

Automatic Proxy Configuration

Björn Knutsson

(Bjorn.Knutsson@DoCS.UU.SE)

Per Gunningberg

(Per.Gunningberg@DoCS.UU.SE)

Dept. of Information Technology, Uppsala University,
P.O. Box 325, SE-751 05, UPPSALA, SWEDEN

ABSTRACT

Running multiple proxies on a flow means a risk that the actions of one proxy interferes with another. To avoid this, every proxy must be made aware of other proxies operating on the flow, and be configured in a way that avoids interference. This paper argues that the way to do this is to describe proxies in terms of abstract properties that describe the effect they have on the flow. These abstract properties are then used in a configuration engine to determine how to configure the proxies in a non-interfering manner.

I. INTRODUCTION

Network proxies are applications that are deployed in networks to improve the service provided to end points. For example, caching HTTP proxies are put in the network to avoid redundant network traffic and to reduce response times. For end points that are connected via links that experience high bit error rates, a transparent proxy that reduces the impact of bit errors would be very beneficial.

It is our belief that the use of network proxies will increase, in order to cope with more and more heterogenous networks and end points. As more and more proxies are deployed, the risk of interference between proxy protocols increases. In a multi-proxy environment, proxies must take the actions of other proxies into account, and this paper is about a method for dealing with this problem.

Some types of proxies have a natural place inside the network—HTTP caches are typically put at the gateway to an autonomous system. There are at least two arguments against implementing many of the services provided by proxies in the end-to-end transport- or application protocols. The first is that end points rarely know when a particular service (e.g. reducing impact of bit errors) is needed and even if this could be detected, this would be a slow process since the properties of the links would first have to be discovered empirically. The second is that additions or alterations of end-to-end protocols are costly, since it involves changing the software implementing the protocol, possibly in every end point, and such alterations thus are unlikely to happen unless they are beneficial to a large

group of users.

Network proxies come in two distinct flavors; explicit and transparent. Explicit proxies are the proxies that the user or application explicitly connect to for a service, e.g. an HTTP cache. Transparent proxies are proxies that operate on the flow between two hosts and are not necessarily directly visible to the end points. This transparency does not mean that they cannot be controlled by the user. Being able to control the proxies is especially important for transparent proxies that adapt to available bandwidth by reducing the informational content of the flow by reducing the quality of the audio and video contents of a flow.

In further discussions we will use the word *proxy* to refer to the instance of an enhanced service running in the network, while we'll use *proxy protocol* to refer to the specific implementation of a proxy service, i.e. a proxy may implement several different proxy protocols.

II. PROXY REGIONS

In [5] we have presented an infrastructure for proxy detection and signalling between proxies. The most important function of this infrastructure is that it provides a way to expose the proxy protocols involved. Each proxy along a flow will see all proxy protocols operating on the flow, allowing them to take into account the effects these proxy protocols will have on a flow.

This infrastructure is based on our *proxy region* concept. A proxy region is the superset of all *regions* created by various proxies operating on a given flow. For example, if we have two proxies that provides compression/decompression of a data flow between two nodes in the network, then that part of the network is a *compression region*.

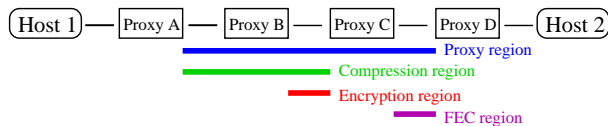


Fig. 1. The *proxy region* is the superset of all other regions.

We have identified various properties that define regions, e.g. *trust* (encryption), *reliability* (error correc-

tion/recovery) etc. Put in another way, a *proxy region* is simply the region along the path of a given flow that is bounded by proxies that understand a unified proxy signalling scheme, as in Figure 1. Within the proxy region, we have a high degree of freedom to transform or re-route packages, but once outside the proxy region, the flow should again look like a normal flow.

Setting up a proxy region is done in the following way: First the proxies in the proxy region are detected and the boundary proxies, which are responsible for the proxy region, are selected. Next, a configuration message is created by the border that will be responsible for proxy configuration, and the message is passed along the path of the flow in the proxy region. Each proxy appends announcements of the proxy protocols it is willing to run. When the configuration message returns to the originator, it decides, based on the information in the configuration message, which proxies and proxy protocols to activate. Finally, an activation message is passed along a path that passes all candidate proxies, telling each proxy which proxy protocols to activate.

III. PROXY CONFIGURATION

To illustrate the need for unified proxy configuration, let us assume that we are watching a video feed, and that the flow spans a wireless segment. To cope with losses inherent in wireless communication, a proxy is used for Forward Error Correction (FEC)[7], [4]. If the link bandwidth is limited, then another proxy can be deployed to reduce the amount of data being sent, for example by transforming images or video in the flow to use less colors and/or resolution[3]. Finally, due to the ease of eavesdropping on wireless traffic, it might be desirable to deploy a proxy that encrypts the flow across the wireless segments.

The problem we run into now is that these three services cannot be implemented in an arbitrary order and at arbitrary places. An encrypted data stream cannot be transformed, and if the flow is altered in any way after FEC has been added, this will nullify the effect of adding FEC since the FEC is valid only for exactly the data it was computed for. We need a way to deduce the ordering constraints, and thus ensure that we transform first, encrypt next and add FEC last.

We can create a simple configuration mechanism that bases configuration decisions on each proxy’s knowledge of the other proxy protocols operating on the flow. In the example above, if the FEC, encryption and transformation proxies know about each other and the effect that each proxy protocol have on the flow, they can arrange to avoid conflicts.

This is done by allowing each proxy protocol to spec-

ify that that it is incompatible with a specified set of other proxy protocols, and also by allowing proxy protocols to set priorities in case of conflicts. This is a step forward compared to the situation without any coordinated proxy handling, but we recognize that relying on each proxy’s knowledge of other proxy protocols will not scale.

A. Abstract Properties

It is our goal in this paper to give an outline of a configuration mechanism that does not rely on each proxy protocol having explicit knowledge of other proxy protocols, but instead can make configuration decisions based on information provided by the proxy protocols about the effect they have on the flow. This reduces or eliminates the need to modify existing proxy protocols when new proxy protocols are added.

Our approach is to describe proxy protocols in terms of abstract properties. These properties are general enough to capture similar behavior in different proxy protocols, while specific enough to allow proxies to deduce the proper configuration of proxy protocols on a flow.

A.1 Properties Derived from Behavior

To deduce a proper configuration we need to take the properties of different proxy protocols into account. In the FEC, encryption and image transform example, we need to capture the property of FEC that any alteration of the packets will disrupt the operation of the FEC proxies. We also need to capture the difference between transforming the flow by encrypting and by reducing the information content.

Other properties do not have ordering implications, but still affect the effectiveness of the proxy protocols. For example, since FEC adds redundant data to allow recovery from errors and losses, the FEC region should be minimized. A compressing proxy, on the other hand, would reduce bandwidth consumption, and thus we’d want to maximize the size of the region to reap maximum benefit from the effort put into compression.

Yet another category of proxy properties specifies permanent effects on flows. Both FEC and encryption egress proxies will revert the ingress proxy’s transformation. This reversible property is, however, not true of an image reduction proxy. This means that any other proxy that must operate on the original content of the image, cannot operate after a proxy that permanently alters the flow. We should not confuse original content with semantically similar content, i.e. the transformation that reduce colors still maintain the approximate semantics of the original flow—it is still a picture of the same object, but not as information rich. This property should also be captured.

At this stage, we cannot present an exhaustive list of properties needed for a general proxy configuration, or even categories of properties. Instead we will present a subset that is enough to understand the implications and problems of proxy configuration.

A.2 Properties Derived from Policy

So far the properties have been derived from the behavior and requirements of proxy protocols. The proxies we focus on are assumed to be *transparent* to the end points. This means that they operate without requiring input from the user or from the end point systems. But when an image is being transferred, there may be no way for the user to indicate to the proxies in the network that image quality is more important than transfer time or vice versa. So how does a transparent proxy know when to reduce transfer time of an image by reducing its quality? The answer is that it cannot.

Our solution is to introduce policy properties to allow end points, and ultimately the user, to control activation of proxy protocols with permanent effects on the content of flows. These proxy protocols must indicate this with the **Permanent** property. A parameter indicates if it alters **image, video, audio, text** or **all** types of data. Permanent alterations of the flow are only allowed if this is explicitly requested by the user or the application, using the policy property **allow**. There is also a **disallow** property that overrides the **allow** property, i.e. if there exists at least one request for an unaltered flow, then the flow shall not be altered.

If the proxy region extends to the end points, then the user can set these properties directly. An alternative is to use a configuration proxy. This proxy has no effect on the flow except during region setup and sets the **allow/disallow** properties. The user controls this proxy via a separate connection.

B. Candidate Properties

Our aim is to define a set of properties that captures the constraints that typical proxy protocols imposes on configuration. As more proxy protocols are examined, properties defined by the already examined proxy protocols will be refined, generalized and/or corroborated. Conflicts between a new proxy protocol and the already examined may force us to add new properties to both the new and old proxy protocols.

We will here formulate properties for six different proxy protocols: FEC, encryption, image reduction, TCP Snoop and an MPEG frame dropper.

B.1 The FEC Proxy Protocol

An ingress proxy adds Forward Error Correction information to packets to enable an egress proxy to recovery from bit errors introduced in an unreliable region of the network. It is characterized with the following properties:
Peers(1): One peer using the same proxy protocol is needed.

Forbid(Inside(Alters(data,packaging))): The packets (data or packet headers) inside the region may not be altered in any way once FEC has been added.

Alters(data): FEC information is added to the packet, changing the content of the packet.

Reverts(data): All the alterations to the packet data are reverted by the peer.

Placement(narrow): The region between peers should be minimized.

Semantic(preserving): Modifies data, but preserves the original semantics of the flow.

B.2 The Encryption Proxy Protocol

An ingress proxy encrypts data which is then decrypted by an egress proxy, providing secure transfer across a untrusted region of the net. Properties:

Peers(1), Reverts(data), Alters(data)

Semantic(hiding): The semantic of the data is hidden, i.e. a proxy that is dependent on the semantic of the data cannot operate on this flow after it has been encrypted.

B.3 The Compression Proxy Protocol

An ingress proxy compresses the data in the packets and an egress proxy uncompresses them, reducing the bandwidth needed to transport the packets across a region. Properties:

Peers(1), Reverts(data), Alters(data), Semantic(hiding)

Placement(wide): The region should be as wide as possible.

Forbid(Prior(Semantic(hiding))): The semantic of data entering from outside the compression region must not have its semantic hidden.

B.4 The Image Reduction Proxy Protocol

The proxy transforms images in the flow to lower resolution, lesser colors or better encoding to reduce the bandwidth needed to transfer the image. Properties:

Semantic(preserving), Forbid(Prior(Semantic(hiding))), Alters(data)

Permanent(image): This proxy will permanently alter the image content of the flow.

Peers(none): Only a single instance of this particular proxy protocol should ever operate on the flow.

	Peers	Forbid	Reverts	Placement	Semantic	Alters	Permanent
FEC	1	Inside(Alters(data)) Inside(Alters(packaging))	data	narrow	preserving	data	-
Encrypt	1	-	data	-	hiding	data	-
Compress	1	Prior(Semantic(hiding))	data	wide	hiding	data	-
Image	none	Prior(Semantic(hiding))	-	server	preserving	data packaging endpoint	image
Snoop	0	-	-	-	-	drops regenerates	-
MPEG	none	Prior(Semantic(hiding))	-	server	preserving	data packaging	video

TABLE I
The property matrix for the examined proxies.

Alters(packaging): This proxy permanently alters the packaging of the flow. (In this case since the number of bytes will be permanently reduced.) This implies, among other things, that this proxy cannot be removed once activated.

Alters(endpoint): This proxy will fake end point behavior. (In this case because it may need to receive a whole picture before transforming it.)

Placement(server): Proxy should be placed close to the server.

B.5 The TCP Snoop Proxy Protocol

The snoop[1] proxy retransmits TCP packets lost due to errors to avoid triggering TCP congestion control mechanisms for non-congestion packet losses. Properties:

Peers(0): No peer is needed, but multiple instances can operate on the flow.

Alters(drops): May chose to drop packets rather than forward them.

Alters(regenerates): May regenerate and retransmit previously seen packets.

B.6 The MPEG Frame Dropper Proxy Protocol

The proxy drops selected video frames (P- and B-frames) to reduce bandwidth use of MPEG stream[2], [6]. Properties:

Placement(server), Forbid(Prior(Semantic(hiding))), Permanent(video), Alters(data), Alters(packaging), Peers(none).

This proxy can be completely described in terms of the properties already defined.

C. Configuration Engine

Each proxy will describe each proxy protocol it is willing to offer using the abstract properties, as can be seen in

the property matrix in Table I.

The configuration information for all proxies in the region is passed on to the proxy region border proxies assigned to do the actual configuration of the proxy region. The configuration is done by solving the constraint problem formulated by the abstract properties of each proxy protocol and what placement options exists, i.e. which proxy protocols each proxy in the region has offered to run.

Once the constraint problem is solved, the activation message is constructed, telling each proxy in the region which proxy protocols to activate and the path the flow should take through the proxy region. This message is sent along the path(s) of the flow. When the message reaches the other proxy region border, the border proxy starts channeling the flow along the proxy path and sends an acknowledgement back.

IV. CONCLUSIONS AND FUTURE WORK

If we want to run more than one transparent proxy protocol on any flow, we are likely to run into configuration problems, especially with ordering. If the number of proxy protocols is low, it may be feasible to encode knowledge of the interactions of all other proxy protocols into each proxy protocol. If we want to support an arbitrary number of proxy protocols, and are not prepared to modify existing proxy protocols every time a new proxy protocol is introduced, a proxy configuration scheme that use some form of abstract description of the properties of proxy protocols is needed.

In this paper we have outlined such a configuration scheme. We have also presented some examples of abstract properties that describe the effects of proxy protocols on flows. Finalizing the set of abstract properties used by the configuration mechanism is an area of future work,

as is the details of the configuration engine.

ACKNOWLEDGEMENTS

This work is funded by Ericsson Radio under the MARCH project.

We would like to acknowledge Larry Peterson of Princeton University for his participation in the work out of which this scheme grew.

REFERENCES

- [1] H. Balakrishnan, S. S. Seshan, and R. H. Katz. Improving reliable transport and handoff performance in cellular wireless networks. In *Wireless Networks I*, pages 469–481, 1995.
- [2] S. Cen, C. Pu, R. Staehli, C. Cowan, and J. Walpole. A Distributed Real-Time MPEG Video Audio Player. In *Proceedings of NOSS-DAV'95*, pages 18–21, April 1995.
- [3] 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.
- [4] S. Keshav. *An Engineering Approach to Computer Networking*. Addison-Wesley, 1997. ISBN 0-201-63442-2.
- [5] Björn Knutsson and Larry L. Peterson. *Transparent Proxy Signalling*. Submitted to *Infocom 2001*, June 2000.
- [6] K. Mayer-Patel and L.A. Rowe. Design and Performance of the Berkeley Continuous Media Toolkit. In *SPIE Proceedings*, volume 3020, Feb 1997.
- [7] William Stallings. *Data and Computer Communications*. MacMillan, fourth edition, 1991. ISBN 0-02-415441-5.