PRELIMINARY DRAFT:
Pip: The 'P' Internet Protocol

Paul F. Tsuchiya
Bellcore
tsuchiya@thumper.bellcore.com
May 19, 1992

Status

1.0 Purpose of this draft

Pip is an IP protocol that scales, encodes policy, and is high speed. The purpose of this draft is to explain the basic concepts behind Pip so that people can start thinking about potential pitfalls. I am proposing Pip 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. Because this proposal, which represents new ideas, is competing with old (and therefore well thought-out) ideas, I wish to circulate it (and get the process started) as quickly as possible, albeit in not as complete a form as I would like. I expect to have a complete proposal by the beginning of September. There will be a plenary presentation and a BOF covering this material at the Boston meeting of IETF.

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 (this paper only briefly touches on mechanisms for controlling connections).
6. The Pip address includes multicasting (potentially substantially more sophisticated than what is for 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 8 and 9, Pip accommodates 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 simple on one hand, but provide a rich set of routing functions. Therefore, it is possible to build a single forwarding engine that will accommodate many different types of routing styles, including traditional hierarchical addresses, policy, source route, and virtual circuit. This way, the forwarding engine can be built in hardware and can remain constant even while internet routing evolves.

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 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 [OSI2]—that it takes advantage of an existing body of work. However, Pip will allow us to move forward into advanced features that CLNP will not handle, while still allowing us to take advantage of existing work (although not as easily as moving to CLNP will).

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 [Ref]. 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. (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).

In the following sections, I show how Pip works *outside* of the context of interoperation with existing addressing and routing schemes.

4.0 Pip Header Structure

Figure 1 shows the Pip header structure. The Pip header has 5 parts not found (at least in this form) in current internet protocols. They are the Handling Directive (HD), the Tunnel, the Logical Router (LR), the Routing Hints (RH), and the IDs. While these parts are fundamental to Pip, the details of their layout, and the layout of other fields, is open to change.

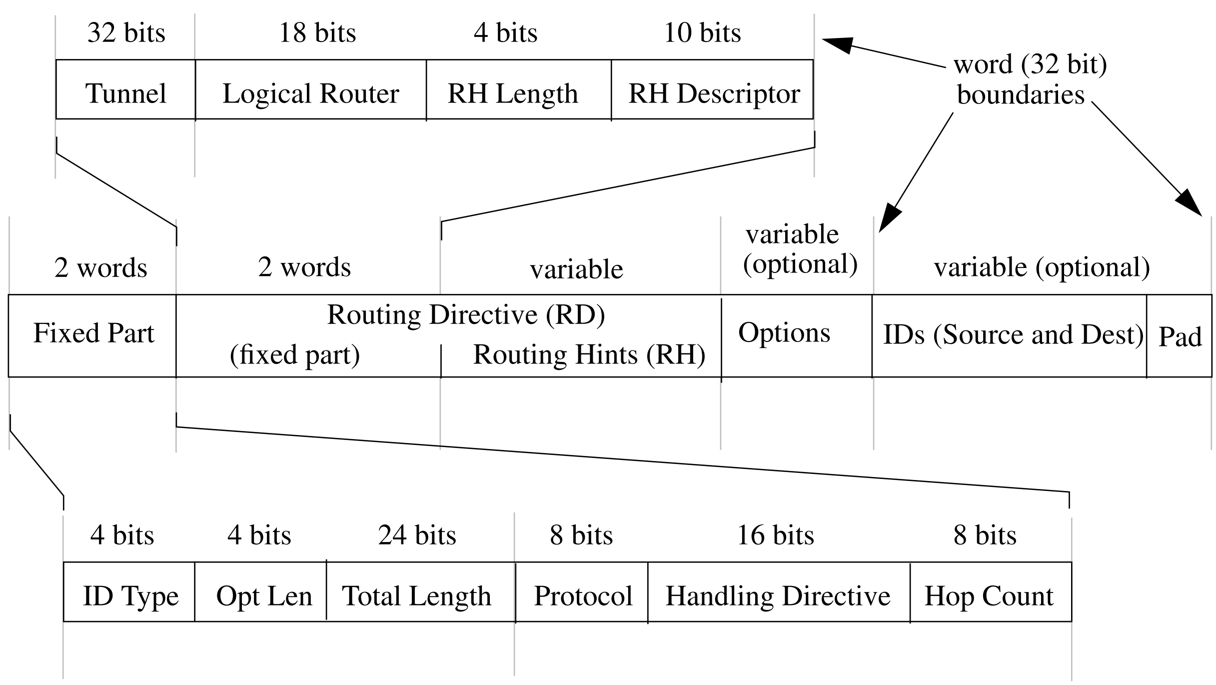

Figure 1: Pip Header Structure

The IDs field contains flat (non-hierarchical) values that do nothing more than identify the source and destination of a Pip packet.[1] The Routing Directive (RD), which consists of the Tunnel, the LR and the RH, contains routing information. Either the Tunnel or the RH are used, but not both. The RH holds routing information such as (hierarchical) addressing, source-route (including policy), and virtual circuit

1. The notion of having separate IDs was most recently proposed by Noel Chiappa in his NIMROD routing architecture [Chi].

information. The Tunnel simply marks entry and exit points of a domain, and is used to temporarily override the RH. The LR holds route-effecting QOS information (such as routing metrics), plus various information needed to make the RH operate properly. The HD holds non-route-effecting QOS information, such as queueing directives, congestion avoidance and control, and priority.

This packet structure better represents internet protocol functions than traditional internet protocols. For instance, traditional internet protocols combine the functions of identification and routing into the address fields. Doing this generally limits the flexibility of the protocol. For instance, host mobility is harder when the address combines these two functions.[2]

Traditional internet protocols also split the routing function over multiple fields (the address and the QOS fields). While this doesn't necessarily limit functionality, it generally complicates the routing table lookup function, or more accurately, it generally results in router implementations that ignore the QOS fields, thus making it harder to add QOS routing to the existing infrastructure.

Traditional internet protocols must use self-encapsulation in order to tunnel through groups of routers. Pip has a specific field for this purpose, thus eliminating the overhead of replicating the entire header.

No Pip header checksum is shown in Figure 1. I am undecided as to whether or not one is necessary, particularly since the HD, Hop Count, Tunnel, and RH fields will commonly change values from router to router. In fact, of the first 5+ (32-bit) words, only the first word will potentially not be modified. No fragmentation/reassembly fields are shown. I am strongly inclined to leave these out, and just depend on dynamic MaxPDU discovery to handle this. Finally, no version number field is shown. Protocol identification (at the previous layer) can serve this function.

The following sections cover the various parts of the Pip header in detail.

4.1 Boring Parts

The "boring" parts of the Pip header are the ID Type field (4 bits), the Options length field (4 bits), the Total Length field (24 bits), and the Protocol field (8 bits), and the Hop Count field (8 bits).

The ID Type describes the length and type of the Source and Destination IDs. The IDs can be 0, 4, 6, or 8 octets each (the actual types, which are not so boring, are described in the separate section on IDs below). The Options Length field gives the number of 32-bit options that come after the RD. The Total Length field gives the total length of the Pip packet, including the Pip header, in octets. The maximum size Pip packet is 2^{24} = 16,777,216 octets. This is substantially larger than the corresponding fields in IP or CLNP, both of which allow for maximum packet sizes of 65536 octets. These fields comprise the first 32-bit word.

The Protocol field indicates the higher layer protocol, and is equivalent to the IP Protocol field. The Hop Count field counts down the number of hops before the packet should be dropped. It is the same size as the corresponding fields in IP or CLNP, allowing for 256 hops. The Hops field falls on a 32-bit (and 64-bit) boundary, making it convenient to modify.

2. Indeed, IP solutions to host mobility invariably require two addresses, one "variable" address (indicating the mobile host's current location), and one "permanent" address. In this case, one address is purely for identification purposes, and the other for routing purposes.

4.2 Tunnel and Routing Directive (RD)

The RD is the most novel and powerful aspect of Pip. The RD is general, compact, and fast. It is general in that it can accommodate any address type and any routing algorithm type, including source-based routing.[3] It is compact in that it encodes hierarchical addresses efficiently. And, it is fast because 1) the number of steps required for the forwarding function is small, even in the worst case, and 2) the same steps are used for forwarding all types of routing, so an efficient and general forwarding engine can be built.

The RD composed of three parts, the Tunnel, the Logical Router (LR), and the Routing Hints (RH).

Because a router can be playing multiple roles, Pip models a router as multiple “Logical Routers”. For instance, a router may be operating at multiple levels of the hierarchy, may be participating in multiple routing algorithms, including multicast, may be operating with multiple routing metrics, and so on. While the function of logical routers is for most purposes a feature, it is required to make the RH mechanism work properly, as is described below.

The basic algorithm for finding a route is to 1) determine the forwarding table index, 2) determine which forwarding table to use (that is, which logical router is active for this packet), 3) index directly into the forwarding table (no search technique such as hashing or tree search is necessary) and retrieve the routing information, 4) modify the RD for the next-hop router. This is explained in more detail below (see Section 4.2.4)Tunnel

The 32-bit Tunnel is composed of two 16-bit fields, the Source Exit ID (SEI) and the Destination Exit ID (DEI). The DEI comes after the SEI, and so falls on the least significant bits of a word boundary.

When the DEI is 0, then the Tunnel is ignored and the RH is used to route the packet. Otherwise, the RD is ignored and the Tunnel is used.

The purpose of the Tunnel is as follows. Consider two routers, X and Y, both of which understand the RH (at the level at which the RH is operating). Between X and Y are a series of routers that do not understand the RH (at that level). Assume that a Pip packet (with a NULL Tunnel) arrives at X and should be routed to Y. In order to get the packet to Y, X fills the DEI field with a value that is understood by the intermediate routers to mean “route to Y”. X fills the SEI field with a value that is understood by the intermediate routers to mean “route to X”. The purpose of the SEI field is to handle the case where a return packet (an error packet or control packet of some sort) needs to be sent (either to X or to the original source host). When Y receives the packet, it recognizes the Tunnel as terminating at itself, writes the Tunnel field to 0, and forwards based on the RH.

Tunneling is traditionally useful for preventing external routing information from being required internally. It is also used by the ISIS routing protocol for repairing area partitions. Pip tunneling can be used for both of these purposes. Because of the way “addresses” (called RH Numbers in Pip) are assigned in Pip, however, tunneling turns out to be necessary just to make Pip work.

---

3. This is undoubtedly an over-statement, and I certainly cannot prove it, but I also cannot think of any address type or routing algorithm that isn’t accommodated by Pip. I will gladly buy a beer or equal-to-less-expensive beverage of choice for each person who can think of one.

There are no nested tunnels in Pip (that is, tunnels cannot have tunnels). While nested tunnels could be of some use, it seems that the usefulness of tunneling diminishes with the number of nested levels. By having only one level of tunneling, the packet format is simplified (and the size kept small). To make nested tunneling work, it would be necessary to either modify the size of the packet en route (to add and delete tunnels), or for the originating host to put in enough Tunnel fields for the deepest nesting. The former case is difficult because it requires changing the packet size, which doesn't work for instance with (cut-through) ATM switching. The latter requires extra complexity and overhead in informing the originating host how many Tunnel fields to include in the packet. For these reasons, I have chosen to limit tunneling to one level.

4.2.1 Logical Router (LR)

As described above, the LR field indicates which of multiple forwarding tables should be used when routing a packet. The many uses of the LR will become clear throughout the coming examples.

Note that in theory one can always use different indexing values, rather than different forwarding tables, as a means of distinguishing logical routers. This, however, couples "addressing" (RH numbering) between different logical domains, thus generally complicating things. For instance, one could use different RH values to indicate different QOSs (cost, delay, etc.), but that would require that each system have an RH Number indicating cost, another indicating delay, and so on. So, unless such coupling is convenient, it is best to decouple RH numbering using the LR field.

Even though the LR field can be treated as a flat field by a router, the individual bits have specific meaning. My goal is that most or all of the bits' meaning be determined dynamically (via system management or the routing protocol or some other distributed protocol), and not be specified in a standards document. This allows for the maximum flexibility in evolving the protocol (adding new features, purging old ones). For instance, upon booting, a host should, as part of its configuration process, contact a local router and learn the meaning of each bit of the LR field. A network debugger, even, could query attached routers for these definitions, so that meaningful information could be logged and displayed.

The following bits are likely to be required:

1. Level. This indicates what level of a hierarchical RH Number is being routed on at a given time. This use of the LR field is only necessary if hierarchical RH Numbers are being used.

2. Multicast. If multicast is used, at least one bit may be needed to indicate whether the packet should be multicast or unicast. If several multicast algorithms are in use, multiple bits may be needed.

3. Route-effecting QOS. This would be any QOS type that influences the route chosen, such as cost or high-bandwidth. Note that QOS need not be route effecting. For instance, a QOS type of low delay might only influence how packets are queued (given priority in the queue), but not influence how they are routed. In this case, the HD would have certain bits set aside for "low delay" (actually, priority queueing), but the LR would not. In other cases, a given QOS might effect both routing and handling.[4]

---

4. See Perlman's paper [Per1] for an excellent discussion of QOS types. Indeed, the "handling directive" terminology comes from Perlman.

4.2.2 Routing Hints (RH)

The RH is the most interesting and novel aspect of Pip. It holds what is normally thought of as the "address" in a traditional internet header. It can also hold many other kinds of routing information, such as policy information.

The RH consists of the RH Descriptor and the Routing Hint Fields (RHF, see Figure 2). The RH Descriptor tells how to interpret the RHFs. The RHFs are a series of fields, listed in the order that they will be required by the routers in the path from source to destination. This should not be taken to assume that the RHFs necessarily specify a source route, in some conventional sense of the term. Most normally, the RHFs will simply contain a hierarchical source and destination RH Number, where each RHF denotes one level of the hierarchical RH Number.[5] This and other uses of the RHFs (such as virtual circuit or path identifiers, true source routes, and Sirpent- or Paris-style source routes) are given later.

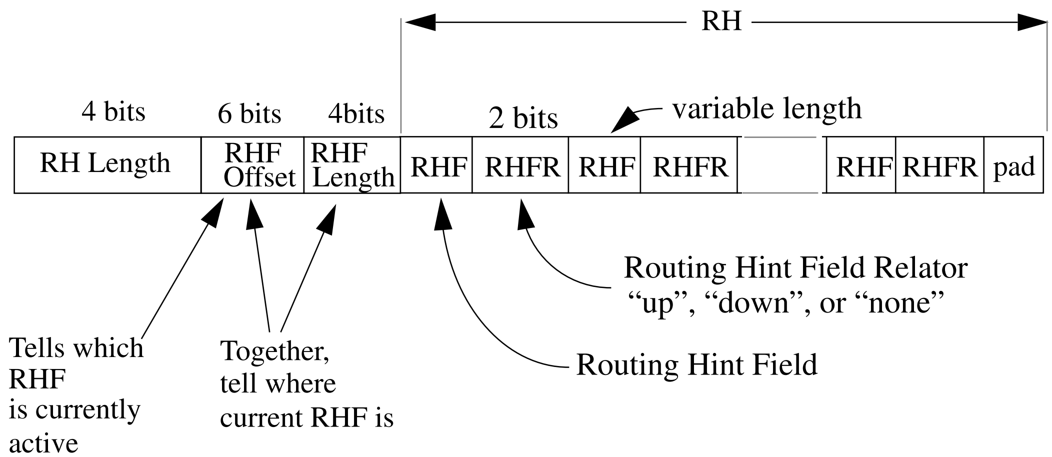

Figure 2: Routing Hints

Each pair of RHFs are separated by an RHF Relator (RHFR). The RHFR is a two-bit field that shows the relationship between the field before it and the field after. It has three values, up, down, and none. If down, the previous RHF is hierarchically above the subsequent RHF. If up, the previous RHF is hierarchically below the subsequent RHF. If none, the two RHFs are not hierarchically related.

The RH Descriptor and RH are parsed as follows. The 6-bit RHF Offset field determines which RHF is currently active. The RHF Length field indicates the size of each RHF (all of which are the same length). The RHF sizes represented by each RHF Length value are given in the following table:

RHF Length value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
actual field length	3	4	5	6	7	8	9	10	12	14	16	20	24	28	32	48

5. In a previous paper [Tsu], I argue that a hierarchical address is nothing more than a certain kind of source route, in the sense that a hierarchical address implies routing *to* a part of the hierarchy, then *to* another (lower) part of the hierarchy, and so on until the destination is reached.

After this is a series of 1 or more RHFs. Where the actual values needed in the RHFs vary greatly (some small, some large), this structure will result in a larger RH than seems necessary. I don't know how to shrink each RHF to its smallest size and still make the header parsing simple (and therefore fast).

After the RHFs comes enough padding to make the RD fall on a 32-bit word boundary.[6]

The combined 10-bit RHF Offset/RHF Length, then, is used to isolate the current RHF that a router should be routing on. A typical implementation on a common CPU/RAM processor would be to use the full 10 bits as a direct index into an array of size 1024, each entry of which contains data on how to isolate the current field. For instance, if RHF Offset = 3 and RHF Length= 8 (meaning each RHF/RHFR is 14 bits long), the data would instruct the processor to fetch the first (32-bit) word of the RH, shift left 10, mask with 0x00003c00, fetch the second word, shift right 22, mask with 0x000003ff, and OR the two results. In this example, the RHF/RHFR straddled 32-bit word boundaries, and so two fetches were needed. (The RHF Relator should also be saved off at this time to be used later.)

Once the RHF is isolated, it is used as a direct index into a forwarding table. The forwarding table can be well populated because (as is discussed later in this paper) the RHF values are chosen not based on how many things *might* have to be encoded at a given level of the hierarchy, but on how many things *are* actually encoded at a given level. In other words, the "address" that is ultimately carried in packets is, unlike current internet protocol addresses, well-utilized.

In addition to the information in the forwarding table described above, the forwarding table entry must also indicate whether the RHF Offset needs to be decremented. The RHF Offset is usually decremented when a packet crosses a hierarchical boundary. For instance, if the packet was being forwarded based on the equivalent of "network number" through a backbone, the router bordering the indicated network would decrement the RHF Offset so that the next router (the router in the indicated network) would automatically look at the "subnet number" field. Often a single router is acting at two or more levels of the hierarchy, for instance a level 2 router in the ISIS routing protocol. In this case, the forwarding table entry and RHFR would indicate that, instead of routing the packet to another router, the next RHF should also be examined (and, another forwarding table used). It would be unusual to find a router operating at more than three levels of the hierarchy. Further, address hierarchies are shallow. Telephone numbers in the USA have only 4 levels of hierarchy (including the international code). Therefore, the number of iterations of this search is well-bounded.

Note that this "field indexing" style of lookup is not just a cute optimization. Pip derives most of its routing flexibility from it, and wouldn't be general without it.

4.2.3 Fowarding Algorithm

This section describes the algorithm for forwarding a packet, based on the contents of the Tunnel and the RD (see Figure 3). For expository reasons, the unicast algorithm is defined, followed by the modifications needed for multicast. These same algorithm is used no matter what kind of routing algorithm is being used (hierarchical, policy, source, virtual circuit). Getting the appropriate behavior, according to the routing algorithm used, requires configuring the tables shown in Figure 3 correctly.

6. It may be desirable to pad this to a 64-bit boundary.

1. If the Tunnel Field is not 0, index into the Tunnel Table using the value in the Tunnel Field, and go to step 2. Otherwise (the Tunnel Field is 0), index into the Logical Router Table (LR Table) with the value in the LR Field, and go to step 3.

2. If the Information column contains forwarding info, then modify the Tunnel Field value according to the instructions in the Information column, and forward the packet. Otherwise, if it contains a pointer to the LR Table, set the Tunnel Field to 0 and go to step 1. Otherwise, if it contains a pointer to a forwarding table, then go to step 4.

3. If the Information column contains forwarding info, then modify the LR Field and Tunnel Field values according to the instructions in the Information column, and forward the packet accordingly. Otherwise, if it contains a pointer to another forwarding table, then go to step 4.

4. Using the RH Descriptor (RHF Offset/RHF Length), isolate the correct RHF and RHFR. Using the RHF, index into the correct forwarding table (determined by the pointer in the previous step). If the Information column contains forwarding info, then modify RHF Offset field, the value of the isolated RHF, the Tunnel Field, and the LR Field value according to the instructions in the Information column, and forward the packet accordingly. Otherwise, if it contains a pointer to another forwarding table, modify the isolated RHF field value according to the instructions in the Information column, and repeat step 4 (using the new forwarding table).

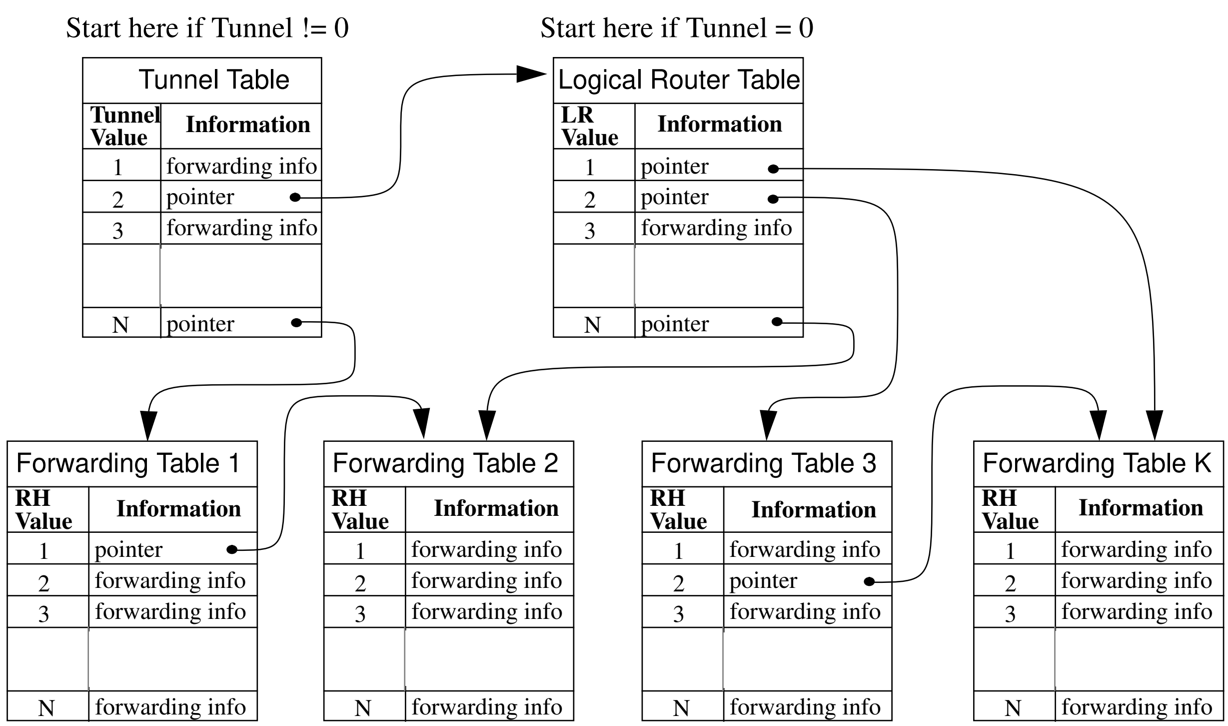

Figure 3: Various Tables Required for Forwarding Pip Packets (only unicast shown)

If tunneling is being used, and the router receiving the Pip packet is not the last router of the tunnel, then the router will find the forwarding information in the Tunnel Table, and not index any other tables. If the router is the last router of the tunnel, and the Tunnel Field has not been set to zero by the previous router, then the router will find a pointer in the Tunnel Table, and forward according to the RH.

If tunneling is not being used, the router receiving the packet will normally find a pointer in the Logical Router Table. When a router finds a pointer in a forwarding table (thus pointing it to another forwarding table), it is normally the result of “routing down the hierarchy”. That is, the router is operating at multiple levels of the hierarchy, and is parsing the hierarchical RH Number.

Section 5 gives examples of the algorithm described above.

Multicast Algorithm

For multicast, the tables in Figure 3 are modified such that the Information column in each table contains a set of information blocks, each one being a pointer or forwarding info. When there are multiple forwarding info blocks (either in the same table entry, or by virtue of multiple pointers reaching multiple tables), then multiple packets are transmitted. Each packet may have the Tunnel or RD fields modified differently, so each information block contains these instructions.

4.3 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 would be queueing directives, such as priority queueing, security directives, such as encryption, and so on.

The meaning of the specific bits is meant to be handled in the same way as the LR—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.

This paper does not further discuss the HD. Most notably, it does not discuss how a dynamic routing protocol would propagate HD information.

4.4 IDs

When an ID is present, it alone is used to identify the source and destination hosts. However, IDs can be mapped to the associated RH, so that the RH implies a certain ID 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 RH Number”. These are the RHFs that describe the “source address” of the source host (see example 1). When Y returns a packet to X, it writes X’s ID in the destination ID field, and X’s Source RH Number in the RH (as the Destination RH Number). This indicates to X that Y has recorded the mapping between X's source RHs and X's ID, and subsequent packets from X that contain the same source RH need not include the ID field.

If the host is mobile, and changes RH Numbers while communicating with another host, then it includes the ID when it uses a new RH Number. This lets the destination host associate another Source RH Number with the ID, so that subsequent packets can again leave the ID off. An out-of-band message can be used to de-associate no-longer-valid RH Numbers. (If both hosts are mobile, then some kind of third party server will be necessary, so that current RH Numbers can be determined, in case both hosts get new RH Numbers simultaneously.) If the hosts get new RH Numbers often, then the ID can simply be included in every packet.

The ID Type field is interpreted as follows. The first two bits indicate the type (and length) of the source ID, and the second two bits indicate the type of the destination ID. The meaning of the four values are: 0 = no IDs; 1 = 32-bit IP number; 2 = 48-bit IEEE 802 number; 3 = 64 bit number. The 64-bit number can have multiple interpretations, including X.121 number, E.164 number, and so on. While the ID field never influences routing, the IP-type ID can be used during transition from IP to Pip to determine how to fill in parts of the RD as the packet traverses the internet.

The ID field is padded out to a 32-bit boundary. It may make sense to pad out to a 64-bit boundary, given the introduction of 64-bit word processors.

4.5 Options

No options are defined at this time. In the future there might be options to establish virtual paths in lieu of policy routes, reserve bandwidth, manage mobile hosts, manage multicast lists, or whatever. In general, I would assume that, if options are present, the packet leaves the normal forwarding code (or hardware) path for special (and slower) processing. Options are not further discussed in this paper.

4.6 Messages

Pip requires the following "ICMP"-type messages:

- Use/don't use tunneling message
- Incorrect RH message (usually means not enough levels of RH Number given)
- Max PDU exceeded notification
- Received ID incorrect (used to flush old RH Number from sending host)
- Normal redirect
- Tunnel redirect
- ARP

The use of these messages are explained by the following examples.

5.0 Examples

Following are descriptions of how various routing and addressing styles are used with Pip. These will further explain the use of the RD.

5.1 Example 1: IP-style Hierarchical RH Numbers (Addresses)

The examples in this section are primarily for the purpose of introducing the various concepts of Pip, particularly the RD. None of the examples are give the complete algorithm, but they get successively more complex and complete. Later examples (Examples 2 and on) will be complete.

Consider the network of Figure 4. The RH Numbers shown correspond to IP-style addressing.

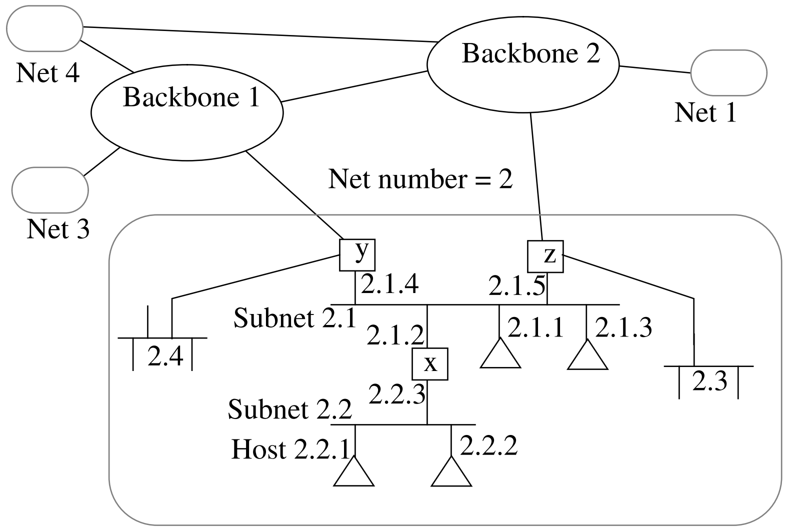

Figure 4: Example 1

The Pip analogue to existing IP and CLNP addressing styles is hierarchical RH Numbers. When plain hierarchical RH Numbers (plain means with no QOS or policy information) are used, the RHFs (and RHFRs) are structured as shown in Figure 5. The first group of RHFs are called the “Source RHFs”. These are separated by “up” RHFRs, and are roughly equivalent to the source address in a traditional IP packet. The second (and last) group of RHFs are called the “Destination RHFs”. These are separated by “down” RHFRs, and are roughly equivalent to the destination address in a traditional IP packet.

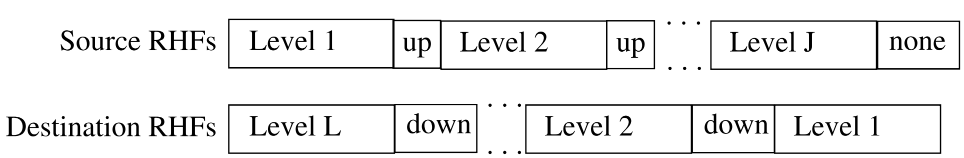

Figure 5: RHFs with Plain Hierarchical RH Numbers (Addresses)

The Source RHFs are listed in order of lowest level of the hierarchy first. That is, this field will come in on the wire first. The Destination RHFs are listed in order of highest level of the hierarchy first. Note that this is the order in which the fields (specifically the Destination RHFs in this case) will be used by routers. The RHFR between the source and destination RH Number indicates “none”.

5.1.1 Example 1.1: No tunneling, no default routing.

Assume that no tunneling is needed, and that default routing is not being used. In other words, the forwarding tables of the routers within the network have network numbers for other networks. The Tunnel Table for router x consists of one entry, indicating that all non-zero tunnel values are invalid. If a Pip packet with a non-zero Tunnel was received, the “Don’t use tunneling” message would be sent to the sender.

The LR table for router x is as follows:

LR table = [<LR.level=3, use FT3> <LR.level=2, use FT2> <LR.level=1, ambiguous>]

For these examples, the only information in the LR Table is that concerning the hierarchical level at which the packet is operating. Since the bits denoting this do not necessarily need to be in the least significant positions of the LR Field, the “LR.level=X” notation implies the index into the LR table.

The reason the LR.level=1 is ambiguous is that router x is attached to two level 1 areas (subnets), and therefore wouldn’t know which level 1 table (FT1a or FT1b) to use. As seen from x’s forwarding tables below, FT2 must first be indexed to determine whether FT1a or FT1b should be used.

The forwarding tables for router x are as follows:

FT1a

I	NH	NL
1	2.2	1
2	2.2	1
3	self	1

FT1b

I	NH	NL
1	2.1	1
2	self	1
3	2.1	1
4	2.1	1
5	2.1	1

FT2

I	NH	NL
1	FT1b	1
2	FT1a	1
3	z	2
4	y	2

FT3

I	NH	NL
1	z	3
2	FT2	2
3	y	3

I = index
NL = next LR level
NH = next-hop (or next forwarding table)

These table are simplified in that they do not show, for pedagogical reasons, information relating to the RHF Relators. This will be shown in later examples.

Example 1.1a: From 2.2.1 to 2.2.2

First consider a packet from 2.2.1 to 2.2.2. Host 2.2.1 would initially make a directory service query and get back an RH Number in the following form: <level 1 = 2; level2 = 2; level 3 = 2>. By comparing its own RH Number with that for the destination, 2.2.1 would conclude that they share the same level 3 and level 2 (that is, are in the same network and subnet). 2.2.1 would then compose the following RD:

RD = < Tunnel = 0; LR.level = 1; RHF Offset = 2; RH = 1 (none) **2** >,

where “LR.level” indicates the bits in the LR field indicating the hierarchical level, and “RH = 1 (none) 2” means that the first RHF is value 1, the second RHF is value 2, and the RHFR between them is “none”.

The source knows to set Tunnel = 0 because of a local parameter indicating that tunneling is not in effect. Normally, a host will assume that tunneling is not in effect unless told otherwise (either by a configuration message or by a “Don’t use tunneling” error message).

The source host initially sets LR.level = 1 because that is the highest uncommon level between source and dest (and therefore a level at which routing must take place). The RH contains the level 1 value from the source (1) followed by the level 1 value from the destination (2). Because the host is setting the RH.level to 1, the host doesn't have to include any RH Number components higher than that in the RH. Since neither value is hierarchically above the other, the RHFR is set to "none". Finally, the RHF Offset is set to point to the beginning of the Destination RHF of the RH (value 2). In all examples, the RHF being pointed to by the RHF Offset will be printed in bold type.

If the host knew that strict subnet-per-LAN IP-style RH Numbering were being used, it could deduce that the destination host is on the same LAN as itself, and ARP for the destination. But assuming that the source host doesn't know this, the source host would send the packet to its "default" router, which is x.

When router x receives the packet, it goes into the LR table with LR.level=1, and determines that the LR is ambiguous in this case. It therefore sends an "LR ambiguous" message to the host. The host would label router x as being ambiguous at level 1, so that future packets (even to different destinations) would start at level 2. Normally, a configuration message from router x (as part of router discovery) would have prevented the need for the error message.

The host composes another RH, this time with level 2 included:

RD = < Tunnel = 0; LR.level = 2; RHF Offset = 3; RH = 1 (up) 2 (none) **2** (down) 2>.

Now, the bottom two levels of the source RH Number (2.1) occupy the first two RHFs (but in reverse order), and the bottom two levels of the destination RH Number (2.2) occupy the last two RHFs.

When router x received this packet, it would index the LR table with LR.level=2, and determine that forwarding table FT2 should be used. Using the RHF Offset, router x would isolate the third RHF (value 2) from the RH. Router x would index 2 into forwarding table FT2, and retrieve a result indicating that it needs to move to level 1, using forwarding table FT1a. Router x would increment RHF Offset, isolate the fourth RHF (value 2) from the RH, use this as an index into FT1a, and determine that the destination is on subnet 2.2. It would then use an ARP function to discover the LAN RH Number of 2.2.2.

Router x would also redirect host 2.2.1. After the redirect, packets from 2.2.1 would go directly to 2.2.2, and would use an RH with only level 1.

To form a return packet, 2.2.2 would reverse the order of the RHFs, and calculate the values of LR.level and RHF Offset similarly to the way that 2.2.1 calculated them. As such, 2.2.2 would copy the level of the incoming packet into the return packet.

Note that the RH for level 1 packets (after the redirect) would only be 1 word long. Only putting as much of the RH Number in the RH as needed is one reason that Pip is compact. Since most traffic is local, most packets will be able to take advantage of this particular optimization.

Example 1.1b: From 2.2.1 to 2.1.3

For a packet from 2.2.1 to 2.1.3, the directory service query would return <level 3 = 2; level2 = 1; level 1 = 3>. By comparing its own RH Number with that for the destination, 2.2.1 would conclude that they share the same level 3 (network), but not the same level 2 or level 1. 2.2.1 would then compose the following RD:

RD = < Tunnel = 0; LR.level = 2; RHF Offset = 3; RH = 1 (up) 2 (none) **1** (down) 3>.

The bottom two levels of the source RH Number (2.1) occupy the first two RHFs (but in reverse order), and the bottom two levels of the destination RH Number (1.3) occupy the last two RHFs. When router x receives this packet, it would parse the packet as described above, go into FT2 with index 1, then go into FT1b with index 3, and route the packet to subnet 2.1.

Example 1.1c: From 2.2.1 to 1.5.11

For a packet from 2.2.1 to 1.5.11 (a host in Net 1), host 2.2.1 would determine that there is no common level, and so would form an RD starting at level 3:

RD = <Tunnel = 0; LR.level = 3; RHF Offset = 4; RH = 1 (up) 2 (up) 2 (none) **1** (down) 5 (down) 11>.

The full three levels of the source address (2.2.1) occupy the first three RHFs (but in reverse order), and the full three levels of the destination address (1.5.11) occupy the last three RHFs. When router x receives this packet, it would go to forwarding table FT3 (based on the RL.level of 3) with an index of 1, and forward the packet to router z without incrementing the RHF Offset or changing the LR.level.

5.1.2 Example 1.2: With default routing, no tunneling

In the previous examples, the level 3 table (FT3), at least in the IP case, would be very large, because it must hold all active network numbers. One way to reduce forwarding table size in general is to use default routing. With current IP networks, default routing works best if there is only one exit point, because since there is only one path out of a private network, default routing doesn't degrade the quality of paths found. If default routing to multiple exits is used, then sometimes a non-optimal exit point can be chosen.

With Pip, tunneling would normally be used to handle default routing with multiple exits. For pedagogical purposes, we give an example here where default routing is used without tunneling (again from the network of Figure 4). The level 1 and 2 forwarding tables for router x (FT1a, FT1b, and FT2) are the same as for Example 1.1. The forwarding table for level 3 (FT3), however, has a single entry of:

FT3 (level 3, tunnel=0) = [*, y, 3],

where * means all possible index values, y means next hop router y, and 3 means the transmitted packet should operate at level 3 (LR.level = 3, RHF Offset = unchanged).

Assume the same host pair as Example 1.1c above (2.2.1 to 1.5.11). Host 2.2.1 would form the same RD as shown in example 1.1.c. Upon receiving this packet, router x would not even need to isolate the RHF, because it knows that all packets at level 3 are routed to y. Assuming that y defaults level 3 packets to Backbone 1, the packet would take a longer path than necessary.

5.1.3 Example 1.3: With default routing and tunneling

Now, we consider the case where tunneling is in use. The level 1 and 2 forwarding tables (FT1a, FT1b, and FT2) for router x are the same as in the first example. There is no level 3 forwarding table. The Tunnel Table (TT) is shown below:

TT

I	NH	NL	NT
1	y	3	1
2	z	3	2

NT = New tunnel value

Note that there is a new column in the table (the 4th column). This is the value the Tunnel field gets written to upon transmission. Note that the Tunnel Table is small (just two entries, one for each exit point). Router x's LR table is modified as follows (to indicate the lack of a level 3 forwarding table):

LR table = [<LR.level=3; error (Send "Use tunneling message")>
<LR.level=2; use FT2>
<LR.level=1; ambiguous>

The Tunnel Table and level 3 Forwarding Table for router y are as follows:

TT

I	NH	NL	NT
1	FT3	3	-
2	z	3	2

FT3

I	NH	NL	NT
1	z*	3	2
2	error	2	0
3	BB1	3	0
4	BB1	3	0

* send tunnel redirect

Example 1.3a: From 2.2.1 to 1.5.11, host fails to use tunnel

Normally hosts would be configured to use or not use tunnels as appropriate (via some router-to-host configuration protocol). Assume for this example though that host 2.2.1 has somehow not been informed to use tunnels for inter-domain (level 3) traffic.

Host 2.2.1 would generate an RD as shown in Example 1.1c. When router x receives this packet, it goes to the LR Table entry for LR.level=3. This results in the error shown. Router x sends an error message to 2.2.1 indicating that it must use tunneling for level 3 traffic.

Example 1.3b: From 2.2.1 to 1.5.11, host uses tunnel value

Now assume that either because of proper configuration or the error message of the previous example, host 2.2.1 knows to use a tunnel for level 3 traffic. Now, host 2.2.1 generates the following RD:

RD = <Tunnel = 1; LR.level = 3; RHF Offset = 4; RH = 1 (up) 2 (up) 2 (none) **1** (down) 5 (down) 11>.

In general, a host will know which Tunnel values are valid, via a configuration message. Barring this, it probably makes sense to have a convention where, lacking better information, a host simply chooses value 1. The routing algorithm could treat this value to mean "route to closest exit point", so that a single exit point doesn't get overloaded with default-tunneled packets.

In this example, host 2.2.1 arbitrarily picks a Tunnel value of 1. Upon receiving this packet, router x indexes into TT by 1 (the Tunnel value), and forwards the packet to router y with no changes in the RD.[7] When y receives the packet, it indexes 1 into its Tunnel Table TT. The resulting entry indicates that the appropriate exit point has been reached (which is y for Tunnel value 1), and that the level 3 (inter-domain) forwarding table FT3 should be consulted. (Alternatively, router x could have written the Tunnel Field to 0 upon transmission to y. In this case, y would go directly to the RH).

For this, router y isolates the appropriate RHF in the RH, which is the 4th RHF (destination network number), value 1.[8] The first entry in FT3 reveals that the appropriate exit point is actually z. Therefore, y puts z's tunnel value (2) in the Tunnel field and forwards the packet to z.[9] Router y also sends a "Tunnel Redirect" message to 2.2.1, indicating that for this particular level 3 value (network number 1), the appropriate tunnel value is 2. As a result, subsequent packets from 2.2.1 to 1.*.* (where "*" means "anything") will go via z.

Discussion

The "Tunnel Redirect" described in Example 1.3a, combined with use of the Tunnel Field, are what make multiple defaults routing work. With multiple defaults routing, the host's relationship with the exit border routers is analogous to a host's relationship with its directly connected (next-hop) routers. In the latter case, the connected router sends a conventional redirect to the host to get to use an alternate router attached to the same network. In the former case, the Tunnel Redirect serves the same purpose with respect to an alternate border router attached to the same stub domain. This is a powerful technique useful for isolating the internal stub routing from external routing.

A few more comments about router y's level 3 forwarding tables is called for. Note first that if router y receives an RD with a tunnel of 2 (FT3a, second entry), it will forward that packet onto z. This would be necessary, for instance, if a host on subnet 2.4 tunneled a packet to z.

If a packet is tunneled to y destined for network 3, y would write the tunnel to 0 (assuming that it didn't subsequently have to tunnel through backbone 1), and forward the packet onto Backbone 1 (FT3b, third entry).

As with router x, router y should never receive an RD at level 3 with a NULL tunnel (except from a mis-configured host). When router y receives a packet from Backbone 1, the RD should indicate level 2, as y's neighbor router in Backbone 1 would know to decrement the LR.level (and increment the RHF Offset) before forwarding a packet to y.

---

7. Note that the tunnel value actually indicates the connected backbone, and not the exit router per se. In this example, there is a one-to-one correspondence between exit router and exit backbone, so it doesn't matter. This is made more clear in the examples associated with Figure 6.

8. I am aware that it is probably confusing to be using the same index value over and over. This is intentional, as it illustrates how RH Number values can be reused for different purposes.

9. Since y is directly connected to z via the LAN, it isn't strictly necessary for y to use tunneling to forward a packet to z. In the general case, where there may be (level 2) routers between y and z, it is necessary to tunnel.

5.1.4 Example 1.4: Using tunneling for policy

This example shows how tunneling can be used as a limited policy mechanism. Later examples will show how full policy information can be encoded in the RD.

For this example, assume that x's and y's level 3 forwarding tables are as shown in example 1.3, and that z's level 3 forwarding tables are structured similarly to y's, except that z uses Backbone 2 to get to Network 1, uses y to get to Network 3, and uses Backbone 2 to get to Network 4. Therefore, there are two ways to get to Network 4, either via Backbone 1 (via y), or via Backbone 2 (via z).

Assume that Host 2.2.1 has a packet to send to a host on Network 4. If host uses a tunnel value of 1, then the packet will travel via Backbone 1. If the host uses a tunnel value of 2, then the packet will travel via Backbone 2. In this manner, the tunnel value acts as a policy mechanism.

Although it is not the best method for getting policy, note that, with the topology of Figure 4, it could be possible for Host 2.2.1 to choose between Backbone 1 and 2 even for sending packets to Networks 1 or 3. This could be done, for instance, by modifying y's and z's routing tables so that they didn't send tunnel redirects, but instead blindly forwarded the packet onto their connected backbones. (This is assuming that Network 2 does not advertise itself as a transit network, and therefore packets would not be routed back to 2, thus causing a loop.)

A variation on this would be to define a bit in the LR to mean "force indicated tunnel", so that if this bit was off, the border routers (y or z) would pick the best path, but if this bit were on, it would override the router's better judgement and force the packet directly onto the backbone as described in the last paragraph. As with all host-initiated policy mechanisms, this requires that the host (or policy server) be knowledgable about the route it is choosing.

5.2 Example 2: Backbone-oriented Hierarchical RH Numbers

It is well-known that IP-style addresses do not scale well. NSAP addresses (at least as defined by RFC 1237 [CGC]) scale better because the addresses are rooted at the backbones.

Figure 6 shows an example topology and backbone-oriented RH Numbers for use with this and subsequent examples. Each backbone has its own number, which is advertised in routing updates to all other backbones. (Hierarchically grouped backbones, for instance, where all backbones in a country are given the same RH Number prefix, are possible, but are not shown in Figure 6.) Note that stub network X has two levels of hierarchy internally, while stub Y only has one.

One of the outstanding problems with the address assignment technique of RFC 1237 is how to handle stub networks that are attached to more than one backbone. One solution is to have multiple RH Numbers, one per attached backbone. This type of solution can be used for Pip. For instance, stub X (and its hosts) is shown to have two RH Number prefixes (1.14 and 26.81), one reflecting its attachment to A and the other its attachment to D. The negative aspects of the multiple addresses solution are not as bad with Pip as with CLNP. Indeed, with Pip, hosts can be completely isolated from inter-domain RH Numbering conventions.

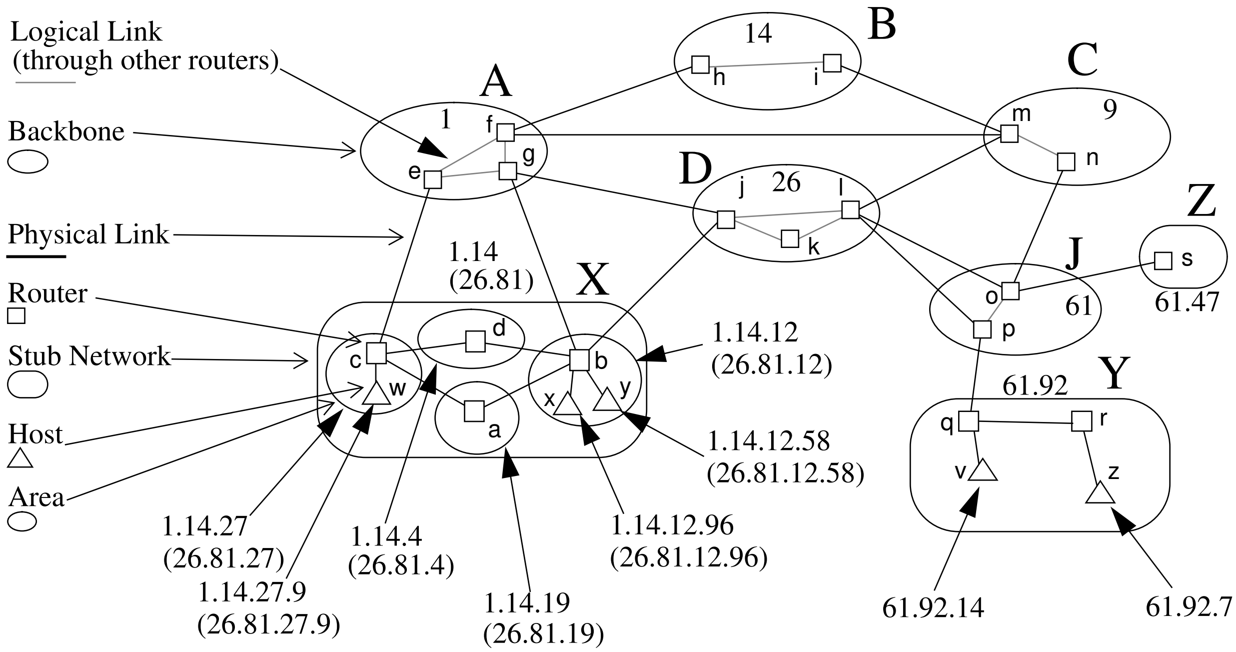

Figure 6: Example Internet Topology and RH Numbers (Addresses)

One reason that multiple RH Number prefixes is easier with Pip is the simple fact that "inter-domain" levels of the RH Number are not included in intra-domain RDs. For instance, the RD for a packet from host w to host y would be:

RD = < Tunnel = 0; LR.level = 2; RHF Offset = 3; RH = 9 (up) 27 (none) **12** (down) 58>.

Neither of the prefixes for stub domain X (1.14 or 26.81) are in the packet. Internal communications are not affected by backbone RH Numbering conventions. Hosts may (or may not) need to know their backbone RH Numbers for inter-domain traffic, and so the functions for reconfiguring these parts of all host RH Numbers may be required. This would be done alongside other host configuration (such as how to use tunnels, etc.), and is not particularly difficult.

Another reason why multiple RH Numbers is less of a problem with Pip is that the transport protocol uses only the ID field for the purpose of labeling connections. This means that the RH Number prefix (or any other part of the RD) can change arbitrarily during a transport connection without effecting the connection.

Appendix A shows the forwarding tables for various routers in Figure 6.

5.2.1 Example 2.1: Inter-domain communications without backbone selection (with tunneling)

For these examples, host x wishes to send a packet to host z, and does not care which backbone (A or D) is used, but would like the routers to choose the best path. Assume that routing will find D as the best backbone for reaching Y from X.

Example 2.1a: Complete host isolation from external RH Numbering conventions.

This example describes a mode of operation where hosts (or internal routers) do not need to know the “inter-domain” components of their RH Numbers (although directory systems still must). This is the extreme case of isolating internal network operation from external influences.

At a minimum, the host must initially know 1) that the stub-domain border routers will handle the inter-domain RH Numbers, and 2) which bit in the LR Field determines that so-called RH-Tunneling will be used to find exit routers. The host must eventually know 1) how many levels of inter-domain RH Number there are, and 2) the minimum RHF length for these levels.

Initially, the host makes its best guess at the number of levels and the minimum RHF length. For example, if host x thought that there was only one level of RH Number above the stub domain, it might create the following RD:

RD = <Tunnel = 0; LR.RH-Tunnel=1; RHF Offset = 3;
RH = 96 (up) 12 (up) **1** (none) 61 (down) 92 (down) 7>.

Note that the host is not using the Tunnel Field per se for this packet. Instead, the use of an “RH-Tunnel” is encoded in the LR Field. The RH-Tunnel number is placed in the third RHF. The entries in the RH-Tunnel forwarding table contain routes to stub exit points. The purpose for using this method of tunneling, which only works for stubs, not for backbones, will become clear later in this example.

The RH-Tunnel value of 1 is just a guess on the part of the host. Since the host has assumed only one level of hierarchy above its own RH Number, it puts one RHF above its known RH Number (21.96). Since this field will need to be written to its correct value by the border router, the RHF Offset initially points to this field.

Through the tunneling mechanism similar to that already described, x will eventually discover a tunnel that will get the packet to router b. Looking at the forwarding tables for router b in Appendix A, we see that router b would first access forwarding table FTt with index 1. This entry contains a pointer rather than forwarding info (as can be seen by the fact that the “next-hop” column is empty). Since the RHFR proceeding the third RHF is “none”, the “none” column in the table is used. The exclamation point (“!”) indicates that this is an error, and that an error message of some sort should be sent. In this case, it is a “Incorrect RH” message indicating to host x that it has not set the correct number of levels in the RH Number.

Upon receiving this message, the host would assume 2 levels of RH Number above the stub domain, and create the following RD:

RD = <Tunnel = 0; LR.RH-Tunnel=1; RHF Offset = 3;
RH = 96 (up) 12 (up) **1** (up) 1 (none) 61 (down) 92 (down) 7>.

This RD shows 4 levels of source RH Number instead of 3. Both source RH Number levels 3 and 4 are filled in with the RH-Tunnel value of 1. When router b receives this packet, it goes through the same steps as before up to the point where it accesses forwarding table FTt, index 1. This time, it refers to the “up” column, writes the RHF to value 14, and increments the RHF Offset (as indicated by the “+”). The question mark (“?”) after the value 14 in the new-value field indicates that a check should be made at this point for sending an error message. In this case, the check is to make sure that the RHF Length is big enough to hold the new value. If it weren't, an error message indicating the correct minimum RHF Length for the inter-domain parts of the RH Number would be sent to the host.

At this point, the RH is as follows:

RH = 96 (up) 12 (up) 14 (up) **1** (none) 61 (down) 92 (down) 7>.

Next, router b goes to forwarding table FT4b index 1, writes the RHF to value 1, checks again for correct RHF Length, increments the RHF Offset, and goes to forwarding table FT4a, index 61. This entry indicates router j (backbone D) as the next hop. The "?" here refers to a check to see if an RH-Tunnel redirect should be sent. In this case the answer is yes, because the RH-Tunnel value of 1 indicates backbone A. The RH-Tunnel redirect would direct host x to subsequently use RH-Tunnel 2 to reach level 4 RH Number "61".[10]

When router b forwards the packet to router j, the RD is as follows:

RD = <Tunnel = 0; LR.level = 4; RHF Offset = 5;
RH = 96 (up) 12 (up) 14 (up) 1 (none) **61** (down) 92 (down) 7>.

The RH-Tunnel bit is not relevant to router j, and its semantics no longer exist in the LR Field. The LR.level has been set at 4, as indicated in the "none" column for FT4a, index 61.

The source RH Number has been filled in by router b. It is as though host x knew its full source RH Number. Note that, in a sense, the wrong source RH Number has been formed. This is because a return packet based on this source RH Number will come back via backbone A instead of backbone D—asymmetric paths. The source RH Number is composed according to the RH-Tunnel value, not according to the actual exit point. Because of the redirect to host x, however, subsequent packets will go via RH-Tunnel=2, and therefore the "correct" source RH Number of 26.81.12.96 will be formed.

Example 2.1b: Partial host isolation from external RH Numbering conventions

Any number of variations on this theme are possible. For instance, hosts could normally not know the inter-domain RH Numbers, but learn them on an as-needed basis.

In this mode of operation, a host could create an RD with RH-Tunnels, as in the previous example, but intentionally incorrectly compose the RD, for instance, by putting no levels above the intra-domain RH Numbers. The error message sent by the border router could include the proper inter-domain RH numbers. In subsequent packets, the host would compose correct RDs, with RH.level = 4 and RHF Offset pointing to the highest-level destination RH Number. This saves the border router from having to work through two extra levels of hierarchy.

The learned inter-domain RH numbers would be used only for the appropriate destination(s), and would be flushed periodically.

Or, the host could operate as in example 2.1a, but when it receives a return packet from the destination host, it can learn the appropriate inter-domain RH Numbers from the Destination RHFs of the received packet. If the host later receives a tunnel redirect (implying that a different outgoing backbone was being

10. Note that from host z's perspective, the RH Number "61" is level 3, not level 4. Hosts always interpret level number based on the number of levels in their part of the hierarchy.

used), the host could again write the inter-domain RH Numbers to zero, thus learning the new RH Number is subsequent return packets.

Once the host learns and uses the correct inter-domain RH Numbers, it may use the Tunnel Field to exit the stub domain.

Example 2.1c: No host isolation from external RH Numbering conventions

This example is quite similar to the previous two examples, except that the function of the border router filling in the proper inter-domain RH Numbers is not used. Instead, hosts are configured with <Tunnel value; matching inter-domain RH Number> tuples, one for each exit backbone. All hosts in stub X would have two tuples: <Tunnel=1; level 4=1, level 3=14> and <Tunnel=2; level 4=26, level 3=81>. A Tunnel value of 1, then, represents exit points that reach backbone A (1), and a Tunnel value of 2 represents exit points that reach backbone D (26). Note that these tunnel values are not pointing to exit routers per se—they are pointing to exit backbones. Therefore, a Tunnel value of either 1 or 2 could cause a packet to go to router b, since it is connected to both backbone A and backbone D.

Since x wants routing to pick the appropriate exit backbone, it creates the following RD:

RD = <Tunnel = 1; LR.level = 4; RHF Offset = 5;
RH = 96 (up) 12 (up) 14 (up) 1 (none) **61** (down) 92 (down) 7>.

The thing about this RD that will force routing to choose the best path (according to routers) is that the RHF Offset points to the destination backbone (61), and doesn't pre-suppose what exit point to use. Presumably, routing will have an opinion about what is the best way to get to backbone 61, and will do the right thing. The Tunnel value (1) was picked arbitrarily.

Looking at the forwarding tables for b (in Appendix A), we see that b will access forwarding table TT (because the Tunnel is non-zero), and index 1. Router b would write the Tunnel value to 0, index the LR Table at LR.level=4, and go to table FT4a, index 61. (This is deduced from the tables shown in Appendix A because the other level 4 forwarding tables, FT4b, is only reached via FT3.)

At this point, the behavior is similar to that of example 2.1a, where a tunnel redirect is sent.

As a result of the tunnel redirect, host x subsequently composes the following RD:

RD = <Tunnel = 2; LR.level = 4; RHF Offset = 5;
RH = 96 (up) 12 (up) 81 (up) 26 (none) **61** (down) 92 (down) 7>.

Discussion

Note that both modes of operation (hosts that do not know the inter-domain RH Numbers and hosts that do) can operate in the same domain using the forwarding tables shown for router b.

Note that the "dumb host" mode of operation (that of Example 2.1a) can work because the ID function has been partitioned from the "routing" function. This allows routers to change aspects of the routing information while still allowing hosts to recognize the source and destination of packets.

I have mixed feelings about the "dumb host" mode of operation. On one hand, the notion of not having to administer inter-domain RH Numbers in machines other than border routers and directory service is appealing. On the other hand, it seems to me that, given the right protocols, it should be easy to manage inter-domain RH Numbers in all hosts and routers. For instance, OSI is in the process of defining a means whereby all hosts in an area can be informed of new NSAP prefixes. This technique is tied to current ISIS and ESIS functions, and is actually quite simple.

5.2.2 Example 2.2: Inter-domain communications with backbone selection, with tunneling.

For these examples, the source host wishes to manipulate the exit backbone chosen, rather than let the routers choose. Note that this use assumes that the host (or user) has the knowledge necessary to choose a backbone that makes sense. For instance, it might be silly for a host to choose backbone A over backbone D, when backbone A forwards the packet onto backbone D anyway.

Example 2.2a: Punching holes, different hierarchy depths, and symmetric paths

As with the previous examples (2.1), host x wishes to send a packet to host z. But, host x wants the packet to go through and return via backbone A. We assume that the hosts in X have the same information as with example 2.1c, that is, that they know which inter-domain RH Numbers are associated with which Tunnel values.

Host x creates the following RD:

RD = <Tunnel = 1; LR.level = 4; RHF Offset = 4;
RH = 96 (up) 12 (up) 14 (up) **1** (none) 61 (down) 92 (down) 7>.

The difference between this and the previous example is that the RHF Offset is set to 4 instead of 5, and is therefore pointing to the highest Source RH Number (1) instead of the highest Destination RH Number (61). As a result, when router b receives this packet, it replicates the actions of example 2.1c, except that it indexes FT4a with value 1 instead of value 61. This retrieves a next-hop of g, which matches the implied Tunnel value, and so no tunnel redirect is necessary.

The RHFR in this case is “none”, and so router b keeps the LR.level at 4. Since the next hop is in backbone A, router b increments the RHF Offset. Note that if router b had not incremented the RHF Offset, router g would have taken the extra step of determining that the RHF (1) indicated itself and incrementing the RHF Offset itself. Router b forwards the packet to backbone A (router g) rather than backbone D, as it otherwise would have.

At this point, it is instructive to follow the packet through the internet to the destination. Router g receives the following RD:

RD = <Tunnel = 0; LR.level = 4; RHF Offset = 5;
RH = 96 (up) 12 (up) 14 (up) 1 (none) **61** (down) 92 (down) 7>.

Router g access its forwarding table FT4, index 61, and routes the packet to router j. (Here we see that, from a purely topological perspective anyway, host x’s choice of A as its backbone does nothing more than incur 2 extra hops.) The RD received by router j is the same as that shown above (router g does not change the semantics of the RD, although it could have modified the bit positions in the LR or HD, if backbone D interprets the bits differently than backbone A).

Note that appropriate exit point from backbone D to backbone J is router l. Within backbone D, however, two ways are shown to get from j to l. By inspecting the forwarding tables for router j, we see that a "QOS" metric determines which way is taken. This metric would be encoded in the LR (along with level), and is used to choose the "Logical Router" (that is, the appropriate forwarding table) for the metric type. Note that this metric example only influences the path inside a backbone. A metric could just as well influence the path of backbones.

For this example, assume that the "QOS" metric bit is 0, and so forwarding table FT4a is used, indexed by 61. This returns a next hop of l, and the RD is not modified. Note that if there are routers between routers j and l, router j would have to tunnel to reach router l.

When router l receives the packet, it indexes 61 into table FT4a. Instead of retrieving an entry indicating that the packet should be routed to backbone J, route l is instructed to look into a level 3 table (FT3b). This is surprising, as the destination stub is not under backbone D. The reason for it in this case is that 1) there are two ways to enter backbone J, and 2) router l would like to pick the most appropriate entry point into backbone J for the given stub. This is analogous to the "east coast/west coast" problem found sometimes in the USA, where a neighbor backbone can be entered on either coast, and more detailed information about the location of the destination is desired to know which entry point to take.

Router l increments the RHF Offset, and indexes 92 into forwarding table FT3b. This entry indicates that router p is the best next hop into backbone J. Note that router l has two level 3 forwarding tables, FT3a and FT3b. It is necessary to separate the forwarding tables for the level 3 destinations within backbone D from those in backbone J. And indeed, it would be necessary to have a separate level 3 table for every level 4 entity whose level 3 details were known. This is in order to distinguish between identical level 3 values in the different level 4 areas.

This form of gathering detailed information about the internal structure of other domains is sometimes called "hole punching", and is a feature of the IDRP routing protocol.

Router p receives a packet with the following RD:

RD = <Tunnel = 0; LR.level = 3; RHF Offset = 6;
RH = 96 (up) 12 (up) 14 (up) 1 (none) 61 (down) **92** (down) 7>.

Router p indexes 92 into forwarding table FT3. This entry returns a next hop of q. Note that the LR.level of the RD transmitted by router p is set to 1, even though it came in as level 3 and even though the RHF Offset was incremented only once. This is necessary because stub domain Y only has one level of hierarchy, and therefore views the "top" of the hierarchy as level 3 rather than level 4. A host in a stub domain will view the top level of the RH Number hierarchy as being the number of levels in its RH Number. This is true whether or not the destination host has the same number of levels.

A router can view the top level of the hierarchy as being any level equal to or greater than the number of levels it is aware of. As such, router g, for instance, could view the top level as level 2. The stub domains would then be level 1. As long as one router translates the level into the proper value for the next router, the level value can be chosen somewhat arbitrarily.

To continue the example, router q receives the following RD:

RD = <Tunnel = 0; LR.level = 1; RHF Offset = 7;

RH = 96 (up) 12 (up) 14 (up) 1 (none) 61 (down) 92 (down) **7**>.

Router q transmits the packet to r, which transmits it to z (the forwarding tables for q and r are not shown).

To form a return packet, host z reverses the order of RHFs, resulting in the following RD:

RD = <Tunnel = 1; LR.level = 3; RHF Offset = 4;
RH = 7 (up) 92 (up) 61 (none) **1** (down) 14 (down) 12 (down) 96>.

The destination RHF pointed to by the RHF Offset (1) signifies backbone A. This means that the reverse path will be symmetric with the forward path (at least at the level of domains).

5.2.3 Example 2.3: General Policy Routing

The previous example showed a small level of policy routing, in that the source host was able to choose the exit backbone. Recent work [BE, LS] indicates that policy routing in general can best be achieved with domain-level source routing. In this example, we show how this can be encoded with Pip.

For general policy routing, but still with hierarchical RH Numbers, the RD is of the form shown in Figure 7.

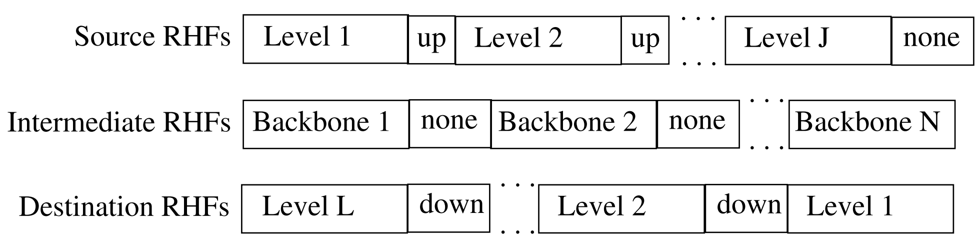

Figure 7: RHFs for Inter-domain Policy and Hierarchical Addressing

In between the source and destination RHFs are the intermediate RHFs. These designate the backbones on the path from source to destination.

Example 2.3a: Choosing the inter-domain path

For this example, assume that host x not only wants the packet to go via backbone A, but to traverse backbones B and C as well. To do this, host x forms the following RD:

RD = <Tunnel = 1; LR.level = 4; RHF Offset = 4;
RH = 96 (up) 12 (up) 14 (up) **1** (none) 14 (none) 9 (none) 61 (down) 92 (down) 7>.

The packet would reach router g similarly to example 2.2a. Router g receives the following RD:

RD = <Tunnel = 1; LR.level = 4; RHF Offset = 5;
RH = 96 (up) 12 (up) 14 (up) 1 (none) **14** (none) 9 (none) 61 (down) 92 (down) 7>.

Instead of pointing to the destination backbone (61), the RD points to backbone B (14). Therefore, router g forwards the packet to router f, which forwards it to router h. When router h receives the packet, it would point to backbone C (9), and so on. The domain path taken by the packet would be X-A-B-C-J-Y.

When host z receives this packet, it knows by inspecting the RD that 14 and 9 are intermediate backbones (because of the “none” RHFRs), and strictly speaking are not necessary for returning the packet to y. If z wanted the return path to be symmetric with the forward path, then it can form an RD by reversing the RHFs. However, if z doesn’t care about the return path, or wishes a different return path, it can remove the intermediate RHFs (14 and 9), and potentially add some of its own.

Example 2.3b: Choosing the intra-domain path

In this example, host w is sending a packet to host z. Host w doesn’t care about the inter-domain path, but wishes the intra-domain path to transit areas 19 and 14 before exiting the domain. To do this, host w forms the following RD:

RD = <Tunnel = 0; LR.level = 2; RHF Offset = 3;
RH = 9 (up) 27 (none) **19** (none) 12 (up) 14 (up) 1 (none) 61 (down) 92 (down) 7>.

When router c receives this RD, it will index 19 into its forwarding table FT2 (not shown, but analogous to router b’s FT2), and route the packet to router a, which will forward it to b (based on an index of 12 into its FT2 forwarding table, also not shown). When router b receives the packet, it will have the following RD:

RD = <Tunnel = 0; LR.level = 3; RHF Offset = 5;
RH = 9 (up) 27 (none) 19 (none) 12 (up) **14** (up) 1 (none) 61 (down) 92 (down) 7>.

Router b will index 14 into its forwarding table FT3, which indicates that it should go to level 4 and route on the next RHF. Note that here is an example where, to save memory, this table could be implemented as a single “wildcard” entry rather than a full table to be indexed into.

When host z receives this packet, it can again either leave in the intermediate RHFs or take them out. In this case, however, the intermediate RHFs are interspersed between source RHFs. This can be detected, however, by inspection of the RHFRs. Assuming that host z leaves the intermediate RHFs in, it would form the following RD:

RD = <Tunnel = 1; LR.level = 3; RHF Offset = 4;
RH = 7 (up) 92 (up) 61 (none) **1** (down) 14 (down) 12 (none) 19 (none) 27 (down) 9>.

When router g receives this packet from backbone D, it forwards the packet to router b. Router b receives the following RD:

RD = <Tunnel = 0; LR.level = 2; RHF Offset = 6;
RH = 7 (up) 92 (up) 61 (none) 1 (down) 14 (down) **12** (none) 19 (none) 27 (down) 9>.

Since the RHFR after the 6th RHF (12) is “none”, router b goes to the “none” column of index 12 in table FT2, increments the RHF Offset, and indexes again into FT2, but this time 19. As a result, the return packet takes the reverse path of the forward packet.

Note that in general for this to work, since backbone A has two ways to reach stub X, backbone A should have hole punching information about stub X. For instance, if backbone A transmits the packet to stub X via router e, then router c would forward the packet to area 12, which would then return the packet to router c via area 19. The packet would not loop more than this once, but none-the-less it is clearly a non-optimal path. This is a natural consequence of doing policy routing without specifying the path adequately, and is not a bug with Pip per se. (Alternatively, to eliminate the need for hole punching information in A’s routers, X could have two level 3 RH numbers under backbone 1. One number would indicate entry via e, and the other entry via g. Each could alternate route to the other in case of node or link crashes making the primary route impossible.)

5.2.4 Comments on Header Size

Even with some policy in the RD, the Pip headers are still relatively (compared to CLNP) small. For instance, assume that there are no more than 1000 top level backbones, and that any hierarchy element has no more than 1000 sub-elements. In this case, the largest RHF is 10 bits. Therefore, the RHFs of Example 2.2 require only 82 bits, or 3 words when padded out to 32-bit words. Including two 6-octet IDs, we get 6 words total (note that not all packets must include the IDs). This can advantageously be compared to CLNP addresses, which require 10 words (two 5-word addresses). The RHFs of Example 2.2, which have a decent amount of policy information in them, require only 106 bits, or 4 words when padded out (7 words when IDs are considered).

5.3 Example 3: Node-level Source Routing

Example 2.3 showed how Pip can do domain-level (or area-level) source routing for policy routing. Other literature [Per2, Che, CG] 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 wished to use Perlman's byzantine routing for internal communications, and to use normal hierarchical RH Numbering for external communications. For external communications, the RH numbering of Figure 6 would be used.

For internal communications, a separate RH numbering scheme is used. In this scheme, each router is given an identifier, counting up from 1. For instance, if a network had 500 routers, they would be numbered 1 through 500, and the RHF for each router would be 9 bits long. Each host would have a number assigned by its connected router. Therefore, even if each router had 500 hosts (for a total of 250,000 hosts), each RHF would still be only 9 bits.

The RD would be composed as follows:

Source Host	up	Router 1	none	Router 2	none	. . .	Router N	down	Dest Host

A separate LR value would be used to distinguish RH numbers in this local scheme from hierarchical global RH numbers. Assuming 9 bits per RHF, Pip can encode a source route of 18 hops plus two 6-octet IDs in the same space required for two NSAP addresses.

5.4 Routing on a path identifier (or VCI number)

There are various advantages to setting up a dynamic path identifier rather than sending full RH Numbering information in each packet. Because part of the forwarding function is to modify the RHF, the RD can be used as a path or virtual circuit identifier. It can also be used as a hierarchical path identifier as with ATM cells.

It might be possible to use an option field in Pip to convey the information necessary to setup a path.

5.5 Multicast Routing

Pip provides enormous potential for increasing the sophistication and efficiency of multicast routing.

For instance, Pip can encode hierarchical multicast routing, where for instance one level of the RH 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 RHF), while in fact the corporation had many multicast groups (multiple lower level multicast RHFs).

Since different applications require different 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 LR Field distinguishing between them.

6.0 Transition from IP

This section outlines an approach for transitioning from IP to Pip.

I presume that the target architecture for Pip is backbone-oriented hierarchical RH Numbers such as shown in Example 2.2. This RH Number structure is essentially the same as what is proposed in RFC 1237 [CGC]. I don’t see any reason to use geographically-oriented RH Numbers, such as proposed by Deering [Ref?}, given that 1) the inter-domain part of RH Numbers can be hidden from stubs, and 2) that with Pip, it is straight-forward to take advantage of backbone-oriented RH Numbers for policy routing. None-the-less, geographically-oriented RH Numbers can be used with Pip, and so the issue remains open to debate.

Because the RH Numbers are semantically equivalent to RFC 1237 NSAPs, it should be possible to use the “CNAT” transition plan being developed by Callon almost as is. The main difference is that Pip will be used instead of CLNP, and RH Numbers will be used instead of NSAPs.

The transition, then, goes roughly as follows:

1. Start running Pip in the backbones.
   a. Modify BGP carry RH Numbers. Once BGP has been modified for general masks as currently planned (BGP4), it will be relatively easy to add RH Numbers, as BGP4 will already have hole punching capability.
   b. An RH Number Authority (perhaps the same authority that assigns IP addresses, or perhaps the Internet Society) will assign RH Numbers to backbones. On one hand, this will result in fewer assignments than are currently done by the IP numbering authority, but on the other hand each assignment will require some screening to insure that the recipient is a valid backbone.
2. Simultaneous with 1, populate border routers with mappings between IP network number and corresponding RH Numbers (i.e., IP net number <=> RH backbone.stub). This is to allow for translation between IP packets and Pip packets at the borders of stubs. These mappings can be distributed using a new BGP attribute.

3. Simultaneous with 1, modify the DNS root servers to issue RH Numbers in addition to IP numbers.
4. One-by-one, modify intra-domain routing to use Pip. Because Pip can use either the subnet/host model of IP or the area/host model of CLNP, and because inter-domain routing information need not be seen within stub domains, both IP and CLNP routing protocols can be modified to carry Pip RH Numbers.
5. Simultaneous with 4, modify the stub DNS servers to issue RH Numbers in addition to IP numbers.
6. One-by-one, modify hosts to run Pip.
    a. At the same time, higher layer protocols such as FTP or TCP that encode IP addresses should be modified to either not require internet-layer identifiers, or to handle multiple types, including Pip IDs. The TCP pseudo-header checksum could be made to include the whole Pip ID.
    b. While any host in a stub is an IP-only host, all Pip hosts should be able to run IP, in order to talk to that host without translation, and intra-domain routing must be able to handle IP or Pip.
    c. Once a stub domain becomes pure Pip (no IP boxes), that stub domain should never have to translate Pip packets into IP packets. The burden of all translations should be up to the stub that still runs IP.

7.0 Further Work

Obviously there is a great deal of work to be done—detailed Pip specification; specification of modifications required to existing protocols, particularly routing but also DNS; development of a transition plan; specification of configuration protocols; establishment of a Pip addressing authority; and experimentation, among others.

While I don't expect anybody to buy completely into Pip based on this paper alone, I hope that this paper convinces most that Pip is an alternative worth expending considerable resources on.

REFERENCES

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

[Che] Cheriton, D.R., "Sirpent: A High-Performance Internetworking Approach", Proceedings of ACM SIGCOMM '89, Austin Texas, September 1989

[CG] Cidon, I., and Gopal, I., "Control Mechanisms for High-Speed Networks", Proceedings of IEEE International Conference on Communications '90, Atlanta Georgia, April 1990

[Chi] 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.

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

[LS] 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.

[OSI2] International Organization for Standardization ISO8473, “Protocol for providing the Connectionless-mode Network Service”

[OSI3] 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)”

[Per1] 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.

[Per2] Perlman, R., “Byzantine Routing”, PhD Thesis, Department of Computer Science, MIT, 19??.

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

Appendix A: Forwarding Tables for Routers of Figure 6

Each table shown is a Forwarding Table or Tunnel Table. The first line gives the table label, followed by the criteria (LR.level, Tunnel, or previous forwarding table) under which the table is accessed. No LR Tables are shown, because the LR Table can be deduced from the criteria that each forwarding table is labeled with.

Within the body of each table, the first column is the index into the table. This index is either derived from the Tunnel or an RHF, depending on which applies for the given table. There are skips in the index values. The intervening index values are not shown when the corresponding network components are not shown Figure 6. Normally, the forwarding tables are well-packed, and all index values are represented.

The action taken after any table access is to either route to the next-hop router, in which case the second column (next-hop) will have an entry, or to access another table, in which case one or more of the three "next-level or next-table" columns will have an entry. The next table chosen depends on the meaning of the RHF Relator after the RHF field. The last column (new-value) is the value written into either the Tunnel or the RHF field, depending on which applies, upon transmission of the packet. In practice, both the Tunnel value and RHF may be modified, but for these examples, it is always only one or the other.

A plus (+) after any entry in these four columns means that the RHF Offset should be incremented (either before transmitting the packet or before accessing the next table). A blank entry simply means that the circumstances under which the entry has been reached should not occur. This may or may not result in an error message. An exclamation point "!" after any entry means that the entry might validly be reached, but that an error message should be sent. A "?" after any entry means that additional checks will be made to determine if an error message is necessary (the text will explain these as they are encountered). An entry or RH (in a tunnel forwarding table) means to evaluate the RH from scratch.

Router b:

FT1:	LR.level=1, RH-Tunnel=0				
index	next-hop	new level or next-table: down	none	up	new-value
58	y				58
96	x				96

FT2:	LR.level=2, RH-Tunnel=0				
index	next-hop	new level or next-table: down	none	up	new-value
4	d	1+	2	3+	4
12		FT1+!	FT2+		12
19	a	1+	2	3+	19
27	d	2	2	2	27

FTt:	RH-Tunnel=1				
index	next-hop	new level or next-table: down	none	up	new-value
1		!	!	FT4b+	14?
2		!	!	FT4b+	81?

FT3:	LR.level=3, RH-Tunnel=0				
index	next-hop	new level or next-table: down	none	up	new-value
*		FT2+	!	FT4a+	*

FT4a:	LR.level=4, RH-Tunnel=0				
index	next-hop	new level or next-table: down	none	up	new-value
1	g?	3+	4+		1
9	j?	4	4		9
14	g?	4	4		14
26	j?	3+	4		26
61	j?	4	4		61

FT4b:	From FTt				
index	next-hop	new level or next-table: down	none	up	new-value
1			FT4a+		1?
2			FT4a+		26?

TT:	Tunnel	
index	next hop or new table	new-value
1	LR Table	0
2	LR Table	0

Router g:

FT4: LR.level=4					
index	next-hop	new level or next-table			new-value
		down	none	up	
1		FT3	FT4+		1
9	f		4		9
14	f		4		14
26	j	3+	4		26
61	j	4	4		61

(FT3 and Tunnel not shown)

Router j:

FT4a: LR.level=4, "QOS"=0					
index	next-hop	new level or next-table			new-value
		down	none	up	
1	g	3+	4		1
9	l	4	4		9
14	g	4	4		14
26		FT3			26
61	l	4	4		61

FT4b: LR.level=4, "QOS"=1					
index	next-hop	new level or next-table			new-value
		down	none	up	
1	g	3+	4		1
9	k	4	4		9
14	g	4	4		14
26		FT3			26
61	k	4	4		61

(FT3 and Tunnel not shown)

Router l:

FT4a: LR.level=4, "QOS"=0					
index	next-hop	new level or next-table			new-value
		down	none	up	
1	j	4	4		1
9	m	3+	4+		9
14	m	4	4		14
26		FT3a			26
61		FT3b+	4+		61

FT4b: LR.level=4, "QOS"=1					
index	next-hop	new level or next-table			new-value
		down	none	up	
1	k	4	4		1
9	m	3+	4+		9
14	m	4	4		14
26		FT3a			26
61		FT3b+	4+		61

FT3b: From FT4a or FT4b					
index	next-hop	new level or next-table			new-value
		down	none	up	
47	o	3	3		47
92	p	3	3		92

(FT3a and Tunnel not shown)

Router p:

FT4: LR.level=4, "QOS"=0					
index	next-hop	new level or next-table			new-value
		down	none	up	
1	l	4	4		1
9	o	4	4		9
14	o	4	4		14
26	l	3+	4+		26
61		FT3			61

FT3: LR.level=3 or from FT4					
index	next-hop	new level or next-table			new-value
		down	none	up	
47	o	3	3		47
92	q	1+			92

(Tunnel not shown)