



UPPSALA UNIVERSITY

Architectures for Application Transparent Proxies: A Study of Network Enhancing Software

Björn Knutsson

A Dissertation submitted
for the Degree of Doctor of Philosophy
Uppsala University, Information Technology
Department of Computer Systems

May 2001

Dissertation for the Degree of Doctor of Philosophy in Computer Systems presented at Uppsala University in 2001.

ABSTRACT

Knutsson, B., 2001: Architectures for Application Transparent Proxies: A Study of Network Enhancing Software. DoCS 01/118. 190 pp. Uppsala. ISSN 0283-0574.

Proxies, software deployed inside the network, play a fundamental role in the Internet by providing enhanced functionality to the network. Deployment of proxies is a flexible way of extending the Internet architecture with new services and to cope with problems that were not foreseen at the time the original Internet protocols were defined. The creation of the Internet is an enormous investment in time, effort and money, and proxies allow us to build on the existing infrastructure to enhance its functionality, rather than replace it.

As the use of proxies increase, so does the problem of proxy configuration and deployment, especially with respect to interference between different proxies. With a limited number of different proxies, this problem can be dealt with manually, or by encoding knowledge of interfering proxies into each proxy. As the number of proxies grow, methods to automatically detect and cope with conflicts must be devised. Therefore, proxies need to coordinate with each other. Towards this end, a signalling protocol that can be used to establish and configure a sequence of one or more proxies along an end-to-end flow is proposed. The protocol is realized as an extension of IP, using an IP option, which simplifies its deployment in the Internet.

In order to facilitate reasoning about coordination, conflicts and deployment of proxies, a model has been developed. The model is based on the concept of regions, also developed in the thesis. Regions are interconnected parts of the network that share a common property, e.g. administrative control or error characteristic. Along with the model, a classification of proxy architectures with respect to how they gather information and deploy proxies is presented. A method based on this model is also proposed.

We also propose an algorithm for controlling compression to maximize perceived throughput in situations where available bandwidth and CPU power varies. Along with the algorithm, experimental results that show that the algorithm approximates the best non-adaptive choice in a number of situations are presented. This algorithm has been implemented as an end-to-end enhancement.

Björn Knutsson, Department of Computer Systems, Information Technology, Uppsala University, Box 325, SE-751 05 Uppsala, Sweden. E-mail: bjorn@docs.uu.se

© Björn Knutsson 2001

ISSN 0283-0574

Printed by University Printers, Uppsala 2001.

Distributor: Department of Computer Systems, Information Technology, Uppsala University, Box 325, SE-751 05 Uppsala, Sweden.

Acknowledgements

The path to a PhD is long and winding, and not without its pitfalls and seemingly impassable obstacles. Fortunately, I have had an excellent guide in my primary advisor, professor Per Gunningberg, who besides guiding me, also have allowed me room to grow as a researcher. Besides being my advisor, I have also come to think of him as a good friend and mentor, and I hope he will remain so even after I leave his research group. My deepest thanks.

I would also like to thank the other members of the research group at Uppsala University: Mats Björkman, my secondary advisor. We collaborated on the first project I was involved in after becoming a PhD student and he helped me get started. He is also the co-author of the first two papers in this thesis, and have spent many hours proofreading it. Jochen Schiller, who spent a year as post-doc with our group during my first year as a PhD student, and who became my friend. Bengt Ahlgren, lab manager at SICS and former PhD student in the group, who soon after I joined the group proved that it *is* possible to graduate, and who also responsible for awakening my interest in Scotch single malt whisky. Bob Melander, Thiemo Voigt and Henrik Lundgren, my PhD student colleagues. Christian Tschudin, associate professor and the group's Swiss connection. Arnold Pears, associate professor and a good friend outside of work, to whom I also extend my gratitude for reading and commenting on this thesis.

My thanks also go to Larry Peterson of Princeton University for hosting me for a seven month visit in 1999, and for our collaboration on proxy architectures. Larry is co-author of one of the papers in this thesis. In his group at Princeton, I also thank Scott Karlin, Wenjia Fang and especially Andy Bavier, for making my stay there a memory for life. Thanks also to Andy for proofreading this thesis.

I would also like to thank all the nice people at the Department of Computer Systems. I have especially had occasion to interact with the department's system administrators, and I'm thankful to Per, Joel, Ylle and Anders for their help and patience with all the experiments I've subjected their networks and systems to. Special thanks also to Annika Nostell, who have often gone out of her way to help me.

The path to a PhD should not be all work, and therefore I extend my thanks to colleagues and friends Gaffe, Göran and Mic, not only for their help with my work, but also for wonderful dinners, game evenings and other fun activities.

Kristina Lundqvist has been a good friend, and I am deeply thankful for her help, support and friendship. As a PhD student colleague, she has helped me with my work, and as a friend outside of work, she has helped me relax from it.

Finally, to Anna Sandberg, my fiancé and soon-to-be wife, I give my love and deepest thanks for standing by me through this work.

This work has been sponsored by the European Union Long Time Research project HIPPARCH, Ericsson CSLAB/SARC and Ericsson Research.

Included papers

This thesis includes, summarizes and discusses the results presented in six papers, listed below. These will be referred to as Paper A—F.

- Paper A** Björn Knutsson and Mats Björkman, *Trading computation for communication by end-to-end compression*, in Proceedings of the Third International Workshop on High Performance Protocol Architectures (HIPPARCH'97), Uppsala, Sweden, June 1997.
- Paper B** Björn Knutsson and Mats Björkman, *Adaptive End-to-End Compression for Variable-bandwidth Communication*, in Computer Networks, volume 31, issue 7, 1999.
© 1999 Elsevier Science B.V.
- Paper C** Björn Knutsson and Per Gunningberg, *Exokernels, Protocol implementation and Erlang*, Technical report DoCS 103, Department of Computer Systems, Uppsala University, Sweden, 1999.
- Paper D** Björn Knutsson and Larry L. Peterson, *Transparent Proxy Signalling*, Accepted for publication in the Journal of Communications and Networks, March, 2001.
© 2001 KICS
- Paper E** Björn Knutsson and Per Gunningberg, *Automatic Proxy Configuration*, The Workshop on Mobile Multimedia Communication (MoMuC'2000), Tokyo, October, 2000.
© 2000 IEEE
- Paper F** Björn Knutsson and Per Gunningberg, *Automating Proxy Configuration and Placement*, Not yet submitted for publication.

Reprints with permission from the respective publishers.

Comments on my participation

In all the papers listed, I have been the primary contributor. Rather than list my contributions, I will instead list the contributions of others:

For the work presented in papers A and B, the ideas evolved in continuous discussions with Mats Björkman, who also co-edited the papers.

Per Gunningberg has discussed the ideas and co-edited paper C. The programming parts of the port of Erlang to XOK discussed in paper C was done by a MSc student under my supervision.

The work presented in paper D grew out of the signalling used for the work in Paper B, and was largely done in cooperation with Larry L. Peterson of Princeton University, during my visit there.

For papers E and F, the bulk of the work was done by myself, as an extension and continuation of the work presented in paper D, but with continuous discussions with Per Gunningberg, who also helped editing the papers.

Other work

Apart from the work included in this thesis, I have also authored or co-authored and presented the following papers:

Björn Knutsson and Mats Björkman, *Adaptive End-to-End Compression for Variable-bandwidth Communication*, Fourth International Workshop on High Performance Protocol Architectures (HIP-PARCH'98), London, Great Britain, 1998. (Paper B is a revised version of this paper.)

Björn Knutsson, *Exokernels + Erlang: Exploring novel operating systems architectures to increase performance*, the Seventh Swedish Workshop on Computer Systems Architecture (DSA'98), Gothenburg, Sweden, 1998.

Björn Knutsson, *Increasing Communication Performance via Adaptive Compression*, the Seventh Swedish Workshop on Computer Systems Architecture (DSA'98), Gothenburg, Sweden, 1998.

Björn Knutsson and Mats Björkman, *Adaptive Compression for Wireless Communications*, First workshop on Wireless Broadband Testbeds (DEMO'98), Berlin, Germany, 1998.

Björn Knutsson, *Proxies and Signalling*, First Extensible Router Workshop, Princeton University, Princeton, USA, 1999.

Most of the contents of these papers and presentations are included in the papers presented in this thesis.

Contents

1	Introduction	1
1.1	Problems With A Growing Internet	1
1.2	Where to Address the Problems	3
1.3	Thesis Research Areas	6
1.4	Contributions	8
1.5	Research Method	9
2	Summary of the Papers	11
2.1	Paper A: <i>Trading computation for communication by end-to-end compression</i>	11
2.2	Paper B: <i>Adaptive End-to-End Compression for Variable-bandwidth Communication</i>	12
2.3	Paper C: <i>Exokernels, Protocol Implementation and Erlang</i>	13
2.4	Paper D: <i>Transparent Proxy Signalling</i>	13
2.5	Paper E: <i>Automatic Proxy Configuration</i>	14
2.6	Paper F: <i>Automating Proxy Configuration and Placement</i>	15
3	Background and Related Work	16
3.1	Proxies	16
3.2	Compression	22
3.3	Exokernels	24
4	Conclusions	26
A	Trading Computation for Communication by End-to-End Compression	33
B	Adaptive End-to-End Compression for Variable-bandwidth Communication	35
C	Exokernels, Protocol Implementation and Erlang	37

D	Transparent Proxy Signalling	39
E	Automatic Proxy Configuration	41
F	Automating Proxy Configuration and Placement	43

Chapter 1

Introduction

That the Internet is growing has become a truism. This thesis deals with one approach to coping with the growing Internet—deploying programs, called proxies, into the network. But let us stop for a moment and examine the growth of the Internet. It has developed from a way of giving researchers without access to local computing resources access to off-site mainframe computers, to become almost as ubiquitous as the telephone service.

As the range of services provided on the Internet increases, along with the bandwidth they consume, the telephone modem becomes less and less attractive as the means of connecting to the Internet. New ways of connecting that offer mobility and more bandwidth are becoming more and more popular. The options range from telephone network ISDN to dedicated cables and wireless solutions, both terrestrial and satellite, giving a wide dynamic range of bandwidth and transmission latency and different approaches to mobility.

An increasing number of households use the Internet for banking, mail order, ticket booking and other services previously handled by mail, telephone or by visiting brick and mortar stores. Additionally, many companies use the Internet to provide an enhanced service only available on the Internet.

To allow us to access these services everywhere, we want the Internet to have near universal coverage. It also means we will need to be able to use the Internet wherever we go. We do not want to carry bulky laptops around, which implies that small hand held devices like Internet-capable mobile phones will be more attractive.

1.1 Problems With A Growing Internet

The growth of the Internet has opened new areas where the Internet can be used, but this has also opened up new problem areas that the Internet was not originally designed to cope with. Below we will examine four important problem areas.

1.1.1 Increased Diversity

We are seeing an increasing diversity both in how Internet hosts are connected to the network, and their capabilities. However, the Internet was not designed for a diversity of this kind. As an example,

TCP, which is the basis for much of the communication over the Internet, does not cope well with networks that lose or garble messages [8], but wireless networks have a tendency to do just that [24]. One way to solve this problem is to replace (or enhance) TCP. However, since this would require TCP to be globally replaced, this is not a very practical solution.

Similarly, most information on the World Wide Web is presented under the assumption that a fairly large color display is used to view it, which is not the case with Internet connected mobile phones. This problem can be handled by convincing Web designers to provide versions of web pages for devices with small displays. This is happening for some sites, but is far from universally deployed. Web pages can also be adapted by the device itself, but since many devices with small displays are also bandwidth limited, this would waste bandwidth, and increase latency.

1.1.2 Service Differentiation

The number of services available on the Internet is steadily growing. One that is still in its infancy, despite having been pushed for many years, is multimedia services. Telephone and video over the Internet is possible today, but the quality is often less than satisfactory. The problem is that these types of services need predictable and bounded latency, and the original Internet protocols were not designed for these services. If you are watching a movie, you will not accept that it stops or stutters while your neighbor downloads a web page.

As operators deploy network infrastructure in larger areas, it is reasonable to believe that they will want to use them to deliver telephony and cable-TV services in competition with the traditional operators of these services. To be able to provide these services, even just inside their own networks, differentiation between types of traffic is needed to give preference to latency sensitive traffic.

1.1.3 Address Space

The address space of IPv4 is running out since more networks and devices are being added than was anticipated when it was designed. In many cases, Internet Service Providers have more customers than they have IP addresses. If all ISPs give one to each customer device, the address space would be exhausted. Instead, most ISPs do not assign permanent addresses to their customers. The addresses are instead assigned dynamically when the customer's device connects to the ISP's network.

This assignment scheme works when most customers used dial-up modems and there is a limited number of modems at the operator. Each modem is then assigned an IP address, and when the maximum number of users are connected, no new users can connect. With broadband access to homes and flat-rate connections, more people will leave their computer connected around the clock, tying up IP addresses. For the Internet's growth to continue, this must be handled. IPv6, the new version of IP, provides a much larger address space, but deployment of IPv6 has been slow.

1.1.4 Security Issues

Commercial interests, often with no prior background in networking, are moving into the Internet arena with new requirements on the network, especially in the area of security. Banking and e-commerce on the Internet require that account numbers, PINs and passwords cannot be intercepted,

and similarly that fields in payment orders cannot be modified by a third party.

The obvious solution to this problem is to extend applications with support for security features, as has been done with e.g. web browsers. There are, however, a great number of applications that deal with sensitive data, but have not yet been extended with security features. They have to rely on the underlying network to provide security.

For offices spread out geographically, it is important that trade secrets and internal information does not leak to outsiders when information is passed between offices in different locations. One costly solution to this is to order a dedicated connection between the offices. This, however, assumes that the provider of the connection can be trusted.

Finally, even if a company network is limited to a single location, it is important that outsiders cannot gain access to restricted information or resources on the inside (intranet), except to the extent the company is willing to allow such access.

1.2 Where to Address the Problems

Examining the Internet architecture, we find that the choices of where to solve these problems without going outside of the architecture are limited. The two most prominent ways of generally dealing with these type of problems are to create new protocols or services, or to modify the existing. This may, however, involve large scale upgrades or deployment of new software, making this type of solution impractical for many cases. The layered architecture itself also limits what can be done within the architecture.

Applications can normally be changed without affecting the architecture at all. However, making changes to the application requires the cooperation of the software vendor, and may also require that the peers run the same version of the software.

A solution in the application layer also means that the application must be able to deal with all types of network conditions¹ that can occur, not just the ones known to occur for the particular access networks being used. A related point is that the need for an enhanced service is not always obvious to the application, e.g. when packet losses are due to bit errors and not congestion. This makes it hard for the application to know what enhanced functionality to activate.

Transport and application layer solutions are similar in many ways. The advantage of changing or adding a new transport layer protocol is that it can be done once and applies to all applications. This is also a disadvantage, since this puts even higher demands on the solution to work under all network conditions, but also with all applications, which contradicts the arguments for Application Level Framing [18]. Additionally, the problem of upgrading all systems is much more difficult, since the transport protocol is typically part of the operating system.

The network layer is the hardest layer to change, since it affects both end systems and network equipment like routers. The slow deployment of IPv6 is an example of how hard it is to add a new network protocol.

Link layer solutions have the advantage of being specific to a certain link, allowing them to be tailored to a specific medium and hop characteristic. The disadvantage of a link layer solution is that unless

¹Combinations of bandwidth, latency, bit error rates etc.

support for service differentiation exists in higher layers, it will affect all traffic going over the link, meaning that application or flow specific problems cannot be addressed at all.

This thesis argues that a better, or at least more convenient way, to deal with a number of these problems is to insert proxies into the network.

1.2.1 Proxy Solutions

Proxies can be added to the network either under the control of the application and application protocol, or independent of the application.

The most common way to add software under the control of the application is by attaching a new server to the network that clients connect to for an enhanced service, e.g. HTTP caching. This is referred to as *application proxies*.

The second option, adding software independently of the end points, is the focus of this thesis. This type of software will be referred to as *application transparent proxies*, or in this thesis, just *proxies*.

Application transparent proxies can be deployed in a variety of ways. The simplest, and still most common, way is manual deployment, i.e. the proxy is explicitly installed on a node in the network. In more recent work, like the MARCH [33] and ICEBERG [65] frameworks, deployment is automated.

Proxies can be deployed by a variety of entities, and for a variety of reasons. A user can deploy proxies on his system, the access network operator can deploy proxies in the access networks and the core network operator in the core network. Additionally, content providers can deploy proxies on e.g. server farms in the network.

In most cases, proxies deployed by an entity will only be deployed within that entity's administrative domain. For some types of proxies, there are, however, many benefits from cooperation across administrative domains, and part of the work in this thesis has the objective of facilitating such cooperation.

The problems outlined in Section 1.1 can be solved in the following ways by adding proxies to the network:

Increased Diversity

Dealing with diversity is one of the most common proxy tasks. It is generally done by hiding the discrepancy between the expectations or needs of the end points, and the actual network conditions.

The problem of diversity in error rates can be dealt with by inserting proxies into the network that mask the errors. In the case of TCP, if the lost or garbled messages can be repaired or resent in such a way that the TCP end points never notice that an error occurred, TCP will continue to work efficiently [7] and will not trigger congestion control mechanisms.

Web pages can be filtered through a program inside the network and the important information in them be extracted and adapted to fit the small display of a hand-held device [66].

Service Differentiation

Service differentiation can be implemented using proxies that examine the IP datagrams and interpret the packet payload to find out what service and flow a particular packet belongs to. The packet can then be tagged for the appropriate service.

The differentiation between different types of traffic classes can be provided by inserting QoS upgrades into the network that give preference to real time multimedia traffic over best effort traffic [11, 20], or by means of reserving network resources. Additionally, other programs can be inserted to improve the utilization of the available bandwidth and give the users better performance. This can, for example, be done by caching or mirroring data close to the user, i.e. keeping local copies of data, and by compressing data, i.e. removing redundancies in the data².

Address Space

The address space problem can generally be dealt with by allowing many end systems to use a single IP address, which implies that some mechanism must exist to route traffic based on both IP address and higher layer identifiers, e.g. TCP or UDP port number.

A typical solution is to put a Network Address Translator [61, 64] into the gateway connecting the local network and the Internet. Essentially, the NAT gateway is a proxy that translates internal IP addresses to one external IP address, and lets all machines on the local network share the address of the NAT gateway. Since addresses are translated in the gateway, both the internal and external networks can be built using standard IP routers and end systems, only the gateway needs to know about the address translation.

Security Issues

Proxies can be set up to encrypt communication between legacy applications that do not provide their own security functions. Depending on the setting and the requirements, this can be done either by proxies specific to certain applications, or by using IPSEC [43, 63] to encrypt all traffic.

Virtual Private Networks (VPN) [31] provide a virtual link connecting different offices via the Internet by sending traffic between offices through encrypted tunnels, providing both the illusion of a single network spanning all offices, as well as protecting the traffic when going between offices. The VPN router is a proxy that takes packets destined for another office, encrypts them and encapsulates them into IP datagrams that are sent over the Internet to the VPN proxy in the other office.

Another example of security proxies is firewalls, that can similarly be deployed in the gateway between the local network and the Internet to both protect the network from intruders on the outside as well as blocking access from the inside to selected hosts.

²Destructive compression will not just remove redundancies, but also reduce informational content.

1.3 Thesis Research Areas

Deploying proxies in the network is often easier than solving the problem by other means, but this does not mean that the deployment of proxies is without problems in itself.

One of the biggest concerns with deploying programs into the Internet in this fashion is that it does not fit very well into the traditional Internet-model. In this model, the network should have limited intelligence, essentially limited to forwarding packets (for efficiency reasons), while most intelligence lies with the end points. This means that there is very limited support for controlling programs deployed inside the network.

More importantly, there is no support for controlling *how* these programs are deployed, nor determining what effect, if any, they will have on each other and the rest of the infrastructure. The lack of architectural support for deploying software in the network means that the very existence of such software must be hidden from the existing protocols, since they do not know how to deal with them. The reverse is also true—since the existing protocols do not know about the software deployed in the network, they cannot interact with the proxies to help them.

This means that software deployed in the network will have to deduce state information relevant to the operation of a proxy from packet contents. As an example, a proxy that selectively operates on a specific application protocol must be able to recognize that this protocol is in fact being used. If the proxy interacts with this protocol, it may also need to be able to deduce the states of the end points for correct operation. One way to do this is by monitoring the flow between the end points and use knowledge about the behavior of the protocol used to deduce what state the end points should be in. For cases where packet content is not enough to deduce the state information, heuristics can be used. Heuristics may also be used instead of a complete state machine to reduce the complexity of the proxy.

For example, if the intention of a proxy is to adapt images in web documents to a small display, then a simple heuristic may be to watch the flows passing by, and intercept and re-code any images it detects. However, unless this proxy can also recognize what protocol is being used, it would perform this operation for all image files being transferred for all applications. This is undesirable, some applications do not want recoded images. By extending the heuristic to only trigger the proxy for special flows e.g. HTTP-flows, the number of images converted in error would drop. But since HTTP also can be used to fetch files, a better heuristic would be to only convert images that are part of an HTML-document, and fetched because of an `IMG`-directive.

Research issues in the general area of proxies include various types of proxy functionality as well as architectural issues. One architectural issue is how to direct the flow to the proxies using the existing routing protocols. The work presented in Paper D has a partial solution, but that solution rely on certain common topological features of the Internet that is not necessarily present in all networks. Other issues that are not addressed by this thesis is how to deal with crashing proxies, proxy architectures for mobile systems and security.

The architectural considerations of proxies and proxy deployment is the major topic of this thesis, and is discussed in Papers D, E and F.

Papers A and B deal with a method for controlling how compression is used. Paper C is an investigation of how the communication performance of machines connected to the Internet can be improved, i.e. to make them able to send and receive data faster. The earlier papers (A–C) concern problem areas that are not specific to proxies or proxy architectures, but the results are relevant to proxies.

The specific research problems that I have worked on and that are presented in this thesis are the following:

1.3.1 Proxy Interference

When a proxy service is deployed into the network to enhance the service provided to a flow, it is different from other entities in the network in that it is not formally a part of the network model. It often needs to deduce the state of the sender and receiver, since no part of the packets in the flow it operates on are intended for the proxy.

A single proxy service that is deployed for a flow will be able to operate using deductions based on normal flow behavior. However, if a second proxy service is deployed that alters the flow in any way, this means that the assumption of a normal flow behavior will no longer be true for the first proxy service. If the proxy no longer can correctly deduce the information needed for correct operation, the proxy will no longer work as intended. This can cause effects ranging from a degradation of the service the proxy service delivers, to a complete disruption of the flow. Unless either proxy becomes aware of the other, and takes appropriate measures, this problem cannot be prevented from happening. As more and more proxies are deployed in the network, the risk of interference increases.

Paper D presents an architecture for proxy detection and signalling that exposes the presence of proxy services to each other, thus allowing them to take steps to avoid conflicts.

1.3.2 Proxy Configuration

When a single entity deploys all proxies, the interactions between proxies can be analyzed off-line, and any interaction problems solved before deployment. When multiple entities are deploying proxies independently of each other, this is not possible. Instead, information about the intended configuration must be communicated, either directly to all proxies, or to an entity responsible for configuring proxies. With this information, the problem of proxy interference can be approached.

If the only available information is the order and type of proxies, the configuration problem can only be handled using the stored knowledge a proxy instance has of potential conflicts with other proxy services. In the case of predicted interference, there are few options except to not run to avoid causing interference. It also means that when faced with an unknown proxy service, it cannot determine if running will cause interference or not.

Just finding out that another proxy service is in operation helps, but if the deployment of proxies can be controlled, many situations where interference might otherwise occur can be avoided by shuffling the order of the proxies being deployed.

The first embryo of a scheme that is not based on explicit knowledge of interference is suggested in Paper D, and an outline for a scheme is presented in Paper E. A more refined version of a proxy configuration and deployment model is presented in Paper F.

1.3.3 Controlling Compression

As the Internet growth continues, we can expect to see the Internet being used in more and more diverse environments. The lower end of this spectrum will include low and very low bandwidth devices.

Compression of data, i.e. removal of redundancies, is often used to improve bandwidth utilization and perceived bandwidth for situations where low bandwidth is a problem. Compression will essentially trade CPU resources for a higher perceived bandwidth by providing better utilization of the available bandwidth. In most cases, the more CPU power applied to compression, the better the compression ratio will be. A better compression ratio means better bandwidth utilization. However, if too much time is spent compressing data, the rate at which compressed data is produced will fall below the rate at which the network can transmit data, and the perceived bandwidth will drop. This can happen either if the bandwidth available in the network increases, or the CPU power available for compression decreases. To avoid this, the compression algorithm will often be configured conservatively, to allow for fluctuations in bandwidth and CPU power, leading to sub-optimal bandwidth utilization and thus lower perceived bandwidth.

The trade-off between compression and bandwidth is explored in paper A and an algorithm that dynamically adjusts the effort spent on compression to match available bandwidth and CPU power is presented in Paper B.

1.3.4 Server Capacity

The amount of bandwidth available in the wired core network is increasing. As the increase in bandwidth migrates outwards to the end points, servers will have an increasingly hard time keeping up with the network.

This problem can be handled in a number of ways. We have explored ways of creating server operating systems and server software that utilizes the hardware better. Others [47] have approached this problem by putting gateway proxies in front of clusters of servers, and distribute the load between them while maintaining the illusion of a single server.

Paper C explores various ways to create better protocol implementations using the low-level access provided by the MIT Exokernel [25]. Looking back on this work with an eye towards proxies, we also note that having operating systems that provide good communications performance and flexibility will be important for the machines that will run proxies in the network.

1.4 Contributions

My scientific contributions are:

- A demonstration of the problem of interfering proxies and that mutually invisible proxies cannot avoid interference, an example of a proxy signalling scheme that makes proxies visible to each other and allow them to avoid interference, and a classification of different types of proxy architectures that helps in understanding and reasoning about their proxy deployment strategies.
- A model for reasoning about proxy deployment, proxy interference and proxy configuration based on the concept of “regions”, parts of the network sharing some common property, and a method for configuring and deploying proxies that can detect, and sometimes resolve, interference between proxy services, without requiring prior knowledge of the interference.

- A method for controlling compression in a way that will automatically balance the available CPU resources against the available bandwidth. In short, it adapts the compression effort to the currently available resources to maximize perceived bandwidth.
- The demonstration through empirical evaluation that the same mechanisms that allow greater flexibility and performance in Exokernels, downloadable code, come with a severe performance penalty when the nature of the function implemented with downloadable code requires that the downloaded code be updated frequently.

1.5 Research Method

The research presented in this thesis has been pursued mainly with an experimental approach. The research process starts with the formulation of an hypothesis, based on observations of an existing system. This is followed by reasoning about the observation and related information to conclude if the hypothesis can account for the observation and related information. Once a candidate hypothesis has been formulated, some tests of the hypothesis are set up (testing single properties of the hypothesis) to test the consistency and coherency of the hypothesis. The hypothesis will then either be abandoned, or modified to account for the results of these tests. Paper A is almost entirely concerned with these steps, and the result of the paper is a hypothesis which has been partially validated.

The next step is to design experiments that test the full implications of the hypothesis. One important aspect of this step is to ensure that the experiments actually test the properties of the hypothesis that the experiments were set up to test, and to keep all other properties constant. Designing experiments is an iterative process, where results are evaluated, and the experiments progressively refined until the confidence in the experiments ability to test the deciding properties of the hypothesis are deemed satisfactory.

When designing experiments, various factors that distort the results must be controlled, or considered and accounted for in the evaluation of the results, and this is especially true of the act of observing the system. Controlling the environment is important to be able to reason about the effects of the different factors in the environment.

In the experiments in Papers B and C, there is a trade-off between instrumenting the environment to capture fine grain detail versus the impact that the instrumentation has on the results. The distortion of the results caused by instrumentation and other means of observing the system must be carefully analyzed, and if possible measured, in order to quantify the confidence in the results.

There is also a trade-off between conducting full-scale experiments, and scaling down the experiment to isolate individual factors. Scaled down experiments can be achieved in two ways, either by building a prototype that does not have all functionality, or by running in an isolated lab environment where external factors can be controlled. Both types of scaled down experiments require that the relevance of the experiment is verified.

The experiments in Paper C were mostly done on stripped down prototypes of the situation that caused problems, while the experiments in Paper B were run on the full implementation in a lab environment.

A major problem with experiments performed on “live” systems is repeatability. The approach taken in Paper B was to run the documented experiments in an controlled environment to achieve repeatability, but running validity experiments in a real network environment to verify that the behavior was

generalizable to “real” or “live” situations. For the experiment in Paper C, the real system was successively stripped down to isolate the interesting problem.

When working with models, scaled down systems or in isolation, it is important to avoid systematic errors. These errors will often not be immediately visible, since being systematic, they will occur in the same way in all experiments. When the hypothesis is tested in a real system the error will be noticeable.

For the work in the latter papers (D–F), only small parts of the system described has been implemented. The method has therefore excluded large scale experiments. Instead a more analytical approach has been taken to complement and reason about the problem. Instead of actually running the experiments in a real environment, the hypothesis has been used to construct a model that is then tested on paper (or in simulations) against the known behavior of the environment. We must reason about the model and the results, and by reasoning show that the hypothesis is sound. Finally, from the experiments, observations and reasoning we have synthesized abstractions and generalizations of the results. In the case of Paper F, this results in the abstract properties and behavior of proxies.

Chapter 2

Summary of the Papers

This section contains a summary of the papers included in this thesis. The order of the papers is the chronological order in which they were written, rather than the order in which they were published, to better show the evolution of the work presented in them.

Paper A laid the foundation of the work presented in Paper B. The settings used in these papers served as an inspiration for expanding the scope to deal with an environment in which multiple proxy services operated, as explored in Paper D. The signalling mechanism developed in Paper D was also inspired by the mechanisms developed in these early papers. Paper D also introduced both the terminology and the problem area that is the subject of Papers E and F.

Paper C shares little context with the rest of the papers in this thesis, although the motivation for the work is similar.

2.1 Paper A: *Trading computation for communication by end-to-end compression*

This paper was written within the context of the HIPPARCH [21] project. A goal of the project is to produce a communications framework that increases communications performance, especially in low-bandwidth wireless mobile systems.

One obvious way to improve communications performance is to compress data before it is sent. As outlined in the previous chapter, when using compression the problem is to know how much effort to put into compression to maximize the perceived bandwidth. An additional problem is at which layer compression should happen.

The paper argues that tuning the level of compression effort requires that three factors are taken into consideration in every situation: The available bandwidth, the available CPU power and the type of data being transferred. To a user, the actual bandwidth utilization is secondary to the perceived bandwidth, or “performance” of the system. Experiment results presented in the paper show both how much can be gained by picking the best compression effort, and also that picking the wrong compression effort may hurt performance.

The paper also argues that it is beneficial to compress data on a per flow (end-to-end) basis, rather

than on a per-link basis. The argument can be summarized in the following points:

- Keeping data compressed over multiple links consumes less bandwidth and thus increases throughput on these intermediate links.
- If multiple links along the path of a flow use compression, this means that multiple compress/decompress cycles will occur, wasting CPU resources and introducing latency.
- Compression performance will usually increase if compression is done on data from a single context, rather than a mixed set of unrelated contexts.

The key contributions of the paper are an investigation showing that selecting the right compression effort is important, and an outline of a method for automatically adjusting the compression effort to match the current resource situation.

In retrospect, the two issues—end-to-end compression and compression adaption—should have been kept separate since while the arguments for doing end-to-end compression have merit, they are in no way linked to the problem of adapting the compression effort. All the arguments for compression effort adaption are just as valid if compression is done at the link level.

The paper was presented at Third International Workshop on High Performance Protocol Architectures (HIPPARCH'97) held in Uppsala, Sweden, June 1997.

2.2 Paper B: *Adaptive End-to-End Compression for Variable-bandwidth Communication*

This paper builds on the framework provided by Paper A. An algorithm for adapting compression is presented and evaluated.

The algorithm was implemented and inserted into the Linux kernel TCP implementation to allow it easy access to information about buffer use, and to allow it to use TCP options to negotiate about the use of compression. Various experiments were made to study how the algorithm behaves in different resource situations. The conclusion is that it consistently finds a good level of compression effort, and also that it will not slow down communication in situations when the compression algorithm cannot produce compressed data as fast as the network can transmit it.

The key contribution is the algorithm and the evaluation that shows that the algorithm behaves as expected in a variety of resource situations. While explored in the context of TCP in this paper, the same mechanism can also be used in link-level compression, e.g. PPP or any other situation where a queue exists.

In retrospect, it was not the best decision to integrate compression into the kernel. At the time, it made sense in order to give easy access to kernel variables during the development, but a better strategy would have been to modify the kernel to provide the needed information about buffer use, but move the compression code out of the kernel. This was later done when the algorithm was implemented as a proxy in the MARCH framework [33].

The paper was published as an article in *Computer Networks*, volume 31, issue 7, 1999.

2.3 Paper C: Exokernels, Protocol Implementation and Erlang

The work presented in Paper C was done under a contract with Ericsson CSLAB/SARC. The objective was to examine how the experimental Exokernel [25] operating system architecture could be used to increase the performance of large legacy programs. We did this by porting the Erlang [4] system to the Exokernel, and analyzing it with a special focus on protocol implementations.

During the benchmarking of the ported Erlang system, a performance anomaly in the benchmarks was discovered that was traced to the ExOS [51] Unix-emulation implementation of the `select()` call, causing this particular benchmark to take 118% more time than in a comparable UNIX system. This call is implemented using a wake-up predicate, which is the Exokernel way of letting processes wait for events. Wake-up predicates are implemented by downloading code into the kernel that checks the specified memory locations for specified values. However, downloading and installing the code is a relatively expensive operation. In communication code, `select()` is a common operation meaning that code is repeatedly downloaded and installed, causing the anomaly.

We suggested a two ways to deal with the specific problem of implementing `select()`:

- Changing the wake-up predicate to use interpreted code, which would be cheaper to install, but more expensive to run.
- Install all tests once, and toggle them on and off with variables.

We believe that the underlying problem is inherent in the Exokernel architecture. The paper also suggests several ways that the flexibility of the Exokernel can be used to improve the Erlang runtime-system and protocol implementations written in Erlang.

The key contributions of the paper are the observations about wake-up predicates and the mechanisms for improving the Erlang runtime system. A more tangible result was the port of the Erlang system to the MIT Exokernel (also described in [10]). Additionally, a zero-copy networking interface was introduced into the Erlang system as a result from the discussions between our group and the Erlang developers during this project.

Looking back, our timing for this project was unfortunate. Work on the Exokernel at MIT was winding down at the time our project started, and by the time the project was coming to an end the effort at MIT to continue developing the Exokernel had stopped, and a company spun off to exploit the ideas commercially. Today even this company seems to have disappeared.

We do, however, believe that many ideas from the Exokernel will survive. The mechanism of downloadable code is a powerful one, as is the general idea of exporting low-level interfaces. The kind of flexibility that Exokernels provides would be an excellent platform for implementing proxies.

This paper was published as a technical report DoCS 103, Department of Computer Systems, Uppsala University, Sweden, 1999.

2.4 Paper D: Transparent Proxy Signalling

The work on this paper begun during my 7-month visit to the Network Systems Group at Princeton University's Computer Science department.

The aim of the work presented in the paper was to find a way of coordinating proxies to avoid interference between proxy services. In the process of working on this problem, the problem of how to reason about proxies and proxy deployment was encountered.

The paper begins by introducing the concept of Regions (parts of the network that share common properties) as a tool for reasoning about proxies and proxy deployment. It goes on to define the problems that motivate a unified proxy detection and signalling scheme and the properties such a scheme should have. One such scheme is presented, with a few examples illustrating how the scheme works. The paper also introduces the problem of proxy configuration.

The key contributions are the Region concept and the proxy interference problem formulation. The scheme and the formulation of the proxy configuration problem are also important contributions.

The proxy detection and configuration scheme was initially intended for TCP only. It used TCP options and was later modified to use IP options and to handle any type of flow it can identify. In retrospect, this should have been done from the beginning. Using IP options has one big advantage: Existing routers are already required to handle IP options, whereas looking for TCP options would require them to be changed. Additionally, this avoids the problems with TCP options in non-SYN packets [39, sec 1.3].

The paper has been accepted for publication in the Journal of Communications and Networks Special Issue on Programmable Switches and Routers, March, 2001.

2.5 Paper E: *Automatic Proxy Configuration*

This paper grew out of the discussions about proxy configuration in Paper D but also in response to the need for proxy configuration in other proxy architectures.

In this paper the problem of configuring and deploying proxies to avoid interference is explored. The aim was to find a way to do this that does not require explicit knowledge about interferences between specific proxy services.

The paper presents a configuration method that requires proxy services to describe the effects they have on their environment in terms of abstract properties. These properties can then be used to determine if two proxy services will interfere, and to derive interference-free configurations. The use of properties means that proxy services can be added to the proxy architecture independently of other proxy services.

The key contribution of the paper is the idea of using properties to avoid requiring explicit knowledge about interference between specific proxy services. The set of properties suggested in the paper gives an idea of what kind of properties that are needed.

This paper was written as an extended abstract describing work in progress, and was presented at the Workshop on Mobile Multimedia Communication (MoMuC'2000) held in Tokyo, Japan in October 2000.

2.6 Paper F: Automating Proxy Configuration and Placement

This paper is the result of the continuation of the work presented in Paper E, and the problems addressed essentially the same.

The paper refines both the concept of regions and the set of candidate properties for describing proxy services and proxy hosts, and describes how an algorithm that uses these properties and regions to derive proxy configurations must behave. A number of proxy services are described using the properties.

The paper goes into more detail than papers D and E about the problems of proxy interference and the milder version of this problem, proxy conflicts (which we define to be non-disruptive conflicts between proxy services). The paper examines how various types of conflicts and interferences can be detected using the set of properties and the regions created by the deployment of proxy services. Additionally, a classification of proxy architectures is presented that helps in understanding and reasoning about proxy architectures and deployment strategies.

Key contributions of the paper include the improvements to the Region concept and the terminology for reasoning about proxy configuration and proxy architectures. Other contributions included are the discussion about and description of the proxy configuration problem, the classification of proxy architectures and, although not finalized, the set of properties. The configuration algorithm presented in the appendix is intended as a proof-of-concept, to show how the abstract properties can be used to derive interference-free configurations.

This paper is as yet unpublished.

Chapter 3

Background and Related Work

In this chapter, some background that will help in understanding the work presented in the papers will be presented, along with comments on related work.

3.1 Proxies

Proxies and proxy architectures are the topics of Papers D–F. This section presents some background information and discussions about these topics, as well as related work. Some related work will be elaborated on in the background sections, and then summarized in the related work section.

3.1.1 Background

Proxies are diverse objects. Beyond being programs that exist inside the network, there are few properties that are common to all proxies. Most proxies do, however, belong to one of a limited number of sub-classes of proxies. The recent Internet Draft on Performance Enhancing Proxies [13] categorizes a number of such sub-classes, and provides a means of classification based on layering, symmetry, transparency and other properties. The draft is mainly concerned with proxies that interact with TCP, but most concepts can be applied to a wider range of proxies.

Proxies can be seen as *ad hoc* additions motivated either by deficiencies of the networking architecture or end points, or by artifacts naturally occurring because of bandwidth limitations, propagation times, latency or cost factors.

The rest of this section will consist of elaborations on a few proxy-related subjects.

Proxy Architectures

The architecture of a proxy consists of two major components, the part which implements the service of the proxy, e.g. FEC, and the part that interfaces the service with the network and integrates the proxy with its environment. The former part is, naturally, completely dependent on the service being implemented. The latter part, however, is more interesting to study, since it deals with essentially the

same type of problems, regardless of the service implemented, and can thus be categorized according to its characteristics.

If a proxy is designed, constructed and deployed to counter one specific problem, chances are that it is not a part of an existing proxy framework, or its architecture is *ad hoc*. If, on the other hand, it is constructed to be part of a larger framework, it has to be consistent with the other components of the framework. One function of such a framework is to decide how to disseminate the information about proxies needed for configuration, activation and placement.

Examples of such frameworks are Protocol Boosters [27], the ICEBERG [65] and MARCH [33] projects. M-TCP [15] can be seen as an early framework that supports multiple proxy services, but support them in a single point. One example of a proxy that has been implemented outside a framework is the Snoop [7] proxy.

Proxy architectures can be classified along two deployment axes—the location axis and the time axis. The location axis deals with how the information needed for proxy deployment is disseminated—top-down or bottom-up. The time axis deals with when and how proxies are deployed—a priori or on-demand.

The top-down model of proxy information dissemination is perhaps the most attractive for a single organization wanting to deploy proxies. In this model, the organization maintains full control over proxy deployment. Examples include the MARCH and ICEBERG frameworks. The bottom-up model essentially means that the configuration information flow is initiated by the proxies in the network, which implies a less centralized and less controlled environment. The model we outline in Paper D is a bottom-up model.

What we refer to as on-demand deployment resembles Active Networking [62, 3, 59, 68] in that proxies are not already in place and ready to run, but must be downloaded, or at least activated. With a priori deployment, the proxy is expected to be in place, and possibly already running, as is the case with HTTP proxies, NAT-gateways [61, 64] and Firewalls.

We do not expect proxy architectures to be completely polarized according to these criteria, but rather that they will exhibit more or less of these types of behavior. The model that is the result of combining Papers D–F will be bottom-up, but support a hybrid of a priori and on-demand proxies, while ICEBERG and MARCH would be characterized as top-down and on-demand. A network with just a NAT-gateway/Firewall could be said to be part of an ad hoc solution, and the best characterization would be bottom-up and a priori.

Proxies and the End-to-End Principle

One of the bigger concerns with the proxy approach to solving problems in the Internet is that it violates one of the principles that has guided the design of the Internet—the end-to-end principle [54]. This principle essentially states that anything that can be handled by the communication end points should be handled by them. David Clark, one of the formulators of the end-to-end principle, recently co-authored an article [19] that argues that in general, the move away from this principle may be unwise.

The article, however, also recognizes that there are cases where this move may be warranted, for example when the network is indeed the best place to put such functionality. The standpoint taken in this thesis is that the end-to-end principle is a attractive and should be followed when reasonable.

There are, however, many situations where the network simply is the best place for the functionality. Another argument is the difficulty and cost of end point modifications.

In RFC2775 [16] the effects on transparency of proxy-type functionality in the network is discussed, also in terms of the end-to-end principle. The conclusion of the document is that while some proxies have “*intrinsic value*”, other proxies “*must be costing the industry very dearly in constant administration and complex fault diagnosis*”. The position taken in this thesis is that this assessment is correct, and that the cause for these problems is the lack of architectural framework for proxies. By providing such a framework, the administration can be reduced and fault diagnosis simplified.

The end-to-end principle [54] and the argument for application level framing [18] are important for proxy work, since even though some proxies will break these principles, they force the proxy designer to perform a careful analysis of the cost/benefit of the proxy to motivate every violation.

Proxy Relations

Most proxies are constructed with specific application or transport level protocols in mind. This means that these proxies are aware of how the protocol works and can deduce, based on the protocol behavior, the sender and receiver states and make modifications to the flow accordingly.

There is generally nothing that prevents two proxies from operating on the same flow, with either identical or different purposes. Since both proxies may operate in isolation, i.e. without knowledge of each other’s presence, they have no choice but to assume that the flow they see is the flow from the original sender. When this assumption is wrong, proxies may interfere with each other.

Proxy Conflicts

We define a conflict between proxies to be a situation where the function of one proxy will nullify or reduce the service provided by one or more other proxies. By this definition, in a proxy conflict, the end-to-end flow will not be disrupted or altered in such a way as to cause problems for the end points.

At first glance, it would seem that applying the same proxy service twice on the same flow would be a fairly clear cut case of a conflict. Imagine, for example, if we have a proxy-pair that compress data between themselves. If we deploy another pair of compression proxies between the two original proxies, as in Figure 3.1, these “inner” proxies would only be able to operate on a previously compressed flow, which should not be possible to compress further¹.

If, on the other hand, the two compression proxy-pairs were applied sequentially, as in Figure 3.2, then there would be no conflict. (One could argue, as in Papers A and B, that it would make more sense to run only a single proxy-pair, like Compression 1 in Figure 3.1, in this case, since this would eliminate one decompress/compress stage, but this is an optimization.)

For some types of non-paired proxies, running the same proxy twice also makes sense. One could imagine deploying two TCP Snoop [7] proxies if there were two independent wireless hops along the path of the flow, while a distilling [29] proxy that reduces the number of colors used in images should

¹There are cases where this is not strictly true, but typically the size reduction is small. One can do a simple experiment with `gzip` [22]: Compress the same text file with `gzip -1` and `gzip -9`. Then try `gzip -9` on the compressed files. For many files, the added compression achieved by this second compression stage is <1%, even though the difference in size between the `gzip -1` and `gzip -9` compressed version of the same original file can be on the order of 10–15%.

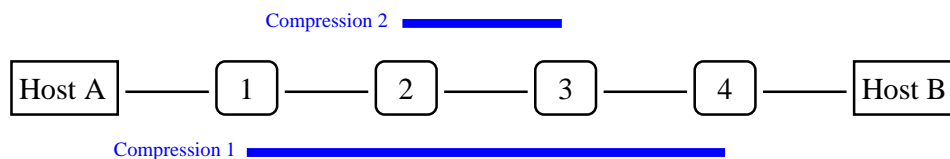


Figure 3.1: Compression 2 is applied “inside” Compression 1, and is essentially nullified. This happens whenever link-layer compression is applied to a previously compressed flow.

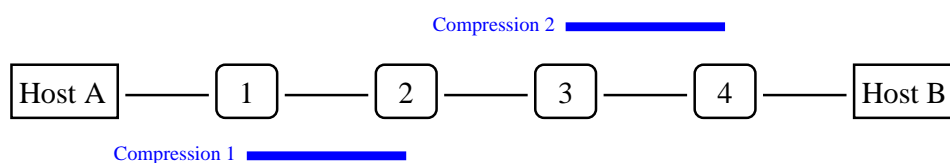


Figure 3.2: With Compression 1 and Compression 2 applied sequentially, there is no conflict.

probably not be applied more than once, except in multicast scenarios where different receivers have different needs.

The case when two different proxy services are running is even more difficult to deal with than multiple instances of the same proxy services, and the situation depends completely on the function of the involved proxy services. For example:

- A compression proxy-pair inside an encryption proxy-pair is not OK, since encrypted data typically cannot be compressed.
- Encryption inside compression is OK, since encryption has no requirement on the data being encrypted.
- An FEC proxy-pair can be applied inside both compression and encryption.
- Compression and encryption applied inside FEC will nullify FEC, since bit errors in the compressed or encrypted data causes decompression/decryption to fail or produce bogus data.

The follow-on question is what the result of a conflict is. The most important property is given by the definition of a conflict above: It should not disrupt the end-to-end flow. This means that avoiding conflicts is primarily an optimization. If compression is applied twice, as in Figure 3.1, data may grow, rather than shrink, and the time and CPU resources will be wasted. Still, the data sent by the sender will reach the receiver correctly.

For many conflicts, the only effect on the flow is an increased per-packet latency. This means that latency sensitive applications will suffer. It will also take TCP longer time to ramp up the transmission rate to match the available bandwidth.

Proxy Interference

We define proxy interference to be the situation where the function or implementation of one proxy interferes with the operation of one or more other proxies in way that causes a complete disruption of the end-to-end flow, effectively terminating it or making the flow unusable to the end point.

Proxy interference appears when one proxy service applies a transformation that invalidates the assumptions about the flow made by another proxy service. Proxy interference can only occur between two or more proxies, when each proxy would work correctly if it was the only proxy operating on a given flow.

One assumption commonly made by most peered proxies is that the packets received by the downstream peer will be identical to the packets sent by the upstream peer. Or put another way, that if any other proxies are operating between the peers, they must be transparent to the proxy peers.

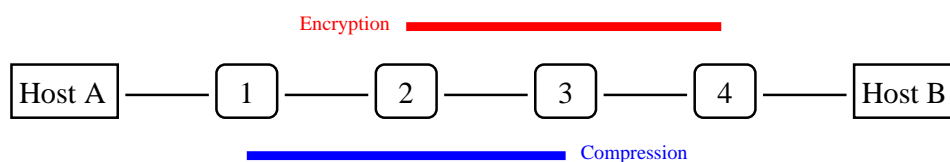


Figure 3.3: An example of proxy interference: Two different transformations are applied to the flow. However, the reverse transformations are applied in the “wrong” order, causing the content of the flow to be garbled.

Figure 3.3 illustrates a situation where this assumption does not hold. The problem in this example is that the packets in the Host A \Rightarrow Host B direction will first be compressed and then encrypted, but when the encrypted packets reach the compression peer, it will either not recognize the packets as compressed packets and just forward them, or attempt to decompress them and instead garble them.

Assuming that the packets are not discarded, they will reach the encryption peer. If the packets were garbled, the encryption peer should discard them. If they were not garbled by the compression peer, they will decrypt correctly, but after this they will still be compressed when they reach the receiver. If the compression proxy has doctored the IP and TCP headers to look correct, the compressed data will now be delivered to the application, leaving it to deal with this unexpected data. However, most likely the headers will not be properly adjusted, and the receiver IP or TCP will discard the packet as incorrect. However, since all packets in this flow will be incorrect, the connection is effectively broken.

Summary

The effects of unhandled conflicts and interference between proxies means that proxies deployed to improve the service provided by the network may end up making the situation worse, or completely disrupting the communication between end points. Without any mechanism to detect and avoid conflicts and interference, the usefulness of proxies will be greatly limited. Then, only proxies whose scope is local to one deploying entity can be safely deployed in the network.

3.1.2 Related Work

The Internet Draft on Performance Enhancing Proxies [13] presents a terminology of proxies. In the process it also goes into an architectural discussion and highlights many of the problems that proxy deployment both faces and causes. It does, however, not go into the mechanics of proxy deployment, nor does it do more than recognize the possibility of interference between proxy services.

The work on Protocol Boosters [27] formulates many of the arguments for deploying proxies and the problem of ordering proxies is recognized, but not elaborated on. The ICEBERG [65] project provides a detailed account of a system for deploying various types of content adapting proxies, and is a good illustration of a top-down architecture, and a contrast to the bottom-up scheme developed in Paper D.

The Conductor [70] framework is a framework that has many similarities with the one presented in this thesis. Among the differences is the requirement in the Conductor framework that the end system is modified to intercept connections and establish a path, and the more extensive work on reliability [71].

The work on Active Networking [62, 3, 59, 68] is related and in some ways complementary to the work on proxies described here. Active Networking can essentially be viewed as on-demand loaded application or transport layer proxies. Some work in the area of active networks come very close to what this thesis would refer to as a proxy framework [30].

Some specific proxies have also influenced this work, and the Berkeley Snoop [7] is one of the most influential. It shows that one of the important functions of earlier work such as I-TCP [6] can be done transparently, instead of requiring a split-connection approach. M-TCP [15] later refined the split-connection approach with splicing to preserve end-to-end semantics, the loss of which was one of the arguments against I-TCP and split-connection approaches in general.

The Snoop proxy has also been used as an example proxy that operates on TCP in this thesis. In a similar way, the distilling proxy [29] has been used as an example of a proxy that has a permanent effect on the flow, and the detour proxy [55] of one that operates completely at the network level.

The introduction of wireless services introduces many new problems for the Internet architecture, and possibilities for proxies [8]. Satellite services diverge so much from the behavior expected from the network by e.g. TCP, with a resulting performance penalty that make improvements very attractive [34, 26].

Issues of mobility are often related to wireless systems, and the area of mobility is also one where proxy service show up [73]. In fact, Mobile IP [49] is implemented using proxy techniques.

In telecommunication systems, the feature interaction problem [14] has been studied extensively. In proxy configuration, this type of problem is likely to occur when the property set of proxies is extended to account for new proxy service. The existing body of work in the telecommunications area can probably be adapted to feature interactions between proxies.

Finally, although this thesis has focussed on application transparent proxy services, there is much work going on in application visible proxies, especially with respect to caching and content adaptations. Examples include HTTP proxies [28] and WAP [66] gateways, and the more recent work on iCAP [40], the Internet Content Adaptation Protocol. For this type of services, application proxies may be preferable, since being visible to the application, the application can control what services are used, and also control how they are applied.

3.2 Compression

Controlling compression is the topic of Papers A and B. This section will give some more background and motivation for the work done and present some related work.

3.2.1 Background

Compression is a popular way to trade one resource for another. It is used for many different purposes in different contexts with different trade offs:

- Trade CPU usage for disk space, as seen in compressed file systems and compressed files on regular file systems.
- Trade CPU usage for memory space — designs for conserving memory in systems with memory size constraints, such as embedded systems. For example it is possible to compress the contents of the main memory and letting the CPU uncompress it on-the-fly as instructions are executed, as in the decompression core for PowerPC 401 [42].
- Trade CPU usage for faster paging — instead of paging directly to disk, schemes have been devised where pages are compressed and put into a compressed page-cache in memory [23]. If not reused, the compressed pages can be paged out to disk. This will require a shorter I/O operation than handling the uncompressed page.
- Trade CPU usage for more bandwidth, by compressing data before sending them over a communications link, as seen in V.42bis [17] modems.

Compression Algorithms

There exists algorithms that are highly tuned to specific areas such as images and audio, and there are general algorithms that only exploit that “interesting” data often contains redundancy (as opposed to random data) [44].

A distinction is made between lossless and lossy algorithms, i.e. those algorithms that will preserve the original data exactly, and those that will discard parts of the data, reducing the quality. The latter type is typically domain specific, i.e. knowledge about what type of data is being compressed is needed to determine what to discard.

For general compression three of the most often used algorithms are:

- Run length coding — A simple and fast scheme that replaces repeating patterns with the patterns and number of repetitions.
- Huffman [37] coding — Huffman coding analyses the frequency of different fixed length symbols in a data set, and assigns to the symbols codes whose length correspond to the frequency of that symbol in the data set. I.e. frequent symbols get short codes, infrequent get long codes.
- Lempel-Ziv [74, 75] — These algorithms basically replace strings (variable length symbols) found in a dictionary with codes representing those strings. The efficiency of these algorithms

is determined by the size of the dictionary and how much effort is spent searching in the dictionaries.

The algorithms used in GIF, several PPP [56, 69, 48] compressors, V.42bis [17], and many popular compression programs such as UNIX Compress, GNU Zip [22], ARC/PKARC [5, 38] and others use general compression algorithms based on the two Lempel-Ziv algorithms LZ77 [74] and LZ78 [75] (or Lempel-Ziv-Welch [67], which is based on LZ78). These algorithms are sometimes augmented with Huffman [37] coding. Huffman coding on its own is typically faster than Lempel-Ziv based algorithms, although it will typically yield less compression.

Run length encoding is extremely fast, but the gain is often small compared to Lempel-Ziv or Huffman coding.

Trade-offs

Compression uses the CPU and memory to achieve compression, which means that the use of compression always is a trade-off between the resources used and the reduction in space.

Time is also a factor — while the system may have the CPU and memory to spare, it may not be able to wait while the data is being compressed or subsequently decompressed, i.e. time in itself is a tangible resource. For example, for storing data on a disk, the CPU and memory used, and the time to compress and decompress the data, is traded off against less static storage space need.

It may seem like a paradox that compression also can be used to trade for shorter access time. If compressed data can be produced faster than a communication channel can transmit it, the time needed to get data to the other end of the channel is reduced because fewer bits need to be transmitted. This is the rationale for modem- and other low bandwidth communication compression. In these cases, it may make sense to compress data before sending it and decompressing it on the other side provided that the compression speed is greater than the bandwidth.

Compression can also be used to reduce CPU consumption for other data manipulations functions. E.g. if data will be encrypted, then by reducing the size of the data, the amount of CPU needed for encryption is reduced. Assume that it takes $t_{crypt}(b)$ seconds to encrypt b bytes and time to compress b bytes into b' bytes is $t_{compress}(b, b')$. Then if $t_{compress}(b, b') + t_{crypt}(b') < t_{crypt}(b)$ time will be saved by compressing data before encrypting it.

In summary, the reduction in size of the original data may translate into a gain in one or more of time, CPU, memory or static storage. But we see that when the gain is a side effect of the reduction in size, the system must be very carefully tuned to achieve that gain, since it relies on the speed of compression relative to other factors. E.g. if we misjudge the bandwidth of a channel or the CPU available, then compression can slow down the transfer instead of speeding it up because compression cannot keep up with the network.

3.2.2 Related Work

Compression can, and is often used to compensate for low bandwidth. This is done in modems with V.42bis [17] and in PPP [58] by CCP [52] link protocol. While a modem has resources dedicated to compression at a known range of speeds, in PPP, the host systems CPU is utilized to perform

compression. When this is done, the result will depend on what resources in the host system can be dedicated to compression.

Compression can also be applied on a per flow basis. SSH [72] can be configured to compress its connection, but since it also can be configured to tunnel traffic, compression will also be applied to the tunneled connection.

In the area of adapting compression, the closest related work that we have found is [12], which is a scheme for rate adaption for packet video. It resembles the work in Paper B in that both use feedback mechanisms to control the rate at which data is produced.

An interesting example of compression is presented in [60]. This scheme is presented more like a caching scheme, but essentially implements compression of (substrings of) network packets. Unfortunately, the scheme lacks a coherency mechanism, which means that packets will be compressed based on the context at the sender, without knowing if the receiver can decompress them.

3.3 Exokernels

This section will present some background and related work to the MIT Exokernel, which served as the basis for the work presented in Paper C.

3.3.1 Background

Exokernels [25, 41] is a new paradigm for operating systems design in which the control of the machine resources (e.g. memory, disk, CPU) to a high degree is exported to the application. The operating system kernel only adds those abstractions that are needed for safe multiplexing of these hardware resources. Any other abstractions desired are added by a library operating system which is linked with the application in much the same way that function libraries are linked to applications in traditional operating systems, e.g. Unix. The purpose of this exportation is both to increase the performance of applications as well as to reduce the resource management effort.

The library operating system is running in the same address space as the application, which means that the application has full access to internal operating system variables and can replace or modify operating system code to adapt it to the application. This shortcuts traditional OS abstractions and will hopefully give a substantial performance gain.

Ideas and Motivations

Many of the ideas and motivations for exokernels are the same as for micro kernels such as Mach [32] [57, Chapter 20]. Mach delegates OS functionality and multiplexing to trusted servers which handle scheduling, IPC, virtual memory, disk handling etc. Applications communicate with these servers via IPC mechanisms in the micro kernel. This means that a single disk read or virtual memory operation will typically cause four crossings of the kernel/user space boundary where a OS call in a traditional OS would cause only two. Since these crossings are expensive, the result will be a performance loss for micro kernels.

In an exokernel most of this functionality is placed in the user level library operating system. The

exokernel ensures that library OSs do not access resources that do not belong to them. Also, since the exokernel is typically only involved in resource allocation, most resource management can be done with no crossings of the kernel/user space boundary.

In summary, the features of exokernel includes a great degree of control over resources and resource management. This means that the application can chose to implement those schemes for resource management that best fit the way the application use the resources.

The backside of this decentralized control is that it is very hard to implement global strategies for resource management that apply to all processes and library OSs. Sharing of resources require cooperation, and may force some type of server architecture similar to that found in micro kernels.

3.3.2 Related Work

Exokernels is one of many efforts to put more control over operating system functionality into the hands of the application programmer, another is micro kernels. The first generation micro kernels such as Mach [32] and Chorus [53] evolved from monolithic kernels, and often carry baggage from their ancestors that burdens them. Still, they offer a great improvement in flexibility over their ancestors. The second generation micro kernels, such as QNX [35] and L4 [36] have been implemented from scratch and have aimed for minimal systems and a clean design, and are not limited in the same way.

Another approach of operating systems specialization is to modify the whole operating system to provide better support for specific applications. Techniques to improve performance of applications with this technique has been explored in [50]. The idea is basically the same as in the exokernel approach, but without the possibility to run multiple OSs at the same time.

Various other approaches to flexible and extensible operating systems have been investigated, including Spin [9], Scout [46] and Nemesis [45]. Their objectives are not the same, e.g. Nemesis focuses on Quality of Service (QoS) guarantees which has guided the designers to trade off performance for predictability, Scout focuses on abstractions² for protocol implementations and Spin on how to safely download code into the kernel from the user level.

In the general area of trying to get better communication performance from end points, there is a sizable body of work on network interfaces and communications software. Examples of methods that are similar in spirit to the work in the MIT Exokernel include the work on Integrated Layer Processing [1] and zero-copy APIs for communication[2].

²Called *paths*.

Chapter 4

Conclusions

The growth of the Internet holds great promise in form of both improved and new services to both make our life simpler and allow us a greater freedom. But the growth of the Internet into new areas has also meant that the original Internet architecture no longer is enough to provide the support and demands that some new services require. Ironically, the very success of the Internet architecture is also a hindrance in the efforts to accommodate these new services, since changes and additions to the fundamental architecture of the Internet would require upgrading millions of system.

Proxies are not formally a part of the Internet architecture, but can be seen as a flexible way of extending the architecture without requiring large scale upgrades. As such, proxies are instrumental to the continued growth of the Internet.

As more and more proxies are deployed in the Internet, architectural issues for proxies become more important. The uncontrolled deployment of proxies can have severe consequences for the Internet. This thesis has highlighted the problems related to interference between proxies, and proposes methods to deal with these problems.

This thesis also proposes a framework and signalling protocol for proxy deployment that can be seen as one example of a proxy architecture. The particular architecture is designed to be able to span both end-to-end, as well as limited parts of the network.

The thesis argues that there are benefits from allowing proxies from different users, network operators, Internet service providers and content providers to cooperate. However, there exists a lot of incentive for commercial entities to isolate their proxies and instead compete for proxy services. The work in this thesis suggest that some level of cooperation may still be necessary for correct operation, but the results presented here also suggests what type of proxy services can be deployed without requiring cooperation.

Bibliography

- [1] Bengt Ahlgren, Mats Björkman, and Per Gunningberg. The applicability of integrated layer processing. *Journal of Selected Areas of Communication*, 16(3):317–331, April 1998.
- [2] Bengt Ahlgren, Mats Björkman, and Kjersti Moldeklev. The performance of a no-copy API for communication. In *IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, August 1995.
- [3] D. Scott Alexander, Marianne Shaw, Scott M. Nettles, and Jonathan M. Smith. Active bridging. In *Proceedings of ACM SIGCOMM'97*, pages 101–111, September 1997.
- [4] Joe Armstrong, Robert Virding, Claes Wikström, and Mike Williams. *Concurrent Programming in Erlang*. Prentice Hall, 1993.
- [5] System Enhancement Associates. Arc file archive utility, 1986. Version 5.1.
- [6] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for mobile hosts. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, pages 136–143, 1995.
- [7] H. Balakrishnan, S. S. Seshan, and R. H. Katz. Improving reliable transport and handoff performance in cellular wireless networks. In *Wireless Networks 1*, pages 469–481, 1995.
- [8] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *IEEE/ACM Transactions on Networking*, December 1997.
- [9] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. Fiuczynski, D. Becker, S. Eggers, and C. Chambers. Extensibility, safety and performance in the SPIN operating system. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, December 1995.
- [10] Stefan Björnelund. Experiments with Erlang on exokernel based systems. Master's thesis, University of Uppsala, Department of Computer Systems, 1998. (Forthcoming).
- [11] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC 2475: An architecture for differentiated services, December 1998.
- [12] J. Bolot and T. Tuletli. Experience with rate control mechanisms for packet video in the internet. *ACM SIGCOMM Computer Communication Review*, 1998(1):4–15, January 1998.
- [13] J. Border, M. Kojo, Jim Griner, G. Montenegro, and Z. Shelby. Performance Enhancing Proxies. INTERNET-DRAFT, October 2000. draft-ietf-pilc-pep-04.

- [14] T.F Bowen, F.S Dworack, C.H. Chow, N. Griffeth, G. E. Herman, and Y-J Lin. The feature interaction problem in telecommunications system. *SETS*, 1989.
- [15] K. Brown and S. Singh. M-TCP: TCP for Mobile Cellular Networks. *Computer Communication Review*, 27(5), 1997.
- [16] B. Carpenter. RFC 2775: Internet transparency, February 2000.
- [17] CCITT. *Recommendation V.42 bis*, F ITU 1990 edition.
- [18] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of ACM SIGCOMM '90*, pages 200–208. ACM, 1990.
- [19] David Clark and Marjory Blumenthal. Rethinking the design of the internet: end to end arguments vs. the brave new world. In *Proc. of the 28th Telecommunications Policy Research Conference (TPRC 2000)*, September 2000.
- [20] David D. Clark. A model for cost allocation and pricing in the internet. In *Proceedings of MIT Workshop on Internet Economics*, March 1995.
- [21] J. Crowcroft, C. Diot, and P. Gunningberg. *HIPPARCH project plan*, 1996.
- [22] P. Deutsch. *GZIP file format specification version 4.3*. IETF, Network Working Group, 1996. RFC1952.
- [23] Fred Dougls. The compression cache: Using on-line compression to extend physical memory. In *Proceedings of 1993 Winter USENIX Conference*, pages 519–529, January 1993.
- [24] David Eckhardt and Peter Steenkiste. Measurement and analysis of the error characteristics of an in-building wireless network. In *Proceedings of ACM SIGCOMM'96*, 1996.
- [25] Dawson R. Engler, M. Frans Kaashoek, and James W. O'Toole Jr. Exokernel: An operating system architecture for application-level resource management. In *Proceedings of the Fifteenth Symposium on Operating System Principles*, pages 251–266, December 1995.
- [26] M. Allman et al. RFC 2760: Ongoing TCP research related to satellites, February 2000.
- [27] D.C. Feldmeier, A.J. McAuley, J.M. Smith, D. Bakin, W.S. Marcus, and T. Raleigh. Protocol Boosters. *IEEE Journal on Selected Areas in Communication*, 16(3):437–444, April 1998. Special Issue on Protocol Architectures for 21st Century.
- [28] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. IETF, Network Working Group, June 1999.
- [29] Armando Fox, Steven D. Gribble, Eric A. Brewer, and Elan Amir. Adapting to network and client variability via on-demand dynamic distillation. In *Proceedings of 7th Intl. Conf. On Arch. Support of Prog. Lang. And Oper. Sys. (ASPLOS VII)*, October 1996.
- [30] Michael Fry and Atanu Ghosh. Application level active networking. In *Proceedings of the Fourth International Workshop on High Performance Protocol Architectures (HIPPARCH'98)*, 1998.

- [31] B. Gleeson, A. Lin, J. Heinanen, G. Armitage, and A. Malis. RFC 2764: A framework for IP based virtual private networks, February 2000.
- [32] D. Goloub, R. Dean, A. Forin, and R. Rashid. UNIX as an application program. In *USENIX 1990 Summer Conference*, pages 87–95, June 1990.
- [33] P. Gunningberg and A. Seneviratne. Services and architectures in the Next Generation Internet using dynamic proxies. In *Future Telecommunications Forum 99*, pages 82–86, December 1999. Beijing, China.
- [34] T.R. Henderson and R.H. Katz. Transport protocols for internet-compatible satellite networks. *IEEE Journal on Selected Areas in Communications*, 17(2):345–359, February 1999.
- [35] D. Hildebrand. An architectural overview of qnx. In *Proceedings of the 1st USENIX Workshop on Microkernels and Other Kernel Architectures*, pages 113–126, April 1992.
- [36] Hermann Härtig, Michael Hohmuth, Jochen Liedtke, Sebastian Schönberg, and Jean Wolter. Performance of μ -kernel-based systems. In *Proceedings of the Sixteenth Symposium on Operating System Principles*, pages 66–77, October 1997.
- [37] D. Huffman. A method for the construction of minimum redundancy codes. In *Proceedings of the Institute of Radio Engineers*, volume 40, pages 1098–1101, 1952.
- [38] PKWARE Inc. Pkarc fast! file archival utility, 1987. Version 3.5.
- [39] V. Jacobson, R. Braden, and D. Borman. RFC 1323: TCP extensions for high performance, May 1992.
- [40] Elson Jeremy. ICAP the Internet Content Adaptation Protocol. INTERNET-DRAFT, February 2000. draft-elson-opes-icap-01.
- [41] M. Frans Kaashoek, Dawson R. Engler, Gregory R. Ganger, Héctor M. Briceño, Russell Hunt, David Mazières, Thomas Pinckney, Robert Grimm, John Jannotti, and Kenneth Mackenzie. Application Performance and Flexibility on Exokernel Systems. In *Proceedings of the Sixteenth Symposium on Operating System Principles*, October 1997.
- [42] T. M. Kemp, R. K. Montoye, J. D. Harper, J. D. Palmer, and D. J. Auerbach. A decompression core for PowerPC. *IBM Journal of Research & Development*, 42(6), 1998.
- [43] S. Kent and R. Atkinson. RFC 2401: Security architecture for the Internet Protocol, November 1998.
- [44] Debra A. Lelewer and Daniel S. Hirschberg. Data compression. *Computing Surveys*, 19(3):261–297, 1987. Reprinted in Japanese BIT Special issue in *Computer Science* (1989) 165-195.
- [45] Ian Leslie, Derek McAuley, Richard Black, Timothy Roscoe, Paul Barham, David Evers, Robin Fairbairns, and Eoin Hyden. The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. *IEEE Journal on Selected Areas in Communication*, 14(7):1280–1297, September 1996.
- [46] D. Mosberger and L.L. Peterson. Making paths explicit in the scout operating system. In *Proceedings of the 2nd USENIX Symposium on Operating Systems Design and Implementation*, pages 153–167, October 1996.

- [47] Vivek S. Paiz, Mohit Aron, Gaurav Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel, and Erich Nahum. Locality-aware request distribution in cluster-based network servers. In *Proceedings of ASPLOS'98*, 1998.
- [48] Gurdeep Singh Pall. *Microsoft Point-To-Point Compression (MPPC) Protocol*. IETF, Network Working Group, 1997. RFC2118.
- [49] C. Perkins. RFC 2002: IP mobility support, October 1996.
- [50] C. Pu, T. Autrey, A. Black, C. Consel, C. Cowan, J. Inouye, L. Kethana, J. Walpole, and K. Zhang. Optimistic incremental specialization: Streamlining a commercial operating system. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, pages 314–324, December 1995.
- [51] Héctor Manuel Briceño Pulido. Decentralizing UNIX abstractions in the exokernel architecture. Master's thesis, Massachusetts Institute of Technology, February 1997.
- [52] Dave Rand. *The PPP Compression Control Protocol (CCP)*. IETF, Network Working Group, 1996. RFC1962.
- [53] M. Rozier, A. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrmann, C. Kaiser, S. Langlois, P. Leonard, and W. Neuhauser. Chorus distributed operating system. *Computing Systems*, 1(4):305–370, 1988.
- [54] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. In *ACM Transactions on Computer Systems*. ACM, 1984.
- [55] Stefan Savage, Tom Anderson, Amit Aggarwal, David Becker, Neal Cardwell, Andy Collins, Eric Hoffman, John Snell, Amin Vahdat, Geoff Voelker, and John Zahorjan. Detour: A case for informed internet routing and transport. *IEEE Micro*, 19(1), Jan/Feb 1999.
- [56] Vernon Schryver. *PPP BSD Compression Protocol*. IETF, Network Working Group, 1996. RFC1977.
- [57] Abraham Silberschatz and Peter B. Galvin. *Operating System Concepts*. Addison-Wesley, fourth edition, 1994. ISBN 0-201-59292-4.
- [58] W. Simpson. *The Point-to-Point Protocol (PPP)*. IETF, Network Working Group, July 1994. STD 51, RFC 1661.
- [59] Jonathan M. Smith, Kenneth L. Calvert, Sandra L. Murphy, Hilarie K. Orman, and Larry L. Peterson. Activating networks: A progress report. *IEEE Computer*, 32(4):32–41, April 1999.
- [60] Neil T. Spring and David Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proceedings of ACM SIGCOMM*, August 2000.
- [61] P. Srisuresh and M. Holdrege. RFC 2663: IP Network Address Translator (NAT) Terminology and Considerations, August 1998.
- [62] D. L. Tennenhouse and D. Wetherall. Towards an active network architecture. In *Proc. of Multimedia Computing and Networking 96*, San Jose, CA, January 1996.

- [63] R. Thayer, N. Doraswamy, and R. Glenn. RFC 2411: IP security document roadmap, November 1998.
- [64] G. Tsirtsis and P. Srisuresh. RFC 2766: Network Address Translation - Protocol Translation (NAT-PT), February 2000.
- [65] Helen J. Wang, Bhaskaran Raman, Chen nee Chuah, et al. ICEBERG: An Internet-core Network Architecture for Integrated Communications. *IEEE Personal Communications*, August 2000.
- [66] WAP Forum. Wireless Application Protocol, 2000. Wireless Application Protocol, Technical Specifications. <http://www.wapforum.org/what/technical.htm>, 2000.
- [67] Terry A. Welch. A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, June 1984.
- [68] David Wetherall. Active network vision and reality: lessons from a capsule-based system. In *Proceedings of SOSP'99*, pages 64–79, December 1999.
- [69] John Woods. *PPP Deflate Protocol*. IETF, Network Working Group, 1996. RFC1979.
- [70] Mark Yarvis, Peter Reiher, and Gerald J. Popek. Conductor: A Framework for Distributed Adaptation. In *Proc. Seventh Workshop on Hot Topics in Operating Systems (HotOS VII)*, March 1999.
- [71] Mark Yarvis, Peter Reiher, and Gerald J. Popek. A reliability model for distributed adaptation. In *Proceedings of the Third IEEE Conference on Open Architectures and Network Programming (OPENARCH 2000)*, March 2000.
- [72] Tatu Ylonen. SSH—Secure login connections over the Internet. In *Proceedings of the Sixth USENIX Security Symposium*, pages 37–42, July 1996.
- [73] B. Zenel and D. Duchamp. A General Purpose Proxy Filtering Mechanism Applied to the Mobile Environment. In *Proceedings of The Third Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'97)*, September 1997.
- [74] Jakob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, May 1977.
- [75] Jakob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

Paper A

Björn Knutsson and Mats Björkman, *Trading Computation for Communication by End-to-End Compression*, in Proceedings of the Third International Workshop on High Performance Protocol Architectures (HIPPARCH'97), Uppsala, Sweden, June 1997

Paper B

Björn Knutsson and Mats Björkman, *Adaptive End-to-End Compression for Variable-bandwidth Communication*, published in *Computer Networks*, vol 31, issue 7, 1999.

Paper C

Björn Knutsson and Per Gunningberg, *Exokernels, Protocol Implementation and Erlang*, Technical report DoCS 103, Department of Computer Systems, Uppsala University, Sweden, 1999.

Paper D

Björn Knutsson and Larry L. Peterson, *Transparent Proxy Signalling*, Accepted for publication in the Journal of Communications and Networks, March, 2001.

Paper E

Björn Knutsson and Per Gunningberg, *Automatic Proxy Configuration*, The Workshop on Mobile Multimedia Communication (MoMuC'2000), Tokyo, October, 2000.

Paper F

Björn Knutsson and Per Gunningberg, *Automating Proxy Configuration and Placement*, March, 2001.

Department of Computer Systems Dissertation Series

- 85/03 Joachim Parrow, *Fairness Properties in Process Algebra*
- 87/09 Bengt Jonsson, *Compositional Verification of Distributed Systems*
- 90/21 Parosh A. Abdulla, *Decision Problems in Systolic Circuit Verification*
- 90/22 Ivan Christoff, *Testing Equivalences for Probabilistic Processes*
- 91/27 Hans A. Hansson, *Time and Probability in Formal Design of Distributed Systems*
- 91/31 Peter Sjödin, *From LOTOS Specifications to Distributed Implementations*
- 93/37 Linda Christoff, *Specification and Verification Methods for Probabilistic Processes*
- 93/40 Mats Björkman, *Architectures for High Performance Communication*
- 94/46 Fredrik Orava, *On the Formal Analysis of Telecommunication Protocols*
- 96/70 Lars Björnfot, *Specification and Implementation of Distributed Real-Time Systems for Embedded Applications*
- 97/80 Bengt Ahlgren, *Improving Computer Communication Performance by Reducing Memory Bandwidth Consumption*
- 98/98 Björn Victor, *The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes*
- 98/100 Ernst Nordström, *Markov Decision Problems in ATM Traffic Control*
- 99/101 Paul Pettersson, *Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice*
- 99/110 Mats Kindahl, *Verification of Infinite-State Systems: Decision Problems and Efficient Algorithms*
- 00/114 Kristina Lundqvist, *Distributed Computing and Safety Critical Systems in Ada*
- 00/115 Jan Gustafsson, *Analyzing Execution-Time of Object-Oriented Programs Using Abstract Interpretation*
- 00/116 Jakob Carlström, *Reinforcement Learning for Admission Control and Routing*
- 00/117 Mikael Sjödin, *Predictable High-Speed Communications for Distributed Real-Time Systems*
- 01/118 Björn Knutsson, *Architectures for Application Transparent Proxies: A Study of Network Enhancing Software*