Addressing in Internetwork Protocols

Paul Francis

PhD Thesis

University College London

September 1994

Abstract

The most important part of an internet protocol is its addressing information—that is, the information that affects routing of an internet packet. While there has been much research of routing in internetworks, there has not been a comprehensive study of addressing information in internet protocols per se. In this thesis, we examine the use of addresses in internet protocols. We start with a taxonomy of addressing functions. Using this taxonomy, we then give a comprehensive description of addressing modes in internetwork protocols. Finally, we present two designs for addressing in internetwork protocols, one based on current internet protocol syntax (SIPP), and one based on a new syntax (SPip). Both of these designs exploit the notion that virtually all routing and addressing semantics can be achieved through the loose source route mechanism, though SPip does this more generally than SIPP. We analyze the capabilities and costs of SIPP and SPip, and compare them with those of OSI's internetwork protocol, CLNP. We show that the general use of the loose source route mechanism is the best way to achieve flexible, efficient, and evolvable routing and addressing.

Acknowledgements

I would like to thank Ramesh Govindan, Satoshi Ono, and Zheng Wang for their help in document preparation. Anybody who has ever put together a large document knows that even a little help in this area goes a very long way.

I would like to thank Steve Casner, Lyman Chapin, Joel Halpern, Bala Rajagopolan, and Benny Rodrig for the information they provided me.

I would like to acknowledge the contributions to this thesis of Steve Deering, Ramesh Govindan, and Sue Thomson. Steve is the author of the SIP protocol, which later merged with Pip to become SIPP. The header format of SIPP is that of the original SIP. The influence of SIP on SPip requires no explanation. Ramesh and Sue were both part of the "Pip Team" at Bellcore, and contributed many ideas and lines of code.

I would like to thank my examiners, Deborah Estrin and Chris Mitchell, for their helpful comments (and, of course, for being my examiners).

I would like to thank NTT, and particularly Shigeki Goto, for supporting me during the final months of my thesis work.

I would like to thank Bellcore for its assistance, financial and otherwise, during the bulk of my time with UCL. Particularly I want to thank Dave Sincoskie for his tremendous professional support and encouragement. I hope it yet pays off for him. I want to thank Liang Wu also for his professional support but especially for his personal support at the time I needed it most. I would also like to thank Neil Haller, Sue McDonald, Stu Personick, and Mary Wardell. Without the assistance of the folks at Bellcore there's no way any of this would have happened.

Finally, I would like to thank my advisor Jon Crowcroft for all kinds of things (such as many fine evenings in the pub and introducing me to the four-note guitar jazz chords). I suppose (or at least hope) that he knows what all the things are, so I do not need to go into detail here. Not only did he make it possible for me to do this, he made it virtually painless and indeed largely a pleasure.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the mid to late 1970's, there was a flurry of creative activity that gave us the basic underpinnings for the connectionless internet. Almost 20 years later, this internet architecture is dominating the data networking world. At the time the internet architecture was being invented, a lot of thought went into the design of the internet protocol (see Chapter 12). This work culminated in Xerox's Pup protocol [9], and shortly afterwards in IP (the Internet Protocol) [86, 87].

Since then, much work has been done related to internetwork protocols, in routing algorithms, router performance, network management, and the like. Until recently, however, virtually no work had been done on internetwork protocol header design per se. That is to say, nobody had asked the question "given what we've learned in the last 20 odd years, what fundamental general statements can we make about the nature of internetwork protocol header design, and how might this lead to a better internetwork protocol?"

This is not to say that there has not been work in internetwork protocol design in the intervening years. For instance, during that time CLNP was developed in ISO [55]. However, these works mainly codified existing practice in a standards or proprietary framework, making small but incremental improvements (though some might say the progress was backwards).

With the recent explosion in growth of the IP internet, IP has been found wanting (through no fault of the original designers—IP has far exceeded its expected useful lifetime). IP's address is simply not big enough [45]. This has resulted in a recent flurry of activity to define IP's replacement [64, 30, 37].

Given the lack of work on fundamental internet header design and the renewed interest in a new internetwork protocol, this is a good time to revisit, in general terms, the topic of routing and addressing in internetwork protocol design. That is the topic of this thesis.

More specifically, we ask "what information must be in an internetwork protocol header to support

routing, and how is that information best encoded?"[1] In particular, we present two new designs for encoding routing information. Both designs exploit the use of the loose source route as a general mechanism for a wide range of routing and addressing functions. One design, SIPP, is based on current internet protocol syntax. The other design, SPip, uses a new syntax designed around the loose source route mechanism. We show that both designs satisfy a wide range of requirements, and that both are superior to existing approaches.

We assume that the reader has a basic understanding of internetworking and internetworking terminology—for example, as described in [68].

1.1 Outline

This thesis has two parts. The first part presents a taxonomy of routing information, particularly addresses, and discusses internetwork protocol routing functionality in the context of that taxonomy. The second part describes the two new internet protocol designs. These protocols, plus an existing internetwork protocol (CLNP) are analyzed and compared.

Part I contains Chapters 2 through 6. Chapter 2 concerns taxonomy. It describes two previous taxonomies, discusses their shortcomings, proposes a new taxonomy, and shows how the new taxonomy cleanly describes existing routing and addressing functionality. Chapters 3 through 6 describe in detail various routing and addressing functions, in the language of the taxonomy of Chapter 2. Part I provides the underpinning from which the protocol designs of Part II can be discussed and understood. Part I also serves as a broad survey of the state-of-the-art.

Part II contains Chapters 7 through 11. Chapter 7 presents the criteria by which the three protocols are analyzed. Chapter 8 describes the three protocols. Chapter 9 analyzes the capabilities of the three protocols. Chapter 10 analyzes the costs of the three protocols. Chapter 11 summarizes Part II and presents the conclusions reached.

Chapters 1 and 12 fall outside of the two Parts. Chapter 1 is the introduction, and Chapter 12 is the epilogue.

[1] Note that we do not address other aspects of internetwork protocol header design, such as checksumming, fragmentation, hop count, and congestion control.

Part I

Internet Protocol Taxonomy and Functionality

Chapter 2

Taxonomy

This chapter first describes previous taxonomies of routing and addressing and discusses their shortcomings. It then suggests a new taxonomy, and shows how that taxonomy supports existing routing and addressing functionality.

2.1 Previous Taxonomies

It is impossible to discuss the definition of address without introducing the taxonomy of names, addresses, and routes discussed by Shoch in 1978 [99], and further discussed by Saltzer in 1982 [96]. Shoch states that the name, address, and route represent the fundamental components of networking. Specifically, Shoch makes the following definitions:

- The name of a resource indicates what we seek,
- an address indicates where it is, and
- a route tells us how to get there.

The notion here is that a name is something with which a human can deal comfortably, such as a character string. The name is mapped into an address, which is something less comfortable to humans but more useful to machines, such as routers. The address then maps into a route, which defines the path from one host to another.

The notion of an address indicating 'where' satisfies our common usage of the word address. Hosts (or telephones, or people in houses) that are in the same locale tend to have similar addresses. When a host (or phone, or person) moves, it gets a new address. By and large, network addresses have these characteristics.

However, the notion of an address indicating where something is can also be misleading. For instance, an Ethernet address (or any flat address), seems not to convey much of a notion of location. If an Ethernet interface is moved from an Ethernet in New Jersey to an Ethernet in Tokyo, it keeps the same address. Thus, the Ethernet address behaves more like an identifier (that uniquely identifies the Ethernet interface among all Ethernet interfaces) than an address.

The notion of an address indicating location is also misleading when an address is hierarchical and the addressing hierarchy is not a strict tree (that is, a single element of the hierarchy falls under multiple higher elements). An example of this is provider-rooted addressing. This is a form of addressing where the top level of the hierarchy indicates a backbone service provider, such as NSFNET or PSI, etc. Provider-rooted addressing is being proposed in some of the proposals for the next generation of IP [30, 37].

With provider-rooted addressing, if a subscriber network is attached to multiple backbones, it obtains multiple addresses.[1] Assume for instance that a subscriber network is attached to two providers, A and B, and therefore the hosts in the network each have two address prefixes A and B. When address prefix A is used to address a host in the network, the packet is routed through provider A. Likewise, when prefix B is used, the packet is routed through provider B (see section 3.4.2).

The notion of an address indicating location is misleading here in two senses. First, the host seems to be in one place (for instance, it may have one network attachment point), and yet it has two addresses, which would imply that it is in two locations. More importantly, the address used in a packet to the destination dictates, at least at the specificity of which provider is chosen, the route to the destination. In other words, the address in part determines *how* a packet gets to the destination, and is therefore, by Shoch's taxonomy, also a route.

These ambiguities in Shoch's taxonomy indicate that a more precise definition is needed. (To be fair, Shoch's paper sheds a great deal of light on what an address is, by pointing out different kinds of addresses and how they behave. It is just that the notion of "where" per se is ambiguous.) Saltzer also recognized ambiguities in Shoch's taxonomy, and so designed one of his own [96].

Saltzer observed that there are actually four entities that commonly appear in networks, and that therefore a tripartite taxonomy is insufficient. These four entities are *users/services*, *hosts*, *network attachment points*, and *paths*. Each entity has a name, and each is bound to the next—users/services can be found on certain hosts, which have certain network attachment points, to which paths lead.

[1] Strictly speaking this does not have to be true. If backbone routers maintain routes to subscriber networks, rather than only to other provider networks, then one address would suffice for routers to know how to route to a subscriber through multiple providers. This, however, defeats one of the purposes of provider-rooted addresses, which is for routing to scale at the rate of providers at the top of the hierarchy. Current IP scales at roughly the rate of subscribers, which is found to be inadequate [45].

Saltzer made two important points. First, that the form of the name of each entity should be distinguished from what the entity is (Shoch thought of a name as being human friendly and an address as being machine friendly). Second, that the nature of the binding between the four entities is of critical importance. For instance, is the binding of the name of a host and the name of its network attachment point tightly coupled or dynamic? If it is tightly coupled, as with an Ethernet address, the network attachment point name stays with a host when the host moves. If it is dynamic, as with IP addresses, the network attachment point name changes with host movement, and the binding is dynamically maintained in tables (for instance, DNS tables).

In comparing his taxonomy with Shoch's, Saltzer makes the following statement: "An address of an object is a name of the object it is bound to. Thus, an address of a service is the name of some host that runs it. An address of a host is the name of some network attachment point to which it connects," and so on. In essence, Saltzer takes away the ambiguity of what an address is by removing it from the taxonomy, and delegating it to its classical meaning in computer science.

Thus, according to Saltzer's taxonomy, an address could be the name of a host (as in Ethernet), or it could be the name of a network attachment point (as in IP), or it could even be the name of a path (as in IP source route). In the end, Saltzer does not attempt, as Shoch did, to come up with a unifying definition of address.

In this thesis, we also do not attempt to come up with a unifying definition of address. But neither do we remove address from the basic taxonomy. Rather, we try to shed as much light as possible on what an address is, and how it contributes to the job of routing. In so doing, we take advantage of (but do not completely embrace) Saltzer's taxonomy of user/service, host, network attachment point, and path, because indeed these are real elements in the network.

But, we also keep Shoch's "name" and "address", because those are also real elements in the network. (We keep Shoch's "route" as well, but this is the same as Saltzer's path.) People and hosts have names that we deal with as real objects, and hosts and network attachment points have addresses that are real objects carried in packet headers or looked up in directories.

2.2 Additional Taxonomy: Defining Functions rather than Elements

Both Shoch's and Saltzer's taxonomies are based on network elements, and cover all aspects of naming/addressing/routing (from the user to the path). This section expands those taxonomies by basing the taxonomy on function rather than network (or header) element. It also limits itself to that information in the packet header that relates to routing.

This section shows that there are three route-affecting functions of a packet header. They are; *destination identification*, *destination location*, and *path modification*. Thus, the three route-

affecting components of the packet header are the *identifier*, the *locator*, and the *modifier*.[2] Briefly stated, the identifier distinguishes the destination among all other destinations. It can be used by the routing function to determine how to route a packet, and has the characteristic that its value is independent of where in the network the destination is located. The locator is used by the routing function to determine how to route a packet, but its value does depend on where in the network the destination is located. A locator has the side-effect of identifying the destination. There are two major types of locators, *source-sensitive* and *source-insensitive*. A modifier is a header element that is independent of the identity or location of the destination, but still influences the path taken to reach the destination.

The remainder of this section discusses various components of the header in the context of these three functions.

2.2.1 Addresses

An address does two things. First, it identifies the addressed object (destination host or network attachment point) among the set of addressable objects. (Note that an address can identify multiple objects, as with IP multicast addresses [27]. In this thesis, when not otherwise stated, the term address refers to the address in its "unicast", or non-multicast, form.) In this sense, it does the "naming" function as defined by Shoch, though it is not a name is Shoch's sense (that is, it is not necessarily human-friendly).

The second function of the address is to aid in routing the packet (or call setup) from wherever it is to the destination. Simply put, it does this by providing whatever information is needed by the routing function.

The reason that a single address typically accomplishes both functions is because usually one and only one destination is at a given place at a given time. Thus, indicating "where" the destination is is paramount to identifying it.

In theory, the address could be split into two components, an identifier (indicates what) and a locator (indicates where). The former would be used only for identifying the destination, and would not change even if the destination moved far enough to justify getting a new "locator". The latter would only assist the routing function[3]. Indeed, this is sometimes done in practice, for instance in the case of mobile IP [104, 58], where the network location of a destination can change multiple times during the course of a TCP connection.

Consider the Ethernet address in the context of locating and identifying. The Ethernet address is only an identifier. When the Ethernet packet is transmitted onto the cable, it is delivered to

[2] The term "locator" was first suggested by Frank Kastenholz on the big internet mailing list [7].

[3] Note that the word "router" would be a better term than "locator" for the address because of its role as aiding the routing function. However, the word "router" is already in common use to refer to the physical switches that forward internet packets.

all Ethernet interfaces, each of which individually determines whether or not the packet is for it. That is, each interface "identifies" the packet as being for it or not for it.

This classification is less clear when an Ethernet packet is transmitted on a cable that is connected to other cables via bridges. In this case, it is received by the bridges (as well as by everything else attached to the cable), which then determine the location (from their individual perspectives) of the destination indicated by the Ethernet address. Thus, some "locating" is going on, but it is not the Ethernet address that is locating the destination, but rather the path, as instantiated by the entries in the bridge's forwarding tables, that locates the destination. The Ethernet address identifies the forwarding table entry within each bridge.

This may seem to be splitting semantic hairs, and one could argue that the Ethernet address, in the case of the bridged Ethernet, is in fact locating the destination, albeit indirectly through the mechanism of the forwarding tables in bridges. However, one must draw a line between locating and not locating, and it seems that a clean place to draw it is according to the following definition:

> If the address (or other header component) of a destination remains the same no matter which switch interface the destination directly obtains access through, then the address is not a locator.

Now consider the IP address, which according to the above definition is a locator (but which, as stated above, effectively identifies as a side effect). The IP address is hierarchically partitioned into three levels—network, subnet, and host [87]. (Actually, recent advances in classless IP address assignment [42] add additional hierarchy to the IP address. For the purposes of this thesis, however, the old model of IP suffices.) While exceptions to the following rule apply, by-and-large routers outside of a given network maintain a forwarding table entry indicating how to route to that network. Routers inside the network maintain forwarding table entries for the subnets within that network, and routers on the subnet maintain forwarding table entries for the hosts on the subnet.

Viewed another way, the information inside routers result in three kinds of paths in the IP Internet. There are paths from all routers to all networks[4], there are paths from routers in a network to all subnets in the network, and there are paths from all routers on the subnet to all hosts on the subnet

The job of the address, then, is to string these three paths together into a composite path that will reach the destination from any point in the internet. The address does this by implicitly coupling the three paths together by including them in the same address. If the destination changes its network access location to a different router interface, and that interface is far enough away from the former access point, then the destination requires a new address. (In the case of IP, "far

[4]Strictly speaking, most routers do not maintain routes to all networks. Most routers maintain so-called default routes to routers high up in the physically topology, for instance in provider networks, which in turn maintain routes to all networks.

enough" is a different subnet, for instance a different LAN.)

It is interesting to observe that the IP address essentially consists of a series of identifiers—the network identifier, the subnet identifier, and the host identifier. Each individual identifier behaves similarly to the Ethernet address. That is, each individual identifier identifies a path to a destination, and the destination itself. The "destination" in this case can be a set of systems rather than a single system (for instance, an entire network).

Similarly to an Ethernet address, if the identified "destination" moves, it does not require a new address. For instance, if an entire IP network moves from one part of the Internet to another, it does not have to obtain a new network identifier (called a "network number" in IP), and none of the hosts inside of the network need to obtain a new address. Likewise, if a whole subnet moves within a network, it also does not need a new identifier.

2.2.2 Source Route

A source route, as used in IP (and CLNP), is a series of IP (or NSAP) addresses in the header. A source routed IP packet visits each of the systems identified by the IP addresses in the source route in the order that the IP addresses are listed in the IP header.

There are two kinds of source route: the strict source route and the loose source route. With the strict source route, only those systems listed in the source route can be visited. With the loose source route, systems other than those listed in the source route can be visited. In other words, the strict source route completely specifies the path, while the loose source route only specifies part of the path.

The IP source route can be partitioned into two parts, the last element of the source route (which is the destination address), and the preceding elements. The two parts have different roles in the source route, and must be classified separately.

The final element of the source route is the destination address, and is therefore a locator.

The preceding elements of a source route, taken as a whole, is a kind of locator. This is easy to see in the context of the strict source route. With a strict source route, if the destination moves, it is almost certain that at least some of the preceding elements of the strict source route must change. This is because a change in the destination's location requires a different path. Since the strict source route describes the complete path to the destination, it must change when the path changes.

The classification of the preceding elements of a loose source route as a locator is less obvious. Strictly speaking, the destination can move to a new location without changing the preceding elements of the loose source route (only the final element will change), and the packet will still be delivered. In this sense, the preceding elements of a loose source route seems to behave as a

modifier. That is, it affects the path to the destination, but does not affect the destination itself.

This having been said, if the destination moves, it is likely that the preceding elements of the loose source route would change also, because the previous source route might become a poor one. Thus, the preceding elements of a loose source route are coupled, albeit loosely, to the final element or destination.

We can further support the statement that the loose source route is a locator by showing the similarity between the loose source route and the hierarchical address. Consider again the IP address. As mentioned before, the IP address is essentially a series of identifiers coupled together. Each identifier identifies the next lower element in the hierarchy.

The fact that an IP address (or any hierarchical address) is a series of identifiers means that its syntax could be a series of equal-sized identifiers, rather than nested hierarchical numbers squeezed into 32 bits. Each of the identifiers could even be globally unique if the identifiers were large enough (for instance, 48-bits in length, as are Ethernet addresses).

When viewed as a series of identifiers rather than as a single "hierarchical address", an IP address has some characteristics similar to a source route. Just as the systems represented by the elements of a source route are visited on the way to the destination, the "systems" represented by each of the components of a hierarchical address are normally visited by the packet on the way to the destination.

In the case of the hierarchical address, the "system" is in fact a collection of systems, such as a network or a subnet. Also, the mechanics of parsing a source route are different from the mechanics of parsing a hierarchical address. None-the-less, their basic behavior is the same.

Viewing the hierarchical address as a source route, consider that the "higher" elements in the hierarchical address correspond to the initial elements of the source route, and the "lower" elements of the hierarchical address correspond to the later elements of the source route. If a destination moves a short distance, it is likely that the lower elements of the hierarchical address will change but not the upper. For instance, if a destination moves within the network, then its subnet number might change, but not its network number. Likewise, with a source route, if a host moves a short distance it is more likely that the latter elements will change than the earlier.

The hierarchical address is in fact a specialized form of the source route. The source route, being more general, could be used as a mechanism for achieving hierarchical addressing. When used in its traditional role, however, the source route and hierarchical address have a fundamental difference. That is, the source route is dependent on the location of both the source and the destination, while the hierarchical address is dependent only on the location of the destination. This is because the source route specifies a path from source to destination, whereas the hierarchical address only specifies the "path" from the top of the hierarchy to the bottom.

The difference between the two is important, because it is in part the source independence of the

hierarchical address that gives it its good scaling characteristics. Thus, it is useful to distinguish between source-sensitive and source-insensitive locators in the taxonomy. Hierarchical addresses are source-insensitive locators, while source routes are source-sensitive locators.

To emphasize the difference between the two types of locators, we use the term *vector* to describe source-sensitive locators. Thus, a source route, as used in IP, is classified as a vector. Although strictly speaking a vector is a type of locator, the term "locator", when used alone, implies a source-insensitive locator.

2.2.3 Type-of-Service (ToS) / Quality-of-Service (QoS) Field

Another element of an IP header that can influence the route is the ToS Field. (The analogous field in CLNP is called the QoS Field. We use the term QoS Field when discussing CLNP, and use the term ToS Field otherwise.)

The ToS Field in IP instructs the network to attempt to give the packet a small set of service characteristics, within the limitation of "best-effort" delivery. Examples of ToS Field types in IP are low delay, high bandwidth, and low error. One way an IP network could theoretically provide these services are by routing the packet over transmission facilities that have the requested characteristics. Thus, the ToS Field can affect routing.

The ToS Field is neither an identifier nor a locator. It is not an identifier because the choice of ToS Field has no influence on which destinations receive the packet. It only influences the path taken. Packets to different destinations can have the same ToS Field value, and packets to the same destination can have different ToS Field values. The ToS field is not a locator because it does not change if the destination moves to a different network location.

Rather, the ToS Field is a path modifier (or just modifier for short), the third term in the functional taxonomy. A modifier is a header element that is independent of the identity or location of the destination, but still influences the path taken to reach the destination.

2.2.4 Mobility

There exists several proposals for mobility in IP [104, 58]. While these proposals differ in detail, they all have one thing in common—that is, two IP addresses are used rather than one. One address is stable throughout a higher level connection (such as TCP), and the other address reflects the mobile host's current location in the internet.

The stable address is the identifier for the connection. It does not change even as the mobile host changes location[5]. Except for possibly the last router in a path, this identifier is not examined by

[5]The stable address (identifier) can simultaneously be a locator for the "home station" of the mobile host. The home station is a node that knows the current location of the mobile node.

routers.

The non-stable address is the true locator for the mobile host. This address is used to route the packet to the current location of the mobile host.

2.2.5 IP Multicast

The final component of the IP header that can influence routing is the source address. The source address influences routing when the destination address is a multicast address [27]. A multicast address is an IP address that identifies multiple destinations, rather than a single destination as is the case with the non-multicast, or unicast, IP address. A multicast address also activates a delivery service whereby all destinations receive the packet.

One technique for delivering an IP packet to multiple destinations is for a multicast routing algorithm to form a spanning tree from the source to the destinations. Routers at branches in the tree replicate the IP packet and transmit one copy over each outgoing branch of the tree. Because the tree is rooted at the source, and because a given router can be on multiple trees for the same group, the source address must be examined for the router to know which neighbors should receive replicas.

According to the taxonomy, the IP multicast address in the destination address field is an identifier. It remains the same no matter where the destinations are (or which destinations belong to the group).

The source address, on the other hand, is a modifier. It influences the path(s) taken, but does not influence the set of destinations that receive the packet.[6] To be clear, the source address is a modifier for the destination. The source address itself is of course a locator of the source, and will change if the source changes location. The focus here, however, is on the destination and on how packets are routed to the destination. With respect to the multicast destination, the source address is a modifier.

Note that there are other forms of multicast [2, 79] that do not form trees at the packet source, and therefore do not depend on the source address. In these cases, the source address has no influence on the route taken, and so does not fall into the taxonomy one way or the other.

Multicast Scoping

Scoping in multicast is the act of limiting the spread of a multicast, usually with respect to distance from the source. To accomplish this, there is a field in the packet header that specifies the scope of the packet, here called the scope field. In the case of IP, this field is the hop count (Time-To-Live)

[6] The exception to this is where a destination wishes to filter all packets from a given source. However, this should be modeled as a filtering function rather than a routing function, and so is orthogonal to this taxonomy.

field [31]. That is, the multicast packets simply travel (away from the source) until the hop count expires. The larger the hop count, the larger the scope of the packet.

In the SIPP protocol [30, 40], there are bits set aside in the address field to indicate scope. The scope bits are set independently of the multicast group identifier.

In either case, it is a little difficult to classify the scope field as identifier, locator, or modifier. By itself, the scope field is none of the three. In combination with the source address and multicast address, however, it makes up part of the identification function. That is, taken together, the source address, multicast address, and scope field define the recipients of a given packet. In this sense, it is neither a coincidence, nor inappropriate, that the SIPP scoping bits are part of the multicast address itself.

2.2.6 Flow Specification / Flow Identifier

"Connection-oriented" network technologies, such as ISDN [17] and X.25 [16] have two phases of operation; call setup and data transfer. In the call setup phase, one or more call setup packets are sent from source to destination (and possibly back again, depending on the protocol). The call setup contains full "addressing" information, similar in function to that information in the header of an IP packet, which is used to determine the path from source to destination.

The call setup may also contain information indicating what service the subsequent *flow* (traditionally called a connection) requires. Such information can be as simple as peak bandwidth required, or can have multiple parameters describing average bandwidth, peak bandwidth, delay sensitivity, and loss sensitivity [75]. In line with the terminology of [75], we call this information the flow specification, or *flow spec*.

The call setup may also contain information that tells routers how to associate subsequent data packets with the routing and flow spec information. This is called the *flow filter* in the RSVP call setup protocol [114].

Traditionally, for instance with X.25 and ISDN, the flow filter is a small identifier that is locally managed by each router. We call this identifier the *flow ID*.[7] The Flow ID can also be managed by the source host, such as with SIPP and SPip (Sections 8.2.5 and 8.1.7 respectively). The flow filter may even have no flow ID per se, but instead identify the fields in an otherwise connectionless packet header that uniquely identify the flow—for instance, source and destination address, protocol number, and port number.

The flow spec in the call setup may or may not influence the path chosen. A significant amount of literature exists on the topic of optimal path selection based on flow requirements, of which the work of Bertsekas and Gallager is exemplary [5]. However, optimal path selection is an exceedingly

[7] X.25 uses the term Logical Channel Number (LCN). ATM uses the term Virtual Circuit Identifier (VCI). ATM uses two nested levels of VCI. It calls the lower level a VCI, and the upper level a VPI (Virtual Path Identifier).

hard problem, and except for very simple flow specs, is not yet practical. It may be useful to use the flow spec to influence the path chosen without looking for an optimal path per se. Recent work on call setup such as RSVP, however, does not yet attempt to choose routes based on the flow spec.

When being used to influence routing, the flow spec is classified similarly to the ToS Field—that is, it is a modifier. Because the problem is so difficult, and because this thesis primarily deals with connectionless internetworking, we do not further examine the use of the flow spec in its role as a modifier.

Note that the data packets of a flow do not contain the flow spec. Instead, they contain one of 1) a flow ID only, 2) a flow ID and other routing information such as addresses, or 3) only routing information. In the first case, each router forwards based solely on the flow ID. In the second case, routers will normally forward on the flow ID, but may also use the routing information, for instance because the state for the flow ID has been lost.

Although the flow ID in the data packet therefore influences the route, it in itself carries no routing and addressing semantics. Rather, it is nothing more than a mnemonic for the flow spec and routing information that was in the call setup (or former packets). As such, the flow ID does not provide new routing information and is therefore outside the scope of the taxonomy presented here.

2.3 Summary of Taxonomies

Each of the three taxonomies presented here are valid and useful within their context. Shoch chose to focus on the terms that are most in the networking vocabulary (now as well as then)—name, address, and route. Shoch remained true to the vernacular meaning of these terms in networking, that names identify what is sought, addresses indicate where the object is, and routes describe the path to the object. Shoch shed much light on the meaning and use of these terms.

Saltzer felt that the vocabulary of Shoch lacked precision, and so delegated the terms name and address to their classical meaning in computer science (an address of an object is a name of the object it is bound to). Saltzer introduced the entities users/services, hosts, network attachment points, and paths, and described how each is bound to the next.

This chapter further examines the role of addressing, or more generally, everything that goes in a packet header that affects routing. Saltzer's taxonomy is inadequate for this purpose because it takes the focus away from the address. Shoch's taxonomy lacks precision in-so-far-as addresses (in Shoch's sense of the word) are concerned.

Thus, a refinement of addressing taxonomy is introduced. In particular, there are three functions in addressing—identifying (the destination), locating (the destination), and modifying (the path to

Table 2.1: Summary of Header Field Classifications

Header Field	*Classification*	*Section*
Ethernet Address	Identifier	2.2.1
IP Unicast Address	Locator	2.2.1
Type-of-Service Field	Modifier	2.2.3
Source Route	Locator	2.2.2
Mobility Stable Address	Identifier	2.2.4
Mobility Non-stable Address	Locator	2.2.4
IP Multicast Address	Identifier	2.2.5
Source Address†	Modifier	2.2.5
Scope/Source Address/Multicast	Identifier	2.2.5
Flow Spec	Modifier	2.2.6
Flow ID	Not Applicable	2.2.6

†with Source-rooted Multicast

the destination). The identifier unambiguously identifies the destination regardless of its location in the network. The locator also unambiguously identifies the destination, but is dependent on the network location of the destination. Locators can be source-sensitive or source-insensitive. Unless otherwise stated, source-sensitive locators are called vectors, and source-insensitive locators are called just locators. The modifier influences the path taken to the destination, but has no bearing on the location or identity of the destination.

The components of well-known packet headers, particularly IP, are classified according to this taxonomy, and are found to fit neatly into the taxonomy. Table 2.1 summarizes the classification.

Chapter 3

Locators

This section discusses various forms of locators that reduce the cost of routing. It has already been mentioned that locators are sensitive to host movement. When a host moves, it must obtain a new locator. This section shows how locators are also sensitive to topology. By topology, we mean the network graph formed by the connection of nodes with links.

There are two major forms of locators, hierarchical and non-hierarchical. Hierarchical locators are by far the more common, and are discussed here first.

3.1 Hierarchical Locators

All hierarchical locators employ some form of clustering. That is, groups of nodes are formed into a cluster, which is represented by a single value in a component of the hierarchical locator. For instance, with IP, the three address components are host, subnet, and network. The subnet component identifies the cluster of hosts consisting of those hosts attached to the subnet. The network component identifies the cluster of subnets that belong to a given network.

In a hierarchical locator, the lowest-level (0th level) component is the host (or router). The next higher level (1st level) component is a cluster of hosts. The 2nd level component is a cluster of 1st level clusters, and so on [62]. This thesis refers to both hosts and clusters as *hierarchy elements*.

The topological constraint placed on clusters is that there must be a path between any two nodes in a cluster that only traverses nodes that belong to the cluster. In other words, it must be possible for nodes outside a cluster to view the cluster as a single component. Thus, large numbers of nodes can be viewed as a single node, thus shrinking forwarding table size, and decreasing the cost of routing. If the cluster is not internally connected, it is not possible to view it as a single component.

There are several aspects of hierarchical addressing that are of interest. One aspect is that of

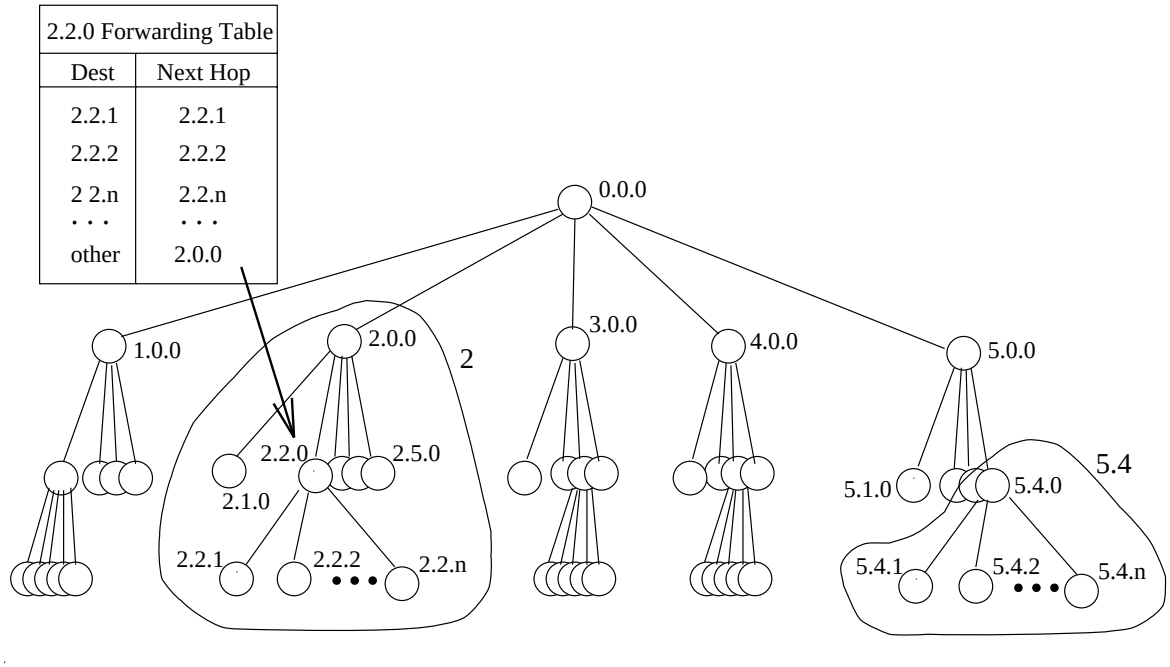


Figure 3.1: Simple Tree Clustering

determining which elements should be clustered at each level, and whether elements can belong to multiple different clusters. Another aspect is that of determining what routing information hosts and routers should contain. Both of these aspects are influenced by the topology, the desired quality of the paths found, the desired robustness to network failure, and the desired algorithmic simplicity. What follows is a number of different approaches to hierarchical clustering.

3.1.1 Tree Topology

The simplest form of hierarchical clustering is where the network topology is a tree, and the address structure follows the tree structure, as shown in Figure 3.1. In the tree topology, a group of nodes is clustered by virtue of having a link to the same higher level node.

The addressing reflects this clustering as shown in Figure 3.1. The addresses in this example have three components, with each successive component identifying the next cluster down. The higher the cluster, the fewer components in the address required to identify it. By convention, an address with three components refers to a node, and an address with fewer than three components refers to a cluster. An address with 0s as the trailing components refers to a node that is not a leaf in the hierarchy. Thus, the top-most node has address 0.0.0, a node below it has address 2.0.0, and so on. The address 2 refers to 2.0.0 and all nodes below it.

With a tree topology, the only explicit information required in a forwarding table for a given node concerns what is below it. When a node receives a packet, it looks up the address of the packet

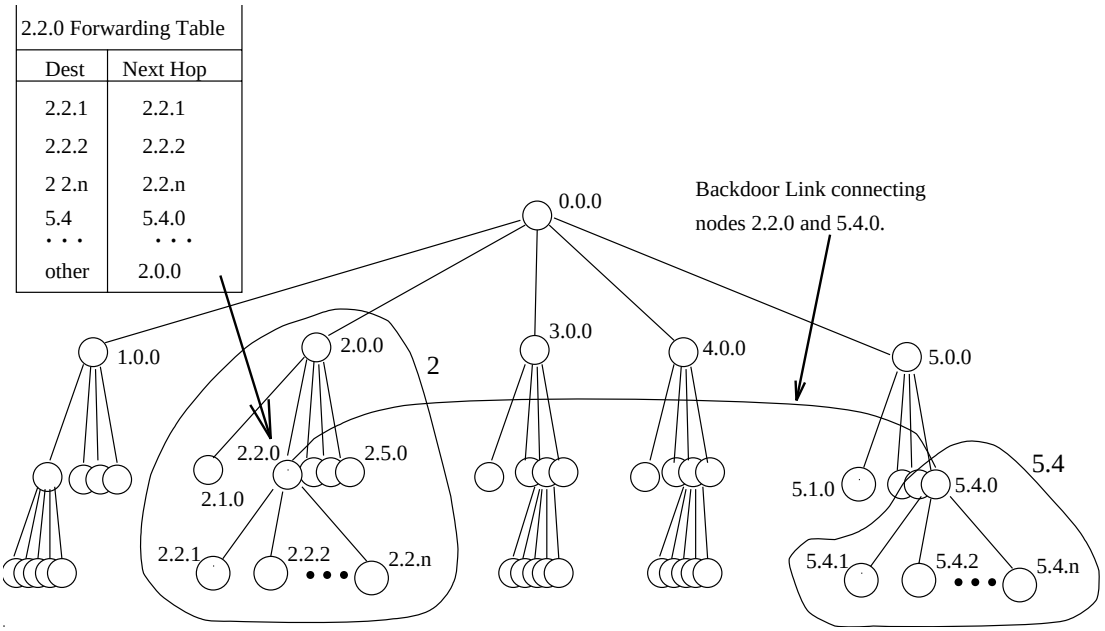


Figure 3.2: Tree with Backdoor Link

in the forwarding table. If the result is that the destination is below it, then the packet is sent down to the appropriate neighbor node. Otherwise, the packet is sent up.

For example, consider the forwarding table of the node labeled 2.2.0, shown in Figure 3.1. It has entries for the nodes below it; 2.2.1, 2.2.2, etc. Any address not of the form 2.2.x, where x implies any value, is simply forwarded up, to node 2.0.0 (shown as *other* in the forwarding table). This forwarding up of packets not destined for things below is called *default routing*. It is an effective means of reducing forwarding table size and routing complexity for nodes not at the top of the hierarchy, and is in common use in the IP internet, even though the topology of the IP internet is not strictly a tree topology.

The tree topology is simple and effective, but has two major problems, both stemming from the fact that all traffic between a given two nodes goes through their parent nodes. First, congestion can occur at the parent nodes. Second, the parent nodes are single points of failure.

Tree Topology with Enhancements

To alleviate these problems, three enhancements to the tree topology are commonly used in practice. First, there can exist links between nodes that do not share a parent-child relationship. These links are called backdoor links, or just backdoors. This is shown in Figure 3.2. In practice, backdoor links often are limited to carry only that traffic between nodes in the clusters that the backdoor connects. Indeed, in the IP Internet, backdoors are usually explicitly prevented from carrying any other traffic.

Thus, the purpose of the backdoor link is primarily to shunt traffic away from the upper portions of the hierarchy (or, from the users perspective, avoid the performance degradation or cost associated with going through the upper portions of the hierarchy). With respect to improving robustness, the backdoor is limited in that it typically only provides an alternate path for traffic between the two clusters, but not for traffic between nodes in other clusters.

The hierarchical locator is not affected by backdoor links. The hierarchical locator still follows the up/down links that are the basis of the tree topology. The forwarding tables of routers, however, are affected by the backdoor links.

Figure 3.2 shows the forwarding table of node 2.2.0 modified to take into account the backdoor link to 5.4.0. In addition to checking packets for addresses 2.2.1, 2.2.2, etc., it must check for packets with address prefix 5.4. Any packets with an address prefix of 5.4 are routed to node 5.4.0, which then forwards them down as appropriate. The ability of a router to choose a finer-grained forwarding table entry (for example, 5.4) over a coarser-grained forwarding entry (for example, *other*) is called best-match routing [73]. Default routing can be viewed as a form of best-match routing where the coarsest-granularity entry is "all addresses".

The second enhancement to the tree topology is where a single "node" in the otherwise tree topology is actually a collection of nodes internally connected, usually but not necessarily by a mesh topology. A mesh topology is a topology with no regular structure. Such a collection of internally connected nodes is called a backbone.

Figure 3.3 compares an element of the tree topology with an element of the *backbone-tree* topology. It can be seen that logically the two are equivalent, but that the backbone reduces or eliminates single points of failure (particularly if the lower-level element is connected in multiple places, such as 2.2.n) and spreads traffic over multiple nodes. Thus, hierarchical locators still follow the tree structure as with the pure tree topology. The contents of forwarding tables also does not change significantly by the introduction of backbones. The only difference is that, in the case of backbones, the forwarding table entry may direct a packet to another node in the backbone before it is directed down the hierarchy.

The top of the backbone-tree topology can be, and in practice typically is, a set of backbones, themselves interconnected by a mesh topology. In terms of addressing, this is equivalent to removing the top element of the tree topology, and interconnecting the resulting top elements with a mesh topology. Thus, forwarding table entries in nodes at the top of the hierarchy will have explicit entries for other top-level elements rather than a default entry pointing up.

In the third enhancement to the tree topology, elements are allowed to have multiple parents. This enhancement is similar to the backdoor in that it is a link that connects different elements of the hierarchy, but different in that the intent is for the link to be a "full participant" in the topology—it can be used to forward traffic between what is below it and everything else in the topology.

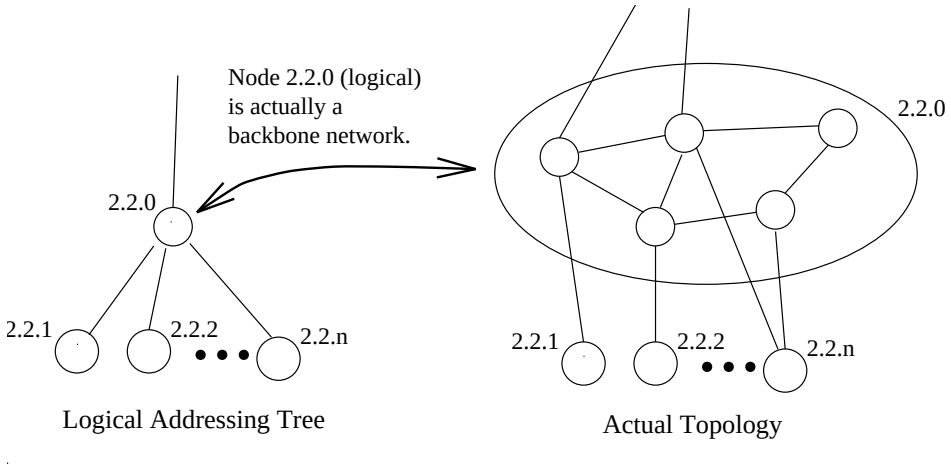


Figure 3.3: Logical Nodes as Backbones

For this level of participation to scale well, the element with multiple parents should obtain addresses from each of its parents, resulting in multiple addresses for multiply connected hierarchy elements.

This adds overhead to the naming system, since each multi-parent element must have multiple entries in the naming system. The increase, however, is a small constant. This enhancement also complicates almost every aspect of addressing and routing. Choosing an address is more complex because systems must be able to choose from multiple addresses. This is not a trivial choice, because the address chosen influences the path taken (it causes the packet to go through the part of the hierarchy indicated by the address [107]). Default routing becomes more complex, because the number of up choices increases.

3.1.2 Mesh Topology

It is possible to form a cluster hierarchy even when the topology displays no hierarchical characteristics. All that is necessary is to logically group hierarchical elements such that each group is internally connected [62].

Figure 3.4A shows a mesh topology. Figure 3.4B shows two levels of clusters superimposed on the mesh topology. It also shows the node addresses that result from this clustering. As stated before, the only hard requirement for this clustering is that each cluster be internally connected. There may, however, be any number of less hard requirements that determine how clustering is done. For instance, a certain amount of connectivity within each cluster may be required (for instance, at least two paths between any two nodes in a cluster). Or, clusters may be formed around nodes that exchange a lot of traffic.

Hierarchical clustering over a mesh topology can result in paths that are longer than shortest

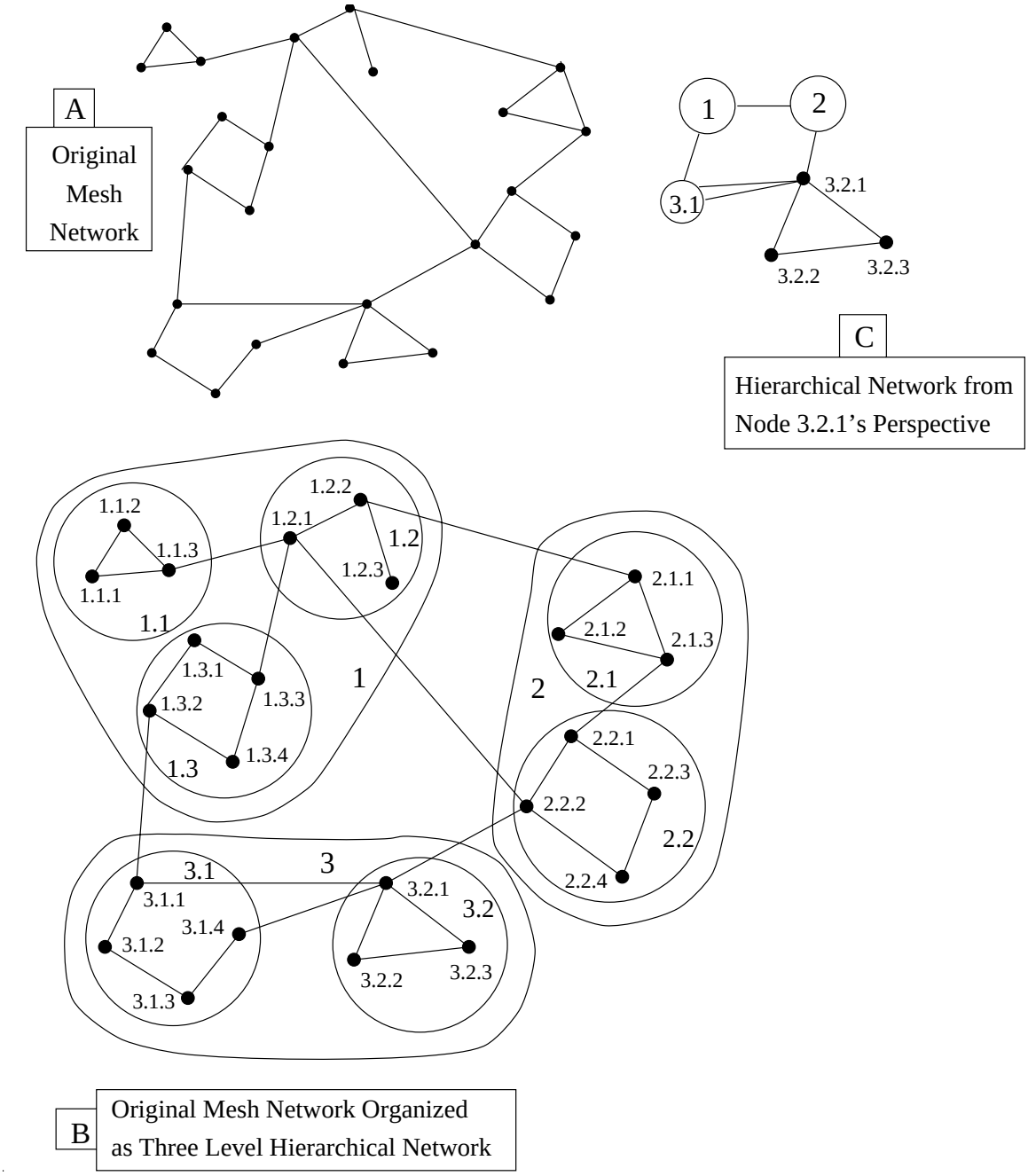


Figure 3.4: Hierarchically Organized Mesh Network

path. For instance, assume that the forwarding tables in every router maintain entries describing how to route to each node in their own level 1 cluster, each level 1 cluster in their level 2 cluster, and so on up to the top level, for which forwarding information about all top level clusters is maintained. This style of forwarding information is proposed in [62].

With this style of forwarding information, the "view" of the network for node 3.2.1 is as shown in Figure 3.4C. With this view, node 3.2.1 only requires entries for 5 clusters/nodes, versus entries

for 23 nodes with flat routing. One cost of this savings, in the case of hierarchical clustering over a mesh topology, is longer paths than those found with flat routing. For instance, consider the path from 3.2.1 to 1.2.1. Because 3.2.1 knows nothing about the internals of clusters 1 or 2, 3.2.1 would likely assume that the shortest path to anything in cluster 1 is via the direct link between its top level cluster and cluster 1. The resulting path is 5 hops long: 3.2.1—3.1.1—1.3.2—1.3.1—1.3.3—1.2.1. The shortest path (in terms of hops, anyway), on the other hand, is 2 hops: 3.2.1—2.2.2—1.2.1.

Note that such non-optimal paths do not exist in the case of hierarchical clustering over a tree topology, because there is only one possible path between all nodes pairs. Such non-optimal paths can exist in the enhanced tree topology, but generally to a lesser extent than with a mesh topology. Backdoor links generally do not result in non-optimal paths because backdoor links usually service a limited community of nodes, and because those nodes can easily hold explicit forwarding information for the backdoor link. The routing information required to choose among multiple parents or multiple addresses scales with the number of parents, which is small [107], so multiple parents need not result in non-optimal paths. The existence of mesh networks within a hierarchy element can result in non-optimal paths, because there may be multiple ways to enter a hierarchy element, and the forwarding information required to indicate the best one could be a significant increase in the overall forwarding information required.

Another interesting characteristic of hierarchical clustering is that a cluster partition can result in a node being unreachable even though there is a physical path to that node. For instance, consider the case where the link between 1.3.3 and 1.2.1 crashes, thus partitioning cluster 1.3 from the other clusters in cluster 1. Given the forwarding information described above, if 3.2.1 tries to send a packet to 1.2.1, it will reach 1.3.3 and then be dropped, because 1.3.3 has no way to forward it back through cluster 3 and cluster 2 into the other partition of its own cluster. Methods exist for repairing such logical partitions [80, 56, 98, 113]. These methods involve 1) renumbering nodes, 2) replicating the packet and sending each replica into each partition, or more commonly, 3) temporarily obtaining extra forwarding information about the internals of other clusters.

Variations in Forwarding Information

Within the context of the hierarchical clustering/addressing described above, there are any number of variations as to the specific forwarding information maintained at nodes. The forwarding information described above, where each node keeps some information about all hierarchy levels, but only that information that pertains to the internals of the node's own clusters, is here called *classical forwarding information*[1].

One variation on classical forwarding information is where a node keeps some internal information about neighboring clusters, that is, clusters with which it shares a link. With this information,

[1] The term classical was suggested by Bala Rajagopolan in personal conversations.

a node could in some cases find optimal paths to nearby nodes even though thoses nodes are in different clusters. Thus, nodes find better paths at the cost of maintaining additional information [3]. Maintaining this type of additional information is often refered to as *hole punching* in the Internet community.

Another variation is to establish a kind of default routing, where nodes maintain classical forwarding information for destinations at and below their level, but only maintain information on how to route to the closest higher level node for each higher level. This style of default routing technique requires less forwarding information than full classical routing, and has simpler operation for lower-level nodes, but may result in longer paths.

Landmark Forwarding

Another style of hierarchical clustering on a mesh network is the Landmark Hierarchy [106]. The Landmark Hierarchy was designed to facilitate complete auto-configuration of the hierarchy. That is, the clustering and addressing happens automatically.

The Landmark Hierarchy is formed as follows. Individual nodes are randomly assigned a hierarchical level with decreasing probability for higher levels. Routing table entries for a given node X are maintained in all nodes a certain number of hops (called the radius) from node X. The higher the level of a node, the larger its radius. Every node becomes the child of the closest next higher level node. This defines the node's address (the parent assigns each child a number from its address space). The radius of any given node must be large enough such that its parent knows how to route to it. Nodes at the top of the hierarchy have an infinite radius (that is, all nodes know how to route to the top-level nodes).

This hierarchy is easier to autoconfigure because the focal points of both routing and cluster definition are individual nodes (the "landmarks"), not entire clusters. It is easier to algorithmically manage single nodes than groups of nodes. At the same time, because of these individual focal points, the Landmark hierarchy is more sensitive to single node failures. The failure of a landmark results in a partition of the cluster that had formed around it.

3.2 Non-Hierarchical-Topology Locators

There are a limited number of non-hierarchical-topology locators. The most common of these are those used with a regular, though non-hierarchical, topology. Examples of such topologies in the context of communications networks are Gridnet [72], Cartesian Routing [43, 44], and the Manhattan Street Network [69].[2] Gridnet's topology is a grid of hexagons (each node has 3 neighbors), and Cartesian Routing and Manhattan Street Network are grids of squares (each node

[2]The large majority of work done on regular networks is in parallel processing, for instance [6, 63, 89, 59].

has 4 neighbors). In all cases, a node is given a 2-element address. One element gives the node's location on a horizontal (or east-west) coordinate, and the other gives the node's locations on a vertical (or north-south) coordinate. To route a packet, a node simply forwards the packet to a neighbor that gives the packet forward progress on one of the coordinates.

Just as a cluster in a hierarchical topology can logically partition, a grid topology can also logically partition. This happens when a node loses connectivity to one or more neighbors and therefore cannot make forward progress on either coordinate. There are a number of approaches to repairing partitions in a grid topology, usually involving spreading information about the broken topology to nearby nodes so that they can route around the partition, but also potentially tagging packets so that they can temporarily make backwards progress in order to go around the partition [44, 69].

Grid topologies have many positive properties. One of them is that the routing scales to an unlimited number of nodes and links. Another is that traffic can be dispersed among multiple paths, for the purpose of either robustness or avoiding congestion.

The main negative property of a grid topology is that the topology must be a grid. Especially as a network covers larger and larger geographic areas, a grid becomes a less and less economical topology. A more economical topology is to place links between locales that are 1) near each other, and 2) exchange enough traffic to justify the link. In addition, formation of a grid topology requires complete cooperation between various network administrations. Such cooperation does not exist, and should not be required to exist, in the Internet.

To create a grid topology where one does not naturally exist requires either installing gratuitous physical links, or creating logical links between nodes not physically connected by adding extra routing information. The latter approach is similar to the techniques used to repair a grid partition, but where the "partition" is a permanent and intentional condition. Either approach has associated costs that may negate the benefits of the grid.

Other common regular topologies are rings and chains. The ring is especially common for local area topologies [51], but has also been used for the wide area [100, 66], particularly as the basis for a bus architecture [74, 33]. Like a grid topology, rings and chains have the advantage of simplicity, and the disadvantage of a forced physical topology that may not be a natural fit for the network user population.

3.2.1 Non-Regular Topologies

An interesting and not widely known (at least in the internetworking community) scheme for assigning non-hierarchical locators on regular or non-regular topologies is called *interval* routing [111, 41]. In such a scheme, every node is given a single unique flat address (called labels in [111]). The labels are assigned such that forwarding tables of the following type can be used to route a packet to any node.

Each node has a forwarding table that consists of a single label for each link. The labels are cyclically ordered. A packet with label (address) x is routed over the link whose label is less than or equal to x (in the cyclical ordering), but numerically closer to x than any other link's label. Thus, the forwarding table size for any given node is equal to the node degree of that node, independent of the size of the network.

In [111], various labeling schemes are presented that work for regular and non-regular topologies, and that utilize every link in the network (unlike an earlier scheme [97], that worked only by creating a logical spanning tree over an otherwise mesh network). For some regular topologies (such as grids and rings), the labeling schemes find the shortest paths (in the case where every link is assumed to have equal cost). For non-regular topologies, the labeling scheme does not necessarily find the shortest path.

Like grid topologies, the labeling scheme is subject to logical partitions when two neighbors become disconnected. In addition, in the general case, nodes must be relabeled when the topology changes. These problems are explored, and limited solutions are offered in [111]. Although interval routing might not (or might) be of practical use, it is interesting work none-the-less.

3.3 Current IP Internet

As already stated, IP addresses are hierarchical locators. The current IP internet is essentially a tree topology, but with all three enhancements. The hierarchical elements of the current IP internet are: provider network, subscriber network, subnet, and host. The CIDR IP address assignment scheme matches this hierarchy [42]. That is, blocks of IP addresses are assigned to providers, which assign sub-blocks to subscribers, which assign sub-blocks to subnets, and then to hosts.

The top level of the IP internet consists of multiple provider networks interconnected in a mesh topology. A provider network is a backbone network established for the purpose of providing packet carriage between subscriber networks. Subscriber networks are typically private networks such as a campus or corporate network. A subnet is a network operating "below" (in the sense of protocol encapsulation) IP, such as an Ethernet or X.25 network [16]. Nodes attached to the same subnet share a subnet address prefix.

The second enhancement mentioned above, where a single element of the hierarchy is a backbone network, is pervasive throughout the IP internet. Provider networks are almost always composed of routers connected by links. Subscriber networks are also usually composed of routers connected by links, except that the links are typically LANs that are also subnets to which hosts are attached. Thus, subscriber networks can be described as a collection of subnets interconnected in a mesh fashion by routers.

Backdoor links, the first enhancement mentioned above, are not so pervasive as backbone topolo-

gies, but do exist. Since backdoor links tend to be hidden (they do not appear in forwarding tables of routers outside of the elements being connected), it is hard to know how many exist in practice.

The third enhancement, multi-parent elements, are also not so pervasive, but do exist. They typically exist either between subscriber and provider, or host and subnet. That is, a subscriber might be connected to multiple providers, or a host might be connected to multiple subnets (within a given subscriber), but it would be strange for a subnet to be connected to multiple subscribers. A subnet normally belongs (in an organizational sense) to a single subscriber network, and so it would not make sense for it to fall hierarchically under two subscriber networks.

Subscribers connected to multiple providers is a particularly interesting example of a multi-parent element. This is because the subscriber-provider relationship is a significant one both with respect to billing and service provided. As the internet becomes commercial, and as multiple services, such as voice and video, become available, this relationship, and the associated use of multiple providers, will likely become still more important.

3.4 Geographic Versus Provider-rooted Addresses

The relationship between a subscriber and multiple providers (either simultaneously or sequentially) raises some interesting new problems in the IP internet. If the top-level hierarchical address component is assigned to providers, then a subscriber network will get new addresses when it changes providers, and will have multiple addresses if it subscribes simultaneously to multiple providers.

The notion of hosts having a single, static address is deeply ingrained in the IP internet. There are no automatic procedures for modifying the addresses of a group of IP hosts, even when all of the IP hosts have the same address prefix and the modification is only to the prefix. In addition, IP hosts generally have little notion of other IP hosts having multiple addresses. For instance, IP hosts generally have no software for choosing among multiple addresses presented to them by directory service, and multiple IP addresses cannot be used to identify a transport connection, even though the multiple IP addresses may identify the same host.

Because of this deeply engrained notion of IP addresses, the introduction of provider-rooted addresses to the IP internet may require significant changes to the operation of the IP internet [107]. While [107] argues that these changes are positive ones, bringing new features and new flexibility to the internet, there is no question that these changes require new functionality and result in added complexity.

An alternative address assignment scheme is geographical addressing, such as exists in the global telephone network [14]. Because geographical addresses remove the dependency of address on provider, a subscriber can change providers or have multiple providers without changing addresses.

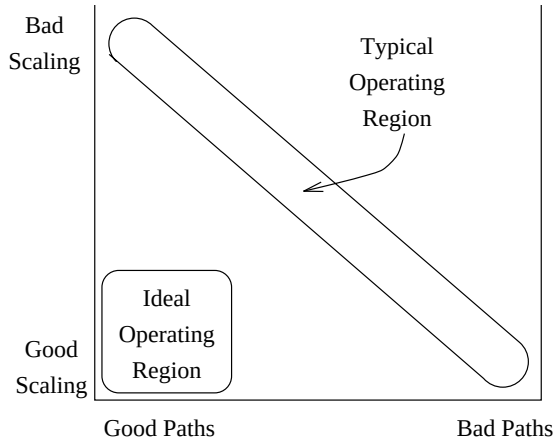


Figure 3.5: Relationship Between Scaling and Path Quality

The use of geographical addresses, however, puts an additional burden on providers, in terms of how much routing information they must maintain and on how they must interconnect.[3]

Because of the timeliness and importance of the issue of geographic versus provider-rooted addresses, a comparison of the two methods is given here. (Note that many of the ideas presented here were discussed on the big-internet mailing list of the Internet Engineering Task Force (IETF) [7].)

3.4.1 Some Background

The assignment of addresses in the internet follows a tree of address assignment authorities. At the root of the tree is the top-level (or level H) address assignment authority. This address assignment authority assigns blocks of numbers to the next level down (level H-1) address assignment authority, which assigns blocks of numbers from the block it owns to level H-2 address assignment authorities and so on. For the sake of discussion, we refer to assigning a block of numbers as simply assigning a number.

The issue is how to assign these numbers so that 1) routing scales well, 2) good paths are found, 3) constraints on the physical topology are minimized, 4) reconfiguration of systems is minimized, and 5) the address assignment process is simple, fair, and politically viable. Consider the graph of Figure 3.5. In general the "physics" of networking forces operating points on this graph to be along a region extending from the upper left to the lower right. That is, one typically can get good scaling but bad paths, or good paths but bad scaling, or something in between. Depending on the type of address assignment scheme used, however, it is possible to move somewhat towards the lower left (good solutions). This may, however, increase topology constraints or reconfiguration requirements.

[3]Steve Deering of Xerox Parc is credited with promoting the idea of using geographical addresses in the Internet. The descriptions of geographical addresses given here derive largely from his work.

Central to the evaluation of any address assignment scheme are answers to the questions 1) what constitutes good scaling, 2) what constitutes a good path, 3) what constitutes unacceptable or costly topology constraints, 4) what constitutes unacceptable or costly reconfiguration, and 5) what constitutes a simple, fair, and politically viable address assignment process. Except for possibly the first question, it is difficult to answer these questions in general terms, partly because the cost of each aspect is borne by different parties, and partly because the cost of each aspect changes over time.

This section generally limits itself to describing the characteristics of the two address assignment schemes, and leaves it to others to determine the extent to which those characteristics are beneficial or detrimental.

3.4.2 Description of Provider-rooted Addressing

The basic approach to provider-rooted addresses is as follows. The top-level address assignment authority assigns numbers directly to providers. This includes both internet protocol service providers and lower-layer (for instance, ATM) protocol service providers. Depending on its size, the provider can either assign the next level internally, or assign the next level directly to its subscribers. The internal assignment would be for clustering groups of subscribers under a single prefix for the sake of internal scaling.

Thus, the address prefixes would be:

provider.subscriber

or

provider.subProvider.subscriber

To understand this notation, consider Figure 3.6. Shown are three providers with subscribers attached to them. The providers have been given top-level numbers 29, 48, and 14. Provider 29 has given two subscribers next-level numbers 12 and 17. Thus, the upper-left subscriber with assignment 12 has a prefix of 29.12. This means that the field of the address that indicates provider is 29, and the field that indicates subscriber is 12. All host addresses in this subscriber network start with the prefix 29.12.

It is possible that the providers themselves are somewhat hierarchically organized. For instance, there may be long-distance and local-access providers. The subscriber is directly connected to the local-access provider, but may also have a service relationship with one or more long-distance providers to which the local-access provider is connected. In this case, the address prefix could be formed as shown above, or could include both the long-distance and local-access providers:

LDprovider.LAprovider.subscriber

In either case, subscribers are given an address prefix from each top-level provider through which

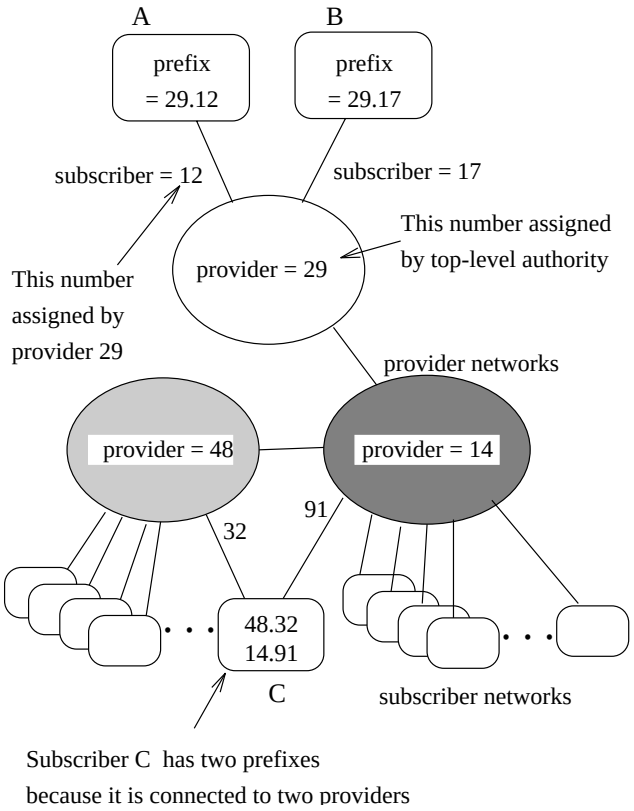


Figure 3.6: Example of Provider-Rooted Addresses

they derive service. Each host in the subscriber network, then, has one address for each provider through which the subscriber network can be reached—for instance, subscriber C, connected to both providers 48 and 14 in Figure 3.6. When the subscriber subscribes to a new provider, or unsubscribes from an existing provider, it must change the address prefix for all of its hosts and routers.[4]

Additional levels are assigned under the subscriber number for use within the subscriber network. These levels are not relevant to this discussion.

3.4.3 Description of Geographical Addressing

Our working definition of geographical addressing is where the top N hierarchical levels of the address are assigned to geographical regions. Each level of geographic area is completely within the next higher level of geographic area. Three examples of geographical address prefixes are:

country.metro.site

continent.metro.site

[4]Strictly speaking, the subscriber may not have to change its prefix. However, this results in worse scaling, as discussed later.

continent.country.metro.site

Note that with geographical addresses, the lowest level of assignment is to "sites" rather than to "subscribers". Both site and subscriber, however, represent private networks that are assigned address prefixes.

Because of the requirement that the elements of a hierarchy cluster must be internally connected, it is necessary in a geographical addressing scheme that all hierarchical elements in a geographic area be internally connected. (Note that this does not necessarily require N^2 connectivity—that is, where all N hierarchy elements are directly connected to each other. Rather, it requires that there be some path from any element in an area to any other element in the area that only traverses elements in the area.) For instance, if the geographic clustering is *country.metro.site*, then all metro networks in a country must be able to reach all other metro networks in the country without going through another country. Likewise for all the sites in any metro, etc.

Consider Figure 3.7. It portrays the same providers and subscribers A, B, and C as Figure 3.6, but shown geographically rather than logically according to provider. The providers overlap geographical area, so the routers of the providers are shown in Figure 3.7. The address convention of Figure 3.7 is *country.metro.site*, where country = 93 and metro = 42. Note that site C (labeled subscriber C in Figure 3.6) has only one address even though it is connected to two providers. Note also that all of the routers in metro 42 are internally connected by virtue of two (heavily drawn) links between routers of different providers.

Taken to the extreme, the assignment of geographic addresses could be carried all the way to individual hosts. That is, geographical areas could be recursively subdivided until every possible host location in the world (galaxy?) defines a unique address. Clearly this is unworkable. At the local (campus or single building) level, one must assign addresses according to network topology, not some pre-determined geographical partitioning. Thus, at some point in the hierarchy, the addressing must change from geographical to network-physical.

A sensible place to make this change is at the boundary between the private network (or site) and the provider. Within a site, address administration should be completely autonomous and not constrainted by geography (or anything else not within the control of the site). Thus it would not be appropriate to make the change at some level below the provider/subscriber boundary. And, since provider coverage does not necessarily conform to geographic boundaries (some providers are global in scope, and provider coverage areas overlap considerably), it does not make sense to make the change from geographical to network-physical at the boundaries between providers.

Thus, geographical addresses have a geographical part, a site part, and an intra-site part:

geographicalPart.sitePart.intra-sitePart

where each part can have internal layers.

The geographical part for a given site is determined according to the geographic location of the

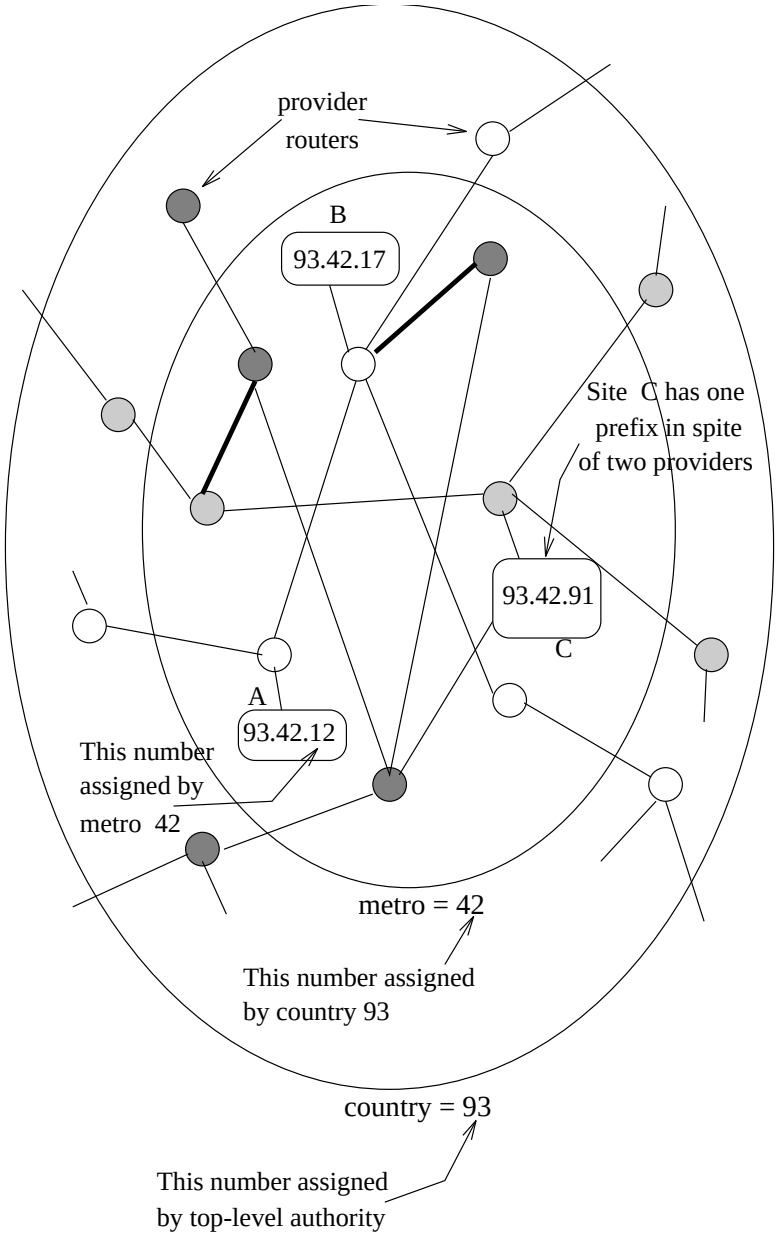


Figure 3.7: Example of Geographical Addresses

site's connection to a provider. This is where the site "appears" in the global topology. Thus, even though a site may cover multiple geographic areas, if it is attached to a provider in only one geographic area, the whole site will have a geographic prefix indicating that geographic area. More typically, a site that covers multiple geographic areas would be connected to providers in multiple geographic areas.

In any event, the main point is that the specific provider that a site attaches to does not affect the site's address. Thus, a site could change from one provider to another in a given geographic area, or attach to multiple providers in a given geographic area, without changing addresses or

having multiple addresses.

3.4.4 Topology Constraints

Provider-rooted addresses place no "unnatural" constraints on the topology. Of course, with provider-rooted addresses, each provider must be internally connected, but a provider would naturally be internally connected, so this represents no real constraint. Provider-rooted addresses place no constraints on how providers interconnect with each other.

Geographical addresses do place an unnatural constraint on topology. That is, they require that the providers that cover a geographical area (that area denoted by the geographic prefix) be connected in that area. While it is natural for providers to be connected to each other somewhere, it is generally (though not necessarily) unnatural to force them to be connected in every geographic area that they cover.

In the current USA Internet topology, provider networks tend to interconnect in a small number places, for instance at FIXs or CIXs (Federal or Commercial Inter-exchange). Thus, requiring connectivity in every metro area, for instance, would require much more inter-connectivity that there currently is. On the other hand, the long-distance phone carriers in the USA have connectivity in every geographical area (called LATAs).

3.4.5 Scaling in Routing

In this section, the information needed in routers' forwarding tables for both geographical and provider-rooted addressing is described and compared. The information may vary, depending on the desired quality and flexibility of paths found. This section also describes methods for improving the scaling characteristics of both schemes.

Scaling of Provider-rooted Addressing

As stated above, provider addresses are of the form:

provider.subscriber

or

provider.subProvider.subscriber

For addresses of the form *provider.subscriber*, routers in provider networks must, at a minimum, maintain routes (forwarding table entries) for 1) other providers, and 2) subscribers within their own provider network. For addresses of the form *provider.subProvider.subscriber*, routers in provider networks must, at a minimum, maintain routes for 1) other providers, 2) subProvider clusters within their provider, and 3) subscribers within their subProvider cluster.

The number of subProvider or subscriber routes that a router must maintain is within the control of the provider. As a provider obtains more subscribers, it can add internal levels of hierarchy (*subProvider*, *sub-subProvider*, etc.) to keep the number of internal routes manageable.

A provider, however, cannot control the number of other providers for which it must maintain routes. Thus, the size of the forwarding tables at the top of the hierarchy (provider) is open-ended. As a result, the forwarding table size may grow beyond acceptable levels. One solution to this problem is to add another level of hierarchy above the provider level.

providerCluster.provider.subscriber

With this address, multiple providers are clustered within a new top-level identifier, the *providerCluster*. One possible basis for provider clustering (that is, the choice of which providers go into which clusters) is geographical location. In this case, a provider that spanned multiple geographic areas would appear as multiple providers, one for each geographic area it appeared in. A perhaps better basis for provider clustering could be the kind of service provided. For instance, all ATM providers could form a cluster, all internet providers another cluster, and so on. Another basis could simply be the amount of interconnection various providers have with each other. Providers with a large number of interconnections would naturally be placed in the same cluster.

Another solution is possible in the case where a relatively small number of providers are long-distance providers, and the rest are local-access providers. This form of address (*LDprovider.LAprovider.subscriber*) is discussed above. In this case, only the long-distance providers are advertised globally.

Routers in provider networks may also wish to maintain certain information about the internals (*subscriber* or *subProvider*) of another provider. This would happen in the case where

1. Two providers are interconnected with each other in multiple places, and
2. the routing policy for one of the providers is to route the packet to the interconnection point closest to the destination (versus simply routing the packet to the nearest interconnection point).

The amount of routing information in this case is also open ended, as it depends on the number of providers with which there are multiple interconnection points (which itself depends on many factors, such as the user traffic matrix), and it depends on the number of subscribers in other provider networks and on how internal clustering is done in other provider networks.

Whether or not it is advantageous for a provider to route a packet to the nearest interconnection point versus route a packet to the interconnection point nearest the destination depends on many factors, not the least of which is the business relationship established between the two providers on how they compensate each other for traffic carried. A discussion of the advantages and disadvantages of this routing policy is outside the scope of this thesis.

Scaling of Geographical Addressing

As stated above, geographical addresses are of the form:

geographicalN.geographicalN-1 ... site

A router in a provider network must, in the general case, maintain routes for

1. all *geographicalN* clusters, all *geographicalN-1* clusters within their own *geographicalN* cluster, and so on,
2. all sites within the lowest-level geographical cluster that the provider router services.

The number of geographical clusters that a router must maintain routes for is fixed. If the top-level geographical clustering (*geographicalN* in the example above) is continent, then the top-level number of routes is 7 (or so, depending on what constitutes a continent). If the top-level clustering is country, then it is three hundred and something, and if it is metro, then it might be around 10,000 or so. In any event, it is fixed and either does not change or changes slowly and minimally over time. Since the geographic clustering is administratively determined (by whichever address assignment authority has control), the number of routes at the top levels can be set to be something reasonable for current technology capabilities, and thus scales well.

The number of sites within a geographic area, however, is open-ended. Thus, the size of the forwarding tables at the site level of the hierarchy is open-ended. As a result, the forwarding table size may grow beyond acceptable levels.

One solution is of course to add another level of geographic hierarchy above the site level, resulting in smaller geographical areas. This results in an prefix change for sites, which is counter to the reason for using geographical addresses.

Another solution is to arrange for a packet to visit all providers in a given geographic area, either by putting the packet on a broadcast medium that all providers listen on, or having the packet routed to each provider in turn. Each router that receives the packet knows if the destination is for one of its subscribers, and accepts the packet if it is. Note that the latter solution is generally preferable to the former one, because 1) the broadcast medium may become a traffic bottleneck, and 2) the broadcast medium solution will result in multiple packet deliveries for the case where a subscriber is attached to multiple providers in the geographic area. On the other hand, with the latter solution, there must be a way to prevent a (mis-addressed) packet that is not for any of the sites in a geographical area to continue looping among the providers.

Another solution to this problem is to place a provider layer of hierarchy between the geographical part and the site part:

geographicalPart.initialProvider.sitePart

The extra layer, initialProvider, indicates which provider the site initially connected to. Routers in a geographic area, then, must maintain routes for each provider in that area, plus routes for every site that is no longer attached to its initial provider. If most sites remain attached to their initial providers, then the number of routes is greatly reduced.

Routers in provider networks may also wish to maintain certain information about the internals (subscriber or subProvider) of another provider. This would happen in the case where

1. Two providers were interconnected with each other in multiple places, and
2. the routing policy for one of the providers was to route the nearest interconnection point (versus routing the packet to the interconnection point closest to the destination).

Note that this is the reverse of the routing policy described in the previous section. That is, with provider-rooted addressing, the natural path is to find the interconnection point closest to the source, and with geographical addressing the natural path is to find the interconnection point closest to the destination. In either case, finding the "unnatural" path requires extra forwarding information.

3.4.6 Address Reconfiguration

This section discusses the conditions under which address reconfiguration in private networks is required for the two schemes.

There are two cases where a private network assigned a geographical address prefix must change that prefix:

1. If the private network changes its provider access location to another geographical area, and
2. If the geographical areas themselves change.

The former would normally happen when a private network moves from one location to another. The latter has happened in the phone network in the USA in the form of area code splits. This happens when the available addresses in an area become depleted, and the area is split in half, assigning a new area code to one of the halves.

Area code splits (or more generally, geographical area splits) can be avoided if the routing supports multiple (overlapping) area identifiers for the same area. If this is allowed, then a new area identifier can be added to a geographical area if the addresses under the existing area identifier become depleted. Thus, no existing systems need to change address. Routers in an area must still maintain routes to all sites, however.

Another way to avoid area code splits is to simply make the address space in an area large enough to handle all growth. This of course requires a large address space.

There are several cases where a private network assigned a provider-rooted address prefix must change that prefix:

1. If the private network subscribes to a new provider,
2. If the provider has an internal layer of addressing and the subscriber moves to a new location with respect to the clustering defined by that layer, and
3. If the provider modifies its addressing scheme, for instance, by getting a new provider number or adding an internal layer of hierarchy.

Items 2 and 3 for provider-rooted addressing are similar to items 1 and 2 for geographical addressing respectively, and need no further discussion. As emphasized above, the main advantage to geographical addressing is that a subscriber can change providers without requiring a new address. Changing providers is likely to be a fairly frequent event, certainly a much more frequent event than either private networks changing location or geographic areas changing. Just how frequent depends of course on the subscriber, but several changes a year seems feasible.

Because of the frequency of provider changes, it is necessary to have an automatic means of changing all the host addresses in a private network at once. This task is greatly simplified by the fact that it is only necessary to change the provider prefix for each host, and that the change is the same across all hosts. The exception to this would be the case where the new provider prefix is so long that it takes up address space used for numbering in the private network.

There are two basic approaches to automatic private-network-wide prefix reconfiguration. One is to use a general purpose network management device that keeps track of the hosts in a private network and individually updates hosts using a general network management protocol such as SNMP [12].

Another approach is to design a specialized protocol that updates hosts. An example of this would be a modified host/router discovery protocol such as ES-IS [53], where routers periodically broadcast advertisements, and hosts discover the routers by listening to the broadcasts. The broadcasts could contain the prefixes of the private network. In this case, the routers would have to be updated individually to reflect the new prefix. This, however, is not so bad since routers need to be individually configured with addressing information for routing purposes anyway.

Given that a general management facility in a private network is useful for many things, it seems to be a better approach to the prefix reconfiguration problem. Note that the directory service, such as DNS [71], would also have to be updated to reflect the new prefix(es).

It is not necessarily true that geographical addressing isolates a private network from any per-host administration resulting from provider changes. For instance, consider the case when a private network is connected to multiple providers (or, is connected to one local-access provider but derives long-distance service from multiple providers) and wishes to be able to choose between

those providers on a connection-by-connection or packet-by-packet basis. This is called provider selection.

Provider selection is a special case of the more general policy routing [11]. The term policy routing is commonly used to describe the function whereby the source of a packet selects the series of providers that the path traverses. In the case of provider selection, only the providers on either end of the path are selected. In [107] it is argued that the providers closest to the source and destination are the most important, primarily because it is those providers with which the source and destination have billing relationships.

For provider selection to work, the following things, at a minimum, are required [107]:

1. The source must know which providers it is connected to.
2. The source must know which providers the destination is connected to.
3. The source must have enough information about the providers, and possibly how they are interconnected, to make an intelligent policy decision.
4. The source must have a way to indicate in the packet which source-end provider should be chosen.
5. The source must have a way to indicate in the packet which destination-end provider should be chosen.

To do provider selection with geographic addressing, hosts must be configured with information about their connected providers, and directory service must be configured with provider information so that remote hosts can obtain the information. Moreover, this information must be updated when a subscriber attaches to new providers. In addition, new mechanisms must be created to cause packets to be routed through the desired providers.

Thus, in order to get provider selection with geographic addresses, the same sort of private-network configuration and packet formatting is required as with provider-rooted addresses. In other words, the network configuration benefits achieved by using geographic addresses are largely lost if provider selection is required.

On the other hand, private-network configuration with geographic addressing is never worse than with provider-rooted addresses. And, if a private network does not require provider selection, for instance because it connects to only one provider, then private-network configuration is easier with geographic addresses in that nothing has to be done if the private network changes providers.

3.4.7 Address and Topology Administration

With provider-rooted addresses, address administration is straight-forward. The top-level address-administration authority assigns provider IDs directly to providers. Providers in turn partition

the address space as best suits their needs.

Alternatively, the top-level address-administration authority can assign blocks of provider IDs to sub-authorities, which can subsequently assign provider IDs to providers in their jurisdiction. For instance, the top-level address-administration authority could assign blocks of provider IDs to per-country assignment authorities.

In order to assign geographical addresses, two administrative tasks are required that are not required with the assignment of provider-rooted addresses.

1. The geographical boundaries must be determined.
2. The connectivity between providers within geographical areas must be determined.

It is hard to know the difficulty of these two tasks in the context of the internet. In areas where the establishment of internet providers has been unregulated, it can be imagined that the two tasks are quite difficult. This is because the positioning of geographical boundaries may have an economic impact on providers.

For instance, consider a provider that covers a certain region. If boundaries are drawn such that the provider is completely within a geographic area, then that provider only needs to interconnect with other providers in one geographic area. If, on the other hand, boundaries are drawn such that the provider covers parts of several geographic areas, the provider must interconnect with other providers in each of the geographic areas.

Since it is likely that one of the arrangements (probably the former) will be more advantageous to the provider than the other, the provider will naturally lobby for one set of boundaries over another. This is likely to conflict with the wishes of another provider, resulting in a difficult negotiating process.

Another difficult aspect of address assignment is that of determining how much address space goes to each recipient (either a provider or a geographic area). This is particularly true in the case where the address space is strongly limited, such as is the case with IP.

This aspect of address assignment has both political and technical difficulties. Politically, one organization may object to getting less address space than another. Technically, if not enough address space is allocated, then it is necessary to either renumber or to represent a single entity by multiple prefixes (or both). If too much address space is allocated, then the address space is poorly utilized. This is discussed further in Section 10.4.

3.4.8 Discussion and Summary of Geographical versus Provider-rooted Addressing

Several aspects of geographical and provider-rooted address assignment have been considered. Each technique has different advantages and disadvantages.

Both geographical and provider-rooted address assignments have potential scaling problems. With provider-rooted addressing, the number of providers is open ended. With geographical addressing, the number of sites in a geographic area is open ended. Techniques for improving their respective scaling problems are presented, but the techniques are not attractive.

While it is impossible to predict future growth with certainty, it seems likely that scaling would be worse with geographic than with provider-rooted addressing. The number of large providers is constrained by competitive and economic factors. It is likely that a relatively small number of large providers will dominate. Smaller providers will likely either be merged into the larger providers, or fall under the larger providers in a local-access/long-distance relationship.

With respect to address reassignment, provider-rooted addresses put a larger burden on private networks, since addresses must be reassigned whenever a private network subscribes to a new provider. Since both schemes can result in subscriber prefix changes, however, automatic host prefix assignment is desirable in any event. In addition, multiple addresses must be maintained for private networks connecting to multiple providers. This burden, however, can be leveraged for provider selection.

Geographic addressing places more constraints on the topology of the network, since providers must interconnect within geographic areas. Finally, geographic addressing has more administrative/political difficulties, primarily because the geographic boundary locations affect the topology.

Because the two addressing schemes have a different set of advantages and disadvantages, it is impossible to say which is better. Some generalizations, however, can be made. For instance, in general, geographic addressing is better for private networks and worse for providers, whereas the reverse is true for provider-rooted addressing. Also, geographic addressing works better in a well-regulated or well-organized environment. Because the internet has historically been, and still is, at best loosely organized, geographic addressing does not seem to be a feasible option at this time. As the internet matures, however, it may obtain better organization, and geographical addressing may become more feasible.

3.5 Relationship Between Cost of Routing and Cost of Deriving Addresses

Mechanistically, addresses do two things. First, they identify the destination. Second, they may describe, at a greater or lesser degree of specificity, the path to the destination. The need for the former function (identification) is fairly obvious, and need not be further discussed here. The need for the latter function (location) may seem obvious when considered in certain ways, but in fact deserves further discussion.

There are two uses for the locator (ignoring for now its role as an identifier). First, it aids in scaling. That is, it reduces the memory, bandwidth, or processing required to route packets compared to what would be required if identifiers were used. Second, it allows the source of a packet (or, whatever writes the locator into the packet header) to control the path. A significant amount of attention has been paid to the locator in its scaling role, both in the research literature [67, 46, 62, 61, 103, 106], and in commonly used networks such as IP [87], and public voice [14] and data networks [15]. Recently, some attention has been paid to the locator in its path control role, primarily in the context of the so-called policy routing problem [11, 107, 36, 34, 35, 101].

The latter function of the locator (path control) is important, and is treated later in this thesis. Here we consider the use of the locator for improving scaling.

A useful way to consider this use of the locator is in terms of what path information routers keep. Routers cannot keep full information about paths to all destinations. In the context of scaling, then, the purpose of a locator is to make up for a lack of path information in routers. That is, the locator describes, at some level, paths in the network topology, so that routers do not have to maintain full path information.

In general, the less path information that exists in the routers, the more path information must exist in the address. Thus, putting more path information in the locator improves the scaling characteristics of routers. However, putting more path information in the locator also increases the cost of deriving locators.

In the following examples, we assume a model where a source has a name for a destination. From this name, an address must be derived, at a certain cost. This address can be an identifier or a locator, depending on what is needed by routers. The form of the address is related to the cost of the path information in routers. The cost of obtaining the name in the first place is outside of the scope of what we wish to illustrate here, and so is not considered.

The total internet-wide cost of the system is the combined cost of deriving the address from the name, and the cost of maintaining path information in routers. We examine the costs here in simple terms, because we are for now only interested in general characteristics. We consider only the cost of total (internet-wide) system memory, and we do not account for caching strategies.

First, consider a routing system that maintains paths to all network destinations. This is called flat routing. The total internet-wide memory required to store this information is roughly N^2. This is because every router must maintain a routing table entry for every named host. (One could model this as a routing system that consisted of a single large database, with entries containing the next hop in the path from each system to every other system. The size of this database would be N^2.)

The address required by routers is a simple identifier. Assuming that names and addresses are separate entities, the cost of deriving an address from a name, in terms of total (internet-wide) system memory, is roughly N, where N is the number of named entities. This is because every name maps into one identifier, so in theory only one entry in the naming system is required for every named host. (One could model this as a naming system that consisted of a single large database, with entries containing the identifier associated with every name. The size of this database would be N.) Thus, the total system memory required is roughly $N + N^2$, which is roughly N^2 for large N.

Now consider a classical hierarchical routing structure, where network destinations are grouped in clusters, clusters are grouped into higher level-clusters, and so on in hierarchical fashion. Any cluster is a member of only one higher-level cluster. The cost of the routing system is roughly $NHN^{1/H}$, where H is the number of hierarchy levels [62]. This is because each of N systems only needs to keep track of the systems in its portions of the hierarchy, which is $HN^{1/H}$ systems.

The address required for this system is a locator, such as an IP address. Because the locator is source-independent, the cost of deriving an address from a name is the same as with identifiers, that is, N. Thus, the total system cost using locators is roughly $NHN^{1/H}$, a significant decrease from flat routing. All of the decrease comes from savings in the path information in routers.

Finally, consider a true source-routing system, where routers only maintain path information for their immediate neighbors, and packet headers contain full source routes (vectors). The cost of the routing system is roughly EN, where E is the average number of neighbors in the topology that each system has. This is a significant reduction over the hierarchical routing system.

The cost of the naming system, however, is roughly DN^2, where D is one-half the diameter of the network. This is because each of N names must map into N vectors, one for each source, and each vector has on average D components.

Thus, the total system cost using vectors is roughly N^2 (D and E are for all practical purposes small constants). This is the same cost as using identifiers for addresses.

Thus it can be seen that the choice of an addressing scheme is, in part, one of balancing the cost of addressing (name to address binding) against the cost of routing (address to route binding). It is easy to decrease the cost of either routing or addressing, but it is hard to decrease the cost of both. Of the three examples above, hierarchical addressing has the lowest overall cost, because it decreases the cost of routing without significantly increasing the cost of addressing. In general,

Table 3.1: Summary of Routing and Naming Costs

Routing Type	Costs		
	Routing	Naming	Rough Total
Flat Routing	N^2	N	N^2
Hierarchical Routing	$NHN^{1/H}$	N	$NHN^{1/H}$
Pure Source Routing	EN	DN^2	N^2

the lowest overall cost requires the use of some form of locator.

It is worth noting that the scaling benefit of locators is not limited to the case where hosts do not move often. If hosts move often, then the naming system must be updated often, which increases the cost of maintaining the naming system, thus increasing the overall cost of using locators. When identifiers are used, however, host mobility increases the cost of routing, because routes to a host must be modified when the host moves. Thus, mobility increases the overall cost when either identifiers or locators are used. Because of this, the overall cost of hierarchical addressing versus flat addressing, even with host mobility, is still less.

Chapter 4

Vectors

Vectors are not in common use, so this section is quite brief compared to the previous section on locators. Indeed, to the author's knowledge, the only internationally standardized protocol that uses a vector for normal operation is the source routing bridge protocol for IEEE 802 LANs [83].

Saltzer, Reed, and Clark [95] give a number of reasons for using vectors (called source routes in [95]) rather than locators (or, the more general notion of hop-by-hop routing in [95]) in the packet header.

1. Separation of routing from identification.
2. Gateway simplicity and network maintenance.
3. Route control.

The first reason, while a good idea, does not necessarily follow from the use of vectors per se. That is, identification can be separated from routing even when locators are used, simply by including an identifier separate from the locator. And, use of vectors does not necessarily mean that pure identification is used, as the elements of the vector can be locators themselves, as with IP.

The latter two reasons are the more commonly cited reasons for using vectors rather than locators. The second reason is the motivation behind source-routing bridges in IEEE 802.5. The use of source routing minimizes the state required in the bridges since no forwarding table is required. Of course, this pushes the burden of finding routes on some other system. In the case of IEEE 802 source routing, the burden is pushed onto hosts, which discover paths using a flooded search. This method of path discovery can, in the worst case, cause an explosion in search packets, creating more problems than it solves [82].

The notion of simplifying forwarding for routers is taken further by Sirpent [19] and by Paris [20]. These two protocols propose that the elements that make up the vector are designed so as to best assist the router with regards to its forwarding implementation. In the case of Sirpent, each

element contains the local tag used by the router to identify the next hop. In the case of Paris, each element contains the binary self-routing code used to route the packet through the switch fabric.

It is interesting to note the similarity between Paris and the the first use of the telephone number. The telephone number was originally nothing more than a means of producing the sequence of electric pulses required to drive the step-by-step switches in the central office. With step-by-step switches, each digit of the dialed number indicated how to route to the next switch element (and in fact electronically drove the switch hardware). With Paris, each bit in each element of the vector indicates how to route through the next binary element of the switch fabric.

With regards to route control, early work in source routing [95] speculated on the potential for allowing the source to choose a path appropriate to the source's requirements. For instance, trouble isolation, policy, class-of-service, and in-order packets are cited as uses for source routing. These topics, however, were not pursued in detail. More recently, Perlman, in her thesis [81] explored in detail the use of source routes to discover and route around misbehaving routers.

Some recent papers consider the use of source routes for policy reasons [11, 35, 94, 101, 21]. Policy routing can broadly be defined as the capability to choose among multiple paths from source to destination. This capability is usually exercised by the source. Common reasons for choosing among multiple paths are 1) some paths are administratively restricted for a given communications, 2) some paths are cheaper than others, 3) some paths do not offer adequate service.

There are several advantages to using a source routing approach for policy routing. First, every source may have its own policy constraints (for instance, certain acceptable use or billing policies). It is most efficient to limit distribution of this policy information to the sources themselves. Second, it may not be feasible to globally distribute policy information about transit networks. Further, some sources may have less need for detailed transit policy information than others. With a source routing approach, it is possible for sources to cache only the information they need, and from that information calculate the appropriate routes.

Note that this caching approach is a fundamentally different approach to scaling compared to aggregation. With aggregation, contiguous portions of the network are abstracted as a single element. This abstraction is reflected through common addressing. Routers maintain some level of information about all portions of the topology, albeit indirectly through abstraction. In the case of hierarchical clustering, routers maintain more detailed information about nearby destinations, and less detailed information about more distant destinations, but none-the-less maintain some level of information about all destinations. In the case of a grid topology, all destinations are viewed at the same level of abstraction.

With the source routing/caching approach described above, no system necessarily has routing information (aggregated or otherwise) about all destinations. At a minimum, routers may only have information on how to reach their neighbors (where a neighbor could be a neighbor router

or a neighbor network). Sources may have no information about many network destinations, for instance, if they have no traffic to send to those destinations. A source can gather and cache new routing information if the need arises.

One of the difficulties of the source routing/caching approach is distribution of the routing information. There is a chicken-and-egg problem whereby some routing information must be established to carry more detailed routing information to where it is needed. One approach to this problem is a hybrid of hierarchical aggregation and source routing/caching, as suggested by Estrin and Rekhter [36]. Hierarchical aggregation is used for common routes. Sources that have no special policy needs can use these routes for everything. Sources that have special policy needs can use these routes to obtain additional topology information.

It is worth noting that caching as used here is not the same as caching as used in the context of a VCI. A VCI is simply a compression of a header with complete routing information (locators or vectors). (A VCI may also be compressing traffic characterization information. This aspect of the VCI is outside the scope of this thesis.) A VCI is useful for a number of things, such as making packets smaller or increasing switching speed, but it has no positive scaling effects (in the context of routing), nor any effect on the overall architecture of routing and addressing.

Chapter 5

Layering Addresses

This section discusses the issue of how a router determines the subnetwork address to use when it forwards a packet over a subnetwork. It first gives some background and describes the problem. Then it describes an approach to the problem whereby subnetwork addresses are embedded in internet addresses. Finally, it discusses the shortcomings of this approach, particularly in the context of CLNP.

5.1 Background

When a router receives a packet, it must decide which internet system (router or host) should next receive the packet. Mechanistically, what the router must do is encapsulate the internet packet in a subnetwork header with the subnetwork address of the next hop and transmit the packet onto the appropriate subnetwork. Exactly how this is done depends on the situation.

Consider the case where the internet protocol is IP, and the router receiving the packet is on the same subnetwork as the destination host, and the subnetwork in question is a broadcast LAN. By a simple comparison of the destination IP address against the subnetwork mask of the subnetwork [10], the router can determine that the destination host is on the subnetwork. The router then broadcasts a search packet (called an ARP request [84]) onto the subnetwork. All hosts receive the packet, and the host whose IP address matches that in the query responds to the router with its subnetwork address. The router caches this information for future packets, and transmits the packet to the host using this subnetwork address.

Consider the same case as the previous paragraph, but where the internet protocol is CLNP [55]. In this case, there is no subnetwork mask that the router can use to determine if the host is on the subnetwork. Instead, the router has a list of all hosts on the subnetwork, along with their subnetwork addresses, that it obtained by listening to announcements from all hosts [53]. Using this information, the router can transmit the packet to the host.

Now, consider the case where the destination host is not on any of the router's attached subnetworks, but where the subnetworks are still relatively small (LANs or point-to-point links, for instance). In this case, the router will have previously learned, probably via a broadcast mechanism, of all other routers on the subnetworks, and will have a forwarding table showing which destinations should be forwarded to which routers. This table can have been created by the network administrator or by automatic discovery and routing protocols.

5.2 The Problem—Large Subnetworks

The common theme in the three cases of the previous section is that the routers on the subnetworks are capable of learning the subnetwork addresses of every system attached to the subnetworks. This is possible because the number of systems on the subnetworks is small. Problems arise, however, in the case where the subnetworks have a very large number of systems attached to them, such as might be the case with X.25, SMDS, Frame Relay, or ATM subnetworks. In this case, it is not possible for routers to maintain information about all other attached systems, or to use a broadcasting method to discover the subnetwork address of any attached system.

Assume that a router has received a packet for some destination host. There are two major cases of interest. The first case is where either the destination host or the destination host's private network are attached to the router's subnetwork. In this case, the problem is one of the router learning the subnetwork address (or addresses) of the host or of the host's private network. This problem is very similar to that of learning the internet address of a host given its name. In the case of naming, it is a relatively static one-to-one or one-to-many mapping of name to internet address(es). In the case of subnetwork address discovery, it is a relatively static one-to-one or one-to-many mapping of destination internet address to subnetwork address(es).

The second case is where either the destination host, or the destination host's private network, is not attached to the router's subnetwork, but instead is reached through one or more transit networks. In this case, the problem is less similar to the name-to-address mapping problem. The destination internet address must be mapped into the subnetwork address of the next hop router on the path to the destination host. The choice of next-hop router may not be relatively static, as it depends on the state of the topology, the routing algorithms being used, and so on.

This second case is a more difficult problem in terms of algorithmic complexity, since it must take routing into account. The scaling problems, however, are not as severe as in the first case. In the second case, the number of potential next-hop routers is proportional to the number of neighbor transit networks. This is likely to be a much smaller number than the number of subscribers attached directly to the subnetwork. Thus, it is likely to be possible for a router attached to the subnetwork to maintain routing and subnetwork address information for routers attached to all other transit networks.

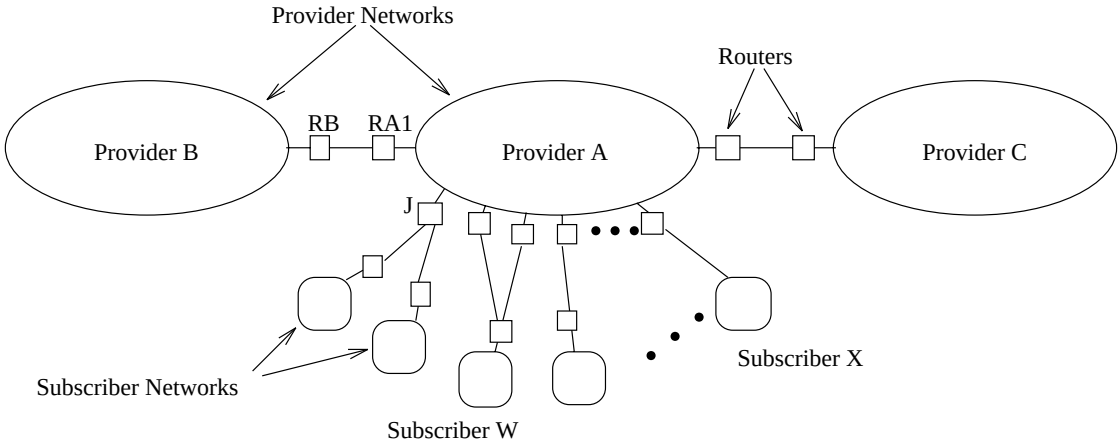


Figure 5.1: Address Layering Example

5.3 Embedded Subnetwork Addresses

There are two basic approaches to the problem of determining the appropriate subnetwork address from the destination internet address. One approach is where the appropriate subnetwork address of the host is embedded in the internetwork address of the host. In this case, the router can simply extract the subnetwork address from the internet address and form a subnetwork header with it. The other case is where the subnetwork address is not embedded in the internet address. In this case, the router must dynamically discover the subnetwork address, and then cache it for later use. The former case is of particular interest here, because it impacts the form and use of addresses. The latter case is discussed in [109].

Consider the topology of Figure 5.1. Provider A is a subnetwork of some sort with routers at its edges. It has routers connected to other providers and routers connected to subscriber networks. Assume that a packet is being sent to subscriber X attached to provider A. Without embedded subnetwork addresses, the destination address might look like:

providerA.subscriberX.area.host

Routers outside of provider A (for instance, Router RB) would look at the first part of the address (*providerA*) and route it to a router at the border of provider A (Router RA1). That router would then look at the next hierarchy level, *subscriberX*, and have to determine from that which subnetwork address to route the packet to. If there is some discovery mechanism available to provide this information, the router can get the information and forward the packet across the subnetwork to the router that connects subscriber X with provider A.

Note that the difficult problem is not one of getting the packet from one provider to another provider. The difficult problem, rather, is in getting the packet from the entry point of the last provider in the path to the subscriber. This is because the number of subscribers for a given

provider is much larger than the number of neighbor providers.

If, on the other hand, the embedded subnetwork method is used instead, the address might look like this:

providerA.subnetAddrX.area.host

In this address, the embedded subnetwork address (*subnetAddrX*) is in part taking the place of the subscriberX part of the previous address. That is, the *subnetAddrX* field uniquely identifies subscriber X among all other subscribers. But, this identification is indirect, because explicitly, the *subnetAddrX* field is identifying the subnetwork interface attached to the router that is attached to the subscriber. If that router is exclusive to subscriber X, then subscriber X has indirectly been identified.

More to the point, however, is that the provider X router receiving this address can extract the *subnetAddrX* field, put it in the destination address field of the subnetwork packet (or call setup), and forward the packet. If a router at a subnetwork interface is attached to multiple subscribers, such as router J in Figure 5.1, then further subscriber identification information in the address would be necessary, for instance:

providerA.subnetAddrY.subscriberY.area.host

Embedding subnetwork addresses in internet addresses is very useful in many cases, and has been specified for use in at least one international standard [56]. It was also used in the early IP internet, to get packets across the ARPANET (the ARPANET switch and port numbers were embedded in the IP address). However, it does not work in all cases.

First, it does not work in the case where the subnetwork address is too large to fit in the internet address and still leave enough space for the other addressing information the internet address must contain. This is generally the case with IP, as most subnetwork addresses (IEEE802, X.121, E.164) are larger than the IP address.

The NSAP address, on the other hand, was designed with incorporation of subnetwork addresses in mind [54]. Thus, it incorporates the subnetwork addresses mentioned in the last paragraph. Interestingly enough, however, a standards group for development of ATM subnetworks (the ATM Forum) subsequently adopted the NSAP address for use in addressing ATM interfaces. Thus, the NSAP address is not big enough to contain all subnetwork addresses[1].

Second, the embedded subnetwork address specifies the path to the destination at a higher level of detail than the other fields of the address. The other fields, provider, subscriber, area, and so on, identify groups of systems, not single systems (or interfaces). The subnetwork address, on the other hand, specifies a single interface. This is overly constraining in the case where there are

[1] From this, one could formulate an axiom—which I call the Deering Axiom, because Steve Deering first pointed out this conundrum—that an internet address can never be big enough to hold all subnetwork addresses, because a future subnetwork can always adopt the internet address.

multiple routers entering a subscriber network from the same provider.

For instance, in Figure 5.1, subscriber W has two connection points to provider A. If one of the subnetwork addresses is embedded in the internet address, then the router connected to the other subnetwork address will not be found in case the original one fails.

Third, the subnetwork address must be embedded in the appropriate part of the address, and in such a way that it's coding is clear. The pitfalls of a bad encoding can be illustrated with the NSAP address. When a subnetwork address is encoded in an NSAP, the subnetwork address is the most significant field with location semantics. (There are fields with more significance than the subnetwork address field, but these fields only tag the address as being a certain type, and do not have any location semantics.)

Depending on the nature of the subnetwork address, it may or may not have enough hierarchy, or the right kind of hierarchy, to scale sufficiently. For instance, the E.164 address has a country code as its most significant field. As discussed in section 3.4.5, geographical addresses may not scale well with the existing internet topology.

A related problem is that it is may not be clear to a router whether the subnetwork address given is on its own subnetwork or another subnetwork. For instance, a packet with an E.164 address embedded in it might cross multiple subnetworks using E.164 addresses—SMDS, ATM, Frame Relay, or BISDN. Because E.164 addresses are geographical, and because multiple providers can cover the same geographic territory, the router has to have explicit information about what E.164 addresses are on the provider's network. The amount of information required to know this may be comparable to the amount of information needed for subnetwork address discovery in the first place.

Note that this problem does not exist with the addresses shown in this section, where the most significant field is provider. If the most significant field is provider, then the router attached to the identified provider knows that the subnetwork address must be on its subnetwork.

Fourth, it is possible that there should be multiple subnetwork addresses in the internet address. For instance, consider the case where Subscriber X's internal network was itself a single large subnetwork with hosts directly attached. In that case, to solve the problem of subnetwork address discovery using the embedded subnetwork address technique, the internet address would be:

providerA.subnetAddrX.subnetAddrHost

where subnetAddrHost is the subnetwork address of a host on the subscriber's subnetwork. The existence of multiple subnetwork addresses only compounds the above problems.

In the cases where the problems or limitations associated with the embedded subnetwork address approach make it unacceptable, a dynamic discovery method must be used, as discussed in [109].

Chapter 6

Group Addresses

The previous chapters dealt essentially with unicast addresses—addresses that identify (or locate) a single system or interface. An increasingly important class of address is the *group address*—an address that identifies multiple systems or interfaces. A group address can have two basic services associated with it—multicast service and unicast service. With multicast service, all systems identified by that address receive a copy (possibly unreliably) of the packet. The term *multicast address* indicates a group address with multicast service.

With unicast service, only one of the group of systems identified by the group address should receive a packet destined for the group. The term *anycast address* indicates a group address with unicast service.

The following sections discuss multicast and anycast addressing.

6.1 Multicast Addresses

A multicast address allows the source of the packet to send that packet to multiple other systems without having to know the separate identities of those systems, and without having to send multiple packets. This simplifies the host considerably (though it complicates the routers). It also decreases the latency of packet delivery compared to sending of multiple packets, because the last of multiple packets would be queued up behind the others leaving the host. Finally, it improves bandwith efficiency, because only one copy of any given packet is transmitted over any link.

The usual model of multicast packet delivery is that the multicast packet traverses some kind of tree, logically overlaid on the physical topology which is typically a mesh of some kind. The sender is at the root of the tree, and the receivers are at the leaves. At every branch in the tree, the packet is replicated, and one copy sent over each branch. Thus, the latency remains low, as no wire carries more than one of a given packet.

There are many important issues concerning multicast—what the nature of the tree is (source rooted [27], centered [2]), how the tree is formed, scaling, managing resources, distributing addresses, controlling group membership, achieving reliable packet delivery, and so on [8, 26, 90, 112, 2]. In keeping with the scope of this thesis, however, the discussion here is largely limited to the form of the multicast address and related information required in the packet header.

A fundamental distinction among tree types is *source-rooted* trees versus *non-source-rooted* trees. Source-rooted trees are shortest path spanning trees where the root of the tree is at the source of the packet [27]. Non-source-rooted trees are any other type of tree, for instance a shortest path spanning tree rooted at some system which is not the source [2], or a Steiner tree [112].

Depending on the nature of the multicast tree, different information in the packet is used by the router in making the forwarding decision. Consider first a case where all multicast packets in a network traverse the same non-source-rooted tree, and where that tree extends to all hosts in the network. This is called a *broadcast non-source-rooted* tree. In this case, all multicast packets reach all hosts, and the hosts either accept or discard the packet based on whether or not they are a member of the group identified by the multicast address. With this kind of tree, each router maintains a list of which interfaces are on the tree. When it receives a multicast packet, the router simply forwards the packet over all tree interfaces except the one it came in on.

Thus, the only information from the packet header that the router needs is the information that tells it that the packet is a multicast packet. In the case of IP, this can be done by looking at the first four bits of the destination IP address. If those four bits are value 1110, then the packet is a multicast packet. Otherwise it is not. The host, on the other hand, looks only at the full multicast address to determine if it should accept the packet (this is true for the following three examples).

Next, consider the case where there is one tree per source host (or, per source host that is actively sending multicast packets), and that each tree extends to all hosts in the network. These are called *broadcast source-rooted* trees. With a broadcast source-rooted tree, the router expects the packet for a certain source host (or set of hosts, as identified by an address prefix) to arrive on a certain link—the link that the router would normally route a unicast packet over to reach the the host (as a destination). The router also knows, for each source, which outgoing links to forward the packet over.

Thus, for the broadcast source-rooted tree, the information from the packet header that the router needs is 1) whether or not the packet is multicast, and 2) the source address.

Next, consider the case where there is one non-source-rooted tree per multicast group, and that each tree extends only to the members of the multicast group. This is called a *multicast non-source-rooted* tree. In this case, a router must know which of its links are on the multicast tree for each group. When a router receives a packet for the group, it forwards the packet over all of the links on the tree except the one it came in on.

For a multicast non-source-rooted tree, the packet header need only carry the identifier for the

multicast group. (The value of the identifier indicates that the packet is a multicast packet. Therefore, no separate "this is a multicast packet" information is required.)

Finally, consider a *multicast source-rooted* tree. In this case, the packet header must contain both the identifier for the group, and the address of the source of the packet.

In the above examples, the source address, when it is used, is functionally a modifier. The (destination) multicast address, when it is used, is an identifier.

6.1.1 Scoped Multicast

With scoped multicast, a multicast packet is only allowed to travel a certain distance from the source of the multicast packet, where distance can be measured in various ways. A simple way to measure distance is by number of hops away from the source. This is used with IP by the setting the Time-to-Live (TTL) field. This field is decremented at least once (and usually once) every time a router forwards the packet.

With IP unicast, routers discard a packet whose TTL has decremented to 0. With IP multicast on the MBONE, the discard threshold is settable, so that routers can tune the amount of multicast traffic they forward. In addition, the initial setting of the TTL is based roughly on the amount of bandwidth generated by the application, so that video, for instance, will have a lower initial TTL than audio. Thus, a router's threshold can be configured to discard video while accepting audio. (Note that this particular use of the TTL on the MBONE is not necessary if multicast pruning is implemented—that is, individually selecting which multicast groups can traverse a link.)

The usual desired scoping semantics is not hops, but rather to keep the packet within certain boundaries, for instance, the local LAN, the departmental networks, the company networks, and so on. Thus, hop count is not usually a good mechanism (except for the local LAN case, where a hop count of 1 suffices).

An alternative mechanism, specified in the SIPP protocol [30], is to have a scope field that explicitly defines the scoping boundary. The hop-count method requires no special handling by the router (it is normal procedure for a router to drop a packet whose hop-count has expired). The scope field method, on the other hand, requires that an explicit forwarding decision be made by routers.

A scope field can either be interpreted in conjunction with the source locator, or independent of the source locator. In the latter case, a router simply associates each of its interfaces with zero or more scope values. A packet with a matching scope value is never transmitted over that interface. In the former case, the router associates each interface with zero or more ⟨scope value, source locator prefix⟩ pairs. A packet with a matching scope value *and* a matching source locator prefix is never transmitted over that interface. Source-independent scoping is simpler, and source-dependent scoping is more general.

With source independent scoping, a single series of expanding concentric scoping boundaries is defined for each host—that is, with respect to all of its locators. With source independent scoping, potentially multiple such concentric boundaries are defined for each host, if it has multiple locators. For either case, the group members for a given packet are defined by the combined ⟨scope, source locator, group ID⟩ information. For forwarding a packet, however, the router does not care about the source locator unless it is doing source-dependent scoping (or of course source-rooted trees).

6.1.2 Well-known Multicast Addresses

Another form of multicast address is the "well-known" multicast address. The well-known multicast address is a normal multicast address in that it identifies a set of nodes. It is different, however, in that the set of nodes that it identifies offer a particular service, and that that service is identified by the value of the well-known multicast address. An example of a well-known multicast address with IP is "all OSPF routers" [73]. That is, all OSPF routers will accept a packet addressed to the "all OSPF routers" multicast address.

The desired recipients of most well-known multicast addresses are local in scope. For instance, the "all OSPF routers" well-known multicast address must only go to all OSPF routers on the local LAN. Thus, the scope must be local LAN (or, hop count of 1). This allows nodes to communicate to other nodes of a certain type without going through an address discovery process.

A well-known multicast address, combined with the scope and source locator, defines multiple groups, each within a prescribed scope.

6.2 Anycast Addresses

Like a multicast address, the anycast address identifies a set of nodes. Unlike a multicast address, however, the delivery semantics for an anycast address is to deliver the packet to one node only. Normally, but not necessarily, the single node is assumed to be the "closest", according to the routing protocols notion of closest [76].

The router forwarding mechanism of an anycast address is closer to that of a unicast address than that of a multicast address. A router can treat an anycast address like a unicast address in that it calculates a single path to one of the members of the anycast group. Indeed, most routing algorithms designed for use with unicast addresses work without modification for anycast. (By and large, a routing algorithm can not distinguish between multiple paths to a single (unicast) destination and multiple anycast destinations.)

Because of this similarity with unicast, anycast addresses can be similar to unicast addresses. For instance, anycast addresses can be hierarchical with the same scaling benefits as unicast addresses. Indeed, one form of anycast address is one that is indistinguishable from unicast addresses, but

that happens to be assigned to multiple nodes.

The SIPP protocol takes advantage of a specialized form of hierarchical anycast addresses that come out of the unicast address space [40]. In SIPP, they are called cluster addresses, and they are for the purpose of sending a packet to any one of the border routers of a cluster of nodes defined by a hierarchical prefix. (A SIPP cluster address is encoded as an address prefix followed by all zeros.)

Another example of an otherwise unicast address having anycast semantics is the Core Based Tree (CBT) multicast protocol [2]. This is discussed in the next subsection.

A potentially useful form of anycast address is the well-known anycast address. Anycast addresses can also be similar to multicast addresses, in that they can be (non-hierarchical) identifiers. This form is particularly useful for well-known anycast addresses where, like well-known multicast addresses, they identify a service as well as a group of nodes.

Scoping can be used with this form of anycast address. The effect of scoping is similar to that used with multicast in that no nodes outside of the area defined by the scope will receive the packet. As with multicast, the source address may or may not be used in defining the scope boundary.

6.3 Two-phase Group Addresses

A useful form of group addressing is two-phase group addressing, where a packet is initially routed as unicast (or anycast), and subsequently is routed as group-addressed (multicast or anycast).

One example of this is remote multicast, where a packet is sent to a remote network (using unicast), where it then becomes multicast with local scope. This causes the packet to be delivered to a multicast group that is local to a remote network. This style of multicast is advantageous compared to regular (single-phase) source-rooted multicast in that with remote multicast, routers in between the source and the remote network do not need to have any knowledge of the multicast group, whereas with regular source-rooted multicast, they do.

A similar case can be made for remote anycast.

Another example of two-phase multicast is the CBT multicast protocol [2]. With CBT, a single multicast tree is formed from some root (core) router that may not be a sender (or may not be the only sender) to the multicast group. A node may send to the group without being a member of the group. When this happens, the sending node forms a packet that has the unicast address of the core and the multicast address of the group. The packet is initially routed towards the core. When it reaches any router on the multicast tree, that router changes the phase of the packet and routes it as a multicast packet (along the multicast tree).

The advantage of this approach over the source-rooted tree is that routers need only keep per

group forwarding information, rather than per group and per source forwarding information. The disadvantage is that the tree formed is less optimal than the source-rooted tree, particularly when the core router is poorly placed.

Part II

Internetwork Protocol Header Design

In Part II of this thesis, we consider the design of the routing and addressing aspects of an internetwork protocol. In particular, we explore two designs—one based on a conventional internetwork protocol header syntax [30, 40] (SIPP), though with expanded semantics, and one based on a new internetwork protocol header syntax (SPip). These two designs are compared with a conventional internetwork protocol, CLNP [55].

We do not consider aspects of internetwork protocol design not related to routing and addressing. These aspects include higher layer protocol identification, hop count limitation, checksum calculation, and fragmentation and reassembly. All of these functions are largely orthogonal to routing and addressing, and are therefore not considered in this thesis.

Since at the time of this writing the SIPP protocol is a candidate for replacement of IP, a note on the history of SIPP is appropriate. During the two years that I worked on this thesis, I was actively promoting my protocol ideas in the Internet Engineering Task Force—the standards body that has oversight of TCP/IP and related protocols—as one of the candidate replacements for IP. My initial proposal was Pip [38, 39, 37], a variation on SPip presented in this thesis (SPip, pronounced "ess pip", stands for Simple Pip). Later, the Pip project merged with another candidate, SIP (for Simple Internet Protocol), creating SIPP (SIP Plus). SIPP kept the syntax of SIP, but expanded its semantics.

Chapter 7

Evaluation Criteria

This chapter describes the evaluation criteria for the protocols. We are interested in the capabilities and the costs of each of the protocols.

7.1 Costs

We consider 4 costs:

1. Processing cost (speed and hardware complexity)
2. Address assignment complexity
3. Control protocol (such as routing) complexity
4. Header size

It is in fact almost impossible to be precise with respect to these costs, as they depend on so many factors. Thus, we treat these costs more in terms of general arguments than precise analysis.

For processing cost, we focus primarily on aspects of software implementation rather than hardware. The reason for this is partly the author's unfamiliarity with hardware, and partly the fact that there are a large number of hardware approaches, and we can not cover them all. In any event, software implementation alone provides a good basis for relative comparison, as discussed in Section 10.1.

For address assignment complexity, we consider both address autoconfiguration (mainly of hosts), and manual assignment along an address assignment hierarchy. With respect to the former, we are interested in whether "serverless" address autoconfiguration is possible or not. With respect to the latter, we are primarily interested in the difficulty of assigning addresses considering internetwork growth and change.

With respect to control protocol complexity, we primarily limit ourselves to pointing out where certain aspects of the internet protocol design make the control protocols more complex. In general, we use the IP control protocols as baselines in this evaluation.

Of these four costs, the header size is the one that is easy to quantify. It is, however, difficult to quantize what the true cost of a large header is. At high link speeds, where most packets can be fairly large[1], a large header constitutes only a small percentage of packet size.

At very low link speeds, even a moderate packet size is intolerable, and so some kind of header compression (for instance, using a VCI [17, 16]) is required. With header compression, however, a large (uncompressed) header will not behave significantly worse than a moderate header if a large percentage of headers can be compressed successfully (that is, fall under an existing VCI).

7.2 Capabilities

The capabilities of interest, summarized in Table 7.1, are the routing and addressing functions described in Part I. The capabilities are partitioned into two categories, required and useful. The purpose of the two categories is to show the different relative importance of the various capabilities.

The required capabilities are widely accepted as necessary for any future internetwork protocol. The useful capabilities are those for which there is less agreement as to whether or not they are worth the complexity. The inclusion of a capability in one category or the other is in some cases a judgement call. In case of doubt, I used the current working status of the capability in the IETF standards community [48] to choose the category. That is, if the IETF is actively working on the capability, I included it in the required category.

[1] The exceptions are interactive data traffic such as telnet (where one character or only one line might be sent) and interactive voice, where low latency and relatively low bandwidth encoding (for instance, 32k bits per second) result in smallish packets. However, at high link speeds, this traffic is likely to constitute only a small percentage of total traffic, the majority taken up by image transmissions.

Table 7.1: Criteria for Routing and Addressing Capabilities

Capability	Section
required	
Big enough hierarchical unicast addressing	2.2.1
Multicast/shared-tree group addressing	6.1
Multicast/source-tree group addressing	6.1
Scoped multicast group addressing	6.1.1
Well-known multicast group addressing	6.1.2
Mobility	2.2.4
Multicast/two-phase group addressing	6.3
Domain-level policy route	4
Host Auto-address assignment	
useful	
Type-of-Service Field	2.2.3
Embedded link-layer address	5
Node level source route	4
Anycast group addressing	6.2
Anycast/two-phase group addressing	6.3

Chapter 8

Protocol Descriptions

This section describes the three protocols analyzed in Part II of this thesis. Note that we only describe the part of the protocol having to do with the identification, location, and path modification functions. Unless otherwise stated, the term *internet protocol* is assumed to refer to only that part of the internet protocol that accomplishes these three functions.

8.1 SPip

Perhaps not surprisingly, the design of SPip follows directly from the main observations made in Part I, namely, that

1. all routing and addressing functions can be classified as one of identification, location, or path modification, and
2. a locator is simply a series of identifiers, and the process of locating is that of routing in turn to each of the identifiers.

Based on item 1 above, we argue that (the routing and addressing portion of) an internet protocol should have three and only three parts—an identifier, a locator, and a path modifier (actually, two identifiers and two locators, one each for source and destination).

Based on item 2 above, we argue that the locator part should be based on a loose source route-like mechanism, with each element of the loose source route being a simple identifier (versus a complete hierarchical address as with IP).

This is, in a nutshell, the design of SPip. The arguments for this approach are made by way of showing, in Chapter 9, how SPip handles each of the routing and addressing capabilities with a single simple forwarding engine, and with straightforward control protocols (comparable to those required for IP or SIPP).

8.1.1 SPip Routing and Addressing Fields

The (routing and addressing part of the) SPip header has the following format:

64 bits	64 bits	variable
Source EID	Dest EID	Route Sequence

The Source EID and Dest EID fields are endpoint identifiers. They have no locator or path modifier semantics. They are flat in-so-far as SPip is concerned[1].

Higher layer protocols such as TCP [88] use the Source and Dest EIDs to identify the connection endpoints. When an SPip host receives a packet, it only need examine the Dest EID to determine that the packet is destined for itself.

The Route Sequence contains the location and path modification information. The Route Sequence is formatted as:

8 bits	8 bits	32 bits	32 bits		32 bits
Num Source RSE	Active RSE	RSE1	RSE2	$\cdots$	RSEn

where RSE stands for Route Sequence Element[2].

The RSE has a 1-bit flag followed by two parts, a 7-bit Path Modifier and a 24-bit RS Identifier:

Route Sequence Element (RSE)		
1 bit	7 bits	24 bits
Last RSE	Path Modifier	RSID

The Last RSE flag is set to 1 if this is the last RSE in the route sequence, and set to 0 otherwise.

The following sections describe the use of the SPip header.

8.1.2 SPip Forwarding Algorithm

Three local variables are maintained during the forwarding algorithm:

activeTable This indicates which of multiple forwarding tables should be accessed. There are two types of tables, *RSETables* and *EIDTables*. RSETables are accessed with RSEs, and EIDTables are accessed with EIDs.

activeRSE This indicates which of multiple RSEs, or the EID, is used to access the activeTable. If activeRSE is non-zero, it indicates an RSE, and indicates the EID if it is zero.

[1] The Source and Dest EIDs have a certain amount of structure to facilitate their assignment (see Section 8.1.4). This structure, however, has no bearing on operation of the SPip protocol.

[2] Route Sequence Elements are called FTIFs (Forwarding Table Index Fields) in the original Pip protocol [37, 38, 39].

transmitRSE This indicates what value the Active RSE field should be set to upon transmission of the packet. For the case of multicast, it applies to all transmitted packets.

The forwarding algorithm is illustrated in Figure 8.1 and described as follows:

1. Set activeRSE and transmitRSE to the value of the Active RSE field in the received packet.
2. If activeRSE is zero, set activeTable to be the Main EIDTable, and go to step 3. Otherwise, set activeTable to be the Root RSETable, and go to step 4.
3. Set activeRSE to zero, and index the active EIDTable (indicated by activeTable) using the value of the Dest EID field extracted from the packet header. The indexed entry will return either:
   (a) a pointer to a Forwarding Information Base (FIB) entry (go to step 6), or
   (b) error.

   The indexed entry also includes a tag that indicates if transmitRSE should be set to the current activeRSE (that is, advanced to point to the EID).
4. Index the active RSETable (indicated by activeTable) using the value of the active RSE field (indicated by activeRSE) extracted from the packet header. The indexed entry will return one of:
   (a) a pointer to a Forwarding Information Base (FIB) entry (go to step 6),
   (b) a pointer to one of the EIDTables (go to step 3),
   (c) a pointer to another RSETable, possibly the same one already accessed (go to step 5), or
   (d) error.

   The indexed entry also includes a tag that indicates if transmitRSE should be set to the current activeRSE (that is, advanced to point to the current RSE).
5. Increment activeRSE. Set activeTable to be the one pointed to by the entry from the previous step. Go to step 4.
6. Using the information from the FIB entry, forward the packet over zero or more interfaces. The FIB information includes the interface and link-layer header for each packet to be transmitted. The Active RSE field of the transmitted packet is set to transmitRSE.

This algorithm describes the RSETables and EIDTables as flat tables that are directly indexed by the RSE or EID respectively, where every entry has one of the choices enumerated above (a pointer to some outcome or an error). Given that the RSE space is 2^{32} values, and that the EID space is 2^{64} values, it is clear that such an implementation is impossible. However, the semantics of such an implementation must be achieved.

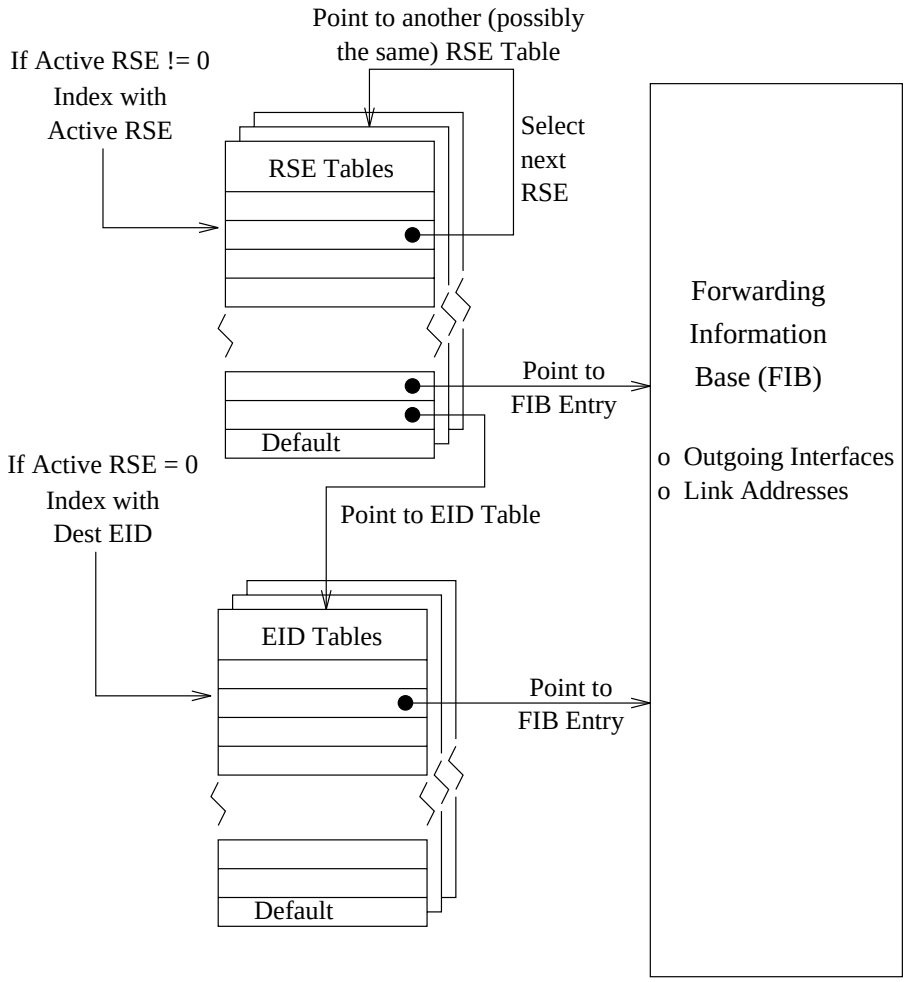


Figure 8.1: SPip Forwarding Algorithm

The large majority of the entries in the RSETables or EIDTables contain either errors or default routes. Both represent the case where no explicit match is found. Thus, the semantics of the RSETable or EIDTable lookup is exact-match with default. That is, either an exact match is found, or a default entry (which may contain a default route or an error indication) is used.

In the algorithm above, the index is described as using the whole RSE—the Last RSE flag, the Path Modifier, and the RSID. These three fields, however, are separate and orthogonal elements of the RSE. Thus, it is possible to default on the RSID but still retrieve different entries based on the value of the Path Modifier and Last RSE. In particular, a set Last RSE flag will generally cause the router to examine an EIDTable, usually the Main EIDTable. The use of defaults in general is described in the examples of Chapter 9, for instance, Section 9.2.3.

Note that the SPip forwarding algorithm is described in completely mechanistic terms. This is intentional, and is possible because SPip reduces routing and addressing to its elemental functions—identification, location, and path modification. In a sense, the SPip forwarding engine is like a

machine language. A machine language is defined in terms of the machine—that is, setting register values, moving words from here to there, and so on. The semantics is derived from the machine language through a computer program.

SPip is a kind of machine language for forwarding packets. SPip executes the basic functions, and various routing and addressing semantics can be derived with various routing algorithms, address assignments, packet formats, and so on.

8.1.3 SPip Packet Formation

Every SPip host stores locally one or more EIDs that identify itself (note that such an EID may represent a multicast group, and therefore identifies other hosts as well). Some of these EIDs are send-capable and others are not. A send-capable EID is one that can validly be used as the host's source EID in a transmitted packet. Multicast EIDs are not send-capable. Unicast EIDs are.

Associated with every EID are zero or more address sequences that can be thought of as representing its location in the internet. An address sequence is a series of RSIDs $\langle A_n,\ A_{n-1}, \ldots,\ A_1,\ A_0 \rangle$. When SPip hierarchical unicast addressing is used, each RSID in the address sequence carries one level of the hierarchical address (for instance, network, subnet, or host). This is discussed in detail in Section 8.1.5.

For every destination (group or individual) that a host sends packets to, the host has one or more EIDs, and associated with each EID is one or more address sequences (where an address sequence can have zero addresses, for instance in the case of multicast).

When a packet is transmitted from a source host S to a destination host D, the Source EID contains one of S's EIDs, and the Dest EID contains one of D's EIDs. The route sequence typically contains two address sequences—one from host S and one from host D. Each address sequence will be one of those associated with the corresponding EID.

The source host's address sequence is in the RSID part of the initial RSEs of the route sequence, in order of lowest order RSID first. The destination host's address sequence is in the RSID part of the trailing RSEs of the route sequence, in order of highest order RSID first. In between the two address sequences is zero or more RSEs, called transit RSEs. The Path Modifier part of the RSEs are filled in separately, depending on the desired routing. The Last RSE flag of every RSE except the last one is set to 0.

For instance, assume that host S's address sequence is $\langle S_n,\ S_{n-1}, \ldots,\ S_1,\ S_0 \rangle$, that host D's address sequence is $\langle D_m,\ D_{m-1}, \ldots,\ D_1,\ D_0 \rangle$, and that there are no transit RSEs. A route sequence in the packet from host S to host D would look like:

$$\langle S_0,\ S_1, \ldots,\ S_n,\ D_m,\ D_{m-1}, \ldots,\ D_0 \rangle$$

With transit RSEs T_0 through T_p, the route sequence would be:

$$\langle S_0,\ S_1, \ldots,\ S_n, T_0,\ T_1, \ldots,\ T_p,\ D_m,\ D_{m-1}, \ldots,\ D_0 \rangle$$

The source EID and address sequence are always locally known by the transmitting host. The destination EID and address sequences can be learned either from DNS or from a previously received SPip packet. The transit RSEs are learned through whatever means is appropriate to the application.

The Num Source RSE field is set to indicate how many of the initial RSEs represent the source address sequence. This tells a receiving host how to learn the address sequence of the remote host. Note that it can encompass more than just the source address sequence. For instance, if the transit RSEs represent a policy route, and the host requires that return packets follow the same route (in reverse), the Num Source RSE includes the transit RSEs. Its value, for the above example, assuming that no transit RSEs are included, is $n + 1$.

The default setting for the Active RSE field (that is, the setting used if no better setting is known) is to point to the first RSE after the source address sequence. Thus, the Active RSE field for the above example is $n + 2$. Other settings are possible, however, depending on the situation.

Obviously, if host S is to send a packet to host D, it must know an EID for host D, and zero or more address sequences associated with that EID (no address sequence may be required if, for instance, the EID represents a multicast group).

As mentioned above, there are two ways that host S can obtain the EID and address sequences for host D. One way is to receive it from DNS, and the other is to derive it from an SPip packet previously received from D. The former would normally be the case if host S initiates the exchange, and the latter would normally be the case if host D initiates the exchange.

The former case is straightforward. DNS (or some directory service) carries the EIDs and address sequences for hosts, and returns them when queried.

For the latter case, when host S receives a packet from host D, the EID of host D is in the Source EID field. The address sequence that host S should use to return a packet to host D is the reverse of the initial n RSIDs of the route sequence, where n is the value of the Num Source RSE field. (The Path Modifier fields of the received RSEs are set at the host's discretion.)

For instance, assume that the received route sequence is $\langle A_0,\ A_1, \ldots,\ A_{k-1},\ A_k \rangle$, and that the value of Num Source RSE is n ($n < k$). Host S forms an address sequence for host D of

$$\langle A_n, \ldots,\ A_1,\ A_0 \rangle.$$

The address sequence for host S used as the source address in the transmitted packet should generally, but not necessarily, match the tail of the route sequence in the packet received from host D.

8.1.4 SPip EID Definitions

There are two SPip EID types, individual and group. The individual SPip EID is used for unicast packet service, and the group SPip EID is used for multicast and anycast packet service. Roughly 3/4ths of the EID space is reserved for future definitions.

The primary goal in the definition of EIDs is to facilitate easy management of EIDs, particularly host autoconfiguration. The definition of EIDs relies heavily on IEEE-802 addresses [50]. It is assumed that systems running SPip will have local access to a globally unique IEEE-802 address—preferably one not associated with an IEEE-802 LAN interface. Thus, SPip systems can automatically create their own EID without coordination with any other systems.

Note that a specific non-goal of SPip EID definition is to allow recognition of a system's organizational affiliation—for instance, by putting an organization ID at the high order end of the EID. Putting organizational affiliation in the EID could be useful for several purposes, such as packet filtering or inverse DNS lookups. The advantages of organizational affiliation, however, are outweighed by the disadvantages, such as the increased complexity of autoconfiguration. Also, organizational affiliation can often be determined from the route sequence.

Individual EIDs have the following format:

16 bits	48 bits
hex 0000	Individual IEEE-802 Address

The first 16 bits of the unicast EID are hex 0000. The low-order 48 bits of the unicast EID contain an IEEE-802 Address. This IEEE-802 address must be an individual IEEE-802 address. If the IEEE-802 address is universally administered, then the corresponding SPip unicast EID is globally unique with high probability.

Group EIDs have the following format:

4 bits	4 bits	8 bits	48 bits
00x1	Scope	Local-Use	Group IEEE-802 Address

The first four bits are 0001 if the EID is multicast, and 0011 if the EID is anycast. Either way, the EID identifies the same set of systems.

The Scope field indicates a boundary over which the packet must not be transmitted. Typical values for the scope field are subnet, subscriber network, and global (no scope limitation). The boundary is independent of the source—that is, a router simply defines each of its interfaces as crossing zero or more boundary types, and any group EID with that boundary type is not transmitted over the link.

A scope value of 0 is reserved to mean "any scope value". Thus, when a host has discovered a group EID (say, through DNS or IGMP [28]) that it will use in a packet, if the scope field is 0, the host can validly set it to any other defined scope value. If the scope field is non-0, on the other

hand, the host may not modify it. A packet must not be transmitted with a scope value of 0 in the Dest EID field.

An IEEE-802 address occupies the low-order 48 bits. This must be a group IEEE-802 address. If the host interface is an IEEE-802 LAN, then the Group IEEE-802 address must be the same one that is used to receive packets for the corresponding multicast/anycast group.

The 8-bit Local-Use field allows a system responsible for the assignment of group addresses to create up to 256 different group addresses from a single IEEE-802 address. This makes the global management of group addresses trivial, as no coordination between hosts is required, and because a single host can create multiple group addresses. Note, however, that if multiple group addresses are created from the same IEEE-802 address, they will all be received over the same IEEE-802 address on the LAN.

8.1.5 SPip RSE Definition

The RSE is defined as follows:

		RS Identifier		
1 bit	7 bits	1 bit	19 bits	4 bits
Last RSE	Path Modifier	0	ID	Level

The Path Modifier field is a path modifier according to the definition given in Part I. As such, it can be set independently of the rest of the RSE. It could in theory be modified in transit by routers, but this is not part of the forwarding algorithm described above.

The low-order bit of the Path Modifier is used to indicate normal forward-path forwarding versus the reverse-path forwarding used for source-rooted multicast. A 0 value in the low-order bit indicates forward-path, and a 1 value indicates reverse-path forwarding. The former type is called the normal form RSID or normal form RSE, and the latter type is called the reverse-path form RSID or RSE.

A high-order bit of 0 in the 24-bit RS Identifier defines the RSID as being for use with unicast hierarchical addressing. A high-order value of 1 is at this time undefined, and can be used for future definitions.

The remainder of the RSID is defined and assigned identically for both normal-form and reverse-path form. The Level field (low-order 4 bits) indicates the hierarchical level of the RSE. Because the ID field is too small to be globally unique, the Level field is required to tell routers at which hierarchical level to route the packet. This is necessary because routers operate at multiple hierarchy levels.

To see this, consider Figure 8.2. Router R is in backbone B. Router R operates at two levels—the top level where it maintains information about other backbones, and at the next level down,

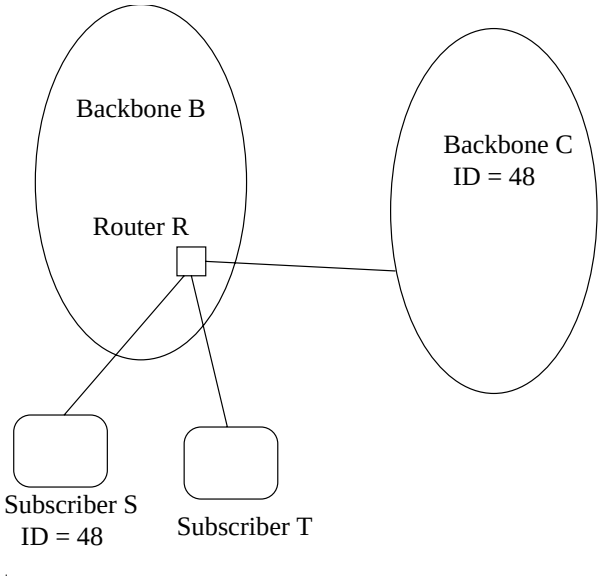


Figure 8.2: Router Operating at Multiple Hierarchical Levels

where it maintains entries for subscriber networks. Assume that the neighbor backbone C has ID number 48 and that one of the subscriber networks S under backbone B also has ID number 48. Without the Level field, router R could not distinguish between the backbone C and subscriber S. The essential purpose of the Level field, then, is to manage the RS Identifier number space so that different elements in the hierarchy have different RS Identifiers.

The following Level values are defined:

Value	Assignment
0	host ID
1	provider ID
2 - 5	intra-subscriber
6 - 9	intra-provider
10 - 15	reserved

Of the 16 Level values, two of them, host ID and provider ID, are globally recognized. A Level value of 1 is the provider level, and is initially the top level of the hierarchy. All providers are assigned an ID at Level 1[3]. Since the ID is 19 bits, this allows for approximately 500,000 providers world-wide.

Level 0 is reserved to mean Host ID. This level can only be used in RSIDs assigned to individual hosts. Setting aside this level for hosts is useful in forming node-level source routes and may be

[3]By provider, we mean a network that provides switching services. That is, from a given entry point to the provider network, the provider can deliver packets to multiple exit points. The actual packet service provided may be SPip, or may be some other protocol running under SPip, such as ATM, X.25, or IP. The size of the provider (in either geographic coverage or number of subscribers) is irrelevant to this discussion.

useful for host autoconfiguration. Note that hosts do not necessarily need a RSID at the host level, since the EID serves to identify hosts.

Levels 10 through 15 are reserved for future use. One potential use is for clustering providers under a higher-level hierarchy level. This could be necessary if the number of providers becomes too large for routing algorithms to handle, or for the 19-bit ID space to handle. As with all clustering, all providers within a given higher level cluster must be interconnected. Note that the levels values are not assigned in order of hierarchy level. That is, a higher level number does not imply a higher level of the hierarchy. This allows the reserved level values to be applied to any point in the hierarchy.

Levels 6 through 9 are reserved for use within a provider. While the use of these levels is left to the discretion of the provider, it is expected that one of them would be used to identify subscribers attached to the provider. The remaining levels may be used for additional clustering within the provider network. This is necessary for providers that have so many subscribers that they need to cluster subscribers internally (as described in Section 3.4.2).

Some discussion is required as to why a range of levels (levels 2 - 5) is globally recognized as being for intra-subscriber use, rather than just letting each provider decide for itself what levels, under its provider ID, are for intra-subscriber use.

The reason is to decouple above-subscriber address assignment from intra-subscriber address assignment. Since a subscriber can be connected to multiple providers, or can change providers, if each provider gave the subscriber a different range of intra-subscriber numbers, then intra-subscriber numbering would have to be modified when providers were changed.

A subscriber network is numbered within the range of levels from 2 to 5, independently of any provider numbering. Thus, a subscriber network might assign level 2 to subnets, and level 3 to areas. This leaves levels 4 and 5 for growth in the intra-subscriber hierarchy either above the area level or below the subnet level. If more than two levels were required for future growth, additional levels could be assigned from the reserved space, though they must be globally recognized as such.

Differentiating by level does not make all RS Identifiers distinguishable, only those that are 1) at the same level, and 2) have the same parent in the hierarchy. Thus, it is still possible for a router to have forwarding table entries for different destinations with the same RS Identifier. To handle this case, the router must use multiple RSEs when calculating the next hop (see Section 9.1.6).

8.1.6 SPip Hierarchical Unicast Address Assignment

An SPip address is a series of RS Identifiers, starting at the top level and continuing down to the host level or to the level above the host level (which can be identified by the EID). This series of RS Identifiers may be preceded by a route fragment of one or more RS Identifiers, all at the top level.

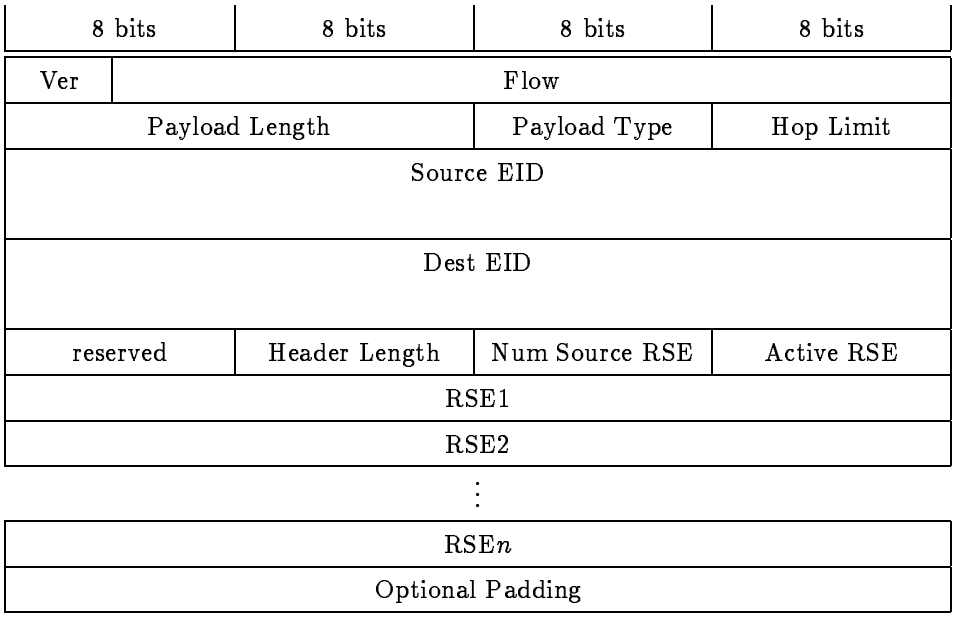
The purpose of the route fragment is to handle the case where the subscriber's immediate top-level provider is not advertised globally, for instance because it is only a local access provider. In this case, the (partial) address must be prefixed with the RS Identifier of a provider that is advertised globally, resulting in the complete address.

To assign SPip addresses, a top-level address assignment authority (such as the Internet Assigned Numbers Authority (IANA) with respect to IP [92] and CCITT with respect to E.163 [14]), assigns IDs at level 3 to providers. Providers assign numbers to elements below them in the hierarchy, etc., within the constraints of the ranges assigned to levels in Section 8.1.5.

Because subscribers can be connected to multiple providers, the subscriber network can have multiple address prefixes, and the subnets within the subscriber network can have multiple address sequences. A host entry in DNS consists of the addresses of the subnets to which the host is attached, plus the host's EID. We do not specify here whether the EIDs are listed separately from the addresses (that is, with different record types), or whether a single construct including both address and EID is used. The choice is one of complexity versus compactness of encoding, and does not affect what information is derived from DNS.

8.1.7 SPip Header Layout

The header layout for SPip is as follows:



The first two 32-bit words of this header are formatted the same as the SIPP header. The Flow field is pseudo-randomly set by the source of the packet such that the flow field and Source EID field taken together uniquely define the contents of the routing information in the packet—that is, the Dest EID field and route sequence.

The Flow field has a number of potential uses. It can be used by routers to cache the results of a

forwarding lookup (see Section 10.1.1). It can similarly be used by hosts to cache the results of a route sequence reversal calculation.

In addition, the Flow field can be used by routers for the purpose of managing flows—that is, the operations necessary to insure that real-time traffic requirements, such as delay and latency, are satisfied (see Section 2.2.6). This latter use of the Flow field is outside the scope of this thesis.

The Source and Dest EID are positioned identically to the SIPP Source and Dest Addresses.[4]

The remainder of the header is unique to SPip. The Header Length gives the length of the entire SPip header in 64-bit words. The remainder of the fields (Num Source RSE, Active RSE, and RSEs) are as explained above.

8.2 SIPP

Like SPip, the SIPP protocol can also use a source route mechanism for routing and addressing flexibility (though in SIPP it is not the only mechanism). SIPP, however, does so within the framework of a traditional (IP-like) packet header.

8.2.1 SIPP Routing and Addressing Fields

The (routing and addressing portion of the) SIPP header has the following format [30, 40]:

fixed		optional				
64 bits	64 bits	8 bits	64 bits	64 bits		64 bits
Source Addr	Dest Addr	Next Addr	Addr1	Addr2	···	Addr$n$

The sequence of addresses (Addr1, Addr2, etc.) combined with the Next Addr field is called the Source Route.

The Source and Dest Addrs are, at a minimum, identifiers for the source and destination of the packet[5]. Like SPip's EIDs, the Source and Dest Addrs are used by higher layer protocols to identify the endpoints of a connection. The Source and Dest Addrs are the only fields in the SIPP header that identify the source and destination.

The Source and Dest Addrs may additionally be locators in that they can be hierarchically structured addresses in the same way as IP or NSAP addresses. In particular, they are bit-wise left-to-right maskable addresses. By this, we mean that the fields in the address identifying the higher elements of the hierarchy are to the left of those identifying the lower elements, and that

[4] This much of the SPip header was taken directly from SIPP, though as of this writing SIPP does not have the same rules for setting the flow ID.

[5] This is true when the optional Source Route is not included. When the Source Route is included, the Dest Addr contains the active address, and the last address in the address sequence holds the identifier for the destination.

the field positions can fall on arbitrary bit boundaries.

A SIPP header may additionally contain a Source Route. Mechanistically, this Source Route is handled the same way as the IP loose source route. That is, when a packet is received by a SIPP node, it checks to see if the Dest Addr matches one of its own. If it does, it swaps the Dest Addr with the active addr in the Source Route, increments the Next Addr field, and forwards the packet to the new Dest Addr.

8.2.2 SIPP Packet Formation

It is easier to describe the formation of SIPP packets if we view the Source Addr, Dest Addr, and the sequence of addresses as a route sequence whereby:

- the 1st address of the route sequence is the Source Addr,
- the 2nd through $(i-1)$st addresses are those in the Source Route starting with the first address and ending with the address before the one indicated by the Next Addr field,
- the ith address is that in the Dest Addr field, and
- the $(i+1)$st through nth addresses are the remaining addresses in the Source Route.

Assume that a source host S is sending a packet to a destination host D. Each host has a sequence of one or more SIPP addresses that represents its locator.

The source host's address sequence takes up the initial addresses of the route sequence, in order of lowest order address first. The destination host's address sequence takes up the trailing addresses of the route sequence, in order of highest order address first. For instance, assume host S's address sequence is $\langle S_n,\ S_{n-1}, \ldots,\ S_1,\ S_0 \rangle$ and host D's address sequence is $\langle D_m,\ D_{m-1}, \ldots,\ D_1,\ D_0 \rangle$. A simple route sequence in the packet from host S to host D would look like:

$$\langle S_0,\ S_1, \ldots,\ S_n,\ D_m,\ D_{m-1}, \ldots,\ D_0 \rangle$$

If $m = 1$ and $n = 1$, then there is no Source Route in the packet.

As with SPip, a SIPP host S can learn the address sequence of a destination host D either through DNS or through the reception of a packet from the destination host. The address sequence for the host D can be extracted from a received packet as follows.

The received packet from host D contains a route sequence $\langle A_0,\ A_1, \ldots,\ A_{k-1},\ A_k \rangle$. For each of host S's address sequences, host S compares the elements of the address sequence against the tail of the received route sequence, looking for a best match. The best match is with the address sequence that has the largest i such that $S_0 = A_k,\ S_1 = A_{k-1}, \ldots,\ S_i = A_{k-i}$, where $\langle S_0,\ S_1,\ \ldots \rangle$ is the source address sequence.

Host S then reverses the remaining (unmatched) addresses in the incoming route sequence, to get

$$\langle A_{k-i-1},\ A_{k-i-2}, \ldots,\ A_1,\ A_0 \rangle.$$

As far as host S is concerned, this is a valid address sequence representing the destination, though actually it may contain the destination address sequence prepended with some additional addresses, representing, for example, a policy route.

This ability in SIPP for hosts to represent their addresses as an address sequence, and to reverse a received route sequence, gives SIPP much of its routing and addressing flexibility, including its ability to arbitrarily extend the address space. This is the second most important difference between SIPP and IP (the first being the fact that SIPP's native address is longer than IP's).

8.2.3 SIPP Forwarding Algorithm

For destination-based unicast forwarding, the SIPP forwarding algorithm is virtually identical to that of IP [87]. The differences are that 1) it operates on 64-bit addresses rather than 32-bit addresses, and 2) SIPP has no field equivalent to IP's ToS Field. In other words, SIPP has no path modifier.[6]

For source-tree multicast forwarding [31], there is a difference from IP's source-tree multicast forwarding due to the fact that SIPP can use its source-routing mechanism to effectively extend the length of SIPP addresses beyond 64 bits, similarly to how SPip creates variable length addressing.

When SIPP addresses are extended in this fashion, the source "address" (or, more accurately, address sequence), covers multiple fields—the Source Addr field and the initial positions of the source route. During forwarding, if upon examining the Dest Addr field the SIPP router determines that the packet is to forwarded according to source-tree multicast, it examines the address immediately preceding that indicated by the Next Addr field, if any, and the address in the Source Addr field otherwise.

If, upon examining an address in the source address sequence, the router finds that it must examine the next lower-order address in the sequence, the router examines the address in the Source Route immediately preceding the address it just examined, if any, and examines the address in the Source Addr field otherwise.

8.2.4 SIPP Address Definitions

There are three SIPP address types, the hierarchical unicast address, the multicast address, and the local-use address.

[6] The ToS Field of IP has been found to be of little practical use. The use of ToS Field routing in general, given the current state-of-the-art in routing and the current internet environment, is questionable. Thus, SIPP chooses not to implement it.

The multicast address is formatted as follows:

1	7 bits	4 bits	4 bits	48 bits
C	1111111	Flags	Scope	Group ID

The initial bit is the IP compatibility bit, or C-bit. The C-bit is used to indicate whether the system owning this address is an IP system. It is used for transitioning IP to SIPP, and is not of particular relevance to this thesis.

The subsequent 7 bits are set to all ones, and indicate that this is a multicast address. No other address type may have these 7 bits set to all ones.

Of the four Flag bits, the high order three are reserved and set to 0. The remaining bit indicates whether the multicast address is well-known (permanently assigned) or transient (not permanently assigned).

The Scope field serves the equivalent function of that in the SPip group EID.

The Group ID identifies the multicast group.

The local-use address is defined as follows:

4 bits	12 bits	48 bits
0110	Subnet ID	Node ID

The initial 4 bit pattern of 0110 identifies the address as being a local-use address. No other address type may have this 4-bit pattern.

The Subnet ID is used to identify a subnet within the network where the local-use address is assigned.

The Node ID identifies the node[7] within the subnet identified by the Subnet ID. The Node ID will usually be, but is not constrained to be, an IEEE-802 address.

The primary purpose of the local-use address is to allow auto-configuration. A SIPP host can assign the Node ID using an IEEE-802 address if it has one, or a link address otherwise, without coordination with other systems. It learns the Subnet ID and higher level addresses from router advertisements.

The remaining address space is used for hierarchical unicast addresses. Hierarchical unicast addresses encoded in a single SIPP address (that is, not an address sequence) initially have the following structure:

1	n bits	m bits	p bits	$63 - n - m - p$ bits
C	Provider ID	Subscriber ID	Subnet ID	Node ID

The assignment of SIPP hierarchical unicast addresses in an address sequence is discussed in Section 8.2.4.

[7]SIPP uses the term node to mean router or host.

SIPP addresses are provider-rooted (see Section 3.4). That is, the high-order part of the address is assigned to providers, which then assign portions of the address space to subscribers, etc. This is similar to assignment of IP addresses under the CIDR scheme [42]. The term "provider prefix" refers to the high-order part of the address up to and including the provider ID.

The subscriber ID distinguishes among multiple subscribers attached to the provider identified by the provider ID. The term "subscriber prefix" refers to the high-order part of the address up to and including the subscriber ID.

The subnet ID identifies a set of nodes on a single link within the subscriber network identified by the subscriber prefix. The node ID identifies a single node among the group of nodes identified by the subnet prefix.

A special case of hierarchical unicast address is the cluster address. A cluster address is an address with a provider, subscriber, or subnet prefix followed by all zeros. Cluster addresses are routed to the routers at the border of the network identified by the cluster address. These routers recognize the cluster address as identifying themselves for the purpose of advancing the source route.

SIPP Address Sequences

The SIPP unicast address format shown in the previous section also applies to the case where a SIPP address is conveyed as an address sequence rather than a single address. This is called an extended address. That is, the high-order field is the provider identifier, followed by the subscriber identifier, followed in turn by subnet identifier and host identifier. The difference, of course, is that with an extended address, these fields (and possibly additional fields, depending on how the internet grows) are spread over multiple 64-bit addresses.

There are two restrictions that apply when an extended address is used. First, at least the high-order and low-order address of the extended address should by itself be globally unique. The high-order address must be unique so that any router, no matter where it is in the hierarchy, can route a packet up to the top of the hierarchy without confusing it with local destinations. The low-order address must be unique because it uniquely identifies the host (or host group) among all hosts.

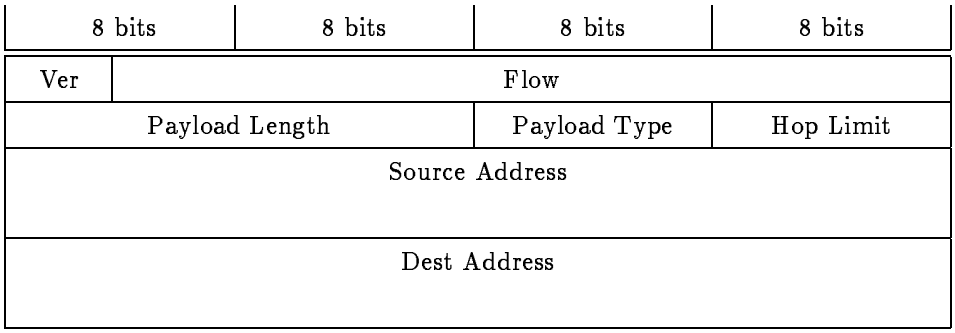
Actually, it seems highly unlikely that the SIPP address would need to be extended beyond two addresses (unless, perhaps, somebody wanted to encode an NSAP address in a SIPP address sequence). Strictly speaking, it is probably not necessary, in the case of greater-than-two address sequences, to make the middle addresses globally unique (just as SPip RSEs are not globally unique). Because of the unlikelihood of greater-than-two address sequences, we do not consider the pros and cons of unique middle addresses.

The second restriction is that a single hierarchy field within the extended address (for instance, the Subscriber ID field) must not cross a 64-bit boundary. This is because the SIPP forwarding

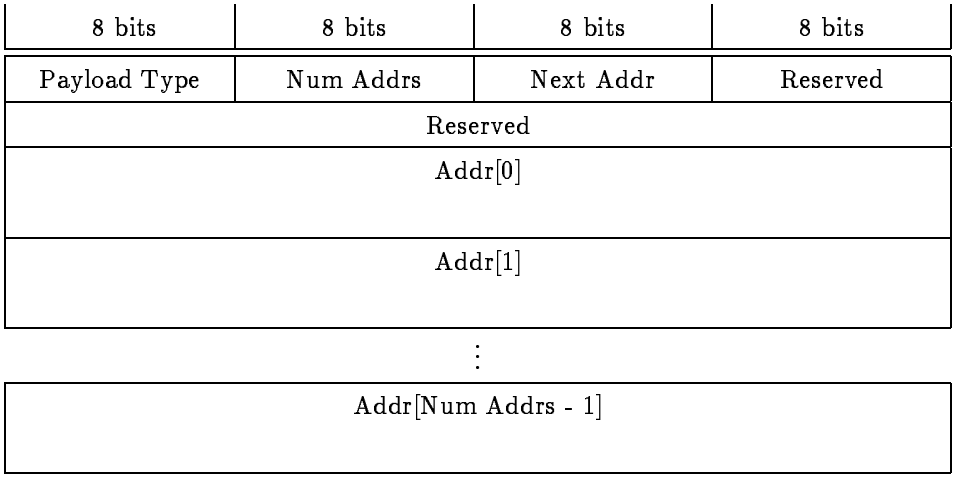
engine operates on only one address at a time.

8.2.5 SIPP Header Layout

The header layout for SIPP is as follows:



Note that this header is the same as the initial part of the SPip header. The Payload Type is used to identify the subsequent header, which can be a SIPP option or a different protocol. If the Payload Type indicates SIPP Source Route[8], then the subsequent Source Route header is formatted as follows:



8.3 CLNP

The (routing and addressing portion of the) CLNP header has the following format [55, 64]:

fixed				optional	
8 bits	variable	8 bits	variable	variable	variable
SA Length	Source Addr	DA Length	Dest Addr	QoS Fields	Source Route

Source and Dest Addrs serve the same role as those of SIPP—that is, they are locators. In CLNP, however, they are variable length (thus the SA Length and DA Length fields).

[8] Called a Routing Header in the SIPP specification

Mechanistically, the optional Source Route is handled similarly to that of SIPP, the main difference being that SIPP swaps the active address in and out of the Dest Addr field (as with IP), while CLNP does not. However, CLNP does not have the enhanced rules for handling the source route that SIPP has (reversing and using as an address sequence). Thus, CLNP is limited in its routing and addressing capabilities compared to SIPP (Chapter 9).

CLNP also has a Type-of-Service (ToS) Field which is intended to influence the route taken, and is thus a path modifier. (In CLNP, however, the field is called the Quality-of-Service (QoS) Field. To avoid confusion, we use the term QoS Field when discussing CLNP, and use the term ToS Field otherwise.) Some of the QoS Field encodings, however, are not specified in the base CLNP specification, and so can be adopted to different uses. (See Section 9.4.1 for more details on this use of QoS Field.) One of its encodings is specified in the base specification. It specifies preferences for sequencing, transit delay, cost, and error probability.

8.3.1 CLNP Address Assignment

The address defined for use with CLNP is the NSAP (Network Service Access Point) address [54]. Like the SIPP address, the NSAP address is bitwise left-to-right maskable. Whereas SIPP and SPip addresses are extensible by virtue of chaining multiple fixed-size addresses in a source-route mechanism, NSAP addresses are by themselves variable length. An NSAP address can be up to 20 bytes in length, in increments of 1 byte.

The high-order portion of the NSAP address defines the addressing authority for the remainder of the address, but does not contain any hierarchical topology information.[9] The authority thus defined determines how to further assign the address.

For instance, the first byte of the NSAP address is the Authority and Format Identifier (AFI). An AFI value of 47 indicates that the subsequent assignment authority is the ISO International Code Designator (ICD) [52]. The subsequent two bytes is the Initial Domain Identifier (IDI). An IDI value of 0005 indicates the US Government. The US Government, through the auspices of GOSIP (Government OSI Profile), has authority over the assignment of the remainder of the NSAP address, the Domain Specific Part (DSP). GOSIP defines the complete NSAP address as [110]:

1 byte	2	1	3	2	2	2	6	1
AFI	IDI	Domain Specific Part (DSP)						
47	0005	DFI	AA	Rsvd	RD	Area	ID	Sel

Where DFI = DSP Format Identifier, AA = Administrative Authority, Rsvd = Reserved, RD = Routing Domain Identifier, Area = Area Identifier, ID = System Identifier, and Sel = NSAP Selector.

[9] Strictly speaking, the high-order part of the SIPP and SPip addresses also define an addressing authority, but the primary purpose is to identify the top part of the topological hierarchy.

The DFI is essentially a type code. The AA is a top-level assignment by the GOSIP numbering authority. The RD is a routing domain within the AA. The Area is defined by the IS-IS routing algorithm [56] as the higher of two hierarchical levels within an RD. The ID identifies a host, and is defined by IS-IS as the lower of the two hierarchical levels. The ID can be but is not constrained to be an IEEE-802 address. The Sel indicates which higher layer protocol the packet is destined for.

There are also AFI values defined for group addresses [57]. Specifically, every AFI for an individual address has a corresponding AFI indicating a group address. Note that group NSAPs do not have a scope field.

8.3.2 CLNP Header Layout

The CLNP header has the following format [55]:

Field	Bytes
Network Layer Protocol Identifier	1
Length Indicator	2
Version/Protocol Id Extension	3
Lifetime	4
Type	5
Segment Length	6,7
Checksum	8,9
Destination Address Length Indicator	10
Destination Address	11 m - 1
Source Address Length Indicator	m
Source Address	m + 1 n - 1
Data Unit Identifier	n, n + 1
Segment Offset	n + 2, n + 3
Total Length	n + 4, n + 5

The options, if any, follow this.

Chapter 9

Routing and Addressing Capabilities of SPip, SIPP, and CLNP

This section describes how SPip, SIPP, and CLNP achieves (or does not achieve) each of the routing and addressing capabilities listed in Section 7.2. In so doing, it further specifies the operation of the protocols.

In all of the examples, the following conventions apply. Addresses (or RSEs) in a sequence of addresses are separated by a colon (':'). Hierarchical levels within an address are separated by a dot ('.'). If the address in a sequence is an SPip RSE, then the normal form RSE is notated as 'xY', where x is the Level (numeric) and Y is the ID (alphabetic). The reverse-path form is notated as 'rxY' (where r is not a variable, just 'r'). An address written as x.y.0... means an address with prefix x.y followed by one or more hierarchical levels of value 0. An address written as x.y.*... means as address with prefix x.y followed by one or more hierarchical levels whose values are wildcarded. That is, it is irrelevant (for instance, to a router) what the values are.

9.1 Big Enough Hierarchical Unicast Addressing

Hierarchical unicast addressing is of course the most important capability of an internetwork protocol. Without it, an internet could not grow to global proportions.

Note that we have explicitly stated that hierarchical unicast addresses must be "big enough". By this we mean big enough to handle all future internet growth. We state this explicitly because a number of internet protocols have in the past underestimated the required address size, not the least of these being IP. Thus we are here particularly sensitive to the requirement that the address

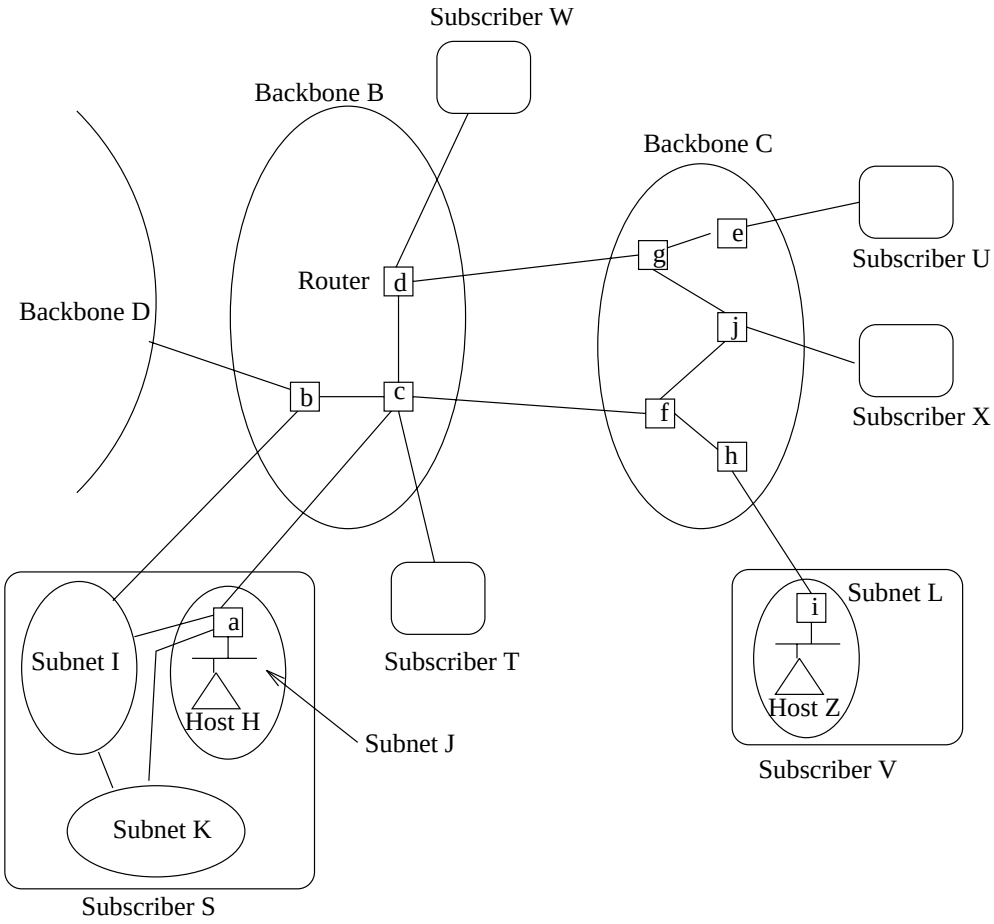


Figure 9.1: Example Topology

can handle all future internet growth.

The examples in this section are based on the topology of Figure 9.1. Figure 9.1 shows only the detail required for the examples. For instance, it is assumed that router c is connected to a router in subscriber network T, even though the router in subscriber network T is not shown.

This section describes two different hierarchical unicast addressing scenarios, one with classical forwarding information and one with additional "hole-punching" forwarding information (see Section 3.1.2).

Classical Forwarding Information

With classical forwarding information, each router knows how to forward to 1) its immediate parents in the addressing hierarchy (up), 2) all peers in the addressing hierarchy that share a parent (across), and 3) its immediate children in the addressing hierarchy (down).

Consider router c of Figure 9.1. It is in backbone B, and therefore at the top of the hierarchy. Since

it is at the top of the hierarchy, it has no parent in the hierarchy, and therefore no up forwarding information. At the top level of the hierarchy, router c maintains (across) forwarding information about all other backbones. Thus, router c has forwarding information for backbones C and D and all other backbones (not shown). Router c also maintains (down) forwarding information about the subscriber networks attached to backbone B, namely subscriber networks S, T, and W.

Additional (Non-Classical) Forwarding Information

Again consider router c of Figure 9.1. Assume it has all of the forwarding information described above for classical forwarding information, plus the following additional forwarding information.

1. Information about some of backbone C's subscribers. Note that backbone B is connected to backbone C in two places. Assume that when backbone B is sending packets to backbone C, it wishes to distinguish between these two connection points based on the destination subscriber network. Specifically, it wishes to send packets destined for subscriber V via the link between router c and backbone C, and to send packets destined for subscriber U via the link between router d and backbone C. Thus, the routers in backbone B have explicit forwarding information for subscribers V and U. Assume further that there are other subscribers in backbone C, such as subscriber X, for which the entry point does not matter, and so no forwarding information is known by routers in backbone B.

2. Information about some of subscriber S's subnets. Note that subscriber network S has two connection points with backbone B, one at router b and one at router c. Assume that backbone B wishes to send packets to subnet I via router b, and packets to subnet J via router c. Thus, routers in backbone B must have forwarding information for subnets I and J in subscriber network S. Assume further that there are other subnets in backbone C, such as subnet K, for which the entry point does not matter, and so no forwarding information is known by routers in backbone B.

9.1.1 CLNP with Classical Forwarding Information

Let's focus for the moment on the forwarding information in router c. From router f, it receives a routing advertisement for backbone C of C.*.... That is, the routing advertisement shows that any NSAP address with prefix C (C in the high-order field) followed by anything (wildcard) should be forwarded to router f. The 'C' of prefix C is some numerical value that is unique among all providers[1]. Likewise, router c receives the following advertisements from the following neighbors:

[1] The numerical value for C will also contain some address assignment authority information, due to the method by which NSAP addresses are assigned. For the purposes of routing, however, the assignment authority information concatenated with provider information amounts to a single field.

From	Advertisement
router f	C.*...
router b	D.*...
router d	B.W.*...
router a	B.S.*...
subscriber T	B.T.*...

Note that the above description of routing information received by router c reads like a distance-vector routing algorithm, whereby routers tell each other how far they are from various destinations, and routers pick the router that advertised the shortest distance to a destination as the next hop on the path to the destination. We are, however, not presupposing any specific routing algorithm. A link-state scheme would result in the same forwarding information being gathered and calculated by router c (see [36] for a description of the two routing algorithm styles).

From the routing information gathered, router c builds a corresponding forwarding table:

Forwarding Table for router c	
Destination	Next Hop
B.W.*...	router d
B.S.*...	router a
B.T.*...	subscriber T
C.*...	router f
D.*...	router b
*... (default)	Error

Conceptually, this forwarding table works as follows. When a packet arrives, the destination address is compared against each of the addresses in the forwarding table in sequence. The destination address is said to match the forwarding table address if all bits except the wildcarded bits match. If the two addresses match, then the forwarding information for that entry (outgoing interface and link address) is used to forward the packet. Note that the last entry matches all addresses. This is called the default entry, and is where packets are routed if no matches occur. Since router c is at the top of the hierarchy, it has no default route per se. Thus, if no match otherwise occurs, the default entry indicates an error. Note that this forwarding table is a simplified version of the more general forwarding table lookup algorithm described in Section 9.1.2.

Forwarding tables for routers f, h, and i are shown in Table 9.1. Router f's forwarding table is similar in content to that of router c (except of course that it is from router f's perspective, not router c's). Router i's forwarding table is very simple. It has local forwarding information (for host Z), but otherwise defaults packets to router h. That is, all packets that are not known to be local are simply routed to the backbone provider network.

Now consider a packet from host Z in subscriber network V to host H in subscriber network S. Assume that the address of host H is B.S.J.H. This is a four-level hierarchy consisting of backbone,

Table 9.1: Forwarding Tables for Classical Unicast CLNP Example

Forwarding Table for router f	
Destination	Next Hop
C.U.*...	router j
C.V.*...	router h
C.X.*...	router j
B.*...	router c
D.*...	router c
*... (default)	Error

Forwarding Table for router h	
Destination	Next Hop
C.U.*...	router f
C.V.*...	router i
C.X.*...	router f
B.*...	router f
D.*...	router f
*... (default)	Error

Forwarding Table for router i	
Destination	Next Hop
C.V.L.Z	host Z
*... (default)	router h

subscriber, subnet, and host. An actual NSAP address for host H might be:

2f00058000065e00000249005d08002001402E01,

or more readably:

AFI	IDI	DFI	AA	Rsvd	RD	Area	ID	Sel
2f	0005	80	00065e	0000	0249	005d	08002001402E	01

This NSAP address comes from the USA GOSIP definition (see Section 8.3.1). The AA value of 00065e identifies backbone B, the RD value of 0249 identifies subscriber S, the Area value of 005d identifies subnet J, and the ID value of 08002001402E identifies host H.

Assume that the address of host Z is C.V.L.Z (backbone C, subscriber V, subnet L, and host Z). Host Z forms a packet with source address C.V.L.Z and destination address B.S.J.H. Host Z sends this packet to router i (for instance, because router i is host Z's default router on the local LAN). Router i compares the destination address B.S.J.H against the entries in its forwarding table, and

finds no explicit match. It therefore matches on the default entry and sends the packet to router h. Router h matches B.S.J.H against entry B.*..., and forwards the packet to router f, which makes a similar match in its forwarding table, and forwards the packet to router c.

Up until this point, the routers have effectively only considered the high-order part of the address (backbone B) in routing the packet. Thus, the packet is being routed towards backbone B. Router c, however, is in backbone B, so it considers more of the packet to make its forwarding decision. Specifically, router c matches B.S.J.H against entry B.S.*..., and the packet is therefore now being routed to subscriber S (as identified by the prefix B.S.*...). When router a receives the packet, it matches in its forwarding table (not shown, but similar in form to router i's) against the full host address (B.S.J.H), and forwards the packet to host H.

When host H wishes to return a packet to host V, it simply reverses the positions of the source and destination address fields, and sends the packet to router a, which defaults it to router c, etc.

9.1.2 CLNP with Additional (Non-Classical) Forwarding Information

Now consider the case where routers in backbone B have the additional forwarding information described at the beginning of this section—that is, where routers in backbone B have explicit entries for subscriber networks U and V, and for subnets I and J, but not for subscriber X or subnet K. With this additional information, router c's forwarding table has the following entries:

Forwarding Table for router c	
Destination	Next Hop
B.S.I.*...	router b
B.S.J.*...	router a
B.W.*...	router d
B.S.*...	router a
B.T.*...	subscriber T
C.U.*...	router d
C.V.*...	router f
C.*...	router f
D.*...	router b
*... (default)	Error

This forwarding table shows the general form of forwarding with CLNP—that is, *best-match with default* forwarding. The best-match comes from the fact that multiple entries in the forwarding table can match a given address. For instance, address B.S.I.Y matches on the first (B.S.I.*...) and fourth (B.S.*...) entries in router c's forwarding table. However, the *best* match is the one that should be used, where the best match is the one that matches on the longest prefix. In this case, B.S.I.Y should match on the first entry rather than the fourth entry.

The address B.S.K.Y (a host in subnet K), on the other hand, would not match on the first (B.S.I.*...) or second (B.S.J.*...) entries, but rather would match on the fourth (B.S.*...) entry. Indeed, it is the fact that there are some subnets in subscriber S for which backbone B has no forwarding information that results in the fourth entry. If the routers in backbone B had forwarding information for all of the subnets in subscriber S, then the fourth entry would not be necessary, as all addresses in subscriber S would match on one of the (B.S.x.*...) entries.

The technique of maintaining forwarding information about destinations in other clusters (that is, more than just classical forwarding information) is called *hole-punching*. Maintaining only part of the total information available in another cluster is called *partial hole-punching*.

Conceptually, the way to achieve best-match is to compare the packet address against the entries in order of longest prefix first. Since a serial search is not fast, software [65] or hardware [78, 70] search techniques are applied to do a faster best-match lookup.

9.1.3 SIPP with Classical Forwarding Information

We are interested in two cases—one where global hierarchical unicast addresses are encoded in a single SIPP address, and one where they are encoded in an address sequence.

Single Address

The former case is identical to the CLNP case described above. That is, the forwarding tables and packet forwarding scenario given in Section 9.1.1 apply exactly to the single-address SIPP case.

The only difference is in the size of the address. CLNP addresses can be expanded to up to 20 bytes, whereas a SIPP address is 8 bytes. Thus, CLNP addresses can handle any imaginable network growth, whereas it is easy to imagine 8 bytes eventually being inadequate[2].

An example encoding of address B.S.J.H in SIPP is

```
0000034e30417058
```

where the first 8 bits are reserved as 00, the next 24 bits are the backbone identifier (00034e), the next 10 bits is the subscriber identifier (c1, but appearing in the above address shifted left two, so 304), the next 10 bits is the subnet identifier (17), and the last 12 bits is the host identifier (058).

Because the single-address SIPP case is identical to the CLNP case, there is no need to discuss it further here.

[2]This is not to say that 8 bytes definitely are inadequate. With careful management of the address space, which in itself exacts a certain cost, an 8 byte address space can handle a network whose size, for instance, well exceeds the global telephone network.

Extended Address

Since an extended address of greater than 128 bits is unlikely to ever be necessary, we assume here that the extended address is 128 bits (that is, an address sequence of two SIPP addresses). There are three ways that the four fields of the hierarchical address could be assigned within a SIPP extended address. They are; B:S.J.H, B.S:J.H, and B.S.J:H, where the colon indicates the boundary between the two SIPP addresses.

Of the three choices, only the latter two make much sense. The first—placing the address boundary between the backbone and subscriber identifiers, makes little sense because it does not evenly distribute the bits of the address well.

The last choice—placing the address boundary between the subnet and host IDs, makes some sense in that it allows the host ID to be completely location-independent, as is SPip's EID. The last choice may indeed be necessary in the case where 1) serverless auto-configuration of host IDs is required, and 2) the internet outgrows 48-bit host IDs. (This would require a new SIPP address definition, for instance, one with a 4-bit preamble followed by a 60bit host ID.)

The middle choice—placing the address boundary between the subscriber and subnet IDs, seems the most logical choice. This boundary represents a clear administrative boundary—that between the provider and the subscriber. Thus, the subscriber has control over the lower 64-bit address, and the provider has control over the upper 64-bit address. It also allows for serverless auto-configuration using the local-use SIPP address type in the lower 64-bits (see Section 9.5). Thus, we assume addresses of the form B.S:J.H in this section.

Given addresses of this form, and the classical forwarding information described above, router c would have the following forwarding table:

Forwarding Table for router c	
Destination	Next Hop
B.W	router d
B.S	router a
B.T	subscriber T
C.*...	router f
D.*...	router b
*... (default)	Error

Actually, this table is virtually identical for the analogous CLNP forwarding table (or single-address SIPP forwarding table). The only difference is that the first three entries indicate a full address rather than showing some wildcard bits. This is because the backbone and subscriber IDs together occupy an entire address.

The forwarding tables for routers f and h similarly reflect those of the CLNP example, and are not repeated here. The forwarding table for routers a or i, however, do have an important difference

from that for the CLNP case, and so the forwarding table for router a is given here:

Forwarding Table for router a	
Destination	Next Hop
J.H	host H
B.S	self
I.*...	subnet I
K.*...	subnet K
*... (default)	router h

The first thing to notice about this forwarding table is that the second entry identifies router a's own subscriber network (B.S), and that the next hop information for this entry is "self". This indicates that router a should advance the source route. For instance, consider the case where router a receives a packet destined for B.S:J.H, with the address B.S in the Dest Addr field, and the address J.H as the Next Addr in the source route. Router a looks up B.S in the forwarding table, and matches on the second entry. This indicates self, so router a advances the source route (puts J.H in the Dest Addr field, puts B.S where J.H was in the source route, and increments the Next Addr field), and then looks up J.H. This matches on the first entry, and the packet is forwarded to host H.

The second thing to notice about this forwarding table is that the third and fourth entries have the subnet ID as the most significant information in the address, and that the rest of the address is wildcarded. Since the subnet ID field alone is not globally unique, another router in another subscriber network could have identical Destination information in its forwarding table entries, but which point to different destinations.

For instance, assume that the local-use SIPP address is used for the lower address of the extended address. Assume further that the 12-bit Subnet ID for subnet I is value 02e. This results in a forwarding table Destination entry in router a of 602e*..., where 6 is the 4-bit preamble indicating that the address is a local-use address, 02e is the subnet ID, and the rest of the address is wildcarded. Assume, however, that subnet L in subscriber V also uses the local-use format, and also has a subnet ID of 02e. The forwarding table Destination entry for some router in subscriber network V would be the same.

This does not result in any particular problem as long as the forwarding contexts are kept distinct. That is, as long as router a does not need to maintain any subnet-level forwarding information about subnets in another subscriber network, the forwarding table entry is unambiguous.

9.1.4 SIPP with Additional (Non-Classical) Forwarding Information

When using single addresses, SIPP works identically to the CLNP case (Section 9.1.2).

Depending on the situation, SIPP with extended addresses can have some problems with hole-

punching. This occurs when hole-punching is across the 64-bit address boundary

For instance, consider the forwarding table of router c if we naively construct it based on that of the CLNP example, but taking extended addresses into account.

Forwarding Table for router c	
Destination	Next Hop
I.*...	router b
J.*...	router a
B.W	router d
B.S	router a
B.T	subscriber T
C.U	router d
C.V	router f
C.*...	router f
D.*...	router b
*... (default)	Error

Strictly speaking, the forwarding table of router c works correctly.[3] If, however, router c had forwarding table entries for subnets in another subscriber network, then the Destination fields for those entries could be identical with those for the subnets in subscriber network S (the first two entries), and routing would fail. (In practice, there would be subnets within backbone B, for use by the operators of backbone B. Thus, these subnet IDs would appear in the forwarding tables and would be ambiguous with respect to subscriber S's subnet IDs.)

Thus, in general, the above forwarding table could not be generated. The first two entries should not appear.

As mentioned in the previous section, this ambiguity results from the facts that 1) the subnet IDs appear at the top of the (lower) address, and 2) the subnet IDs are not globally unique.

Note that the problem does not come up with the other hole-punching of router c's forwarding table—namely, where router c is maintaining entries for subscribers of backbone C. These addresses have the backbone ID at the top (entries 3 through 7 in router c's forwarding table). Since backbone IDs are globally unique, the subscribers in different backbones are distinguishable.

[3]Note that the entries are in the same order as that of the corresponding CLNP forwarding table. According to the best-match algorithm of comparing against longest-prefix first, the entry order shown here is incorrect—the I.*... and J.*... entries should come after the C.V entry. However, if the numbering spaces for the lower and upper addresses of the extended address are separate, then no address will match on both a lower and an upper entry, so the ordering of lower versus upper addresses is irrelevant.

9.1.5 SPip with Classical Forwarding Information

Like CLNP and SIPP, SPip forwarding is best-match with default. Whereas we describe CLNP and SIPP forwarding in terms of a single forwarding table with best-match semantics, we describe SPip forwarding as a tree of forwarding tables, each with *single-match with default* semantics (the default occurring if there is otherwise no match). This description style follows naturally from the SPip header format, which presents addresses as a series of flat identifiers rather than as a single (or small number of) hierarchical addresses.

In the following examples, backbone IDs are assigned at level 1, subscriber IDs are assigned at level 9, and subnet IDs are assigned at level 3. The host IDs are the EIDs, and so do not have a level per se. (The level is unnecessary because EIDs are in a separate number space, with separate forwarding tables, and so are never confused with RSE IDs.)

The forwarding tables for classical routing information are straight forward to derive, and are shown in Table 9.2.

Table 9.2: Forwarding Tables for Classical Unicast SPip Example

Root RSETable for router c		
Destination	Next Hop	T-tag
9W	router d	√
9S	router a	√
9T	subscriber T	√
1C	router f	√
1D	router b	√
1B	Root RSETable	√
default	Error	

Root RSETable for router f		
Destination	Next Hop	T-tag
9U	router j	√
9V	router h	√
9X	router j	√
1B	router c	√
1D	router c	√
1C	Root RSETable	√
default	Error	

Root RSETable for router h		
Destination	Next Hop	T-tag
9U	router f	√
9V	subscriber V	√
9X	router f	√
1B	router f	√
1D	router f	√
1C	Root RSETable	√
default	Error	

Root RSETable for router i		
Destination	Next Hop	T-tag
9V	Root RSETable	√
3L	Main EIDTable	√
default	router h	√

Main EIDTable for router i		
Destination	Next Hop	T-tag
Z	host Z	√
default	ARP	√

Root RSETable for router a		
Destination	Next Hop	T-tag
9S	Root RSETable	√
3I	subnet I	√
3K	subnet K	√
3J	Main EIDTable	√
default	router c	√

Main EIDTable for router a		
Destination	Next Hop	T-tag
H	host H	√
default	ARP	√

There are several things to note about the forwarding tables in Table 9.2.

First, the title of each router's forwarding table is changed from just "Forwarding Table" to "Root RSETable" or "Main EIDTable". Since SPip is modeled as a tree of forwarding tables, the different forwarding tables require different labels. If an RSE is active (as opposed to the EID), the router always accesses the Root RSETable first. Note also that routers a and i have an EIDTable as well as a Root RSETable. This is because both routers a and i have host-level forwarding information.

Second, the Destination entries in the forwarding tables have no wildcard bits. This reflects the fact that the individual forwarding tables are single-match rather than best-match.

Third, there is an additional column, the T-tag column. T-tag stands for transmitRSE tag (see Section 8.1.2. It has an '$\surd$' if the RSE should be tagged for setting upon transmit, and is blank otherwise.

Fourth, some entries indicate a subnet or a subscriber network in the Next Hop field (for instance, the third entry of router c's Root RSETable). By this we mean to indicate a router within the subnet or subscriber network. Since Figure 9.1 does not actually show those routers, we indicate just subnet or subscriber network instead.

Finally, neither the Last RSE flag nor the Path Modifier appears in these tables. The Path Modifier does not appear because all of the Path Modifier values for the unicast examples are assumed to be the same—they indicate normal form RSEs and are therefore 0.

We are not showing the Last RSE flag just to keep things simple. The entries that point to other RSETables would do so only when the Last RSE flag is not set. If the effect of the Last RSE flag were shown in the tables, then each of these entries might have a companion entry pointing to the Main EIDTable (or a default entry based on the Last RSE flag not being set pointing to the Main EIDTable). The entries that point to EIDTables would only do so if the Last RSE flag is set. If the effect of the Last RSE flag were shown in the tables, then each of these entries might have a companion entry pointing to an RSETable. Examples showing the use of the Last RSE flag are given in section 9.8.3.

Consider a packet sent from host Z to host H. Host Z formats the packet as follows:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
Z	H	4	3L	9V	1C	**1B**	9S	3J

The (globally unique) host IDs are placed in the Source and Dest EID fields (SEID and DEID). The subnet, subscriber, and backbone IDs of the source are placed in the first three RSEs of the route sequence. The backbone, subscriber, and subnet IDs of the destination follow those. The active RSE is set to the backbone of the destination (the 4th RSE, 1B). It is shown in bold for clarity.

This packet is transmitted to router i, which indexes its Root RSETable with the active RSE, 1B.

This matches nothing in the Root RSETable, so the default entry is chosen, which indicates that the packet should be forwarded to router h. This entry is tagged (the T-tag is set) to show that the Active RSE should be set upon transmission to point to the RSE that was used to access the table. Because this RSE is the same as the one active when the packet was received, the packet is transmitted unchanged (except perhaps for the hop count, which we do not consider here) to router h.

Router h accesses its Root RSETable with value 1B, matching on the 4th entry. This entry indicates that the packet should be forwarded to router f. As with router i, only one RSE was examined during forwarding, so the Active RSE is again not changed.

Router f accesses its Root RSETable, with similar results, and forwards the packet to router c. Thus, router c receives the packet as it was transmitted by host Z. In other words, the packet has been so far routed "to backbone B".

When router c accesses its Root RSETable with 1B, it gets a match on the 6th entry. The Next Hop field for this entry says "Root RSETable". This indicates that the router should advance to the next RSE, and index the Root RSETable using it. Thus, router c indexes its Root RSETable with value 9S. This also produces an exact match—the 2nd entry.

This entry indicates that the packet should be forwarded to router a. Its T-tag indicates that the Active RSE should be set to point to the RSE used to access this entry—the 5th RSE—9S. Thus, router c transmits the following packet to router a:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
Z	H	5	3L	9V	1C	1B	**9S**	3J

In this packet, the Active RSE field has been changed from 4 to 5, so that now the packet is being forwarded to subscriber S rather than to backbone B. In other words, router c had recognized in its first access to the Root RSETable that the intermediate destination, backbone B, had been reached, and so now forwarding must take place at the next level of granularity—the subscriber level.

When router a receives the above packet, it accesses its Root RSETable with value 9S. This matches exactly with the first entry, which instructs the router to access the Root RSETable again with the subsequent RSE, 3J. This access matches the 4th entry, which instructs router a to access the Main EIDTable.

This final access, made using the Dest EID field, produces an exact match on host H's EID, and the packet is forwarded to host H. Since the entry in the Main EIDTable has the T-tag set, the Active RSE field is set to 0 upon transmission, which indicates that forwarding should now take place on Dest EID rather than on an RSE.

Note that the default entry for the EIDTable says "ARP". This is an indication that the router should ARP for the host LAN address [84]. If an explicit host entry exists in the EIDTable,

it indicates that either 1) host H was previously ARP'd for and the answer was cached in the EIDTable, or 2) host H advertised itself with an ES-IS style host hello packet, and router a stored the discovered information in its EIDTable. Thus, the EIDTable can substitute for the traditional IP ARP table.

To return a packet to host Z, host H follows the route sequence reversal rules given in Section 8.1.3. That is, host H takes the RSEs indicated by the Num Source RSE field from the beginning of the route sequence (3L:9V:1C), reverses it (1C:9V:3L), and prepends its own address sequence (3J:9S:1B:1C:9V:3L). Host H also swaps Source and Dest EIDs, and sets the Active RSE field to point to the RSE after its own source address sequence. Thus, the packet transmitted by host H is:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
H	Z	4	3J	9S	1B	**1C**	9V	3L

9.1.6 SPip with Additional (Non-Classical) Forwarding Information

With the additional forwarding information described in Section 9.1 router c has the forwarding tables shown in Table 9.3.

Because SPip is described in terms of nested forwarding tables, the forwarding tables of Table 9.3 are unwieldy to read. Figure 9.2 gives the same information pictorially. The boxes of Figure 9.2 indicate forwarding tables. The arrows indicate forwarding table entries. The arrows are shown to go through what is the Destination column of the forwarding table, and terminate either at another forwarding table, or at the what is the Next Hop column of the forwarding table. Entries with active T-tags are shown with a dot at the beginning of the arrow. Subsequent complicated examples of SPip forwarding present the tables only pictorially.

To see how the forwarding tables for the hole-punching case works, let's pick up the previous example at the point where the packet from host Z to host H arrives at router c from router f. The packet at that point contains the following header:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
Z	H	4	3L	9V	1C	**1B**	9S	3J

Router c accesses its Root RSETable with RSE 1B (the 6th entry in Table 9.3, or the arrow going off to the right from the Root RSETable of Figure 9.2. This indicates that router c access the Root RSETable again, with RSE 9S, the subsequent RSE.

The entry for 9S instructs the forwarding algorithm to access RSETable 1 with the subsequent RSE, 3J. This access matches on the 3J entry, indicating that the packet is to be forwarded to router a. Note that the entry in RSETable 1 for 3J does not have the T-tag set. It was set, however, for the previous access (the one on the Root RSETable with 9S). Thus, the Active RSE field is modified to point to the 5th RSE, 9S. More to the point, the Active RSE field is set to

Table 9.3: Router c's Forwarding Tables for Non-classical Unicast SPip Example

Root RSETable for router c		
Destination	Next Hop	T-tag
9W	router d	√
9S	RSETable 1	√
9T	subscriber T	√
1C	RSETable 2	√
1D	router b	√
1B	Root RSETable	√
default	Error	

RSETable 1 for router c		
Destination	Next Hop	T-tag
3I	router b	
3J	router a	
default	router a	

RSETable 2 for router c		
Destination	Next Hop	T-tag
9U	router d	
9V	router f	
default	router f	

point to 9S because the subscriber network identifier level (level 9) is the level at which the packet is now being unambiguously forwarded.

To see this more clearly, consider a packet from host H to a host Y in subscriber U (with address 1C:9U:3M) (not shown in Figure 9.1). The packet arriving at router c from router a is:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
H	Y	4	3J	9S	1B	**1C**	9U	3M

Note that this packet is unchanged from what was transmitted by host H, because the target destination is still (the top-level) backbone C.

Router c accesses its Root RSETable with 1C, retrieving an instruction to access RSETable 2. This is accessed with the subsequent RSE, 9U. That access produces the next hop, which is router d. The Active RSE field stays at the 4th RSE, 1C, because this was the last tagged entry.

Note what would have happened if the Active RSE in the packet transmitted from router c had

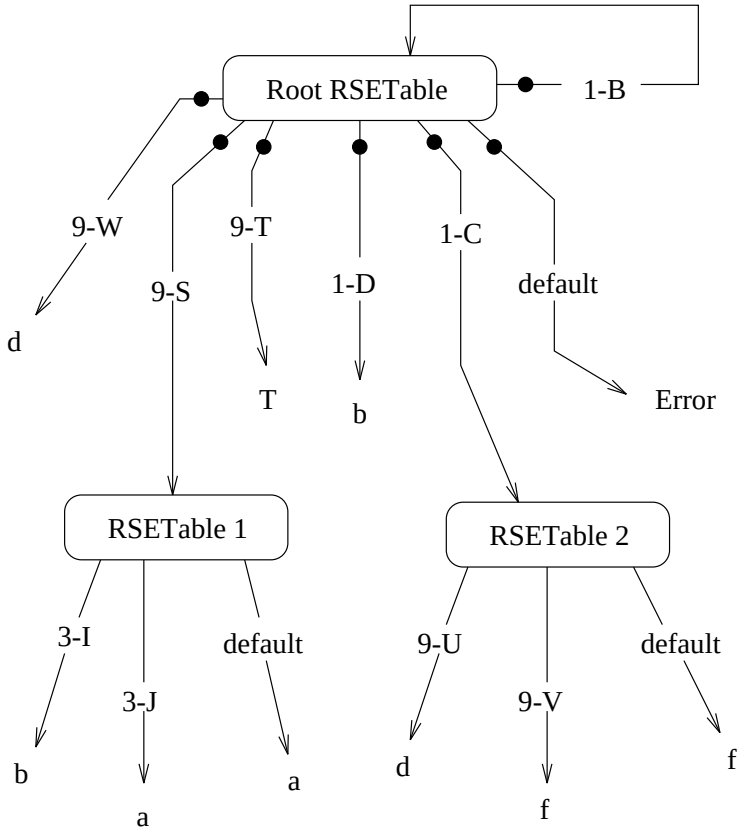


Figure 9.2: Router c's Forwarding Tables for Non-classical Unicast SPip Example

been 9U—the RSE that ultimately gave router c its forwarding information. Router d would in that case have received the following packet:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
H	Y	5	3J	9S	1B	1C	**9U**	3M

It would have accessed its Root RSETable with 9U. Depending on the actual numerical value for U, however, this packet may have matched exactly with one of the subscriber IDs for backbone B's own network. For instance, the numerical values for both W and U might be, say, 17. The packet in this case would be incorrectly routed to subscriber W. By leaving the Active RSE to be 1C, router d is able to know the appropriate context for the subscriber ID—that is, a subscriber attached to backbone C.

The technique of looking ahead in the route sequence without advancing the Active RSE pointer is different from SIPP (or IP or CLNP) route sequence handling, where the active address is always the last one examined. This looking forward is called *peek-ahead*. Peek-ahead is necessary in SPip because each level of the hierarchy is one RSE, and multiple hierarchy levels must be examined to do hole-punching. It is not necessary in CLNP, nor to a large extent SIPP, because the semantics

of best-match lookup on a hierarchical address allow hole-punching.

Note that all of the entries in router c's Root RSETable (Table 9.3) have the T-tag set, and that none of the entries in router c's other RSETables have the T-tag set. Not also that by-and-large the Root RSETable holds classical forwarding information, and that the other RSETables hold additional hole-punching forwarding information. For instance, router c's Root RSETable has Next Hop information for 9W, 9T, and 1D, all classical forwarding information. Router c's additional RSETables contain Next Hop information for 3I, 3J, 9U, and 9V, all hole-punching forwarding information.

The Root RSETable is the only RSETable that is accessed without accumulated context. Accesses to other RSETables have some context (for instance, the context for router c's RSETable 1 is subscriber S, and for RSETable 2 is backbone C). Classical forwarding is unambiguous and does not need context, whereas hole-punching forwarding information can be ambiguous and needs context. Thus, classical forwarding information by-and-large goes in the Root RSETable, and hole-punching information does not.

There are two pieces of classical forwarding information, however, where the Next Hop information does not appear in router c's Root RSETable. They are for 1C and 9S. This information shows up in the default entries of the two extra RSETables. The forwarding information for 9S appears in the default entry for RSETable 1, and the forwarding information for 1C appears in the default entry for RSETable 2. Note that the entries in the Root RSETable for 9S and 1C point to RSETable 1 and RSETable 2 respectively.

To see why this information appears in the default entries, consider the case of a packet from Host H to a host Q (not shown in Figure 9.1) in subscriber X arriving at router c:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
H	Q	4	3J	9S	1B	**1C**	9X	3N

RSE 1C points router c to RSETable 2, but the next RSE, 9X, does not have a match in RSETable 2. This is because routers in backbone B do not care how packets are routed to subscriber X—both entries points to backbone C are equally preferable. The default causes router c to fall back on the (classical) forwarding information for 1C. In essence, router c dipped into RSETable 2 because there might be more detailed forwarding information, but discovered, by not matching, that there was not after all.

Note that in certain cases, it is possible to not include the complete address sequence in a packet. This is the case where two hosts are in close proximity to each other, for instance in the same subnet or subscriber network.

For instance, consider a packet from host H to a host in subnet I (say, host F). Since they are in the same subscriber network, there is no need to include RSEs encoding subscriber and above information. Thus, the packet from host H to host F can be formatted as:

SEID	DEID	Active RSE	RSE1	RSE2
H	F	2	3J	**3I**

When router a receives this packet, it indexes its Root RSETable with 3I, matching on the second entry, and forwards the packet to subnet I.

The problem with this optimization is that it does not work in all cases. For instance, if router a had hole-punching forwarding information pertaining to the interior of another subscriber network, then router a may not be able to distinguish between subnet level RSEs in its own subscriber network and another subscriber network. However, it is difficult for host H to know if router a, or some other router, has hole-punching forwarding information, and thus needs more addressing information.

Therefore, this kind of optimization is in general not a viable option.

9.1.7 Discussion

All three protocols can do hierarchical unicast routing. All three protocols have effectively unlimited address size, CLNP by virtue of its large single address, and SIPP and SPip by virtue of being able to string fixed-sized addresses together to create a large address space.

SPip has a slight limitation in that it can only devote 19 bits (the size of the RSE ID field) to identification at each hierarchy level. This allows for approximately 500,000 hierarchical clusters within a higher-level cluster. If a cluster has more than this many sub-clusters, then it would have to introduce another layer of hierarchy to handle the additional clusters.

All three protocols can handle both classical and hole-punching forwarding information. The one exception is with SIPP, when it is using extended addressing and when the hole-punching is across an address boundary. This limitation results from the fact that SIPP does not do peek-ahead when it processes the source route.

The limitation of SIPP not being able to hole-punch across an address boundary is not a serious one. First, hole-punching is not so common. Second, it is possible to place the address boundary where peek-ahead is most rare, for instance at the boundary between provider and subscriber.

Finally, if it is absolutely necessary for, say, a provider to be able to choose among multiple entry points into a subscriber's network based on the location of the destination, there are at least two ways to make it happen. First, the subnet number could be replicated in the low-order part of the higher address. The backbone routers could forward on the subnet number in the higher address, and the subscriber routers could forward on the subnet number in the lower part of the address.

Second, the subscriber could be given multiple subscriber numbers, one for each entry point.

Both methods have the effect of putting intra-subscriber information in the higher address, and

both methods require coordination between subscriber and provider. The former method is easier for the subscriber, and the latter easier for the provider.

9.2 Multicast

This section covers all forms of multicast listed in Table 7.1—broadcast[4] and multicast, shared-tree and source-tree, scoped, well-known addresses, and two-phase.

In all the descriptions, we assume the following action/response. That is, that the initiating host transmits a multicast, and that a receiving host be able to transmit a unicast packet back to the transmitting host.

The descriptions for CLNP and SIPP are only rough outlines. Detailed descriptions of how to handle multicast for traditional packet formats can be found in the literature [27, 28, 29, 31].

9.2.1 CLNP

The mechanism for formatting a multicast packet in CLNP is similar to that of unicast. The host learns of a group address through the normal means (directory service or an IGMP-like protocol [28] yet to be defined for CLNP). It puts the group address in the destination address field, puts its own unicast address in the source address field, and transmits the packet.

There is no scope field in the NSAP group address (or any other place in the CLNP header). There are two ways to achieve scoping in CLNP. One is to do what IP currently does, which is to use the hop-count field (see Section 6.1.1). The other way is to assign separate group addresses for each scoping.

The technique for forwarding a multicast packet in CLNP has not yet been defined, so we assume here that it is done similarly to how it is described in Section 6.1, which is basically how Deering specifies it in his PhD thesis [29] (some of which exists in IP today, and some of which does not).

In other words, the router examines the Dest Addr, determines that it is multicast, and either determines the tree links from this and forwards the packet(s), or determines that it must also examine the Source Addr, and uses the combined information to determine the tree links. Thus, CLNP can (or will be able to) handle all four combinations of broadcast/multicast and source-tree/shared-tree.

If a host receiving the packet wishes to return a packet to the source host, it places the Source Addr of the received packet in the Dest Addr field, places its own unicast hierarchical address in the Source Addr field, and transmits the packet.

[4]Since broadcast is a degenerate form of multicast, we do not consider it explicitly, but instead assume it works if multicast works.

Well-Known Multicast Group Addressing

Well-known group addresses can be assigned from the already defined NSAP group address space. For well-known group addressing to be practical, however, an effective scoping mechanism is required (Section 6.1.2). Hop count is an effective scoping mechanism where the scope is the local LAN (hop count of 1). Where the desired scope is larger, hop-count does not provide enough control over the recipients. Because CLNP does not have a scoping mechanism other than hop-count, CLNP's capability to do well-known multicast is limited.

Two-Phase Multicast

The mechanism where by two-phase multicast could *in theory* be made to work is the CLNP loose source route (LSR) option (called partial source route in CLNP). The reason we say *in theory* is because there is a bug with CLNP's LSR mechanism that makes it effectively unusable. The bug derives from the fact that the LSR mechanism is such that the active address of the source route is in the LSR option rather than in the destination address field. In addition, the LSR option is a "type 3" function [55], meaning that it does not need to be supported in routers, and that a router not supporting it simply ignores it—the router does not discard the packet.

As a result, if there are some routers that do not support the LSR option (non-LSR routers) on the path between the source and the active address, those packets will route on the address in the destination address field rather than on the active address. With unicast, this can result in a forwarding loop whereby the packet is routed by a non-LSR router to an LSR router, which then routes it back to the non-LSR router that already handled it.

With two-phase multicast, the consequences of this bug can be really disastrous. The packet would go as unicast initially (routed on the active address of the source route), then reach a non-LSR router, which multicasts it. Some of the multicast replications reach LSR routers which forward it unicast to more non-LSR routers, which multicast it further. If there is a loop, then the packet can replicated each time it traverses the loop. Eventually the hop-count would cause the replicas to be discarded, but only after generating potentially an enormous number of packets.

Even if this problem were fixed (for instance, by making implementation of the LSR option mandatory), use of the LSR for two-phase multicast is generally difficult because, as with IP, the notion of handling a route sequence is foreign to CLNP. Thus, none of the hooks that make its use convenient, such as route sequences in directory service or in the API (Application Programming Interface) are in place.

9.2.2 SIPP

Before being able to format a SIPP multicast packet, a SIPP host must first determine the group address. In particular, the host must determine the setting of the scope field. While the SIPP specification defines a scope field in its group address, there is nothing yet in the specification that indicates how the scope field should be set.

There are basically two approaches. In one, the scope can be tightly coupled with the group ID, resulting in 64-bit identifiers that are treated individually. That is, group addresses with the same group ID but different scope are learned individually, for instance via IGMP, and the host chooses among them depending on the desired scope of the multicast. Alternatively, the scope can be uncoupled from the group ID, so that the host considers the scope field to be separately settable. In this case, the host would learn a single group ID, and then compose the full group address by setting the scope field according to the desired scope.

We do not discuss the pros and cons of the two approaches here. We assume in what follows that a SIPP host is able to obtain the appropriate group address.

SIPP multicast packets are formatted identically to SIPP unicast packets. In particular, the destination "information" can be encoded as an address sequence. This address sequence appears anywhere a single group address would otherwise appear—in IGMP, SD, or DNS.

The low-order address (and identifier) of the address sequence is the group address, and the higher order addresses of the sequence, if present (for instance for two-phase) are unicast addresses. As with unicast, the source address sequence of the SIPP host is placed at the beginning of the route sequence, and the destination address sequence is placed at the end.

SIPP multicast packets are forwarded by routers similarly as described for CLNP above, taking into account parsing the extended source address as described in Section 8.2.3. The only difference is that the SIPP router must interpret the scope field. Thus, once a router has determined which links a packet should potentially be forwarded over, it must eliminate zero or more because of scoping.

It has not yet been defined whether scoping in SIPP is source-independent or source-dependent (see Section 6.1.1). If the former case, then the router simply does not transmit the packet over links whose defined scope values match that in the packet. In the latter case, the router must consider both the source address sequence and scope value to prune the outgoing link set.

If a host receiving the packet wishes to return a packet to the source host, it executes the reversing rules given in Section 8.2.2.

Well-Known Multicast

SIPP has reserved a portion of its group address space for well-known group addresses, and has scoping. Thus, SIPP can do well-known multicast groups.

Two-Phase Multicast

SIPP can do two-phase multicast, but only in the case where the multicast phase is shared-tree.

With two-phase multicast, the destination address is an address sequence whereby the low-level address is the group address, and the remaining addresses is the unicast address sequence of the system that will originate the multicast phase. If the unicast address sequence is a cluster address, then the first cluster border router that that packet reaches will originate the multicast phase (by advancing the route sequence to the group address).

SIPP can do CBT-style two-phase multicast, where the multicast phase of the packet starts when the packet reaches any router on the core tree. However, care must be taken in the assignment of the core address (the unicast address of the core). That is, the core address must not be the one used by the core for its normal unicast communications. This is because of the way the route sequence is advanced in SIPP. That is, a router only advances the route sequence if it believes that it is the destination for the active address. If the normal unicast address of the core were used, then every router on the CBT tree would install that address as its own, an would therefore receive unicast packets addressed to the core.

9.2.3 SPip

With SPip, every EID, including group EIDs, has associated with it zero or more address sequences. The address sequence is required only for two-phase multicast. Otherwise, there is no address sequence.

Before a host can format a multicast packet, it must set the scope field of the group EID. If the EID has a scope field of 0, then the host must set the scope field to the desired value (see Section 8.1.4). If, on the other hand, the host has multiple EIDs with identical group IDs but different non-zero scope fields, then it chooses among them according to the desired scope.

To form a multicast packet, an SPip host initially does the same as it does for unicast. That is, the group EID is placed in the Dest EID field, the EID of the transmitting host is placed in the Source EID field, and the address sequence of the transmitting host is placed at the front of the route sequence.

In addition to this, the host does one of the following:

1. If the group EID is associated with one or more address sequences, then one of the address

sequences is placed after the source address sequence exactly as with unicast.

2. Otherwise, the host places its own address sequence, in reverse-path form, into the routing sequence after the source address sequence. The reverse-path form address sequence is placed in order of highest-level RSE first (same order as the destination address sequence would normally be placed). The Active RSE field is set to point at the first reverse-path form RSE, and the Num Source RSE field is set to point to the last normal form RSE.

It what follows, we assume single-phase multicast. That is, we assume that there is no address sequence associated with the group EID—case 2 above.

Consider the case of host H in Figure 9.1 transmitting a multicast packet with group address G1.

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
H	G1	4	3J	9S	1B	**r1B**	r9S	r3J

Note that host H's normal form address sequence occupies RSEs 1 through 3, and that host H's reverse-path form address sequence occupies RSEs 4 through 6.

As shown below, the purpose of the reverse-path form RSEs is for source-tree multicast. With shared-tree multicast, they are not necessary. However, the host has no way of knowing (or, more accurately, should not be required to know) if the packet will be transmitted on a source-tree or a shared-tree.

The router, on the other hand, of course knows whether the packet is being forwarded as source-tree or shared-tree, and builds its forwarding tables accordingly.

Shared-Tree Multicast

If the router is operating exclusively in shared-tree mode, then it does not care about the reverse-path form RSEs in the packet. It must look beyond them to the multicast EID.

Thus, a router with shared-tree multicast adds one entry to its Root RSETable. That entry is the *r-default* entry—that is, it is retrieved for any revere-path form RSE.

For instance, the Root RSETable for router a for the classical routing information case would be the same as that given in Table 9.2, but with the addition of the r-default entry. This is shown in Figure 9.3.

The multicast entries of Figure 9.3 are shown with dashed lines. Note that what was previously the single default entry has here been changed to n-default (meaning normal form default). This is the entry retrieved if the RSE is normal form, but does not match any of the explicit normal form entries. The r-default entry is retrieved for all reverse-path form RSEs.

The r-default entry indicates that the Main EIDTable should be accessed. The Main EIDTable, in addition to the unicast entries it had from the unicast example (Table 9.2), has a number of

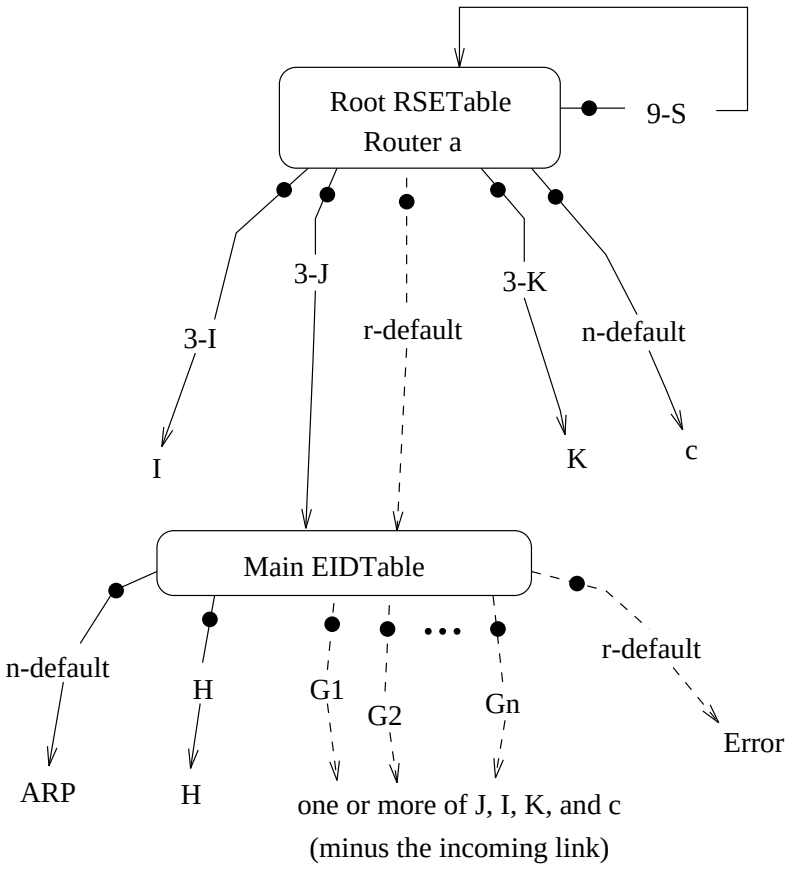


Figure 9.3: Router a's Forwarding Tables for Classical, Multicast, Shared-Tree Only SPip Example

group EIDs (G1 - Gn), each of which indicate one or more of router a's links. The actual set of a's links depends on the group membership, and on the scope of the group EID. Conceptually, each different scope results in a separate entry in the EIDTable. An implementation may, however, treat the scope field as a separate field.

These entries have the T-tag set. Note that, since these entries are for shared-tree multicast, the packet would not be transmitted over the link from which it was received.

Consider the multicast packet from host H shown above. It would arrive at router a as follows:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
H	G1	4	3J	9S	1B	**r1B**	r9S	r3J

Router a would access its Root RSETable with value r1B. This would retrieve the r-default entry, since there are no other reverse-path form entries in router a's Root RSETable. This causes router a to access its Main EIDTable with Dest EID G1. This entry indicates which links the packet should be forwarded over. The T-tag for this link is set, so the Active RSE would be set to zero

in all transmitted packets:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
H	**G1**	0	3J	9S	1B	r1B	r9S	r3J

Because the Active RSE in this packet is zero, subsequent routers would not look at the reverse-path form RSEs, but rather would directly access the Main EIDTable with the group EID. Thus, the processing overhead of the reverse-path form RSEs is only incurred once—at the first router. The packet length overhead of the reverse-path form RSEs, however, remains throughout the packets lifetime. Inclusion of the reverse-path form RSEs is the cost of hiding the nature of the multicast tree from hosts.

Source-Tree Multicast

With IP source-tree multicast, routers build reverse-path trees[5] rooted at the destinations of their unicast forwarding tables. This results in a set of potential outgoing links for multicast packets from each unicast destination. This set is then pruned based on the membership of each group. Thus, when a multicast packet is received, the source address and group ID together are used to determine the outgoing links.

SPip must of course do the same thing. In the case of SPip, however, the reverse-paths built from the unicast destinations are installed in the forwarding tables as reverse-path form RSEs. The reverse-path form RSEs in the packet, then, are used to indicate which source tree the packet is forwarded on.

For instance, assuming the hole-punching forwarding information case, the forwarding tables for routers a and c are as shown in Figures 9.4 and 9.5 respectively.

Consider the forwarding table for router a. The normal form (unicast) entries are drawn in solid lines, and the reverse-path form (multicast) entries are drawn in dashed lines.

Router a is in a subscriber network (S). If the source is inside subscriber network S, then router a must know which subnet the source is on so that the packet can be multicast over the appropriate tree. If the source is not inside subscriber network S, then router a does not care which backbone, subscriber, or subnet the packet came from, because all packets from outside enter subscriber network S via the same two interfaces[6].

Thus, one of the multicast entries of router a's Root RSETable is there simply to determine if the packet might be from inside subscriber network A by matching against backbone B (r1B). If no match occurs, the packet is assumed to come from outside of subscriber network S, and EIDTable 3 is chosen (r-default).

[5] The use of the term reverse-path does should not necessarily imply that a reverse-path routing algorithm [27] was used to form the tree. Forward-path routing information can also be used.

[6] We are assuming that backbone B transmits multicast packets to subscriber S via both of the interfaces. This is not necessarily the way all tree-building algorithms would work.

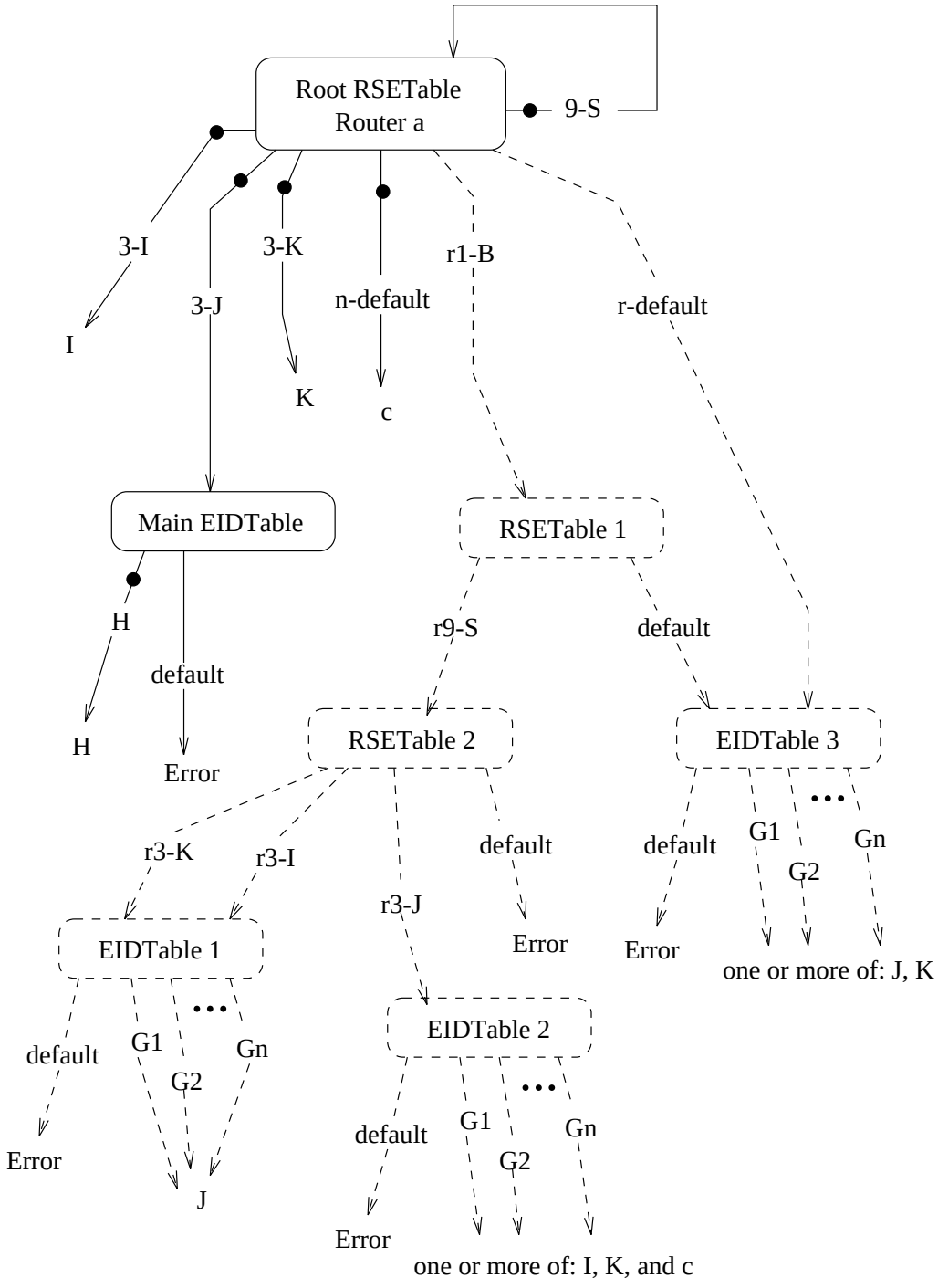


Figure 9.4: Router a Forwarding Tables for SPip Non-classical Source-Tree Multicast Example

EIDTable 3 indicates that the packet should be forwarded to neither, either, or both subnets J and K[7]. Which combination depends on the group membership for the group address. If both J

[7]This assumes that packets entering subscriber S from router b will be routed to subnet I, so there is no need for router a to forward them to subnet I.

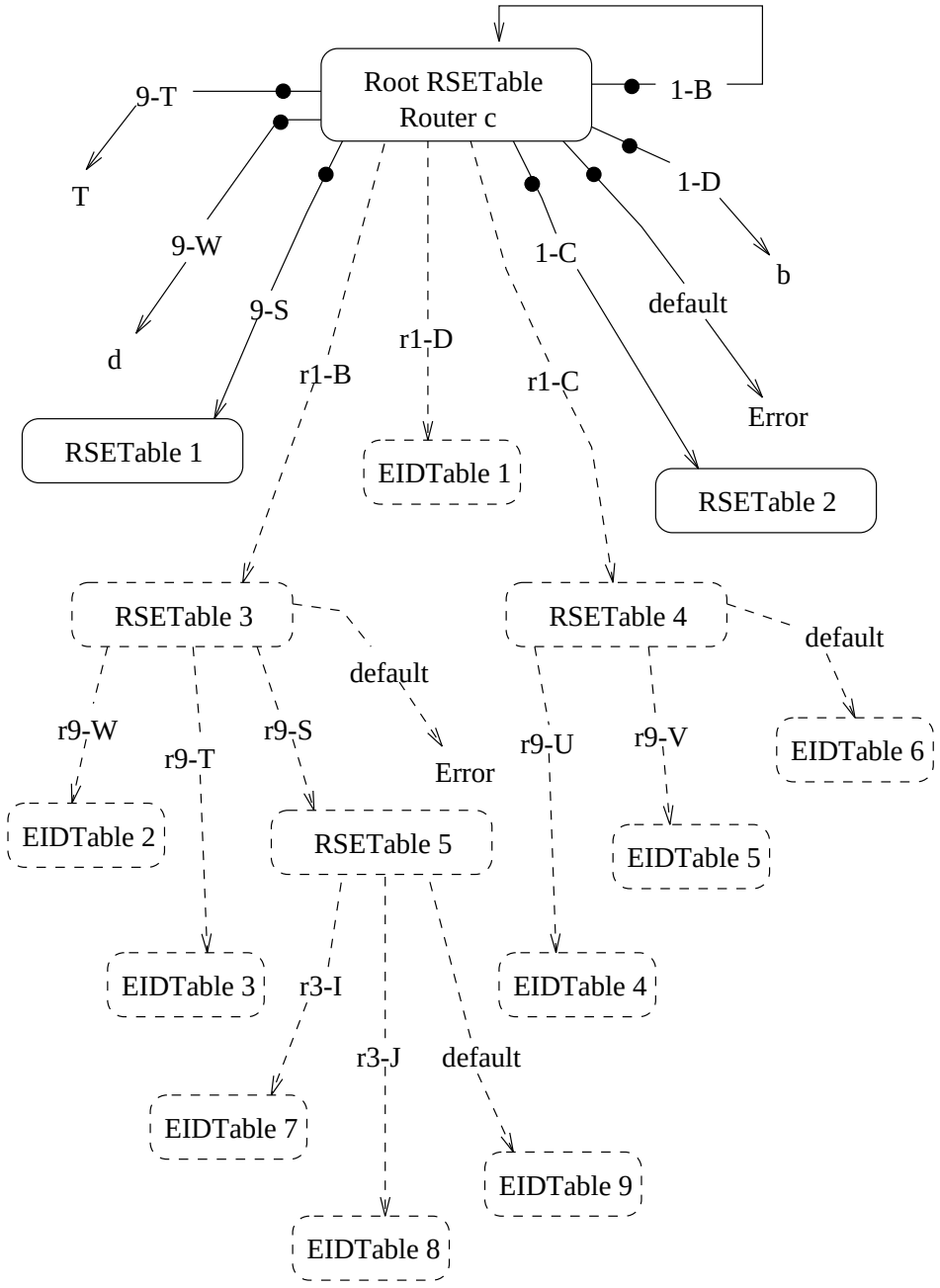


Figure 9.5: Router c Forwarding Tables for SPip Non-classical Source-Tree Multicast Example

and K have members then the packet will go to both. Only groups that have members on one or more of J and K are explicitly listed in EIDTable 3. Thus, if neither have members, then the group is not listed and the packet defaults to neither (that is, Error). (Note that EIDTable 3 has only one default, so there is no need to distinguish between n-default and r-default.)

If a packet at router a has a top-level RSE of r1B, then the entry in router a's Root RSETable

indicates a secondary forwarding table—RSETable 1. In RSETable 1, router a continues to check to see if the packet originated in subscriber network S by comparing against r9S. If, again, there is no match, the packet is assumed to be from outside and uses the default entry (EIDTable 3). If it does match, then the packet is known to be from a subscriber S subnet, and forwarding table RSETable 2 is accessed with the next RSE, the subnet number. Depending on the subnet, one of two EIDTables is chosen. (In this case, the forwarding is the same whether the packet is from subnet I or K. In the general case, there is a different tree for each origin.)

Now consider the forwarding tables of router c. Like Figure 9.4, Figure 9.5 shows the unicast entries in solid lines, and the multicast entries in dotted lines. To simplify things, Figure 9.5 is incomplete in that it does not show the entries for the leaf forwarding tables (RSETables 1 and 2 and EIDTables 1 through 9). The entries for RSETables 1 and 2 are identical to those in Figure 9.2, so do not need to be repeated here. The entries for EIDTables 1 through 9 are similar in form to the multicast EIDTables in Figure 9.4, so also do not need to be shown here.

Router c is in backbone B, and as such has no default routing per se. Thus, the single error default entry from the Root RSETable serves both normal form and reverse-path form addresses. Of the three reverse-path form entries in the Root RSETable, the one for backbone D (r1D) points directly to an EIDTable. This indicates that there is a single source-tree for any packets from backbone D (or its subscribers). The entries for backbones B (its own) and C point to RSETables. This indicates that there are multiple trees with the same top-level RSE, and thus additional RSEs must be examined.

In the case of the backbone C entry (r1C), the multiple trees come from the fact that routers in backbone B have explicit forwarding information for subscribers U and V in backbone C. This information is reflected in RSETable 4. Note that the default entry in RSETable 4 handles the backbone C subscriber networks not explicitly known to router a (EIDTable 6).

In the case of the backbone B entry (r1B), router a must distinguish between the various subscriber networks attached to backbone B (RSETable 3). Router a must also distinguish between subnets I and J in subscriber S (RSETable 5).

A notable fact about the multicast forwarding tables for routers a and c is that at no time is the Active RSE in the packet changed. There are no set T-tags in any of the multicast entries. For the full duration of the packet transmission through the network, the Active RSE always points to the top-level RSE.

The reason for this is that, contrary to the unicast case, the more detailed addressing information is needed at the start of the path, not at the end. With unicast, routing considers only the top-level addressing information until the packet approaches the destination, at which time lower-level addressing information is examined. With source-tree multicast, the forwarding is with respect to the unicast address of the source (and the group ID), and so the detailed (lower-level) addressing information is examined at the start of the path, and later in the path only the top-level

information is examined.

Because routers later in the path look at the top-level information, the Active RSE must point to the top-level RSE. Since there is no way to back-up the Active RSE, routers at the beginning of the path cannot advance the Active RSE to point to the lower-level RSEs.

Lowest-Level First

An alternative approach to packet format for source-tree multicast is to reverse the order of the reverse-path form RSEs in the packet header. Thus, the RSEs are in order of lowest-level RSE first. All other aspects of header formation are the same as described above for highest-level first. The potential advantage of this method is that routers at the beginning of the path can examine only the lower-level RSEs rather than having to examine all of the RSEs. Later in the path the Active RSE can be advanced to point at the higher-level RSEs.

The problem with this approach is that it has a scaling problem. When the RSEs are parsed in order of lowest RSE first, there can be many possible lower-level RSEs that ultimately resolve to the same higher-level RSE. It turns out in many cases, especially with hole-punching, that each of these lower-level RSEs results in a different set of branches, ultimately leading to the same EIDTable, but replicating the higher-level RSEs multiple times.

Rather than give an example (which requires quite a lengthy explanation—just to explain a bad idea), I leave it to the reader to try to create the forwarding tables necessary for lowest-level first RSEs for the non-classical forwarding information case. The reader will quickly discover that it results in a large and complex set of forwarding tables.

Well-Known Multicast

SPip has reserved a portion of its group EID space for well-known multicast, and has scoping. Thus, SPip can do well-known multicast groups.

Two-Phase Multicast

SPip can do two-phase multicast with both shared-tree and source-tree multicast. Recall from the formatting rules given at the beginning of this section that an SPip host only includes the reverse-path form RSEs if there is no address sequence associated with the group EID. With two-phase multicast, there is an address sequence associated with the group EID—that of the unicast destination from which the multicast phase originates. Thus, the route sequence of two-phase multicast packets contain the source address sequence (in normal form) followed immediately by the address sequence associated with the group EID.

Shared-Tree Two-Phase

If the multicast phase is shared-tree, then the address sequence associated with the group EID is nothing more than the unicast address sequence of the multicast-phase origin. For instance, the destination information associated with a two-phase multicast to group G1 that starts its multicast phase at subnet J is ⟨EID = G1, ASeq = 1B:9S:3J⟩ [8]. If host Z is to send a packet to this group, it would receive that destination information from some source (such as IGMP), and format a packet as follows:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
Z	G1	4	3L	9V	1C	**1B**	9S	3J

The packet would reach router a (or some router on subnet J) as per unicast forwarding. Using a forwarding table such as that of Figure 9.4, router a would access its Main EIDTable with the group EID G1. In this case, EIDTable 1 would have an explicit entry for G1, match on that entry, and forward the packet according to the multicast tree for G1. The entry would have the T-tag set, so the Active RSE would be set it 0, and the packet would from then on be forwarded as multicast.

If, rather than reaching a router on subnet J, it is desired that any router on backbone B initiate the multicast phase, then the destination address sequence only requires the backbone-level RSE, as follows:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4
Z	G1	4	3L	9V	1C	**1B**

Note that, because 1B is the last RSE in the sequence, it has the Last RSE flag set. This is how the backbone router knows to access the EIDTable rather than look for another RSE.

Now consider how to do CBT-style two-phase. The destination information contains the unicast address of the core of the CBT tree, say for instance a router z in backbone c: ⟨EID = G2, ASeq = 1C:9X:3Y:0Z⟩, where 9X and 3Y are subscriber and subnet RSEs, and 0Z is the host-level RSE for router z. Every router on any CBT tree with z as the core installs a chain of RSETables with 1C, 9X, 3Y, and 0Z in turn as entries. None of the entries have the T-tag set. The last RSETable entry (0Z) points to an EIDTable containing the group EIDs. The entries in the EIDTable have the T-tags set. The default entry in the EIDTable has the normal unicast forwarding information that would have been retrieved for a unicast packet to 1C:9X:3Y:0Z.

When a router on the tree receives a CBT packet with z as the core, it traverses the chain of RSETables and looks up the group G2 in the EIDTable. If no match occurs, the packet is forwarded towards z as per normal unicast. If a match occurs, then the Active RSE is set to 0, and the packet is forwarded as per multicast.

[8] Note that if the multicast phase is to start from a host, a host-level RSE in the address sequence is required. Host-level RSEs are discussed in Section 9.8.3

Source-Tree Two-Phase

To do two-phase where the packet is source-tree multicast from the origin of the multicast phase, it is necessary to include the reverse-path RSEs of the multicast phase origin. Thus, the destination information for a packet to group G2 and originating at router z with address sequence 1C:9X:3Y:0Z is:

$$\langle \text{EID} = \text{G2},\ \text{ASeq} = \text{1C:9X:3Y:0Z:r1C:r9X:r3Y} \rangle$$

The 0Z RSETable entry points back to the Root RSETable. Thus, router z accesses the Root RSETable with r1C. This causes the normal source-tree multicast actions to occur as described above.

9.2.4 Discussion

All three protocols can do the most basic multicasts—single-phase, shared-tree and source-tree, broadcast and multicast.

CLNP is currently lacking scoped addresses. Multicast in CLNP, however, is still in the early stages of definition, so it is presumed that eventually CLNP will have scoping. In any event, there is certainly nothing in CLNP per se that would prevent scoping—there is plenty of room in the NSAP address to define a scope field. It would be preferable if this field were in the same well-known place for all group addresses. This, however, is not the direction that OSI is taking for group addressing [57]. In any event, it remains to be seen whether well-known addressing beyond the local LAN is useful.

Because of inadequate scoping, CLNPs ability to do well-known addressing is somewhat limited. Fortunately, CLNP can do a very important class of well-known group address—the one with a scope of the local LAN. In fact, existing CLNP control protocols such as ES-IS and IS-IS take advantage of local LAN multicast, though only by using well-known IEEE-802 link-level group addresses, not CLNP group addresses.

Both SIPP and SPip have scoping and well-known group addressing.

CLNP cannot handle any two-phase multicast, primarily because of a bug in CLNP's LSR mechanism, but also because CLNP hosts are not commonly expected to format LSR packets. SIPP handles two-phase multicast with shared-tree, but not source-tree, in the multicast phase. In particular, SIPP handles CBT-style multicast (two-phase with shared-tree, where the multicast phase can start anywhere on the shared-tree). This is a potentially important form of multicast. For instance, it is part of all ongoing work in multicast in the IETF standards organization [49].

SPip handles all the forms of multicast discussed.

A general observation is that, in order to make multicast work, CLNP and SIPP do "something

special" in the routers, while SPip does "something special" in the hosts. That is, upon discovering that a packet is multicast, CLNP and SIPP break out of the "normal" lookup algorithm (that is, of looking at destination address and possibly source route), and look at the source address. SPip routers, on the other hand, do the same thing for unicast and multicast, but in order to format a multicast packet, SPip hosts break out of the "normal" formatting algorithm (that is, of putting the source address sequence at the beginning of the route sequence and the destination address sequence afterwards), and adds the reverse-path form source address sequence between the two.

Thus, while SPip has perhaps succeeded in creating an efficient yet general purpose router forwarding mechanism, it has not as successfully created a general purpose host packet formatting mechanism.

9.3 Mobility

There are several aspects to mobility—how hosts and routers discover each other, how it it determined when a host needs a new address, how the new address is assigned, how a correspondent host learns the new address of the mobile host, and how a correspondent host authenticates the new address.

This section does not concern itself with these aspects (though Section 9.5 discusses auto-address assignment). Rather, it considers only 1) how packets are routed between a mobile host and a correspondent host once the correspondent host has learned the new address of the mobile host, and 2) how the two hosts identify each other.

We consider the following scenario. Host H is attached to subnet J and is exchanging packets with host Z attached to subnet L. Host H then moves to subnet I. We consider two cases, one where host H does not obtain a new address on subnet I, and the other where host H does obtain a new address. In the former case, host H's individual address must be advertised off of the subnet, for instance to router a. In the latter case, host Z must learn host H's new address and start using it.

9.3.1 CLNP

Initially, host H on subnet J has address B.S.J.H. Host Z has address C.V.L.Z. Packets are being exchanged using these addresses according to description in Section 9.1.1.

Host H moves to subnet I but does not obtain a new address. Thus, packets from host Z to host H still have the address B.S.J.H. These packets are forwarded to router a. Now, however, router a must forward the packet over to subnet I rather than on subnet J. Thus, the routing algorithm running in subscriber network S must now carry an explicit entry for host H on subnet I (but with address B.S.J.H).

If the IS-IS routing protocol is being used [56], and subnet I is in the same area as subnet J, then host-level routing information is carried as a matter of course.

Assuming that router a has obtained the new routing information for host H, router a forwards the packet to subnet I, and the packet is delivered to host H.

Now, assume that host H obtains a new address, B.S.I.H, on subnet I. This would be necessary if, for instance, subnet I were in a different IS-IS area. This would imply that the host has moved "far enough" that maintaining host-level routing information all the way back to the host's former position is too much overhead.

Under the current level of CLNP specification, there is no way that host Z can continue exchanging packets with host H (at least, not without tearing down the active application associations). This is because the NSAP address doubles as the host identifier. Once host H gets the new address, it also gets a new identifier. Unless the previous identifier (the true identifier) is carried in the new packets, and conveyed to host Z's applications as before, host Z cannot recognize that the packets with address B.S.I.H are coming from the same hast as B.S.J.H.

Presumably CLNP standards will continue to progress, and this shortcoming will be solved. There are two basic approaches available to CLNP. One is to declare some subset of the NSAP address to be globally unique. This could be byte positions 1 through 6 (where byte position 0 is the low-order byte), since those positions already hold the IEEE-802 address in many cases. Or it could be byte positions 1 through 8, to allow for more definitions in the future.

The other approach is to convey the entire previous address in some part of the header—an option or an encapsulated header. This is the approach being pursued by IP, which does not have the luxury (or overhead, depending on what side of the fence you're on) of incorporating the IEEE-802 address.

Either way, the new address could either be learned by host Z, and conveyed in packets from host Z, or it could be learned by some system on subnet J—perhaps router a. In this latter case, router J would modify the packet to contain the appropriate information. The latter case has the advantage of keeping host Z simple (it does not need to know anything about the new location of host H), but results in a longer path, as now packets must be forwarded through router a (whereas if host Z learned the new location of host H, the packet would be routed through router b and directly to subnet I).

9.3.2 SIPP

We initially assume that single (non-extended) addresses are being used. Thus, the addresses used at the start of the communications (when host H is on subnet J) are B.S.J.H and C.V.L.Z, and packets from host H to host Z have the following simple format:

Source Addr	Dest Addr
B.S.J.H	C.V.L.Z

For the case where host H keeps the same address after moving to subnet I, host-level routing information is distributed as described for CLNP above, and the packet format remains the same.

For the case where host H obtains a new address, host H learns the cluster address for subnet I, which is B.S.I.0[9]. Host H forms an address sequence using the cluster address as the high-order address, and its original address as the low-order (and identifying) address. Thus, host H's address sequence is B.S.I.0:B.S.J.H. Packets from host H to host Z are formatted as follows:

Source Addr	Dest Addr	Next Addr	Addr1
B.S.J.H	C.V.L.Z	2	B.S.I.0

where a Next Addr value of 1 points to the first address in the source route (thus, the above Next Addr value of 2 is pointing beyond the single address in the source route).

This packet is routed to host Z. Since host Z only uses the (low-order) identifying address to identify host H, host Z recognizes this packet as being from host H even though the source route has been added. Host Z reverses this packet according to the rules in Section 8.2.2, producing a return packet of:

Source Addr	Dest Addr	Next Addr	Addr1
C.V.L.Z	B.S.I.0	1	B.S.J.H

Host Z could also have learned of host H's new address by some other mechanism, such as a query to host H's base station (the system that keeps track of host H's current address).

This packet is routed to a router on subnet I, which recognizes the cluster address B.S.I.0 as being for itself, and advances the packet, producing:

Source Addr	Dest Addr	Next Addr	Addr1
C.V.L.Z	B.S.J.H	2	B.S.I.0

The router knows that address B.S.J.H is on its own subnet, because host H advertised it there, and delivers the packet to host H.

Now consider the case where host H is using extended addresses of the form described in Section 9.1.3. That is, the identifying address is the local-use address and the high-order address is the subscriber prefix—B.S:J.H. Host Z, likewise, has address C.V:L.Z

In this case, packets leaving host H on subnet J are formatted as:

Source Addr	Dest Addr	Next Addr	Addr1	Addr2
J.H	C.V	2	B.S	L.Z

and arrive at host Z as:

[9]This cluster address is normally advertised by routers for the purpose of host auto-address configuration.

Source Addr	Dest Addr	Next Addr	Addr1	Addr2
J.H	L.Z	3	B.S	C.V

After host H moves to subnet I, it keeps its identifying address J.H, and gets a new prefix of B.S:I.0. Packets from host H's new location to host Z are initially formatted as:

Source Addr	Dest Addr	Next Addr	Addr1	Addr2	Addr3
J.H	C.V	3	I.0	B.S	L.Z

And return packets leave host Z as:

Source Addr	Dest Addr	Next Addr	Addr1	Addr2	Addr3
L.Z	B.S	2	C.V	I.0	J.H

9.3.3 SPip

Forming packets for mobility in SPip is straightforward.

Packets from host H on subnet J are formatted as:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
H	Z	4	3J	9S	1B	**1C**	9V	3L

When host H moves to subnet I, its address sequence changes from 1B:9S:3J to 1B:9S:3I. Thus, packets from host H on subnet I are formatted as:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
H	Z	4	3I	9S	1B	**1C**	9V	3L

And return packets as:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
Z	H	4	3L	9V	1C	**1B**	9S	3I

The same comments made for the CLNP case—namely that the packets to host H on subnet I could have been formatted by host Z or by host H's base station—apply here (as well as to the SIPP case).

9.3.4 Discussion

Both SPip and SIPP handle mobility without changes to the basic packet formatting rules. Indeed, within the context of the reversing rules, it is possible for an SPip or SIPP host that is not mobile and that has no notion of mobility of other hosts to successfully communicate with a mobile host without going through a base station. This requires, however, that the mobile host sends a packet to the correspondent host when it gets a new address, so that the correspondent host can record the new address. If, on the other hand, the correspondent host is mobility-smart, it could send a query to the mobile host's base station to learn the new address.

To handle mobility, CLNP must be modified to carry both the initial address (for identification) and the new address (for routing) in the packet header.

In terms of control protocols to handle mobility (update and query messages, discovery messages, and so on), none of the three protocols (assuming that CLNP packets were modified to handle mobility) has any particular advantage over the others.

9.4 Domain-Level Policy Route

A *domain-level* policy route is a route whereby the path is specified in terms of the high-level clusters that a packet should go through, particularly backbones. The term domain is used here (rather than, say, backbone) because this term is commonly used when discussing policy routes (for instance, [11]). The policy routing discussed in this section is limited to backbones.

There are two policy routing applications of interest—provider selection[10] and full policy route. Provider selection is where only the providers on either end of the path are selected. These providers are either directly connected to the subscriber networks or reachable through a local-access provider.

A full policy route is where the source can specify a number of backbones on the path from source to destination, not just the providers for the source and destination.

The examples for this section are from the topology shown in Figure 9.6. Figure 9.6 shows two subscriber networks, S and V, and 10 backbones, A through D and M through P. All examples are for packets between hosts H and Z.

Note that both subscriber networks are connected to two providers each, B and D in the case of subscriber network S, and A and C in the case of subscriber network V. Assuming provider-rooted addressing, this gives the hosts in both networks two addresses each—one with a prefix from one provider, and one with a prefix from the other provider.

9.4.1 Provider Selection

We are interested in the following scenarios:

1. Host H in subscriber network S initiates an exchange with host Z in subscriber network V. Packets leave subscriber network S via provider B and arrive at subscriber network V via provider C. Return packets take the symmetric reverse path (that is, they go through the same two providers).

[10] The terms provider and backbone are used somewhat interchangeably. In general, provider is used when referring to the backbone in its provider/subscriber role, and backbone is used when referring generically to a top-level network.

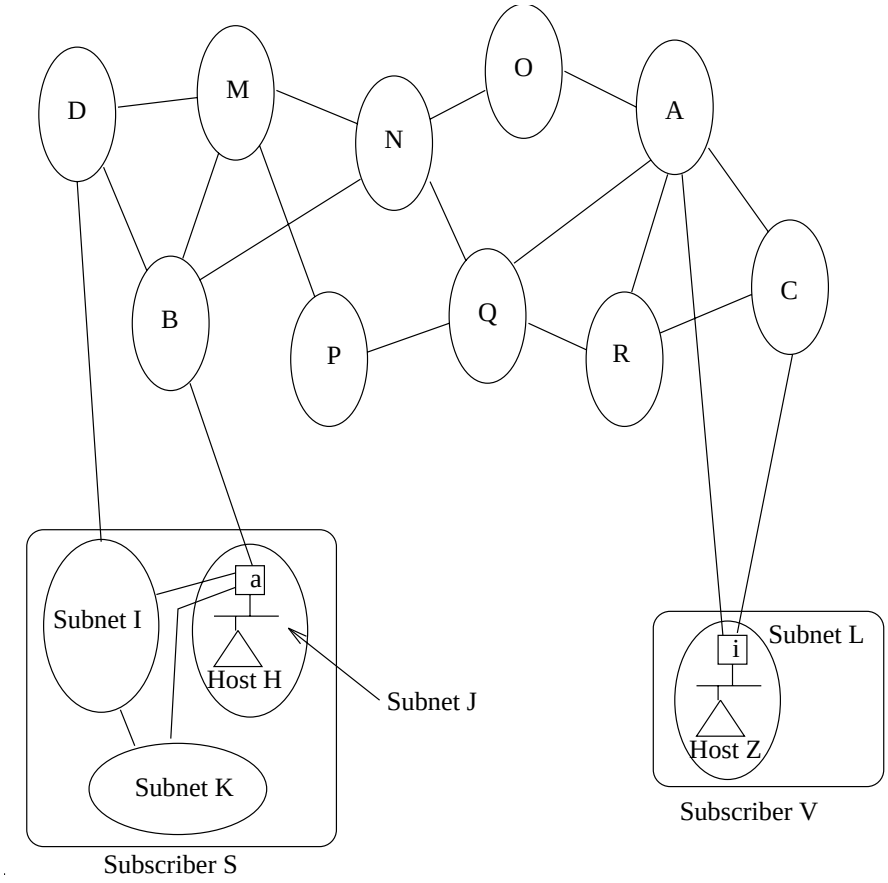


Figure 9.6: Example Topology for Policy Examples

2. Assume scenario 1 is underway and packets have been exchanged. In the middle of the exchange, a different provider network is selected. (This would be desired if, for instance due to a failure, service was no longer available through provider B. This could also happen after the first packet is sent because the destination-end provider assumed by the initiator is not the one preferred by the destination.) Now, packets should exit and enter subscriber network S via provider D (but still enter and exit subscriber network V via provider C).

3. A new exchange is started between hosts H and Z. As with scenario 1, packets leave subscriber network S via provider B and arrive at subscriber network V via provider C. Return packets, however, take an asymmetric return path, leaving subscriber network V via provider C but entering subscriber network S via provider D.

CLNP

CLNP can use the choice of destination (provider-rooted) address to influence the destination-end provider chosen, as discussed in Section 3.4. CLNP can use the source specific QoS field to influence the source-end provider chosen.

QoS in CLNP is encoded as a single value. The first two bits of the value, however, indicate one of three classes of QoS—source specific, destination specific, and globally unique. If the class is source specific, then the value is interpreted in conjunction with the source address (or, more commonly, a source address prefix). That is, the value is concatenated with the source address to create a composite QoS value. Therefore, the same QoS value with different source addresses is interpreted differently. The destination specific value is interpreted in conjunction with the destination address. The globally unique value is not specific to either source or destination address.

For source-end provider selection, the subscriber network assigns a source specific QoS value for each provider network. To choose the source-end provider, a host inserts the QoS value matching the provider chosen. The forwarding tables in the subscriber network routers are formed such that any packet with a source specific QoS value for a given provider gets forwarded to that provider. If the destination is not reachable through that provider, the packet may either be discarded or routed through another provider (or routed to the selected provider to be discarded there, for instance if default routing is in effect). If the chosen provider is reachable through a local-access provider (see Section 3.4.2), then the routers in the local-access provider must keep entries for all of its subscriber's QoS values.

Thus, to send a packet according to scenario 1 above, host H forms the following header:

Source Addr	Dest Addr	SS QoS
B.S.J.H	C.V.L.Z	B

Host H chooses B.S.J.H rather than D.S.J.H as its source address so that return packets will come via provider B. This packet is forwarded by routers in subscriber network S by examining the SS QoS field (a value indicating "provider B") in conjunction with the source addr. The routers in provider B do not have any forwarding information for SS QoS = ⟨B, B.S.J.H⟩, and so route the packet on destination address only. This causes the packet to be routed to provider C, from which it is routed to subscriber network L and then to host Z.

To return packets on a symmetric path to host H, host Z reverses source and destination address, and must also choose an SS QoS commiserate with the address it is using for itself. In other words, since the address it is using shows provider C, it picks an SS QoS that indicates provider C:

Source Addr	Dest Addr	SS QoS
C.V.L.Z	B.S.J.H	C

This packet will exit subscriber network V through provider C (because of the SS QoS), and enter subscriber network S through provider B (because of the destination address B.S.J.H). Note that hosts H and Z could have done this without any particular coordination between themselves as long as 1) the providers chosen by host H are acceptable to host Z, and 2) host Z assumes that host H wants symmetric routes.

For scenario 2, host H chooses a new SS QoS to cause packets to be routed through provider D:

Source Addr	Dest Addr	SS QoS
B.S.J.H	C.V.L.Z	D

While this does cause packets to exit via provider D, return packets from host Z will continue to be formed as shown above, causing return packets to enter via provider B. Short of using loose source routing, there is no way that host Z can cause packets to enter subscriber network S via provider D, because the addresses cannot be changed during an application association.

For scenario 3 above, host H forms the following packet:

Source Addr	Dest Addr	SS QoS
D.S.J.H	C.V.L.Z	B

Host Z turns this packet around the same way as the previous example, producing:

Source Addr	Dest Addr	SS QoS
C.V.L.Z	D.S.J.H	C

Thus, host H can cause packets to return via a different provider from the one over which it was received.

SIPP

Like CLNP, SIPP uses provider-rooted addresses to route packets through the selected destination-end provider.

To route packets through the selected source-end provider, SIPP uses a cluster address in the route sequence.

Assume simple (non-extended) addresses. Under scenario 1, host H formats its packets as follows:

Source Addr	Dest Addr	Next Addr	Addr1
B.S.J.H	B.0	1	C.V.L.Z

The cluster address B.0 in the Dest Addr field causes the packet to be routed to provider B. The border router of provider B advances the route sequence, thus routing the packet to C.V.L.Z. This address causes the packet to go to provider C, and from there to host Z.

To return a packet, host Z reverses the route sequence, and adds a cluster address of C.0 to cause the packet to be routed through provider C outgoing:

Source Addr	Dest Addr	Next Addr	Addr1	Addr2
C.V.L.Z	C.0	1	B.0	B.S.J.H

Note that the cluster address of backbone B is in this route sequence. This cluster address is redundant in that host H's address alone is sufficient to cause the packet to be routed through backbone B. Host Z cannot, through inspection of the packet header alone, easily know that address B.0 is in fact a cluster of B.S.J.H, and is therefore redundant. This is because there is no

information in the addresses that indicate the location of the field boundaries of the hierarchical address. This information is passed around in routing algorithms, but hosts are not usually privy to this information. In fact, even routers in subscriber network V are unlikely to have such high-level address boundary information, as they would most likely use default routing to exit the subscriber network.

In particular, a host cannot simply compare a cluster address and a full (non-cluster) address and determine with certainty that the cluster address is for a cluster that the full address is in. For instance, assume that the numerical value of B.S.J.H is the 64-bit address 12.34.56.78.9a.bc.de.f0, and that the prefix 12.34 represents backbone B. Thus the cluster address B.0 is 12.34.00.00.... Assume that there is another backbone Q whose backbone prefix is 12.34.56 (and whose cluster address Q.0 is 12.34.56.00...). If a host receives a route sequence of Q.0:B.S.J.H, the prefix of both addresses (up to the 0's of Q.0) is the same. However, the cluster address is obviously not redundant information.

When host H sends another packet to host Z, it reverses the packet received from host Z, producing:

Source Addr	Dest Addr	Next Addr	Addr1	Addr2
B.S.J.H	B.0	1	C.0	C.V.L.Z

Thus, both cluster addresses are in all subsequent packets, even though one of them is always redundant.

Now assume that scenario 2 begins. To cause packets to go through provider D, host H must at a minimum replace the cluster address of provider B with that of provider D:

Source Addr	Dest Addr	Next Addr	Addr1	Addr2
B.S.J.H	D.0	1	C.0	C.V.L.Z

This header does cause the packet to be forwarded through provider D outgoing. However, there is a problem with the return packets. The reversed packets from host Z are formatted as:

Source Addr	Dest Addr	Next Addr	Addr1	Addr2
C.V.L.Z	C.0	1	D.0	B.S.J.H

These packets are routed through provider C to provider D. The border router of provider D advances the route sequence, producing:

Source Addr	Dest Addr	Next Addr	Addr1	Addr2
C.V.L.Z	B.S.J.H	3	C.0	D.0

With classical forwarding information, the border router of provider D would forward this packet to provider B. The packet, however, should not go through provider B. Instead, the packet should be forwarded directly to subscriber S without going through provider B.

One way to solve this problem is by having subscriber S advertise its subscriber prefix from provider B to provider D. Thus, routers in provider D would have forwarding table entries for

prefix B.S.*... (as well as for B.*...)

This solution has two problems. First, it can result in scaling problems in provider D. If provider D has a large number of subscribers, it can internally cluster those subscribers so that its routers do not require per-subscriber forwarding information except for the subscribers in their cluster. If subscribers advertise subscriber prefixes assigned from other backbones, these prefixes will not in general fit into provider D's internal clustering scheme, and so routers in provider D must keep per-subscriber information.

The second problem is that host H has no (simple) way of knowing if prefix B.S.*... has been advertised to provider D or not. Host H therefore does not really know if it can form the above route sequence or not. Thus, host H must instead generate the following route sequence:

Source Addr	Dest Addr	Next Addr	Addr1	Addr2	Addr3
B.S.J.H	D.0	2	D.S.0	C.0	C.V.L.Z

This is reversed by host Z, producing:

Source Addr	Dest Addr	Next Addr	Addr1	Addr2	Addr3
C.V.L.Z	C.0	1	D.0	D.S.0	B.S.J.H

Thus, when the border router of provider D receives the packet and advances the route sequence, it produces:

Source Addr	Dest Addr	Next Addr	Addr1	Addr2	Addr3
C.V.L.Z	D.S.0	3	C.0	D.0	B.S.J.H

This packet gets forwarded on address D.S.0, which causes it to go to subscriber S. The border router of subscriber S advances the route sequence to address B.S.J.H, which causes the packet to go directly to host H.

The third scenario requires some kind of coordination between the two hosts. That is, simple reversal of host H's packets will not result in the right behavior.

For instance, host H could format its packets as follows:

Source Addr	Dest Addr	Next Addr	Addr1
D.S.J.H	B.0	1	C.V.L.Z

The cluster address B.0 will cause packets to exit via provider B. The reversed packet is:

Source Addr	Dest Addr	Next Addr	Addr1	Addr2
C.V.L.Z	C.0	1	B.0	D.S.J.H

This packet will be routed to provider B before being routed to provider D, which is not the desired behavior. Instead, host Z requires specific knowledge that host H does not want return packets to go through provider B. Then host Z could remove the cluster address B.0 from the returned route sequence.

SPip

Under scenario 1, host H formats packets as follows:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
H	Z	3	3J	9S	**1B**	1C	9V	3L

The only difference between this packet and the packets for the unicast example in Section 9.1.5 is that the Active RSE is set to be the top-level RSE in the *source* address sequence. This causes the packet to be routed through provider B on the way to provider C.

Return packets are formatted as follows:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
Z	H	3	3L	9V	**1C**	1B	9S	3I

When, in scenario 2, host H decides to route its packets through provider D, it simply changes the third RSE as follows:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
H	Z	3	3J	9S	**1D**	1C	9V	3L

For scenario 3, host H forms packets as follows:

SEID	DEID	Active RSE	Num Source RSE	RSE1	2	3	4	5	6	7
H	Z	4	3	3J	9S	1B	**1D**	1C	9V	3L

Note that the above header shows the Num Source RSE set to point to RSE3 (1B, the top-level RSE of the source address sequence). (Previous SPip headers have not bothered to show the Num Source RSE.) We show it this time to underscore the fact that host Z's default behavior on returning this packet is to use just the RSEs indicated by the Num Source RSE field—that is, the first 3 RSEs. Thus, the 4th RSE (1D) would not be included in the return packet:

SEID	DEID	Active RSE	RSE1	RSE2	RSE3	RSE4	RSE5	RSE6
Z	H	3	3L	9V	**1C**	1B	9S	3I

This packet does not go through provider D in the return path.

Discussion

CLNP is surprisingly (to me) adept at handling provider selection. The only thing it can not do is change providers after establishing an application association. CLNP hosts must, however, be configured with appropriate SSQoS information.

Both SIPP and SPip handle scenarios 1 and 2, though SPip more simply than SIPP. For SIPP to do provider selection, the host must learn the cluster address of its providers. As the provider cluster address is not something otherwise needed by the host, some additional configuration or discovery is required to do provider selection. Discovery is complicated by the fact that the only

systems that naturally know the provider cluster address are the subscriber border routers, which are not directly connected to hosts.

Ironically, CLNP can do scenario 3 while SIPP cannot—at least, not without coordination between the two hosts. (SPip does scenario 3 as well.) On the other hand, since with CLNP the provider cannot be changed after the first packet has been sent, coordination is required between the two hosts to make sure that the initiating host knows which destination-end provider the receiving host desires.

9.4.2 Full Policy Route

In this section, we describe how the three protocols can cause packets from host H to host Z to go through backbones B, N, Q, R, and C.

CLNP

CLNP is not capable of forming the above policy route.

If CLNP's loose source route bug were fixed, then CLNP could approximate the policy route by targeting individual routers in each of the desired backbones. Host Z, however, could not reverse the policy route, so the return path would be asymmetric.

SIPP

Assume simple (non-extended) addresses. To send a packet along the above policy route, host H forms the following packet:

Source Addr	Dest Addr	Next Addr	Addr1	Addr2	Addr3	Addr4	Addr5
B.S.J.H	B.0	1	N.0	Q.0	R.0	C.0	C.V.L.Z

Packets reversed by host Z would follow the reverse path.

SPip

To send a packet along the above policy route, host H forms the following packet:

SEID	DEID	Active RSE	RSE1	2	3	4	5	6	7	8	9
H	Z	3	3J	9S	**1B**	1N	1Q	1R	1C	9V	3L

Packets reversed by host Z would follow the reverse path.

9.5 Host Auto-Address Configuration

In this section, we consider the capability of the internet protocol to support plug-and-play operation. Specifically, we are interested in considering the amount of manual configuration necessary for hosts to bring up the internet protocol itself. We do not consider routers because routers require a certain amount of configuration, for instance for the routing protocols, as a matter of course. Thus, auto-configuration of addresses in routers is not particularly useful (in addition to being much harder).

There are two kinds of auto-address configuration—serverless and server-based. With serverless auto-configuration, a host can configure a complete, globally routable internet address without talking to a server *on an individual basis.* By individual basis, we mean that any server that might be involved does not give different hosts different information.

For instance, a common form of auto-configuration (indeed, the form we are interested in here) is where a host listens to a router advertisement of the subnet prefix, and appends its own host ID, thus creating its address. In this case, the router is a kind of server, but its prefix advertisement is not dependent on individual hosts. Thus, this is server-less auto-configuration.

All three protocols can do server-less auto-address configuration, and all roughly the same way. That is, each protocol allows the host to append its IEEE-802 address (either the one on its interface card, or the one associated with its CPU) to the prefix. The prefix is globally routable to the subnet, and the IEEE-802 address is guaranteed to be unique on the subnet[11]. Thus, the resulting address is globally unique and routable.

In the case of CLNP, the IEEE-802 address is positioned in the ID portion of the CLNP address.

In the case of SIPP, the local-use address format is used. To form a globally routable SIPP address using a local-use address requires an address sequence. This increases packet header size and reduces forwarding performance (see Section 10.1). To avoid using an address sequence, a SIPP host could form a local-use address only for the purpose of exchanging packets with an address server, which then gives it a single, globally routable address. This "server-based" address configuration, however, is more complex

In the case of SPip, the standard unicast EID is formed using an IEEE-802 address. The address sequence is learned from the router advertisement.

[11] This is true if the address is of the "universally administered" class, and if the vendor assigning the address has done it properly.

9.6 Type-of-Service (ToS) Field

This section describes how the three protocols can do ToS Field style routing in the IP sense. That is, where there are a small number of service types that influence the route the packet takes. In the case of IP, the ToS Field may also influence how the packet is internally queued, or how it is transmitted over the link. This aspect of the ToS Field is outside the scope of routing and addressing, and is not considered here. In IP, the available types are precedence, delay, throughput, and reliability.

9.6.1 CLNP

CLNP's globally unique QoS Field is in many ways similar to IP's ToS Field. That is, a set of values are defined to provide a small number of well-known service types. In CLNP, those types are sequencing, delay, cost, and residual error probability.

As shown above in the context of provider selection (Section 9.4.1), CLNP can also use its source specific and destination specific QoS parameters to define additional types of service.

9.6.2 SIPP

SIPP has no equivalent to IP's ToS Field. Note that this is an intentional design decision on the part of the SIPP designers. The is because ToS Field, especially in the routing sense, has not been proven to be a useful tool in the internet. Since every feature has a cost, SIPP is careful not to include features whose benefit is marginal.

SIPP, on the other hand, does have encodings for different traffic classes, plus a flow identifier. The traffic class information is only for the purpose of determining how to queue the packet in a router, not for determining how to route the packet. The flow identifier could be used for routing, though the current SIPP specifications do not discuss this. As Section 2.2.6 discusses, the flow identifier does not contain routing information per se. Rather, it is a short hand for other routing information already in the packet.

9.6.3 SPip

The 8-bit Path Modifier can be used for ToS Field routing in SPip. The low-order bit determines whether the corresponding RSID is normal form or reverse-path form. The remaining 7 bits are free for future assignment. One possible future assignment is IP-style ToS Field.

Since each RSE has a Path Modifier field, the ToS Field could be set separately for each RSE—for instance, for each level of the hierarchy. Normally, however, all of the Path Modifiers for a given address sequence would be set to the same value.

Note that the 7 free bits of the Path Modifier can be set on reverse-path form RSEs as well as on normal form. Depending on how the reverse paths were calculated, this could have the effect of forwarding a multicast packet over a tree appropriate to the requested ToS.

9.7 Embedded Link-Layer Addresses

Chapter 5 discusses the use of embedded link-layer addresses. Of the three protocols, only CLNP explicitly encodes link-layer addresses in its header. However, as described in chapter 5, the way they are used in CLNP is faulty and limited.

An alternative approach is to include the link-layer address(es) in an options field.

This could work with SPip as follows. An options field is included in the header that contains the link-layer addresses for each subnetwork that requires it (that is, for each subnetwork for which the link-layer address lookup is too expensive or impossible).

The link-layer addresses in the options field are labeled to indicate which RSID each link-layer address applies to. That is, each address in the options field is of the form:

RSE Number	Link-layer Addr

Since each RSID refers to only one subnet, it always clear where the link-layer address applies. Thus, the problem that exists with CLNP, where it is not necessarily clear which subnet the link-layer address refers to, is avoided.

When a packet arrives at a router that does not know the appropriate link-layer address for the next hop, the router looks into the options field and either extracts the appropriate link-layer address, or discovers that the link-layer address is not listed.

If the link layer address is listed, the router can cache it for future use. Thus, it does not have the processing overhead of looking into the options field on every packet. If the address is not listed, the router can send an error report to the source indicating as much, and subsequent packets from the source can give the link-layer address, or indicate that it is not known, or select a different route (for instance by using a different address). Since the link-layer address can be cached at routers and requested when needed, the host need not include it in every packet.

This approach can also be used by SIPP and CLNP. In this case, the list of link-layer addresses in the option is labeled with an address prefix rather than with an RSE, so that the router knows which link-layer address it should use.

9.8 Node-Level Source Route

By node level source route, we mean a source route that identifies individual nodes (hosts or routers) that the packet should visit. We are interested in two cases—one where the returned packet follows the (reverse) source route, and one where it does not.

9.8.1 CLNP

IP and CLNP both have source route options that operate at (and only at) the node level. In neither case is the receiving host able to, in practice, return the packet along the reverse path. This is not because the reverse path information is not in the header—it is. It is instead because the specifications do not indicate that hosts should reverse the source route.

9.8.2 SIPP

SIPP's route sequence can serve as a source route option. If SIPP addresses are non-extended, then node-level source routing in SIPP works similar to IP. That is, each address in the source route fully identifies a node, and the source route is advanced each time an identified node is reached.

The difference between SIPP and IP is that the host receiving the source route will reverse it and use it for return packets. If this routing of the return packets is not the desired behavior, or if the SIPP addresses are extended, then the route sequence of SIPP cannot be used for node-level source routing. Instead, multiple encapsulations of the SIPP header are required.

The problem with using extended addresses is as follows. Assume that a packet from host H to host Z should visit routers A, B, and C on the way. Assume that all addresses are extended, such that system x has address sequence x1:x0. If a route sequence were used to route the packet, the route sequence would be H0:H1:A1:A0:B1:B0:C1:C0:Z1:Z0, with A1 as the initial active address (that is, it would initially appear in the Dest Addr field, and A0 would be pointed to by the Next Addr field). The couplet A1:A0 would route the packet to router A, the next couplet B1:B0 would route the packet to router B, and so on.

When host Z reverses this sequence, however, it produces Z0:Z1:C0:C1:B0:B1:A0:A1:H1:H0, with the initial active address being C0. This is broken, because in the general case it is impossible to route a packet from host Z to "C0" (it must be routed to "C1" first).

Thus, to do node-level source routing whereby the return packet does not follow the reverse path, or where extended addresses are used, each node in the route must be encoded in a separate SIPP header, and the headers encapsulated in sequence.

Thus, for the example above, the first (outermost) header contains the route sequence H0:H1:C1:C0,

the next header the sequence H0:H1:B1:B0, and so on. The higher-layer protocol indicator (Payload Type in SIPP, Protocol in IP) indicates SIPP in each header except the last. Thus, when for instance router C received the first header, it decapsulates it and submits the next header to the SIPP layer. This causes the packet to be routed to host B, which does the same.

When host Z receives the packet, the single header has a route sequence of H0:H1:Z1:Z0. All of the source route information is lost. Host Z reverses this header to send subsequent packets to H.

9.8.3 SPip

An SPip host can do node-level source routes, both where the return packets follow the reverse path and where they do not. In order to do node-level source routing, however, a node-level RSID must be assigned to each node that the packet will visit. In all but one of the examples up to now (two-phase multicast), the node-level information is the EID itself. Assigning node-level RSEs is an extra configuration task not otherwise needed. Thus, the assignment of node-level RSEs is unlikely to happen in practice.

The technique of header encapsulation shown for SIPP can be used with SPip when a node-level source route is needed but node-level RSEs have not been assigned. In what follows, we assume that the node-level RSEs have been assigned.

In addition, the forwarding tables must be set up such that the node-level RSE is checked for by the router. For instance, say router a of Figure 9.1 is given a node-level RSE of value 0a so that it can be the target of a node-level source route. It would configure its (classical) forwarding table as follows. Note that these forwarding tables show the effect of the Last RSE bit by putting a '+' or a 'o' before each Destination entry. A '+' indicates that the Last RSE bit is set (that is, it is the last RSE in the route sequence).

Root RSETable for router a		
Destination	Next Hop	T-tag
o9S	Root RSETable	√
o3I	subnet I	√
o3K	subnet K	√
o3J	RSETable 1	√
+3J	Main EIDTable	√
o0a	Root RSETable	√
odefault	router c	√
+default	Main EIDTable	√

RSETable 1 for router a		
Destination	Next Hop	T-tag
∘0a	Root RSETable	√
default	Main EIDTable	√

Note that one difference between router a's Root RSETable here and the one shown in Table 9.2 is the addition of the +3J entry[12]. The 0a entry in RSETable 1 indicates that router a should go back to the Root RSETable with the next RSE (presumably the top-level RSE of the next node in the source route). The default entry indicates what the +3J entry would otherwise have indicated—that the Main EIDTable should be examined.

If there are other routers on the same LAN, the node-level entries for these routers would also appear in router a's RSETable 1.

Assume that router a receives a packet with a node-level source route. A packet is received with an active RSE of ∘3J. The next two RSEs are ∘0a and ∘1B. That is, the top-level RSE of the next node is 1B.

Router a accesses its Root RSETable with value ∘3J. This points to RSETable 1, which is accessed with ∘0a. This points back to the Root RSETable, which is accessed with ∘1B. This matches on the default entry, and the packet is routed to backbone B.

Now, assume that router a receives a packet destined for it, and with the node-level RSE as part of router a's address sequence. In this case, the packet is received with an active RSE of ∘3J, but the next and final RSE is +0a ('+' because this is the last RSE in the route sequence).

This time, router a accesses its Root RSETable with value ∘3J, which, as before, points to RSETable 1. RSETable 1 is accessed with +0a. This matches on the +default entry, which points to the Main EIDTable. This is accessed with the EID, which is router a's, and router a accepts the packet.

The above example illustrates the need for the Last RSE flag.

Note that a host could remove the node-level RSE from the address sequence when sending a packet to router a (as a destination). Since the EID identifies the node, the node-level RSE is not necessary. Since the node-level RSE always has level 0, the sending host knows that it can eliminate this RSE from the address sequence.

Next we examine the route sequence format for node-level source routes.

First consider the case where packets do not follow the reverse path. Assume the scenario from SIPP above, where a packet visits routers A, B, and C on the path from host H to host Z. Assume that the address sequences of hosts H and Z are 1H:9H:3H and 1Z:9Z:3Z respectively, and the the

[12] Actually, the 3J entry in the forwarding table of Table 9.2, had we been indicating the Last RSE bit status, would have been for +3J, since the expectation in that table is to retrieve the EID.

address sequence of router x is 1x:9x:3x:0x, where 0x is the node-level RSID.

The route sequence for the packet is:

3H:9H:1H : 1A:9A:3A:0A : 1B:9B:3B:0B : 1C:9C:3C:0C : 1Z:9Z:3Z

The Active RSE is 1A, and the Num Source RSE is 1H (the spaces are added in the route sequence just to make it easier to read, and have no special meaning). The packet follows the route sequence, visiting routers A, B, and C as a result.

When host Z receives the packet, it reverses only the address sequence of host H. The rest is discarded because the Num Source RSE did not include them.

Now assume the same path, but where the return packets must take the reverse path. The route sequence, covering two lines of text, is formatted as follows:

3H:9H:1H : 0A:3A:9A:1A : 0B:3B:9B:1B : 0C:3C:9C:1C :

1A:9A:3A:0A : 1B:9B:3B:0B : 1C:9C:3C:0C : 1Z:9Z:3Z

The Num Source RSE field is set to 15. That is, it includes everything in the first line. The Active RSE field is set to 16—the beginning of the second line.

When host Z reverses this packet, it reverses what is on the first line above, resulting in:

1C:9C:3C:0C : 1B:9B:3B:0B : 1A:9A:3A:0A : 1H:9H:3H

Host Z's own address is prepended to this, and the resulting packet is routed through routers C, B, and A—the reverse path.

9.9 Anycast Group Addressing

We are interested in both one- and two-phase anycast.

Neither CLNP nor SIPP have defined a separate anycast address space, though both could. While perhaps preferable, it is not necessary that anycast addresses come from a separate address space. Any unicast address can be declared an anycast address.

One-phase Anycast

For CLNP and SIPP, there is no difference in the format of the packet header between (single-phase) anycast and unicast. For SPip, anycast addresses do not require an RSE address sequence, so SPip anycast packets have only the source address sequence in the route sequence.

There is essentially no difference between the way routers handle anycast and unicast. In both

cases, the destination address/EID is examined and forwarded according to what is for all practical purposes a unicast forwarding table entry.

SPip anycast addresses have a scope field. If SIPP defines anycast, then it could define them similar to multicast, and therefore get scoping. The issue of how to set the scope is the same for anycast as it is for multicast.

Two-phase Anycast

With SPip, two-phase anycast is achieved by simply putting the relevant (unicast) RSIDs in the route sequence. For instance, if the desired service is that the packet goes unicast to a subscriber network and then anycast to one of the group's hosts, the destination address sequence contains the RSIDs up to and including the subscriber-level RSID. Thus, two-phase anycast destination information contains an anycast EID and an address sequence, and a source host formats the header exactly as it does a unicast address, with the exception that it may have to fill in the scope field of the anycast EID. If the target of the unicast phase is an individual host, then the host-level RSID is included in the address sequence.

With SIPP and CLNP, there are two ways to do a two-phase anycast. One is to define the low-order part of an otherwise unicast address as anycast. For instance, if the desired service is that of the last paragraph, unicast to a subscriber network and then anycast to one of the group's hosts, then the provider and subscriber IDs in the address could be as unicast, but the subnet ID and after could be replaced with an anycast ID.

The other way is using the source route mechanism, similar to how two-phase multicast is done. That is, the first address in the source route brings the packet unicast to a host, in the case of CLNP or SIPP, or a border router, in the case of SIPP cluster addresses. (The comments made above regarding the CLNP partial source route bug of course still apply.)

9.10 Summary

Table 9.4 summarizes the results of this Chapter. Those boxes marked with $\surd$ indicate that the capability can be supported by the protocol. Those boxes marked with $\approx$ indicate that the capability can be supported but with major limitations. Blank boxes mean that the capability cannot be supported by the protocol.

Note that one form of multicast (two-phase with source-tree) and full policy routes are shown in the "useful" section of Table 9.4, even though the basic function (multicast and policy routing respectively) are in the "required" section.

Table 9.4: Summary of Routing and Addressing Capabilities

Capability	CLNP	SIPP	SPip
required			
Big Enough Hierarchical Unicast Addressing			
Classical Forwarding Information	√	√	√
Hole-Punching Forwarding Information	√	√[6]	√
Multicast Group Addressing			
Shared-Tree	√	√	√
Source-Tree	√	√	√
Scoped	≈	√	√
Well-Known	≈	√	√
Two-Phase/Shared-Tree		√	√
Mobility		√	√
Domain-Level Policy Route			
Provider Selection	√	√	√
Host Auto-Address Assignment	√	√[1]	√
useful			
Two-Phase/Source-Tree Multicast			√
Full Policy Route		√	√
Type-of-Service Field	√		√
Embedded Link-Layer Address	≈		
Node-Level Source Route			
With Reversing			√[2]
Without Reversing	√[3]	√[4]	√[2]
Anycast Group Addressing	√	√	√
Anycast/Two-Phase	√[5]	√	√

[1] Not with non-extended addresses

[2] Requires extra router configuration

[3] Strict source route only, unless encapsulation technique used

[4] Not with extended addressing, unless encapsulation technique used

[5] Using hierarchical anycast address only

[6] With minor limitations

Chapter 10

Costs of SPip, SIPP, and CLNP

In this chapter, we discuss the relative costs of SPip, SIPP, and CLNP, in terms of processing cost, header size, address assignment complexity, and control protocol complexity.

10.1 Processing Cost

Processing cost is difficult to analyze comprehensively and in detail. Processing cost differs from implementation to implementation, and within a given implementation, processing cost is influenced by many factors—the mix of traffic (influences caching strategies), forwarding table size (influences forwarding table lookup time or CAM (Content Addressable Memory) size), and packet format (options are slower).

There are two basic elements of processing cost; processing speed and hardware complexity. These two elements can be traded off—reductions in processing speed generally require more complex hardware. None the less, one protocol can have lower overall cost than another, in terms of speed or hardware complexity or some combination of the two, as shown in Figure 10.1.

In spite of the difficulty of precise and comprehensive analysis of processing cost, we can make some useful generalizations. First, we can place a lower bound on processing speed. That is, a packet cannot be processed faster than the time it takes for the bits relevant to the forwarding decision to arrive on the wire.

This lower bound is of practical consequence, as it is desirable (and often achievable) for a router to transmit a received packet as soon as possible, thus minimizing switching latency. With cut-through switching (that is, starting the transmission of a packet before the entire packet is received [105, 4, 1]), the lowest possible switching latency is the time it takes for the relevant bits to arrive on the wire.

The second generalization we can make is that processing cost is directly related (as a first ap-

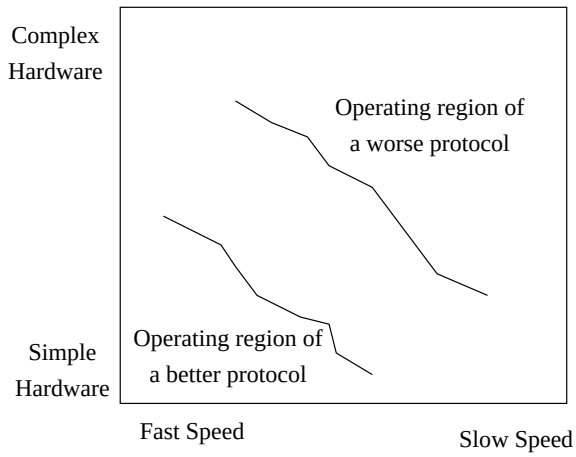


Figure 10.1: Processing Speed Versus Hardware Complexity

proximation) to the number of memory accesses (reads and writes) required, by a traditional CPU/memory hardware architecture, to process a packet. This analysis includes both accesses to the packet header and to other memory, such as a forwarding table. The following arguments to back up this claim are not at all rigorous, but defensible none the less.

It is fairly well established that the most expensive aspect of packet processing is memory transfers [24, 23, 77, 91]. While memory transfer overhead is worst in the case of copying whole packets, the fact is that any memory access is expensive relative to other processor operations, especially when the access is across a shared bus.

In the case of packet processing, it is possible to reduce the cost of accesses with fast, local, dual-ported memory. This, however, is increasing the complexity of hardware. So, either way, there is an increase in processing "cost".

In general the cost of hardware implementation is roughly related to the number of memory accesses. For instance, if a packet header is large, and most of the bits of the header are relevant to the forwarding process, then this can be reflected either as a lot of accesses or as a "wide" hardware structure. If a given field read from a header requires a lot of processing, this can result in a lot of accesses to memory, for instance to do a forwarding table lookup, or in a "deep" hardware structure. For instance, it is shown in [70] that more hardware is required for a best-match with default forwarding table search than for that of an any-match search, for the same size address. That is, the width of the hardware structure (address size) is the same while the depth is greater.

A final argument supporting the count of memory accesses to compare performance derives from the fact that we are doing comparative performance measurements rather than absolute. We can assume that every read has associated with it a small amount of processing (shift, mask, compare, etc.). It would make no sense to read something without acting on it in some way. And, internet

packet processing does not require heavy-duty processing operations, such as a divide operation.

While some memory accesses will have more processing associated with them than others, we can assume as a first approximation that on average the extra processing is the same for all memory accesses. Thus, processing is just a small (probably less than two) multiplier on the number of memory accesses, which can be ignored because roughly the same multiplier applies to the memory accesses of each protocol.

Based on the above observations, we settle here for using a simple count of memory accesses to approximate processing cost for the three protocols. We assume a processor with a native word length of 64 bits. Thus, up to 64 bits can be read in a single read. However, we consider reading or writing any single field to be a single read or write, even though multiple such fields might be otherwise accessible with a single 64-bit read.

Even with the simplification of counting only memory accesses, the analysis of processing cost is still complicated. The number of accesses depends on a number of variables, for instance whether the forwarding cache is hit or missed, and for SIPP and CLNP, whether the forwarding table is large or small.

Since a cache lookup and best-match forwarding table lookup are common to a number of routing cabilities, we first analyze them separately, followed by the analysis of each routing cability (under cache miss conditions).

10.1.1 Cache Hits

The caching of forwarding table lookup results is a common practice among routers [93, 47]. Typical cache size is multiple hundreds of entries. When a packet is received, the cache is indexed with the destination address, using a CAM or a hash. If there is a cache hit, the forwarding information is extracted and the packet forwarded. If there is a miss, a forwarding table lookup is done, and the result is written into the cache.

In the following three subsections, the caching strategy for each protocol is described, along with its cost.

CLNP

At a minimum, CLNP indexes the cache with the destination NSAP address. This strategy allows CLNP to cache unicast and shared-tree multicast forwarding table lookups, but not source-tree multicast lookups. Source-tree multicast, however, is in the future likely to be a significant and perhaps even dominant traffic type. Thus, we should consider two caching strategies for CLNP—one where only the destination address is used, and one where both source and destination address are used.

For the destination address only caching strategy, we assume the following algorithm. The low order part of the destination NSAP address is read and used to compute the hash value (the low order part is most effective for this because it differs most between different addresses). If this results in a hit, then the rest of the NSAP address is read and compared against the cache entry.

To read the low order part of the destination NSAP address the router must first locate it by reading the Destination Address Length Indicator field. So, the total number of memory accesses are as follows:

1.	Read Destination Address Length Indicator	1 access
2.	Read low-order part of Dest Addr	1 access
3.	Read low-order part of address in cache entry	1 access
4.	Read high-order parts of Dest Addr	2 accesses
5.	Read high-order parts of address in cache	2 accesses
	Totals:	7 accesses for hit
		3 for miss (min)

Of course, there are other memory accesses as well, such as retrieving the pointer to the FIB (Forwarding Information Base), accessing the FIB, and so on. But these are common to all of the protocols and so need not be counted for comparison.

The fastest possible destination-only cache lookup time is 30 bytes at wire speed—10 bytes for the beginning of the header, and 20 bytes for the destination address[1].

The caching algorithm for combined source/destination caching requires a decision point—where the router determines if the lookup requires the source address. This is necessary because the source address should not be part of the compare if it was not used for the original lookup. This avoids caching per source/destination pair (versus just per destination) for unicast and shared-tree multicast packets.

Thus, there are two caches, one for dest-only and one for source/dest combined. The dest-only cache is always accessed first to determine if the combined source/dest cache should be accessed.

Thus, the initial part of the cache lookup is:

1.	Read Destination Address Length Indicator	1 access
2.	Read low-order part of Dest Addr	1 access
3.	Read "which-cache" indicator	1 access

The which-cache indicator can be the same word as the low-order part of the address, but with a special value. If the which-cache indicator indicates dest-only cache,[2] then the remainder of the lookup is as described above, and costs 7 accesses. Otherwise, the following additional steps are

[1] The destination address can be smaller, but in practice it is almost always 20 bytes.

[2] Note that it is not adequate to only determine that the destination address is a multicast address to decide to use the source/dest cache. This is because the multicast address might be for a shared-tree, in which the dest-only cache suffices.

executed:

4.	Read Source Address Length Indicator	1 access
5.	Read low-order part of Source Addr	1 access
6.	Read low-order part of source address in s/d cache entry	1 access
7.	Read low-order part of dest address in s/d cache	1 access
8.	Read high-order parts of Dest Addr	2 accesses
9.	Read high-order parts of Source Addr	2 accesses
10.	Read high-order parts of addresses in s/d cache	4 accesses
	Totals:	15 accesses for hit
		6 for miss (min)

The fastest possible lookup for combined source/destination caching is 51 bytes at wire speed.

Note that these caching algorithms do not allow caching of packets that have ToS or source routing. Of the "required" capabilities, this eliminates the ability of caching to be used with provider selection, which uses the ToS facility. If the Length Indicator field indicates the presence of ToS or source routing, then the cache is not searched.

The cache performance of CLNP is summarized in the following table:

CLNP Cache Performance			
Cache Method	Cache Hit (accesses)	Cache Miss (accesses)	Best Possible (bytes at wire speed)
Destination-only caching	7	3	30
Source/Dest caching, source not required	7	3	30
Source/Dest caching, source required	15	6	51

SIPP

SIPP caching is similar to CLNP caching in that it can either just cache on the Dest Addr, in which case the cache can not be used with source-tree multicast, or it can cache on source address information as well. The strategy in the latter case would be the same as CLNP—first access a dest-only cache using the Dest Addr field of the packet, and then determine if the source information should be examined. As with CLNP above, we consider both cases.

Destination-only caching requires only one read of the Dest Addr field, and one read to the memory location holding the cache—two accesses total. A cache miss also costs two accesses.

Note, however, that the result of the cache lookup may be to advance the route sequence. In this case, the advance is done, and the cache lookup is done all over again using the new Dest Addr.

The cost of advancing the route header in SIPP is as follows:

1.	Read Payload Type field	1 access
2.	Read and write Next Addr field	2 accesses
3.	Read Num Addrs field	1 access
4.	Read and write Addr in route sequence	2 accesses
5.	Write Dest Addr field	1 access
	Total:	7 accesses

This first read, to the Payload Type field, is to insure that the Routing Header is the one immediately following the SIPP header.

So, the total cost of destination-only caching with SIPP is 2 accesses with no route sequence advance, 11 accesses with one route sequence advance, 19 accesses with two route sequence advances, and so on. Even one route sequence advance, however, is the minority case, and two route sequence advances should almost never happen.

The same strategy for CLNP described above can be used for SIPP—that is, the destination address is examined first to determine which cache, the dest-only cache or the source/dest cache, must be used. For the source/dest cache, the extended address must be examined in order of high-order part first[3].

To determine which cache is used requires two memory accesses—a read of the destination address in the SIPP header, and a read of the which-cache indicator. If source/dest caching is indicated, then at least 3 more memory accesses are required—read the source address of the packet, read the source address of the cache, and read the destination address of the cache—for a total of 5 accesses.

In almost all cases, this suffices for a cache hit. It does not in the case where 1) the host is using extended addressing, 2) the host is transmitting packets to the same group using multiple source extended addresses, and 3) the router in question is on multiple of the resulting trees. If this is the case, then the source/dest cache will indicate that more of the source address must be examined. This requires three more memory accesses—one to insure that the Source Route is in the expected location, one to read the upper address of the extended address, and one to read the corresponding address in the source/dest cache.

Given the rarity of this situation, we assume that the source/dest cache requires a total of 5 accesses. Note that if the source/dest cache is used, then it is not necessary to advance the route sequence, since the route sequence is not advanced with multicast (see Section 9.2.2).

The fastest possible forwarding speed of cached SIPP again depends on whether or not the higher addresses of the source extended address must be examined. Again, we assume the common case where it is not.

[3]I thank Steve Deering for pointing this out to me. Initially I thought that the Source Addr field alone was sufficient, although this error did not change the analysis results for the common case.

Thus, the SIPP packet can commonly be forwarded as soon as the destination address is received—24 bytes at wire speed. If a route sequence advance takes place, then the fastest forwarding speed depends on where in the route sequence the subsequent address lies. If we assume that an extended address has two addresses, then the second address of the routing header will be read when the route is advanced. This address can be read after 48 bytes have arrived.

The following table summarizes SIPP's caching performance:

SIPP Cache Performance			
Cache Method	Cache Hit (accesses)	Cache Miss (accesses)	Best Possible (bytes at wire speed)
Destination-only caching	2	2	24
with one route sequence advance	11	11	≈48
with two route sequence advances	19	19	≈56
Source/Dest caching, source not required	3	2	24
Source/Dest caching, source required	5	2 or 5	24

SPip

SPip uses the flow ID combined with the source address for every cache lookup. This is possible because SPip hosts assign a unique flow ID for every Source EID, Dest EID, and route sequence combination.[4]

As part of its cache management, however, SPip routers also record the hop count field in the cache, and check subsequent packets against it. If a cache hits, but the hop count of the received packet does not match that of previous cache hits, the cache is erased and the packet is routed on full routing information.

The purpose of monitoring hop count is to discover when caches have become invalid due to routing changes. For instance, consider the following case:

A packet is routed from host X to host Y, with a route sequence that indicates that the packet should visit router B on the way. Assume that the resulting path takes the packet through router A on the way from host X to router B:

$$X \longrightarrow A \longrightarrow B \longrightarrow Y$$

Assume that one or more packets have taken this path, and that therefore the packet is cached based on flow ID and source EID. Subsequent to that, the path from B to Y breaks such that the new path from B to Y takes the packet through router A. If router A does not check the hop count field, it will simply access the cache using flow ID and source EID and route the packet

[4] It is not clear that such an approach is advantageous with SIPP. First, SIPP's regular caching is already fast. Second, a flow-based cache hit does not eliminate the cost of advancing the source route if that is necessary. This is the most expensive part of SIPP caching.

towards B, resulting in a loop.

If, however, router A checks the hop count against the previous value, it will find that the hop count is different, flush the cache, read the route sequence, and correctly route the packet towards host Y. This method will catch and break all loops.

The memory accesses for SPip caching are as follows:

1.	Read flow ID field in packet	1 access
2.	Read Source EID field in packet	1 access
3.	Read flow ID from cache entry	1 access
4.	Read source EID from cache entry	1 access
5.	Read Hop Count field in packet	1 access
6.	Read Hop Count field from cache entry	1 access
7.	Read active RSE from cache entry	1 access
7.	Write Active RSE field in packet	1 access
	Total:	8 accesses

An SPip cache hit requires six reads—two for the flow ID and Source Addr in the packet, two to read the corresponding fields in the cache entry, one to read the hop count field, and one to compare it against the one stored in the cache. SPip cache misses cost four reads.

Once a cache hit takes place, however, the SPip packet is not yet necessarily fully processed. If the full forwarding process (the first one—when the cache was created) involved advancing the route sequence, then the router must also do so after the cache hit. The route sequence does not have to parsed, however, to do this. Rather, the router can store the originally calculated value of the Active RSE field and write that value into the transmitted packet. This costs two memory accesses—one read and one write.

Thus, the total cost of a cache hit with SPip is 8 memory accesses. This cache hit is valid for every type of forwarding—unicast, multicast, source routing, and so on.

Since the cache hit is completed after the Source EID field is read, the fastest possible forwarding time for SPip is 16 bytes at wire speed[5].

The cache performance of SPip is summarized as follows:

SPip Cache Performance			
Cache Method	Cache Hit (accesses)	Cache Miss (accesses)	Best Possible (bytes at wire speed)
(SPip has just one cache method)	8	4	16

[5]Note that the Active RSE field does not need to be read before the packet can be forwarded. It does, however, need to be written once it arrives.

10.1.2 Forwarding Table Lookup

Two forwarding table lookup styles are described in this section, one for CLNP and SIPP, and a different one for SPip.

CLNP and SIPP

It is hard to characterize the performance of CLNP and SIPP lookups in terms of number of reads, because the number required depends on a several things, particularly the number of entries in the forwarding table, and on the fanout value of the search tree, which impacts how much memory is needed to store the forwarding information (more memory, fewer reads).

A fairly common form of search tree has a binary fanout—that is, each decision point divides the possible number of outcomes roughly in half. There are a number of such schemes, such as Patricia and the binary trie [65]. For this analysis, we assume a perfect binary search tree—that is, one for which the number of search iterations is $\log_2(n)$, where n is the number of entries in the forwarding table. (That is, each iteration perfectly divides the number of remaining possible outcomes in half.)

The search tree works as follows. The tree is a collection of data structures, each with a compare value and two pointers. Each pointer points either to another data structure, one to the "right" and one to the "left", or to nothing if the data structure is a leaf in the tree (as indicated by the compare value).

The tree search starts with the root data structure. For each iteration of the search, the compare value is retrieved from the data structure and compared against the address being searched. This compare can be a greater/less than compare, as with a binary trie, or a 0/1 mask compare, as with Patricia. Either way, the result is to retrieve either the right pointer or the left pointer, and do another iteration.

When a leaf is reached, a full compare (possibly involving a mask) is done against the address to determine either if a match has occurred, or if the lookup has failed. The size of the pointer is the native word size of the machine, and so is accessed with a single read. The size of the compare value, however, is related to the size of the address. For SIPP, the compare value is accessed in one read (assuming a 64-bit machine), as shown in the following C code segment for a patricia tree:

```
for (;1;) {
    if (currentPointer->value && address == 0)
       currentPointer = currentPointer->right;
    else
```

```
        currentPointer = currentPointer->left;
    if (currentPointer->value == 0)
        break;
}
```

The total cost of this iteration is 2 reads, one for the compare value and one for the pointer.

For CLNP, the compare value is three 64-bit words, but this does not mean that every compare requires three reads. This is because the comparing (whether it be masks or greater/less than) works left-to-right on the address. For instance, with Patricia, only one bit is compared at a time, and the current bit is always to the right of the previous bit. Thus, even though the CLNP address is 3 64-bit words long, only one of them need be compared at any given iteration. Thus, each iteration must indicate which 64-bit word of the address should be examined:

```
for (;1;) {
    address = addressArray[currentPointer->addressWord] ;
    if (currentPointer->value && address == 0)
        currentPointer = currentPointer->right;
    else
        currentPointer = currentPointer->left;
    if (currentPointer->value == 0)
        break;
}
```

Thus, each iteration of a CLNP search costs three reads.

For SIPP, the final compare costs one read, while for CLNP it costs three reads.

Thus, the total cost of the lookup for SIPP is:

$$2\log_2(n) + 1 \tag{10.1}$$

and for CLNP, it is:

$$3\log_2(n) + 3 \tag{10.2}$$

where n is the number of entries in the forwarding table.

For our analysis, we assume two cases, a large forwarding table (8192 entries), and a small forwarding table (16 entries). Based on the above equations, the cost of the forwarding table operations for SIPP and CLNP are:

Protocol	Small Forwarding Table	Large Forwarding Table
SIPP	9	27
CLNP	15	42

SPip

The above search tree style does not apply to SPip because SPip does not have bit-wise maskable addresses. Instead, SPip has a series of flat identifiers (RSEs).

Typical forwarding operations for an SPip router involve one or two RSEs. A lookup of two RSEs happens when either a router is hole punching, or when the packet passes from one hierarchy level to another. Occasionally three or more RSEs may be examined in a single forwarding operation. For the purpose of evaluation, we assume that two RSEs are examined.

The cost of examining two RSEs is calculated as follows:

1.	Read Active RSE field	1 access
2.	Read active RSE	1 access
3.	Read value from RSE Table	1 or 2 accesses
4.	Read next RSE (or EID)	1 access
3.	Read value from RSE Table	1 or 2 accesses
	Total:	6 accesses

Reading a value from an RSE Table may sometimes require two accesses. This happens when the read entry indicates that a default entry should be accessed. For our analysis, we assume that an RSE Table read requires on average 1.5 accesses. Thus, the total cost of a forwarding table lookup with SPip is 6 accesses.

10.1.3 Hierarchical Unicast Addressing

In practice, the cost of a hierarchical unicast lookup is more expensive with hole-punching than without. With SPip, hole-punching results in an additional RSE being examined. With SIPP or CLNP, the router will look deeper in the address to dig out the relevant forwarding information.

Our analysis, however, is too rough to distinguish between the two. Therefore, we ignore the distinction.

CLNP

The cost of hierarchical unicast addressing in CLNP is equal to the cost of a cache miss plus the cost of reading the Destination Address plus the cost of the forwarding table lookup. The cache miss requires three accesses. The low order part of the Destination Address is already read when the cache is missed, so two more accesses are required to read the full Destination Address. Finally, the forwarding table lookup takes 15 or 42 accesses (for small and large forwarding table respectively).

Thus, the total cost of a hierarchical unicast address forwarding with CLNP is 20 or 47 accesses. Note that the cost here is dominated by the forwarding table lookup.

The fastest possible processing time is the same as that of the destination-only cache (because a decision is reached after the destination address is read)—30 bytes at wire speed.

SIPP

For SIPP, we consider two cases: one where the route sequence is not advanced, and one where it is advanced once.

With no route sequence advance, the cost is 2 accesses for the cache miss (at which point the Destination Address has been read), and 9 or 27 accesses for the forwarding table lookup, for a total of 11 or 29 accesses. Again the cost is dominated by the forwarding table lookup.

If the route sequence is advanced, then we can assume that the first lookup resulted in a cache hit (2 accesses), since the address in the first lookup is one that the router recognizes as its own. The cost of advancing the route sequence is 7 accesses. The cost of the forwarding table lookup is 9 or 27 accesses, resulting in a total cost of 18 or 36 accesses.

The fastest possible forwarding speed is the same as that of the destination-only cache hit—24 bytes for no route sequence advance, and 48 bytes for one route sequence advance.

SPip

The cost of a hierarchical unicast address lookup for SPip is equal to the cost of a cache miss (4 accesses), plus the cost of the forwarding table(s) lookup (6 accesses), plus the cost of updating the Active RSE field, which is 2 accesses (one read, from the FIB, and, often, one write). The total cost is therefore 12 accesses.

The fastest possible lookup time for a hierarchical unicast address depends on how deep the relevant RSE is in the route sequence. If we assume four-level addresses—host, subnet, subscriber, and provider—then there are six RSEs, and the last RSE is the subnet ID for the destination. Assuming that the subnet ID must be read, the fastest possible lookup time is 52 bytes at wire

speed.

10.1.4 Single-phase Shared-tree Multicast

For CLNP and SIPP, we assume that the multicast address has been inserted into the cache at multicast tree setup time and therefore results in a cache hit even if the multicast packet has not yet been received.[6] This assumption applies to all of the multicast examples.

Thus, for shared-tree multicast, the forwarding cost is the same as for a (source/dest) cache hit, which is 7 accesses and 30 bytes for CLNP, 3 accesses and 24 bytes for SIPP.

For SPip, the forwarding cost is a cache miss (4 accesses) plus a lookup on the Destination EID field. This lookup requires only three memory accesses, one to read the Active RSE field (which will be value 0), one to read the Dest EID field, and one to read the EID Table. Thus, the total cost is 7 accesses.

The fastest possible lookup is the time it takes for the Active RSE field to arrive—28 bytes.

10.1.5 Single-phase Source-tree Multicast

CLNP

For source-tree multicast, the cache hit (7 accesses) on the destination address indicates that the source address must be examined. The cost of examining the source address is 3 accesses to read the source address plus the cost of the forwarding table lookup (15 or 42 accesses). Thus, the total cost of a source-tree multicast lookup is 25 or 52 accesses.

The fastest possible forwarding time is the time it takes to read the source address—51 bytes at wire speed.

SIPP

With SIPP, the cache hit (3 accesses) on the destination address indicates that the source address must be examined. To examine the source address, the router must first decide if the source address is extended or not. This requires examination of the Payload Type field (1 access).

If the address is not extended, then a forwarding table lookup is done on the Source Address—10 or 28 accesses, one to read the source address and 9 or 27 to do the lookup. Thus, the total cost for a simple address is 14 or 31 accesses.

The fastest possible time is the time it takes to read the Dest EID—24 bytes at wire speed.

[6] Note that some implementations of some multicast algorithms, such as MOSPF, in order to save memory, may do extensive processing to calculate a multicast tree when the first multicast packet of a group is received.

If the address is extended, then the router must read the Next Addr field (1 access) to determine the location of the high-order part of the source address. For this analysis, we assume that the extended address sequence has two addresses. Next, the high-order part of the source address sequence is read (1 access), and a forwarding table lookup is performed.

This lookup may result in an answer, or may indicate that the low-order part of the source address sequence should be examined. Note, however, that each forwarding table lookup should cost roughly half that of a full forwarding table lookup, because only half of the address is examined each time. Given this, plus the fact that the first lookup may suffice, we assume that the cost of searching both addresses is equivalent to the cost of a full search—9 or 27 accesses. Add to this the cost of reading the source address, and the total cost for extended addresses is 16 or 34 accesses. The fastest possible speed is the time it takes to receive the first address of the route sequence—40 bytes at wire speed.

SPip

To do source-tree multicast, SPip examines one or more RSEs before examining the Dest EID. Unlike unicast forwarding, it may have to parse through the entire source address before examining the Dest EID. Thus, we can assume that 4 iterations on the RSE lookup are done, rather than 2 as we assumed for the unicast hierarchical lookup. This results in 5 additional accesses over the unicast case, or 17 accesses total.

Like the unicast case, the fastest possible forwarding time depends on the number of RSEs in the address sequence. We can assume here that it is the same as for the unicast case—52 bytes.

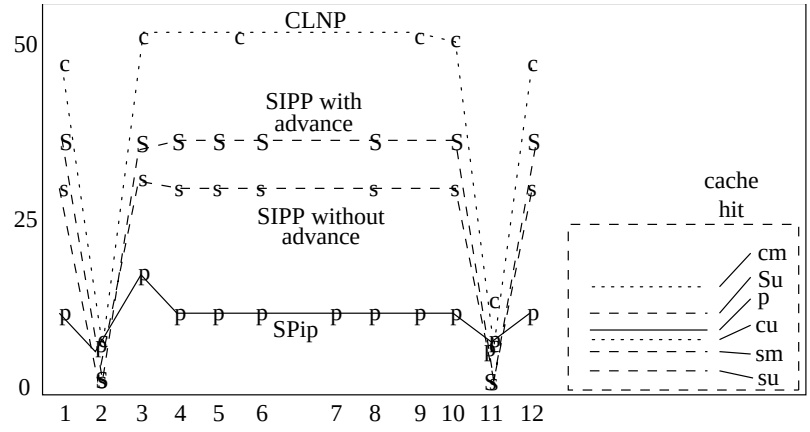
10.1.6 Others

The other capabilities can be analyzed along the lines of the previous sections. Rather than go through each one in detail, we present the results of the cost analysis in Table 10.1.

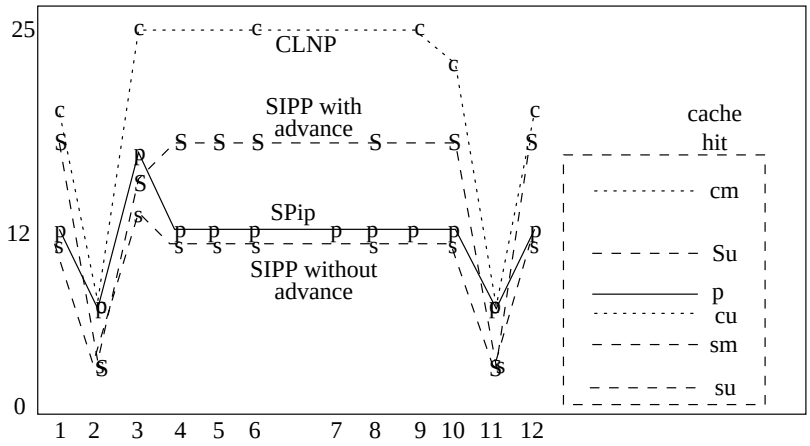
10.1.7 Discussion of Processing Costs

Figure 10.2 summarizes in graphical form the processing costs for the three protocols. Figure 10.2 has three graphs, for 1) the number of accesses with a large forwarding table, 2) the number of accesses with a small forwarding table, and 3) the best possible forwarding time, in number of bytes at wire speed. Each graph also shows the performance of a cache hit. Keep in mind that most lookups will result in a cache hit.

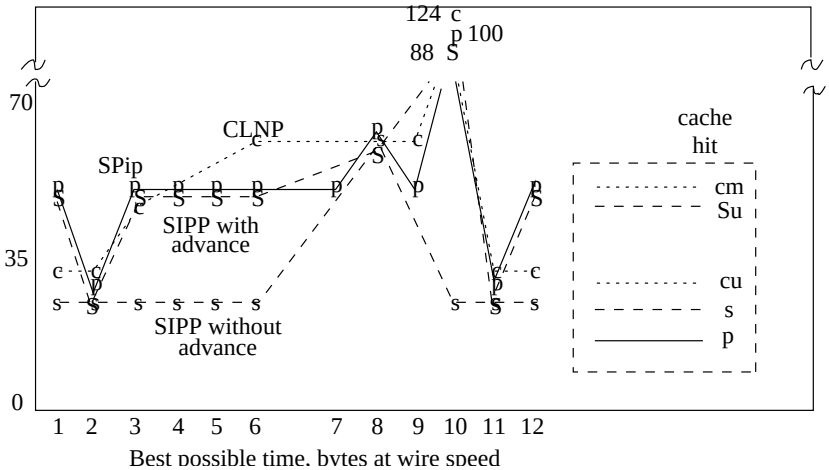
Each graph gives four traces—one for CLNP (dotted line with the letter 'c' for data points), one for SPip (solid line with the letter 'p' for data points), and two for SIPP (dashed lines). One of the SIPP lines is for the case where the route sequence is not advanced (lower-case 's' for data



Number of accesses, large forwarding table



Number of accesses, small forwarding table



Best possible time, bytes at wire speed

Figure 10.2: Summary of Forwarding Costs

Table 10.1: Summary of Forwarding Costs

Capability	CLNP			SIPP			SPip	
	small	large	best	small[1]	large[2]	best[3]	both[4]	best
required								
1. Hierarchical Unicast	20	47	30	11/18[5]	29/36	24/48	12	52
2. Shared-tree Multicast	7	7[6]	30	3	3[6]	24	7	28
3. Source-tree Multicast	25	52	51	14/16[7]	31/34	24/40	17	52
4. Two-Phase/Shared-tree	–[8]	–	–	11/18–3[9]	29/36–3	24/48–24	12/7[10]	52/28
5. Mobility	–	–	–	11/18[5]	29/36	24/48	12	52
6. Provider Select	25	52	61	11/18[5]	29/36	24/48	12	52
useful								
7. Two-Phase/Source-tree	–	–	–	–	–	–	12/17[10]	52/64
8. Policy Route[11]	–	–	–	11/18[5]	29/36	56	12	64
9. Type-of-Service Field	25	52	61	–	–	–	12	52
10. Source Route [12]	23/31[13]	50/58	124	11/18[5]	29/36	24/88	12	100
11. Anycast	7	7[6]	30	3	3[6]	24	7	28
12. Two-Phase Anycast	20	47	30	11/18–3[9]	29/36–3	24/48–24	12/7[10]	52/28

[1] Number of memory accesses for small forwarding table (16 entries)

[2] Number of memory accesses for large forwarding table (8192 entries)

[3] Fastest possible forwarding time (bytes at wire speed)

[4] Number of memory accesses (SPip has no dependency on forwarding table size)

[5] X/Y where X = no routing sequence advance, Y = 1 advance

[6] Does not depend on forwarding table size

[7] X/Y where X = simple address, Y = extended address

[8] Capability does not exist

[9] X/Y–Z where X/Y = unicast phase no-advance/advance, Z = group address phase

[10] X/Y where X = unicast phase, Z = group address phase

[11] Assume 3 backbones in policy route

[12] Assume 3 routers in source route, no reversing (extended addresses for SIPP)

[13] X/Y where X = no source route advance, Y = 1 advance

points), and the other is for the case where it is (upper-case 'S' for data points).

Each data point gives the performance for the associated capability. The capabilities are numbered 1 through 12, corresponding to the numbers of the cabilities in Table 10.1. The first 6 cabilities are "required", and the latter six are "useful". Data points are missing where the protocol cannot do the capability. Thus, for instance, the CLNP traces have fewer data points than the SPip traces.

Note that strictly speaking there should not be lines connecting the data points, as the data points are discreet and have no relation to each other—at least not in the sense of a continuous measurement. The lines, however, make it easier to group the data points for a given protocol,

and also give a stronger visual sense of the performance of each protocol as a whole.

All of the cache hit data shows the performance for combined source/dest caching. Specifically, the data is for:

'cm' CLNP with a hit on the combined source and destination address (the 'm' in 'cm' stands for multicast).

'cu' CLNP with a destination address only hit.

'p' SPip under all conditions.

'S' SIPP with two destination address only hits (that is, the route sequence is advanced once).

'sm' SIPP with a hit on the combined source and destination address.

'su' SIPP with a destination address only hit. (Note that these last two are combined, labeled 's', in the best possible time graph because they have the same performance).

The major insights shown by the graphs of Figure 10.2 are as follows.

The first two graphs show that CLNP and SIPP are heavily dependent on forwarding table size for their performance, while SPip in independent of forwarding table size. SPip performance is significantly better than CLNP or SIPP when the forwarding table is large. When the forwarding table is small, SPip performs better than CLNP, performs roughly the same as SIPP with no route sequence advance, and better than SIPP with an advance.

In terms of number of memory accesses, CLNP performs significantly worse than SIPP or SPip across the board. In terms of best possible forwarding times, CLNP and SPip are comparable, with SIPP performing generally better for the no route sequence advance case.

An interesting point regarding SPip is that, in terms of the number of accesses, its non-cache lookup performance is in most cases not that much worse than its cache-hit performance. In terms of best possible time, however, cache hit lookups perform significantly better than cache miss lookups.

10.2 Header Size

Calculation of header size for the three protocols is straightforward. Table 10.2 shows the header sizes for the three protocols. We assume two sizes of SIPP addresses ("large" and "small" SIPP).

Figure 10.3 gives the results of Table 10.2 in graph form. It shows that CLNP, large SIPP, and SPip have generally similar header size, with CLNP being slightly worse than SPip, and SPip being generally worse, particularly for the common unicast and multicast cases (capabilities 1 through 3), than large SIPP. Small SIPP has a significantly smaller header, particularly for plain

Table 10.2: Summary of Header Size

Capability	CLNP [1]	Small SIPP [2]	Large SIPP [3]	SPip [4]
required				
1. Hierarchical Unicast	57	24	48	56
2. Shared-tree Multicast	57	24	40	56
3. Source-tree Multicast	57	24	40	56
4. Two-Phase/Shared-tree	–	40	56	56
5. Mobility	–	40	56	56
6. Provider Select	61	48	64	56
useful				
7. Two-Phase/Source-tree	–	–	–	64
8. Policy Route[5]	–	56	72	64
9. Type-of-Service Field	61	–	–	56
10. Source Route [6]	123	56	96	104
11. Anycast	57	24	40	40
12. Two-Phase Anycast	57	40	56	56

[1] The CLNP NSAP address size is always assumed to be 20 bytes

[2] "Small" SIPP assumes simple addresses

[3] "Large" SIPP assumes 2-address address sequence

[4] Three RSEs per address sequence

[5] Assume 3 backbones in policy route

[6] Assume 3 routers in source route, no reversing

hierarchical unicast addressing (capability number 1), for which it is half the size of large SIPP and about 40% the size of CLNP or SPip. For source- or shared-tree multicast, small SIPP is 60% the size of large SIPP, and 40% that of SPip or CLNP.

Given the small header size of small SIPP, we must ask if the advantage of the smaller header is worth the cost—namely, that small SIPP (SIPP without extended addresses) does not have "serverless" auto-address configuration). Thus, auto-address configuration is significantly more complicated with small SIPP than with large SIPP or CLNP or SPip.

Header size has two major effects on protocol performance. First, a large header can increase the switching latency of a packet. Second, a large header takes up bandwidth and reduces throughput on a link.

Concerning latency, a large header can increase latency in two ways, depending on whether or not cut-through switching is being used. If cut-through switching is being used, then a large header per se does not increase latency—it depends on where the information relevant to forwarding is. For instance, the SPip packet for unicast hierarchical addressing has a large header (56 bytes),

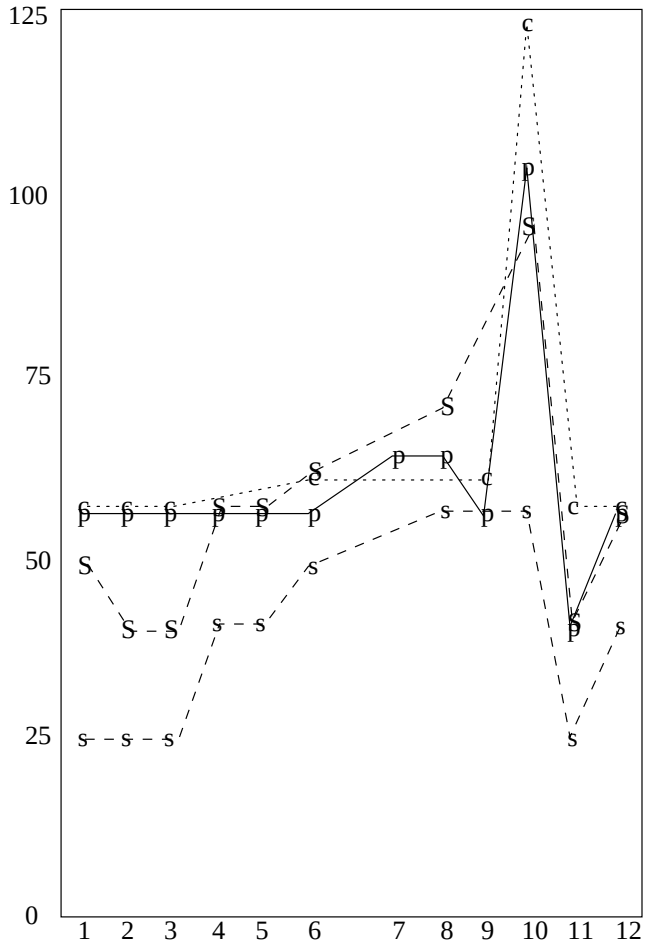


Figure 10.3: Summary of Header Size

but in the case of a cache hit, the switching latency is only 16 bytes (see the cache hit section of the bottom graph of Figure 10.2). Without a cache hit, however, the latency is indeed higher (52 bytes). We refer to a header where the relevant forwarding information is at the front as a "shallow" header. Likewise, a header where the relevant forwarding information is at the back is called a "deep" header. Thus, SPip with a cache hit is shallow, and SPip with a cache miss is deep.

If cut-through switching is not being used, then a large packet header increases latency simply by making the packet larger. Even if cut-through switching is being used, a large header can increase latency for the case where a packet is queued up behind other packets. If the data portion of the packet (the portion behind the internet header) is large, then even a large internet header does not make much difference. For instance, if the data packet is 4000 bytes, the large (57-byte) CLNP header contributes to less than 2% of the total packet size, and so is irrelevant. If the data portion is small, on the other hand, then header size is a significant factor.

Thus, we are concerned with two things:

- Switching latency with cut-through (deep or shallow header), and
- Percentage of packet size contributed by the header (large or small header).

As shown in Section 10.1.7, SIPP with no route sequence advance has significantly lower cut-through latency (fastest possible forwarding time) than SIPP with a route sequence advance, CLNP, or SPip. Small SIPP has no route sequence and so never requires a route sequence advance. Thus, small SIPP best satisfies both of the above concerns.

There are environments where deep and large headers are a problem, and environments where they are not. In general, as bandwidth and distance increase, deep or large headers become less of a problem.

Deep headers becomes less of a problem simply because faster bandwidth leads to lower latency. For instance, the latency due to CLNP's 57 byte header on Ethernet (10 Mbps) is 0.046ms. A typical packetization rate for VAT voice encoding (VAT is a conferencing application running over the IP internet) is 20ms—that is, the voice packetization itself adds 20ms latency [13]. Thus, the large CLNP header contributes to only 0.2% of the total latency. Even when switched through 10 routers, the cumulative latency due to the header is just 2% of the packetization latency.

Large headers become less of a problem at high speeds for two reasons. First, in the non cut-through case, the latency of a larger header is less at high speeds. Second, applications can generally produce larger packets when using high speed links, precisely because the latency is lower with higher speed. Thus a large header contributes less overhead at higher speeds.

For instance, interactive voice or video applications can tolerate only a small amount of end-to-end latency (a couple of hundred milliseconds). As link speed increases, however, packets can be larger while still maintaining an acceptable latency. Of course, some applications have small packets independent of link speed—telnet and DNS, for instance. But as link speeds increase, the use of "large-packet" applications (sound and image) tends to dominate link usage patterns. Thus, even if a large header consumes a significant portion of small packets, large packets make up most of the link usage, and thus the large header accounts for only a small fraction of the total link usage.

Of course, there exist many slow speed links—for instance voice-band modems, and increasingly, wireless links. On these links, header size is an important issue. In fact, even the IP header, at 20 bytes, is too big for dial-up links, and requires header compression [60].

Thus, we have situations where the largest (CLNP) of the three headers is no problem, and others where the smallest (small SIPP) is too big.

Fortunately, there are mechanisms for reducing the negative impact of a large or deep header. Both of them have already been mentioned. If the problem is just deep headers, then a flow ID style caching scheme such as that of SPip or SIPP (with flow setup) can be used. If the problem is large headers, then a header compression scheme as used with IP can be used. These schemes

are in common use and are known to work well.

Given that these mechanisms exist, the difference between using small SIPP and one of the larger headers is nothing more than a question of at what point a compression scheme must be used. With small SIPP, the bandwidth at which no compression can be tolerated is lower than with one of the larger headers.

Given that serverless auto-address configuration is an important feature, the only situation that could justify choosing small SIPP over one of the (auto-configurable) larger headers is that where 1) a widely deployed technology worked without compression when using small SIPP, but requires compression when using a larger header, and 2) the use of compression is expensive, for instance because the caching characteristics were poor.

Such a situation does not exist today, and in my mind is highly unlikely to exist in the future. Therefore, I conclude that the benefits of using small SIPP are outweighed by the costs.

10.3 Control Protocol Complexity

Most of the control protocols used for CLNP, SIPP, and SPip, such as routing protocols, operate similarly. This is not surprising, since for the capabilities that are covered by all three protocols, the three protocols differ more by mechanism than by semantics.

There are, however, three cases where the control protocol complexity is different between the protocols. One is with small SIPP auto-address configuration. Because small SIPP does not have serverless auto-address configuration, an auto-address configuration protocol, such as DHCP for IP [32], is required. Such protocols are not as complex as distributed routing protocols, but are more complex than, say, ARP, and so are not trivial.

The other two cases where control protocol complexity is different are 1) with provider selection, and 2) certain aspects of the unicast routing protocol. These two cases are covered below.

10.3.1 Unicast Routing Protocols

SPip and CLNP have exactly the same semantics vis a vis hierarchical unicast routing. As a result, their routing protocols operate almost identically. The routing protocols for both advertise longest-match prefixes from the most significant part of the address on down. CLNP does so in the form of an address prefix, and SPip in the form of a sequence of RSEs, but the information conveyed is the same.

The primary difference between CLNP and SPip is that, at forwarding time, CLNP always examines the address from the most significant part, whereas SPip does not necessarily start at the most significant part. As discussed in section 9.1, this means that SPip must understand the

context of the packet (that is, the most significant part) without looking at the most significant part. Insuring that this works correctly requires some additional complexity in SPip's routing protocols.

SIPP shares this difference with SPip. SIPP has an additional difference in that it does not do hole punching across the 64-bit boundary. The routing protocol for SIPP also should operate in terms of longest-match prefixes. However, because SIPP does not do hole punching across the 64-bit boundary, it is conceivable that a SIPP routing protocol could advertise only 64-bit addresses—that is, it does not necessarily advertise addresses from the most significant part.

These differences in SPip and SIPP result in some additional complexity in their routing protocols, and potentially in some new failure modes. SPip and SIPP are discussed in turn in the following two sections.

SPip

The extra complexity to SPip's routing protocol is minimal. As discussed in Section 9.1.6, SPip advances the Active RSE for classical forwarding information but not non-classical (hole-punching) forwarding information. Thus, an SPip router must format its forwarding tables so that it advances the Active RSE appropriately.

Fortunately, this can be done without additional router configuration. When an SPip router receives a routing update, it compares the RSEs in the sequence of RSEs against those of its own address(es). The router can always advance the Active RSE beyond the level where its own RSE matches that in the advertised sequence.

This style of operation introduces the possibility of a new failure mode in the case where a router's address has been misconfigured. In this case, it is possible that such a misconfigured router will advance the Active RSE for what is essentially non-classical forwarding information.

For instance, assume the non-classical forwarding information case of Figure 9.1 in Section 9.1. Assume further that router b has been misconfigured so that it thinks that its own top-level RSE is 1C—that is, that of backbone C.

Assume that a packet for host Z with address sequence 1C:9V:3L arrives at router b from subnet I. Since router b has hole punching information for 1C:9V, and since it thinks it is in 1C (because it was misconfigured), router b advances the Active RSE and forwards the packet to router c. Router c examines 9V, but assumes the context is for an address sequence with 1B in the top-level RSE. If there is a subscriber-level RSE under backbone B whose value matches 9V, router c will incorrectly forward the packet towards that subscriber. The packet will subsequently either be delivered to the wrong subscriber network, or will loop back to router b, which will again interpret the RSE 9V incorrectly, thus forming a loop.

In actual practice, it seems likely that if router b were misconfigured this way, then some other

aspect of the routing algorithm would not operate correctly. For instance, router c might not successfully obtain router b as a neighbor, since it would be expecting router b to have an address prefix similar to its own. None the less, the possibility for the above failure mode exists with SPip, where it does not for SIPP.

SIPP

When the SIPP routing protocol is advertising full prefixes (from the most significant part of the address), SIPP is subject to the same extra complexity and failure mode as described for SPip above.

The SIPP forwarding engine always forwards based on single (64-bit) addresses only. That is, even though multiple addresses may be examined in the course of processing the source route, any single instance of forwarding operates on a single address, and does not consider previous addresses in the route sequence. Because of this, it is theoretically possible for SIPP routing algorithms to advertise only 64 bit addresses.

This requires very careful configuration of the routing algorithm. If the lower part of an address sequence with a certain upper part (say X) is advertised into an area with a different upper part (say Y), the routers in Y will not be able to detect that the lower part is for a destination outside of Y. If there are addresses in Y with the same lower part, then routing will fail.

Because of this, SIPP routing algorithms should always advertise full address sequences, even though the forwarding engine operates on single addresses.

10.3.2 Provider Selection

With CLNP, for a host to select a provider, it must know the appropriate QoS option value to put in the header. With SIPP, for a host to select a provider, it must know the cluster address of the provider. Both of these bits of information must be learned via some discovery mechanism.

With SPip, on the other hand, the provider is selected using the top RSE of the address sequence. This RSE is already known by hosts via the normal address sequence advertisement done by routers (as part of the auto-address configuration process).

Exactly how important this difference is is hard to say, but it is likely to be significant. On one hand, the header "tag" (the provider information in the header—QoS, cluster address, or RSE) is not the only information needed by a host to make an informed provider choice. The host must also know the ramifications of choosing one provider over another—for instance cost and performance. One could argue that, given that this information must be distributed to hosts, the additional complexity of adding the header tag to this information is minimal.

One the other hand, having the header tag is likely to make obtaining the other information needed

easier. For instance, current DNS could be leveraged to give provider information on request. The host could make an inverse query using the top-level (provider) RSE. The record stored by DNS could contain such provider information as provider name, type of network (ATM, X.25, Internet), available types-of-service, and the like. This is particularly useful for learning about the providers of destinations, as it is naturally harder to learn information about remote sites compared to one's own site.

Another advantage of being able to isolate the provider ID is in comparing two addresses to aid in the choice of provider. An SPip host can know definitively if a destination shares the same provider just by comparing its top-level RSE with the destination's. This comparison cannot be made with certainty with SIPP, because the host does not naturally know which bits of the address are the provider part.

10.4 Address Assignment Complexity

This section considers how easy or hard it is to assign addresses with CLNP, SIPP, and SPip. Specifically, it considers the human administrative process of assigning addresses. (Host auto-address configuration is discussed in previous sections of this chapter and is Section 9.5.)

Address administration is easier with SPip than it is with CLNP or SIPP. The fundamental reason for this is that with SPip, each "field" of the address is in a separate RSE, and so is in certain respects is independent of the other fields. With CLNP and SIPP, the fields of the address abut against each other. Thus, the size and position of one field affects the size and position of adjacent fields. This tight relationship between fields in CLNP and SIPP put constraints on address assignment that SPip does not suffer.

This fundamental difference manifests itself in a number of ways. First, the initial assignment of CLNP and SIPP addresses (as opposed to reassignment later on) requires that the field positions of each address be determined. Making this determination is by no means impossible, but it is not easy either, especially if the address is small. In the case of IP, great care must be taken in the assignment of subnet and host fields, precisely because the IP address is small [108]. The problem is not as bad with SIPP, and even less so with CLNP. Still, the process of choosing field sizes has certain administrative costs associated with it, which must be borne by a large number of organizations. For instance, with CLNP, many countries and large organizations have gone through the exercise of assigning field positions.

A more serious problem is that of reassigning addresses. Addresses have to be reassigned from time to time for a number of reasons:

1. Network growth may require a new level of hierarchy.
2. Network growth may cause a field of the hierarchy to be too small.

3. Changing providers may require a new prefix.

4. Changing network location may require a new prefix.

Consider the first case—adding a new level of hierarchy. With SPip, adding a new level of hierarchy is, at least with respect to the address assignment procedure, very easy. A new RSE is inserted at the point in the address sequence where the new hierarchy level is. No reassignment of numbers in existing RSEs is necessary. Of course, hosts and routers under the new level must have new addresses configured, but this is true with CLNP and SIPP also.

With SIPP or CLNP, however, the process is more involved. Consider an address p1.p2.s1.s2. We wish to insert a new hierarchy level in between levels p2 and s1, creating p1.p2.nl.s1.s2. There are several ways in which the new field can be inserted. First, if the address is variable length and p1.p2.s1.s2 is not already the maximum length, then the length is increased by the size of nl, and it is inserted. This is similar to and just about as easy as the SPip case. Unfortunately, NSAP addresses tend to already be at their maximum length (20 bytes) [110], and lengthening the SIPP address requires a minimum of 64 more bits plus additional forwarding overhead.

Second, there may already be space (unassigned bits) in the address between p2 and s1. In this case, adding the new level is easy. Sometimes this may in practice be possible. For instance, the GOSIP definition of CLNP has a "reserved" field. More generally, however, such space does not exist.

Third, the fields s1 and s2 can be shrunk and shoved to the right. This may involve a renumbering—meaning that the value of the fields s1 or s2 may need to be changed for a given hierarchy element. This is a significantly more involved process than the first two methods or than SPip.

If there is not enough address space to the right of p2 to accommodate the new hierarchy level, then it is necessary to requisition a new portion of the address space to create the extra room. For instance, a new prefix p1.p3 might be obtained, where the field size of p3 is smaller than that of p2. This process is also more involved than the first two methods or SPip.

The second potential reason for reassignment, where a field becomes too small to assign all of the required numbers, should virtually never happen with SPip. This is because SPip's field size is 19 bits, or more than 500,000 assignments at a single level of the hierarchy. It is more likely with CLNP, where fields, such as the area field, are as small as 16 bits, though 65,000 is still a large number of assignments. With SIPP, especially small SIPP, it is more likely still.

Just because a field overflows does not mean that the field size must be changed. Another alternative is to assign the hierarchy element that ran out of numbers another prefix. In this case, the hierarchy element would appear twice in forwarding tables. This method trades off forwarding table size for ease of address assignment. In the worst case, however, the field must be resized, requiring a process similar to the one described above for adding a hierarchy level (that is, it is easy for SPip and potentially more involved for SIPP and CLNP).

The last two potential reasons for reassignment, getting a new prefix because of provider change or moving, can result in lower-level field resignments if the new prefix is longer than the old one. This, again, requires a process similar to the one described above. Either the subscriber must shrink its existing fields, or the provider (or geographic area) must obtain a new prefix space. One way to avoid this situation is to make sure that all provider prefixes are the same length. This, however, adds constraints to the initial assignment process, making it more difficult.

Chapter 11

Summary and Conclusions for Part II

Part II of this thesis describes and analyzes three protocols—one with conventional syntax and semantics (CLNP), one with a conventional syntax but expanded semantics (SIPP), and one with a new syntax and semantics (SPip). The new semantics and syntax are the major contributions of this thesis.

The new semantics come from an expanded use of the source route mechanism to achieve flexible routing and addressing while maintaining good performance.

To demonstrate the flexible routing and addressing, the capabilities of the three protocols are analyzed. Specifically, ten "required" and eight "useful" capabilities are considered. All of the capabilities are ones that have been discussed in the IPng (IP next generation) process of the IETF, and represent a wide range of internetworking applications.

Of these capabilities, both SIPP and SPip handle all of the ten required ones. CLNP handles eight of the required ones, including the four most critical ones. CLNP does not handle mobility, which is a serious weakness. CLNP also does not handle two-phase shared-tree multicast, which may prove to be a serious weakness, as that style multicast may prove important for global scaling.

Of the eight useful capabilities, SPip handles seven of them. The only one that SPip does not handle is embedded link-layer addresses. Embedded link-layer addresses, however, is not so much a routing and addressing problem as an address resolution problem, and in my opinion its solution should not come from the address per se. It is better to convey the link-layer address in an option or out-of-band. This opinion is supported by the fact that CLNP's embedded link-layer addresses have scaling and operational problems (see Chapter 5).

Both SIPP and CLNP handle four of the eight useful capabilities.

Next, Part II analyzes the costs of the three protocols. These costs are analyzed primarily for forwarding cost (speed or hardware complexity) and header size, but also for control protocol complexity and address assignment complexity.

The major results of this analysis are as follows:

- SIPP generally has the lowest forwarding cost when a cache hit occurs. SIPP's caching performance depends on the situation, however, and can be much better or somewhat worse than SPip's or CLNP's. SPip's and CLNP's caching performance is similar. Most router forwarding takes place under cache hit conditions.
- SPip's "best possible" caching performance (cut-through switching at wire speed) is the best of the three, CLNP's the worst.
- SPip's forwarding cost is independent of forwarding table size, whereas SIPP's and CLNP's are dependent on forwarding table size.
- For large forwarding tables (8192 entries), SPip's forwarding cost is significantly better than SIPP's, which is in turn significantly better than CLNP's.
- For small forwarding tables (16 entries), SPip and SIPP without a route sequence advance perform similarly, SIPP with a route sequence advance performs somewhat worse, and CLNP performs worse still.
- SIPP has a smaller header than SPip or CLNP, but primarily under the conditions where simple (non-extended) addresses are used. This eliminates the serverless host auto-address configuration feature of SIPP, however, and so is considered not worth doing. Otherwise, the packet size of the three protocols are comparable, with CLNP generally being slightly worse and SIPP generally being better.
- Ignoring small SIPP's need for a host address assignment protocol, the control protocol complexity among the three protocols is generally comparable, though SPip is slightly less complex in the area of provider selection, and SIPP and SPip require a slightly more complex routing protocol than does CLNP.
- The address assignment process for SPip is simpler than that of SIPP and CLNP.

The conclusion of this thesis is that the generalization of the source routing mechanism increases internet protocol capabilities over conventional methods. This is achieved at lower cost in terms of performance and operation compared to conventional methods. This conclusion is particularly true for the comprehensive use of the source routing mechanism—that is, where it is used for locators—of SPip. It also holds, though less strongly, for the hybrid approach—a source route of bitwise maskable addresses—of SIPP.

This conclusion has two ramifications—one for the present and one for the future. For the present, it means that we can get the required features needed for continued growth, both functionally and actually, of the internet.

For the future, it means an increased probability that the protocol will more easily be able to accommodate as-yet-unconceived features. This is particularly true of SPip, which is designed in terms of a "routing and addressing engine". That is, SPip executes according to the elemental routing and addressing functions, as described in Part I of the thesis, rather than according to the desired semantics of the protocol. The desired semantics are achieved by establishing the appropriate forwarding tables in routers and the appropriate address sequences in hosts. New semantics are derived by modifying the control protocols and system configuration (such as DNS), but without modifying the basic internet protocol.

Chapter 12

Epilogue

Just to confirm the old saying that there is nothing new under the sun, I offer the following epilogue.

As stated in Chapter 1, there was a flurry of creative activity in the mid to late 1970's that resulted in the internetwork architecture, and in particular Pup [9] and IP [86, 87]. These two protocols, however, do not in my mind adequately reflect the depth of understanding that the early internet architects had of routing and addressing.

Specifically, one RFC and three IENs (Internet Engineering Notes) [85, 25, 22, 18] document some of the debate that took place during the year from mid-1977 to mid-1978 and that led to the IP protocol. (Of course, I discovered these documents well after I had done the basic work on Pip :-). These four documents show that 1) one of the central ideas of Pip, namely that of putting the fields of the address in individual header fields, had been proposed by Jon Postel [85], and that 2) most of what the internet community is currently "discovering" about routing and addressing was already thought of by Postel, Sunshine, Cohen, Clark, Cerf, and perhaps others who did not bother putting out IENs (or whose IENs or other publications I have not bothered to read).

In what follows, the aspects of that debate that relate to the findings of this thesis are summarized.

The opening paragraph in Postel's May 1977 RFC730, "Extensible Field Addressing", says:

> This memo discusses the need for and advantages of the expression of addresses in a network environment as a set of fields. The suggestion is that as the network grows the address can be extended by three techniques: adding fields on the left, adding fields on the right, and increasing the size of individual fields. Carl Sunshine has described this type of addressing in a paper on source routing [102].

This, in a nutshell, is Pip's (or SPip's) route sequence, which allow adding fields (RSEs) on the left, the right, and in the middle (which Postel's can do too, though he does not mention it above).

Later in the RFC, Postel says:

> The prospect of interconnections of networks to form a complex multinetwork system poses additional addressing problems. The new Host-IMP interface specification has reserved fields in the leader to carry network addresses. There is experimental work in progress on interconnecting networks. We should be prepared for these extensions to the address space.

Talk about understatement!

And later still:

> A problem with simple field addressing is the desire to specify only the fields that are necessary given the local context. A program interpreting the address is then unsure what the first field represents. Some clues are needed in the address specification for correct parsing to be possible. Dave Crocker has described a syntax for a similar problem in network access of data.

Trying to do this (only including the fields of the address relevant to local context in the header) with SPip turned out to be problematic. Still, the "clue" that Postel refers to is useful for efficient processing of the header, as the router only needs to parse the relevant fields. This "clue" is SPip's Active RSE field.

Postel gets to the meat of the thing in the following excerpt:

> Specifically I suggest that we adopt a field based extensible address scheme where each field is separated from its neighbors by a delimiter character and each field has a name. When an address is specified the name of the most general field must also be indicated.
>
> Definitions:
>
> ⟨address⟩ ::= ⟨field-name⟩ ":" ⟨fields⟩
>
> ⟨field-name⟩ ::= "NET" — "IMP" — "HOST" — "MESSAGE-ID"
>
> ⟨fields⟩ ::= ⟨field⟩ — ⟨field⟩ "/" ⟨fields⟩
>
> ⟨field⟩ ::= a decimal number

SPip has a few differences from this, but the basic idea is there. Something to give the context of the field to be parsed (SPip's Active RSE = Postel's field-name), and a series of fields (SPip's are binary not decimal, but had Postel's found their way into a packet, I'm sure they would have been binary).

In April of 1978, Danny Cohen shed more light on the nature of addresses in his IEN31, "On Names, Addresses and Routings (II)" [25]:

I HATE TO ADMIT IT, BUT ...

At the beginning of this note, and in an earlier note, I used a great line telling that "names tell what the processes are, and addresses tell where they are." It continues by "routings tell how to get there."

I hate to admit that by now I have some reservations about this definition. My name is "Danny." My address is "ISI." When I was at Tech, my name was the same, but the address was different. This supports the definition. How about the addresses in a broadcasting media network? When a host changes its position (location) on the same Ethernet, its address does not change. Well, maybe these addresses are not real addresses, according to the definition. Admittedly, this is an uncomfortable thought.

I believe that there is a better explanation. I suggest that an address is "the canonic routing from the root of the addressing-tree." It sounds recursive, does not it?

To be more precise, an addressing scheme is a hierarchical organization of elements, with code assignment such that each element has a unique set of codes, corresponding to its position in the hierarchy.

The notion that the address tells how-to-get-there from the root of the tree is very similar to the notion that absolute coordinates are really relative, with respect to the origin.

Since we know (by default) how to get from the source to the UA root, and since the address tells how to get to the destination from the root, the address tells how to get from the source to the destination.

Hence, by definition, addresses are routings.

This last conclusion, that addresses are routes, is a key "finding" of the taxonomy section (Section 2.2) of this thesis, and is the basis of SPip's route sequence.

Later on in the IEN, Cohen makes a proposal:

Our proposal for addressing and routing is as follows:

- Establish a UA (Universal Address) scheme, of variable level structure.
- Disseminate as much knowledge to each participating node as deemed practical.
- Allow the option of routing to be included in the headers of the messages.
- Refuse delivery of messages to a destination with unknown routing.
- Establish internet-directory-assistance service.

This last point is crucial. "Internet-directory-assistance" (now known as DNS) must advertise the "route" from the root of the hierarchy to the leaves. In particular, if the packet format is a string of fields (or addresses, as in SIPP), then DNS should advertise that string.

So, at this point in the discourse (April 1978), Postel has provided the address format, and Cohen the architectural underpinning from which to understand that address. So, what happened? Why

did we end up with IP and not something more like SPip?

We find a clue to the answer from IEN46, written by Clark and Cohen in June of 1978 called "A Proposal for Addressing and Routing in the Internet" [22]. After discussing several problems with routing and addressing, they make the following statement:

> The solution which has been proposed in the past to cope with this is to replace the address in the packet with a route, called a source route since it is provided by the source of the packet. The disadvantage of having a route in a packet instead of an address is that the concept of an address is very useful one. For example, for accounting purposes it may be necessary to note the source and destination of a packet as it passes through a transit net. Clearly, it is desirable that the source and destination be uniquely identified for this purpose, something not easily done if the source and destination are specified only by a route. Thus, we propose that the address continue to be the primary piece of information in the packet, but that it be possible to include, in addition, an optional source route.

So, here they recognize the need for a compact, simple, fixed length *something* to identify the source and destination of a packet. But, this is nothing more than the EID of SPip. So, the need for both an identifier and an "address" (still at that time arguable to be a route) was clearly recognized. However, they added the source route to handle the routing bit, and kept the address as the primary piece of information.

I think this would have been fine (indeed, this is SIPP's approach) except for the crucial thing mentioned by Cohen in IEN31:

- Establish internet-directory-assistance service.

Well, DNS was of course established, but it did not contain the source route, just the address. So, the "routing" information in the packet was effectively limited to a single 32 bit field.

I was interested to find the following in the Clark/Cohen paper:

> 5. Migration
>
> What is the relationship between the scheme proposed here and the current internet header with a fixed size address field? Happily, adoption of the addressing strategy involving regions together with the optional internet source route implies no immediate upheaval to packet formats or gateway code. Currently, every network is a region, and every gateway thus contains code for doing inter-region routing. Eventually, gateways will want to be modified as follows. When a region finally is defined which contains more than one network, then gateways inside that region will need to understand an additional component of the internet address. Thus, unless gateway code is rewritten

> for different regions, it will be necessary to write code which can deal, eventually, with a variable size component of the address. The address itself, however, can reasonably be a fixed size, since it is merely an address and not a route. In fact, it seems that the field as specified for the current internet header is sufficient in size, although perhaps marginally so.

Well, something happened here. An argument was put forth that 32 bits is enough because the address does not have to do routing–the source route can handle the rest. Clearly it was recognized that a variable length *something* was needed, but the source route was deemed sufficient for that, and the 32-bit address won out in the end. So, perhaps what killed IP is not that the address is too short (though probably it is), but that the ability for DNS to hand a host a source route (which it could then put in the header so that the right thing could happen in the network) was not created.

So, indeed SPip's routing sequence is a combination of Postel's Extensible Field Addressing (EFA) and Clark and Cohen's "address", though with SPip the "routing" part of the "address" has been largely moved over to the EFA (route sequence in SPip), and the "address" (EID in SPip) is left with the identification function.

An IEN from Cerf the following month (July 1978) seems to meld with Clark/Cohen (IEN48, "The Catenet Model for Internetworking" [18]). It generally confirms the Clark/Cohen proposal. It, however, makes some additional interesting statements:

> In order to limit the overhead of address fields in the header, it was decided to restrict the maximum length of the host portion of the internet address to 24 bits. The possibility of true, variable-length addressing was seriously considered. At one point, it appeared that addresses might be as long as 120 bits each for source and destination. The overhead in the higher level protocols for maintaining tables capable of dealing with the maximum possible address sizes was considered excessive.

Not only is it interesting that a longer address (120 bits, almost as long as an NSAP), was seriously considered, but the reason for not going with it (memory overhead to "upper layer protocols") really shows how times have changed.

Finally, Cerf's IEN seems to delegate source routing to its current, and very limited, role:

> One of the major arguments in favor of variable length "addressing" is to support what is called "source-routing." The structure of the information in the "address" really identifies a route (e.g., through a particular sequence of networks and gateways). Such a capability could support ad hoc network interconnections in which a host on two nets could serve as a private gateway. Though it would not participate in catenet routing or flow control procedures, any host which knows of this private gateway could send "source-routed" internet datagrams to that host.

It is interesting that the original ideas of Postel and Cohen (very SPip-like) evolved into the source route, which was then limited to a "special service" role (i.e., routing a packet through a private host on two nets).

To conclude, I must say that when I read these four documents, I found it fascinating and delightful to discover that my work, with the considerable aid of hindsight, was able to confirm, and put in a modern context, the early thinking of the internet architects.

Bibliography

[1] A. Autolitano, F. Bernabei, M. Ciampi, and M. Listanti. Application of Generalized Parallel Delta Networks to a Hybrid Broadband Switch. *International Conference on Communications*, 1:123–127, June 1989.

[2] A. Ballardie, P. Francis, and J. Crowcroft. An Architecture for Scalable Inter-Domain Multicast Routing. *Proceedings of ACM SIGCOMM 93*, pages 85–95, September 1993.

[3] A. Bar-Noy and M. Gopal. Topology Distribution Cost vs. Efficient Routing in Large Networks. *Proceedings of ACM SIGCOMM 90*, pages 242–252, September 1990.

[4] F. Bernabei and M. Listanti. A Hybrid Switching Exchange for Broadband Communications. *Proceedings of the Ninth International Conference on Computer Communication Technologies for the 90's*, pages 61–65, October 1988.

[5] D. Bertsekas and R. Gallager. *Data Networks.* Prentice-Hall, 1987.

[6] L. Bhuyan and D. Agrawal. Generalized Hypercube and Hyperbus Structure for a Computer Network. *IEEE Transactions on Computing*, C-33:323–333, 1984.

[7] big internet@munnari.oz.au. *Big Internet Mailing List.* archive at munnari.oz.au:big-internet/list-archive/*.

[8] K. Birman, A. Schiper, and P. Stephenson. Lightweight Causal and Atomic Group Multicast. *ACM Transactions on Computer Systems*, 9(3):272–314, August 1991.

[9] D. Boggs, J. Shoch, E. Taft, and R. Metcalfe. Pup: An Internetwork Architecture. *IEEE Transactions on Communications*, COM-28(4):612–624, April 1980.

[10] R. Braden and J. Postel. Requirements for Internet Gateways. Request For Comments 1009, University of Southern California Information Sciences Institute, June 1987.

[11] L. Breslau and D. Estrin. Design of Inter-Administrative Domain Routing Protocols. *Proceedings of ACM SIGCOMM 90*, pages 231–241, September 1990.

[12] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A Simple Network Management Protocol (SNMP). Request For Comments 1157, University of Southern California Information Sciences Institute, May 1990.

[13] Steve Casner. *Private Communications.* ISI.

[14] CCITT. *CCITT E.163, Numbering Plan for the International Telephone Service*, Blue Book, 1988.

[15] CCITT. *CCITT X.121, International Numbering Plan for Public Data Networks*, Blue Book, 1988.

[16] CCITT. *CCITT X.25, Interface Between Data Terminal Equipment (DTE) and Data Circuit Terminating Equipment (DCE) for Terminals Operating in the Packet Mode on Public Data Networks*, Blue Book, 1988.

[17] CCITT. *Draft Recommendation Q.931, ISDN User-Network Interface Layer 3–General Aspects*, Blue Book, 1988.

[18] V. Cerf. The Catenet Model for Internetworking. Internet Engineering Note 48, University of Southern California Information Sciences Institute, July 1978.

[19] D. Cheriton. Sirpent: A High-Performance Internetworking Approach. *Proceedings of ACM SIGCOMM 89*, Sepember 1989.

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

[21] D. Clark. Policy routing in Internet Protocols. Request For Comments 1102, University of Southern California Information Sciences Institute, May 1989.

[22] D. Clark and D. Cohen. A Proposal for Addressing and Routing in the Internet. Internet Engineering Note 46, University of Southern California Information Sciences Institute, June 1978.

[23] D. Clark, V. Jacobson, J. Romkey, and H. Salwen. An Analysis of TCP Processing Overhead. *Transactions on Communications*, 27(6):23–29, June 1989.

[24] D. Clark and D. Tennenhouse. Archictectural Considerations for a New Generation of Protocols. *Proceedings of ACM SIGCOMM 90*, pages 200–208, September 1990.

[25] D. Cohen. On Names, Addresses and Routings (II). Internet Engineering Note 31, University of Southern California Information Sciences Institute, April 1978.

[26] J. Crowcroft and K. Paliwoda. A Multicast Transport Protocol. *Proceedings of ACM SIGCOMM 88*, pages 247–256, August 1988.

[27] S. Deering. Multicast Routing in Internetworks and Extended LANs. *Proceedings of ACM SIGCOMM 88*, August 1988.

[28] S. Deering. Host Extensions for IP Multicasting. Request For Comments 1112, University of Southern California Information Sciences Institute, August 1989.

[29] S. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, Palo Alto, California, 1991.

[30] S. Deering. SIP: A Simple Internet Protocol. *IEEE Network*, 7(6):16–28, May 1993.

[31] S. Deering, D. Waitzman, and C. Partridge. Distance Vector Multicast Routing Protocol. Request For Comments 1075, University of Southern California Information Sciences Institute, November 1988.

[32] R. Droms. Dynamic Host Configuration Protocol. Request For Comments 1531, University of Southern California Information Sciences Institute, October 1993.

[33] W. Edmond, K. Seo, M. Leib, and C. Topolcic. The DARPA Wideband Network Dual Bus Protocol. *Proceedings of ACM SIGCOMM 90*, pages 79–89, September 1990.

[34] D. Estrin. Policy Requirements for Inter Administrative Domain Routing. Request For Comments 1125, University of Southern California Information Sciences Institute, November 1989.

[35] D. Estrin and K. Obraczka. Connectivity Database Overhead for Inter-Domain Policy Routing. *Proceedings of IEEE Infocom 91*, April 1991.

[36] D. Estrin, Y. Rekhter, and S. Hotz. Scalable Inter-Domain Routing Architecture. *Proceedings of ACM SIGCOMM 92*, pages 40–52, August 1992.

[37] P. Francis. A Near-term Architecture for Deploying Pip. *IEEE Network*, 7(6):30–37, May 1993.

[38] P. Francis. Pip Header Processing. Request For Comments 1622, University of Southern California Information Sciences Institute, May 1994.

[39] P. Francis. Pip Near-term Architecture. Request For Comments 1621, University of Southern California Information Sciences Institute, May 1994.

[40] P. Francis and R. Govindan. Flexible Routing and Addressing For a Next Generation IP. *Submitted to ACM SIGCOMM 94*, 1994.

[41] G. Fredereckson and R. Janardan. Designing Networks with Compact Routing Tables. *Algorithmica*, 3:171–190, 1988.

[42] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless Inter-domain Routing (CIDR): an Address Assignment and Aggregation Strategy. Request For Comments 1519, University of Southern California Information Sciences Institute, September 1993.

[43] G. Finn. Routing and addressing problems in large metropolitan-scale internetworks. Research Report ISI/RR-87-180, Information Sciences Institute, Marina del Rey, California, March 1987.

[44] G. Finn. Reducing the vulnerability of dynamic computer networks. Research Report ISI/RR-88-201, Information Sciences Institute, Marina del Rey, California, June 1988.

[45] P. Gross and P. Almquist. IESG Deliberations on Routing and Addressing. Request For Comments 1380, University of Southern California Information Sciences Institute, November 1992.

[46] J. Hagouel. *Issues in Routing for Large and Dynamic Networks.* PhD thesis, Columbia University, New York City, 1983.

[47] Joel Halpern. *Private Communications.* Network Systems Corp.

[48] E. Huizer and D. Crocker. IETF Wroking Group Guidelines and Procedures. Request For Comments 1603, University of Southern California Information Sciences Institute, March 1994.

[49] idmr@cs.ucl.ac.uk. *Inter-Domain Multicast Routing Mailing List.* ftp cs.ucl.ac.uk:darpa/idmr-archive.Z.

[50] IEEE. *Logical Link Control*, IEEE 802.2, 1988.

[51] IEEE. *Token Ring Access Method and Physical Layer Specifications*, IEEE 802.5, 1989.

[52] International Organization for Standardization. *Data Interchange - Structures for the Identification of Organizations*, ISO 6523, 1984.

[53] International Organization for Standardization. *End System to Intermediate System routeing exchange protocol for use in conjunction with the Protocol for providing the connectionless-mode network service (ISO 8473)*, ISO 9542, 1988.

[54] International Organization for Standardization. *Information Processing Systems - Data Communications - Network Service Definition - Addendum 2: Covering Network Layer Addressing*, ISO 8348 AD2, March 1988.

[55] International Organization for Standardization. *Protocol for providing connectionless-mode network service*, ISO 8473, 1988.

[56] International Organization for Standardization. *Intermediate System to Intermediate System routeing exchange protocol for use in conjunction with the Protocol for providing the connectionless-mode network service (ISO 8473)*, ISO 10589, 1991.

[57] International Organization for Standardization. *Information Technology - Telecommunications and Information Exchange Between Systems - Network Service Definition for Open Systems Interconnection: Amendment 5: Addition of Group Network Addressing*, ISO/IEC 8348/Am. 5, 1993.

[58] J. Ioannidis, D. Duchamp, and G. Maguire Jr. IP-Based Protocols for Mobile Internetworking. *Proceedings of ACM SIGCOMM 91*, pages 235–245, 1991.

[59] J. Liu and W. Hsu. Routing and broadcasting algorithms for a large family of interconnection topologies. Technical Report CPS-91-05, Dept. of Computer Science, Michigan State University, October 1991.

[60] V. Jacobson. Compressing TCP/IP Headers for Low-Speed Serial Links. Request For Comments 1144, University of Southern California Information Sciences Institute, February 1990.

[61] J. Jaffe. Hierarchical clustering with topology databases. *Computer Networks and ISDN Systems*, 15(5):329–339, 1988.

[62] F. Kamoun and L. Kleinrock. Hierarchical Routing for Large Networks: Performance Evaluation and Optimization. *Computer Networks*, 1(3):155–174, January 1977.

[63] H. Katseff. Incomplete Hypercube. *IEEE Transactions on Computing*, C-37:604–607, 1988.

[64] D. Katz and P. Ford. TUBA: Replacing IP with CLNP. *IEEE Network*, 7(6):38–47, May 1993.

[65] D. Knuth. *The Art of Computer Programming, Vol. 3 (Sorting and Searching).* Addison Wesley, 1973.

[66] J. Kolar and H. Wu. A Study of Survivability Versus Cost for Several Fiber Network Architectures. *IEEE International Conference on Communications '88*, pages 61–66, vol. 1, June 1988.

[67] G. Lauer. Hierarchical routing design for SURAN. *IEEE International Conference on Communications (ICC) '86*, 1:93–102, June 1986.

[68] D. Lynch and M. Rose. *Internet System Handbook.* Addison Wesley, 1993.

[69] N. Maxemchuck. The Manhattan Street Network. *IEEE GLOBECOM '85*, pages 255–261, vol. 1, December 1985.

[70] A. McAuley and P. Francis. Fast Routing Table Lookup Using CAMs. *Proceedings of INFOCOM 93*, pages 1382–1391, March 1993.

[71] P. Mockapetris. Domain Names - Implementation and Specification. Request For Comments 1035, University of Southern California Information Sciences Institute, November 1987.

[72] R. Moore, N. Geer, and H. Graf. GRIDNET- An Alternative Large Distributed Network. *IEEE Computer Magazine*, pages 57–66, April 1984.

[73] J. Moy. OSPF Version 2. Request For Comments 1131, University of Southern California Information Sciences Institute, July 1991.

[74] R. Newman, Z. Budrikis, and J. Hullett. The QPSX Man. *IEEE Communications Magazine*, 26(4):20–28, April 1988.

[75] C. Partridge. A Proposed Flow Sepcification. Request For Comments 1363, University of Southern California Information Sciences Institute, September 1992.

[76] C. Partridge, T. Mendez, and W. Milliken. Host Anycasting Service. Request For Comments 1546, University of Southern California Information Sciences Institute, November 1993.

[77] C. Partridge and S. Pink. A Faster UDP. *Transactions on Networking*, 1(4):429–440, August 1993.

[78] T. Pei and C. Zukowski. VLSI Implementation of Routing Tables: Tries and CAMs. *Proceedings of INFOCOM 91*, April 1991.

[79] R. Perlman. A Protocol for Distributed Computation of a Spanning Tree in an Extended LAN. *Ninth Data Communications Symposium*, 1985.

[80] R. Perlman. Hierarchical Networks and the Subnetwork Partition Problem. *Computer Networks and ISDN Systems*, 5:297–303, 1985.

[81] R. Perlman. *Network Layer Protocols with Byzantine Robustness.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1988.

[82] R. Perlman and G. Varghese. Pitfalls in the Design of Distributed Routing Algorithms. *Proceedings of ACM SIGCOMM 88*, pages 43–54, August 1988.

[83] D. Pitt and R. Dixon. Addressing, Bridging, and Source Routing (LAN Interconnection). *IEEE Network*, 2(1):25–32, January 1988.

[84] D. Plummer. Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.Bit Ethernet Address for Transmission on Ethernet Hardware. Request For Comments 826, University of Southern California Information Sciences Institute, November 1982.

[85] J. Postel. Extensible Field Addressing. Request For Comments 730, University of Southern California Information Sciences Institute, May 1977.

[86] J. Postel. Internetwork Protocol Approaches. *IEEE Transactions on Communications*, COM-28(4):604–611, April 1980.

[87] J. Postel. Internet Protocol. Request For Comments 791, University of Southern California Information Sciences Institute, September 1981.

[88] J. Postel. Transmission Control Protocol. Request For Comments 793, University of Southern California Information Sciences Institute, September 1981.

[89] D. Pradhan. Dynamically Restructurable Fault Tolerant Processor Network Architectures. *IEEE Transactions on Computing*, C-34:434–447, 1985.

[90] B. Rajagopalan. Reliability and Scaling Issues in Multicast Communication. *Proceedings of ACM SIGCOMM 92*, pages 188–198, August 1992.

[91] K. Ramakrishnan. Performance Considerations in Designing Network Interfaces. *Journal on Selected Areas in Communications*, 11(2):203–219, February 1993.

[92] J. Reynolds and J. Postel. Assigned Numbers. Request For Comments 1340, University of Southern California Information Sciences Institute, July 1992.

[93] Benny Rodrig. *Private Communications.* RAD Network Devices, Ltd.

[94] V. Rutenburg and R. Ogier. Fair Charging Policies and Minimum-Expected-Cost Routing in Internets with Packet Loss. *Proceedings of IEEE Infocom 91*, April 1991.

[95] H. Saltzer, D. Reed, and D. Clark. Source Routing for Campus-Wide Internet Transport. *Proceedings of the IFIP WG 6.4 Workshop on Local Networks*, pages 25–32, August 1980.

[96] J. Saltzer. On the Naming and Binding of Network Destinations. *Local Computer Networks*, pages 311–317, 1982.

[97] N. Santoro and R. Khalib. Routing Without Routing Tables. Technical Report SCS-TR-6, Carlton University School of Computer Science, Ottawa, 1982.

[98] N. Shacham. Organization of Dynamic Radio Network by Overlapping Clusters: Architecture Considerations and Optimization. *Proceedings of the Tenth International Symposium PERFORMANCE '84*, pages 435–447, December 1984.

[99] J. Shoch. Inter-Network Naming, Addressing, and Routing. *Proceedings 17th IEEE Computer Society Internation Conference*, pages 72–79, September 1978.

[100] T. Starling. Network Reconfiguration Saves Society Corp. $1 Million a Year. *Bank System Equipment*, 25(11):48–49, 1988.

[101] M. Steenstrup. An Architecture for Inter-Domain Policy Routing. Request For Comments 1478, University of Southern California Information Sciences Institute, June 1993.

[102] C. Sunshine. Source Routing and Computer Networks. *Proceedings of ACM SIGCOMM 77*, January 1977.

[103] C. Sunshine. Addressing Problems in Multi-Network Systems. *Proceedings of INFOCOM 82*, pages 12–18, March 1982.

[104] F. Teraoka, Y. Yokote, and M. Tokoro. A Network Architecture Providing Host Migration Transparency. *Proceedings of ACM SIGCOMM 91*, pages 209–220, 1991.

[105] D. Train and A. Carlton. Error Handling and Other Implementation Details of Cut-Through Switching. *Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 125–127, June 1987.

[106] P. Tsuchiya. The Landmark Hierarchy: a New Hierarchy for Routing in Very Large Networks. *Proceedings of ACM SIGCOMM 88*, pages 35–42, August 1988.

[107] P. Tsuchiya. Efficient and Robust Policy Routing Using Multiple Hierarchical Addresses. *Proceedings of ACM SIGCOMM 91*, September 1991.

[108] P. Tsuchiya. On the Assignment of Subnet Numbers. Request For Comments 1219, University of Southern California Information Sciences Institute, April 1991.

[109] P. Tsuchiya. Internet Routing over Large Public Data Networks using Shortcuts. *Proceedings of ACM SIGCOMM 92*, October 1992.

[110] U.S. Department of Commerce, National Institute of Standards and Technology, Gaithersburg, MD, USA. *Government Open Systems Interconnection Profile (GOSIP) Version 2*, federal information processing standard 146-1 edition, April 1991.

[111] J. Van Leeuwen and R. Tan. Interval Routing. *The Computer Journal*, 30(4):298–307, 1987.

[112] D. Wall. *Mechanisms for Broadcast and Selective Broadcast.* PhD thesis, Stanford University, Palo Alto, California, 1980.

[113] J. Westcott and G. Lauer. Hierarchical Routing for Very Large Networks. *IEEE Military Communications Conference MILCOM '84*, 2:214–218, October 1984.

[114] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, September 1993.