Generic Proxying as a Deployment Tool (GEPROD)
draft-nikander-ram-generix-proxying-00

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on July 28, 2007.

Copyright Notice

Copyright (C) The IETF Trust (2007).

Abstract

This document presents a generic way of using Forwarding Proxies, designed to be used as a transition mechanism in implementing various flavors of the so called Identifier / Locator separation, including both "above IP" and "below IP" approaches.

This version of this draft is an very incomple version, intented to induce discussion.
# Table of Contents

1. Introduction ................................................. 3  
2. Terminology ................................................. 3  
2.1. Terms common to other documents ......................... 3  
3. Background .................................................. 3  
4. The Problem .................................................. 4  
4.1. Basic approaches ........................................... 5  
5. Architectural approaches to identifier / locator split ... 6  
5.1. Using locators at transport and IP layer ............... 6  
5.2. Rewriting within the transport or ‘upper’ IP layers .... 7  
5.3. Rewriting within the network ............................ 8  
6. Using Proxies as an interconnected medium ................ 9  
6.1. Forward proxying in the ‘above’ transport approach .... 9  
6.2. Forward proxying in the ‘in-stack’ approach ............ 10  
6.3. DNS with forward proxying ............................... 12  
6.4. Reverse proxying ......................................... 13  
6.5. Un-proxying in the ‘in-the-network’ approach .......... 13  
7. Stretching the wire .......................................... 14  
8. Security considerations ..................................... 15  
9. Discussion ................................................... 15  
10. IANA considerations ........................................ 16  
11. Acknowledgments ............................................ 16  
12. Informative references ..................................... 16  

Author’s Address .............................................. 16  

Intellectual Property and Copyright Statements ................ 17
1. Introduction

Discussions at the IAB Routing and Addressing workshop, followed by discussions at the 67th IETF in San Diego, have revealed a potential problem in our ability to keep growing the routing and forwarding tables in the core routers in the default free zone. This, in turn, has led to a lively discussion on various architectural options towards solving the problem. A tentative consensus seems to be that implementing the so called identifier / locator separator by injecting a new naming layer either "below" or "above" the current routing layers (IPv4 and IPv6) appears to be one very potential architectural approach. However, there seems to be no consensus on the stack-wise location or semantics of the resulting new name space.

In this document, we discuss an implementation approach that can be used as a transition tool to help introducing such a new name space. The approach can be used for injecting a new layer both "below" or "above", though in a slightly different manner. Both approaches are discussed.

2. Terminology

2.1. Terms common to other documents

<table>
<thead>
<tr>
<th>Term</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>End-point</td>
<td>A communicating entity.</td>
</tr>
<tr>
<td>Locator</td>
<td></td>
</tr>
<tr>
<td>Identifier</td>
<td></td>
</tr>
<tr>
<td>Legacy address</td>
<td></td>
</tr>
<tr>
<td>Upper Layer Protocol</td>
<td></td>
</tr>
<tr>
<td>Upper Layer Protocol Identifier (ULID)</td>
<td></td>
</tr>
</tbody>
</table>

3. Background

As discussed elsewhere (cf. RFC4423 [1]), the current IPv4 and IPv6 name spaces can be considered to be confoundings of two semantically different name spaces: interface locators and node identifiers. In other words, an IP address functions both in a locator role,
identifying a location of an interface in the terms of the forwarding
name space topology, and in an identifier role, identifying the
communicating end-point (node) for transport and other upper layer
protocols. The need for or non-desirability of separating these two
functional roles have been discussed in the IETF for the past ten
years or so.

Past and present proposals for implementing the so called identifier
/ locator split include at least the following:

HIP, where current IP addresses are demoted to mere locators and a
new name space of identifiers is created. In HIP, the identifiers
are essentially hashes of public cryptographic keys.

SHIM6, with two variants, where in both IP addresses continue to
be used both as identifiers and locators, but where the identifier
addresses are formed either from a fixed set of routing prefixes
and the also including a verifiable reference to public key into
the identifier address.

Others to be added. Please provide text.

In this document, we take the stance that in order to be
architecturally sound, any layer-injecting solution must eventually
be implemented by all hosts, at least in an conceptual sense. ... XXX Elaborate.

When considering any such architectural solution, it becomes clear
that any eventual solution must take care of a longish transition
period where some part of the network has implemented the identifier
/ locator separation while another part of the network has not.

In this document, we describe how the transition period can be eased
by introducing proxies into the network. These proxies represent the
'new' part of the network to the 'old' part as if the new part still
had not implemented the separation, and vice versa.

4. The Problem

The identifier / locator split introduces a new name space to the
networking architecture, either "below" the current IP addresses or
"above" it. In any case, the result is an architecture that has two
separate naming layers: a (node) identity naming layer, and an
(interface) locator naming layer; see Figure XXX.

Now, consider a situation where the network is partitioned into two or
more parts, some of them implementing the new approach and some not.
For brevity, we call ‘old’ the parts of the network that have not (yet) implemented the identifier / locator separation (whatever it is), and ‘new’ those parts of the network that have implemented the separation. Note that depending on the the implementation details, being ‘new’ may imply that the hosts implement the new functionality, that the routers implement the new functionality, that there are new boxes that implement the new functionality, or some combination thereof; this will become clearer shortly.

As a result of partial deployment, those hosts that are located in the ‘old’ part of the network continue to send packets that in their source and destination fields have the old semantics. At the same time, those hosts that are in the ‘new’ part of the network send (either directly or due to conversion deeper in the network) packets that eventually will have mere locators in their source and destination addresses.

In this world, we have the following four situations:

An ‘old’ host sends a packet to another ‘old’ host. The packet contains legacy addresses in both source and destination fields. As long as there still exists routing infrastructure that connects the hosts, there are no problems.

A ‘new’ host sends a packet to another ‘new’ host. The packet (will eventually) contain locators in the source and destination fields. As long as the mapping from identifiers to locators and vice versa works as it should, at both ends, there are no problems.

An ‘old’ host attempts to send a packet to a new ‘new’ host. As a result of this, the outgoing packet will have the ‘new’ host’s identity or locator in the destination field, while the source field contains the ‘old’ host’s legacy address. There are a number of details here that differ from approach to approach.

A ‘new’ host attempts to send a packet to an ‘old’ host. The source field will now be the a locator. There are a few options for what the destination field will contain; see the discussion below.

In this document, we discuss the latter two situations and describe a number of particular ways of solving them, based on proxies.

4.1. Basic approaches

To facilitate ‘old’ hosts sending packets to ‘new’ hosts and vice versa, we can construct the following taxonomy of approaches.
'Old' sending packets to 'new'

'New' still directly reachable through the legacy routing system.

'New' no longer directly reachable through the legacy routing system.

'New' sending packets to 'old'

'New' still having direct access to the legacy routing system.

'New' no longer having direct access to the legacy routing system.

These cluster naturally into two, depending on whether the the 'new' host has direct access to the legacy routing system or not. In this document we only look at the latter cases, where the 'new' host is only connected to the 'new' network and the 'old' system is only connected to the 'old' network, thereby needing an intermediate box to interconnect them.

5. Architectural approaches to identifier / locator split

Architecturally, the identifier and locator roles of IP addresses can be separated at three different locations within the IP stack:

Above the transport layer; e.g., at a library that implements the sockets API.

Somewhere within the transport layer or the "upper" part of the IP layer; i.e., logically within the host stack "proper". Examples of this approach include SCTP multi-homing, HIP, and SHIM6.

In the network. From a host point of view, it doesn’t make a big difference if this change takes place at the device driver or in a separate box well within the network. However, in order to better match with the mind set of the 'jack up' proposals, below we introduce this model in terms of two boxes, and then "un-proxy" it by moving the functionality to the host itself.

5.1. Using locators at transport and IP layer

This approach has received relatively little study.

One fundamental problem is the need for TCP to resynchronise in the case of locator change. That is, if the layer above detects a need
to change locators, it needs to open a new TCP connection that uses the new locators. If the old TCP connection was not properly shut down, the hosts cannot be sure how much of sent data the other host received, necessitating a new resynchronisation mechanism.

Figure 1

5.2. Rewriting within the transport or ‘upper’ IP layers

Approaches belonging to this class have received most scrutiny to date.
Characteristic to these approaches is that the data items at the socket API, down to the transport, are identifiers, and the packets passed between the ‘upper’ and the routing and forwarding parts of the IP layer have locators. That is, somewhere between the ‘top’ of the transport and the routing and forwarding function the identifiers get rewritten to locators and vice versa.

Proposals that more-or-less follow this approach include the following:

- SCTP multi-homing and add-IP
- HIP
- SHIM6

5.3. Rewriting within the network

This approach has been named ‘jack up’ by Noel Chiappa in the late 2006 / early 2007 IAB architecture-discuss mailing list discussions.

From the outset, the apparent property of these approaches is that it feels natural to perform the rewriting at the network side. In other words, a main motivator for the proposals in this category is to keep the hosts intact by providing the functionality with extra boxes at
the network side.

In this approach, there is a local subnetwork between the unmodified host and a rewriting box. The rewriting box implements the ‘new’ functionality for the whole subnetwork, thereby alleviating the need to change the individual hosts.

However, as will become apparent momentarily, the ‘network side’ property of these approaches is more circumstantial than characteristic in the sense that also the other two approaches can be implemented at the network side. Hence, the main property of this category is that the rewriting is done at the ‘routing’ side of the IP stack, i.e., after IPsec, fragmentation, and reassembly, rather than ‘above’ those functions.

6. Using Proxies as an interconnected medium

In this section we momentarily assume a network setting where most hosts are ‘new’ hosts, and we just have to take care of interconnecting a few ‘old’ hosts to the ‘new’ world via proxies. We will return to other cases, e.g., where there are just a few ‘new’ network segments while most of the world is still using confounded identifiers and locators, in a later section. Hence, in this section we assume that there is a local subnetwork, or even a direct wire, between the ‘old’ host and a proxy.

Common to all proxy-based solutions is that there is a local subnetwork where the source and destination addresses in IP packets contain identifiers. In other words, the rewriting function has been moved from the host stack into a separate box in the network. Respectively, when ‘un-proxying’ the ‘jack-up’ approach, the rewriting function is moved from the separate box in the network into the lower part of the host stack.

6.1. Forward proxying in the ‘above’ transport approach

Architecturally, these approaches are quite similar to the existing TCP ‘splitters’, with the difference that the identifiers get rewritten to locators and vice versa.

The properties of this class of approaches have not been analysed at all; the main motivation for illustrating this to be architecturally complete and present all alternatives.
6.2. Forward proxying in the ‘in-stack’ approach

As above, the idea here is to move the rewriting function from the host stack into a distinct box in the network. Depending on where exactly within the transport/internetworking the rewriting would be done at the host side, the proxy needs to implement different amounts of functionality and state.

Let us consider, as an example, the case of HIP and SHIM6. There the rewriting is performed roughly colocated with IPsec at the stack; i.e., at the point when the host is ready to send an outbound packet but before the packet gets forwarded towards a specific interface, or where the host has decided that an inbound packet is destined to itself and not to be forward.

In the proxied approach, instead of rewriting the packet the host either sends an outbound packet, unmodified, to the proxy, or accepts an inbound packet from it. The proxy, in turn, maintains a rewriting table, replacing the locators with identifiers on inbound traffic and picking suitable locators for outbound packets. If the host has multiple interfaces, each interface should be logically configured with the same local IP address, since the goal is to have just one identifier for each host, and all the proxies at the various local paths must perform compatible rewriting.

From a logical point of view, this approach can be illustrating by ‘strething’ the host stack to the new middle box. That is, we split
the host stack at the rewriting function (wherever it happens to be), leave everything above at the host itself, and move everything below to the proxy. This is illustrated in Figure XXX. Physically, of course, the host stack remains complete and the 'stretching' is performed by the local subnetwork; see Figure XXX.

![Diagram of host and proxy stack](image)

Figure 5
6.3. DNS with forward proxying

In the forward proxying approaches above, we have assumed that the local wire between the ‘old’ host and the proxy carries IP packets that have identifiers in their IP header. To work nicely with the DNS, this may require minor DNS filtering or proxying, depending on how the locators and identifiers are stored in the DNS.

If the identifiers are stored in the DNS as AAAA (or A) records, and locators under some other record type, there is no need to play tricks with the DNS. The ‘old’ host, making a DNS query (through the proxy) will get an AAAA record containing the identifier, and everything will work as if the remote host had that identity as its IP address. Respectively, the host’s identity will be stored at the public DNS as an AAAA record along with the proxy’s locators.

If the locators are stored in the DNS as AAAA or A records, and the identifiers under some other record type, the proxy (or some other box) must provide translation so that the ‘old’ host will get only the identifier as a response to its address queries.

If both locators and identifiers are stored in the DNS as AAAA or A records, then it may suffice to order the records so that the identifiers come always first, or it may be necessary to filter the DNS responses.
6.4. Reverse proxying

In the approaches above we assumed that we were connecting ‘old’ hosts to a predominantly ‘new’ world, by providing the new functionality at a proxy near the ‘old’ host. We now consider the situation where the goal is to connect a completely ‘new’ host, one that no longer has direct ‘old’ connectivity, to a predominantly ‘old’ world. In this case we need a proxy, but this time a different one. This proxy will represent a huge number of ‘old’ hosts, out there in the Internet, towards the ‘new’ host.

In this case, the actual proxy structure and functionality does not differ much in the architectural sense. The proxy still provides the ‘new’ functionality to the ‘old’ hosts. The difference is that the proxy has no a priori information about the identity of the ‘old’ hosts, and cannot be assumed to have big enough stable storage to represent them all. Secondly, it must be assumed that there may multiple of such ‘reverse’ proxies in parallel and it is not possible to provide coordination between them.

To provide connectivity from ‘new’ hosts to ‘old’ hosts, the proxy must implement active DNS proxying. That is, when a ‘new’ host makes a DNS query asking for the identity and locators for an ‘old’ host, the proxy answers with a fabricated identity and its own locators. Correspondingly, when an ‘old’ host makes a DNS query asking for the IP addresses of a ‘new’ host, the proxy provides with its own IP addresses.

Hence, while there is little difference in handling the actual IP packets, other than using fabricated rather than stable identifiers for the ‘old’ hosts, there are relatively large operational differences between ‘forward’ and ‘reverse’ proxying.

Note that a reverse proxy can be a distributed one.

6.5. Un-proxying in the ‘in-the-network’ approach

In the same spirit but in the reverse fashion, the ‘in-the-network’ or jack up approaches can be implemented within the host instead of in a separate box. In this case, we ‘squeeze’ the local subnetwork into the host itself, making it to physically disappear.
In these approaches, the host stack ‘proper’ handles only identifiers. Even when the packets enter the routing and forwarding part of the IP layer they still contain identifiers. The identifiers get rewritten to locators, and vice versa, somewhere ‘below’ the host IP; for example, a "shim" layer between the IP layer and the device layer can do this, somewhat similar to the so called bump-in-the-stack (BITS) IPsec implementations.

7. Stretching the wire

So far we have considered moving the rewriting functionality either out from a host to the network or vice versa, considering only a local piece of wire between the host and the proxy. We will now extend the discussion into other configurations, where there may be a larger piece of network between the old host and the proxy.

When using a proxy at the local network, there are few restrictions on addressing. Basically, it suffices if the identifiers can be locally distinguished from locators. All outgoing packets containing identifiers can be processed by the proxy, as needed. The ‘old’, proxied hosts can use their own identifiers as their local IP addresses.

However, when stretching the wire between an ‘old’ host and a proxy, routing becomes involved. In a word, it becomes a necessary to somehow route packets that contain identifiers in IP headers. There
are multiple potential approaches to this, depending on the exact nature of the identifiers and on how far the packets need to be routed:

The easiest case is when the identifiers are fully routable, like in SHIM6. In that case there are few limitations where and how the proxies can be located at, as long as they are on the default path between the ‘old’ and ‘new’ hosts. For example, each site exist could be equipped with a SHIM6 proxy, knowledgeable about all the other prefixes the site has, and being able to redirect the traffic on demand.

Even if the identifiers are not fully routable but still contain a prefix system that allows them to be routed towards either the ‘old’ hosts or the proxy, forward proxying is relatively easy to arrange.

Add more cases here.

It is always possible to arrange ‘reverse’ proxying, at any location in the network, at an added operational cost.

8. Security considerations

9. Discussion

The main message in this draft is that there is only little architectural difference between implementing the identifier / locator split ‘within’ transport/host IP layer or ‘below’ the host IP layer. In other words, while at the outset it appears that a ‘host’ based id/loc split must be implemented in a host and a ‘network’ based id/loc split must be implemented in separate boxes in the network, the ‘implementation’ location is actually relatively orthogonal to the ‘stack’ location. Even ‘host’ based approaches, such as SHIM6 or HIP, can be implemented within the network with the help of proxies. And ‘network’ based ‘jack-up’ approaches can be implemented in a host, by providing the host with sufficient information.

The real differences are in trust, state, and operational models.

Considering trust, ‘host’ based approaches grow from the assumption that the host is in the best position of representing itself as an integral identity. Conversely, they may be more vulnerable to possession attacks than other approaches. Respectively, ‘network’ based approaches grow from the assumption that the hosts cannot be trusted (as, e.g., they may be compromised) and therefore it is
better for the network to provide the functionality.

In reality, if high security is needed it seems like a good approach to combine both: start by assuming that the hosts represent themselves as integral entities having an identity but operationally forcing the hosts to delegate the representational functionality into the network. That is, in a high security environment it may be best to force the hosts to use a proxy. This divides the responsibility and therefore lowers risks.

TBD: Discuss state and operational models.

10. IANA considerations

This document has no actions for IANA.

11. Acknowledgments

12. Informative references


Author’s Address

Pekka Nikander
Ericsson Research Nomadic Lab
JORVAS FIN-02420
FINLAND

Phone: +358 9 299 1
Email: pekka.nikander@nomadiclab.com