SelNet: A Translating Underlay Network

Christian Tschudin[*]
Department of Computer Systems
Uppsala University, Box 325
S-75105 Uppsala, Sweden
tschudin@docs.uu.se

Richard Gold[†]
FhG FOKUS
Kaiserin-Augusta-Allee 31
D-10589 Berlin, Germany
gold@fokus.fhg.de

November 2001

Abstract

The Internet has successfully promoted address uniformity and a node centric forwarding semantics. However, NAT and wireless networks among others have shown the advantage and the need of revising basic assumptions of the Internet model. In this paper we review several of these basic networking concepts and introduce a new set of network abstractions like ”membranes” which are individual physical or virtual networks and ”wormholes” which link one or more membranes together. This leads us to an active network architecture called SelNet that is based on tunnelling and translation mechanisms. Besides the architecture we present several network services and abstractions that can be built on top of it. A brief status report on a prototype implementation is also provided.

Keywords: Network Architecture, Function Selectors, Translation & Resolution, Ad-Hoc Routing, Highway Routing

1 Introduction

Address uniformity and stateless packet switching have heavily contributed to the success of the Internet. Any node can become a fully qualified Internet citizen once it has obtained an IP address; remarkable robustness has been achieved by relying on as little state as possible inside the network. However, this essentially flat network architecture has been the subject of modifications both “behind the scenes” and “openly”. NAT, for example, violates the address uniformity by inserting a middlebox that translates IP addresses forth and back between two separate address spaces, blocking some addresses in case no mapping is available. Blocking, in the context of packet filters, is a desirable security property, but various tunnelling techniques have been devised that lift IP forwarding to the level of transport protocols that by themselves are already running over IP. Mobile IP, Virtual Private Networks and other overlays use the same encapsulation technique, although their goal is to gain control over IP’s routing functionality at the price of more state in the network. Finally we also mention wireless ad-hoc routing

[*]This work was done at the University of Basel, Switzerland. We gratefully acknowledge the support from the University Computing Center (URZ).

[†]Richard Gold was supported by the European Union COST 263 Action on *Quality of future Internet Services*.

protocols for which existing IP routing protocols do not work well and where proposed replacement protocols essentially create stateful path maintenance environments. In short: What we see is a need for multiple address spaces and, related to address mapping and route maintenance, the requirement to put state at various places inside the network. Our aim is to come up with a network architecture where translation and tunnelling naturally fit in the network's mode of operation. The path we are taking is that of separating addresses from packet processing semantics and providing a system where this binding is established and changed at run time. This leads to a deconstruction of current networking architectures and enables the recombination of networking functionality in new and more flexible ways.

In this paper we present SelNet. SelNet contains very few networking primitives: Node addresses, for example, do not exist in this model, nor do the concept of subnets or overlays. Instead, SelNet is a frugal underlay that acts as a shim network for different network personalities, including IPv4, VPNs or content addressable networks. In this paper we will introduce the SelNet architecture and show re-implementations of basic network activities in our SelNet prototype. As an example we show how ad-hoc routing can be easily derived from an address resolution protocol. Also, we introduce a new routing abstraction similar to add/drop multiplexing that is neither end node addressing nor source routing. Before explaining in Section 2 the rationale for SelNet we briefly review previous work that closely relates to our approach.

1.1 Related Work

We classify relevant related work into the following categories: Shim networks, resolution systems, translation systems and overlay networks. MPLS [1] is an example of a shim network, it uses protocol labels to allow fast switching in routers. By allowing the routers to make switching decisions based upon simple labels they can avoid the overhead of an IP lookup. We use a similar concept to MPLS labels called *selectors* in SelNet to allow fast switching and lightweight addressing at the lower layers of the protocol stack. Our selectors, however, label more generic elements than the path-only approach taken by MPLS. Two integral parts of our architecture are resolution and translation mechanisms. Such mechanisms have also been investigated by the IPNL [3] and TRIAD [2] systems. In IPNL (IP Next Level), the authors propose a new Internet architecture based upon an extended version of NAT. They used Fully Qualified Domain Names as global end-to-end addresses and then use IPNL addresses to perform dynamic translation between different domains. Instead of tightly integrating our system with IP in the way that IPNL functions, we place our translation mechanisms below IP in order to avoid dealing with the complexity of IP operations. The TRIAD system is a resolution system which provides content-to-name or content-to-number resolution. They define an explicit content layer above IP and use content aware routers and DNS servers to introduce their new resolution functionality into the network. The Resilient Overlay Networks (RON) project [4] deals with creating overlay networks to allow collections of RON nodes to deal with routing failure or network congestion by letting traffic be dynamically re-routed through other RON nodes. Link failures, for example, are less likely to be a problem as a RON node will have multiple routes to a single destination which can be chosen at will if a certain route is not performing optimally. We take this further with SelNet as we also provide choice but with a wider variety of networking functionality to select from than just different forwarding paths.

The paper is structured in the following way: Section 2 provides a discussion on the notions of addresses and selectors. In Section 3 we describe the architecture of our system and in Section 4 we detail a proof-of-concept implementation. We discuss a new networking abstraction named “Highway Routing” in Section 5 and we close the paper with the concluding Section 6.

2 Addresses, Paths and Underlays

The goal of this section is to provide a background discussion on addresses and forwarding paths. These two concepts are the defining elements of any packet switched network architecture, to which SelNet also belongs. Readers interested in the hard implementation bits can safely skip this section and come back to it later on.

2.1 Addresses are Function Selectors

The role of a packet's destination address is to identify the forwarding function that will carry it closer to its target node. This definition of addresses as being function selectors suits classical networking as well as active networking. In classical networking, the destination is regarded as a parameter to the real forwarding function – parameter passing just factors out the potentially huge number of forwarding functions that a network might need. In active networking, addresses can take the form of actual functions (programs) that contain explicit forwarding instructions.

In SelNet we generalize this even further in order to unlock the full range of functionality that could be performed in the network. Instead of saying that packets can be sent (addressed) only to functions, we apply the function selection view to the whole packet, not just the destination address: it is up to the called function to take care of the packet, packet forwarding being just one particular useful case out of many others. Functions can change packet headers (header rewriting, compression), work on the packet's data (proxies), or react based on the packet's data as a server and send back a completely new reply packet.

2.2 Local addresses

In SelNet, packet processing functions are addressed by a name which we will call a selector. In order to avoid spanning the name space for all potential functions (and having to enumerate them), we impose that function selectors are purely local names. Only those functions can be addressed that actually are resident on a specific host. Without limiting the concept of addressing potentially any function that might exist, we can restrict the number of bits that we have to set aside for selectors simply because each host only has a finite number of function implementations available. When it comes to actual packets sent over the wire, it suffices for us to know that the packet carries a selector value in a fixed size header field and a payload: the selector will be used at the receiving host to find the function that should be applied to the packet. This is the approach taken in the SAPF architecture [5] which is what we use for the implementation of SelNet.

The local naming scheme means that selectors have no global meaning and that, for example, packets being forwarded from one SelNet node to the other have to change their selector value on each hop. Selector (or header) translation thus is an intrinsic property of SelNet. From a sophistic point of view this property already exists in other packet switched networks like the Internet where the Ethernet header changes on each leg of the IP delivery path, as well as the IP header changes because of the TTL field, which in fact selects a different function depending on it's value, namely forward or drop.

2.3 Name resolution

Due to the fact that local function names are the only addressable items in SelNet, we need a way of mapping other kinds of identifiers like end-to-end destination addresses or server names to local selector values. To this end we provide a generic resolution function for which we assign a global, well–defined selector value. Using this value, an application can request address resolution and get a local function name that ”stands for” the item to resolve. For example, a request packet for resolving a neighbour’s IPv4 name to the Ethernet address (à la ARP) would result in a local delivery function being instantiated on the fly which takes care of delivering any packets sent to it to this neighbour. In fact, such state creation on the fly occurs already today inside protocol stacks where an ARP cache keeps that mapping in memory. The selector view makes this state explicit and presents the associated forwarding function in the form of a selector. We note here the difference between resolution and translation. First, the IPv4 name has to be resolved. Subsequent packets however will be subject to translation (which is much cheaper), essentially rewriting the local delivery selector into the associated Ethernet address details.

Extending this view and coming back to the note on the forwarding functions, we map routing to selectors too. Finding the next hop is in fact a resolution request. IPv4 does this resolution on a per packet basis. In SelNet it is also possible to do the resolution once and from that point on use translation. This means that resolving an IPv4 name would return a selector to which any packet can be sent that has to be sent to the given node: downstream nodes would cache the resolved path and make it accessible via forwarding functions that will be chained through the selector rewriting mechanism. In fact, this results in a tunnel where the sender only sees the entry point of the tunnel in the form of a selector value.

2.4 Underlay Networks

We have described our view of addresses as function selectors: In the SelNet architecture, we generalize this concept so that we no longer address end nodes, just functions residing on nodes. This allows us to have not just forwarding functions inside the network, but also header rewriting and payload manipulation since the functions that we can select are not pre-defined.

We position SelNet’s function demultiplexing functionality as an underlay network. The aim is not to refine or extend an existing overlay network like IP [6, 4] in order to form new overlays, but to provide minimalistic abstractions which us to recombine parts or all of existing protocol stacks in arbitrary ways. IP stacks, for example, are intercepted at the subnet level including all address resolution processes going on. With SelNet, we can then handle requests in self-chosen ways, as well as tunnelling low-level frames at higher protocol stack levels if desired. With overlay you would be restricted to a specialization of the existing functionality provide by the substrate. In order to achieve a comprehensive emulation of the upper layers of the protocol stack, all exit points from the protocol stack need to be covered which is only possible with an underlay.

3 SelNet: Wormholes & Membranes

In this section we introduce our basic networking abstractions that we use as building blocks for the rest of our system. Our three networking abstractions are:

Membranes : We define membranes as compartments that represent individual physical or logical networks e.g. LANs, VPNs or virtual topologies of end nodes. Hosts themselves are also membranes - they are the place where wormholes reside.

Wormholes : Wormholes are transition/translation points between membranes. We imagine addressable wormholes located inside membranes representing tunnel entries to neighbour membranes. Nodes, default gateways and remote locations reachable over multiple transit membranes are then mapped to wormhole entries.

Wormhole Glue : This abstraction allow us to splice together wormholes so that we can conflate multiple forwarding paths into one tunnel. We can also use the wormhole glue to insert other kinds of packet mangling functionality into the network. For example, header compression and other header-rewriting activities, drop points (see section 3.4), data transformations such as proxies, protocol boosters etc.

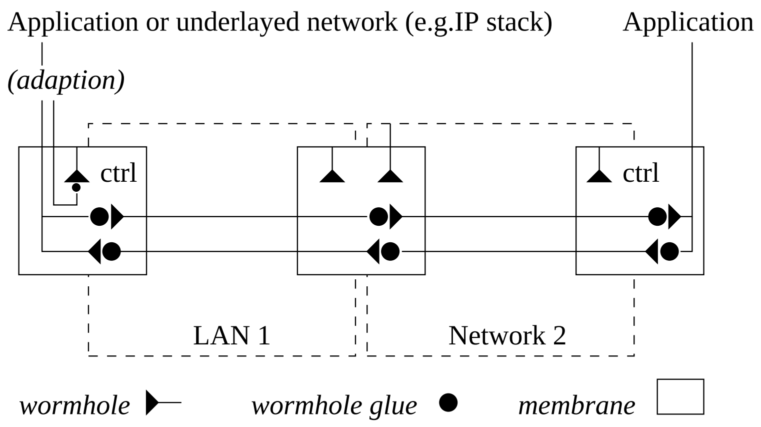

Figure 1: SelNet network architecture

3.1 Membranes

Membranes typically are a collection of network nodes which share a common addressing and name resolution scheme and which enable transparent datagram exchange between its members. On each node of a membrane there is a well defined control wormhole through which we can query and steer membrane functionality. Typically a membrane could be an Ethernet segment, but the classical IPv4 Internet is also by itself a membrane as would be an IPv6 cloud. A membrane may correspond sometimes to existing layer 2 and layer 3 topologies, the more interesting aspect of this work is that it can just as easily represent completely virtual topologies.

3.2 Wormholes

Wormholes are locally addressable simplex datagram tunnel entries. Wormholes point in a transparent way to some data sink which can be a local service (e.g., the control wormhole of the current membrane), a remote service or some remote glue that will pass datagrams over to some other wormhole. Whilst the membranes create their own view of the world, the wormholes allow the membranes to make their forwarding services available to the rest of the world.

A wormhole is a basic abstraction that can represent many different entities in the network. The abstraction comes more into its own when a wormhole is used to represent a host in another membrane that may have a complete different addressing scheme, that is, when a host creates a "remote presence" in the form of a wormhole that points back to itself.

3.3 Resolution

Due to the fact that all communication in SelNet is mediated through wormholes we need a way of resolving names and other identifiers into the local wormhole names.

In order to perform generic resolution, we insert our own resolution protocol, which we dub XRP (eXtensible Resolution Protocol). The XRP service performs the appropriate resolution to satisfy the request which usually leads to an addressable wormhole that stands for the entity whose name we wanted to resolve. Various styles of resolution are supported by our architecture:

ARP on Ethernet : An IPv4 name is provided, resulting in a wormhole leading to the resolved name i.e., the wormhole hides the underlying Ethernet address.

IPv4 routing : An IP number is provided and the routing lookup is performed i.e., which hop node is next on the path. We can either request a wormhole to be provided which points directly to the final destination or we can request just the IP number of the next hop.

DNS resolution : A logical hostname is provided and a resolved IP number is returned. In the pure DNS case no wormhole is returned, just the information that was requested. We can combine resolution styles together so that the DNS resolution is used in conjunction with the IPv4 routing system so that we give the system a name and get back a wormhole pointing to the end system.

Content-based routing : A content description is provided and a wormhole is returned through which the content is accessible. It is also possible to ask for a wormhole to the node which is capable of processing this query, or to a content group where many further queries are possible, as e.g. outlined in [7].

Content, DNS, IP routing and ARP-style resolution can be thought of in a similar way and we can take both a recursive or iterative approach to them. We can either ask another node to forward a request for us (the recursive approach) or we can ask the node to return to us the next node to query (the iterative approach). In the recursive approach, the wormhole or information that is returned to us represents the end system or content that we wish to access, offering a full resolution service. In the iterative approach the wormhole or information that is returned to us represents the next node on the path to query e.g., the next DNS server to query or the next router on the forwarding path, making this a natural way of embedding traceroute-style functionality.

3.4 Wormhole Glue

Since wormholes have purely local scope we need to have a mechanism to bridge membranes. If we wish to communicate with an entity that is outside the current membrane, then some intermediate membrane must be able to splice wormholes together so that they appear to the source as a single logical path. For this task we use what we term "wormhole glue" to allow intermediate membranes to splice wormholes together so

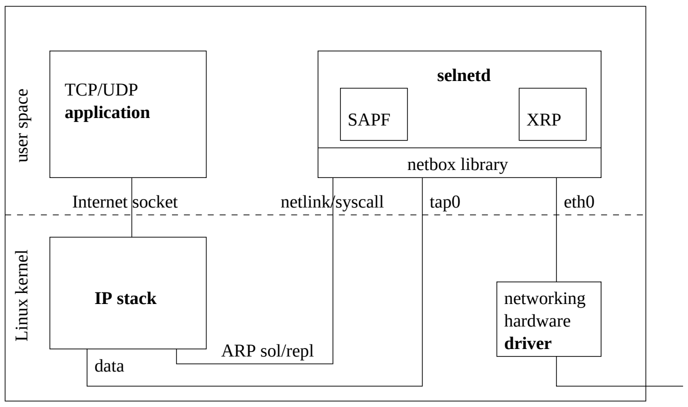

Figure 2: SelNet node architecture

that traffic from one wormhole is forwarded directly in to another wormhole. This is how multiple hop paths are constructed in SelNet. The wormhole glue points are also natural points to introduce any additional functionality e.g., programs injected by an active network framework.

4 Implementation

Here we discuss our current implementation efforts. We have currently implemented a virtual Ethernet (layer 2) layer at layer 2.5 (i.e., between link layer Ethernet and network layer IP) using a user-space Ethernet frame grabber called netbox. See figure 2 for the node architecture. With netbox we are able to take every Ethernet frame from the kernel, manipulate it in userspace e.g., perform header re-writing and then inject the frame back into the kernel again. As well as the underlying netbox functionality, we have two additional software modules: SAPF (the Simple Active Packet Format [8]) is used as the basic demultiplexing engine that also keeps track of wormholes and their associated state, and the XRP module (eXtensible Resolution Protocol) for name lookup and discovery.

4.1 SAPF: Network Shim and hardware mappings

SAPF is a framework for switching datagrams based on a single demultiplexing field called the selector. Selectors typically are dynamically assigned at run time and have only local scope. When sending a payload together with the selector to the next SAPF node, the selector is used to lookup forwarding data linked to this selector and, after possibly rewriting the selector field for the next hop, the packet is sent out again.

The SAPF engine also permits to send packets into the local node's IP stack or call other packet handler routines. The packet handler routines are used to implement the wormhole glue as well as providing the hooks into the XRP module. Wormholes themselves are implemented as selector entries in the SAPF module.

Currently we have two wire formats in our SelNet implementation, both for Ethernet standards: One format is the classical protocol header approach where we prepended a selector in front of the payload. This results in a packet being the sequence of Ethernet header, SAPF selector and payload. In the other wire format we "reuse" the Ethernet header's source address field and store the selector's significant 48 bits there (the other selector bits are used for redundancy purposes and, on an error protected channel, can be omitted): this results in a “stealth” selector field which does not eat up any payload space. Of course, such a selector stuffing approach only works when the Ethernet is not bridged and the source field is not used in the media access protocol which is not the case in wireless LANs for example. Which wire format is used depends on local configuration choices and is not visible from within SelNet. Mappings to other datagram services than Ethernet can be easily added, be it UDP, email or any other tunnelling technology (*-over-HTTP, *-over-DNS etc).

4.2 XRP - eXtensible Resolution Protocol

XRP is our generalized query and steering interface to SelNet. How resolution (from ARP over IPv4 routing to DNS) is handled was already presented at a high level in section 3.3. Here, we present XRP in more detail.

XRP is implemented as a set of request and reply messages. The basic model is that of “remote patching” i.e., sending a command for triggering some side effects and not expecting a reply for this. For those requests where we need some return information, the sender itself has to take care about retransmissions or further investigations as to why some reply did not come back. Note that the XRP interface is available either as procedure calls (inside the SelNet implementation) or via the local control wormhole which can intercept local packets as well as packets coming from a neighbour node. For the presentation of a subset of the XRP command messages we assume that they are sent to a neighbour node:

createWH(dsteth, dstsel, whsel) : This command creates a wormhole on the remote node addressable by the *whsel* selector. When packets are sent to this (remote) wormhole, they are forwarded to the node reachable via the *dsteth, dstsel* tuple (note the rewriting of the selector field that occurs en route). This command typically is used to establish a back channel to the node wanting to query a remote node, but can also be used to splice together multiple wormholes: in this case a wormhole has to be created that points to the downstream wormhole.

resolveName(scheme, value, resolution-depth, replysel) : Once a remote wormhole has been created at other network nodes, we can then request a resolution for a name. The *scheme* field identifies the name space. A common case would be “IPv4-to-eth”. Additionally we also define the scope for this query in terms of resolution depth. In case of a successful resolution, the reply is then sent to the *replysel* wormhole (which has to be present at the node where the resolution is performed and which usually points back to the requester).

reply(data) : The reply itself is shipped inside a *reply* control message. It is defined to have no effect within the XRP engine: The querying application however will be able to pick the requested information from the reply data.

Figure 3 shows the XRP packet format for requests and replies. Note that multiple XRP commands can be put back-to-back in the same underlying datagram carrier: the commands will then be executed sequentially. As a first example on how to perform useful resolution requests with the SelNet command set, we discuss our re-implementation of the ARP service. In a second step we show how this can be easily extended to do multihop path resolution in an ad-hoc routing style.

```
 0                              16                            31
+--------------------------------------------------------------+
| vers  | rsrvd |    msg type   |          data len            |
+--------------------------------------------------------------+
|                  request/reply selector                      |
|                                                              |
+--------------------------------------------------------------+
|                    request/reply data                        |
~                                                              ~
|               .----------------------------------------------+
|               |   ... possible padding if concatenated ...   |
`---------------'
```

Figure 3: XRP Packet

4.3 IPv4 to Ethernet Resolution (ARP)

ARP traditionally resolves IPv4 numbers to their related hardware addresses (Ethernet in our case). To this end, the requesting host broadcasts an ARP query that will be answered by the host having an interface with a matching IP number. This resolved “IPv4-to-eth” mapping is then cached by the requesting host. In SelNet we will represent that mapping state as a wormhole: any packets sent to this wormhole will be sent to the neighbour’s IP stack whose name we resolved. In the following section we describe how this resolution protocol is organized in SelNet.

We start with the IP module issuing an ARP solicit request that we translate to a sequence of XRP messages. First, the requesting node broadcasts a “createWormhole” request, installing a remote wormhole that points back to itself. Second, the requesting hosts broadcasts a “resolveName” command, specifying the previously installed wormhole to be the tunnel through which the result should be delivered. The host with the matching IP number then will send back a reply command that includes the Ethernet hardware address and the selector value that identifies the remote IP stack. With this information we can then create a local wormhole that points to the remote IP stack and inform the local IP stack about the successful ARP resolution. Note that the “createWormhole” request can be put back to back with the “resolveName” command in the same Ethernet frame, resulting in the same message complexity as in traditional ARP.

4.4 IPv4 to Ad-hoc Routing Path Resolution

With a slight extension to the way our new “IPv4-to-eth” resolution works (20 lines of C code) we were able to implement a simple ad-hoc route discovery protocol that we successfully tested with our WaveLan equipped laptops. The basic idea is to have nodes relay a resolution request and to leave sufficient state on intermediate nodes such that we can build a multihop delivery path from the requesting node to the node whose name we wanted to resolve. The state in intermediate nodes will, of course, again be represented by wormholes. This process is depicted in Figure 4.

An intermediate node I will, if it cannot successfully resolve a given IP number, relay the request to nodes that potentially are out of reach of the original requester S. Before doing so, it will install a “reply-forwarder” function that will intercept a successful reply message. Triggered by such an event, the intermediate node creates a wormhole pointing to the final destination T and constructs a reply that lets the original requester S point its wormhole to this tunnel entry. To the requester it looks like an ordinary name resolution request resulting in a wormhole. Behind the scenes, however, the delivery tunnel was transparently extended to forward data packets for one additional hop.

The interesting aspect of this example, beside the very low implementation complexity of very useful ad-hoc routing functionality, is that the usual distinction between

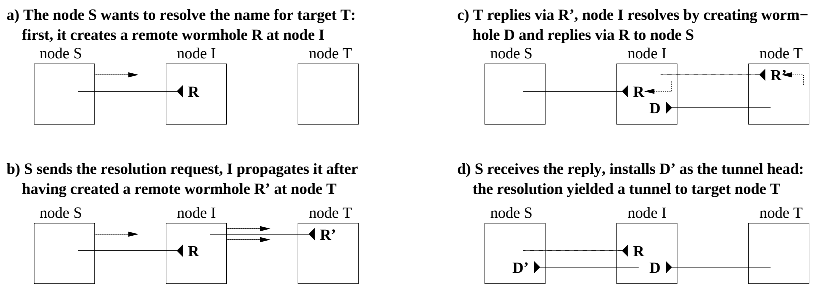
a) The node S wants to resolve the name for target T: first, it creates a remote wormhole R at node I
node S
node I
node T
R
c) T replies via R', node I resolves by creating wormhole D and replies via R to node S
R'
D
b) S sends the resolution request, I propagates it after having created a remote wormhole R' at node T
d) S receives the reply, installs D' as the tunnel head: the resolution yielded a tunnel to target node T
D'

Figure 4: SelNet node architecture

layers (DNS resolving to IP resolving to hardware addresses) is blurred. All an application needs to know about a delivery path is its entry point (the wormhole). IP has its own balanced but hardwired way of resolving names at different points in time and in the network, specifically tuned to the handling of IP addresses. Although connectionless, the IP network stores considerable amount of state in the network. For example, our delivery wormhole corresponds to the cached ARP mapping; Default gateway information in the route table is another example of implicitly stored state, or the IP number of the primary DNS server to contact, which might be stored in the end node itself or in some DHCP configuration server.

Coming back to the case of ad-hoc routing we can observe that ad-hoc routing protocols for wireless networks essentially maintain stateful delivery paths between communicating nodes, as do BGP routers at the AS level, hence our extended ARP is as stateful or stateless as its ad hoc routing counterparts in the IP environment. Note that our model does not dictate applications to setup and maintain individual delivery paths: depending on the node's configuration, resolution requests can be handled by different modules either working statefully (resolve and cache) or stateless (resolve for every new packet) or any mixture thereof. In the case of running our extended ARP inside SelNet, we effectively do level 2.5 routing and fool IP about the ad-hoc nature of the underlying wireless network.

With an appropriate resolution mechanism that merges address resolution with route selection we can create completely virtual (IP) networks which can be accessed transparently. It is noteworthy that the address types used for such networks (logical names, IP numbers, content identifiers, email addresses, house numbers etc.) are not part of the SelNet architecture, which only provides the hook to map them to its own, native selector and the wormhole concept. This in fact is also the reason why the SelNet architecture does not feature addressable nodes as there is no way of directly addressing a node (selecting a delivery function) without the resolution step.

5 Application of SelNet: "Highway Routing"

Here we describe one application of our SelNet system: "Highway Routing". We model a new style of routing which uses a motorway or highway metaphor. The main concept here is the usage of SelNet to provide an add/drop multiplexing mechanism for packet-switched networks i.e., a routing style which sits between end node addressing and source routing. We imagine a route between locations, for example, between America and Europe - we call this route the *"Atlantic Highway"*. Along this route which runs from San Francisco to St. Petersburg there are many drop points. These points are addressable and represent various locations en route e.g., Berlin, Stockholm,

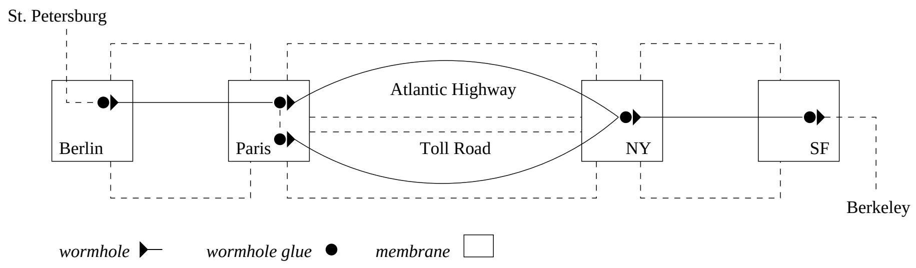

Figure 5: Highway Routing

Paris, London, New York etc. The drop points are freely accessible to traffic flowing through the highway. All that a packet has to do to exit the highway at a certain point is set its exit selector to the name of the appropriate drop point and then it will be ejected from the highway.

We envisage the Atlantic Highway to be a provider backbone which advertises its destinations to membranes in various locations. The cities represented on the Highway would be one such set of locations where the Highway destinations would be advertised. A node inside a membrane representing Berlin will be able to see a wormhole entry point to the Atlantic Highway and the drop points that are available. The node is then able to send traffic to destinations that are reachable via the Atlantic Highway and can set its exit point to be any of the drop points advertised by the highway.

The Atlantic Highway can also be used by a node to create its own paths through the network. This is the scenario shown in figure 5. If a node in Berlin wishes to access a node in Berkeley it can build a path which uses another Highway for part of the route, perhaps a high-speed but costly "toll-based" link stretching from Paris to New York. Another reason for making such a detour is due to link congestion and availability. If there is an outage in part of the Atlantic Highway then such a scheme allows users to route around this. This dynamic re-routing to avoid congestion or outages is similar to the motivation behind some work on Overlay Networks e.g.,the RON project [4]. In such a scenario, the node could send packets via the Atlantic Highway to Paris by setting the exit point on the packet to be the Paris drop point and then selecting the *Paris New York* toll highway, after having paid for the corresponding wormhole entry. The packet will then exit the Toll Road at New York and resuming the journey on the Atlantic Highway. Note that there would have to be multiple exit points on the packet's stack to enable it to specify its route through the network.

6 Conclusions

With the SelNet architecture we introduce two basic constructs: "wormholes", which are essentially underspecified sockets, and a unified resolution entry point where DNS, ARP and route discovery are merged. At the core we have selectors which are used to address locally defined functions. Using the resolution mechanism, names or other identifiers are dynamically mapped to selector values i.e., wormholes. These wormholes can then be viewed as a tunnel entry which points to the entity that we wished to resolve. It is also the case that wormholes can be pointed at other wormholes, enabling the concatenation of tunnels. Due to the local naming of wormholes, packets travelling along the path will have their selector value rewritten at each hop.

In order to allow wormholes to be pointed to the desired functions on nodes, we use XRP to allow name resolution and route discovery and map the results to wormhole entries. For example, we trap IP-level ARP requests and translate them into XRP procedure calls. Typically ARP works on a one segment network view, whereas we extended our resolution protocol to allow the relaying of requests: this results in a resolution request yielding a transparent tunnel traversing multiple segments. The implementation of such Ad-Hoc routing functionality was performed with around twenty lines of C code. We plan make the source code of the implementation publically available. An Internet Draft documenting both the XRP protocol and the Ad-Hoc routing scenario to accompany this release is in preparation.

Our direction for future work mainly concentrates on XRP. A possible evolution of XRP could take it in the direction of becoming a scripting language instead of a Remote Procedure Call system. Extending the instruction set to provide support for persistence storage could be important for maintaining long-lived wormholes in the face of node failure. Finally, many questions surrounding multicast, security and resource control remain to be addressed.

SelNet is an extensible framework that provides a redirection toolbox where selectors play the role of pointers. The functions that the selectors point to can be statically configured as in conventional networks or dynamically instantiated by an active networking approach. As an underlay network it can be used to build new networking abstractions such as Highway Routing.

References

[1] E. Rosen, A. Viswanathan, and R. Callon. Request For Comments 3031: Multiple Protocol Label Switching, 2001. `http://www.ietf.org/rfc/rfc3031.txt`.

[2] David Cheriton and Mark Gritter. TRIAD: A new next generation internet architecture, 2001. `http://www-dsg.stanford.edu/triad/triad.ps.gz`.

[3] Paul Francis and Ramakrishna Gummadi. IPNL: A NAT-Extended Internet Architecture. In *Proceedings of SIGCOMM 2001*, 2001. `http://www.acm.org/sigcomm/sigcomm2001/p6-francis.pdf`.

[4] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient Overlay Networks. In *Proc. 18th ACM SOSP, Banff, Canada*, 2001. `http://nms.lcs.mit.edu/papers/ron-sosp2001.pdf`.

[5] Tilman Wolf, Dan Decasper, and Christian Tschudin. Tags for high-performance active networks. In *OpenArch'00*, 2000. `http://www.docs.uu.se/~tschudin/pub/cft-2000-tan.pdf`.

[6] Jon Crowcroft. 20 things we got wrong with IP, 2001. `http://www.cl.cam.ac.uk/users/jac22/talks/cl-talk.pdf`.

[7] Richard Gold and Dan Tidhar. Towards a Content Aggregation Network. In *Proc. of the International Conference on Peer-to-Peer Computing (P2P2001)*, Linköping, Sweden, August 2001. `ftp://ftp.fokus.gmd.de/pub/glone/usr/rgo/pub/p2p2001.ps.gz`.

[8] Christian Tschudin and Dan Decasper. Internet draft: Simple Active Packet Format (SAPF), 1998. `http://www.docs.uu.se/~tschudin/pub/cft-1998-sapf.txt`.

[9] David Plummer. RFC 826: Ethernet Address Resolution Protocol, 1982. `http://www.faqs.org/rfcs/rfc826.html`.