# Design, Implementation and Evaluation of IPv4+4[*]

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

## ABSTRACT

In this paper we present and evaluate the 4+4 architecture. 4+4 extends the IPv4 address space without requiring changes to existing routers. It builds on the existence of NATs and multiple address realms, but it does not use address translation and provides end-to-end address transparency. Existing address translation is used only as a transition tool. The paper also presents an implementation of 4+4 and related experimental results. We conclude that 4+4 is simple to introduce and may represent a medium-term solution if IPv6 transition does not take off quickly enough. The source code of our implementation can be downloaded from http://ipv44.comet.columbia.edu.

## 1. INTRODUCTION

One of the greatest 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 around 2008 [6]. Since that time, the Internet has experienced the introduction of Network Address Translators (NAT) [17]. NATs have helped reduce the rate of address depletion, enabling continued operation even in regions with shortages in IP addresses. NATs, however, have also contributed to the loss of IP transparency [21]. The negative impact of NATs on the architecture and operation of the Internet is discussed in the literature [24, 26]. In addition to the architectural impact, NATs also have a number of practical drawbacks. For example, NATs prevent IP level end-to-end security, reduce robustness, break a number of application level protocols, complicate the network and inhibit some novel uses of the Internet (e.g., such as peer-to-peer networking). Despite such disadvantages NATs are becoming popular and are widely deployed. Although NATs adversely impact the global Internet, in many cases they represent an attractive alternative for individual networks. First, if the acquisition of public IP addresses is hard, complicated or expensive, setting up a NAT is easy, quick and cheap. Second, for most end users a "NAT-ed" connection seems good enough. Third, alternative solutions seems too complicated at present.

In 1994 the IETF selected IPv6 [15] 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 [30] developed several transition mechanisms that would allow the temporary co-existence of IPv4 and IPv6 and the communication in mixed environments. However, despite availability of IPv6 and transition procedures, IPv6 has seen little practical deployment. One factor that prevents the rapid deployment of IPv6 is the complexity associated with transition. If one cannot replace all routers and hosts in a site at once, then a (possibly lengthy) period of co-existence follows. During such a period network administrators must manage both IP versions plus a number of transition mechanisms. Individual networks or users may be hesitant to undergo such change preferring instead to live with the limitations of NATs. While transition to IPv6 would benefit the whole Internet at the expense of extra work at individual networks, using NATs benefits the individual networks directly with negative side-effects on the global Internet. This dilemma between NATs and IPv6 is nicely described in [25].

Transition to IPv6 may last for a long time in the global Internet. In fact, it is possible that it will never be completed with a significant portion of the Internet remaining IPv4 only. Such a situation could occur if a certain population (e.g., the cellular industry or regions with shortages of IP addresses) saw sufficient incentives to transition to IPv6 while others remained content with IPv4. This could result in a lengthy partitioning of the Internet with intensive use of protocol translation and tunneling mechanisms, a result that may be even worse than the present situation with NATs.

In short, it is possible that IPv4 and NATs will remain and become permanent. If this is the case, then a markedly different approach is necessary to solve the address extension and transparency problems. One such alternative approach is represented by *NAT extended architectures* [28], which rely on the existence of multiple address realms and extend the address space by requiring changes only to hosts and NAT devices.

The 4+4 architecture [16] presented in this paper is one example of such a NAT-extended architecture. It provides a way back to unique, global, network layer host addresses. End-to-end transparency is restored in the Internet to the extent of 4+4 deployment. If deployment becomes complete then IP address transparency will also be fully restored. There is no need to change routers. The transition process provides incentives for networks with both private or 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 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 related to 4+4. Section 3 and Section 4 describe the architecture and operation of IPv4+4, respectively. The transition process with a brief comparision to IPv6 is outlined in Section 5. A minimal impact implementation is described in Section 7. The resilience, performance and application compatibility of 4+4 is analysed through experimentation with local and wide area 4+4 testbeds in Section 8. Finally, Section 9 presents our 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) [9], that 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 address space quite similar 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 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 [14]. In the 8+8 scheme half of the 128-bit IPv6 address is used as locator (termed "routing goop") and the other half as 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 fundamental differences exist. First, in 4+4 no part of the address is used as a location independent host identifier. Second, border routers of 4+4 do not rewrite addresses[1]. Third, IPv4+4 hosts are aware of their full addresses. Finally, the aim of the two proposals are 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. An analysis of 8+8 can be found in [18].

An alternative approach proposed for IPng is Nimrod [12], 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 kind of source route. Hosts may learn parts of it from the configuration, incoming packets and the directory. Later, PIP has been merged into SIPP. The idea of using a list of fields for addressing has already been considered during the design of IP [1]. The primary reason is for address extensions, but the final 32-bit address seemed large enough. See [8] for a detailed discussion on addressing.

Contrary to IPng proposals, a number of other ideas build on the reuse of the existing 32-bit address space. The separation of private and public address space was first proposed in [4] then in [10]. Address translation and NATs emerged as a way to connect private networks to the global Internet.

Realm Specific IP [29] 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 the public address realm. Such hosts would then tunnel their traffic destined to the public peer through the private realm to the NAT. The NAT, in turn, de-capsulates such 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.

Robert Hinden proposed a "medium term" routing and addressing scheme [11] 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 are 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 Hinden suggests the use of an (AS-address, host address) pair as an extended address adding that "this could be the basis for the long term solution." The idea is very similar to 4+4. 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." The IETF also investigated the use of IP options for address extension, as suggested by Brian Carpenter [5]. The idea is quite similar to 4+4, but beacause it is based on IP options it requires changes to ARP, DNS, SNMP and routers within a site. This idea has been abandoned and never implemented.

The ideas discussed above point toward NAT-extended architectures. The recent IP Next Layer (IPNL) [28] proposal is one such 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 is 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* [28] — an nl-router that connects private realms to the public Internet. The public IPv4 address of the frontdoor concatenated with the realm number identifies the 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 initial packet exchanges peers use FQDNs in packet headers to address each other. At the end of the exchange they learn both their own and each other's IPNL address and use

[1] Although realm gateways associated with 4+4 rearrange addresses when forwarding packets, no new addressing information is added to the packet.

that address for subsequent communication. 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, the peers automatically detect it 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 address, only their name. Host identifiers are DNS names once and for all. Moreover, the routing infrastructure becomes irrevocably intermingled with the DNS. Besides architectural impact this may raise 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. A distinction between the current public address realm and the private address realms are maintained in the architecture. It is not clear if, or how, this distinction can be removed and the Internet made homogeneous again.

Finally, TRIAD [27] 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* (CR). CRs hold DNS information and forward combined name lookups/connection setup requests toward destinations. The result of this lookup is 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 and enable *relays* at the border of address realms to forward packets. IP addresses become locally significant with names representing globally unique identifiers. In comparision, 4+4 proposes simple changes to the network layer focusing entirely on address extension, while retaining existing semantics as far as possible.

## 3. THE 4+4 ARCHITECTURE

The basic idea of 4+4 is as follows. The primary goal of address extension is to provide nodes currently in private address realms with an end-to-end address. The 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.

IPv4+4 packets are minimally encapsulated IP packets [13]. They contain two 32-bit fields for each of the source and destination address. 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 traveling in. 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 there. This ensures that routers can forward the packet toward the destination without understanding IPv4+4.

The following sections 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 unambigously 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 [10]. Both the public and private realms contain usual TCP/IP networks with routers and hosts. Today, private realms connect to the public realm via NATs.

Figure 1 shows the scenario assumed by IPv4+4. Grey networks belong to the public address realm and each white 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*, are integral part of the 4+4 architecture. In addition to the address translation function[2], realm gateways perform a few simple operations on IPv4+4 packets. 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 realm gateways 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 continuous.

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[3]. 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**).

IPv4 addresses in the public realm are termed as *level 1* addresses (addresses **A**, **B**, **C** and **D** in Figure 1), while addresses in private realms are termed as *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 IPv4+4.

The routers inside private address realms are configured with the routing information about 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. The packet uses public source and destination addresses and will be 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 con-

[2]The use of the address translation function will diminish as transition progresses and it can be fully removed if deployment is complete.

[3]These addresses are separate from the address pool assigned for the address translation function.

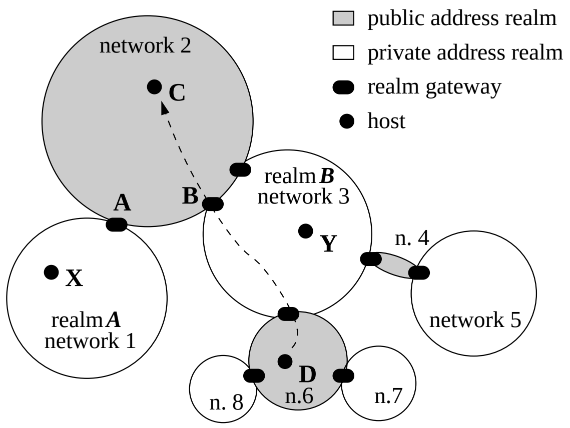


**Figure 1: Example networks with arbitrarily interconnected realms.**

figured to route toward public addresses only. This means that realm gateways must filter out routing information of private addresses when talking to routers in the public address realms. Note that realm boundaries do not have to coincide with autonomous system boundaries.

## 3.2 Addressing

The IPv4+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 termed *level 1* and *level 2* parts, respectively. IPv4+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 source address in outgoing packets. Transport layer semantics are unchanged, sockets are bound to a tuple of two IPv4+4 addresses and two port numbers.

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

A host may have multiple of both 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 communication inside the realm. Likewise, 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.

## 3.3 Header Syntax

The IPv4+4 header is shown in Table 1. On one hand it can be viewed as and IPv4 encapsulated IPv4 packet with a syntax similar to minimal IP-in-IP encapsulation [13]. 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 end-hosts view it.

| | | | | |
|---|---|---|---|---|
| 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 IPv4+4 header**

The first three rows of fields in the IPv4+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 IPv4+4 encapsulation[4]. The `Source Address 1` and `2` fields collectively contain the full IPv4+4 address of the source node, while the `Destination Address 1` and `2` fields contain the full destination address.

The `SPos` and `DPos` fields indicate how the source and destination IPv4+4 addresses are partitioned to 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*. Later in the paper more `SPos` and `DPos` values will be introduced.

The `Protocol 2` field indicates the protocol above IP (e.g., TCP), while the `Header Checksum 2` covers the end-to-end information in the IPv4+4 header[5].

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 IPv4+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.

We note that an alternative way of storing 4+4 addressing information in packets would be the use of a new IP option. Legacy routers would only need to transparently pass this option unchanged. How-

[4]Currently this value is 233

[5]This includes the addresses, the `Protocol 2` field and the payload length, which is the total length minus the IP header length.

ever, it may seriously impact performance by cause slow-path processing in many routers for all 4+4 packets. In addition, packets with (unknown) IP options are often dropped in the Internet. Therefore we did not pursue this alternative any further.

## 4. ROUTING

One of the benefits of IPv4+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 old 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 if they 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 is in the public Internet, then address translation is performed as today using a NAT. In this case the known problems of NATs apply. These limitations can be resolved by upgrading the nodes to IPv4+4. Finally, if both nodes are in different private address realms then it is impossible for them to communicate unless they upgrade to IPv4+4. In what follows, we describe, how two IPv4+4 nodes in different address realms 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 (solid arrows). Assume further, that both nodes are IPv4+4 aware, that is, their operating system is prepared to send and receive IPv4+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 use 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 this packet is swapped, while the destination address is unswapped. This packet header is denoted symbolically as

$$\begin{matrix} \mathbf{X} & \multirow{2}{*}{$\rightarrow$} & \mathbf{B} \\ \mathbf{A} & & \mathbf{Y} \end{matrix}$$

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 $\mathbf{X} \rightarrow \mathbf{B}$.

As a result, the routers will forward the packet toward node **B**. (See Figure 2.) When it reaches the border of realm ***A***, the realm gateway exchanges the content of the fields `Source Address 1` and `2`

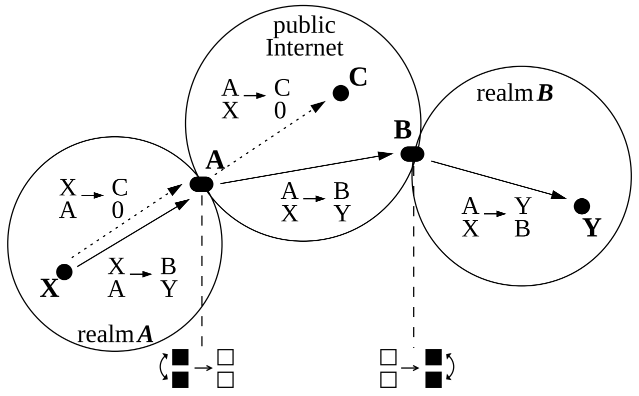


**Figure 2: Routing between two IPv4+4 nodes in different realms.**

making the source address unswapped. (This is depicted visually at the bottom of Figure 2.) Then 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 ‘multihoming’ 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 an IPv4+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 finds the complete IPv4+4 address of the sender in the packet. The return 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

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

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

The above swapping procedure ensures that private IPv4 addresses are never used in the outer header outside the private realm they belong to. Therefore, IPv4 routers of both private and public realms see only such addresses on which they have routing information.

When realm gateways swap source or destination addresses, they also set the `SPos` or `DPos` fields appropriately. In addition, they decrement the `TTL` field and recalculate `Header Checksum 1`. This way TTL scoping works as currently, 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** as illustrated in Figure 2).

If both nodes are IPv4+4 capable, then they can use the full IPv4+4 header to communicate. The IPv4+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 as described in the previous section. The addresses in a packet sent from **X** to **C** are shown in Figure 2 beside the dotted arrows. On the way back, the fields are the same with the source and destination exchanged.

If any of the nodes is not IPv4+4 capable, then IPv4 and traditional address translation is used. Observe that here an already existing mechanism (i.e., address translation) is used 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 IPv4+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 was done in the case of IPv6).

ICMP messages generated by IPv4+4 end-hosts, such as Echo or Port Unreachable are addressed and routed just like any other IPv4+4 packet. End-hosts include the full IPv4+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, however, require special attention from realm gateways and IPv4+4 hosts[6]. Since routers along the way may be IPv4 only 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 possibility more in detail.

Assume that node **A.X** sent a packet (packet $p$) to node **B.Y** but the packet can not 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, that is **X**. In this case the ICMP message will reach the source node without special treatment. The source node is able to recognize that the ICMP message was sent in response to an IPv4+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 an IPv4+4 packet and converts the IP header of the ICMP message to IPv4+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 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 (**B** in our case). Recognizing the ICMP message as a response to an IPv4+4 packet, the realm gateway converts the ICMP message header into IPv4+4, with 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 IPv4+4 packet.

[6]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 IPv4+4 or are end-to-end messages.

## 5. TRANSITION TO IPV4+4

Transition to IPv4+4 represents a straightforward, stepwise upgrade of NATs, hosts and optionally routers.

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 are (1) the ability to swap addresses in IPv4+4 headers; (2) the conversion of ICMPv4 message headers; and (3) the participation in routing and filtering of private addresses. The latter function is already part of many NATs today. If a realm is concerned about the realm gateway as a single point of failure, it shall set up two realm gateways with the same address. Since realm gateways hold no per-flow state, if one fails, the other can take over. If the realm wish to do robust multihoming to multiple ISPs, the same realm gateway address have to be advertised to each ISP. Although such a construct is known to increase core routing table sizes, lacking alternatives it is commonly used today.

After the private realm has 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 IPv4+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 (see Section 3.2). The address of the upgraded hosts, however, needs to be included in the DNS zone files and made available to the outside world.

Optionally, routers can also be upgraded at least in certain regions to understand the 64-bit 4+4 addresses. This requires modifications to routing protocols and the forwarding engine.

### 5.1 Transition Incentives

Transition will probably be started by networks that have 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 new architecture aims to solve, i.e., address depletion.

Upgraded hosts immediately gain access to all other IPv4+4 nodes globally regardless of location. Upgraded hosts use IPv4 inside a realm and IPv4+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.

As the number of upgraded hosts increases in private realms, hosts in the public address realm also have growing incentive to upgrade to IPv4+4. They can do so at any time individually. As a result they gain access to hosts behind NATs, 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 IPv4+4 deployment. When (if ever) deployment becomes complete then IP address transparency will also be fully restored. Note that this is accomplished without replacing or even reconfiguring routers. Backbone operators, for example, may remain completely unaffected[7].

As the number of hosts reachable via IPv4+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 IPv4+4 provides a way out of NATs.

If routers are also upgraded in an area, the special role of the realm gateways diminishes and they simply reduce to ordinary routers. In addition, the borders of address realms "dissolve" and routers in a private realm become no different from routers in the public address realm. In fact, the notion of address realm also disappears. IPv4+4 becomes "classless" by softening up the strict distinction between level 1 and 2 parts of the address. If the transition goes on in this direction then eventually we get back to a totally transparent Internet where all routers are equal and where the address space is 64-bits. We note, however, that there may be no need ever to upgrade routers, if the operator community decides so. Also, selective regions of the network may decide to upgrade independently.

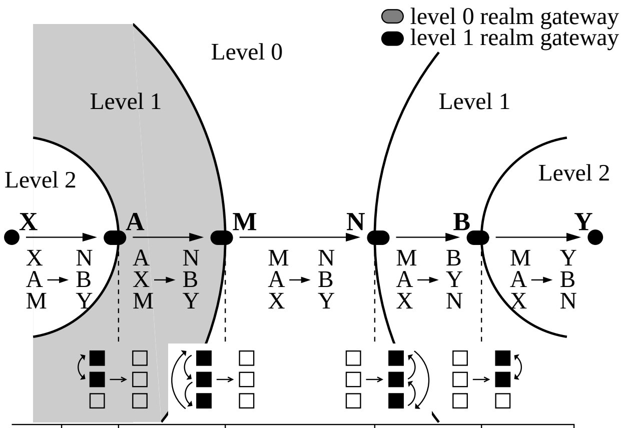
One benefit of the above transition process is that the upgrade of hosts and routers are de-coupled and may happen at different pace — or, in case of routers may not happen at all. The most complex transition tool is the NAT itself. This provides communication between hosts if no native IPv4 or IPv4+4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

## 5.2 Comparison to IPv6

The major benefits of IPv4+4 over IPv6 are its incentives, backwards compatibility, and the ease of transition. Due to the backward compatibility of the packet header and addressing, the transition can be gradually started, gradually blending to the new architecture. There is no need for temporary transition mechanisms (such as tunneling, tunnel brokers, 6to4, 6over4 or DSTM; see [30]), 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 already transitioned. IPv4+4 and IPv4 only hosts and networks can co-exit without new overhead. On the negative side, the IPv4+4 architecture is far from being as clean as IPv6. Many of its features (e.g., the header format) include a compromise for backward compatibility. The extended address space is substantially smaller than that of IPv6. If operators and users choose to undergo the IPv6 transition, IPv4+4 is not needed. However, if IPv4 and NATs prevail, IPv4+4 provides a plausible solution to the known problems discussed in this paper.

## 6. FURTHER EXTENSION OF THE ADDRESS SPACE

Currently, approximately a 24-bit address space is designated as private [10]. Since private realms may use only such addresses, the "effective size" of the extended address space is very roughly $32 + 24 = 56$ bits only. Furthermore, this address space is hard to utilize efficiently, as the vast majority of the current public addresses do not have a corresponding private address realm. Also, it is naive to expect private address realms to be as large as to efficiently utilize the entire private address space. Thus we conclude that an effective new address space might be even smaller than 56-bit. This poses questions regarding the expected lifetime of IPv4+4 addresses.



| | | | | | |
|---|---|---|---|---|---|
| Spos | 2 | 3 | 4 | 4 | 4 |
| Dpos | 4 | 4 | 4 | 3 | 2 |

**Figure 3: Further extending the address space.**

To add more addresses another level, level 0 can be introduced. This means a new network, in which the current Internet is merely a "private address realm" connected via one or more *level 0 realm gateways* (like node **M** in Figure 3). The realm gateways connecting level 1 and level 2 become *level 1 realm gateways*. The nodes in the current Internet then would have a number of level 0 address parts and the packet format would be double encapsulation[8].

The level 0 address space has to be disjoint from the addresses used in both level 1 and level 2. In addition, routers at level 1 and level 2 should have routing information on the level 0 addresses. This allows such routers to route toward level 0 realm gateways. The

[7]Of course, if a router has not been upgraded, its control plane cannot be reached from outside the address realm for example for management purposes. But since routers are usually managed from within the same domain this problem may not be serious. In addition, a control software upgrade of the router solves this problem.

[8]If this feature is desirable, then IPv4+4 hosts should be upgraded with preparedness to this scenario from day one.

level 1 realm gateways need to perform the same address swap operations as described in Section 4.1, while level 0 realm gateways need to cycle through the three address parts. The realm gateway operations are illustrated in Figure 3. The state of the addresses are reflected in the `SPos` and `DPos` fields using the values 2, 3 and 4, as shown in Figure 3.

The resulting address space is quite sizable, although not as large as the 128-bit IPv6 address space[9]. If, for example, one class A address block is used as the level 0 address space, then the effective size of the total address space is approximately $24 + 32 + 24 = 80$ bits.

The level 0 extension may likely be introduced by the efforts of network operators. While level 2 realms would most likely be introduced independently by individual sites or organizations, the level 0 infrastructure can be added only as a global effort that requires more co-ordination.

Finally, we note that another direction of address space extension is possible, namely by introducing level 3 inside private address realms, instead of level 0. However, for us it seems that the private address space available is sufficiently large for most private address realms to make such nested realms unnecessary.

## 7. IMPLEMENTATION

We have implemented an early version of IPv4+4 under the Linux 2.4.18 operating system. The IPv4+4 source code is publically availablefrom [33] for experiments. One of the key design goals of the implementation was to have minimal impact on the existing kernel and applications. Hence no modification has been made to the kernel itself. All the 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 price for minimal impact, we sacrificed some performance. The implmentation includes both the end-host and the realm gateway functionality. It does not contain NAT functions itself but interworks with the standard Linux NAT support instead. The kernel module and the userspace daemon comprise roughly 2200 and 1200 lines of C code, respectively. In the following, we describe the operation of the implementation and the issues we faced during experimentation.

### 7.1 Peer Identifiers

To implement socket network programming with a new address space, the obvious solution is to define a new address family as in case of 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 also modified.

To achieve backward compatibility with existing applications and to minimize implementation work, we choose a different strategy. The end-host functions are implemented using a transparent protocol translation mechanism similar to [19]. IPv4+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. The module transparently translates between IPv4 packets with peer identifiers and IPv4+4 packets. Applications are only presented with peer identifiers. Mapping between peer ids and 4+4 addresses are established by incoming IPv4+4 packets and DNS queries. Mappings time out if not used by incoming or outgoing packets for a long time.

[9]We note, however, that 64 bits of the IPv6 address is used solely to address hosts within a subnet. In IPv4 (and 4+4) much less bits are used for that purpose.

In addition to the translation function, an API is defined and implemented. It provides functions to establish, query and remove peer mappings. This way the full functionality is available to 4+4 aware applications without compromising backward compatibility.

The kernel module is configured with the list of level 1 and level 2 addresses of the node. Configuration is highly automated: if an interface has a private address it is considered level 2 by default, level 1 otherwise. If a node has no level 2 address `0.0.0.0` is used assuming that it is in the public Internet. If a node has no level 1 address, the DNS is queried for the hostname (see later) to obtain the level 1 address parts.

The kernel module operates using the Netfilter architecture of the Linux 2.4 kernel [31]. Each locally terminated and originated packet is captured (on the LOCAL_IN and LOCAL_OUT hooks). If the protocol field of an incoming packet is 233 (the value indicating 4+4 encapsulation), the 4+4 header is removed and a peer identifier is put into the source address field. In addition a new mapping is established between the peer identifier and the source 4+4 address if the 4+4 address had no mapping yet. Similarly, if the destination address of outgoing packets is in the range of peer identifiers, then the appropriate mapping is looked up and a 4+4 header is inserted after the IPv4 header. If the packet leaves on an interface that belongs to the public realm, the source address is placed unswapped into the 4+4 header, and swapped otherwise. The destination address is placed swapped if the host and the destination are both in the same private realm, and unswapped otherwise.

Adding and removing a 4+4 header also requires the recalculation of transport layer checksums. A new, 4+4 specific pseudo-header is defined that contains the entire 4+4 source and destination address. At the receiving side this checksum is validated against the 4+4 pseudo-header. If the checksum is found valid it is recalculated using the IPv4 pseudo-header with the appropriate peer identifier as source address. If the checksum is invalid, it is left as invalid, so that the transport layer can catch the checksum error. We note that in a more integrated implementation, the transport layer can be made aware of the peer being reached by 4+4. This eliminates double checksum calculations.

### 7.2 DNS Translation

To maintain compatibility with the deployed DNS infrastructure, 4+4 addresses are stored as two set of A records under names created by prepending "`l1.`" and "`l2.`" to the domain name. For example, the level 1 and level 2 address parts of `foo.bar` is stored under `l1.foo.bar` and `l2.foo.bar`, respectively. Incoming DNS reply messages to type A queries are also stopped by the kernel module if they contain no valid answer. Such packets are passed to the userspace daemon, that prepends the mentioned domain name by "`l1.`" and "`l2.`" and performs two type A queries on the resulting names. If both the queries are succesful, then a new peer id is allocated to the 4+4 address (or an existing is reused) and this peer id is 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 an IPv4+4 address then a querying application will receive host address that is a peer identifier corresponding to the 4+4 address of the host.

Reverse DNS queries are also captured and translated. 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 DNS names for 4+4 hosts of which they know only peer identifiers.

### 7.3 ICMP Translation

Certain ICMP messages may carry IP packets in their payload. Upon receiving such ICMP packets, the kernel module checks if the included packet is a 4+4 packet. If yes, then its 4+4 header is removed and the source and destination addresses are translated to peer identifiers. This will allow the kernel to match ICMP error messages to sockets that are based on peer identifiers. In addition, if the host is sending an ICMP message in reply to another packet, the kernel module checks, if 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 with ICMP).

One problem with ICMP messages carrying generated by routers is that the ICMP protocol [2] mandates the inclusion of the original IPv4 header (including options) plus 8 bytes of payload only. In case of a legacy IPv4 router generating an ICMP reply in response to an IPv4+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[10]. However, this is not as serious as it first seems, as most ICMP messages generated by routers correspond to all transport sessions with the destination host. If the route is congested, the peer is unreachable, or the path MTU has changed then it affects all such transport sessions. In this case our 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 with TTL exceeded or Parameter Problem messages this behaviour is not meaningful, however, sockets usually ignore these messages.

Fragmentation Needed ICMP messages need special handling. These messages usually carry the new MTU value to aid path MTU discovery [3]. This value is used by the kernel and applications to choose outgoing packet sizes. Since the packets generated by the kernel to 4+4 destinations will be augmented by the 4+4 header in the module, the resulting packets will not fit into the reported size. To overcome this problems, the kernel module decreases the reported mtu size in ICMP packets sent in response to 4+4 packets. This practice ensures that path MTU discovery works toward 4+4 destinations as well. Again, in a more integrated implementation the kernel could directly take the size of 4+4 header into account when calculating TCP segment sizes.

### 7.4 Realm Gateway Operation

If the kernel module finds that a host has interfaces with both private and public addresses then it will start operating as a realm gateway (if not disabled). The Linux kernel routing features are used to provide the actual packet routing. The kernel module captures packets on the FORWARD and PRE_ROUTING Netfilter hooks to perform 4+4 specific swapping functions. In addition, realm gateways also operate as 4+4 hosts as described above.

[10]Some router implementations (e.g., Linux) return more than 8 bytes. To upgrade most routers to do so, would completely solve the problem.

The Linux NAT implementation can be used to provide the NAT functionality. In addition, 4+4 packets should be let through the NAT unmodified. This requires a simple addition of rules to the NAT configuration. If the private realm is not a "stub" realm, then additional rules shall protect packets with public source and destination addresses from being translated by the NAT.

Packets being forwarded by the realm gateway are tested on the FORWARD hook to see if they are 4+4 packets coming from a private realm going toward the public realm. If so, then the source 4+4 address is swapped. This ensures that only the level 1 part of the source address is visible to routers in the public address realm.

Incoming packets are tested on the PRE_ROUTING hook to see if they are 4+4 packets with an IPv4 destination address being one of the realm gateway's public addresses. If so, then this is a 4+4 packet entering the private realm. In this case the realm gateway swaps the destination address placing the level 2 part of the destination into the IP header. Then the packet is routed based on this address into the private realm.

The realm gateway also captures all ICMP packets that (a) carry a 4+4 packet inside, (b) are forwarded from one realm to another, and (c) do not have a 4+4 header themselves. These packets are augmented with a 4+4 header before forwarding. The 4+4 source and destination addresses are calculated as described in Section 4.3.

### 7.5 Configuration Tasks

The configuration of an end-host is essentially the same as an IPv4 host. There are only two additional tasks. First, the host must know if it is in a private or in the public realm. Second, if it is in a private realm, it shall 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 can be read from the DNS, since the host is able to communicate using IPv4 even without its full IPv4 address.

Configuring the DNS is easy: only the two type A records need to be added to represent the 4+4 address[11]. Since CNAME records works well in this scheme, the administrator does not have to repeat the level 1 address part(s) of multiple hosts in a private realm. Instead, he can use a canonical name representing all the level 1 address parts of the realm and specify a CNAME record for each individual host pointing to this name.

Configuring realm gateways requires somewhat more tasks. 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 pirvate 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, while routes toward private addresses are filtered out. This can be achieved by current routing tools, by either using a different routing protocol in the two realms (e.g., BGP/OSPF or OSPF/RIP) or using OSPF with different areas.

### 7.6 Renumbering and 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

[11]Other alternatives of storing the 4+4 address are possible. We take this approach as it requires no changes to DNS servers.

switches providers or a realm gateway is added or removed. To implement a smooth change, new external sessions shall use the new level 1 address parts, while existing sessions shall keep the old one. In this case the old level 1 part can be finally removed once all existing session has closed[12]. Note that since the level 2 address parts did not change, communication internal to the realm is totally unaffected and need not be reconfigured (think of e.g., networked file systems). This means, that a private realm can easily switch providers. 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 several ways. One extreme is to assign a different IPv4 address to each realm gateway. Any of those addresses can be used as the level 1 part of the 4+4 address of the hosts inside the realm. In this case, the hosts shall be configured with all or some of the possible level 1 address parts. By selecting the level 1 address part they can effectively select the ingress realm gateway for the traffic addressed to them. An additional benefit of such realm gateway configuration is that the address of realm gateways can be easily aggregateable by 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. An extreme of this 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. However, using such configuration it may not be possible to aggregate the addresses advertised by realm gateways in the core of the Internet.

We note that the situation described above is very similar to the issues 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 (the number of prefixes) and the number of changes. In addition, there is no need to request new address blocks whenever a realm grows.

Finally, we note that ingress filtering [22], a technique used by ISPs to prevent address spoofing may be affected by 4+4. In case of a multihoming domain with more than one level 1 address parts, a realm gateway may emit packets into the public Internet that have the address of another realm gateway as IPv4 source address. This is, however, easy to fix as the set of possible other addresses are both limited and known. By configuring ISP firewalls to allow these as exceptions, ingress filtering can continue to operate.

## 8. EXPERIMENTAL RESULTS

In this section we discuss a number of aspects of 4+4. First, we demonstrate its resilience to Realm Gateway failures then illustrate its performance. Finally, we detail a set of experiments to test application compatibility.

### 8.1 Resilience

In the current routing fabric of the Internet failed routes are detected and repaired. Although the various protocols and techniques result in different convergence times and route stability, the Internet has a certain resilience to network component failure. We expect no less from any new architecture. In the 4+4 architecture realm gateways represent the only possible single point of failure, as other network components use existing routing protcols.

To test the robustness of 4+4 to realm gateway failures, we constructed the topology shown on Figure 4. The topology contains two realms (the public and a private) two routers (`R1` and `R2`), two realm gateways (`RGW1` and `RGW2`) and two end-systems (`taygeta` and `galaxy`). The realm gateways act as routers, NATs and carry out the 4+4 specific functions as well. The routers are legacy, unmodified IPv4 routers. We used the gated-3.6 routing software in all four routers. Two OSPF areas were configured, the backbone consisted of `R1`, `RGW1` and `RGW2`, whereas an additional stub area consisted of `RGW1`, `RGW2` and `R2`. Networks in the stub area were private addresses and were filtered in the realm gateways. Realm gateways also advertised an extra address `128.69.67.213` that was exclusively used as the level 1 part of all 4+4 addresses in the private realm (using AS external routes).

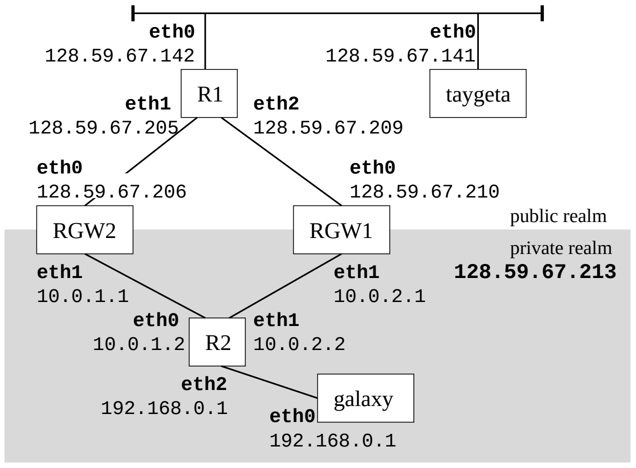


**Figure 4: Test network 1 topology (multihoming private realm)**

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 at Table 2. The listing was generated using a modified version of the traceroute utility (see more in Section 8.3).

At around two seconds after the start of the TCP connection we disconnected both cables of `RGW1`. The routing software did receive a link-layer disconnection signal, so the failure was detected by the missing Hello packets. Using the defaults of `gated` the connecting routers declared `RGW1` dead after 40 seconds. Then they quickly established the new routes and the TCP connection resumed soon after. A trace of the TCP packets is shown in Figure 5 while the second row of Table 2 contains the traceroute after the route change.

### 8.2 Performance

Realm gateways have two 4+4 specific packet processing tasks. First, they swap the source and destination addresses in 4+4 pack-

[12]We did not implement this feature in our host implementation as it requires additional state in sockets. Changing sockets, however, requires changes in the kernel itself and cannot be done as a kernel module. Our 4+4 module needs to be restarted (with existing sessions broken) if the set of level 1 address parts change.

```
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**

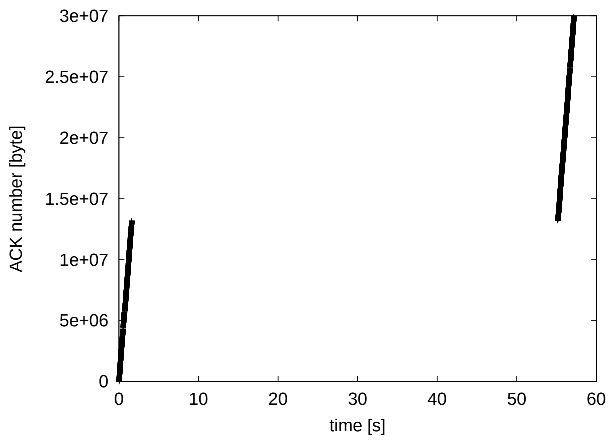


**Figure 5: TCP trace with re-routing.**

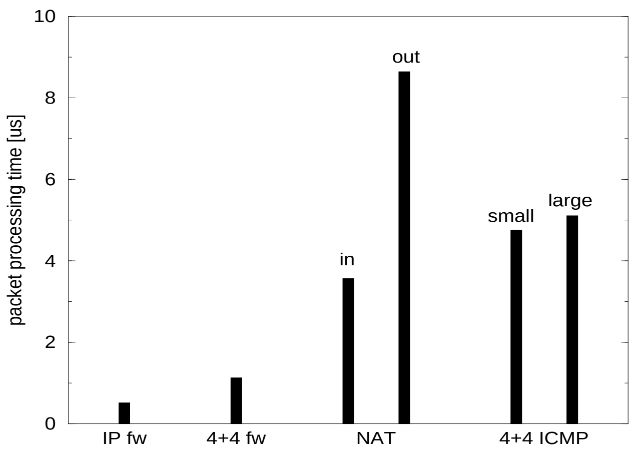


**Figure 6: Network layer processing time during forwarding**

ets forwarded between different realms. Second, they add a 4+4 header to certain ICMP packets.

To illustrate the costs associated with various packet processing tasks, we performed a series of measurements in the Linux kernel. In all experiments, we used starnard Linux forwarding, NAT and our implementation. We measured the time of processing a packet by timestamping it at the PRE_ROUTING hook and checking the time at the POST_ROUTING hook. The time difference between the two gives the time the packet spent in the IP layer including routing lookup, header manipulation, etc. Figure 6 shows the results. The measurements were performed using an unloaded machine with one 1GHz Pentium III processor and 256 megabytes of memory.

Each group of bars on Figure 6 correspond to a packet type. The height of the bar shows the processing time of the packet in the network layer[13]. The first bar shows the time of 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: leaving packets may establish entries[14]. Finally, 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 need 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.

4+4 packet forwarding (t.i., 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 harware. However, this poses no problem as ICMP processing can and should be rate limited. Realm gateways are free to drop excess ICMP traffic.

End-host processing in our implementation is more expensive than realm gateway functionality. Adding a 4+4 header usually requires an additional memory copy of the entire packet. Calculating transport checksums also reuires an extra pass over the packet. These steps, however, are not inherent in the architecture, but are the consequence of our minimum impact implementation strategy. Our experience was that performance of end-systems is not visibly affected even with this implementation.

[13]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 decrementation, IP option processing and fragmentation (none of the last two in our experiments).

[14]ICMP echo requests/replies were used to take the measurements.

## 8.3 Applications

We experimented with a number of applications to test interoperability with 4+4. In this section we report on these experiments. In general, applications and protocols that carry no IP addresses in the payload work well with 4+4. The testbed used to experiment with applications was a simplified version of the one shown on Figure 4 and contained parts both at New York and Budapest (see Figure 7), the two parts were about 17 hops away.

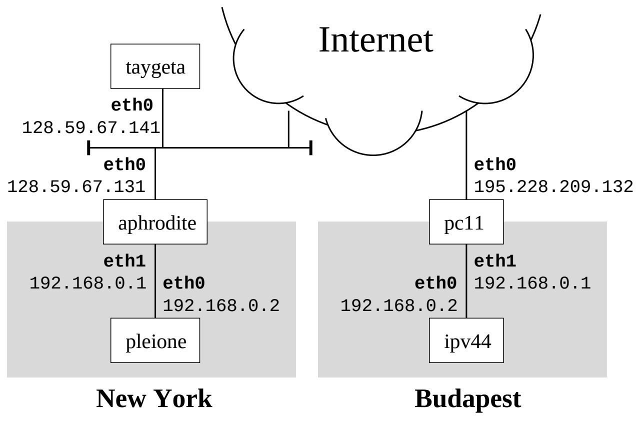


**Figure 7: Test network 2 topology (wide area)**

**Hypertext Transfer Protocol (HTTP)** was tested by setting up a 4+4 compatibile 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 (Netscape, Mozilla, Opera, Lynx). Webservers are reachable by specifying DNS names that point to 4+4 addresses.

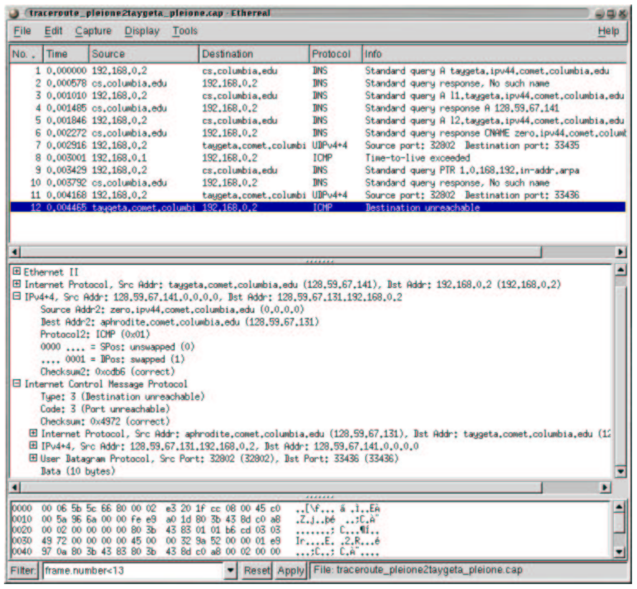
**Email** protcols namely SMTP, IMAP and POP3 were tested by setting up mail forwarders in both realms using the `exim` utility. The popular `pine` mail program and Netscape's mailer were used as user agents. The mail servers were specified using domain names. Again, 4+4 clients were able to download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

**Secure tools (`ssh`, `scp`)** were used on a regular basis to communicate between the machines. The host database of these tools can get mixed up if using peer identifiers to identify targets, but apart from this, they work seamlessly.

The **`ping` utility** works unmodified. The only issue is that it displays peer identifiers instead 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 manipulate UDP ports directly. (The port information is usually not returned in ICMP messages with 4+4. See Section 7.3 for a detailed explanation.) Therefore we created a simple version of `traceroute` that uses only 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 process 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, were used unmodified. This tool was used to generate the listings in Table 2. The tool can be downloaded from [33].

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 (see Figure 8). The plugin is part of the source code package at [33].


traceroute_pleione2taygeta_pleione.cap - Ethereal
File
Edit
Capture
Display
Tools
Help
Filter:
frame.number<13
Reset
Apply
File: traceroute_pleione2taygeta_pleione.cap

```
No. . Time     Source               Destination          Protocol Info
    1 0.000000 192.168.0.2          cs.columbia.edu      DNS      Standard query A taygeta.ipv44.comet.columbia.edu
    2 0.000578 cs.columbia.edu      192.168.0.2          DNS      Standard query response, No such name
    3 0.001010 192.168.0.2          cs.columbia.edu      DNS      Standard query A 11.taygeta.ipv44.comet.columbia.edu
    4 0.001485 cs.columbia.edu      192.168.0.2          DNS      Standard query response A 128.59.67.141
    5 0.001846 192.168.0.2          cs.columbia.edu      DNS      Standard query A 12.taygeta.ipv44.comet.columbia.edu
    6 0.002272 cs.columbia.edu      192.168.0.2          DNS      Standard query response CNAME zero.ipv44.comet.columt
    7 0.002916 192.168.0.2          taygeta.comet.columbi UDPv4+4 Source port: 32802  Destination port: 33435
    8 0.003001 192.168.0.1          192.168.0.2          ICMP     Time-to-live exceeded
    9 0.003429 192.168.0.2          cs.columbia.edu      DNS      Standard query PTR 1.0.168.192.in-addr.arpa
   10 0.003792 cs.columbia.edu      192.168.0.2          DNS      Standard query response, No such name
   11 0.004168 192.168.0.2          taygeta.comet.columbi UDPv4+4 Source port: 32802  Destination port: 33436
   12 0.004465 taygeta.comet.columbi 192.168.0.2         ICMP     Destination unreachable

⊞ Ethernet II
⊞ Internet Protocol, Src Addr: taygeta.comet.columbia.edu (128.59.67.141), Dst Addr: 192.168.0.2 (192.168.0.2)
⊟ IPv4+4, Src Addr: 128.59.67.141.0.0.0.0, Dst Addr: 128.59.67.131.192.168.0.2
    Source Addr2: zero.ipv44.comet.columbia.edu (0.0.0.0)
    Dest Addr2: aphrodite.comet.columbia.edu (128.59.67.131)
    Protocol2: ICMP (0x01)
    0000 .... = SPos: unswapped (0)
    .... 0001 = DPos: swapped (1)
    Checksum2: 0xcdb6 (correct)
⊟ Internet Control Message Protocol
    Type: 3 (Destination unreachable)
    Code: 3 (Port unreachable)
    Checksum: 0x4972 (correct)
  ⊞ Internet Protocol, Src Addr: aphrodite.comet.columbia.edu (128.59.67.131), Dst Addr: taygeta.comet.columbia.edu (12
  ⊞ IPv4+4, Src Addr: 128.59.67.131.192.168.0.2, Dst Addr: 128.59.67.141.0.0.0.0
  ⊞ User Datagram Protocol, Src Port: 32802 (32802), Dst Port: 33436 (33436)
    Data (10 bytes)

0000  00 06 5b 5c 66 80 00 02  e3 20 1f cc 08 00 45 c0   ..[\f... ã .Ì..EÀ
0010  00 5a 96 6a 00 00 fe e9  a0 1d 80 3b 43 8d c0 a8   .Z.j..þé  ..;C.À¨
0020  00 02 00 00 00 00 80 3b  43 83 01 01 b6 cd 03 03   .......; C...¶Í..
0030  49 72 00 00 00 00 45 00  00 32 9a 52 00 00 01 e9   Ir....E. .2.R...é
0040  97 0a 80 3b 43 83 80 3b  43 8d c0 a8 00 02 00 00   ...;C..; C.À¨....
```

**Figure 8: Captured traceroute from pleione to taygeta**

## 9. CONCLUSION

In this paper, we presented and analised 4+4, a new address extension architecture for the Internet. 4+4 relies on existing private address realms and NATs and represents an evolutionary approach to address extension. It provides a lightweight, well defined, incentive-driven transition process and can be incrementally deployed today. Upgraded hosts can immediately gain access to upgraded hosts in all other realms, while existing communication is not influenced in any way. 4+4 is simple, affects only the network layer and retains the existing semantics of 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 utilised only as a transition tool. Their use will diminish as 4+4 deployment progresses. In fact, removing NATs is one of the motivations for transition.

We analysed the properties of 4+4 based on experimenting with our implementation. A wide area testbed has been set-up. A number of configuration alternatives and possible pitfalls were explored. We found that 4+4 is easy to implement, scalable, introduces no single points of failure and its performance implications are acceptable. We believe that 4+4 is a viable alternative of IPv6, in case of the latter is deemed to expensive or complicated to transit to. Key benefits are a rewarding and incentive driven transition model and the ability to keep the exiting routing infrastructure.

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

[9] R. Hinden, “Simple Internet Protocol Plus White Paper,” *Internet RFC 1710,* October 1994.

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

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

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

[13] C. Perkins, “Minimal Encapsulation within IP,” *Internet RFC 2004,* October 1996.

[14] 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.

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

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

[17] Srisuresh, P. and M. Holdrege, “IP Network Address Translator (NAT) Terminology and Considerations,” *Internet RFC 2663,* 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] P. Ferguson, D. Senie, “Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing,” *Internet RFC 2827, Best Current Practice 38,* May 2000.

[23] M. Crawford, “Router Renumbering for IPv6,” *Internet RFC 2894,* August 2000.

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

[25] 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.

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

[27] 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.

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

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

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

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

[32] Z. Turányi, A. Valkó, “IPv4+4,” *10th International Conference on Networking Protocols (ICNP 2002),* November 2002.

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