Network Pointers

Christian Tschudin
Uppsala University and
CS Dept, University of Basel
Bernoullistrasse 16
CH–4056 Basel, Switzerland
Christian.Tschudin@unibas.ch

Richard Gold
Department of Computer Systems
Uppsala University
Box 325
SE–75105 Uppsala, Sweden
rmg@docs.uu.se

ABSTRACT

The Internet architecture can be characterized as having a rather coarse grained and imperative style of network packet handling: confronted with an IP packet and its source and destination addresses, the infrastructure almost blindly and unalterably executes hundreds of resolution, routing and forwarding decisions. There are numerous attempts that try to "extend" the Internet in order to either reduce the immediate impact an arbitrary packet can have (e.g., NAT) , or to insert diversions from the normal processing paths in order to better use the existing resources (e.g., content delivery). In this paper we argue that we need a more fine grained control, in the hands of end nodes, over how packets are handled. The basic abstraction presented here is that of networking pointers, which we show to relate to low level concepts like ARP caches, but also high level routing decisions for terminal mobility, content delivery networks, or peer-to-peer overlay forming. We report on first implementation experiences of an "underlay" networking approach which uses pointer tricks underneath IP in order to provide new network layer services.

1. INTRODUCTION

The universal connectivity provided by the Internet architecture is the major source of the stress that it is currently experiencing. Because the Internet has one, and only one method to identify a destination (IP address), to deliver datagrams (routing tables managed by third parties) and to address services (ports), there have been many attempts to introduce more flexibility into the Internet:

- Overlays, be it for peer-to-peer, virtual private networks, or route optimization à la the Resilient Overlay Networks project [1], redo what the IP layer is supposed to do: packet forwarding. The difference is that end nodes explicitly want to have a say in how forwarding should be done.
- The IP address merges three different networking concepts in a perhaps elegant but troublesome way: identity, location and access. Changing location whilst preserving identity is the source for the Mobile IP proposal. On the other hand, the conflict between identity and access is where firewalls and NATs intervene.

Changing the Internet's packet forwarding behaviour really means changing the way the Internet manages state. We ask the question: based upon which header fields, which routing tables and which lookup processes should forwarding decisions be taken? There is a myth associated with the Internet that it is stateless. However, this only refers to per-connection state and does not apply to the infrastructure itself. On the contrary, a surprisingly large amount of configuration state is present in the network: default routes, address ranges in DHCP, BGP tables, OSPF tables, DNS content, port to service mappings, and more recently the addition of per-connection state in the form of NAT mappings.

1.1 Network Pointers in a Nutshell

Network pointers provide a conceptual and programming framework for packet processing in general. In the first place, a network pointer is an arbitrary packet processing function that can be "addressed" through an opaque pointer value.

A simple example would be the sending of an IP packet to an end point: the destination IP address is mapped to the IP address of the default gateway. This is then mapped to the ethernet address of the gateway's interface and cached in an ARP table entry. In fact, after the aforementioned mapping procedure, the use of any (local) pointer value identifying the cache entry would suffice to forward data to the destination. What we propose is to make that resolution from names and other entities to pointers an explicit architectural element across the whole network. Network pointers and their remote instantiation play a key role in providing "directable indirection" to the network users and operators.

Before going into more details we now review some items of related work which concern themselves with the manipulation of state (and thus packet processing functionality) in the network. This state is typically used to alter the amount of direction and indirection in the network.

This work was done under the VINOVA project SCANET (Self-Configuring Ad hoc Networks). We gratefully acknowledge the support from the European COST.263 action on "Quality of Future Internet Services".

1.2 Related Work

The *Resilient Overlay Networks* (RON) project [1] is designed to increase the reliability of the routing paths present in today's Internet routing infrastructure. It works by creating an overlay network of RON nodes which all agree to forward packets on behalf of the other nodes in the overlay network. During typical operation a RON node probes the standard paths provided to it by the network for failure. When such path failure is detected, the node then attempts to route around the failure. We see RON as re-creating loose source routing at the overlay level. It builds alternate routes (indirection) over an IP routing layer that cannot be steered i.e., which is too direct. Similarly to RON, Mobile IP had to also invent its own routing agents in order to work around the lack of influence an end-user has on routing decisions.

The fact that the Internet suffers from being overly direct has also been picked up by the Internet Indirection Infrastructure (i3) project [9]. They see indirection, as a generic concept, is sorely needed in the current Internet. In order to provide this, they use a Peer-to-Peer lookup service which implements a Rendezvous approach i.e., meeting in the middle. A receiver puts a key called a trigger into the lookup service. This trigger is then used by the sender to route a packet through the i3 overlay network to the receiver. Triggers are much like network pointers, except that i3 restricts itself to IP addresses as the only type of pointer values and to IP forwarding as the single supported packet processing function.

Indirection can also be a problem, as exemplified by NAT boxes and Firewalls which introduce indirection in an implicit way. Many tunneling mechanisms have been devised (e.g., PPP-over-SSH [3]) in order to bring back a direct connection (IP-wise) between two end systems when it is desired. Being unable to control when direction and indirection occurs is a major hindrance in the Internet of today and has resulted in many patches in order to get over these problems. We see NAT and overlay networks as a general symptom of this lack of control.

1.3 Got state?

In the face of this inaccessible state and inability to influence the directness of the current Internet, we propose a program of Internet deconstruction: breaking the packet processing black box into components and allowing its recombination in a user-controlled manner via pointer manipulation. Network Pointers, like their programming language cousins, allow the user to control the amount of in- and re-direction. We then present our own architecture based on the idea of Network Pointers and late binding of address semantics. By shifting the focus to underlaying IP (instead of overlaying IP), much richer and optimizable network recombinations are possible, as we are able to combine the components of the IP stack in different ways. IP (and its associated socket library) then becomes an access mechanism and emulation target, whereas overlays – now supported at the same level as IP – are a way to create partial and transient clouds of emulated directness.

2. NETWORK DECONSTRUCTION

In this section we briefly discuss three network usage scenarios that serve as prototypical examples for the deficiencies of the Internet's one-model-fits-all approach. We will also show how these scenarios would benefit from "network pointers". A network pointer basically stands for a packet processing function. If a packet is "sent to a pointer's value", it will be processed according to that pointer's function. This function can be, for example, the forwarding of a packet whose header contains a link layer address and the pointer value of a next (remote) network pointer. How these pointers are allocated and managed will be discussed in the following section.

2.1 Case 1: Routing in Ad hoc Networks

Ad hoc networks are supposed to set up Internet connectivity in a completely decentralized way, without any help from preconfigured networking elements, and in very short time. One class of successful ad hoc routing protocols works "on-demand" i.e., they explore the network topology only when there is traffic to be delivered. In this case, the source node establishes delivery *paths* that run through the network. These paths funnel all traffic for a destination in the right direction. Expressing this in other terms, on-demand ad hoc routing protocols establish per-target "connection trees" whose root is located at the target node.

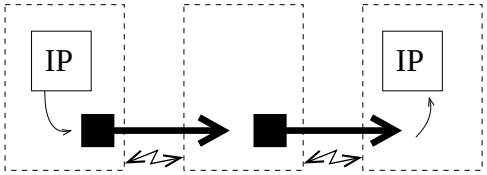

Figure 1: ARP forwarding and pointer selection in wireless ad hoc networking.

We can easily represent that tree with forwarding pointers being the nodes *and* the connecting edges. Instead of labeling these pointers with the target's name, we imagine that a pointer (node) has a *local* pointer value and that all neighbours just point to that address (i.e., "link layer" details plus pointer value at the remote node). This is shown in figure 1.

Our approach differentiates the "access" (pointer value, which is a simple label) from the "location" (full delivery path). Data packets sent to the first *access* pointer will be correctly forwarded, regardless of the final destination's *location*.

2.2 Case 2: Asymmetric Connectivity (NAT + Firewalls)

NAT boxes, either introduced for coping with the scarcity of IP addresses or for controlling network traffic, break the basic universal connectivity model of the Internet. Some nodes or services behind the NAT are not accessible from the "ordinary" Internet, while they can see and connect to the "ordinary" nodes. In order to overcome this asymmetry, people have started to setup tunnels and reverse tunnels to access machines behind NAT boxes. A legitimate use of such tunneling is a traveling user wanting to access some files that he left on his home machine unfortunately located behind a NAT. First, before leaving, the user would have had to create a persistent tunnel from his home machine to some

waypoint machine that has full connectivity to the Internet. Secondly, when abroad, the user has to login to the waypoint machine, and from there through the reverse tunnel connect to the home machine. Overall, this exports one port i.e., SSH from behind the NAT to the outside world.

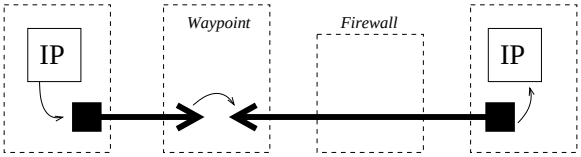

Figure 2: Using an outgoing tunnel to pass through the firewall.

By using network pointers in this scenario, it is possible to concatenate together application-layer tunnels (e.g., SSH) as well as standard IP paths into one seamless construct. Typically the traffic from a node in the traditional Internet to a node behind a NAT box has to be manually directed to a waypoint, and then from the waypoint into the tunnel to traverse the NAT box as shown in figure 2. By allowing user-defined mappings to be established between pointer values and their associated packet processing functions, we can use different layers of the networking stack on a per hop basis. This enables us to present nodes behind NAT boxes as nodes with full Internet connectivity.

2.3 Case 3: Mobile Personal Area Networks (PAN)

An important new challenge for computer networks are small area networks linked to a person, which interconnect the cellular phone with the embedded PC in the belt, the wireless ear plug, and the MP3 player in the pocket etc. Networked devices travel as a clique and rely on special nodes (e.g., the cellular phone) to connect them to the rest of the Internet. Because of the intermittent connectivity and often changing access points to the fixed network, it would be problematic if all devices in the PAN were to handle routing and addressing on an individual basis.

In this case we use a routing data structure comprised of two trees where one tree (i.e., set of pointers) encapsulates the PAN and its components. In figure 3 this is the tree from the fixed network to the PAN, where the dotted lines represent logical paths to nodes inside the PAN and the unbroken line represents the physical path. Another tree is then used from inside the PAN to point from the current gateway to the PAN components to allow them access to the fixed network. This process is shown in figure 3. This pointer from the fixed network to the PAN is addressable i.e., other nodes can then use this pointer as a way of communicating with the PAN.

The pointer inside the fixed network can also be called a "point of presence", or rendez-vous place in terms of the Internet Indirection Infrastructure proposal. Again, we dissociate identity from location and change the pointer values and names locally and on demand: the data structure's end points (the PAN members, and the "point of presence" in the fixed network), remain stable, while the intermediate pointers can change without requiring the end nodes to re-

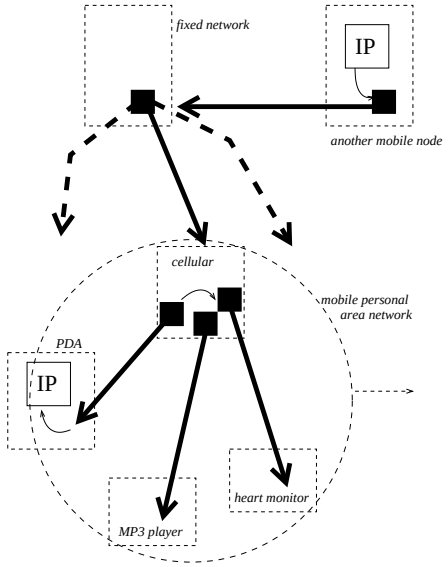

Figure 3: Changing forwarding pointers to the PAN as a whole.

act on this.

2.4 The Deconstruction Plan

Our general approach with network pointers is to facilitate the introduction of new functionality into the network in the following way:

We keep IP and applications running on end nodes as they are. We then go under IP as we wish to have the ability to influence the functionality of the lower layers rather than just treating them as a black box. This allows us to configure the logical layer 2 and the ability to perform redirection at this layer. Addresses at the IP layer are translated to internal access names i.e., pointer labels. Pointers are responsible for either forwarding a datagram to a neighbour, or for modification of packet headers (address translation, tunneling, header compression, etc).

Once we are in the "pointer space", we can begin to build our own abstractions. For example, we can bring an Ad-Hoc networking cloud into the IP space without IP having to be concerned about the multihop conditions of the underlying network. Simplification is also the order of the day with the Firewall/NAT traversal scenario. Rather than having multiple manual steps from the NAT'd machine to the waypoint and then from the user's machine to the waypoint, we make pointer allocation methods available that enable a semi-automated setup of NAT tunnels.

An important aspect of a pointered network architecture is the management of pointer state which we require to pass through a mandatory resolution step. Pointers become a natural translation point not only for addresses and tunnel splicing, but also for management domains. By restricting resolution e.g., linking it to authentication, we yield more control over a network's infrastructure to the operator. Ultimately, the plan is to deconstruct IP forwarding and addressing semantics in order to build new network layer ser-

vices in our own "Pointer Space" that can be mapped back into the IP space. How resolution and pointer allocation works is explained in more detailed in the following section.

3. SELECTORS AND RESOLUTION

3.1 Terminology

A network pointer is a packet processing function that has a local address, called a selector. To send a packet to a network pointer, you need to know the pointer's address (selector) and the context in which that selector is valid. Inside a context, which is typically a host, packets can be routed through different pointers based on selectors only. In order to send packets to pointers residing in other contexts, one also needs to specify a transport specific context address (Ethernet, IP, Application-layer tunnels etc.). In this case of a packet forwarding function, the network pointer itself contains the address of the downstream function i.e., the link layer address and the remote selector.

In case of Ethernet, packets would carry a link layer address and a selector. The link layer address determines the context in which the function can be selected. A forwarding chain of network pointers would rewrite a packet's link layer address and selector value on each hop i.e., the network pointers contain the address of the downstream node as well as the selector value of the remote pointer.

Selectors are local i.e., they apply to the local context. This enables one to route packets through local pointers in a uniform way: network pointers become the back plane of protocol stack components (see also section 4). At the same time, selectors can point to network pointers that transport packets to other contexts. In this way contexts can be nested, which leads to layers and tunnels. In the tunnel case, the selector would point to a tunnel entry function that pushes its own addressing data in front of the packet header. The tunnel exit function, activated by the tunnel selector on the last forwarding leg, strips out that additional addressing. Protocol layering is obtained by letting each protocol entity of a layer sending packets for higher layers to the selector of the next higher protocol entity.

We can use the network pointer abstraction to represent forwarding state (possibly instantiated by end nodes) in the network *or* in the packet header. If contexts are globally addressable we can use network pointers as remote relay points in order to implement a cached form of loose source routing: packets only need to carry the name of the first pointer, from where they will be passed on from one pointer to the other. Alternatively we can send packets to specific transport pointers that use the packet header as a parameter stack by reading the destination parameters immediately following the packet's selector field.

3.2 Network Pointers and C-Pointers

The term "network pointers" relates to the standard forwarding behavior of network nodes: Routers "redirect" incoming packets to other directions. This is basically a dereferencing operations (usually involving some lookup in a router table) and matches well the classical concept of pointers in programming languages. The main difference is of course the distributed control: in our network pointer view, each data packet becomes a thread of control that works its way through a chain of dereferencing operations. Whereas a list traversal in the classical programming world would be carried out by a single CPU. The other difference is that we generalize the concept of pointer and include with it the possiblity to mangle packets, hence we see network pointers as generic packet processing functions. Note that there is not a one-to-one mapping between packets in and packets out for a network pointer. It may produce two or more outgoing packets (if multicast semantics are required) for each incoming packet or it may require two or more incoming packets per outgoing packet.

3.3 Selectors as Local Addresses

Network pointers are addressed by a pointer value (selector) that we prepend to each data packet. In order to avoid spanning the name space for all potential pointers (and having to enumerate them at a global level), we require that selectors are purely local names. Only those pointers can be addressed that actually are resident in a specific context. The selector, typically implemented as an opaque number, will be used at the receiving context to find the network pointer that the packet should be processed by. We leave the issue of end-to-end addressing to whichever higher-level protocols are using the network pointer infrastructure.

The local naming scheme means that selectors have no global meaning and that, for example, packets being forwarded from one context to the other have to change their selector on each hop. Selector (or header) translation thus is an intrinsic property of a pointered network architecture. To some extent this also applies to current IP networking, where the destination address is repeatedly rewritten to some local ethernet address that is different for each hop.

3.4 Name Resolution

Due to the fact that local pointer names shall be the generic and sole addressable items, we need a way to map end-to-end destination addresses or server names to local selector values to other kinds of identifiers, including global addresses. To this end we provide a generic resolution function residing on all network nodes to which we assign a well–known selector value. Using this well-known selector, an application can request address resolution and get a local pointer name that "stands for" the item to resolve.

For example, a request for resolving a neighbour's IPv4 name to the Ethernet address (à la ARP) would result in a local delivery pointer being instantiated on the fly which takes care of delivering any packets sent to it to this neighbour. In fact, such function instantiation on the fly already occurs today inside protocol stacks where an ARP cache keeps that mapping in memory. The selector view makes this state explicit and presents the associated network pointer in the form of an addressable 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 header translation (which is much cheaper), essentially rewriting the local delivery selector into the associated Ethernet address details.

Extending this view we map routing to selectors too. Finding the next hop is in fact a resolution request for a route.

This can be done for each single packet or for a complete packet flow, as for example, in on-demand routing protocols for ad hoc networks. As a result, selectors become the (local) names of delivery path entries.

Explicit name resolution is also the entry point for substituting content descriptors or locators with access paths to the (closest) place from which content can be fetched, which is very similar to the approach proposed in TRIAD [7].

3.5 Start-Addresses, not End-Addresses

Unlike IPv4 addresses, a selector does not define a packet processing semantics in advance and across the whole network. Instead, explicit resolution activities are required to bind a selector to some network pointer (packet handler). We have thus moved from an end-address point of view to a start-address point of view. This enables to repointer parts or all of a packet processing chain in order to cope with mobility without having to change higher layer applications (see e.g., example 3 on moving personal area networks).

3.6 SelNet - An Implementation for Network Pointers

We have implemented a Selector Network for the Linux operating system. From former explorations we already knew that packet forwarding based on simple selector lookup rather than IP routing, can be done with 30% less overhead [13]. What we wanted to understand better was the role of the resolution protocol as a general tool for setting up the network pointers.

Consequently we implemented an "eXtensible Resolution Protocol" (XRP) and combined it with the SAPF [11] packet format which represents our network pointers. As a proof of concept for our underlay approach we applied the pointer networking approach to wireless ad hoc networks. The goal was to fool IP about the multihop nature of an ad hoc network by implementing ARP forwarding: ARP requests would return the name of a full delivery tunnel to use instead of a next hop link layer address.

The resulting LUNAR ad hoc routing protocol (Lightweight Underlay Network Ad hoc Routing [12]) works by trapping ARP requests and translating them into XRP queries that are sent to neighbouring nodes. Based on these queries, the neighbours create new network pointers and propagate the request until the destination is found. To our surprise, we could get a first LUNAR version up and running very fast (in hours). Also, the implementation matched the performance of well-established ad hoc routing protocols, although requiring only 30 to 60% of their code size. A stripped down version of LUNAR was even ported to the Lego Mindstorm embedded devices [8].

4. FUTURE DIRECTIONS

We envisage network pointers to become a pivotal element in future protocol architectures, both at the conceptual and the implementation level. We describe in this section some possible future applications of network pointers.

Protocol heaps are proposed in [2] as a middleware concept for letting applications influence the processing of packets through the network. Beside the correspondence at the representation level (heap and pointers), we believe that network pointers provide an essential building block for steering packets through processing components and for identifying processing instances. For example, the NAT example presented in [2] makes use of opaque "cookies" which we would map to selectors in a much more natural way: network pointers would then play the role of a cabling back plane for protocol heap modules.

Related to protocol stack engineering we point to work done a decade ago on speeding up packet processing called **active messages**: several header parsing steps can be avoided by putting the upcall's memory address directly into the packet's header [6]. This works well in the controlled environment of distributed systems: for an open network environment, however, additional protection of handler addresses is required. In our network pointer approach we use local selectors as well as an explicit resolution step for retrieving the dynamic selector values.

The use of selector-like state identifiers has been recently proposed in [4]: **Ephemeral state processing** (ESP) enables to instantiate short and predefined remote computations in a lightweight fashion. Each packet can carry a single ESP instruction in order to operate on the "ephemeral state store" residing in each node. State entries are identified with a randomly chosen 64-bit tag, which corresponds very well with our selector concept to address state and functions.

Finally we point to our own ongoing work on **stored program router** [10] where the forwarding table is turned into a general programming area. Individual table entries contain single instructions (e.g., forward, push label, conditional forward) such that packets can be processed by routing them through a chain of table entries. Because packets can also influence the table's content, we obtain a complete computing model where threads of computations are carried by single packets. The chaining of execution steps is done via the packet's selector value which becomes a direct *instruction pointer*. In terms of the network pointer concept, we organize the routing table as a single context consisting of an array of tiny network pointers.

5. SUMMARY & OUTLOOK

A "network pointer" basically is a packet processing function similar to the indirection support proposed in the i3 project. However, instead of using IP addresses as the target we propose to use a separate local name space (selectors) for labeling network pointers. This enables a precise separation of the IP address meanings (identity, location, access), although only a single naming concept is used.

More generally, network pointers represent packet processing functions that can range from ad hoc path discovery to multicast forwarding, content access, or VPN encryption. Network pointers bring back part of the Catenet architecture [5] where explicit control of address translation and concatenation was a key concern before the flat Internet model began to rule the world. Now that the Internet has become asymmetric and fragmented for good or for bad reasons, we

need to empower the end nodes, enabling them to recreate end–to–end services by rewiring the network's packet processing functions.

Acknowledgements

The authors wish to gratefully acknowledge the useful discussions and comments of , Per Gunningberg, Erik Nordström, Arnold Pears and David Lundberg. The comments of the anonymous reviewers also helped form the later version of the paper.

6. REFERENCES

[1] David G Andersen, Hari Balakrishnan, M. Frans Kaashoek and Robert Morris. Resilient Overlay Networks Proc. 18th ACM SOSP 2001. http://nms.lcs.mit.edu/papers/ron-sosp2001.html.

[2] R. Braden, T. Faber, and M. Handley. "From Protocol Stack to Protocol Heap – Role-Based Architecture". To appear in: First Workshop on Hot Topics in Networks (HotNets-I) 2002.

[3] Scott Bronson. VPN PPP-SSH Mini HOWTO. http://www.ibiblio.org/pub/Linux/docs/HOWTO/mini/other-formats/html_single/ppp-ssh.html.

[4] Kenneth L. Calvert, James Griffioen and Su Wen. Lightweight Network Support for Scalable End-to-End Services. In: SIGCOMM 2002. http://www.acm.org/sigcomm/sigcomm2002/papers/esp.pdf

[5] Vint Cerf. The Catenet model for Internetworking, 1978. http://www.isi.edu/in-notes/ien/ien48.txt.

[6] T von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active Messages: a Mechanism for Integrated Communication and Computation. In: Proceedings of the 19th Int'l Symp. on Computer Architecture. May 1992.

[7] M. Gritter and D. Cheriton. An Architecture for Content Routing Support in the Internet. Usenix Symposium on Internet Technologies and Systems 2001. http://www-dsg.stanford.edu/triad/usits.ps.gz.

[8] Anders Linden, Johan Loennberg, Olof Rensfelt, and David Rosen. μLUNAR, 2002. http://www.docs.uu.se/selnet/lunar/ulunar/.

[9] Ion Stoica, Dan Adkins, Sylvia Ratnasamy, Scott Shenker, Sonesh Surana and Shelley Zhuang. Internet Indirection Infrastructure. IPTPS 2002. http://www.cs.berkeley.edu/~shelleyz/papers/iptps2002.pdf.

[10] Christian Tschudin. Stored Program Routers. Work-in-Progress, 1999–2002. http://user.it.uu.se/~tschudin/pub/cft-wp-spr.pdf

[11] Christian Tschudin and Dan Decasper. Active Networks Request for Comment: Simple Active Packet Format (SAPF), 1998. http://www.docs.uu.se/~tschudin/pub/cft-1998-sapf.txt.

[12] Christian Tschudin and Richard Gold. LUNAR: Lightweight Underlay Network Ad-hoc Routing. Submitted for publication. 2002. http://www.docs.uu.se/selnet/lunar/lunar.pdf.

[13] Tilman Wolf, Dan Decasper and Christian Tschudin. Tags for high-performance Active Networking. Proc *Openarch 2000*.