such as SIP [6]. The goal of our approach is to provide as much freedom as possible to the sensor network designers as to how they wish to implement their solutions. We do not present new protocols or algorithms for data diffusion or aggregation inside of the sensor network, but rather present an architecture where it is possible to deploy new functionality both inside and outside the sensor network in a straightforward manner. Whilst there already exist work on Active Sensor Networks [7] and Mobile Agent Middleware [8] which provide remote code deployment to a sensor network, our approach differs in three main ways: 1. we use a function-orientated approach to model the sensor network i.e., we represent the sensor network as a collection of functions, 2. we do not use a virtual machine such as Maté [9] and 3. we provide a uniform query/response mechanism to interact with the middleware outside of the sensor network as well as the code deployed in the sensor network. We have implemented the middleware portion of Janus on an example sensor net-

∗Janus is the two-faced Roman god of doorways
work and the code deployment portion of Janus in the Contiki [10] network simulator.

The rest of the paper is organized as follows: we first present the architecture of Janus and our implementation work. We then discuss some related work and describe our ideas for future work. We close with our conclusions.

2 Architecture

The design of the Janus architecture is presented in figure 1. Janus comprises of three entities:

1. The eXtensible Resolution Protocol (XRP) which the engine and the agent use to communicate with each other.
2. an XRP engine (henceforth referred to as the engine), which runs either on the sink node in the sensor network or in the sensor network itself.
3. an XRP agent (henceforth referred to as the agent) that communicates with the engine.

![Figure 1: Janus architecture. Janus itself is comprised of the XRP Agent, the XRP Engine and the XRP signaling protocol. The example HTTP, SMS gateway & SQL servers are collectively known as Front Ends servers.](image)

The agent and the engine exchange XRP messages with each other in order to:

1. Dynamically discover available sensor network resources. This enables the agent to ascertain which resources are available to it via the engine. This is useful in situations where the agent does not have explicit knowledge
of the exact nature of the sensor network resources. This is also used to
discover new functions which have been deployed dynamically into the
sensor network.
2. Send queries from the agent to the gateway concerning the state of sensor
network. Examples of these involve querying the current temperature, light
levels and motion sensing.
3. Send information from the gateway to the agent about the state of the sensor
network. An agent can be notified by the sensor network when a specific
event occurs. Examples of these include when motion is detected or when
there is a significant change in temperature.
4. Push functions into the sensor network which can then be used as described
in 1. and 2.

The gateway uses XRP to export the sensor network resources as functions
which the agent can access. The agent can then invoke these functions by
sending XRP messages to the gateway. The functions are named by selectors,
which are opaque identifiers and uniquely map to functions at the gateway.
When a new function is added to the sensor network, its selector is added to
the nodes in the sensor network and/or to the engine so that it can be accessed
in the same way as existing functions can.

2.1 Function Placement

One of the key design decisions of the Janus architecture is to attempt to have
as much choice as possible about where to place functionality. We note here
that, for us, a sensor network system is not just the sensor network itself but
also the mechanisms that allow the sensor network to be manipulated from
an external location. In our architecture, functionality can be deployed in the
following places:

Front Ends : The front ends are the part of the Janus architecture which in-
terface directly with the user. As show in figure 1, examples of which
are HTTP servers or SMS gateways. Although they are not exactly part
of the Janus architecture they still need to be able to interface with it. A
new front end can be added to a deployed system by adding the ability
to process the application-specific front end protocol to the agent.

Agent : The Agent is typically located outside of the sensor network and is
therefore suitable to be upgraded as it does not have the typical sen-
sor network deployment restrictions of power consumption, processing,
memory or connectivity. We imagine the Agent to usually reside on a
commodity PC or equivalent. Since the Agent is located outside of the
sensor network, we believe that upgrading this single entity manually to
be a straightforward task.
**Engine** : The Engine is located at the edge of the sensor network. In certain deployments we can imagine the Engine residing on a lightweight PC or PDA where access to power is not an issue. Other deployments, such as those in animal habitats, may not have this luxury. In such situations we imagine the engine to run on the sensor network node which is functioning as a sink. Adding functionality to the Engine dynamically from a remote location is a crucial part of Janus. This is described in section 3.3. The same approach applies for every node in the sensor network including the sink.

### 2.2 XRP – eXtensible Resolution Protocol

The eXtensible Resolution Protocol (XRP) defines a presentation format for the internal Janus messages. It permits to specify a variable and extensible number of parameter fields. The parameter format resembles the encoding of RSVP objects [11]. An XRP message is a sequence of XRP commands. Each command is introduced by a 32-bit command header:

```
+-----------+----------+----------+
| 1 | XRP command | ttl     |
|    |            |  reserved |
+-----------+----------+----------+
```

followed by zero or more command parameters. Command parameters have the format:

```
+----------+----------+----------+
| 0 | length (in bytes) | class | class-type |
|    |                     |       |           |
+----------+----------+----------+
| ... contents ... |
|             |
+----------+----------+----------+
```

The XRP command parameters are of variable length and their semantics is given by a *class* field, which names the parameter and a *class-type* which defines the parameter’s data type. XRP parameters can appear anywhere inside the parameter block, thus are not bound to a specific position. See previous work by the authors for a more detailed description of the XRP packet format and its possible uses. XRP is used by Janus to communicate between the agent and the engine as well as inside of the sensor network itself. Janus uses XRP as a basis upon which to construct an RPC-style interface to the sensor network.

### 2.3 Agent

The role of the agent is to exchange XRP messages with the engine to query the resources of the sensor network. The agent exchanges XRP messages with the engine in order to dynamically discover available sensor network resources, to send queries to the gateway concerning the state of the sensor network and to send or receive information from the gateway about the events.
Figure 2: The client sends queries to the agent (1) which will translate these into XRP commands to an engine (2). When the reply has been acquired from the sensor network the engine will reply (3) with an XRP message that is translated and sent to the client (4).

in the sensor network. Figure 2 shows a client that has connected to an agent in order to send queries to the sensor network. The agent acts as the translation point between the client and the engine to provide access to the sensor network resources.

Janus uses dynamic discovery of sensor network resources by letting the agent negotiate available services with the engine on the gateway. This is particularly useful in situations where the agent does not have explicit knowledge of the exact nature of a sensor network. It enables the agent to find out which resources are available to it via the engine. One of the main advantages of the Janus architecture is that the same agent can be used to access different sensor networks without being forced to implement this explicit application-specific knowledge, as shown in Figure 1, as long as the same API is implemented at the engine. After the API has been negotiated between the agent and the gateway the agent can issue queries to the sensor network through the gateway in a way similar to the RPC client-server model. This is done by sending a request XRP messages over the network targeting one of the selectors exported with the API. Examples of these requests involve querying the current temperature, humidity levels and motion sensing. Through the interface offered by the gateway it is also possible for an agent to register an interest for a particular type of event within the sensor network. The agent will then be notified when this specific event occurs. Examples of these events include when motion is detected anywhere inside the network or when there is a significant change in temperature in a given region. The agent can be invoked by separate front ends to provide user interaction and present data to existing clients such as web browsers.

2.4 Engine
The engine is in charge of the agent interaction on the gateway. This interaction takes the form of XRP messages sent back and forth between the agent
and the engine. When an agent has connected to the gateway it begins to issue XRP commands. These commands are interpreted by the engine. The agent can discover the available sensor network resources in the gateway by sending a request for the API to the function module on the gateway. The engine processes this request and composes an XRP message with the API represented as special XRP parameters. This XRP message is then transmitted back to the agent. This message contains a list of the selectors bound to the individual functions of the function module on the gateway.

Functions are invoked on the gateway when the engine processes an XRP message targeting one of these selectors in a way similar to the RPC-style invocation. When a local function is triggered it will communicate directly with the sensor network to retrieve the answer to the posed question, build a reply message containing this answer and transmit it back to the agent. In some sense the engine can be viewed as a proxy interface to the sensor network. This enables us to provide a uniform interface to the sensor network by enabling us to abstract away the internals of the sensor network if required.

The engine is also capable of installing newly defined functions dynamically which are sent via XRP messages. Subsequently these functions can then be referenced in the same way as existing functions.

2.5 Gateway

The gateway sits between the sensor network and the access network and contains an engine as seen in Figure 3. Its role is to handle incoming connections from agents in the access network and to listen to incoming events from the sensor network. When an agent connects to the gateway it can begin to issue XRP commands, which are interpreted by the engine at the gateway.

For every sensor network accessed with the Janus architecture there would have to be at least one gateway. This is to provide the sensor network with an

Figure 3: An XRP message comes in from an agent in the Janus network (1) and are received by the gateway which hands it over to the engine for processing. The engine invokes a packet processing function which queries the sensor network (2). The reply comes back from the sensor network (3) and is passed on to the agent (4).
ingress / egress point in which it can be accessed. If the need arises to create a gateway for a previously unsupported sensor network only the access and function modules of the gateway risk to be coded. In other words it would be necessary to implement an access method to be able to communicate with the network and functions which can be used to query the sensor hardware on the actual sensor nodes. Due to the dynamic nature of the Janus architecture adding a new sensor network type like this would not warrant any changes outside the gateway. If we decide to upgrade our nodes with a new type of sensors, we would only have to change the functionality module and could leave the access module intact.

The functionality module contains all the functions which can be used to query the sensor network. These functions can be exported using XRP to allow an agent to access and query the sensor network. Simplified the API is represented as a list of the available functions and the corresponding selectors used to invoke them. The list is implemented using the special API-style XRP messages and is described in more detail in Section 3.4.

3 Implementation

We have implemented a prototype as a proof-of-concept which comprises of a subsection of the architecture running against a sample sensor network. We stress here that this is just an example of how a sensor network could be implemented, there are many different ways that this could actually be performed. Our prototype was written in C++ under Linux and we connected it to our sensor network using a USB connector attached to a sink node. We have successfully run tests with the agent and the gateway + engine on separate machines and with a sensor network of up to 16 sensor nodes.

3.1 Agent

In our implementation a client connects to the agent using Berkeley TCP sockets. To be able to offer the client the ability to actively query the sensor network the agent must first request and parse the API of the gateway in a way explained in Section 3.2. The agent can also register an interest to receive event notification from the sensor network by sending an XRP SUBSCRIBE message to the gateway. An agent can either subscribe to events of a given verbosity (e.g., silent, critical, verbose, very verbose) or to specific events (e.g., motion in a given area). The engine will then add the agent to its list of subscribers and inform it when an event takes place.

In our current implementation the location (i.e., IP address) of the gateway is statically defined, but in the Janus architecture there would ideally be some
kind of service discovery to locate gateways in your vicinity. This could be done using broadcasted XRP resolution requests in a way similar to the one described in previous work by the authors.

3.2 Engine

The agent can discover the available sensor network resources on this gateway by sending an XRP message containing a QUERY for the API to the gateway. This process is shown in Figure 5. When our implemented engine receives this request it will respond with a pre-built XRP message containing a DATA REPL Y with the API. After this initial handshaking the agent will be able to invoke functions at the engine by sending QUERY style messages for the selectors listed in the API. When this happens the QUERY is demultiplexed in the engine and a local function invoked with the data payload as a function argument. In our implementation the function argument is either the address of a node (position 1-16 in a grid) or the alias of an area (e.g., beach, field, forest and selected) inside the network. The local function communicates with the sensor network and returns a result. An XRP DATA REPL Y message is then built and sent back to the agent.

3.3 Code Deployment Platform

![Diagram of the Janus Code Deployment platform]

*Figure 4: The Janus Code Deployment platform: The left-most boxes are elements of the Contiki OS. The right-most boxes are the Janus components. <SEL> is a selector and Appctrl is the function which manages the installation of other functions.*

Our code deployment platform (shown in figure 4), which is designed to be run on a sensor network node, uses the engine described above. Currently we
have an implementation of the code deployment platform running in Contiki network simulator and we are planning to port it to our example sensor network platform. The code deployment platform interfaces with the Contiki OS at the link layer i.e., before the network layer processing takes place. Janus packets are diverted from the typical packet processing path as show on the left-hand side of Figure 3.5 and into the Janus function table. Currently our implementation provides two main services:

**Multi-hop ad-hoc routing and forwarding** : This functionality is provided through the μLUNAR system that we have ported to Contiki. This allows us to communicate point-to-point over multi-hop inside of the sensor network if we need to communicate with a specific node rather than a function for debugging and testing purposes. Future application scenarios may even require such a communication approach.

**Code deployment** : Code deployment using μLUNAR to forward code-carrying packets through the network. We broadcast XRP messages into the network which carry a piece of code to be installed and define which area we wish them to be valid in. Contiki currently divides the sensor network into a geographic area addressable by \( x \) and \( y \) co-ordinates. This is useful for reprogramming a certain part of the network. The code is then installed at the appropriate locations in the network and a selector is returned to the source (usually the sink, although this is not necessary) which allows this newly installed functionality to be invoked.

Our current unoptimized version of the code deployment platform is 8KB in size and we expect that this size can be further reduced.

### 3.4 Signaling between Agent and Engine

All XRP messages in our implementation are transmitted via UDP between the agent and the engine residing on the gateway. All the functions to interact with the sensor network are implemented at the gateway. We provide both low-level functions to query individual nodes and high-level functions to allow for more advanced queries, like the mean value inside a given area.

The gateway listens to events that originate inside the sensor network and waits on incoming connections to reach its engine. Once an agent connects, it negotiates with the gateway to retrieve an API of functions to reach the sensor network. After the API has been exported to the agent, it can issue requests by invoking functions at the gateway.

The signaling is achieved using a combination of XRP DREQ (Data Request) and DREP (Data Reply) messages as shown in figure 5. The figure
Figure 5: Signaling: The left column in the table represents function addresses and the right table column represents function names. (1,2) Exporting the API of the sensor network (3,4) Issuing an RPC by addressing a selector bound to a function together with the packet payload as an argument.

shows how XRP packets are sent between the agent and the engine at the gateway. A selector to handle the reply is always installed prior to sending a request and we will refer to this selector as the reply selector. Each DREQ contains the reply selector along with the address of the sending node. This information is used by the gateway to target its reply.

When an agent wishes to retrieve the API from a gateway, it installs a reply selector \( r \) and issues:

1. a DREQ for the API to the gateway. The engine running at the gateway then processes the request
2. The engine subsequently responds with a DREP to the \( r \) selector on the agent. This reply contains the API represented as XRP parameters inside the packet. In figure 5 we show the \( \text{query\_max()} \) function along with its selector. This function takes an area as the argument and returns the node with the maximum sensor value. After the API has been exported, the agent can invoke functions at the gateway by addressing selectors on the gateway with a QUERY style message. The agent installs a reply selector \( s \) to issue
3. a DREQ with the function argument as the payload, in this case the requested area inside the network. The engine running at the gateway processes this request by invoking the corresponding local function. The gateway issues individual requests for the sensors within the given area to find the node with the highest sensor value. The engine proceeds by building a DREP containing the returned value as an XRP parameter
4. The engine then sends this to the reply selector at the agent.
3.5 Sample Sensor network

The sample sensor network used in the prototype implementation was built on sensor boards developed at Freie Universität in Berlin as shown in figure 6. Technical details regarding the sensor boards can be found at the Scatterweb ESB home page [12]. The sensors can detect motion, light, temperature, vibration and sound. The sensors nodes run Contiki [10], a small operating system for tiny devices which allowed us to implement the desired functionality in C. We have implemented a network where nodes store sensed sound data, aggregate it and deliver it to the edge of the network in response to individual queries. The sensors also notice if they are moved and this special event is then automatically reported. Connectivity is supported by attaching a USB serial cable to the sink node.

4 Related Work

There is relevant related work in the areas of introducing active networks and mobile agents to sensor networks and access techniques for sensor networks.

4.1 Active Network and Mobile Agents in Sensor Networks

There has been recent work on Active Sensor Networks [7] and Mobile Agent Middleware [8]. Active Sensor Networks present an extension to the Maté [9] virtual machine system to provide application-specific virtual machines (ASVMs) which are virtual machines customized for specific tasks. The focus of this work is to provide an appropriate programming abstraction for programming sensor networks. Janus does not use a virtual machine approach, instead relying on the programming environment provided by
the Contiki OS. Additionally, the ASVM approach does not provide a middleware which provides an abstraction of the sensor network. We are interested in providing a complete architecture for accessing sensor networks. Mobile Agent Middleware provides a mobile agent platform called Agilla which enables agents to proactively crawl the sensor network and spawn to appropriate locations to execute. The Agilla approach uses a tuple space on the sensor network node which the mobile agents can use to communicate and store data. We do not wish to provide such a detailed abstraction with our work, but rather provide the support to deploy such systems as Agilla which provide a certain set of tools to the sensor network designer.

4.2 Sensor Network Access Techniques

TinyDB [1] is perhaps the most well-known example of a sensor network database abstraction. TinyDB makes the sensor network appear as a database that can be queried for data. Data queries are made using an SQL-like syntax, and data is transported through the sensor network back to a gateway node. Data aggregation is used to reduce the amount of communication in the network. Whilst TinyDB provides some hooks to extend its SQL-like syntax, the extensibility of the system is not its fundamental goal. We envisage TinyDB being one of many systems that could be deployed through the Janus approach.

5 Future Work

We wish to explore the following topics in future work: Firstly, performance profiling and power consumption are extremely important to the viability of Janus. We need to quantify the impact of our architecture on the operational efficiency of the sensor network. This will enable us to understand the trade-offs inherent in deploying such a system. Secondly, security is a major issue for active networking approaches and for sensor networks. Without comprehensive answers to these questions, we expect sensor networks to remain a purely research issue, limited to a few experimental deployments. Existing research in active network security [3, 55, 26] point out a set of directions that we should pursue. There also exist proposals for security in sensor networks which should also be carefully examined [13].

6 Conclusions

We have presented Janus, an architecture for accessing sensor network resources which comprises of an extensible middleware located outside of the
sensor network and a code deployment platform which resides inside of the sensor network. Our architecture provides a uniform query/response API to access the sensor network resources. Uniformity is achieved by using the same protocols both inside and outside of the sensor network. Our goal is to have a system for providing access to sensor network resources which is both extensible and lightweight. The system needs to be extensible so that it can be upgraded and modified during deployment without physical access to the sensor network itself since many typical sensor network deployments are in harsh or remote conditions. How lightweight the system is, is also a concern as sensor networks typically are extremely resource-constrained and do not have access to external power sources. Janus presents a view of the sensor network as a set of functions which can be invoked on-demand or can trigger responses to events. We have implemented a prototype of our architecture and successfully run tests which demonstrate typical sensor network scenarios.
11. References


We weep for the bird’s cry, but not for the blood of a fish. Fortunate are those who have a voice.

Motoko Kusanagi