4+4: An Architecture for Evolving the Internet Address Space Back Toward Transparency*

Zoltán Turányi, András Valkó, Andrew T. Campbell
COMET Group, Columbia University
2960 Broadway New York, NY 10027
{zoltan,andras,campbell}@comet.columbia.edu

ABSTRACT

We propose 4+4, a simple address extension architecture for Internet that provides an evolutionary approach to extending the existing IPv4 address space in comparison to more complex and disruptive approaches best exemplified by IPv6 deployment. The 4+4 architecture leverages the existence of Network Address Translators (NATs) and private address realms, and importantly, enables the return to end-to-end address transparency as the incremental deployment of 4+4 progresses. During the transition to 4+4, only NATs and end-hosts need to be updated and not the network routers. The 4+4 architecture retains the existing semantics of Internet names and addresses, and only proposes simple changes to the network layer that focus entirely on address extension. Encapsulation is used as the main tool to maintain backward compatibility. We present the design, implementation, and evaluation of the 4+4 architecture and discuss our implementation experiences and results from local and wide-area Internet experimentation. The 4+4 source code is freely available from the Web (comet.columbia.edu/ipv44) for experimentation.

1. INTRODUCTION

One of the major challenges facing the current Internet is the exhaustion of the IPv4 address space. In 1994, the IETF Address Lifetime Expectations (ALE) Working Group projected the exhaustion of the IPv4 address space to be around 2008 [6]. Currently, there are two competing solutions to IPv4 address extension: Network Address Translators (NAT) [8] and IPv6 [16]. NATs are routers that can modify the IP address and port numbers of passing packets. This enables the reuse of a small number of public addresses by many hosts behind a NAT, if only a few of those hosts communicate with the outside world simultaneously. NATs have helped reduce the rate of address depletion, enabling continued operation even in regions with shortages of IP addresses. As discussed widely in the literature [23, 25], NATs also contributed to the loss of IP transparency [21]. As a result NATs prevent IP level end-to-end security, reduce robustness, break a number of application level protocols, complicate the network, and inhibit novel uses of the Internet (e.g., peer-to-peer networking). Despite such disadvantages NATs are widely deployed in the network. And, although NATs adversely impact the global Internet, they represent an attractive technology for many private network operators for a number of reasons. First, if the acquisition of public IP addresses is hard, complicated, and expensive, then setting up a NAT is quick, easy, and cheap. Next, for many end users a "NAT-ed" connection seems good enough. Third, alternative solutions (e.g., IPv6 deployment) seem too complicated and disruptive to deploy at present. Finally, NATs enable the isolation of a private domain's addressing and routing from that of the public Internet.

In 1994, the IETF selected IPv6 [16] as the next generation of the Internet Protocol. At that time, however, NATs were not yet in widespread use. The ngtrans Working Group of IETF [29] developed several transition mechanisms that would allow the temporary co-existence of IPv4 and IPv6. However, despite the availability of IPv6 and transition procedures, IPv6 has seen little practical deployment. One reason is the complexity associated with the transition. If one cannot replace all the routers and hosts in a site at once, then a period of co-existence follows. During such a period network of co-existence administrators must manage both IP versions, plus a number of transition mechanisms. While transition to IPv6 would benefit the whole Internet at the expense of the extra work at individual networks, using NATs benefits the individual networks directly, but negatively impacts the global Internet. This dilemma between NATs and IPv6 is best discussed in [24].

If IPv6 is ever fully deployed it is likely that the transition to IPv6 may last for a long period of time in the global Internet. In fact, it is possible that it may never be completed with a significant portion of the Internet remaining IPv4 only. Such a situation could occur if a certain population or sector (e.g., the cellular industry because of the projected growth in cellular Internet devices, or regions with shortages of IP addresses) found sufficient incentives to transition to IPv6 while others remained content with IPv4. This would result in a possibly lengthy partitioning of the Internet with intensive use of protocol translation and tunneling mechanisms, a result that could be even worse than the present situation with NATs. In short, it is possible that IPv4 and NATs will be a permanent fixture of the communications infrastructure for sometime to come. If this is the case, then a markedly different approach is necessary to solve the address depletion and transparency problems. One important alternative approach is represented by *NAT extended architectures* [27], which rely on the existence of multiple address realms and extend the address space by requiring changes only to hosts and NAT devices.

In this paper, we propose the 4+4 architecture [17], which presents one example of a NAT-extended architecture. 4+4 provides a way

back to unique, global, network layer host addresses, while maintaining some of the address isolation features of NATs. End-to-end transparency is restored in the Internet to the extent of 4+4 deployment. If 4+4 deployment becomes complete then IP address transparency is fully restored. There is no need to change routers. The transition process provides incentives for networks with both private and public addresses to upgrade and increase transparent reachability. During transition, existing IP and NAT mechanisms are used to communicate with, and between IPv4 hosts, thus, transition does not affect existing reachability. All transition mechanisms are part of either the current practice or the "final architecture", thus, there is no need to set up temporary transition features. The final state of the network is a homogeneous and transparent Internet that uses 64-bit addresses and a packet format similar to tunneling.

The paper is organized as follows. Section 2 provides a brief overview and discussion of previous address extension proposals. Section 3 and Section 4 describe the architecture and operation of 4+4, respectively. The 4+4 transition process with a brief comparison to IPv6 is presented in Section 5. A minimal impact 4+4 implementation is described in Section 6. In Section 7, the resilience, performance, and application compatibility of 4+4 is evaluated through experimentation using local and wide area 4+4 testbeds. Finally, Section 8 presents some concluding remarks.

2. RELATED WORK

There has been a considerable amount of work on IPng documented in the literature. Much of this work, however, predates the design and deployment of NATs. The Simple Internet Protocol Plus (SIPP) [10], which later became IPv6, includes a built in address extension mechanism. In SIPP, host addresses were originally 64-bits long with an additional "cluster-addressing" mechanism. Cluster addresses are 64-bit unicast addresses referring to a set of nodes behind boundary routers. When complemented with a 64-bit host address they enable the extension of the address space in quite a similar manner to 4+4. The cluster address selects the boundary router and the host address selects the host. Other uses of cluster addressing include mobility and provider selection. Later the IPv6 address space was extended to 128 bits and cluster addressing was omitted and merged into source routing.

Mike O'Dell proposed the separation of identifiers from locators in his 8+8 proposal, later known as GSE [15]. In 8+8 half of the 128-bit IPv6 address is used as a locator (termed "routing goop") and the other half as a host identifier. End systems are not aware of their full routing goop, only the part that describes their location inside their site. Site border routers place the missing part of the routing goop into the source address of outbound packets. Although there are some similarities between 4+4 and 8+8, a number of differences exist. First, in 4+4 no part of the address is used as a location independent host identifier. Second, border routers in 4+4 do not rewrite addresses; although realm gateways associated with 4+4 rearrange addresses when forwarding packets, no new addressing information is added to the packet. Next, 4+4 hosts are aware of their full addresses. Finally, the aim of the two proposals is different. While 8+8 introduces new address semantics to achieve a number of goals, the purpose of 4+4 is address space extension with minimal changes to address semantics. Analysis of 8+8 can be found in [18].

An alternative approach proposed for IPng is Nimrod [13], which also separates host identifiers from routing locators. Routing locators are hierarchically organized based on provider-customer relationships to allow natural prefix aggregation. The PIP proposal [7] is similar in separating identifiers and locators. The locator used by PIP is a list of values that can be thought of as locally significant addresses at a given level of the topology. The list effectively specifies a source route. Hosts may learn parts of the locator from the configuration, incoming packets, and the directory. PIP eventually merged into SIPP and its locator semantics are reflected mostly in SIPP's cluster addressing. We note that the idea of using a list of fields for addressing has already been considered during the design of IP [1]. The primary reason was for address extension, but again the idea was abandoned as the final 32-bit address seemed large enough. 4+4 is similar to PIP, cluster addressing and [1] in using a two-level address as a locator. However, 4+4 is different because it is incremental to the existing Internet addressing and protocol. See [9] for a detailed discussion of addressing issues.

In contrast to the IPng proposals, a number of other ideas are built on the reuse of the existing 32-bit address space. The separation of private and public address space was first proposed in [4] and then in [11]. Address translation and NATs emerged as a way to connect private networks to the global Internet. Realm Specific IP [28] starts from private address realms and NATs. It provides an explicit way for hosts in private address realms to obtain a public address when there is a need to contact a peer in a public address realm. Such hosts would then tunnel their traffic destined to a public peer through a private realm to the NAT. The NAT, in turn, de-capsulates the traffic and forwards it to the public Internet. The drawback of realm specific IP is that while it does restore network layer transparency it does not extend the address space, and as such, can only be considered a temporary fix to the problem.

Robert Hinden proposed a "medium term" routing and addressing scheme [12] that also uses tunneling as its main tool. Border routers of Autonomous Systems (AS) encapsulate egress traffic and put the source and destination AS numbers into the outer IP header. A block of IP addresses is set aside to identify ASes in such headers. These addresses are injected into the interior routing of transit ASes so only border routers need to recognize them as being special. When the packet reaches its target AS, the ingress border router de-capsulates the traffic and forwards it into the AS. If IP addresses are not globally unique then Hinden suggests the use of an AS address/host address pair as an extended address and adds that "this could be the basis for the long term solution". This idea is very similar to 4+4, but 4+4 addresses have no AS significance. Also the swapping of the two parts is new and central in 4+4 routing. The use of tunneling as a tool to solve the NAT problem is also mentioned briefly in [21, sec. 5.2.1.3] as something that "has never been fully developed, although is fully compatible with end-to-end addressing". 4+4 aims to fill that gap.

The IETF also investigated the use of IP options for address extension, as suggested by Brian Carpenter [5]. This idea was abandoned and never implemented mostly because it is based on IP options and hence requires changes to ARP, DNS, SNMP, and routers within a site. 4+4 is similar to this idea, as well, but uses tunneling instead of IP options to carry the extended addressing information.

The ideas discussed above naturally point toward NAT-extended architectures. The recent IP Next Layer (IPNL) [27] proposal is an early example of a NAT-extended architecture. IPNL uses existing IP address realms as "links" to an overlay (or next layer) protocol. The new protocol is tunneled in IP packets and operates

below the transport layer. Enhanced NAT boxes, called *nl-routers*, route packets realm by realm toward the destination. IPNL introduces *realm numbers* to identify different realms behind the same *frontdoor* - an nl-router that connects private realms to the public Internet. The public IPv4 address of a frontdoor concatenated with a realm number identifies a private realm. The (frontdoor address, realm number, host address) triplet, called IPNL address, fully identifies a host in a private realm. However, IPNL addresses are not used as long-term host identifiers, only as locators that can change frequently. One reason for change might be a switch to a new frontdoor (e.g., when the old one fails). IPNL uses fully qualified domain names (FQDNs) as host identifiers. Nl-routers are able to route packets based on both FQDNs and IPNL addresses. During the initial packet exchanges peers use FQDNs in packet headers to address each other. At the end of such an exchange peers learn both their own and each other's IPNL addresses, and uses those addresses for subsequent communications. If a connection toward two peers exits through different frontdoors then a host may learn two or more different addresses for itself. Hosts are not aware of all their possible addresses. This is not necessary, as the FQDN can be used in packets to identify the host. If the IPNL address changes during a session, then peers can automatically detect this and switch to using the new address. To facilitate such routing a new routing protocol is installed among nl-routers behind the same frontdoor that distributes DNS and realm number information among nl-routers. Nl-routers are also aware of the FQDNs and host addresses of all hosts in the realms they are attached to. This enables them to resolve FQDNs to IPv4 addresses for incoming FQDN-addressed packets. Nl-routers are also capable of performing realm number translation to allow two realms behind the same frontdoor to use the same realm number.

While we believe that IPNL is much easier to deploy than IPv6, IPNL is a fairly complex architecture that represents a significant departure from the current routing and addressing paradigm. Hosts are no longer aware of their full network layer addresses, only their names. Host identifiers are DNS names once and for all. Moreover, the routing infrastructure becomes irrevocably intermingled with the DNS. Domain names are intentionally aggregate administratively and not topologically. A distinction between the current public address realm and private address realms is maintained in the architecture. It is not clear if, or how, this distinction can be removed later and the Internet made homogeneous again. Besides the more important architectural impact, IPNL routing raises some performance issues too. Nl-routers need to manage DNS related routing information and a per-host database. Per-packet processing is complicated for some packets. The packet header is also quite large, 44 bytes for intra-realm packets and 60 bytes for global packets, plus the size of FQDNs if used.

TRIAD [26] is also a new Internet architecture that builds on the existence of IPv4 address realms and uses names as identifiers. Its primary focus is content distribution. TRIAD introduces a *content layer* and *Content Routers*. Content routers hold DNS information and forward combined name lookups/connection setup requests toward destinations. The result of this lookup represents transport connection information and a list of relay identifiers that specify a path through several consecutive address realms. A shim header is added to IP packets that contains the list of relays, enabling *relays* at the border of address realms to forward packets. IP addresses become locally significant with names representing globally unique identifiers. In comparison, 4+4 proposes simple changes to the network layer focusing entirely on address extension while retaining existing semantics as far as possible.

3. 4+4 ARCHITECTURE

The primary goal of the 4+4 address extension is to provide nodes currently in private address realms with an end-to-end address. The 4+4 extended address is formed by concatenating two 32-bit IPv4 addresses: a public and a private one. The public address selects the address realm, while the private address selects the node inside the realm. In fact, the public part is the address of the NAT connecting the realm to the public Internet. Nodes in the public Internet use their existing address as the public part and 0.0.0.0 as the private part.

4+4 packets are minimally encapsulated IP packets [14]. They contain two 32-bit fields for each of the source and destination addresses. The two parts of the extended address are placed into the two 32-bit address fields. The fields are managed such that the outer header always contains addresses that are understood by the IPv4 routers in the realm the packet is transiting; that is, in a private realm the private half of the extended address is visible in the outer header, while in the public Internet the public half is visible. This ensures that routers can forward packets toward the destination without understanding 4+4.

In what follows, we describe addressing and the header format, while routing is discussed in Section 4.

3.1 Network Model

We define an address realm as a collection of networks using addresses from the same address block, while using one address only once; that is, an IP address unambiguously identifies an interface within an address realm. We further differentiate the one and only public address realm that uses the public IPv4 address space and several private address realms, each of which may re-use the private address space designated by [11]. It is also possible for a private address realm to use a block of IP addresses not belonging to [11], if none of those addresses are used anywhere in the public address realm. This allows a private realm to maintain any existing addressing when transiting to 4+4. For the sake of clarity in this paper, we use the term "private" for realms and addresses outside the public realm. Both the public and private realms contain the usual TCP/IP networks with routers and hosts. Today, private realms connect to the public realm via NATs.

Figure 1 shows a typical network scenario assumed by 4+4. The "grey colored" networks belong to the public address realm and each "white colored" network represents a separate private realm. For example, networks 1 and 3 represent realm ***A*** and realm ***B***, respectively, and both can re-use the entire private address space.

Upgraded NATs, called *realm gateways*, represent an integral part of the 4+4 architecture. In addition to the address translation function, realm gateways perform a few simple operations on 4+4 packets. Note that the use of these address translation functions will diminish as transition to 4+4 progresses and be fully removed once full deployment is complete. Realm gateways also act as legacy routers that forward IPv4 packets with public destination addresses. Realms may be interconnected using arbitrary topology and an arbitrary number of realm gateways. The only requirement is that each interface of a realm gateway must strictly belong to either a private or the public realm to separate the address spaces. Note that the public address realm need not be contiguous.

In Figure 1 private realm ***A*** and ***B*** are connected to the public network 2 via realm gateways with public addresses **A** and **B**, respectively. These addresses are separate from the address pool assigned for the address translation function. The figure also shows four hosts, two in the public realm (nodes **C** and **D**) and two in private address realms (nodes **X** and **Y**).

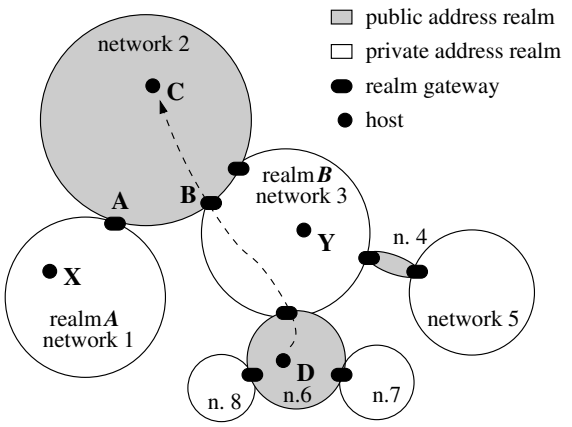

Figure 1: Example networks with arbitrarily interconnected realms.

IPv4 addresses in the public realm are called *level 1* addresses (addresses **A**, **B**, **C** and **D** in Figure 1), while addresses in private realms are called *level 2* addresses (**X** and **Y**). In the remainder of this paper bold capitals are used to denote 32-bit IPv4 addresses and the nodes having those addresses. The term *router* always denotes a legacy IPv4 router that is unaware of 4+4.

The routers inside private address realms are configured with the routing information for both private and public addresses; that is, they know how to route toward both level 1 and level 2 destinations. For example, assume that node **D** in network 6 posts a packet to node **C** in network 2, as illustrated in Figure 1. The packet uses public source and destination addresses and is delivered unaltered to the destination through the routers and realm gateways of network 3. The routers in the public address realm, on the other hand, are configured to route toward public addresses only. This means that realm gateways must filter out routing information of private addresses when communicating with routers in the public address realms. Note that realm boundaries do not have to coincide with autonomous system boundaries.

3.2 Addressing

The 4+4 address of a node inside a private realm is a concatenation of two IPv4 addresses: the public address of a realm gateway and one of the node's own private addresses. We denote this as **A.X**, where **A** represents the realm gateway and **X** represents the node's own address. The first and second parts of the address are called *level 1* and *level 2* parts, respectively. 4+4 nodes may have multiple addresses if the host has multiple interfaces or the realm is 'multihomed' (i.e., there are multiple realm gateways with different addresses). Any (level 1 part, level 2 part) combination constitutes a valid address of a node. Multiple addresses of the same node are treated the same way as in IPv4, that is, nodes accept packets on all of their addresses and may use any of their addresses as the source address for outgoing packets. Transport layer semantics remain unchanged where sockets are bound to a tuple of two 4+4 addresses and two port numbers.

The DNS is used to store and retrieve 4+4 addresses in conceptually the same manner as with IPv4. There are two possible alternative ways to store 4+4 addresses in the DNS. First, a new record type could be defined. Second, two type A records could be used to store the level 1 and level 2 parts of the 4+4 address. The two records would be accessible through a prepended domain name, similar to SRV records [20]. For example, the level 1 and 2 parts of the 4+4 address of the machine `foo.bar.edu` could be stored under `_l1.foo.bar.edu` and `_l2.foo.bar.edu`. The benefit of the latter approach is that it requires no modification to DNS servers. Our implementation, which is discussed later in this paper, uses this approach.

A host may have multiple level 1 and level 2 address parts. This is similar to a host having multiple IPv4 addresses today. The level 2 address of a host inside a private realm is also advertised as a legacy IPv4 address to allow IPv4 communications inside a realm. Similarly, the level 1 address of a host in the public Internet is also available as an IPv4 address. Reverse DNS can be provided by prepending the reverse of the level 2 address part to the usual d.c.b.a.in-addr.arpa DNS name.

One important feature of 4+4 is the isolation of routing and addressing between various realms. The administrator of one private realm may choose arbitrary routing and addressing plans inside the realm without affecting other realms. Realms can also be extended without involving any globally coordinated address allocation process. The isolation also permits the migration of a private realm (e.g., because of a change of provider) without changing the addresses and communication inside the realm. Naturally, if a private realm changes the public address of its realm gateway, the 4+4 address of all the hosts inside will change. This, however, does not affect IPv4 routing and the traffic inside the domain. Provider change affects only the level 1 part of the 4+4 address and the old level 1 part can co-exist with the new one for extended periods of time. During such a coexistence new sessions with external hosts can start with the new level 1 part allowing smooth migration. We note that such isolation also exists when using NATs. Besides easy address expansion, this is another important feature of NAT that makes it a popularity technology.

Finally, we note that more than two levels of address parts is possible, if the amount of address extension provided by the two levels is not sufficient. Details can be found in [31] that address this issue.

3.3 Header Syntax

The 4+4 header is shown in Table 1. On the one hand it can be viewed as an IPv4 encapsulated IPv4 packet with a syntax similar to minimal IP-in-IP encapsulation [14]. This is how legacy IPv4 routers view the packet; they process only the IPv4 header part leading to backward compatibility. On the other hand, the full 4+4 header can be viewed as a new network protocol header with 64-bit source and destination addresses. This is how 4+4 capable endhosts view it.

The first three rows of fields in the 4+4 header are interpreted in the same manner as the IPv4 header, with the exception that the `Protocol 1` field is set to a value that specifies 4+4 encapsulation. Currently this value is 233. The `Source Address 1` and `2` fields collectively contain the full 4+4 address of the source node, while the `Destination Address 1` and `2` fields con-

Ver.	Hlen	DS byte	Total Length	
Identification			Flag	Fragment offset
TTL		Protocol 1	Header Checksum 1	
Source Address 1				
Destination Address 1				
Source Address 2				
Destination Address 2				
Protocol 2		SPos	DPos	Header Checksum 2

Table 1: The 4+4 header

tain the full destination address.

The `SPos` and `DPos` fields indicate how the source and destination 4+4 addresses are partitioned into the two 32-bit fields. A value of 0 means that the address is *unswapped*, that is, the level 1 address part is in the outer header and the level 2 is in the inner header. A value of 1 means that the two address parts are exchanged and the address is *swapped*.

The `Protocol 2` field indicates the transport protocol, while the `Header Checksum 2` covers the end-to-end information in the 4+4 header. This includes the addresses, the `Protocol 2` field, and the payload length, which is the total length minus the IP header length.

Similar to IPv6, only end hosts may fragment IP packets. This is accomplished by setting the `Don't Fragment` bit in the IPv4 header and using path MTU discovery [3]. The fragmentation related fields of the IPv4 header are used exactly as in IPv4. All fragments contain the inner header as well. Reassembly is performed at the final destination only.

A system using extension headers similar to IPv6 can be defined for 4+4. This would allow the reuse of several mechanisms defined for IPv6. Fragmentation can also be made part of the extension header mechanism leaving the second row of the header mostly unused. Due to the change in addressing, the TCP and UDP pseudo-headers also change to include the full 64-bit address in calculating checksums when sending 4+4 packets.

We note that an alternative way of storing the 4+4 address information in packets would be the use of a new IP option. Legacy routers would only need to transparently forward this option unchanged. However, it may seriously impact performance caused by slow-path processing in many routers for all 4+4 packets. In addition, packets with unknown IP options are often dropped in the Internet. As a results of these issues, we did not pursue this alternative further.

4. ROUTING

One of the benefits of 4+4 is that IPv4 only nodes can essentially communicate as today. They use IPv4 packets, the existing IPv4 routers and if they are in different address realms then they use NATs. Therefore, no new transition mechanisms are needed to provide service to legacy IPv4 hosts.

Four different scenarios are possible with regards to the relative location of two IPv4 only nodes. If the two nodes are located in the same private realm, the private IPv4 addresses and the IPv4 protocol are used. If both nodes reside in the public address realm, then they can use their public addresses and the IPv4 protocol to communicate, even when the nodes are separated by one or more private realms. This is possible because the routers inside the private realms have public address routing information and are capable of forwarding packets with public addresses. If one of the IPv4 nodes is in a private realm and the other node is in the public Internet, then address translation is performed as it is done today using a NAT. In this case, the known problems of NATs apply. These limitations can be resolved by upgrading the nodes to 4+4. Finally, if both nodes are in different private address realms then it is impossible for them to communicate unless they upgrade to 4+4. In what follows, we describe, how two 4+4 nodes in different private address realms can communicate with each other.

4.1 Routing between two Private Realms

Assume that a node **X** wishes to send a packet to node **Y**, as illustrated in Figure 2 and shown with solid arrows. Assume further, that both nodes are 4+4 aware, that is, their operating systems can send and receive 4+4 packets. First, node **X** checks if any of the level 1 address parts returned by DNS for node **Y** match any of its own level 1 address parts. If that is the case then the two nodes are in the same address realm and the node uses IPv4 packets to communicate. If this is not the case then **X** selects one level 1 and level 2 part from the set of address parts of node **Y** to form the destination address (e.g., **B.Y**). A source address is also selected (e.g., **A.X**).

Next, the source node creates a 4+4 header and fills in the source and destination address fields as follows. The level 1 part of the source address is placed in `Source Address 2` and the level 2 part in `Source Address 1`. The level 1 and level 2 parts of the destination address are placed in `Destination Address 1` and `2`, respectively. In other words the source address in the packet is swapped, while the destination address is unswapped. This packet header is denoted symbolically as,

$$\begin{matrix}\mathbf{X} & \\ \mathbf{A} & \rightarrow\end{matrix}\ \begin{matrix}\mathbf{B} \\ \mathbf{Y}\end{matrix}$$

where the upper row represents the addresses in the outer IPv4 header (source address is **X**, destination address is **B**) and the lower row represents the addresses in the inner header (source address is **A**, destination address is **Y**). The full 4+4 source address **A.X** is in the left column and is swapped. The full 4+4 destination address **B.Y** is in the right column and is unswapped. The IPv4 routers in realm ***A*** only see $\mathbf{X} \rightarrow \mathbf{B}$.

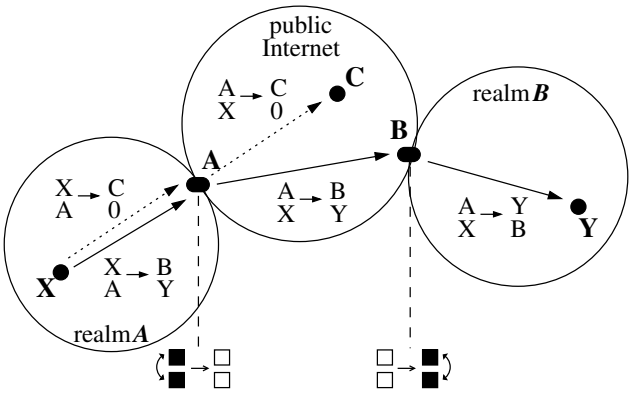

Figure 2: Routing between two 4+4 nodes in different realms.

As a result, the routers will forward the packet toward node **B**, as shown in Figure 2. When it reaches the border of realm ***A***, the realm gateway exchanges the content of the fields `Source Address 1` and `2` leaving the source address unswapped. This is depicted visually at the bottom of Figure 2. Following this the packet is forwarded into the level 1 (public) Internet. At this point the addresses in the packet are as shown in Figure 2. The IPv4 routers see only $\mathbf{A} \rightarrow \mathbf{B}$ and continue forwarding the packet toward node **B**. We note that in the case of a 'multihomed' realm, it may be a realm gateway other than **A** that executes the change. When node **B** receives the packet and sees that it is a 4+4 packet, then it swaps `Destination Address 1` and `2`. The outer header contains $\mathbf{A} \rightarrow \mathbf{Y}$ allowing the routers inside realm ***B*** to forward the packet to node **Y**. When node **Y** receives the packet, it recognizes itself in the full destination address. If it is required to send a response, it gets the complete 4+4 address of the sender from the packet. The response packet is routed across realm boundaries in the same way as the forwarded packet. Realm gateways **B** and **A** will swap source and destination addresses, respectively. The addressing fields of the packet will be,

$$\begin{matrix}\mathbf{Y}\\ \mathbf{B}\end{matrix} \rightarrow \begin{matrix}\mathbf{A}\\ \mathbf{X}\end{matrix}, \quad \begin{matrix}\mathbf{B}\\ \mathbf{Y}\end{matrix} \rightarrow \begin{matrix}\mathbf{A}\\ \mathbf{X}\end{matrix}, \quad \begin{matrix}\mathbf{B}\\ \mathbf{Y}\end{matrix} \rightarrow \begin{matrix}\mathbf{X}\\ \mathbf{A}\end{matrix}$$

as the packet travels through realm ***B***, the level 1 realm and realm ***A***, respectively.

This swapping procedure ensures that private IPv4 addresses are never used in the outer header outside the private realm they belong to. Therefore, IPv4 routers in both private and public realms only see the addresses on which they have routing information. When realm gateways swap source or destination addresses, they also set the `SPos` or `DPos` fields accordingly. Realm gateways also decrement the `TTL` field and recalculate the `Header Checksum 1`. In this manner TTL scoping remains unchanged with each router and realm gateway counting as one hop.

4.2 Routing between Public and Private Realms

In what follows, we describe routing between a node in the public Internet and a node in a private address realm (e.g., node **C** and node **X** in Figure 2).

If both nodes are 4+4 capable then they can use the full 4+4 header to communicate. The 4+4 address of node **C** is **C.0**. Here **0** refers to the all-zero IP address 0.0.0.0. The realm gateway performs exactly the same address swapping operation as described in the previous section. The addresses in a packet sent from **X** to **C** are shown in Figure 2 next to the dotted arrows. On the return path, the fields are the same with the source and destination exchanged. If any of the nodes is not 4+4 capable, for example when no 4+4 address is available from the DNS, then IPv4 and traditional address translation is used. Note that here we use an existing mechanism (i.e., address translation) as a transition tool to enable communication between upgraded and non-upgraded hosts in different realms.

4.3 ICMP Message Routing

ICMP messages provide important error feedback, control, and debugging functions that are an integral part of the Internet Protocol suite. ICMP messages are used in 4+4 conceptually the same way as in IPv4. The current definition of the ICMP protocol is used unchanged, although it is possible to restructure the ICMP protocol as done in the case of IPv6. ICMP messages generated by 4+4 end-hosts, such as Echo or Port Unreachable are addressed and routed just like any other 4+4 packet. End-hosts include the full 4+4 header plus 8 bytes of the original packet. This allows for the inclusion of protocol and port numbers.

Some ICMP messages generated by routers in response to packets not addressed to them require special attention from realm gateways and 4+4 hosts. These ICMP messages include Redirect, Host and Network Unreachable, Fragmentation Required, Time Exceeded, Parameter Problem and Source Quench messages. Other messages, such as Router Discovery messages or Echo and Echo Reply are either always sent inside subnets and thus are not affected by 4+4 or are end-to-end messages. Since routers along a path may only be IPv4 routers, the ICMP messages may not be sent to the original source, but to the outer IPv4 source address of the packet. The following paragraphs discuss this issue in more detail.

Assume that node **A.X** sent a packet (packet p) to node **B.Y**, but the packet cannot be delivered and router **R** generates an ICMP message in response. If **R** is in realm ***A*** then the outer source address field of p contains the level 2 address of the source node **X**. In this case, the ICMP message will reach the source node without any special treatment. The source node is able to recognize that the ICMP message is sent in response to an 4+4 packet by looking at the ICMP payload.

If the router **R** is in the public address realm, then the ICMP message will be sent to the realm gateway **A**. This realm gateway determines that the packet included in the ICMP message is a 4+4 packet and converts the IP header of the ICMP message to 4+4. The destination address will be **A.X** (swapped) copied from packet p included in the ICMP message. The source address will be **R.0**. This allows the original sender to identify the router that generated the ICMP message.

If the router is in realm ***B***,or in any private address realm different from ***A***, then the ICMP message will be routed toward the realm gateway **A**. However, because the ICMP packet is an IPv4 packet containing a private source address, it needs address translation and will be captured by the realm gateway at the realm border, (i.e., **B** in our case). Recognizing the ICMP message as a response to a 4+4 packet, the realm gateway converts the ICMP message header into 4+4, with the destination address **A.X** unswapped and source address **B.R** unswapped and forwards it into the public address realm. This packet is then routed to node **A.X** as a regular 4+4 packet.

4.4 Routing Configuration

Realm operators may use their own addressing plan inside a private realm. Nodes need not renumber their level 2 address parts when the level 1 address parts of the realm changes, e.g., the realm switches providers or a realm gateway is added or removed. It is also possible to partition a private realm by separating the two networks and changing the level 1 parts of the two partition differently.

Multihoming private realms may be configured in several ways. One extreme is to assign a different IPv4 address to each realm gateway. Any of these addresses can be used as the level 1 part of the 4+4 address for hosts inside the realm. In this case, the hosts need to be configured with all or some of the possible level 1 address parts. By selecting the level 1 address part, nodes can effectively select the ingress realm gateway for the traffic addressed to them. An additional benefit of such a realm gateway configuration is that the address of realm gateways can be easily aggregated in

the core of the Internet. This, however, comes at the expense of resilience. If a realm gateway or its provider fails, no other realm gateway can take over.

To overcome this problem, realm gateways can advertise and use the address of another gateway in addition to their own. If one realm gateway fails, traffic addressed to it will be rerouted to another realm gateway advertising its address. The extreme case is when all realm gateways are configured with the same address, resulting in a single possible level 1 address part for the hosts inside. Since realm gateways hold no per-flow state, if one of the realm gateways fails, another one can take over in forwarding the flows for the failed realm gateway. This feature of realm gateways opens the way for a number of further multihoming setups that are outside the scope of our current work.

We note that the situation discussed above is very similar to the issue of multihoming in IPv4 or IPv6. With 4+4, however, the internal addressing of the realm can be hidden from the public Internet. This reduces both the amount of routing information (i.e., the number of prefixes) and the number of changes needed in the core of the Internet. In addition, there is no need to request a new, possibly separate, address block whenever a realm grows beyond its current allocation.

5. TRANSITION TO 4+4

Technically the transition to 4+4 represents a straightforward, stepwise upgrade of NATs and hosts. To upgrade a private access realm at least one of its NATs must be upgraded first to act as a realm gateway. The new functions required include (1) the ability to swap addresses in 4+4 headers; (2) the conversion of ICMPv4 message headers; and (3) the participation in routing and filtering of private addresses. The last function is already part of many NATs today.

Once the private realm has at least one realm gateway, hosts inside the realm can start upgrading. To upgrade a host, its operating system must be augmented with the ability to send and receive 4+4 packets. Auxiliary protocols, such as DHCP, ARP, RARP, router discovery, etc., need not be modified. Similar to IPv6, some applications also need to be upgraded at least to handle larger addresses. Bump-in-the stack address translation [19] developed for IPv6 might be used allowing applications that do not carry IP addresses in payloads to run unchanged.

The DNS itself need not be modified if 4+4 addresses are stored as two type A records, as discussed in Section 3.2. However, if a particular upgraded host needs a domain name its address needs to be included in the DNS zone files and made available to the outside world.

5.1 Transition Incentives

Transition is likely to be started by networks that have an insufficient amount of IPv4 addresses. These may be existing networks using NATs or new networks that find the acquisition of many IP addresses too costly. This is part of the incentives, as transition is directly motivated by the problem the 4+4 architecture aims to solve, i.e., address depletion.

Upgraded hosts immediately gain access to all other 4+4 nodes globally regardless of location. This immediately enables several new applications, such as file sharing or peer gaming that are complicated today because many hosts and therefore users are behind NATs. Upgraded hosts use IPv4 inside a realm and IPv4 plus NATs outside a realm to communicate with non-upgraded hosts, just as they did before the upgrade. This means that hosts can be upgraded one-at-a-time without impacting other hosts. Also, transition builds on the popularity of NATs by using a similar network setup.

As the number of upgraded hosts increases in private realms, hosts in the public address realm also have a growing incentive to upgrade to 4+4. They can do so at any time individually. As a result, upgraded nodes gain access to hosts in other already established private 4+4 domains, (e.g., to run peer-to-peer applications). At this point IP transparency between such hosts is accomplished. End-to-end transparency is restored in the Internet to the extent of 4+4 deployment. When, if ever, deployment becomes complete then IP address transparency will also be fully restored. Note that this would be accomplished without replacing or even reconfiguring routers. Backbone operators, for example, may remain completely unaffected. Of course, if a router has not been upgraded, its control plane cannot be reached from outside the address realm, (e.g., for management purposes). But because routers are usually managed from within the same domain this problem may not be serious. In addition, a control software upgrade of the router would solve this problem.

As the number of hosts reachable via 4+4 increases, organizations that had no NATs before may see the benefits of setting up their own address realms. By doing so, they have the opportunity to install new equipment without obtaining more public IP addresses. The existing nodes need not be renumbered; public addresses may remain to be used inside the realm even for communicating with the outside world. In addition, the routing information of the realm may be hidden from the outside world. The address translation function of realm gateways is required only for communicating with IPv4 only hosts. As transition progresses it will be invoked less frequently and can be completely removed once the majority of the nodes have transitioned. This way 4+4 provides a way out of using NATs.

One benefit of the above transition process is that the upgrade of individual realms is de-coupled and may happen at different paces. The most complex transition tool is the NAT itself. This provides communication between hosts if no native IPv4 or 4+4 path is available. There is no transition mechanism to allow communication between certain hosts (e.g., IPv4 only hosts in different private address realms). However, such a possible lack of reachability would not discourage starting the transition, as it is already in place today. In contrast, it provides an incentive as transition provides the missing reachability.

5.2 4+4 and IPv6

The major benefits of 4+4 over IPv6 are its backwards compatibility, the ease of transitioning, and the isolation of realms. Due to the backward compatibility of the packet header and addressing, the transition can be gradually started, gradually evolving to the new 4+4 architecture. There is no need for temporary transition mechanisms (such as tunneling, tunnel brokers, 6to4, 6over4 or DSTM, as discussed in [29]), all new mechanisms are final. There is no need for a new addressing plan, dual routing, new network management tools, new routing protocols, or new routers. The transition requires little new software and minimal changes to a running network. Full backward compatibility is maintained even when all hosts have transitioned. 4+4 and IPv4 only hosts/networks can coexist without new overhead. 4+4 immediately provides a large address space for realms without introducing new routers and a new

protocol. This may be beneficial for larger organizations where most of the traffic is local.

On the other hand, many of 4+4 features (e.g., the header format) include a number of design compromises necessary for backward compatibility. Therefore, in comparison to the IPv6, 4+4's design is not based on a clean slate. The extended address space is substantially smaller than that of IPv6. Applications placing IP addresses in payloads also need to be modified if they are to be used between realms. This, however, is unavoidable if the address space is truly extended, since the number of possible destinations may not fit into 32-bits. If operators and users choose to undergo the IPv6 transition, 4+4 is not needed. However, if IPv4 and NATs prevail, 4+4 provides a plausible solution to the known problems discussed in this paper.

6. IMPLEMENTATION

We implemented 4+4 under the Linux operating system using kernel version 2.4.18. The 4+4 source code is publicly available from the web [32] for experimentation. One of the key goals driving the 4+4 implementation is a minimal impact implementation on the kernel and applications. Hence, no modification has been made to the kernel itself. All the 4+4 functionality is provided in the form of a kernel module and an accompanying user space daemon that can be loaded and unloaded to/from a running kernel. As a result of our minimum impact implementation we sacrificed some performance. The implementation includes both the end-host and the realm gateway functionality. It does not contain NAT functions itself but interworks with the standard Linux NAT. The kernel module and userspace daemon comprise roughly 2200 and 1200 lines of C code, respectively. In what follows, we describe our implementation.

6.1 Peer Identifiers

To implement socket network programming with a new address space, the obvious solution is to define a new address family as is the case with most IPv6 implementations. This, however, requires the revision and porting of existing networking code to the new address family. In addition, applications need to be modified. To achieve backward compatibility with existing applications and to minimize the implementation work, we adopted a different strategy. The end-host functions are implemented using a transparent protocol translation mechanism similar to [19]. 4+4 addresses are mapped to 32-bit *peer identifiers* that are of local significance only and are taken from a yet unused block of the IPv4 address space. By default the block 1.0.0.0/8 is used. Our implementation transparently translates between IPv4 packets with peer identifiers and 4+4 packets. Applications are only presented with peer identifiers. The mapping between peer ids and 4+4 addresses is established by incoming 4+4 packets and DNS queries. These mappings are automatically timeout if they remain unused by incoming or outgoing packets for a period of time. In addition to the translation function, an API is defined and implemented that provides functions to establish, query, and remove peer mappings. In this manner, the full functionality is available to 4+4 aware applications without compromising backward compatibility.

The 4+4 kernel module is configured with the list of level 1 and level 2 addresses of the node. Configuration is automated: if an interface has a private address it is considered level 2 by default, and level 1 otherwise. If a node has no level 2 address then `0.0.0.0` is used assuming that it is in the public Internet. If a node has no level 1 address then the DNS is queried for the hostname to obtain the level 1 address. The kernel module operates using the Netfilter architecture of the Linux 2.4 kernel [30]. More information on the 4+4 implementation can be found on the project webpage [32].

6.2 DNS Translation

To maintain compatibility with the deployed DNS infrastructure, 4+4 addresses are stored as discussed in Section 3.2. The 4+4 kernel module intercepts incoming DNS reply messages from type A queries if the reply contains no valid answer. Such packets are passed to the 4+4 userspace daemon, which prepends the existing domain name with "`11.`" and "`12.`" and then performs two type A queries on the revised names. If both the queries are successful, then a new peer id is allocated to the 4+4 address. This peer id is then placed in the original DNS reply packet, which is then passed back to the kernel and from there on to the querying application. As a result, if a host has a 4+4 address then a querying application will receive a host address that is a peer identifier corresponding to the 4+4 address of the host.

Reverse DNS queries are also captured and translated in this manner. For example, a query to `12.0.0.1.in-addr.arpa` is translated into a query to `2.0.168.192.131.67.59.128.in-addr.arpa`. Similarly, replies are translated back. As a result, applications have total DNS transparency, (e.g., ping and tcpdump are able to show the DNS names for 4+4 hosts of which they only know the peer identifiers).

6.3 ICMP Translation

Certain ICMP messages may carry IP packets in their payload. Upon receiving such ICMP packets, the kernel module checks if the packet in the payload is a 4+4 packet. If this is the case, then the 4+4 header is removed and the source and destination addresses are translated to peer identifiers. This allows the kernel to match ICMP error messages to the sockets associated with peer identifiers. If the host is sending an ICMP message in reply to another packet, the kernel module checks whether the included packet's source or destination address are peer identifiers. If so, then a 4+4 header is added to the included packet with the appropriate 4+4 addresses. This also ensures that peer identifiers are never exposed on the wire at least for ICMP.

One problem with ICMP messages generated by routers is that the ICMP protocol [2] mandates the inclusion of the original IPv4 header including options plus 8 bytes of the payload only. In case of a legacy IPv4 router generating an ICMP reply in response to a 4+4 packet, this excludes the transport protocol and port information. As a consequence, the source host cannot identify the transport connection for which it received an ICMP message. Some router implementations (e.g., Linux) return more than 8 bytes, however. To upgrade most routers to do this would of course solve this problem. This problem, however, is not as serious as it first seems because most ICMP messages generated by routers correspond to all transport sessions at the destination host. With this in mind, our current implementation delivers the signal to all relevant transport identities, that is, to all sockets with the same source and destination addresses. In other cases, such as when the TTL is exceeded or when a parameter problem message is generated this action is not meaningful, however, sockets usually ignore these messages. The fragmentation needed ICMP message needs special handling because they are part of the Path MTU discovery process [3]. In response to this the kernel module decreases the reported MTU size in fragmentation needed ICMP packets sent in response to 4+4

packets. This forces the transport to generate smaller packets that fit the path requirements even for 4+4 packets. The kernel in a more integrated 4+4 implementation could directly take the size of the 4+4 header into account when calculating TCP segment sizes.

6.4 Realm Gateway Operation

If the kernel module determines that a host has interfaces with both private and public addresses then it will start operating as a realm gateway, assuming that this mode is not disabled. The Linux kernel routing features and NAT implementation are used to provide the actual packet routing and NAT functionality, respectively. 4+4 packets are allowed through the NAT unmodified. If the private realm is not a "stub" realm, then additional rules are needed to protect packets with public source and destination addresses from being translated by the NAT. In the realm gateway mode, the kernel module detects 4+4 packets crossing the realm boundary and performs address switching. ICMP packets are handled as discussed in Section 4.3.

6.5 Configuration Tasks

The configuration of an end-host is essentially the same as an IPv4 host. There are only two additional tasks that need to be considered. First, the host must know if it is in a private or public realm. Second, if it is in a private realm, then it needs to be configured with the level 1 part(s) of its address. The first piece of information can reliably be estimated from the IP address of the host. The second part can be read from the DNS because the host is able to communicate using IPv4 even without its full 4+4 address. Configuring the DNS is straightforward to do with only the two type A records needing to be added to represent the 4+4 address.

Configuring realm gateways requires somewhat more effort. First, a fully functional NAT must be maintained during the transition period. Second, it needs to specified which interfaces of the realm gateway belong to public or private realms, and what are the level 1 address parts of that realm. Third, routing and route filtering must be set up such that routing information on public IPv4 addresses are propagated both in and out of the gateway, while routes toward private addresses are filtered out. This can be achieved using the current routing tools, either by using a different routing protocol in the two realms (e.g., BGP/OSPF or OSPF/RIP) or using OSPF with different areas.

7. EXPERIMENTAL RESULTS

In this section, we discuss our results from the evaluation of our 4+4 implementation using local and wide area Internet experiments. We first discuss some experiences with the resilience of our approach. In particular, we demonstrate 4+4's resilience to realm gateway failure. Following this, we present some performance results of our implementation. Finally, we discuss a set of experiments and results that test application compatibility with using 4+4.

7.1 Resilience

To test the robustness of 4+4 to realm gateway failure, we constructed the topology shown in Figure 3. The topology contains a public and private realm, two routers (`R1` and `R2`), two realm gateways (`RGW1` and `RGW2`) and two end-systems (`taygeta` and `galaxy`). The routers are legacy, unmodified IPv4 routers. We used the gated-3.6 routing software in the routers and gateways. Two OSPF routing areas are configured, the backbone consisted of `R1`, `RGW1` and `RGW2`, whereas an additional OSPF stub area consisted of `RGW1`, `RGW2` and `R2`. Networks in the stub area are private addresses and are filtered by the realm gateways. Realm gateways also advertised an extra address `128.59.67.213` that is exclusively used as the level 1 part of all 4+4 addresses in the private realm.

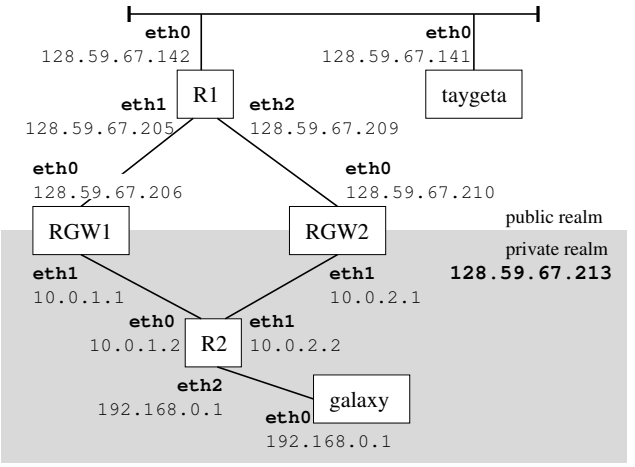

Figure 3: Test network 1 topology (multihoming private realm)

We started a long TCP data transfer from `taygeta` to `galaxy` using the `ttcp` utility. The initial routing preferred `RGW1` in both directions, a traceroute listing of the path from `tagyeta` to `galaxy` is shown in Table 2. The listing is generated using a modified version of the traceroute utility, as discussed in more in detail in Section 7.3.

At around two seconds after the start of the TCP connection we disconnected both cables of `RGW1`. The routing software did not receive a link-layer disconnection signal, so the failure can only be detected by missing Hello packets. Using the defaults of `gated`, the connecting routers declared `RGW1` down after 40 seconds. At this point the routers establish new routes and the TCP connection was resumed soon after. Since realm gateways hold no flow specific state, nothing prevented the session from continuing once routing stabilized. Apart from the temporary loss of connectivity, end systems were not affected.

7.2 Performance

Realm gateways have two 4+4 specific packet processing tasks. First, they swap the source and destination addresses in 4+4 packets forwarded between different realms. Second, they add a 4+4 header to certain ICMP packets.

To illustrate the costs associated with packet processing tasks, we performed a series of measurements in the Linux kernel. Figure 4 shows the results. The measurements are performed using an unloaded machine with one 1GHz Pentium III processor and 256 megabytes of memory.

Each group of bars in Figure 4 corresponds to a packet type. The height of the bar shows the processing time of the packet in the network layer. This is the time between passing the PRE_ROUTING and POST_ROUTING Netfilter hooks. For IP packets the time includes routing table lookup (usually a cached value), TTL decrement, IP option processing and fragmentation; none of the last two functions were exercised in our experiments. The first bar shows

```
traceroute4+4 from taygeta.ipv44.comet.columbia.edu (128.59.67.141.0.0.0.0)
1:  r1-eth0.comet.columbia.edu (128.59.67.142):  0.308ms 0.262ms 0.159ms
2:  rgw1-eth0.comet.columbia.edu (128.59.67.206):  0.274ms 0.264ms 0.196ms
3:  r2-eth0.ipv44.comet.columbia.edu (128.59.67.213.10.0.1.2):  0.365ms 0.611ms 0.343ms
4:  galaxy.ipv44.comet.columbia.edu (128.59.67.213.192.168.0.2):  0.445ms 0.630ms 0.370ms
```

```
traceroute4+4 from taygeta.ipv44.comet.columbia.edu (128.59.67.141.0.0.0.0)
1:  r1-eth0.comet.columbia.edu (128.59.67.142):  0.284ms 2.582ms 0.214ms
2:  rgw2-eth0.comet.columbia.edu (128.59.67.210):  0.414ms 0.292ms 0.241ms
3:  r2-eth1.ipv44.comet.columbia.edu (128.59.67.213.10.0.2.2):  0.437ms 0.669ms 0.321ms
4:  galaxy.ipv44.comet.columbia.edu (128.59.67.213.192.168.0.2):  0.444ms 0.676ms 0.385ms
```

Table 2: Traceroutes before and after the topology change

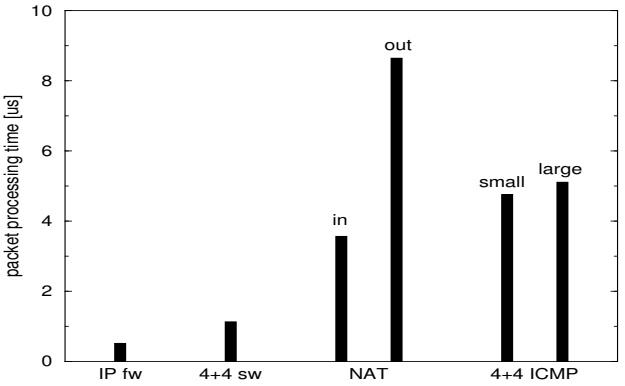

Figure 4: Network layer processing time during forwarding

the time for regular IP packet forwarding averaged over 1000 measurements. The second bar shows the forwarding time of a 4+4 packet, including the address swap operation. The third group shows the time of an address translation operation for packets entering and leaving the private realm, respectively. The time difference may be due to connection state management where packets leaving the private realm may establish entries. ICMP echo requests/replies are used to take the measurements. The last four bars show the processing time of router-generated ICMP messages that carry a 4+4 packet inside. In this case the realm gateway needs to insert a 4+4 header into the packet. In the Linux kernel, this usually involves a memory copy of the packet due to linear packet buffers. This explains the time difference between small (84-byte) and large (1428-byte) packets. We note that ICMP packets generated by end-systems do not fall into this category.

Although the above figures may certainly be different for different router platforms, we argue that 4+4 packet forwarding (i.e., the swap operation) is a simple operation that requires a small and constant number of steps. We believe it is amenable to hardware implementation in the fast path of routers. ICMP masquerading, on the other hand, is an operation that requires more changes to the packets and may be too expensive to implement in hardware. However, this poses no problem as ICMP processing can and should be rate limited. Realm gateways are free to drop excess ICMP traffic.

7.3 Applications

We experimented with a number of applications to test interoperability with 4+4. In general, applications and protocols that do not carry IP addresses in the packet payload work well with 4+4. The testbed used to experiment with Internet applications is a simplified version of the one shown in Figure 3 with parts of the network in New York in Budapest (see Figure 5) and separated by 17 hops.

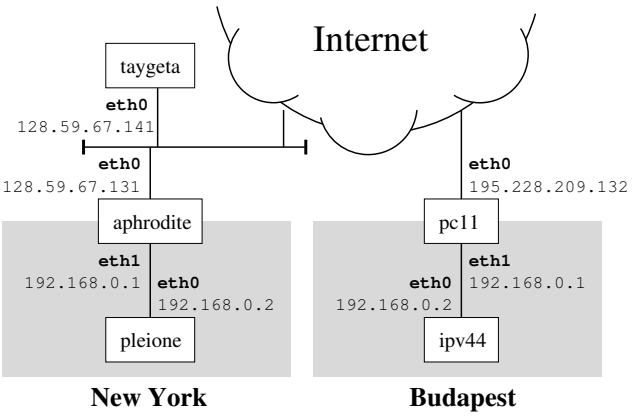

Figure 5: Test network 2 topology (wide area)

Hypertext Transfer Protocol (HTTP). The HTTP was tested by setting up a 4+4 compatible webserver both in a private and the public realm. In both cases we used the `apache` webserver. 4+4 aware clients were able to reach both webservers from anywhere using popular browsers including Netscape, Mozilla, Opera, Lynx. Webservers are reachable by specifying DNS names that point to 4+4 addresses.

Email Protocols. The SMTP, IMAP and POP3 protocols were tested by setting up e-mail forwarders in both realms using the `exim` utility. The popular `pine` e-mail program and the Netscape's mailer were used as user agents. The e-mail servers were specified using domain names. Again, 4+4 clients were able to download and send e-mail even if the 4+4 aware server was at a different realm.

Secure Tools. The `ssh` and `scp` tools were used on a regular basis to communicate between the machines. The host database of these tools can get mixed up if peer identifiers are used directly to identify targets hence the use of domain names is preferred.

File Transfer Protocol (FTP). FTP did not work as expected because it places the peer identifier in the protocol payload.

We also tested a number of network tools. The **`ping` utility** works unmodified. The only issue is that it displays peer identifiers in-

stead of full 4+4 addresses when pinging a 4+4 machine (e.g., "`PING pleione (1.0.0.2) from ...`").

The **`traceroute` utility** does not work, as it uses raw sockets and manipulatess UDP ports directly. The port information is usually not returned in the ICMP messages with 4+4. See Section 6.3 for a detailed explanation of the reasons for this problem. Therefore, we created a simple version of `traceroute` that only uses UDP sockets and does not code sequence number information into the port numbers. The only drawback is that two traceroutes performed on the same host at the same time toward the same destination by two different processes may get mixed up. The benefit is that since no raw sockets are used no root privileges are needed. In addition, as the 4+4 module passes incoming ICMP messages to all relevant sockets, the utility works well with 4+4 as well. We added a small piece of code to display the 4+4 numeric address, if the IP address seen is a peer identifier. Reverse DNS, however, was used unmodified. The new version of `traceroute` can be downloaded from [32] and was used to generate the listings shown in Table 2.

The **`tcpdump` utility** works well, but cannot decode 4+4 packets. To this end, we have written a small plugin to the `ethereal` utility to dissect 4+4 packets. The plugin is part of the 4+4 source code package that can be downloaded from [32].

8. CONCLUSION

In this paper, we have presented and evaluated 4+4, a new address extension architecture for Internet. 4+4 leverages existing private address realms and NATs, and represents an evolutionary approach toward Internet address extension. 4+4 offers a lightweight, well defined, incentive-driven transition process that can be incrementally deployed in the network today. Upgraded hosts immediately gain access to upgraded hosts in all other realms, while existing communication is not disrupted in any way. 4+4 is simple and retains the existing semantics of Internet names and addresses. Encapsulation is used as the main tool to maintain backward compatibility with existing routers that need not be modified. 4+4 does not employ address translation and provides end-to-end address transparency. Existing NATs are only used as a transition tool, with their use diminishing as 4+4 deployment progresses. In fact, removing NATs is one of the motivations for such a transition. However, the address isolation feature of NATs is retained by the 4+4 architecture.

We have evaluated the properties and performance of 4+4 based on local and wide area testbed experimentation, and discussed our experiences using a number of applications with 4+4. A number of configurations and possible pitfalls were explored and discussed. We found that 4+4 is easy to implement, scalable, introduces no single points of failure, and its performance look very promising. We believe that 4+4 provides one alternative should IPv6 be deemed too expensive or complicated for transitioning.

9. REFERENCES

[1] J. Postel, "Extensible Field Addressing," *Internet RFC 730,* May 1977.

[2] J. Postel, "Internet Control Message Protocol," *Internet RFC 792,* September 1981.

[3] J. Mogul, S. Deering, "Path MTU discovery," *Internet RFC 1191,* November 1990.

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

[5] Minutes of the Address Extension by IP Option Usage BOF, *proceedings of 29th IETF meeting,* Seattle, April 1994.

[6] Minutes of the Address Lifetime Expectations working group, *proceedings of 29th IETF meeting,* Seattle, April 1994.

[7] P. Francis, "Pip Near-term Architecture," *Internet RFC 1621,* May 1994.

[8] K. Egevang, P. Francis, "The IP Network Address Translator (NAT)", *Internet RFC 1631,* May 1994.

[9] P. Francis "Addressing in Internetwork Protocols," PhD Thesis, *University College London,* available at `www.ingrid.org/francis/thesis.ps.gz`, September 1994.

[10] R. Hinden, "Simple Internet Protocol Plus White Paper," *Internet RFC 1710,* October 1994.

[11] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. de Groot, E. Lear, "Address Allocation for Private Internets," *Internet RFC 1918,* February 1996.

[12] R. Hinden, "New Scheme for Internet Routing and Addressing (ENCAPS) for IPNG," *Internet RFC 1955,* June 1996.

[13] I. Castineyra, N. Chiappa, M. Steenstrup, "The Nimrod Routing Architecture," *Internet RFC 1992,* August 1996.

[14] C. Perkins, "Minimal Encapsulation within IP," *Internet RFC 2004,* October 1996.

[15] M. O'Dell, "8+8 – An Alternate Addressing Architecture for IPv6," *Internet Draft,* named as draft-odell-8+8-00, Work in progress, November 1996.

[16] S. Deering, R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," *Internet RFC 2460,* December 1998.

[17] Z. Turányi, A. Valkó, "4+4: Expanding the Internet Address Space without IPv6," *Ericsson Internal Report,* August 1999.

[18] M. Crawford, A. Mankin, T. Narten, J. Stewart, L. Zhang, "Separating Identifiers and Locators in Addresses: An Analysis of the GSE Proposal for IPv6," *Internet Draft,* named as draft-ietf-ipngwg-esd-analysis-05, Work in progress, October 1999.

[19] K. Tsuchiya, H. Higuchi, Y. Atarashi, "Dual Stack Hosts using the "Bump-In-the-Stack" Technique (BIS)," *Internet RFC 2767,* February 2000.

[20] A. Gulbrandsen, P. Vixie, L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)," *Internet RFC 2782,* February 2000.

[21] B. Carpenter, "Internet Transparency," *Internet RFC 2775,* February 2000.

[22] M. Crawford, "Router Renumbering for IPv6," *Internet RFC 2894,* August 2000.

[23] T. Hain, “Architectural Implications of NAT,” *Internet RFC 2993,* November 2000.

[24] G. Huston, “To NAT or IPv6 – That is the question,” *Satellite BroadBand magazine,* available at the author’s page `http://www.telstra.net/gih`, December 2000.

[25] M. Holdrege, P. Srisuresh, “Protocol Complications with the IP Network Address Translator,” *Internet RFC 3027,* January 2001.

[26] M. Gritter, D. R. Cheriton, “An Architecture for Content Routing Support in the Internet,” *Usenix Symposium on Internet Technologies and Systems,* `http://gregorio.stanford.edu/triad`, March 2001.

[27] P. Francis, R. Gummadi, “IPNL: A NAT-Extended Internet Architecture,” *SIGCOMM’01,* August 2001.

[28] M. Borella, J. Lo, D. Grabelsky, G. Montenegro, “Realm Specific IP: Framework,” *Internet RFC 3102,* October 2001.

[29] The IETF Next Generation Transition (ngtrans) working group, `http://www.ietf.org`

[30] Linux 2.4.x Netfilter homepage, `http://www.netfilter.org`

[31] Z. Turányi, A. Valkó, “4+4,” *10th International Conference on Networking Protocols (ICNP 2002),* November 2002.

[32] The IP4+4 project webpage at `http://comet.columbia.edu/ipv44`