A New Location Layer for the TCP/IP Protocol Stack

André Zúquete
DETI, IEETA / IT, University of Aveiro
Portugal
andre.zuquete@ua.pt

Carlos Frade
IT
Portugal
gilfrade@hotmail.com

ABSTRACT

The IPv4 address space is quickly getting exhausted, putting a tremendous pressure on the adoption of even more NAT levels or IPv6. On the other hand, many authors propose the adoption of new Internet addressing capabilities, namely content-based addressing, to complement the existing IP host-based addressing. In this paper we propose the introduction of a location layer, between transport and network layers, to address both problems. We keep the existing IPv4 (or IPv6) host-based core routing functionalities, while we enable hosts to become routers between separate address spaces by exploring the new location header. For a proof of concept, we modified the TCP/IP stack of a Linux host to handle our new protocol layer and we designed and conceived a novel NAT box to enable current hosts to interact with the modified stack.

Categories and Subject Descriptors

C.2.0 [**Computer Communications Networks**]: General—*Open Systems Interconnection reference model (OSI)*; C.2.1 [**Computer Communications Networks**]: Network Architecture and Design—*Store and forward networks*; C.2.2 [**Computer Communications Networks**]: Network Protocols—*Protocol architecture (OSI model), Routing protocols*; C.2.5 [**Computer Communications Networks**]: Local and Wide-Area Networks—*Internet*; C.2.6 [**Computer Communications Networks**]: Internetworking—*Routers, Standards*

General Terms

Design, Standardization

Keywords

Internet, addressing, identification, routing, location

1. INTRODUCTION

The end-to-end paradigm of the original IPv4 proposal [26] assumes an addressing space where an IP address, a 32-bit number, by itself identifies unequivocally an end-host. IP addresses are formed by two parts, a network part and a host part, which are used for routing optimization (namely, for reducing routing tables on core routers).

For tackling the fast depletion of the IP address space in the last decade of the last century, Wang & Crowcroft [31] proposed a two tier addressing space, using internal and external IP addresses. In their proposal, hosts could use one or both types of address, but external addresses should be used only if required and hosts could allocate them on a needed basis from a specific server. Therefore, the Internet could be separated in one global Internet and many, isolated private networks, with no specific, well-defined bridging between any of them.

The separation of the IP address space in private and public addresses was effectively adopted [28] but, again, no mechanisms where defined, at the IP level solely, for allowing hosts from private networks to interact with hosts in the public network, or for allowing independent private networks to interact among themselves.

Nevertheless, Network Address Translation (NAT [9]) was already defined and could be used to enable hosts in private networks to initiate interactions (TCP streams or UDP query/response dialogs) with hosts in the public network. The bridging among the address spaces was possible at the expense of changing transport ports (or other flow selectors) and keeping state on the hosts performing the NAT bridging (NAT boxes).

A NAT box screens a network of hosts behind a single IP address, i.e. uses the same public IP address for identifying (simultaneously) a set of hosts, which was a modification of the original IP addressing paradigm (one IP address, one host). The immediate consequence of this fact is that hosts screened by a NAT box cannot be deliberately addressed by hosts in the public network. This may be an interesting feature for security purposes (prevents hosts from receiving unattended requests), but it does also introduces limitations when hosts effectively want to be contacted from outside their private network. Port forwarding partially mitigates this limitation, allowing transport ports to be bound to screened hosts.

1.1 Goal

Our goal is to have a mechanism for allowing end-hosts on any IP network, public or private, to address each other without limitations. The key feature that we explore is a novel hierarchical routing mechanism, inspired in two existing routing concepts: source routing and route recording. With our hierarchical routing mechanism, we imagine the Internet to evolve hierarchically (see Figure 1), just like DNS, from an IPv4 or IPv6 public backbone to several private leaf networks, each of which with a focused purpose, possibly interconnected by intermediate, private mid-size networks.

Furthermore, IP addresses do not need to be omnipresent in all networks, and we may also have either IPv4 or IPv6 networks. Consequently, we propose a new identification paradigm for Internet endpoints, other than the current IP

addresses, because we want to be independent from them for identification. This identification paradigm enables a complete separation of entity identification from entity location, while keeping the current Internet host identification paradigms (either IPv4 or IPv6).

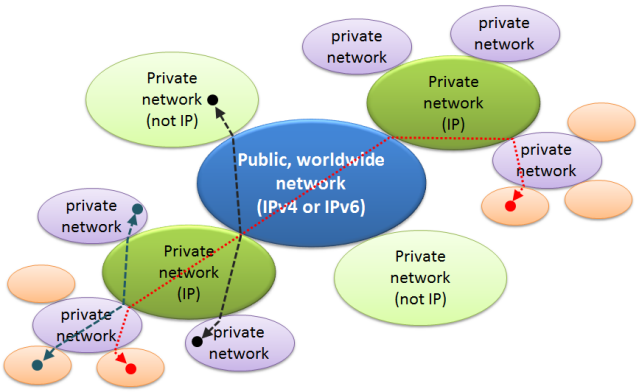

Figure 1: Hierarchical Internet vision, with a central, IPv4 or IPv6 public backbone and many hierarchies of private networks, using IP or not. The diagram shows three interactions among pairs of endpoints, both located in leaf, private networks, and a hierarchical traffic routing between them passing through a public or private IP network, indirectly accessible to both entities.

Our goal is not to replace current routing algorithms in existing networking domains, such as IPv4 or IPv6. Instead, our goal is to provide end-to-end, overlay routing mechanisms enabling datagrams to travel across different networking domains following a source routing, multi-hop strategy.

In addition, we recognize that the possibility of using the Internet for addressing endpoints that may not have a public IP address, or even an IP address, opens new addressing possibilities, namely the possibility to address content. In our opinion, an hierarchical, heterogeneous addressing paradigm, such as the one we propose, may help to implement content addressing, because (i) hierarchical addresses may accommodate different addressing components, such as identifiers for content or for content providers, and because (ii) it may enable dynamic and alternative routing strategies across a hierarchical infrastructure between content providers and consumers.

1.2 Contribution

This paper presents a new layer for the TCP/IP protocol stack. The new location layer stays between the network and transport layers. Its purpose is to enable the deployment of addressing bridges on top IP. Such bridging enables a seamless routing between many types of heterogeneous addressing domains, such as public/private IPv4 and IPv6 networks (but not limited to these ones).

The location layer will be responsible for adding flexible addressing, location and routing facilities for IP packets. This layer allows any IP host to become a locator node between an IP host and some "addressable entity", which can also be another IP host. Addresses handled by these agents are flexible data structures that use a stacking paradigm to memorize routes (e.g. the next IP hop) and/or entity identifiers (e.g. a latitude-longitude location for a wireless sensor or a signed content hash). We propose a simple meta-structure for implementing location layer headers but we do not impose specific policies to manage the activities of a locator node.

The location layer will also be responsible for an effective separation between endpoint identification and location. Traditionally, IP addresses are used for host identification and location. Since with the new location header we may support the interaction between endpoints belonging to different address spaces, endpoints are identified within a new, wide identification space, which is totally location-free, and such identifiers are handled on the context of the location layer.

For a proof of concept, we implemented this layer on Linux kernels and we exploited it for implementing an alternative NAT mechanism. Our NAT box, which is now a host locator, is totally stateless, allows hosts in a private and public network to address each other, in both directions, with no limitations of any kind and, finally, does not need to interfere with upper protocols layers to operate properly. This is a striking difference to the currently existing NAT, which is stateful, mainly unidirectional, and requires transformations of packet contents belonging to upper protocol layers.

Just for demonstration, we describe a simple, world-wide content addressing infrastructure using our location layer for routing content consumers to content suppliers or some intermediate content caches.

1.3 Deployment and transition issues

The exploitation of our location layer does not require any support from the Internet core infrastructure, either IPv4 or IPv6. Therefore, the workload and cost of its deployment for core network operators is null.

The handling of location headers only needs to be deployed on the edges of the Internet (personal computers and servers), and possibly on Internet edge routers (ISP's). Although massive, such deployment can be mostly automated using the nowadays common operating system update tools. Existing applications do not stop working because of the new header, because we keep all the protocols they currently use (IP, TCP, etc.). We will come back to this topic on Section 7, after describing our prototype implementation.

2. CONTRIBUTION: LOCATION LAYER

The location layer will optionally exist between the network layer (IP) and the transport layer. The purpose of the location layer is to enable the deployment of overlay addressing, location and routing strategies over IP.

The location layer changes the way an endpoint is identified and reachable within the Internet. In the classical IPv4/IPv6 addressing and identification paradigm, an IP address is both a host identification tag (enabling a unique pinpointing of a host) and a host location tag (enabling the routing of a packet to the end-host network and, from there, to the end-host). With the location layer, we clearly separate endpoint identification from location and routing information (cf. Fig. 2):

- An endpoint is uniquely identified with a unique identifier (Endpoint Unique Identifier, EUI) placed in the new location header. An EUI can be used to identify hosts or other Internet endpoints.

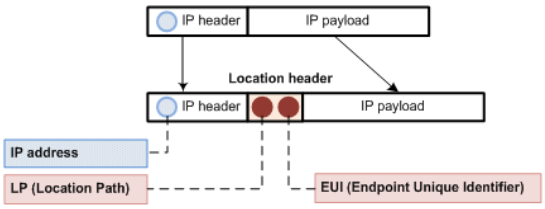

Figure 2: Diagram showing the placement of the Location Header and the fields used to identify and locate an Internet endpoint: the Endpoint Unique Identifier (EUI) for endpoint identification, the IP address (of a Location Node, LN) and the Location Path (LP) for endpoint location (through the LN).

- An endpoint is reachable using the IP address of a Locator Node (LN) and a Location Path (LP). The IP address of LN is an ordinary IP address, belonging to an IP header, and the LP is a variable-length, opaque byte array located in the location header.

The purpose of the EUI is to provide a constant, location-independent endpoint identification along packet exchanges belonging to the same logical session (e.g. a TCP virtual circuit). By using the EUI for identifying session endpoints we are also anticipating a potential modification on the location of an endpoint during a session, which may imply a modification on the LN and LP values used by the peer to reach it.

The location header contains 6 fields: next protocol and header length; source and destination EUIs; source Location Path (SLP) and Destination Location Path (DLP).

SLP and DLP are variable-length, implementation-free stacks of location information used by locator nodes. These push routes into SLP and pop routes from DLP. This way, SLP and DLP together enable the deployment of multiple, heterogeneous stackable routing policies. SLP and DLP are similar to the IP optional fields Record Route and Strict Source Routing [26], respectively, but they can store any data, not only IP addresses, and they are managed by locator nodes, not by IP routers. This topic will be further addressed in Section 2.5.

In the example of Fig. 3 we show how node A and B (for instance, two smartphones) establish a VoIP session. First, node A finds a path to B, using name services and brokers. For instance, for this kind of service both A and B can adhere to a naming schema suitable for the purpose of exploiting VoIP with mobile phones, such as the existing, hierarchical phone numbering space (e.g. country code plus national area code plus phone number). At the end of the search, A will get B's EUI (if not known from start), the path to reach B from where A stands, through several locator nodes (LP_AB), and the nearest locator (LN1), the one that would be the first to interpret LP_AB. The exact protocol to obtain all this data is out of the scope of this paper. We anticipate that many different name services may be used to provide this data for different purposes, therefore it would be shortsighted to describe a single one in this paper that could be used for all purposes (note that DNS is not also the unique Internet name service capable of translating names into IP addresses).

Then, A initiates the interaction with B by sending packets to the nearest LN (LN1) and using the location header for including its EUI and B's identification and location path (EUI_B and LP_AB). LP_AB will be progressively dismantled by locators LN1, LN2 and LN3, until reaching B. Simultaneously, they create in SLP a route back towards A through themselves. When the packets finally reaches B, it can extract A's EUI (e.g. its phone number) from the location header and A's location path (LP_BA) to send packets back through LN3. Using these, B can send packets that will travel again through LN3, LN2 and LN1 until A. At this point, A can update the current LP_AB for B, and the same can do B with LP_BA of the next received message.

2.1 Semantic and generation of an EUI

An EUI is a semantic-free value used only for singularization of endpoints. In other words, it should be used only for discriminating endpoints, although it may be used by locator nodes to manage session routes. Consequently, an EUI is a random 128 bit value; it can be generated by anyone without any coordination with others.

A host can use a particular EUI for self-identification in several interactions using different network interfaces, possibly explored simultaneously. Or, on the other way around, a host can use as many EUI values as wanted, possibly simultaneously, in order to get some added value. For instance, a host can use a different EUI per session (e.g. UDP query/response, TCP virtual circuit, HTTPS session) in order to complicate traffic aggregation analysis.

2.2 Global vs. local addressing paradigms

The IP addressing paradigm assumes that public addresses have the same interpretation (identify the same host) everywhere. On the contrary, private IP addresses have a local interpretation and can be bound to different hosts belonging to different domains (or private networks).

With the location header, we enlarge the possibilities to identify a host, which can now happen with:

- **Direct identifier**: an ordinary IP address.
- **Indirect identifier**: an EUI, complemented by the direct identifier (IP address) of a locator node and a DLP. We anticipate that these last two components may be updated for the same EUI, in order to facilitate the roaming of endpoints identified this way.

An indirect identifier introduces a new paradigm in the way endpoints can be identified in the Internet, as it allows them to be globally identified using values that have a local interpretation. Local interpretation means, in this context, that the interpretation of an indirect identifier depends on the particular set of locator nodes that carry on its interpretation, and different endpoints can use different locators for referring the same target endpoint.

This type of semantic allows the exploitation of the Internet to reach global resources that should not be bound to IP addresses (e.g. a TV show, a device bound to a phone number, etc.) using identifiers that change from client to client, depending on the way the resource access is provided.

2.3 Impact in upper protocol layers

Transport layers, such as UDP and TCP, are naturally very intertwined with the IP paradigm. For instance, a TCP virtual circuit is uniquely identified using a tuple containing two IP addresses and two TCP ports. The inclusion

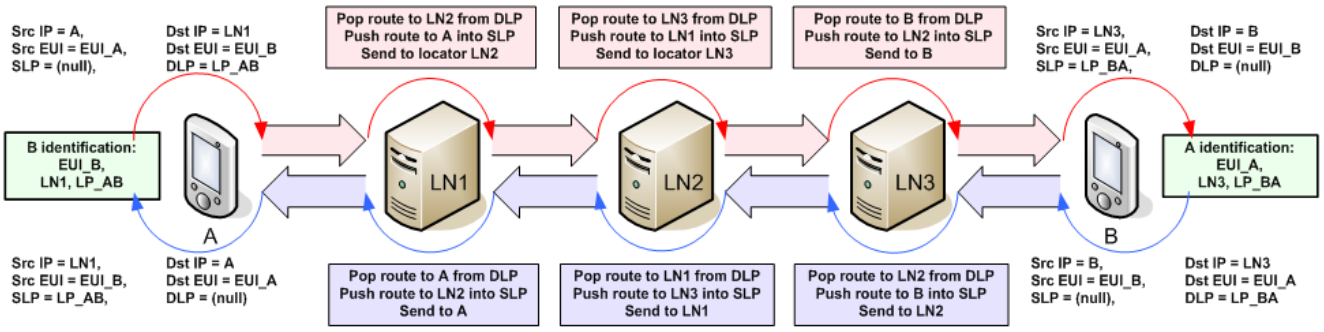

Figure 3: Example of how traffic can be routed from A to B and vice-versa using three locator nodes (LN1, LN2 and LN3) and the location header. Locators push location information into SLP fields and pop location information from DLP fields. The information pushed and popped is defined by each locator node.

of the location layer introduces an alternative paradigm for identifying end-hosts: their EUI.

Therefore, IP-based transport protocols, such as UDP and TCP, will have to deal with two types of endpoint identifiers: direct and indirect. When using direct identifiers (ordinary IP addresses) they should use them with ports for matching endpoints and for addressing endpoints. When using indirect identifiers, they should use EUIs and ports for matching endpoints and should use locators' IP addresses and location paths for addressing endpoints. Consequently, with the introduction of the location layer, transport endpoints (sockets) naming keeps IP addresses and ports and includes three extra fields: the local EUI, the peer EUI, and the peer location path (closest locator node IP and peer DLP).

Another issue for upper protocol layers is their use of IP addresses for computing checksums. In fact, protocols such as UDP and TCP have on their header a checksum field that is computed from the transport data (header and payload) prefixed by a pseudo header. As this pseudo-header includes the source and destination IP addresses, changing IP addresses along locators, as displayed in Fig. 4, creates checksum validation errors at receiving end-hosts, ultimately leading to a discard of all UDP and TCP packets. A solution for this checksum issue is to completely remove data of lower layers from the checksum computation; this means removing IP addresses from it.

2.4 EUI spoofing issues

The addressing space of EUIs is large enough for preventing their involuntary collision. Even in such case, collision is only a problem when it creates confusion, e.g. when a server host receives two TCP segments from the same EUI and using exactly the same source and destination ports, but not necessarily the same source IP address.

However, attackers may explore this sort of collisions to interfere with others' communications: we can call it EUI spoofing. But this is not exactly a new threat for the Internet, since IP spoofing is also possible nowadays. Currently, the Internet only ensures source authentication for received data (using IP) when some sort of secure communication channel is used over IP (IPSec [20], SSL [7], etc.).

The location layer may also be properly explored in order to reduce the risk of EUI spoofing attacks. First, EUIs of endpoints initiating a remote session should be random whenever possible, for reducing spoofing capabilities. Second, TCP and UDP endpoints may enforce a strict constancy of the peer indirect address (EUI, IP of locator node and DLP) on all received datagrams belonging to the same session. This way, EUI spoofing would not be enough for interfering with an existing session; IP spoofing and knowledge of the DLP used by the spoofed endpoint would also be required. This way, some anti-spoofing techniques used for IP (e.g. ingress filtering [12]) could also prevent the spoofing of indirect addresses.

2.5 Routing security issues

Strict or Loose Source and Record Routing (SSRR, LSRR) and Record Routing (RR) are three options of the current IPv4 standard. SSRR and LSRR enables the datagram sender to specify (force with SSRR, suggest with LSRR) the ordered set of IP addresses of the hosts responsible for routing the packet until its destination. SSRR, LSRR and RR enable a datagram to record the IP address of all its actual routers. IPv6 also has support for a Routing Header (RH) extension, which is similar to SSRR.

To the best of our knowledge, SSRR, LSRR and RH are not used nowadays, mainly for security reasons, but some protocols use source routing information for routing IP datagrams. An example is Dynamic Source Routing Protocol (DSR [19]), which uses a new protocol layer and a set of IPv4 addresses for implementing strict source routing in ad-hoc IP networks. To the best of our knowledge, RR is only used by some route tracing tools.

Our location layer has some similarities with the referred routing mechanisms (SSRR/LSRR, HR, RR and DSR) but there are many fundamental differences.

First, the existing source routing mechanisms only work with homogeneous sets of IP addresses, i.e., routes are always lists of IPv4 or IPv6 addresses (not mixed). In our case, locator nodes are free to use other kinds of addresses, as they manage alone one specific portion of SLP or DLP.

Second, the routes used in the existing source routing mechanisms may be created by anyone, there are no protection mechanisms to their authenticity or integrity. In our case, locator nodes are responsible for managing their specific portion of an SLP or DLP, therefore they may introduce arbitrary protection mechanisms, such as ciphers for hiding routing information and integrity control checksums for detecting spoofing attempts.

Third, IP routes used in existing source routing mechanisms may artificially introduce routing overheads, or create DoS scenarios in particular routers or links by directing

overwhelming routing workloads to them. This is possible because anyone can play around with source routes. In our case, this issue can be avoided because locators nodes may protect themselves from DLP forgery or DLP spoofing in locator headers.

Fourth, IP routes used in existing source routing mechanisms may be used to overcome addressing limitations, for instance, to address hosts with private addresses reachable through a host with a public address. Note that this is precisely part of our goal, but we want to do it on a controlled way, with proper routing, access control policies and mechanisms implemented by locator nodes. The current mechanisms and policies, however, do not enable routers to implement complex decision policies for separating legitimate from illegitimate source routes; therefore, for security reasons, packets carrying them are usually discarded. We want to enable such traffic to flow, but with the proper controls implemented by the locator nodes.

Finally, IP routes used in existing source routing mechanisms may be used to explore IP spoofing attacks, because source routing overrules the default IP routing and enables datagrams to bounce and travel along the network according to the wishes of attackers. In our case, however, we are not changing the way IP routing is performed, but we are adding a new routing level, across domains linked by locator nodes. And since locators will mainly be used to establish hierarchical routes across domains, and not alternative routes as in IP, the source spoofing risks raised by source routing in IP do not directly apply to our locator-based routing.

2.6 Routing inefficiency issues

Our location header enables the implementation of multi-hop, overlay routing policies on top of IP. Our goal is not to replace IP routing, but rather to complement it to extend routing across several IP domains (public and private networks) and possibly non-IP networks.

We cannot ensure that one would not create routing inefficiencies by using our location layer. Ultimately, that depends on the exact overlay routing functionality implemented by a set of locators nodes. For instance, multi-hop overlay networks providing end-hosts' anonymity introduce notorious routing inefficiencies (e.g. Tor [8]), but that is a price that one has to pay for getting the desired anonymity; and the same may happen as well in particular exploitations of our location layer.

Source routing is always a more static and restrictive routing paradigm than the one implemented by default in IP networks, where the network has the capability of adjusting its routing to provide the best possible routing service. However, to incorporate richer routing policies we need to explore other routing services, namely the ones implemented on top of our locator nodes, which may be unique and located in specific network locations. Therefore, routing inefficiencies may occur due to bottlenecking situations created by some locator nodes. However, such bottlenecks may as well be solved by a careful provisioning of network and computing resources to highly-demanded locator nodes.

3. ROUTING ACROSS NETWORKS

Let us illustrate the functionality of the location layer with two locators acting somehow as NAT boxes. In Fig. 4 we have a diagram where A initiates a request/response interaction with B. Both A and B belong to separate, private

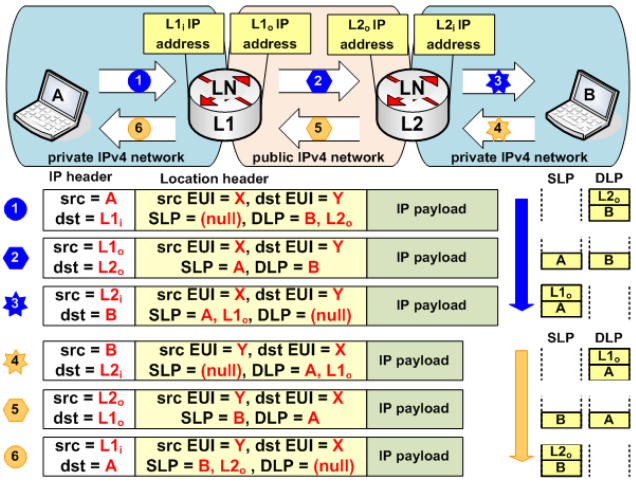

Figure 4: Example of how traffic can be routed from A to B and vice-versa using two locators (L1 and L2) connecting separate private networks to the public Internet. In this example, locators L1 and L2 push and pop IP addresses from SLP and DLP fields of the locations header.

IP networks connected to the publicly-addressable Internet through edge locators (L1 and L2). Both L1 and L2 have public IP addresses and use the existing core IP network to reach each other.

First, A uses some name service to get the identification and location of B: its EUI (Y), the locator node ($L1_i$) and LP_A→B (B, $L2_o$). Given these values, traffic from A to B will first be IP-routed towards L1 and L1 changes the packet as follows: (1) pushes the source IP address into SLP; (2) changes the source IP address to its outer IP address ($L1_o$); and (3) pops the IP address of L2 ($L2_o$) from DLP and uses it to replace the destination IP address.

The packet is then IP-routed to L2, which changes the packet similarly to L1: (1) pushes the source IP address into SLP; (2) changes the source IP address to its inner IP address ($L2_i$); and (3) replaces the destination IP address with the IP address of B popped from DLP.

The packet is then IP-routed to B. When B receives the packet, it realizes that the peer EUI is X (the EUI of A), its locator node is L2 (namely its $L2_i$ address) and L2 knows how to locate the peer using the location path SLP. Consequently, B stores $L2_i$ and the received SLP as the location of the peer endpoint, as well as its EUI X.

When B replies, it will send a packet to X through $L2_i$ and including a location path DLP equal to the received SLP. When L2 receives the packet, it pops $L1_o$ from DLP, pushes the IP of B into SLP, changes source and destination addresses to $L2_o$ to $L1_o$, respectively, and sends it. Similarly, when L1 receives the packet, it pops the IP address of A from DLP, pushes $L2_o$ into SLP, changes source and destination addresses to $L1_i$ to A, respectively, and sends it.

Unlike the current NAT, we do not need to mangle with transport flow selectors (UDP and TCP ports, GRE keys [10], etc.) to multiplex traffic between hosts screened by NAT. Instead, we use the location header to add multiplexing information. Consequently, we are able to multiplex any transport protocol.

Similarly to the current NAT, we change the source IP of the packet that flows from private network to the public one (or to the outer one) and vice-versa. Currently, with NAT, this is not dramatic, except with protocols that do not tolerate such modifications (e.g. FTP, IPSec, H.323, cf. [15, 1]) and require extra actions to be taken by one end-host (e.g. adoption of NAT-T [21, 17] for IPSec or H.460 extensions for H.323 [14]) or by the NAT boxes (e.g. deployment of FTP application-level gateways [29]). But in our case we may follow a more general approach to overcome address modification issues. Namely, such protocols may be updated in order to use the locator header (when present), and peer EUI addresses, to enforce the constancy of the peer identity. This is interesting because end-peers may depend exclusively on their own to keep working currently on a future Internet exploring the location layer.

3.1 Prevention of DLP prediction

One relevant security feature of the current NAT is that only hosts referred in port forwarding tables can be addressed by traffic initiated from outside the private network. And, apparently, this feature is ruined by using location headers and locators to reach hosts inside private networks.

However, the values pushed into SLP by the NAT box, and latter popped from DLP on the return traffic, can be unpredictable for external observers. This way, the NAT box can prevent others from guessing a valid DLP leading to particular host behind the NAT. For example, they can be implemented as random, sparse values on a large addressing space (e.g. 64 bit values), each bound in a table to a particular IP address of an internal host. Alternatively, one can use ciphers and per-locator secret keys. The locator encrypts the (randomly) padded private IP address in order to produce the value to push into SLP; and, inversely, decrypts the value popped from DLP to retrieve the private IP address.

4. CONTENT ADDRESSING

As previously referred, we believe that our location layer can be used to implement some forms of content addressing in the Internet. In this section we will give a useful and realistic example of how it could be implemented. Note, however, that this example does not represent a unique view of how content addressing could be implemented with the location layer.

Nowadays there are many static contents that are fully downloaded by Internet clients (leaf hosts). Examples of such contents are self-contained documents (e.g. PDF), public software packages, software patches, news articles, etc. All these contents may be known by a single, unique number, for instance, a digest of themselves, and could be stored in a Content Addressable Storage (CAS) using the digest as index (as SHA-1 based content-hash keys of Freenet [6]).

Lets assume that we use a very basic application-level content transfer protocol: a TCP-based protocol where the server expects a client connection on a fixed port and sends the complete contents immediately after the connection. The contents to provide are identified using a local, content EUI (which can indirectly identify a file, for instance).

Lets assume that all ISP's providing access to the public Internet collaborate in order to implement a world-wide Distributed Hash Table (DHT), acting as a Content Addressable Network (CAN [27]). ISP's allow local content publishers to register contents in the DHT and allow world-wide content consumers to reach local content publishers using location headers.

Content publisher	DHT entries
$\text{EUI}_\text{X} \rightarrow X$	$h(X) \rightarrow \text{EUI}_\text{X}$, IP address of supplier's ISP, DLP (referring the content publisher)

Figure 5: Values used to reach content X from its publisher using indirect addresses stored in a DHT indexed by $h(X)$

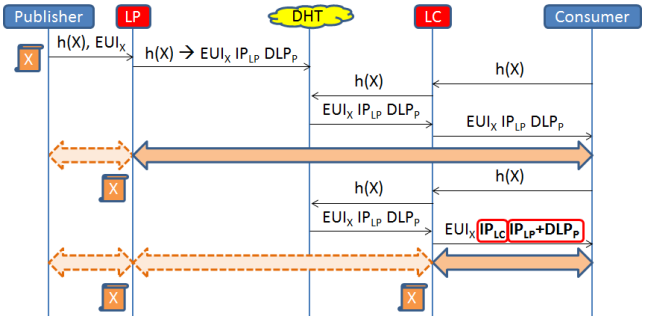

Figure 6: Multi-hop content addressing example. LP is the locator node used by the content publisher to register X in the DHT; LC is the locator node closer to the content consumer. Thin arrows represent procedure calls; thick arrows represent single-hop content transfers. On the upper transfer, X is fetched from the supplier or from LP's cache; on the bottom transfer, X can be fetched either from the publisher or from LP or LC caches.

The entries of the DHT could be identified by content hashes (e.g. SHA-1 digests), and each entry would contain an indirect identifier: content EUI, IP address of a locator node and a DLP (see Figure 5). Locators nodes would be ISP's, and DLP could contain the following stacking of data (outermost to innermost): internal ISP tag for content addressing and the local IP address of the content publisher.

Given this infrastructure, content publishing and fetching would take place as follows (see Figure 6). Content X publisher generates an EUI for it (EUI_X) and stores a mapping between them. Then, it registers the tuple $\langle h(X), \text{EUI}_\text{X} \rangle$ in the content addressing service of its ISP (ISP_P), which stores it in the DHT, together with its own IP address and a DLP that enables future clients to reach the publisher through ISP_P. The DLP would contain a content addressing tag and the IP address of the content publisher.

For content fetching, the client must first get $h(X)$ in some Internet service (e.g. a new kind of URI in Web pages, such as a Named Information URI [11]). Then, it contacts the local ISP (ISP_C) for resolving $h(X)$ into some indirect address. ISP_C uses the DHT to fetch the related indirect address: EUI_X, IP of the locator node (ISP_P) and DLP for reaching the content publisher from the locator node. Once having the indirect address, the consumer can use it to contact the content publisher, together with basic transport protocol above referred, in order to get the intended content.

This basic content access methodology can be enriched with other features, such as caching in ISP's for increasing locality of reference. In this case, ISP's implementing caches could resolve $h(X)$ to a slightly different indirect address, using their own IP as locator host and complementing the DLP with the IP address of the original locator. In this case, for fetching the content the client would contact its own ISP, which could use a local cache or fetch the contents from the original location (expressed in the DLP of the indirect address used by the client). For the client, however, everything is transparent, it always gets the required content.

We could also cache content ahead of content publishers, in their own ISP's, for further increasing the performance of content downloads. In fact, ISP's act as locator nodes, and thus have the opportunity to cache contents on their first download from local content publishers. Again, this would be totally transparent to clients and would not interfere in any way with the previously described caching strategy.

This example cannot be seen as a final and totally specified proposal for implementing a context-addressable infrastructure for the Internet. Namely, in this description we did not address security issues, such as malicious manipulations of the content of the DHT. Our unique goal was to provide a simple example of how our location layer and our indirect addresses could be used to address content, instead of hosts, with transparent caching for improving download efficiency (one publisher, many possible suppliers).

5. PROTOTYPE IMPLEMENTATION

For testing our location layer we changed the TCP/IP protocol stack of a Linux and we created a new NAT functionality, exploring location headers and using a userland packet processing application invoked from the kernel using the IPTABLES queueing functionality.

5.1 Location header structure

The location header used in our implementation has the following structure:

protocol: next level protocol (1 byte) (ICMP, UDP, TCP, etc.), just like in IP. For the location layer we used the number 200.

length: header length (2 bytes), the total length of the following data field. The first byte provides the size of the source-related data (source EUI and SLP); the second byte provides the same information about destination EUI and DLP. Both bytes have the same structure: the most significant bit indicates the presence of the EUI; the remaining 7 bits give the size of the LP (SLP or DLP), allowing at most 127 bytes for each LP. The maximum length of a location layer is $1 + 2 + 16 + 16 + 127 + 127$, thus 289 bytes.

data: variable-size payload field containing source and destination EUIs, SLP and DLP (by this order).

5.2 NAT scenario

Since we do not have yet name servers providing our indirect addresses, and existing applications cannot explore them directly, we cannot implement exactly the NAT functionality described in Section 3 and illustrated by Fig. 4. Therefore, we decided to implement an alternative NAT approach that mixes the existing behaviour (source IP screening) with the exploitation of the new location header (traffic in the public network uses the location header).

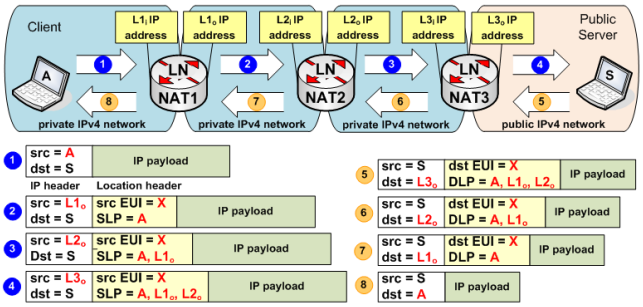

Figure 7: Tested scenario, using client A with the current TCP/IP stack and server S with a modified stack. Traffic is routed from A to B and vice-versa using NAT boxes that act as locators, connecting separate private networks or a private network to the public Internet. In this scenario, the locators push IP addresses into SLP in outbound traffic and pop IP addresses from DLP of inbound traffic. The innermost NAT, NAT1, has the additional task of adding a complete location header to outbound traffic and removing it from inbound traffic.

In the scenario we deployed, illustrated by Fig. 7, there is a host with the current TCP/IP stack in a private network (host A, client) and a host with a modified stack in a public network (host S, server). To interconnect both networks we used three of our novel NAT boxes, all of them implemented on top of the current TCP/IP stack.

The server's modified TCP/IP stack recognizes and uses EUIs when matching transport packets to communication endpoints (UDP or TCP sockets), thus fully complying with our proposal. However, by default it does not add a location header to all outbound traffic; it only adds that header when it was first used by a client. Thus, if a socket refers a peer using an indirect identifier (EUI, IP address of locator and DLP), the location header is used in all outbound traffic to that peer. Otherwise, the location header is not used.

The server's modified TCP/IP stack also disregards checksums on UDP and TCP packets that arrive with a location header. This checksum for outbound packets is also left zeroed, as we can see in Fig. 12.

In this scenario we never use an EUI for the host S because it was not necessary; we can reach it from A using only IP addresses and the ordinary IP routing.

5.3 NAT functionality

All the outbound traffic generated by the NAT boxes has a location header. This means that the innermost NAT must generate a fake EUI for outbound traffic of host A. It does so in such a way that a) the same host gets the same EUI and b) there is a minimum state kept in the NAT to maintain the coherence of fake EUIs. Namely, the EUI is computed by hashing the source IP address, the NAT outer IP address and an internal key.

Our NAT box implementation must deal with traffic generated from current TCP/IP stacks (e.g. from host A) and from hosts that are using the location header (e.g. outbound traffic that reaches NAT2 and NAT3 coming from NAT1). The NAT box supports both types of clients very easily (cf. pseudo-code in Fig. 8). If an outbound datagram includes

Listing 1: Outbound NAT processing

```
if (pkt->hasLocationHeader==FALSE) {
    srcEUI=hash128(pkt->srcIP,myPublicIP,key);
    dstEUI=0;
    SLP=DLP=0;
    pkt->addLocHdr(srcEUI, dstEUI, SLP, DLP);
}
pkt->pushIntoSLP ( pkt->srcIP );
pkt->srcIP=myPublicIP;
```

Listing 2: Inbound NAT processing

```
dstIP=pkt->popFromDLP();
if (pkt->dstEUI==hash128(dstIP,pkt->dstIP,key)) {
    pkt->remLocHdr();
    pkt->computeLegacyTransportChecksums ();
}
pkt->dstIP=dstIP;
```

Figure 8: C-like pseudo-code of the datagram processing executed by the NAT box and involving the location header and the IP header.

```
-i iIF -t mangle -A PREROUTING -j MARK --set-mark 1
-i oIF -t mangle -A PREROUTING -j MARK --set-mark 2
-A FORWARD -j QUEUE
-A INPUT -p 200 -j QUEUE
```

Figure 9: iptables rules used to send IP packets to the userland application that performs NAT using the location layer (protocol number 200). The names `iIF` and `oIF` represent the inner and outer interfaces, respectively. Marks 1 identifies outbound packets, which may require a totally new location header; mark 2 identifies inbound packets, which must have a location header in order to be routed to an inner network. The userland application should process all (outbound) forwarded packets and all (inbound) packets targeted to itself and possessing a location header.

a location layer, then the NAT only updates the SLP field. Otherwise, it adds a new location header, computes a fake EUI of the source, and updates the SLP field. For inbound traffic the distinction is slightly more complex: if the destination EUI was generated by the NAT, the location layer is removed before routing the datagram to the destination; otherwise, the header remains in the datagram, only its DLP field is updated.

In any case, the NAT box uses the IP address of the screened host (belonging to the private network) when pushing information for the SLP in outbound traffic and popping information from the DLP in inbound traffic.

The NAT box is completely stateless, as all packet transformations are performed using data from the packets themselves or local, constant configuration parameters. Routed packets are processed in the PREROUTING chain to add IPTABLES' packet marks, which will be used by the userland application to distinguishes inbound from outbound packets (see Fig. 9). Outbound packets will be queued for that application using a rule in the FORWARD chain (as they are targeted to some other host); inbound packets will be queued in the INPUT chain (as they are targeted to the current host). As it happens with current NAT, our NAT model cannot handle IP fragments; both cases are handled similarly using the IPTABLES' defragmentation module (`nf_defrag_ipv4`).

Comparing this NAT paradigm with the current one, both solutions update source or destination IP addresses in IP headers. On the other hand, current NAT changes fields on transport headers, while we change fields of the location header. The current NAT must keep a table of port mappings performed to multiplex traffic and must manage lifetimes of those mappings, while we do not need to keep and manage any translation state. Finally, the current NAT must recompute the checksums of all inbound and outbound UDP and TCP packets, while we only have to recompute the checksum of inbound UDP and TCP that have no location header (hosts that know how to handle the location layer, such as S, ignore checksums on inbound packets).

5.4 Testing and performance evaluation

We tested several types of client-server communication protocols, including Ping, Telnet, OpenVPN and HTTP, and all worked without problems. Appendix A shows some packet captures that illustrate the behavior of our NAT.

We did not test, but the server host could also initiate communications towards the client host, since it could generate all the location header parameters that could lead to a successful inbound NAT box traversal. Note that with this approach to NAT, ISPs could use only one public IPv4 address to provide unconstrained, bidirectional access between hosts using private and public addresses.

Using the timestamps of Wireshark captures we can have a notion of the latency overheads imposed by each of our NAT boxes. For reducing overheads, we shift the IP header instead of the IP payload when adding or removing the location header. Nevertheless, we could not avoid the memory copy costs imposed by the packet transfers between kernel and the userland NAT process. To remove this cost we need to implement the NAT functionality in the kernel, as it happens today with the current NAT technology.

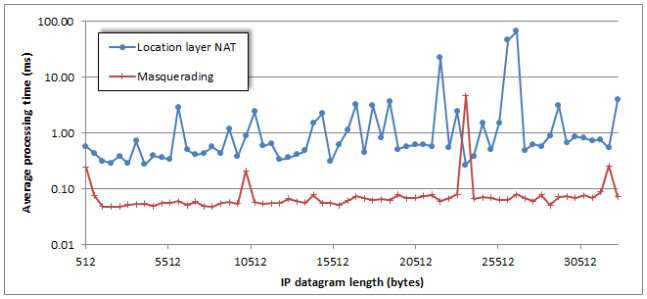

Figure 10: Overhead of NAT operation for outbound traffic in NAT1, both for the current NAT (masquerading) and for NAT with the location layer, as a function of the original IP datagram length.

Figure 10 shows a graphic where we display the average latency overhead of our NAT for outbound packets as a function of the original IP datagram length. The traffic samples for the graphic were collected in the two interfaces of NAT1. The sampled traffic was generated with ICMP pings with crescent payloads (total IP lengths from 512 bytes to 32 KiB); for each datagram length we generated 10 pings. The figure also shows the latency of the actual NAT (masquerading), just to have a reference for comparison.

As we can observe in the collected values, both overheads increase with the length of the IP payload, but this fact is

more notorious with our NAT, as we copy packets from kernel memory to user memory, and vice-versa, for adding a location header. Furthermore, those memory transfers and the process switching involved in our NAT are responsible for an overhead that is less stable and about an order of magnitude higher than masquerading. However, we are comparing an optimized, in-kernel NAT implementation with a prototype implemented in a user process. We believe that a future in-kernel implementation may be competitive, in terms of performance, with masquerading.

6. RELATED WORK

Readers with good networking background may be led to compare our work with tunneling or adaptation technologies developed to overcome locality problems or coexistence issues raised by incompatible IP address spaces. Examples of such technologies are GRE encapsulation [10] used in PPTP [16], that enables PPP connections to be established over the Internet, or the 6to4 transition mechanism [4], that enables IPv6 domains to interact through an IPv4 cloud. But our contribution is quite different, as we do not provide tunnels, we provide overlay multi-hop routing.

Our contribution is comparable to three types of related works: overlay networks, data-oriented routing and proposals for separating host identity from location in the Internet.

Overlay networks are flexible and variable sets of Internet hosts that arrange themselves to provide a particular service to clients, usually data storage and retrieval using Peer-to-Peer (P2P) Distributed Hash Tables [24]. To fulfil this goal, overlay networks use some particular applicational protocol to transfer commands and data, implementing some kind of specific addressing and routing protocol. Therefore, our work can be used to implement many different routing strategies directly over IP within overlay networks.

In content-oriented routing [2], such as in DONA [22] and CCN [18], communications are routed in order to link a content consumer to the closest copy of some content, instead of a specific providing host. DONA uses the route-by-name paradigm of TRIAD [13] and a flat name space to dynamically connect content consumers to content providers. CCN relies on the broadcast of data interest packets to reach any satisfying content store, which can send the same content to many consumers using multicast. CCN uses a hierarchical name space and intermediate network elements can cache contents to optimize future transfers of the same content. We believe that both functionalities can be achieved using our locators and by embedding names in DLP, thus keeping the basic TCP/IP stack and the IP core routing, while CCN proposes a completely new protocol stack.

Regarding the separation of identity from location in the Internet, we can find many common ideas in the work of Balakrishnan et al. [3], which also borrowed ideas from Nimrod [5], HIP [25] and I3 [30] that are related with our work. Balakrishnan et al. proposed a new layered naming architecture for the Internet. In their architecture, there is a new layer (End Point Identifier Resolution), between transport and network layers, that is responsible for resolving an EID into an IP address. Comparing with our proposal, an EID can be compared to our indirect identifier (EUI, locator IP and location path), but they are completely different: an EID is resolved by the source into an IP address using name resolvers, while our indirect identifiers already contain an IP address (of the locator) and are iteratively resolved by one or more locators, and not necessarily into IPv4 addresses. This enables us to expand object addressing within the internet beyond the closed addressing space of IPv4 public addresses. Furthermore, we combine name resolution (or addressing) with routing, while they use independent name servers for name resolution and the standard IP routing for packet routing. The EID delegation mechanism, allowing a host to resolve an EID to an IP address that will then forward the packet to another IP address, is somehow similar to the service implemented by our locators. However, its not clear how they route traffic back towards an initiator that reaches a target host using many EID delegations. Furthermore, it is not clear how they can implement delegation with stateless servers.

The location path of our indirect identifiers is similar to Nimrod's Connectivity Specification Chains (CSC) or Sequences (CSS) of locators, which are Nimrod's host identifiers [5]. Both CSC and CSS can be used to implement packet forwarding according to paths completely controlled by users or devices acting on their behalf. But we go one step further, since the stacking strategy used by our locator nodes for managing location paths' elements allows them to manage private forwarding rules and data, possibly transparently to endpoints.

HIP [25] uses an Host Identity Tag (HIT) to identify each host, and an HIT is a cryptographic hash of an Host Identifier, which is the public key of an asymmetric key pair. HITs are resolved to IP addresses when used to address a destination host, this way decoupling host identification from host addressing. Our approach is different, host identification is given by the EUI, which can have some meaning (as an HIT) or may be fully random, and EUIs may be used or not by locators to find or memorize routes, but endpoints do not do so. We use an EUI mainly for session identification at end-hosts, and location paths to reach targets, while HIP uses the HIT for find the current IP address of a target host. Therefore, HIP cannot be used to address Internet endpoints other than hosts with IP addresses.

I3 [30] uses rendez-vous points for decoupling the act of sending data from the act of receiving data. Rendez-vous points use triggers inserted by receivers to forward senders' packets; triggers contain an identifier and the destination IP address. Our locators can be used to implement I3 rendez-vous points, enabling the implementation of specific routing policies, in this case the ones designed for I3 (mobility [32], multicast [23], etc.). On the other hand, I3 naturally assumes that rendez-vous points are stateful, while we allow our locators to perform stateless routing decisions based on received location paths. This way, we provide a more flexible platform to implement novel addressing and routing strategies than I3.

7. DEPLOYMENT EVALUATION

Our location layer was not invented exclusively for tackling IPv4 scarcity issues, but rather to allow the Internet to evolve in a hierarchical fashion, instead of a large, flat addressing space (e.g. using IPv6). NAT did something similar, and was widely accepted and adopted because it could be used with some limitations without changing existing TCP/IP stacks. We cannot do the same with our NAT mechanisms based on the location layer, because they require TCP/IP stacks of endpoints to recognize location layer headers. Nevertheless, after an update phase of those

stacks we could replace existing NAT boxes by ours with clear benefits, such as unrestricted bidirectional access. This is possible with a gradual, phased migration strategy: first, TCP/IP stacks are updated, mainly through automatic updates; then, after a reasonable update period, network services exploring the location layer could start to be used.

Current applications exploring the Internet for remote interactions need to manipulate information of both network and transport layers, i.e. IP addresses and TCP/UDP communication endpoints (sockets). Therefore, any migration among IP addressing paradigms, namely from IPv4 to IPv6, has a tremendous impact on applications, because they are only prepared to handle IPv4 addresses, and not IPv6 addresses.

On the contrary, the location layer is not mandatory, it is an option layer between the existing IP and TCP/UDP layers. This means that applications that are prepared to deal with IP and TCP/UDP data structures continue to observe the exact same data structures, because they do not disappear or are modified. Nevertheless, future versions of those applications may include extra functionalities for observing and exploiting location headers, in order to provide enhanced features related with the location layer.

With our prototype we demonstrated that an existing server using a modified TCP/IP stack can interact with a client that is using the location layer as long as the server's stack recognizes the existence of the location layer and uses it correctly when sending traffic to the client. The server application doesn't need to handle the location header, just like it doesn't need to handle many operational options of TCP/IP; the kernel, by itself, is able to automatically handle location headers with a few default policies.

Concluding, the location layer can be introduced gradually, without dramatic migration steps. First, operating systems may be updated to include support for handling location headers, just like we did. Simultaneously, name servers, such as DNS or others, may be updated or created from scratch to provide the resolution of names into indirect addresses. Once this done, existing applications may be updated or new applications may appear in order to provide new functionalities exploring the location layer, such as our content-addressing system described in Section 4. Last, but not least, the Internet core routers are not affected by the use of the location layer, and don't need to be updated.

8. CONCLUSIONS

We presented a new location layer for the TCP/IP stack, inserted between IP and transport protocols. This layer is optional and enhances the addressing capabilities of the Internet, by allowing locator nodes with specific addressing and routing tasks to reach target endpoints. Target endpoints may be hosts or other entities, such as contents. It also allows the Internet to survive to the depletion of the IPv4 address space, without requiring modifications on the core routing infrastructure, or a worldwide migration to IPv6 or a massive mitigation with the current NAT model. As we have shown with our prototype, the TCP/IP stack of hosts on the edge of the Internet can be progressively updated until reaching a state where the location layer could be explored ubiquitously.

Among the many possible research lines using the location layer, we anticipate new name services, mobility, multihoming, multicast and content-addressing.

9. REFERENCES

[1] B. Aboba and W. Dixon. IPsec-Network Address Translation (NAT) Compatibility Requirements. RFC 3715, Mar. 2004.

[2] M. Alduán, F. Álvarez, T. Zahariadis, N. Nikolakis, F. Chatzipapadopoulos, D. Jiménez, and J. M. Menéndez. Architectures for Future Media Internet. In *2nd Int. Conf. on User Centric Media*, Palma de Mallorca, Spain, Sept. 2010.

[3] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A Layered Naming Architecture for the Internet. In *ACM SIGCOMM 2004*, Portland, OR, USA, Aug. 2004.

[4] B. Carpenter and K. Moore. Connection of IPv6 Domains via IPv4 Clouds. RFC 3056, Feb. 2001.

[5] I. Castineyra, N. Chiappa, and M. Steenstrup. The Nimrod Routing Architecture. RFC 1992, Aug. 1996.

[6] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Int. Works. on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, Berkeley, California, USA, 2001.

[7] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Aug. 2008.

[8] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *Proc. of the 13th Conf. on USENIX Security Symposium*, San Diego, CA, USA, Aug. 2004.

[9] K. Egevang and P. Francis. The IP Network Address Translator (NAT). RFC 1631, May 1994.

[10] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Generic Routing Encapsulation (GRE). RFC 2784, Mar. 2000.

[11] S. Farrell, D. Kutscher, C. Dannewitz, B. Ohlman, and P. Hallam-Baker. The Named Information (ni) URI Scheme: Core Syntax. Internet-Draft, Oct. 2011. draft-farrell-decade-ni-00, expires in April 26, 2012.

[12] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827, May 2000.

[13] M. Gritter and D. R. Cheriton. An Architecture for Content Routing Support in the Internet. In *3rd USENIX Symp. on Internet Tech. and Systems*, San Francisco, CA, USA, 2001.

[14] H. S. Group. Packet-based Multimedia Communications Systems. ITU-T Recommendation H.323 (Revised Version 7), 2009.

[15] T. Hain. Architectural Implications of NAT. RFC 2993, Nov. 2000.

[16] K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, and G. Zorn. Point-to-Point Tunneling Protocol (PPTP). RFC 2637, July 1999.

[17] A. Huttunen, B. Swander, V. Volpe, L. DiBurro, and M. Stenberg. UDP Encapsulation of IPsec ESP Packets. RFC 3948, Jan. 2005.

[18] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard. Networking Named Content. In *ACM CoNEXT*, Rome, Italy, Dec. 2009.

[19] D. Johnson, Y. Hu, and D. Maltz. The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4. RFC 4728, Feb. 2007.

[20] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301, Dec. 2005.

[21] T. Kivinen, B. Swander, A. Huttunen, and V. Volpe. Negotiation of NAT-Traversal in the IKE. RFC 3947, Jan. 2005.

[22] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A Data-Oriented (and Beyond) Network Architecture. In *ACM SIGCOMM 2007*, Kyoto, Japan, 2007.

[23] K. Lakshminarayanan, A. Rao, I. Stoica, and S. Shenker. End-host Controlled Multicast Routing. *Elsevier Computer Networks, Special Issue on Overlay Distribution Structures and their Applications*, 2005.

[24] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. *IEEE Comm. Surveys and Tutorials*, 7(2), 2005.

[25] R. Moskowitz and P. Nikander. Host Identity Protocol (HIP) Architecture. RFC 4423, May 2006.

[26] J. Postel. Internet Protocol. RFC 791, Sept. 1981.

[27] S. Ratnasamy, P. Francis, S. Shenker, R. Karp, and M. Handley. A scalable content-addressable network. In *Proc. of ACM SIGCOMM*, San Diego, California, USA, Aug. 2001.

[28] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. RFC 1918, Feb. 1996.

[29] P. Srisuresh and M. Holdrege. IP Network Address Translator (NAT) Terminology and Considerations. RFC 2663, Aug. 1999.

[30] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *ACM SIGCOMM 2002*, Pittsburgh, PA, USA, Aug. 2002.

[31] Z. Wang and J. Crowcroft. A Two-Tier Address Structure for the Internet: A Solution to the Problem of Address Space Exhaustion. RFC 1335, May 1992.

[32] S. Q. Zhuang, K. Lai, I. Stoica, R. H. Katz, and S. Shenker. Host Mobility using an Internet Indirection Infrastructure. In *1st Int. Conf. on Mobile Systems, Applications, and Services (ACM/USENIX Mobisys 2003)*, San Francisco, CA, USA, May 2003.

APPENDIX

A. PACKET CAPTURES

Figures 11 and 12 show packet captures with a modified Wireshark that is capable of showing the contents of the location header as implemented by our NAT mechanism.

Figure 11 shows a Ping dialog between client A and server S, captured in the interfaces of both hosts. The expanded datagram is an inbound Echo Response, showing a destination EUI of A (faked by N1) and a DLP that enables the datagram to reach A through N1, N2 and N3.

Figure 12 shows the setup of a TCP/IP connection between A and S, captured in the interface of S. The expanded segment, a SYN/ACK sent by S, uses a location header that happens to be equal to the one used for the Ping. This happens because the policy to fake the EUI is the same for all protocols, and the path along NAT boxes is also the same.

In both expanded location headers, S replicates the received Source EUI as Destination EUI, the received Source Location Path (created by N1, and N2 and N3) as Destination Location Path, and the received source IP address (`192.168.1.3`, $L3_o$ address of Fig. 7) as destination IP address. IP addresses `192.168.1.1` and `192.168.1.2` in the Destination Location Path represent addresses $L1_o$ and $L2_o$ of Fig. 7. Client A has a faked EUI ($\texttt{661CA75D80E0B447}_H$) created by NAT1; the server S does not have an EUI.

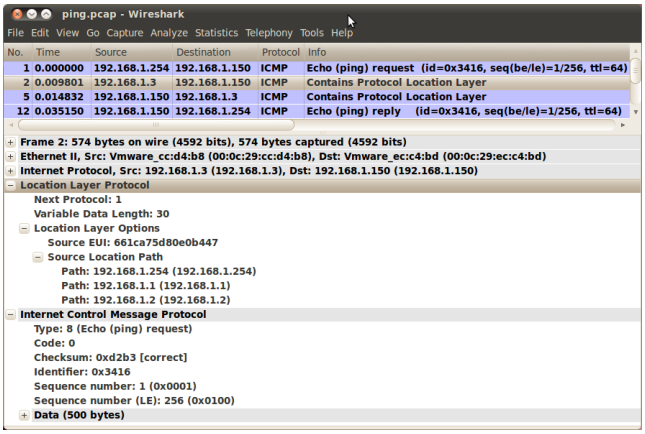

Figure 11: Traffic captures of a single Ping dialog between client A and server S on their interfaces (IP addresses `192.168.1.254` and `192.168.1.150`, respectively) using the network architecture of Fig. 7. The expanded Echo Reply, captured on the interface of S, shows the structure of the location header added by S. All the traffic created and received by client A does not have location headers.

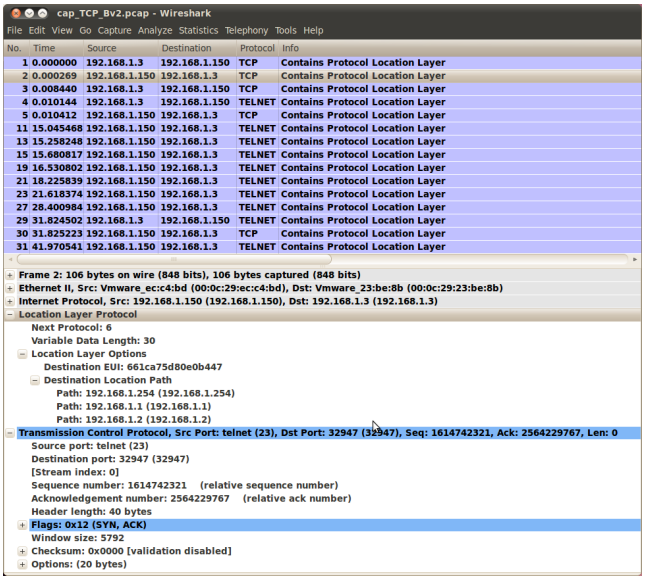

Figure 12: Traffic captures of a Telnet connection between client A and server S on the interface of the latter (IP address `192.168.1.150`). The expanded SYN/ACK segment shows the structure of the location header added by S.