Daigan: Constructing Proxy Networks using SelNet

Richard Gold and Mats Uddenfeldt
Department of Information Technology
Uppsala University, Box 337
S-75105 Uppsala, Sweden
Email: Richard.Gold@it.uu.se, Mats.Uddenfeldt.9771@student.uu.se

***Abstract*— We present Daigan, a tool for constructing proxy networks using the SelNet indirection layer. Using the indirection capabilities of SelNet, we can construct on-demand chains of proxies for adapting content for mobile devices. Daigan takes a specification of a device's capabilities and from that identifies which proxies are needed to meet these requirements. Based on this specification Daigan sets up a chain of proxies and transparently redirects an application's traffic to the proxy chain. We demonstrate Daigan's abilities through a scenario which performs the instantiation of a proxy chain consisting of multiple proxies. We describe this scenario in detail and report on implementation progress.**

I. Introduction

There has been an increase in the heterogeneity of the devices attached to the edge of the Internet. Previously the domain of powerful workstations and personal computers, the Internet is now also host to devices with a limited set of capabilities. For example, terminals such as mobile phones connected via GPRS or 3G, PDAs connected via WiFi or Bluetooth or laptops with in-flight Internet access via satellite are constrained by CPU processing power, display resolution, battery life, memory capacity and connectivity limitations. This means that content such as webpages with embedded multimedia objects are not suitable to be rendered on small displays or the cost of downloading such objects on a low-bandwidth connection is prohibitive. However, if the content can be adapted before being transmitted over the last-hop link, then it may be possible for the multimedia objects to be viewed. The set of possible content transformations or protocol enhancements that could be made are wide and include content caching, compression or transcoding of images, transcoding of audio/video streams and header compression. Due to the restrictions of the mobile device, it is not always possible to have all the content adaptation systems present on the device itself. Additionally, it is not necessarily the case that each content server on the Internet has the ability to host all the different content adaptation systems either. There is a wide spectrum of possible enhancements that could be made and these may not all be available at one particular site. In this paper we discuss a third way, similar to [1], which establishes a third party which can host the necessary functionality.

Proxy networks already exist in the form of proposals such as MARCH [2] and ALAN [3]. We are not attempting to create another application layer overlay solution, but rather to explore the potential of SelNet's indirection mechanisms to implement a proxy network solution. We believe that an indirection layer placed as far down in the protocol stack as possible provides a better basis for constructing such proxy networks than a bespoke overlay. This is mainly due to the inherent reuseability of the lower layers of the protocol stack. In order to investigate this claim, we have created a system called Daigan which is an application which uses the SelNet indirection layer. SelNet is an indirection layer which sits at layer 2.5 i.e., between the network and link layers.

Daigan allows the creation of an on-demand, per-application chains of proxies to satisfy user preferences and devices restrictions. It is not necessarily the case that only one proxy is required to meet these requirements. In order to allow more than one proxy to be interposed between the client and the server, Daigan uses the signalling capabilities of the SelNet indirection layer to set up the proxy chains and the indirection capabilities to redirect application traffic to the proxy chain. These mechanisms are described in section II. Daigan itself provides the specification of user requirements and the decision engine to work out which proxies need to be applied in which order so that the requirements are satisfied.

We show that only two extra message types has to be added to SelNet in order for Daigan to cast its requirements into the SelNet indirection layer. The flexibility and reuseability of SelNet's indirection mechanisms mean that implementing such proxy networks becomes just a simple extension of SelNet. Traditionally such indirection mechanisms have involved the overloading of existing mechanisms such as DNS in order to achieve indirection. Also, applications which build overlays on top of IP then inherit the syntax and semantics of the IP layer. For example, having to use the five tuple of source & destination IP address, source & destination port number and protocol number or the three tuple of destination IP address, destination port number and protocol type as a means of identifying a connection. Daigan then implements all the functionality necessary for the application and can leave the indirection functionality to be implemented by an indirection layer. This is in contrast to overlay solutions since their indirection mechanisms are typically not reuseable by other applications. Additionally stacking overlay mechanisms on top of each other in order to achieve reuse can cause problems due to header inflation and excessive virtualization [4].

II. Interlude: SelNet

We provide an interlude here to briefly explain how SelNet functions since it is crucial to the understanding of Daigan.

SelNet is based on the *Network Pointers* concept detailed in [5] and is explained in more detail in [6]. SelNet comprises of two main parts: XRP (eXtensible Resolution Protocol) and SAPF (Simple Active Packet Format). XRP is our signalling mechanism and SAPF is our forwarding mechanism. XRP and SAPF will be touched upon within this paper, but are explained in detail in [7].

SelNet is an indirection layer for the Internet which sits at layer 2.5 i.e., between the network and link layer. At its heart it is a label switching architecture. In SelNet, SAPF *selectors* are used as labels. In contrast to other systems such as MPLS, SAPF selectors do not correspond to paths or interfaces but rather to packet processing functions (PPFs). A PPF is located inside a node and the node itself is addressed by its link layer address. Thus a packet is sent to a node by using the link layer address to reach the node itself and the SAPF selector to address the PPF inside the node which takes care of the actual packet itself. A PPF can be standard packet handling functionality such as forward, deliver or drop but can also be more rich functionality. For example, content adaptation or user-controlled routing.

For this packet forwarding to work within SelNet we must know the link layer address of the node and the selector to communicate with. In a manner similar to the principle behind ARP for Ethernet. A name is specified and this is broadcast into the network, asking for a resolution to a link layer address and a selector. XRP is used for this resolution in the following way. An XRP resolution request (RREQ) is a broadcast of a name to a well-known selector address on the network. This request specifies the name we wish to resolve and how the resolution should be done. This request propagates until it reaches the target, which will reply using an XRP resolution reply (RREP) with its ethernet address. This principle is extended by Daigan to allow for the resolution of a URL into a proxy chain. We note that since SelNet virtualizes the link layer to the upper layers of the protocol stack [6], it is possible to use any datagram service as a "link layer" to SelNet. We have implemented a UDP backend to SelNet to illustrate this.

A. Static Forwarding in SelNet

To introduce the concept of how SelNet functions we are going to describe how forwarding is done in a static environment. Packet forwarding inside SelNet's indirection layer is based on *selectors*. Each SelNet packet carries the selector as an address field. This packet format is defined by SAPF. The selector address, a flat 64-bit value, identifies the function which is to process an incoming packet (similar to a flow or path ID). The payload is handled by whatever function is assigned to the selector in question. Selectors have different values depending on how they are assigned.

Figure 1 shows a scenario where selectors are static and preallocated. In this example an application on Node A wishes to communicate with an application on Node C. In order to do so, the application on Node A opens a socket to SelNet and writes to it, thus invoking the **selnet_demux** operation.

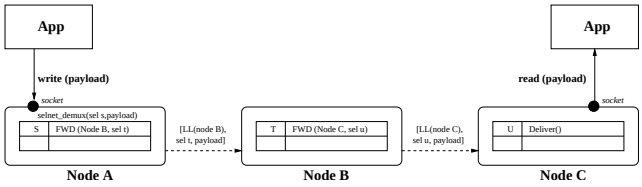

Fig. 1. Forwarding in a static SelNet setup. (FWD= forwarding function, LL=link layer)

When this operation is called, the forwarding function (FWD) is invoked since the selector s is used in this example to communicate with the remote application. The forwarding function performs two tasks: it rewrites the selector from s to t since selector s is only valid inside Node A, and then sends the packet with the rewritten selector to Node B. Selector t on Node B corresponds to the forwarding function which will carry the packet over the next hop to Node C. Once again the selector is rewritten, this time from t to u. When the packet reaches the destination, i.e. selector u on Node C, it is demultiplexed and the payload is passed to the function which is associated with selector u. In this case, it is a local delivery function which passes the payload of the packet through a socket to the application. In section III-B we show how XRP can be used to dynamically resolve a name to a set of PPFs

Note that it is not necessary to rewrite applications to use selector sockets instead of IP. Because SelNet positions itself between the network layer and the link layer, SelNet maps IP addresses to selectors in the same way as IP addresses are mapped to ethernet addresses. Thus, existing applications can still continue to access networking functionality via the IP sockets API although the IP traffic will be carried over SelNet. This functionality was implemented in [7].

III. Daigan Framework

We now present the Daigan framework. It is loosely based upon the MARCH framework detailed in [2]. The purpose of the research presented in this paper is not to create a completely new proxy network solution, but rather to show how existing proposals can benefit from the reuseability of the SelNet indirection layer and also from the native support for indirection that it contains. We first present a general overview of a typical Daigan session, then discuss how the dynamic signalling of SelNet can be used to set up proxies. Then we go into more detail about the components presented in section III-A.

It is important to differentiate between the functionality that SelNet provides and the functionality that Daigan provides. SelNet provides the support for building a chain of proxies and the redirection of application traffic to the proxy chain. Daigan receives the client specification of what the user preferences and device restrictions are and then works out which proxies need to be used to satisfy that request. In other words, Daigan works out what needs to be done and then SelNet does it.

A. Typical Daigan Session

Figure 2 presents an overview of a Daigan session. **(1)** The SelNet client entity intercepts application traffic and instead of forwarding the traffic, contacts the proxy provider. The proxy provider, which is running Daigan, then computes the appropriate proxies necessary to satisfy the user request based upon the URL, user preferences and device restrictions. **(2)** The proxy provider then sends an XRP message to the first proxy in the chain which contains the configuration information for the first proxy and the messages which need to be forwarded on to any other proxies needed to construct the chain. **(3)** The first proxy then forwards the message. **(4)** The proxy provider then returns an XRP message to the client which contains the link layer address and selector of the first proxy in the chain so that the application data can finally be forwarded to the correct proxy.

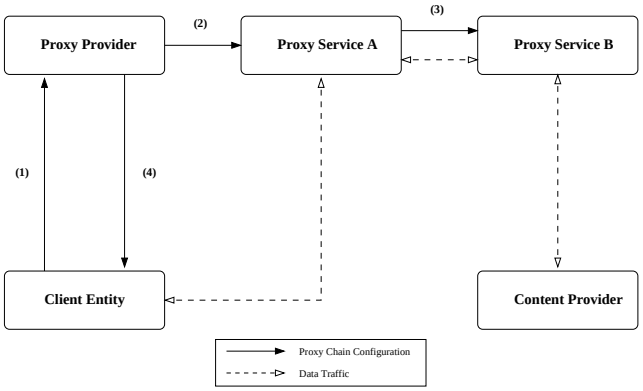

Fig. 2. (1) Prefs sent to proxy provider from client (2) Proxy sends chained XRP configuration to first-hop proxy. (3) Proxy configures itself and relays the next proxy in chain (4) Proxy provider replied with first-hop node+selector pair.

Client Entity, Proxy Provider and Server Entity are defined in sections III-C, III-D and III-E respectively.

B. Selector allocation

Selectors are dynamically assigned by SelNet since selectors only have validity on the node that assigns them. The selectors at each node are bound to a unique session and can be used to set up proxy chains through the network. This is a natural way inside of SelNet to identify a proxy chain rather than overloading the semantics of traditional TCP/IP mechanism such as IP address or port number. To introduce a framework of content adaptation in SelNet we will use the following types of selectors: *forward* functions, *proxy* functions, *transcode* functions and *deliver* functions. The forwarding function, as previously explained, will carry a packet to its destination, the proxy function will help set up a proxy path from the client to the server. The transcode function will perform the content adaptation and pass the transcoded data along the path on the way back to the client. Finally the delivery function will pass the packet payload through a socket to a waiting application.

C. Client Entity

The client entity is a configurable SelNet node which resides at or close[1] to the client terminal. It will sit and listen to the network traffic requested by applications on the terminal. The client entity has been configured with information about the capabilities of the terminal and a ruleset which will decide whether or not it should intercept outgoing traffic. This ruleset can be bound to activate when certain protocols, ports or hosts are accessed.

When the ruleset specifies that the client entity should intercept the traffic, it will try to set up a content adaptation path through the network. This is achieved by contacting a proxy provider within the content adaptation network with information about its own capabilities and which target is requested. The client entity will then proceed to wait for a reply. The reply will contain the first-hop proxy and a unique selector to be used to initiate a session with the server entity over the specially designed adaptation path. When the session has been initiated and transcoded data begins to arrive, the client entity will use its *deliver* function to pass the payload of the packets through a socket to the application.

D. Proxy Provider

The proxy provider is the heart of Daigan. The main task for the proxy provider is to set up an adaptation path from the client entity, through various proxies, to the server entity. This path should be dynamically built to suit the different needs of the many different terminals trying to connect to the server entity. The proxy provider makes a decision based on the information provided by the client and the knowledge of the proxies in its control to create an ordered sequence of the proxies to be used for this specific session. This ordered sequence is then to be imprinted on the chosen proxy nodes. This is where the features of SelNet become very useful. By using the SelNet signalling mechanism to set up the path through the network we will not only be able to define a sequence containing which proxies to invoke, in which order they should be invoked and how to invoke them on a global level. We can at the same time set up a network proxy path from the client to the server, which the client can access simply by sending a packet to the first hop node+selector pair.

The client sends an `XRP_MAS` message to the proxy provider which contains the desired URL and also encodes the user preferences and device restrictions. This is the first XRP message type added to SelNet in order to obtain the necessary functionality for Daigan. The ultimate effect of this message will be the resolution of the URL into a chain of proxies which will fulfill the user requirements. The proxy provider then invokes Daigan to interpret the XRP message and to decide which proxies and which configurations need to be used to set up a proxy chain which satisfies the specification from the user. Once Daigan has computed the proxy chain, it sends an `XRP_PROXY_CONFIG` message to the proxy selector of the

[1]For example a 3G phone which does not run a client entity, could have a client entity located in the 3G gateway as suggested in [2].

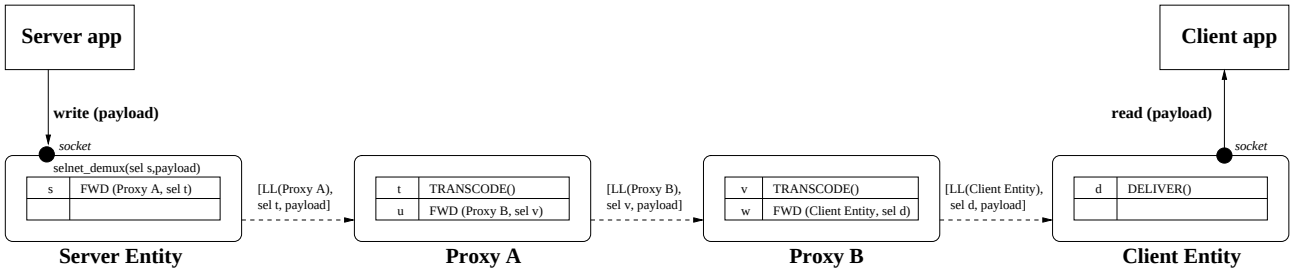

Fig. 3. Content Adaptation in SelNet. (FWD=forwarding function, TRANSCODE=transcoding function, LL=link layer)

first proxy in the chain. This is the second XRP message type added to SelNet. The side effect of this message is the establishment of a transcode selector at the proxy. The transcode selector receives the application data for transcoding. The XRP message contains not only the required configuration information for the first proxy, but also the configuration instructions for the rest of the proxies in the chain. This XRP message is received by the SelNet instance running on the first proxy and configures the proxy appropriately. Once this has been done, the first proxy then forwards the `XRP_PROXY_CONFIG` message to the next proxy in the chain. Typically the last proxy in the chain is a caching proxy which allows inter-operability with legacy server entities. Naturally, if SelNet is running on the server entity, it allows even more flexibility and control. On the way back the server entity will use a *forwarding* function to reach the transcoding selector on the end proxy, which in turn will forward the transcoded data to the next transcoding selector until the data reaches the client entity delivery selector.

By using its knowledge of the client terminal and its control of the different proxies in the network, the proxy provider is able to set up an optimal way through the network. It is in total control of the adaptation process in the sense that only the proxy provider has knowledge of the entire sequence of transformations taking place. Each invoked proxy will only have local information about what itself is supposed to do and its immediate neighbors.

E. Server Entity

The server entity is the node on which the content server runs. When the chain has been created by the proxy provider and the client has resolved a control path traversing the proxies through the network, as discussed above, there is no need for any additional functionality other than being able to receive requests and forward data to the nodes in the network. Figure 3 shows how this forwarding and transcoding takes place on the way back to the client. This process will be explained in detail the next section.

Within the discussed framework it is quite possible to have the server entity and the proxy provider residing on the same physical location within the network. However, we feel that it would be more suitable to let the content provider run a single server entity node to respond to direct requests. The proxy provider, and the proxies under its control, would be more suited to be run by a third party with which the content provider has a service level agreement. This way the proxy provider can cater its functionality to a number of different content providers and still ensure the integrity and copyright issues important to the content provider.

F. Scenarios

We demonstrate a scenario with two proxies. One example of this could be a image transcoding/compression proxy and a TCP header compression proxy. For the sake of brevity we do not include a proxy cache in this scenario, however the way that it would be set up is a trivial extension of the process detailed here. The proxy and adaptation path through the network is initiated when the proxy provider configures the path by creating the *proxy* and *transcoding* selectors on the chosen two proxy nodes. However, the path is not activated until the client entity has resolved a control path through the entire network to the server entity. The control path is established using an `XRP_MAS` message between the client and the proxy provider and `XRP_PROXY_CONFIG` messages between the proxy provider and the proxies which are specifically tailored to trigger the proxy selector on the nodes. Once the XRP signalling is complete, the proxy provider returns an `XRP_RREP` (Resolution REPly) message to the Client Entity. This message contains the selector + link layer address of the first proxy in the chain. Any packet sent to that tuple will be forwarded and processed by all the proxies on the configured path. The packet will exit the proxy network and be forwarded to the Server Entity. Any reply from the Server Entity will follow exactly the same path in reverse to the client, allowing for the content carried in the reply to be processed.

IV. Related Work

Policy-based content adaptation has been discussed in the mobile aware server architecture (MARCH) [2]. The idea of a centralized point, which depending on the operating conditions of the client terminal device, offer content adaptation in the form of a proxy path is also the foundation of our proposed framework. But instead of implementing this negotiation at the application level, we use the extended functionality of SelNet to implement it using the indirection layer i.e., between the network and link layer. Additionally, using SelNet means that we avoid having to identify flows using the IP five or three tuple. Since we have selectors which uniquely identify a proxy

path through the network we do not have the fragility problem associated with having to reuse existing mechanisms for the task. For example, security mechanisms such as IPsec make such a proposal difficult.

Plutarch [8] is a network architecture proposal for bridging disjunct networking contexts to form a cohesive network. Contexts are bridged together using interstitial functions (IFs) and provides indirection by the ability of choosing which context to map a particular packet flow on to. The approach of making the heterogeneity in the Internet explicit and controllable is shared by Plutarch and SelNet. However, Plutarch does not specify mechanisms to perform this task, but leaves it to the actual implementation details of each particular context.

Our approach shares similar goals to the Protocol Boosters work [9]. Especially in terms of improving performance in heterogeneous networks. In some sense, SelNet could be viewed as one way of implementing protocol boosters. However, SelNet does not restrict itself to only hosting network elements which are transparent. Controlled transparency is an important part of SelNet since we believe that embracing middleboxes as first class objects must be a crucial part of any future network architecture.

Multi Protocol Label Switching (MPLS) is an underlay network which uses label switching into the network for faster and simpler packet forwarding. When a packet enters an MPLS network a label is added to it. This label identifies an action for the next hop. When the packet reaches the boundary of the MPLS enabled network the label is stripped away and regular routing is performed. SelNet, like MPLS, also introduces label switching between the network and link layer. An attempt to internetwork SelNet and MPLS to study their behavior have been made in [10]. MPLS distributes its labels among all MPLS routers, whereas in SelNet the labels are local to each SelNet node. MPLS labels are used to address paths, whereas SelNet labels address functions. The last property gives us extended flexibility as to which functionality we can add to the network.

V. Conclusions

In this paper we present Daigan, a tool for constructing proxy networks using the SelNet indirection layer. Daigan decides which proxies are necessary to meet user requirements based upon user preferences and mobile device restrictions. When this has been established Daigan uses the SelNet indirection mechanisms to build the proxy chain and redirect application traffic to the proxy chain. We demonstrate this with a sample scenario showing the setting of two proxies (TCP header compression proxy and image transcoding/compression) using Daigan. Currently we have implemented a simple version of Daigan which can perform the indirection to one proxy. We are currently implementing a dynamic version of Daigan which can perform indirection to multiple proxies which will be completed in the coming months.

The advantages of the approach of using SelNet are that it does not require building yet another indirection system over the existing Internet. The reuseability of the SelNet indirection layer means that other systems requiring indirection functionality do not have to implement it themselves. Another advantage of the SelNet approach is that it does not require overloading the semantics of DNS à la Akamai in order to achieve indirection. Whilst hacking DNS works for the present, it is not clear how this will survive in the future since DNS is becoming evermore fragile as more and more systems encode their particular preferences into it. SelNet also avoids having to reuse existing features of the TCP/IP stack i.e., IP five- or three-tuples, in order to identify flows or network service points. A selector and a link layer address uniquely identify a packet processing function (PPF) and a context where the PPF resides, respectively. This means that SelNet does not have to rely on the semantics of these TCP/IP features remaining static.

We note that the approach proposed in this paper requires that at least the client entity, the proxy provider and the proxies themselves are running SelNet. This could be construed as a disadvantage of the Daigan/SelNet approach when compared to overlay solutions. However, since we are arguing for an architectural change to the Internet i.e., the introduction of an indirection layer, which would allow the Internet to be extended, we do not feel that such a long term view is necessarily a drawback. In [6] we detail an incremental deployment scenario that enables SelNet traffic to be tunnelled through certain areas of the Internet where SelNet cannot currently be installed.

References

[1] B. Knutsson, "Architectures for application transparent proxies: A study of network enhancing software," Ph.D. dissertation, Uppsala University, 2001, http://publications.uu.se/theses/abstract.xsql?isbn=99-3468411-X.

[2] S. Ardon, P. Gunningberg, B. Landfeldt, Y. Ismailov, M. Portmann, and A. Seneviratne, "MARCH: a distributed content adaptation architecture," *International Journal of Communication Systems, Special Issue: Wireless Access to the Global Internet: Mobile Radio Networks and Satellite Systems*, 2003.

[3] M. Fry and A. Ghosh, "Application level active networking," *Computer Networks*, 1999.

[4] P. Keleher, S. Bhattacharjee, and B. Silaghi, "Are virtualized overlay networks too much of a good thing?" in *1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, 2002, http://www.cs.rice.edu/Conferences/IPTPS02/169.pdf.

[5] C. Tschudin and R. Gold, "Network Pointers," in *ACM Workshop on Hot Topics in Networking (HotNets-I)*, 2002.

[6] R. Gold, P. Gunningberg, and C. Tschudin, "A virtualized link layer with support for indirection," in *ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA)*, 2004, http://user.it.uu.se/~rmg/pub/fdna05-gold.pdf.

[7] C. Tschudin, R. Gold, O. Rensfelt, and O. Wibling, "LUNAR: a Lightweight Underlay Network Ad-hoc Routing Protocol and Implementation," in *Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN'04)*, 2004.

[8] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield, "Plutarch: An Argument for Network Pluralism," in *ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA)*, 2003.

[9] D.C.Feldmeier, A. McAuley, J. Smith, D. Bakin, W. Marcus, and T. Raleigh, "Protocol boosters," *IEEE JSAC, Special Issue on Protocol Architectures for 21st Century*, 1998.

[10] A. Westling, "Internetworking MPLS and SelNet," Uppsala University, Tech. Rep. 2004-013, 2004, http://www.it.uu.se/research/reports/2004-013/2004-013-nc.pdf.