Pip Overview and Examples

Paul F. Tsuchiya
Bellcore
tsuchiya@thumper.bellcore.com
July, 1992

Status

This document is an Internet Draft. Internet Drafts are working documents of the Internet Engineering Task Force (IETF), its Areas, and its Working Groups. Note that other groups may also distribute working documents as Internet Drafts.

Internet Drafts are draft documents valid for a maximum of six months.Internet Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet Drafts as reference material or to cite them other than as a “working draft” or “work in progress.”

Please check the I-D abstract listing contained in each Internet Draft directory to learn the current status of this or any other Internet Draft.

Differences with Previous Draft

This draft (July 1992) differs from the previous draft (June 1992) only slightly. It changes some terminology. The Logical Router field is now the Routing Context field. The Routing Hints Fields is now the Forwarding Table Index Fields. These trerms better reflect the real mechanistic function of these fields.

I have also cleaned up certain text. I thought about taking the tunnel field out of the protocol (as per several peoples comments), but am still not quite ready to do that, and would like a bit more debate about it. In particular, I think the tunnel field might be very commonly used, especially for default routing, and so am still not comfortable with encapsulation for the common case.

1.0 Purpose of this draft

Pip is an internet protocol that scales efficiently encodes policy. It facilitates many advanced features, such as mobility and multiple defaults routing, and has a well-bounded routing table lookup, and is therefore fast. Pip is proposed as an alternative to the two “medium term” proposals that emerged from the Road (Routing and Addressing) group to deal with the dual IP problems of scaling and address depletion.

This paper presents a high-level overview of the basic concepts of Pip, plus an appendix (which constitutes half the page count of this paper) containing examples that give somewhat more detail and describe features not covered in the main body. This paper does not discuss the dynamic algorithms such as routing and host configuration that are required to make Pip work. One or more companion documents will describe Pip in complete detail. The overview and examples are suggested preliminary reading to the companion documents.

2.0 Pip General

Pip has the following features:

1. Pip carries multiple address types in a common format. As such, it is beneficial for transition from one address to another, and for future evolution (of routing techniques as well as of addressing schemes).
2. The Pip address is completely general (multiple levels of hierarchy, expands to any number of systems).
3. The Pip address is compact—it grows with the number of systems.
4. The Pip address efficiently encodes policy (source-based) routes, both in “long form” (explicit path) and “short form” (path identifier).
5. Because the Pip address can be a path identifier (multi-layer if desired, like the ATM VCI/VPI), Pip can be used in a connection-oriented fashion.
6. The Pip address includes multicasting (potentially substantially more sophisticated than what is possible with IP multicast numbers, for instance, hierarchical multicast).
7. Pip efficiently encodes QOS (Quality-of-Service) information.
8. The routing table lookup with Pip is well-bounded (by the depth of the address hierarchy).
9. Pip accommodates “multiple defaults” routing from (multi-homed) stub domains.
10. Pip allows intra-domain routing and hosts to operate with no notion of the “inter-domain” parts of their address, if desired. This is equivalent to current IP hosts and intra-domain routers not needing to know their own network number.
11. Pip accommodates tunneling across transit domains.
12. By virtue of 9, 10, and 11, Pip accommodates strong separation of interior and exterior routing.
13. Pip simplifies handling mobile systems (by having flat network layer identifiers).

In short, Pip is a “next generation” protocol, intended to allow the internet to evolve over the foreseeable future.

One of the design philosophies behind Pip is that it encodes all “routing” information (what is traditionally spread over the address and QOS fields) in a single structure (the Routing Directive). The rules for parsing the structure are fairly simple, but yet provide a rich set of routing functions. Therefore, it is possible to use the same header for many different routing styles, including traditional hierarchical addresses, policy, source route, and virtual circuit. This will make evolution of the internet much easier, as it will be possible to add new functions incrementally.

Another design philosophy behind Pip is that it delays the definition of how internet packet should be composed and interpreted. The meaning of addresses and QOS information are dynamically determined by information in Directory Services, distributed protocols such as routing protocols, and MIBs, rather than in a protocol specification. Current internet protocols have continuously been moving towards this philosophy, but with header formats that are not conducive to late semantic definition. Pip facilitates late semantic definition of the internet protocol header. This on one hand makes it easier to evolve the internet incrementally, but requires that all systems (hosts, routers, and directory servers) be a little smarter, and that algorithms be a little more complex. This, in a nutshell, is the trade-off being made by Pip.

3.0 Pip Description

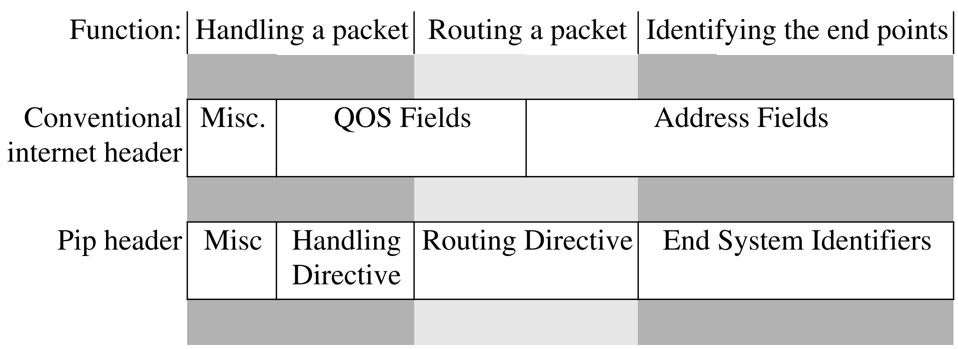
The Pip header has the following parts:

Misc	Handling Directive (HD)	Routing Directive (RD)	Router Options	End System Identifiers (IDs)	Host Options

Ignoring the miscellaneous parts and the options[1], there are basically three parts in the Pip header: The Handling Directive (HD), the Routing Directive (RD), and the End System Identification. The HD encodes those (conventional) QOS functions that have no effect on where the packet is routed[2]. This includes such functions as priority queueing, the congestion bit, and so on. The RD influences how a packet is routed. This is the only part of the packet that does this. The End System Identification simply identifies the source and destination of the Pip packet, and remain the same regardless of the contents of the RD.[3]

This header format accurately reflects the three fundamental functions of an internet protocol, which are 1) identifying the source and destination of a packet, 2) routing a packet, and 3) handling a packet. The identifying function simply determines if an internet packet is destined for the box that received it, and determines which box sent the packet. The routing function determines which is the next box on the path to the destination. The handling function is a catch-all category for all functions that are neither identification nor routing. This category includes such functions as decrementing hop-count, setting the congestion-experienced bit, and queueing the packet a certain way (for instance, because of a certain QOS setting), fragmentation.

The following figure shows how the three functions are split among the fields of a conventional internet header and a Pip header.



The address of a conventional internet header is used for two functions, identifying and routing. The routing function itself is split between the addresses and the QOS fields. The QOS can be used for both routing and handling. For instance, the QOS "low delay" can mean either (or both) to 1) give the packet priority queueing, which is a handling function, or 2) route the packet on a low-delay path.

1. Miscellaneous includes such fields as packet length, protocol, hop count, and so on. Router Options are those options fields, such as record route, that a router needs to process. Host Options are options that do not effect router operation.

2. See Perlman's paper [15] for an excellent discussion of route effecting and non-route effecting QOS. Indeed, the term "handling directive" is taken from Perlman's paper.

3. The notion of separating the ID from the address has most recently been proposed by Chiappa [6].

The poor alignment of header field to function in conventional internet headers has an adverse effect on the use of the protocol. For instance, it is hard to accomodate mobility with IP (or CLNP), because on one hand it is useful to get a new address to denote the new location, but on the other hand, getting a new address confuses the identification function. Indeed, mobility mechanisms in IP invariably involve a second address (either as a new IP header or an option), where one address serves the *identifying* function, and the other serves the *routing* function.

Another example concerns the difficulties in trying to assign backbone-oriented addresses in CLNP. This kind of addressing can result in multiple addresses for multi-homed stub domain, which leads to, among other things, the problem of what to do when a path via one of the backbones becomes unusable. It would otherwise be straight-forward for a host to obtain multiple addresses via directory service, and to switch from one address to another upon receiving an ICMP destination unreachable. However, the fact that this would confuse the identification function complicates this considerably.

Having argued that the Pip partitioning of header parts is preferable to that of conventional internet protocols, we now describe each of the major parts (routing, identifying, and handling).

3.1 The Routing Directive (RD)

Of the three internet protocol functions, the routing function is by far the most difficult. The conventional internet protocol routing information (the combined address/QOS/source routing) is weak in the context of the routing function. The Pip RD provides much more powerful routing information.

The most striking weakness of conventional internet protocols are in their inability to provide policy-based routing. It is generally accepted that the best way to provide for many important policies is through source-specified routing [2][6][9]. The only means of source specified routing in IP and CLNP are the source route option. But, these source routes 1) specify at a level of too much detail (the individual routers on the path, versus domains on the path), and 2) result in excessively large headers. This is especially true of CLNP, which commonly requires 20 bytes per element of the source route.

With IP and CLNP, the source is able to manipulate the path taken to some degree by setting the QOS bits. This, however, is weak both from a theoretical and from a practical standpoint. From a theoretical perspective, the QOS bits still do not allow the source to explicitly specify the path, and so is not a complete policy mechanism. For certain kinds of policy routing, however, it is more desirable for the routing algorithm to find the path than for the source to specify it. In this sense, QOS theoretically provides good routing information. From a practical perspective, this has turned out not to be the case because QOS is rarely implemented in hosts or in routers. This is because 1) examining the QOS bits is not generally crucial to routing, and 2) it has not always been clear what to do with the QOS bits. Therefore, it is difficult to introduce QOS when it becomes desirable to do so, because so much of the existing infra-structure ignores it. Unless the benefit in using it is greater than the pain in providing it, it doesn't happen.

The Pip RD has three parts, the Tunnel, the Routing Context (RC), and the Forwarding Table Index Fields (FTIF). These are discussed (in reverse order) in the following sections.

Routing Directive (RD)		
Tunnel	Routing Context (RC)	Forwarding Table Index Fields (FTIF)

3.1.1 Forwarding Table Index Fields (FTIF)

The design of the Forwarding Table Index Fields (FTIF), which is the part of the Pip header that contains, among other things, the traditional hierarchical "address", is based on the observation that *all* routing/addressing information in an internet header is essentially a source route of one kind or another. The argument for this given in [18] is briefly repeated here.

Consider the hierarchical IP address (in its current form, ignoring the multicast addresses):

IP Address		
Network	Subnet	Host

It has three parts, the Network, the Subnet, and the Host. These three fields essentially describe a loose source route. As an IP packet is routed through an internet (assuming that the packet is between hosts on different networks), the initial routers in the path route based on the Network information. Once the packet reaches a router within the destination network, the packet is routed on the Subnet information. Finally, once the packet reaches the last router on the path, the packet is routed on the Host information. The same argument can be made for any hierarchical address.

One of the main differences between the IP address and a true loose source route is that a loose source route has an explicit means of indicating which element of the loose source route is active (see below), while the IP address must be examined left-to-right by every router to determine which element of the IP address (loose source route) is active. The difference between a loose source route and a hierarchical address, then, is not fundamental but rather simply one of mechanics.

	Loose Source Route (LSR)		
Active LSR field indicator	Field 1 (Network)	Field 2 (Subnet)	Field 3 (Host)

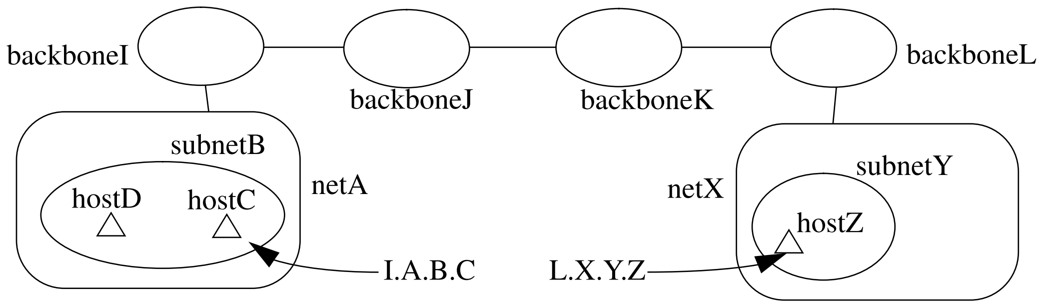
There may be various (less important) advantages and disadvantages to encoding a hierarchical address in the Loose Source Route form versus the single-address form, but the significant advantage of the Loose Source Route form is that it is more general, and can be used to encode loose source routes of all kinds, including policy routes. What this means is that a single routing information structure and associated lookup engine can be used for a variety of routing styles. Different aspects of routing can evolve (for instance, adding a layer of hierarchy to the address, adding policy, adding virtual-circuit style routing) without having to change the packet format or the forwarding engine. This should, for instance, allow implementors of routers to invest in a hardware-based forwarding engine without having to modify it every few years as routing evolves.

The Pip FTIF is a Loose Source Route. At a high level of description, it is encoded similarly to the above Loose Source Route representation of the IP address, and is shown here using the terminology of Pip:

	Forwarding Table Index Fields (FTIF)			
FTIF Offset	FTIF 1	FTIF 2	. . .	FTIF N

The first field of the FTIF is the FTIF Offset. As with the Active LSR field indicator, this indicates which of the FTIF Fields (FTIF) is currently being routed on. This is followed by a series of RHFs. Each FTIF is composed of two pieces of information, 1) the field value, and 2) the relationship with the subsequent field, which can be hierarchically up (↑), down (↓), or none (-). These relationship indicators are necessary to distinguish hierarchical addresses from true source routes. Note that the last FTIF has no relator, because there is no subsequent field to relate it to.

Consider the following topology:



It consists of two stub networks (A and X) each with subnets and hosts, and four transit backbones (I-L). Assume backbone-oriented hierarchical addresses, such as described in RFC 1237[7]. This results in the addresses shown (I.A.B.C and L.X.Y.Z, assuming that the capital letters here represent numerical components of an address hierarchy).

If HostC wishes to send a packet to HostZ, it forms the following FTIF:

FTIF Offset	Forwarding Table Index Fields (FTIF)							
	Source Address				Destination Address			
5	(host) C↑	(subnet) B↑	(net) A↑	(backbone) I -	$\boxed{\text{(backbone) L↓}}$	(net) X↓	(subnet) Y↓	(host) Z

The first four RHFs contain the source address, in order of lowest level first. The last four RHFs contain the destination address, in order of highest level first. This order represents the Loose Source Route of the packet—starts at hostC, goes to subnetB, then to netA, and so on to the destination. Because routing in this case actually only considers the destination address, the FTIF Offset initially points at the highest level of the destination address (backboneL, as indicated by the bold border). As the packet is forwarded through the internet, the FTIF Offset is incremented accordingly, so that at various stages in transit, the FTIF is:

Before backboneL	5	C↑	B↑	A↑	I -	$\boxed{\text{L↓}}$	X↓	Y↓	Z
While in backboneL	6	C↑	B↑	A↑	I -	L↓	$\boxed{\text{X↓}}$	Y↓	Z
While in networkX	7	C↑	B↑	A↑	I -	L↓	X↓	$\boxed{\text{Y↓}}$	Z
While in subnetY	8	C↑	B↑	A↑	I -	L↓	X↓	Y↓	$\boxed{\text{Z}}$

If host C wished to form a policy route to host Z, it would form the following FTIF:

				Policy Route						
4	C↑	B↑	A↑	$\boxed{\text{I -}}$	J -	K -	L↓	X↓	Y↓	Z

Backbones K and L have been added to the "loose source route", forming a policy route. The FTIF Offset initially points at backbone I, which is the first element in the policy route. The same FTIF structure (and resulting forwarding mechanism) encodes both hierarchical address and policy route. As is shown in the examples, the resulting FTIF size for this policy route is substantially less than the NSAP addresses in a CLNP packet.

For host C to send a packet to host D, the following FTIF is created:

2	C -	D

Because the source and destination are in the same subnet, only the "host" part of the addresses are needed. This shows that local traffic can have very small and simple addresses.

3.1.2 Routing Context(RC)

Conventional internet protocols have QOS fields. These fields can be used both to instruct the router as to how to handle a packet (for instance, by queueing a "low delay" packet in front of a "low-cost" packet), and to instruct the router as to how to route a packet. The RC field of Pip serves the latter function, only with more generality.

The reason that the RC field has the name it does is that, from a modeling perspective, a router that does QOS-routing can be viewed as multiple Routing Contexts, one per QOS type. Indeed, a QOS routing algorithm, such as OSPF [14] or ISIS [11], behaves like multiple Routing Contexts to the extent of running multiple spanning tree algorithms, one per QOS, and keeping separate forwarding tables.

Mechanistically speaking, the function of the RC field in Pip is to distinguish between one of multiple forwarding tables. Each of these forwarding tables is directly indexed with the FTIF value (thus allowing a fast forwarding table lookup). Since the RC is therefore a general purpose "logical routing domain" discriminator, it can be used to discriminate between more than what is conventionally considered to be "QOS" parameters. For instance, the RC field can be used to discriminate between multiple routing algorithms (for instance, multicast vs. unicast), multiple address families (such as globally unique vs. virtual path identifier), multiple user groups (packet coloring), and so on.

Unlike the QOS fields of IP or CLNP, routers are permitted to modify the RC field on in-transit packets. This can be a powerful mechanism, allowing for instance routers to tag packets for any number of purposes such as alternate path routing (see example 8 the appendix) and limiting certain packets to certain links. Each router (in theory, but almost certainly each routing domain in practice) can locally define the semantics of each bit in the RC. This allows each router (domain) to establish private RC uses. Therefore, where semantics are different across domain boundaries, or where semantics are common but the bits used to encode those semantics are different, it is necessary to modify the RC before transmition.

Even though the RC field is comprised of multiple parameters (Level, QOS, Routing Type, etc.), it is treated as a flat field *by the forwarding engine*. In other words, the forwarding engine does not care whether a given forwarding table is being accessed because, for instance, a certain QOS should be applied or because a packet is active at a certain level of the hierarchy. By separating the *reason* that a packet is being forwarded a certain way from the *mechanics* for doing the forwarding, forwarding engines can be built that do not have to be modified every time a new QOS type, routing type, or what have you, comes along.

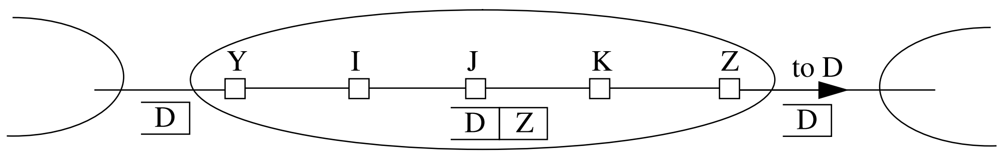
IP does not cleanly separate these two notions. The TOS[4] (Type-of-Service) field of IP states explicitly what the purpose of the TOS bits are (delay, error, bandwidth). This of course doesn't prevent routers from interpreting the TOS bits in whatever way is beneficial, but it makes it unlikely that the necessary generality will be built into routing algorithms, directory services, and the like, to be able to easily evolve the interpretation of the TOS bits. CLNP does only slightly better in this regard. It has some QOS fields (although not all) whose specific meaning is not part of the CLNP specification per se, but 1) these fields are optional, 2) the encoding of these fields is awkward, and 3) there are multiple such fields (QOS, security, priority). As a result, these fields are not parsed by most forwarding engines, thus making it more difficult to introduce QOS later on. In addition, neither IP nor CLNP routers are allowed to modify the QOS parameters of an in-transit packet, whereas Pip routers can. This further limits what can be done with the (IP and CLNP) QOS fields.

A typical implementation of Pip would use the RC field as a direct index into a table (the RC Table), which consists of pointers to forwarding tables. Thus, the correct forwarding table is reached quickly. The forwarding table entry contains a pre-calculated "new RC" value, which is copied into the RC field on transmitted packets.

3.1.3 Tunnel Field

A common mechanism in internetting is tunneling. Tunneling is temporarily changing the destination of the packet in order to traverse one or more routers that are incapable of routing to the real destination. One example of this is in routing through a backbone whose internal routers do not contain entries for external destinations. In this case, it is necessary for the entry border router to "tunnel" to the exit border router. Another example is partition repair in the ISIS routing protocol [11].

A common tunneling technique is self-encapsulation. For instance, assume that the entry border router is Y and the exit border router is Z, and routers I, J, and K are between them. A packet arrives at Y, destined for



host D. Y and Z know how to route to D, but I, J, and K do not know how to route to D. Therefore, Y encapsulates the packet in another internet header, with a destination address of Z. I, J, and K route this packet to Z, which then decapsulates it and forwards it towards D. A less common means of tunneling is to install a loose source route (in the true IP sense, not the Pip sense) into the packet.

This self-encapsulation technique could work with Pip, but self-encapsulation has drawbacks. For instance, in a cut-through routing such as ATM switching, where the Pip header is the first cell of a stream of cells, and is used to forward the cell, self-encapsulation requires that additional cells be created and inserted into the cell stream on the fly.

The Pip header therefore contains a Tunnel field. The field contains two parts, the Source Tunnel and the Destination Tunnel. The Source Tunnel is needed only in case an intermediate router needs to form a return

4. This paper uses the acronyms TOS (Type-of-Service) and QOS (Quality-of-Service) interchangeably.

packet, such as an error packet, to the router indicated by the Source Tunnel or to the Source Host. If the Destination Tunnel is 0, then forwarding takes place on the RC/FTIF. If the Destination Tunnel is not 0, then routing takes place on the Destination Tunnel field. In other words, the Tunnel field over-rides RC/FTIF routing.

Tunnel	
Destination Tunnel	Source Tunnel

Pip tunneling can also be used as a default routing mechanism (that is, a means of routing a packet out of a stub domain without requiring internal stub routers to have routing table entries for external destinations). In this case, the originating host puts a value in the Destination Tunnel field indicating the desired exit backbone. By convention, the value 1 can be used to mean “closest exit point”. If the closest exit point is not the best exit point, or if the exit point chosen by the host is not the closest, a “Tunnel Redirect” packet is used to inform the host of the correct exit point. This way, optimal default routing is possible without requiring the internal routers to know external routing information.

As with the RC and FTIF fields, the Tunnel field can be modified by the router before transmission. When a tunneled Pip packet reaches the tunnel exit router, that router writes the Destination Tunnel to 0, and forwards based on the RC/FTIF.

3.2 End System Identifiers

The End System Identifiers consists of two flat (non-hierarchical) fields—the Source ID and the Destination ID. The ID fields only identify the source and destination of a Pip packet. Each ID field is globally unique. They have no bearing on the current location of source or destination. Therefore it is possible to modify the addresses of a Pip packet, for instance because a mobile host moved to a new network location, without confusing the hosts.

End System Identifiers	
Destination ID	Source ID

When an ID is present, it alone is used to identify the source and destination hosts. However, IDs can be mapped to the associated RC/FTIF, so that the RC/FTIF implies a certain ID when the ID is not present. The ID therefore need not be carried in most packets. This works as follows. When a packet is first sent from a source host X to a destination host Y, the ID is included. The destination host Y, upon receiving the packet, associates the source ID with the source address, as determined by 1) the RC, and 2) the RHFs adjacent to “up” relators. When Y returns a packet to X, it writes X’s ID in the destination ID field, and X’s source address in the FTIF (as the destination address). This indicates to X that Y has recorded the mapping between X’s source address and X’s ID, and subsequent packets from X that contain the same source address need not include the ID field.

The ID fields are internally structured so that numbers from different number spaces can be used to create IDs. For instance, IDs can be composed out of IP, IEEE 802, E.164, and X.121 numbers, and perhaps even out of credit card numbers or national citizen identification numbers such as the Social Security Number in the USA. Each ID field is 64-bits in length.

3.3 Handling Directive (HD)

The HD is something of a catch-all field for any packet handling mechanisms that don't influence the route taken by a packet. Typical handling types are queueing directives, such as priority queueing, security directives, such as encryption, congestion handling directives, such as the congestion experienced bit, and so on.

The definition of individual bits is handled in the same way as the RC—that is, the meaning of the bits is defined dynamically through system management or configuration protocols, not through hard-coded definition in a standards document.

Each domain autonomously determines what meaning is assigned to each bit. When different domains use different bits for the same purpose, the value of the HD must be modified when a packet crosses domain borders so that the next domain may correctly interpret the meaning of the HD. The border router determines the proper translation via protocol exchange with the neighboring domain or via system management.

By packing all of the handling bits together, an implementation style whereby the HD is used as a direct index into a RAM memory, thus retrieving the appropriate handling mechanisms and values, is possible.

4.0 Number Assignment in Pip

As described in section 3.1.2, there are multiple number spaces in Pip. Many of these are local number spaces (such as a virtual path identifier number space, or a local source route number space), but at least one of the number spaces is global. That number space is the one from which hierarchical addresses are formed. A hierarchy of Pip number authorities is required to assign these numbers.

Pip addresses are rooted at and described in terms of the "top-level" (see examples in Appendix A). Therefore, at least one (and hopefully just one) global Pip number authority is required. The global Pip number authority assigns top-level numbers only. Each recipient of a top-level number assigns numbers hierarchically subservient to that number. This process is recursed to form a hierarchy of Pip number assignment authorities.

Pip is most efficient (has the fewest levels of hierarchy and has the smallest routing table sizes while still maintaining good paths) when Pip addresses are assigned in accordance with the topological hierarchy. Therefore, top-level Pip numbers should be assigned to backbones (as is described for the assignment of NSAPs in [7]). In addition, there should only be one top-level assignment authority. If there are multiple top-level assignment authorities, then global Pip addresses must be distinguished using the RC, which is possible, but places an unnecessary burden on the use of LRs. But, since politics may prevent the universal recognition of a single global Pip top-level assignment authority for assigning numbers to backbones, the following scheme seems appropriate.

Assign the lowest several hundred top-level numbers to countries, one per country. Each country can establish its own policies for becoming a number authority under its top-level number, if they wish[5]. This

5. In fact, a country could simply map its existing numbering plan for E.164 numbers into the Pip number hierarchy.

way, each country may operate independently of the top-level number authority. Top-level numbers greater than these are assigned as-needed to backbones, so that backbones can operate at the top level, and not under a country number, if desired. Each top-level number is rented (per year) at a fee that is large enough to inhibit trivial misuse of the numbers, but small enough that it doesn't prevent a serious backbone from obtaining a number. (Something between $1000 and $5000 dollars per year seems appropriate. Any profits from this fee could be cycled back into activities that support the internet, perhaps through an organization such as the Internet Society (ISOC).[6]) Backbones may also charge each other for advertising top-level numbers. This would be another inhibitor to over-propagation of top-level numbers.

5.0 Transition Approach

Like IP, Pip by itself is nothing more than a header format and some rules about how to forward the header. It is nothing without routing and addressing and related algorithms behind it. But since Pip can encode the semantics of existing internet headers (addresses, QOS, etc.), it can take advantage of existing routing protocols and addressing schemes. This is one of the main virtues of the proposal to move to CLNP—that it takes advantage of an existing body of work (IP as well as CLNP). However, Pip will allow us to move forward into advanced features that CLNP does not handle, while still allowing us to take advantage of existing work.

Since Pip can encode backbone-oriented addresses that are semantically equivalent to NSAP addresses, transition to Pip will be almost identical to the transition to CLNP already described by Callon [3]. It might in fact be easier to transition to Pip than to CLNP, because Pip mechanisms can be used for transition purposes. For instance, to aid in translating Pip packets to IP packets, a bit of the HD can be used to indicate that the Pip packet can be translated into an IP packet, and that the ID field carries the IP addresses.

Once most of IP has disappeared (and therefore scaling and address depletion are no longer concerns), we can evolve advanced features into the internet (policy, mobility, flow control) without having to change the internet protocol per se. (Of course not having to change the internet protocol doesn't mean not having to change routers. But not having to change the internet protocol is still better than having to change it, especially because it facilitates piece-wise evolution).

REFERENCES

[1] Bennett, C.J., "The Overheads of Transnetwork Fragmentation", Computer Networks, Vol. 6, No. 1, February, 1982, pp. 21-36.

[2] Breslau, L. and Estrin D., "Design of Inter-Administrative Domain Routing Protocols", Proceedings of ACM SIGCOMM '90, Philadelphia PA, September 1990

6. Mention of ISOC here is entirely my doing. At the time of this writing, no officer of member of ISOC has been consulted concerning any aspect of Pip number assignment.

[3] Callon, R, “A Simple CLNP-based Proposal for Internet Addressing and Routing”, available by anonymous FTP somewhere.

[4] Cheriton, D.R., “Sirpent: A High-Performance Internetworking Approach”, Proceedings of ACM SIGCOMM ‘89, Austin Texas, September 1989

[5] Cidon, I., and Gopal, I., “Control Mechanisms for High-Speed Networks”, Proceedings of IEEE International Conference on Communications ‘90, Atlanta Georgia, April 1990

[6] Chiappa, J.N., “A New IP Routing and Addressing Architecture”, IETF Internet Draft, draft-chiappa-routing-00.txt, available by anonymous FTP at nnsc.nsf.net.

[7] Collela, R., Gardner, E.P., Callon, R.W., “Guidelines for OSI NSAP allocation in the internet”, RFC-1237, USC/Information Sciences Institute, July 1991.

[8] Kent, C., Mogul, J., “Fragmentation Considered Harmful”, Proc. SIGCOMM 87 Workshop, August, 1987.

[9] Lepp, M., Steenstrup, M., “An Architecture for Inter-domain Policy Routing”, IETF Internet Draft, draft-chiappa-routing-00.txt, available by anonymous FTP at nnsc.nsf.net.

[10] International Organization for Standardization ISO8473, “Protocol for providing the Connectionless-mode Network Service”

[11] International Organization for Standardization ISO10589, “Intermediate System to Intermediate System Intra-Domain routeing exchange protocol for use in Conjunction with the Protocol for providing the Connectionless-mode Network Service (ISO 8473)”

[12] McQuillan, J.M., Richer, I., Rosen, E.C., “The New Routing Algorithm for the ARPANET”, IEEE Transactions on Communications, Vol. COM-28, No. 5, May 1980, pp. 711-719.

[13] Mogul, J., Deering, S., “Path MTU Discovery”, RFC 1063, USC/Information Sciences Institute, November, 1990.

[14] Moy, J., “OSPF version 2”, RFC-1247, USC/Information Sciences Institute, July, 1991.

[15] Perlman, R., “Incorporation of Service Classes into a Network Architecture”, Proceedings of the Seventh Data Communications Symposium ACM SIGCOMM, Vol. 11, No. 4, October 1981, pp. 204-210.

[16] Perlman, R., “Byzantine Routing”, PhD Thesis, Department of Computer Science, MIT, 19??.

[17] Postel, J.B., “Internet Protocol”, RFC-791, USC/Information Sciences Institute, September, 1981.

[18] Tsuchiya, P.F., “Scaling and Policy Routing using Multiple Hierarchical Addresses,” Proceedings of SIGCOMM ‘91, Zurich, September 1991.

[19] Wang, Z., Crowcroft, J., “Shortest Path First with Emergency Exits,” Proceedings of SIGCOMM ‘90, Philadelphia, September 1990.

Appendix A: Examples

This appendix contains a series of examples that demonstrate the power and flexibility of Pip. These examples show a number of features not mentioned in the main body of the paper, such as isolating stub domains from external addressing conventions.

The examples are based on the following (greatly simplified) Pip forwarding algorithm:

1. If the Destination Tunnel field is non-zero:
    - 1a. Directly index into the tunnel forwarding table with the Tunnel value.
    - 1b. Potentially modify the Destination Tunnel value (based on forwarding table entry info).
2. Else:
    - 2a. Use the FTIF Offset to determine appropriate FTIF
    - 2b. Directly index into the forwarding table[7] indicated by the Routing Context (RC), using the FTIF value concatenated with the relator as the index.
    - 2c. Potentially modify FTIF value (based on forwarding table entry info).
    - 2d. Potentially increment the FTIF Offset (based on forwarding table entry info).
    - 2e. Potentially modify the RC value (based on forwarding table entry info).
3. Depending on information in forwarding table entry, either:
    - 3a. Execute forwarding algorithm from step 2a (to examine original or subsequent FTIF), or
    - 3b. Forward packet.

Most of the examples relate to the network of figure 1. This network shows a (private) multi-homed stub network with two internal levels of hierarchy (X), two single homed stubs (Y and Z), and 5 backbones (A - E). The numbers shown are the numbers (FTIF values) associated with each element of the hierarchy. Using these numbers, hierarchical addresses can be composed for the hosts. For instance, stub Y has number 92, and is “under” backbone E, which has number 61. Therefore, the “address prefix” for stub Y is <top-level = 61, top-level - 1 = 92>. Since host v (in stub Y) has number 14, the “address” for host v is <top-level = 61, top-level - 1 = 92, host-level = 14>. In IP “dot notation”, host v’s address is written as 61.92.14. Note that because stub X is attached to two backbones, it has two “address prefixes”, 1.14 and 26.81. And, host x has two addresses, 1.14.12.96, and 26.81.12.96.[8]

Example 1: Packet from host x (in stub X) to host v (in stub Y).

For this example, host x sends a packet to host v, and host x does not specify which backbones to traverse, but rather leaves this decision up to the routers.

---

7. The forwarding table is the table that is accessed to determine how to route an internet packet. Conceptually, it contains only that information needed to route a packet. The forwarding table is derived from other tables, such as the routing tables, which contain more information.

8. As shown in example 6, it is possible to operate Pip in such a way that no intra-domain routers or hosts need have any knowledge of their stub’s “address prefix”. As such, having multiple addresses, or obtaining new prefixes, for instance because of subscribing to a different backbone, would not impact internal routers or hosts.

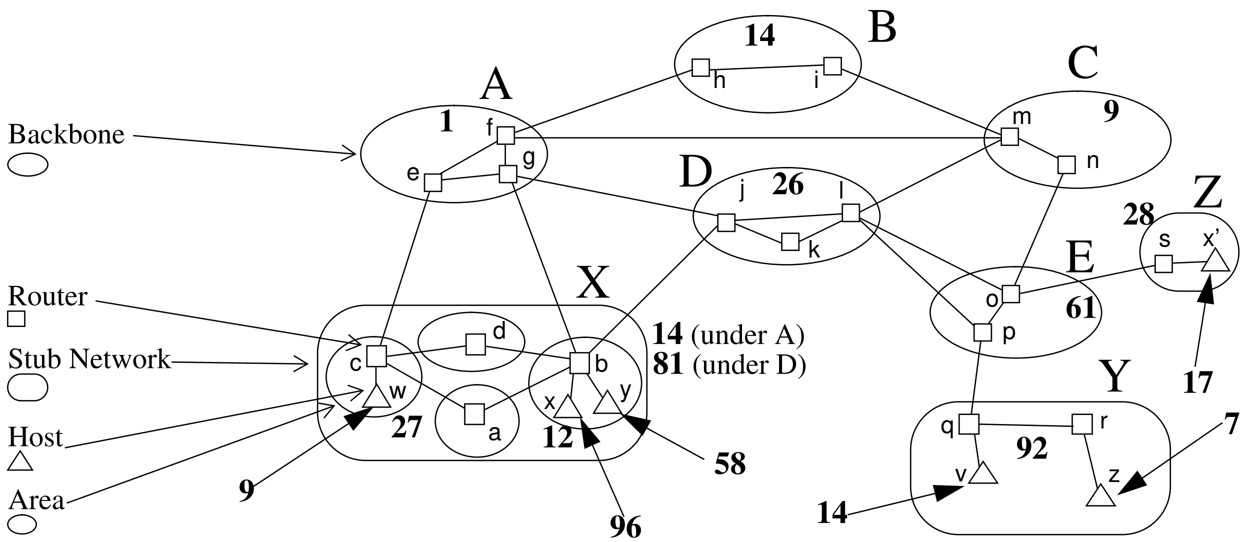


Figure 1: Example Internet Topology and FTIF Addresses

Assume that host x queries directory service with the name of v and obtains the following information: <top-level = 61, top-level - 1 = 92, bottom-level = 14>. This is host v's "address", but expressed as a series of hierarchical components rather than as a string of bits.[9]

The following RC/FTIF is formed by x:

Routing Context	FTIF Offset	Forwarding Table Index Fields (FTIF)						
Level = Top-level	5	96↑	12↑	14↑	1 -	**61↓**	92↓	14

The first two fields (before the double lines) are the Routing Context (RC) and FTIF Offset. The remaining fields are the FTIF <value, relator> duples. The FTIF in a bold box is the one currently pointed to by the FTIF Offset.

The RC field has certain bits defined to indicate which level of the hierarchy the packet is being routed at. This is necessary because each router has one (or more) forwarding tables per hierarchical level *that the router maintains destinations for*. This is necessary to allow FTIF values to be reused at different levels of the hierarchy. For instance, there could be an area within stub X with area address "61" (the same as the backbone number for E). If router b, for instance, did not maintain separate routing tables for different levels, it would not be able to determine if the 61 in the FTIF meant "area 61" or "backbone 61". For this and subsequent examples, we assume that there may be other semantics in the RC, such as QOS type or address type. These semantics are not shown if they are not pertinent to the example.

The first four RHFs form what in a conventional internet packet would be the source address. The hierarchical fields of the source address are listed in order of bottom up. The last three RHFs form what would be the destination address. The hierarchical fields of the destination address are listed in order of top down. The reason the source address goes bottom up and the destination address top down is because this

9. In the general case, directory service would have to return more information than this, potentially including QOS information, multiple hierarchical addresses, and preferences associated with the addresses.

is the order in which the elements of the hierarchy are traversed. That is, this is (a superset of) the loose source route of the packet.

Because host x wishes routing to find the path of the packet, it sets the FTIF Offset to point to the top-level component of the destination address. This is analogous to how an IP packet would be initially routed (on the destination network number, which is the top-level address component).

All of the routers on the path from x up to and including router l (b-j-l, assuming that the shortest path is via backbone D) forward based on <top-level, 61>. But, router l increments the FTIF Offset and decrements the Level. Router p (which is in "top-level, value 61" and therefore doesn't care how to route to it), therefore routes on <top-level - 1, 92>. "92" is the value hierarchically under "61" assigned by backbone E to stub Y. Router p also increments the FTIF Offset and decrements the Level, so that router q can route based on a "host level" forwarding table.

Note that "decrementing the Level" is accomplished by modifying the RC value. This is done by retrieving the new RC value, which has been pre-calculated based on information from the routing algorithm, from the forwarding table entry and writing it into the packet. In so doing, the router also remaps RC bits into those expected by the next router, and potentially modifies the meaning of other RC semantics, such as QOS (see example 8).

Example 2: Return packet from host v to host x.

When host v sends a return packet to host x, host v simply reverses the order of the RHFs, sets the Level and FTIF Offset appropriately, and forwards the resulting RC/FTIF:

Top-level	4	14↑	92↑	61 -	1↓	14↓	12↓	96

Note that this return packet is routed via backbone A, which is 1) a non-optimal route (assuming that D is the optimal route), and 2) is asymmetric with the forward path. This is one of the dangers associated with backbone-oriented hierarchical addresses. By use of default routing and default-route Redirects from border routers to hosts, symmetric paths can be found in all cases without requiring the host to know backbone routing information (see example 6).

Example 3: Packet from host x to host w (intra-domain).

For host x to form an FTIF to send to host w, it compares w's address (that it received from Directory Service) with its own address and determines that they share two top-level address components. Since these components are therefore not needed by routing, they are not included in the RC/FTIF:

Top-level - 2	3	96↑	12 -	27↓	9

Routers b and d route on <top-level - 2, 27>. Router d decrements the Level and increments the FTIF Offset before forwarding the packet, so that router c routes at the "host level". Because the complete address does not have to be encoded in Pip packets, Pip packet size is small for local traffic. For instance, a packet from host x to host y contains only 2 RHFs.

Example 4: Choosing the exit backbone.

Assume that, as with example 1, host x has a packet for host v. However, host x wishes to choose the exit backbone that is used. This is equivalent to choosing the long-distance carrier in the USA telephone system, and so is a common policy requirement. Host x forms the following RC/FTIF:

Top-level	4	96↑	12↑	14↑	1 -	61↓	92↓	14

The only difference between this FTIF and the one formed in example 1 is that the FTIF Offset points to the highest component of the *source* address, rather than that of the destination address. As a result, router b forwards the packet to router g (backbone A) instead of router j (backbone D). Router b increments the FTIF Offset to point to <top-level, 61> before forwarding to g, so that router g can subsequently route to backbone E.

Router b knows not to decrement the Level, because the relator indicates “none”, not “down”. Or, put another way, the “none” relator causes router b to retrieve a different forwarding table entry than the “down” relator. The new RC value in this entry has been pre-calculated to not decrement the Level.

Example 5: Choosing a policy route.

Assume again that host x has a packet for host v. However, this time host x wants to specify the entire backbone path as A-B-C-E. Host x forms the following RC/FTIF:

Top-level	4	96↑	12↑	14↑	1 -	14 -	9 -	61↓	92↓	14

Two RHFs, indicating backbones B (14) and C (9) have been inserted between the source and destination addresses. As with example 4, router b increments the FTIF Offset and forwards the packet to router g. The FTIF Offset will then be pointing to <top-level, 14> (rather than <top-level, 61>, as in example 4), and so router g routes towards backbone B rather than backbone E. Subsequent routers increment the FTIF Offset at the appropriate points, and the packet follows the backbone path desired.

Host v can create a symmetric return path by reversing the order of all RHFs. Or, host v can isolate the source address (by finding RHFs adjacent to “up” relators) and destination address, reverse their order in the FTIF, and then insert its own backbone path based on its’ policy considerations.

Note that this FTIF is quite small, because each of the FTIF values in the backbone path only need to be distinguishable among the set of (top-level) backbones. If there are, say, 10000 top-level backbones globally, each FTIF field need be only 14 bits long, plus 2 bits for the relator. The above FTIF would only be 18 bytes long, which is less than one NSAP address. This small packet size means that a significant amount of policy information can be employed without having to resort to path setup [2][6][9].

Example 6: (Multiple) default routing and inter-domain address isolation.

As with example 1, host x sends a packet to host v, and host x does not specify which backbones to traverse, but rather leaves this decision up to the routers. However, stub X operates in such a way that 1) its internal routers do have any routing table entries for external destinations (default routing), and 2) neither internal routers nor hosts know or need to know what the inter-domain components of their addresses are.

Because internal routers do not have routing table entries for external destinations, they do not have a "top-level" forwarding table. Instead, they have a "default routing" forwarding table.

For this to work, an RC bit is defined to mean "use default routing". Hosts know to set this bit for packets destined for hosts outside of their stub (inter-domain destinations). This bit has only local meaning (within X), and is lost when a packet exits the stub. Since stub X has two exit backbones, the default forwarding table has only two entries.

The following RC/FTIF is formed by x:

Default Routing	3	96↑	12↑	**2↑**	2 -	61↓	92↓	14

The RHFs that previously held the top two levels of the address now hold default route values (2). Each internal non-border router routes on this FTIF, and the packet is routed to a border router. The default forwarding table for the border router (b in this case) indicates that, rather than forward the packet to the next router, it 1) modifies the FTIF value to be 12 (see step 2c of the forwarding algorithm), 2) increments the FTIF Offset, and 3) goes to a second default forwarding table (step 3a of the forwarding algorithm). This second forwarding table indicates that 1) the FTIF value should be modified to be 1, 2) the FTIF Offset should be incremented, and 3) the RC changed to indicate routing at the top level. Any other RC bits (such as QOS indicators) are translated into those positions expected by the next router. The forwarded packet therefore has the following RC/FTIF:

Top-level	5	96↑	12↑	14↑	1 -	**61↓**	92↓	14

This packet has the complete source address of the host. It is as though the host knew its own source address. Being able to isolate internal operation from external addressing conventions like this is a very powerful mechanism. It allows stub domains to subscribe to different backbones without the inconvenience of having to restructure internal addressing.

If the default route chosen by the host is not the optimal one, a "Default Redirect" can be sent to the host by the border router. For instance, in this example, the default route FTIF value "2" indicated backbone A (as evidenced by the fact that the prefix from backbone A was inserted into the address). Since this is not the optimal path to the destination, the border router would send host x a "Default Redirect", informing x to use the default FTIF value indicating backbone D. By convention, a default value of "1" can be used to mean "closest default route".

Note that it is in part the fact that the identification function has been separated from the routing function that makes this kind of address manipulation possible. The fact that the individual fields of the hierarchical address have been separately and explicitly encoded (as RHFs) also contribute to this capability.

Example 7: Virtual path routing.

As with example 1, host x sends a packet to host v. But, assume that previous to transmitting the packet, a "virtual path" has been established (with virtual path identifier 4822). This virtual path might have been established as a policy route (although virtual paths are less necessary with Pip than with conventional internet protocols, because the policy route can efficiently be encoded in the Pip header), or for reasons having to do with flow control.

The following RC/FTIF is formed by x:

Virtual Path	1	4822

The RC indicates that the FTIF comes out of the virtual path number space (versus for instance a level of the global hierarchical address space or the default routing space). Only one FTIF is needed if there is only one virtual path identifier. Each router on the path could modify the FTIF value according to its local virtual path identifier assignment.

If the virtual path number is hierarchical, as is the case with ATM (VPI/VCI), then the following RC/FTIF is formed by x:

Virtual Path	1	4822↑	1297↓	1998

With this virtual path, the first FTIF might be routed on by X's routers, the second FTIF by backbone routers, and the third by Y's routers. The path setup procedure indicates to host x which FTIF values to use.

Example 8: Alternate path routing.

One method of avoiding congestion without running a delay-based routing algorithm (such as the "New Arpanet" algorithm [12], now quite old), is to temporarily alternate route around localized congestion. An example of this is Emergency Exit routing [19]. Since alternate paths can sometimes back-track over nodes from which a packet came, it is necessary to tag packets that are taking an alternate route so that these packets do not revert to the primary path and loop back to the congested spot.

A way to implement this with Pip is to define one or more RC bits to mean "Primary path, alternate path 1, alternate path 2" and so on. When a packet takes an alternate path, the RC is changed to indicate this fact. It is subsequently routed on the (pre-computed) alternate path. If additional congestion is encountered, it can take a third alternate, and so on, until the packet is either delivered or dropped.

This example illustrates the advantage of allowing routers to change the RC (QOS, so to speak) of in-transit packets. Note as well that the HD can also be modified when a packet takes a alternate path, for instance to lower its drop priority so that it doesn't continuously take alternate paths.

Example 9: Node-level source routing.

Examples 4 and 5 showed how Pip can do backbone-level source routing for policy routing. Other literature [16][4][5] suggests that node-level source routing has advantages. In the case of Perlman, source routing is used to make a network more robust. In the case of Cheriton (Sirpent) and Cidon (Paris), it is to speed up the forwarding process. Perlman encodes node identifiers in the source route, Sirpent encodes outgoing link identifiers, and Paris encodes self-routing switch codes.

Consider a case where a stub domain uses Perlman's byzantine routing for internal communications, but still uses normal hierarchical addresses for external communications.

An RC bit (or a value corresponding to several RC bits which collectively mean "address family") indicates the number space used for internal routers. In this numbering space, each router is given a flat identifier, counting up from 1. For instance, if a network has 500 routers, they are numbered 1 through 500, and the FTIF for each router is 11 bits long (9 for the value and 2 for the relator). Each host has a number assigned by its connected router. Therefore, even if each router has 500 or fewer hosts (for a total of up to 250,000 hosts), each FTIF is still only 11 bits. A source route of 16 hops plus two 8-octet IDs could be encoded in the same space required for two NSAP addresses.

Example 10: Multicast routing.

As with the previous examples that show the use of the RC for identifying separate number spaces (6-9), RC bits could be used to identify multicast addressing. Pip provides enormous potential for increasing the sophistication and efficiency of multicast routing.

For example, Pip can encode hierarchical multicast routing, where for instance one level of the FTIF indicated a multicast at the backbone level, while the next level down indicated multicast within a stub. This could be used, for instance, to allow a backbone to view the various stub locations of an international corporation as the group members of a single multicast tree (a single upper level multicast FTIF value), while in fact the corporation had many multicast groups (multiple lower level multicast FTIF values).

For another example, since different applications may benefit from different types of multicast trees (for instance, applications that don't require smallest possible delays could get away with a single multicast tree instead of multiple source-rooted multicast trees), multiple multicast algorithms could run in parallel, with bits in the RC Field distinguishing between them.

Example 11: Mobile Hosts

Assume as in example 1 that host x is exchanging packets with host v. The initial packet from host x to host v is as shown in example 1, however here we show it with the Destination and Source ID fields (DID and SID) attached (after the second pair of double lines):

Top-level	5	96↑	12↑	14↑	1 -	$\boxed{61↓}$	92↓	14	DID=v	SID=x

When host v receives this packet, it relates ID=x with address 1.14.12.96, and returns a packet that includes both IDs, so that host x relates ID=v with address 61.92.14. Now, subsequent packets need not contain the ID fields.

Assume that later x moves, so that now it is attached to router s in stub Z (labeled x' in Figure 1). As a result of this movement, host x is assigned a new address of 61.28.17. Host x immediately sends a packet with the following header (if there is no application-level data to send, host x can send a packet with a NULL body):

Top-level - 1	3	17↑	28 -	$\boxed{92↓}$	14	DID=v	SID=x	Host Option

Upon receiving this packet, host v determines that host x has a new address. The backbone number, 61, is not necessary because, since that backbone is common to both source and destination, routing does not take place on the address component for the backbone. This packet includes a Host Option for managing the status of the new address. For instance, a sequence number indicating that this is in fact the most recent address, and perhaps information pertaining to the status of the previous address (still valid, no longer valid). If both hosts are mobile, then some kind of third party server is necessary, so that current addresses can be determined in case both hosts get new addresses simultaneously. If the hosts get new addresses often, then the IDs can simply be included in every packet.