A New IP Routing and Addressing Architecture

J. Noel Chiappa

[ Author’s note: This document is not yet fully complete, but it is being distributed in this rough form since there appears to be a considerable appetite in the network community for discussion on this topic. It is hoped that this note, even in its present form, will facilitate useful debate about the issue. ]

Status of the Memo

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

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

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

1 Introduction

This document presents one possible new IP routing and addressing architecture. By routing and addressing it is meant that part of the overall IP architecture which is responsible for identifying computing nodes, where they are in the Internet, and how to get traffic from one to another. It represents one person’s view of a good overall system answer to this question, and is not to be taken as anything more than that.

While this document will be of most interest to those with some degree of familiarity with routing concepts, it is aimed at the general network community, which will have to agree with the general direction taken in evolving this aspect of the IP architecture. While the subject is a complex one, the document has been written to be accessible to a general audience. As such, it contains much material which will be familiar to those who are versed in routing work, which has unavoidably lengthened the document.

The goals of the new architecture are more or less those outlined in RFC1126 [Little], particularly Appendix A, but modified somewhat to include what I feel are real constraints, such as more complex external policy concerns. To review them briefly, they are to allow a system substantially larger than the current one (if possible of essentially unlimited size), to address the policy concerns which are of increasing importance, to allow topologies of a much more complex nature, and to retain the flexibility of the current system. A concise outline of the goals believed necessary are given in the section below.

These design goals are a very aggressive set of requirements. They exceed considerably the existing capabilities of other large communication systems, such as the telephone system. Two aspects in particular represent a major leap. The first such aspect is the attempt to create a ubiquitous
communication system (and route traffic through it) out of a very large number of cooperating autonomous units (although in some senses the road network does currently accomplish this goal). The second is the assumption that the system will be extremely dynamic; i.e. respond in real time, without human intervention, to changes in the topology and other configuration aspects of the system.

To meet these goals requires a substantial modification to the basic philosophy and mechanism of the existing IP packet layer architecture. Since the system is now widely deployed, a change of this sort — an in-place modification to a large, operating, system — presents a substantial challenge.

To meet it it is first necessary to solve the problem of routing, and only then turn to addressing. As will be seen, routing (and especially the abstraction of data which will be necessary to handle a very large system) is a sufficiently difficult problem that any way of making the problem easier is desperately needed. Addressing needs to be completely sublimated to the needs of the routing architecture; the form of addresses should be whatever is most useful to the routing. Dividing the address up in such a way as to ease administrative tasks is not correct, although a separate mechanism to handle such administrative concerns is possible.

It should be noted that this document is not an engineering design document. No detailed mechanisms have been laid out, and undoubtedly many pitfalls remain along the path to a complete system. Little attempt has been made to consider optimizations for common cases and other engineering practises; no study has been made of traffic patterns, and of course the cost of the actual mechanisms is unknown since the design has not been done; both are necessary to guide the cost/benefit choices inherent in good engineering. Rather, this document contains a study which attempts to clarify what are the fundamentals of the problem, and then identifies what seems to be the best path available to solve that problem.

Also, it was felt that rather than proceed forward along a visible path from where we are now, the best method (when considering the long term health of the the network) was to act as if a blank piece of paper were available. When the best possible design was created, it would at that point be appropriate to see whether it was feasible to reach that design from the current system. Proceeding from the optimal end point backward seems more likely to result in the best long term results than proceeding forward from the current state of affairs in an incremental fashion.

Thus, what this document is an attempt to decide on a general plan of attack on the problem; a broad-brush architecture based on a long study of the fundamental issues and problems in routing and addressing in a very large network.

2 Design Goals

The list of goals for the new architecture are of three types. The first are those capabilities that the existing architecture already has, which are still needed. The second are things that it cannot do, and which are proving to be major hindrances in actual operation. The third are those necessities which will be required in the future, and which may not be obvious now.

There are also certain longstanding goals of the IP architecture, such as the ability to support mobile hosts, which have previously been difficult. It proves possible to support these in the new architecture, as will be seen.

2.1 Current Capabilities

First and foremost, the new architecture must be incrementally deployable with respect to the existing system. The capital investment in
existing software and plant, and the day-to-day operational dependence of many people, is so large that to start from scratch would be phenomenally expensive, and utterly impractical. If there were no way to do it without a fresh start, perhaps it would be practical, but since it seems to not be necessary, incremental deployability is a must. This does not preclude eventual replacement of the existing architecture, obviously; to retain it indefinitely would unduly cramp the system.

In specific terms, a new system should not require any changes to hosts to become fully operational (although eventual changes to hosts are allowable to take full power from the new scheme); too many unmodifiable existing hosts exist, and support for them must be maintained for some time in the architecture.

Second, changes to routers (which are almost certainly unavoidable if anything concrete is to be attained) need to be interoperable (for a period, at least) with the existing system, although it is reasonable in the case of routers to insist that basically all be converted before the new architecture can be fully operational. Some feel that even changing all the routers is too ambitious, and unnecessary, but it appears that this goal is unneeded, and conflicts with the general principle of providing the best possible system for the long term.

Next, the current system allows minimal firewalls between AS’s; errors in routing data from outside the AS cannot (in a properly designed AS) interfere with communication inside the AS. While the concept of an AS is perhaps outdated, this property is desirable, although it needs to be strengthened and made more flexible. As things stand, a single misbehaving AS can cause almost limitless problems in the inter-AS routing. A systematic way of minimizing such problems needs to be included.

Finally, it must be vendor-independent; that is, specified in enough detail that an implementation by a new vendor, operating solely from the specifications, will interoperate correctly with other existing implementations.

2.2 Current Limitations

The things about the current system that are most limiting are the limit on the total size of the system, the lack of support for policy routing, and the inability to support arbitrary topologies (although this last restriction is being lifted by the deployment of BGP).

The size restriction has two phases. First, a key existing routing protocol (EGP) has a small limit on the total number of networks, but, as with the problems of topology, as BGP is deployed as a replacement, this limit will fade.

The second set (which consist of three far more fundamental problems) is that the system is running out of network numbers, the ability to route them, and eventually, out of address space completely. This topic is not treated here in detail [Chiappa91], but briefly, the first problem involves the accelerating use of the limited number of network numbers, the second concerns the difficulty of routing in a flat address space which is growing (very quickly, to boot), and the third recognizes the eventual limits of the 32 bit address.

Policy routing is the phrase used to describe external, administrative input to the data routing process. Currently, the system provides little to no support for this. The ad hoc mechanisms being used are so complex as to confuse all but the most skilled, and contain substantial potential for errors. The main problem is that the existing mechanisms are fairly unrelated to the desired goals, and the use of these mechanisms to produce the desired goals is thus complex and often impossible; they simply do not address existing policy requirements.

Policy requirements can be divided into three areas, which can be
called a) ‘access control’, control over which external users are allowed to use a given resource; b) the ‘trust model’, control over which external resources a given user is willing to use; and c) ‘information hiding’, control over which external users are allowed knowledge of a resource. While these divisions are not necessarily the theoretical minimum, they do appear to accurately follow the needs of the majority of users; providing these will make configuration of the system to provide the desired policies far simpler by virtue of the close match between what is desired and the tools available to do it.

BGP will do little to radically increase capabilities in this area. About the only additional capability provided by BGP will be that since complete path information to the destination is available, a limited amount of additional ‘trust model’ control will be available, as will some ‘access control’. However, as the actual controls are still basically those currently available, the ills of the existing system remain.

One thing to note is that while enough policy requirements have been verbalized to build a general model, no complete list of what will be required is by any means possible at this point. This means that whatever policy system is created will have to be able to add new policies as time goes on.

This latter requirement will be discussed in more detail later. Note also that the second policy class (trust model) implies control by the source of the route through the network. This also has substantial implications.

The current restriction on interconnection, which prohibits cycles, is untenable on reliability grounds (since alternate paths are eliminated) as well as difficult to enforce. As noted, this restriction is based on the technical limitations of the EGP protocol, and as BGP is deployed as a replacement, this limit will fade.

Finally, another problem noted with the current system is the use of toll networks. This includes not only the cost of running the routing protocols over such networks, but also distributing the information that certain paths cost money, and allowing traffic the choice of whether or not to use such networks. In some sense this latter requirement is a variant of policy routing, but in another it is yet another requirement, often labelled ‘type of service routing’, by which it is meant that different types of traffic needs links of different characteristics. For instance, remote login traffic needs low delay links, while bulk data transfer needs high bandwidth.

In the cases of both type of service, and access control, it is also necessary to allow for future expansion of the types of each supported, since it is unlikely that the complete set can be specified at this time.

2.3 Future Requirements

In the category of requirements that are not yet restrictive, but of which we are becoming aware, one very important one is to provide a system that is more immune to problems, of both accidental and deliberate cause. The fact that this system will connect effectively everything means that great reliance will be placed on it, and equally great confusion will be caused should it fail. Engineering failures need to be prevented by thorough simulation as well as use of large scale testbeds. Deliberate attack needs to be prevented by the inclusion of provisions for security; these must provide for the use of one of a (extendable) variety of optional security mechanisms, to allow the cost and level of security to be optimized depending on the needs and cost of failure.

Another important need is to minimize the amount of configuration needed, to minimize the possibility of having conflicting configuration information, and to prevent such conflicting information from causing a problem. The existing routing architecture suffers from being overly complex to configure. While this may not be obvious to technically sophisticated people with long association with the network world, many new users find the
existing architecture confusing and difficult to set up. While they perhaps expect too much, since you cannot set up a complex system with little or no study and engineering, improvements can be made. In addition, conflicting configuration information currently can cause severe problems; this another reason why such information needs to be minimized. Finally, the architecture must be robust in response the inevitable errors in configuration.

3 Routing and Addressing Fundamentals

After some consideration of the problem, it became obvious that the most difficult of the requirements is the size of the system. While some approaches make light of this problem, they do so at the cost of restrictions which make other stated requirements difficult to meet. The new routing architecture had not only to provide capabilities unheard of in the older system, but also do so in a very large system. Providing the new capabilities proved to be not so difficult, but the size problem was less tractable. It is not practical to always provide complete information about every detail of the connectivity and routing of the entire to everyone; some way of reducing the data was necessary.

At this stage it is useful to introduce some technical terms. ‘Compression’ is taken to mean reducing one set of routing information to another which, while more compact, is effectively the same; i.e. would produce the same routing decisions in all cases. ‘Thinning’ is taken to mean discarding some of the data to reduce it to a manageable size. In this case, routing decisions taken using the thinned data are possibly non-optimal, since some necessary information is unavailable. ‘Abstraction’ is a term including either of these two possible ways of reducing the volume of data. ‘Attenuation’ means that information becomes less complete the further you are from the source of the information.

However, before we examine this part of the problem in detail, it is necessary to review some fundamentals of routing and addressing.

3.1 Routing Algorithms

The family of routing algorithms with the most promise is the ‘link-state’ (LS) group. They are so named because the information that is distributed among the nodes running the routing protocol is about the existence and state of the connecting links in the system (i.e. a topology map of the system) rather than distances to destinations, as in the ‘distance-vector’ (DV) algorithms.

3.1.1 Distance-Vector Algorithms

In a DV algorithm, each node sends to all of its neighbours a complete table of its distance (measured in some agreed upon metric(s)) to all the destinations in the system (hence the name). For destinations it can reach directly, it advertises some low metric; for destinations it reaches through some neighbour, it lists what that neighbour told it plus some increment. Each recipient node then compares the advertised distance to the one it currently has for that destination; if what it just heard was better, it records it and routes traffic to that destination to the neighbour node that sent it the update.

There are many variants, which have been designed to add capabilities or fix problems with the algorithm, the most common of which are that DV algorithms tend to form temporary routing loops when the topology changes, before things stabilize, and that they are unable to detect that a destination becomes unreachable until the distance to it passes some arbitrary ‘infinity’.

3.1.2 Link-State Algorithms

In the simplest form, LS algorithms distribute a complete topology
map to each switching node in the system. This is done quite simply; each node generates a message describing the state of the links attached to it, and this message is then quickly flooded through all the nodes in the system using a reliable flooding protocol. A topology map can easily be constructed from the messages from all the nodes in the system.

These nodes may then run in parallel an algorithm which generates from this topology map a routing table which can be used to forward packets to various destinations. The MILNet is currently using a LS algorithm where the algorithm which computes the routing table is one due to Dijkstra called the ‘Shortest Path First’. [Dijkstra]

However, it should be noticed that the second step, creation of a routing table from the topology map, is purely an engineering decision. It is not necessary to precompute the entire routing table in an LS algorithm, whereas it is in a DV algorithm, since this precomputed routing table is the data that is passed on to the neighbour nodes. In an LS algorithm, it is perfectly plausible to compute routes on demand, from the topology map, and store them in a cache. This reduces the computational burden substantially, and, as we shall see, is a key element in the solution.

Also, in the classic LS routing scheme, the one essential is that all the nodes run the same route generating algorithm from a consistent map database, otherwise it is possible to form routing loops when two nodes disagree about the best ‘next hop’. However, if traffic is not routed using the ‘hop-by-hop’ method, both of these requirements can be relaxed; this is critical, as will be seen.

3.2 Routing Algorithm Choice

Several advantages were noted about the LS class of algorithm that recommended it to the ARPANet implementors [McQuillan]. Most of these are traceable to the fact that since the basic DV algorithm is a distributed algorithm for calculating routes, it is in many senses less robust and characterizable (in a response time sense, not an ultimate stability after a single change) than the LS algorithm, which runs in parallel locally on all machines.

First, it stabilized quicker after a topology change. This was important; since the ARPANet used load-based routing, the metrics on links changed fairly frequently. The speed was due to the fact that when a link changed state, that fact became know almost instantly everywhere (due to the fast flooding), after which the nodes all quickly recomputed their routing tables in parallel, at which point the routing had stabilized in the new pattern. This is in contrast with DV algorithms, where the effects of a change in the topology had to pass through computations in each node on its way across the network; additionally, the process might have to be repeated several times before the new stable pattern was achieved.

Second, it detected destinations which had become unreachable more quickly; this enabled the switches (which offered a reliable transmission service) to discard packets for the dead destination more quickly, avoiding buffer congestion from undeliverable and undiscardable traffic in the network. Third, it was less likely to form loops (formation of which had been a severe problem under the old DV routing algorithm, due to certain ill-advised modifications in the algorithm which had promised other gains), and quickly broke those which did form.

These advantages were not particularly useful to this application. Since the system was so large, it was felt that practical load-based routing was not possible. In such a large system there would be frequent topology changes in any case; there were concerns that with the added changes of load based routing that the routing would be less stable. The stability is certainly useful, but not as crucial when the metrics are essentially static. Likewise, the other two advantages, while notable, can be met with modifications to the basic DV algorithms.

The DV algorithms do appear have one advantage, which is that they
are apparently more suitable to routing in large nets. Since data is processed through the routing table before being passed on, they very easily can be modified to provide attenuation as a way of abstracting data. (The algorithm is quite simple; as soon as the routes to all the subelements of a routing element have the same next hop, it is no longer necessary to send out routing tables listing each subelement individually; a single entry for the element will suffice.)

However, this advantage is overwhelmed by a disadvantage, which is that they cannot easily be enriched to handle the problems of both policy based and type of service routing (which are technically very similar) without exponential explosion in the amount of data which must be computed and transferred. Since route calculation in DV algorithms depends on continuous calculation of all possible routes in a distributed computation, handling policy considerations means that routes to all destinations under all conceivably desirable combinations (or allowed combinations, depending on the system design) of policy requirement must be maintained. This is the only way to have a route available for a given destination/policy combination when traffic for that combination appears (or within any reasonable time of such traffic appearing, as explained below).

(A possible fix for this problem involves only computing a routing table entry for given destination/policy combination when traffic for that particular combination arrives. Before that traffic could be forwarded, however, the distributed algorithm would have to run, probably to stabilization (discovering when that happens would be a good trick in and of itself), to calculate the next hop at the first hop. This almost certainly presents an unacceptably long delay before the traffic can be forwarded.)

In any case, the distributed nature of the DV algorithm requires far more line bandwidth and computational power than the LS algorithm when policies are included; in a large network these alone will present substantial difficulties. Use of a DV algorithm in a world with policy routing requirements is simple infeasible.

A key component of handling the requirements is the realization that LS algorithms can handle this problem quite easily. [Chiappa84] Since each node has a complete map of the system, including all the links, it is quite easy to indicate on each link what ‘attributes’, of either policy constraints or service type, that link possesses. Since the LS algorithms can compute routes on demand, it is only necessary to create a route for a particular combination of requirements when it is needed. It is easy, and costs little, to tag each link with its attributes before the link information is flooded through the system.

It is this characteristic of LS algorithms, that the actual route calculation is separate from the distribution of the data, and can be delayed, that is the key one. The fact that the calculation can be delayed (which makes handling complex policy routing possible) is a fundamental difference between LS and DV algorithms, and a key reason that DV algorithms are fundamentally unsuited for use in the new IP routing architecture.

The problem with LS algorithms is that they do require a complete map of the topology. As explained above, this is impractical, due to the size of the target system. However, it is possible to modify LS algorithms to use abstraction, so this is the approach chosen. This decision brings other problems, though.

It turns out that compression alone is not an adequate means of attack on the problem. There are too many topologies that simply cannot be represented by a simpler but equivalent topology. Since the data must be reduced to make the problem manageable, it is necessary to discard some, and use thinning. The ramification of this step is that when routing data is thinned, the lost information means that in many cases the system will fail to detect possible routes with the required characteristics. This can lead to the creation of non-optimal routes, or in the worst case, failure to find any route at all, even though one does in fact exist.
The hardest problem thus becomes managing the discarding of information, based on cost-benefit tradeoffs which the protocol cannot possibly comprehend. Simply put, data must be discarded to make the cost of running the protocol reasonable. However, taking this step has a cost elsewhere, in non-optimal traffic routing; the problem is a classic cost-benefit tradeoff. Thus, thinning involves value judgements, and is thus a matter of policy as well as a technical problem.

The ramifications of these issues will be addressed later.

3.3 Addressing Fundamentals

Before moving on to review the proposed architecture, the final piece of background that is necessary is to briefly review the fundamental network concepts of addresses, routes, etc.

Although a number of papers have been written on the subject of addresses and routes [Shoch] [Saltzer82], a considerable lack of understanding of the basic aspects of these fundamental concepts still remains. Readers need to be clear in their minds the difference between fundamental concepts (and the objects which are the concrete instantiations of these fundamentals), and the different kinds of names (and the reasons for and uses of these names) which we may give these objects.

3.3.1 Object Spaces and Names

There has been confusion as to exactly what the fundamental object spaces of networking are; most previous architectures have tried to make do with too few. Previous work has also pointed out that common practice in networking has been to too tightly bind the forms of names for these objects to the objects themselves.

Two key object spaces used in this architecture are i) the node identifier, which indicates one particular computing node to which packets may be delivered, and ii) the network address, which specifies an attachment point to the network; a network interface to which traffic may be routed. (In fact, the term "address" is usually taken to refer to a particular kind of name for a network attachment point.) Multicast concepts are not considered here, since multicast can be built as a layer on top of this, and is thus less fundamental.

Each of these object types may have one or more kinds of names. The names may or may not have structure. Names with structure are invariably adopted to make some mechanism (often a mapping from that name to something else) easier to implement. For instance, a attachment point may be identified either by a unique, fixed length binary flat identifier (e.g. an Ethernet interface number in a bridged network), or, as is more usual, a structured hierarchical form (e.g. a typical IP address). In addition, there are mappings between names and objects, and between different object classes.

As was mentioned, problems in thinking about the issues of routing and addressing can occur when these object spaces are not clearly differentiated, or when the form of the name for an object is not separated from the concept of the object. An example of the former is the lack of distinction in the IP architecture between attachment points and node identifiers; this has left us with great difficulties in handling multi-homed hosts and mobile hosts. An example of the latter is that a network address is usually taken to be a structured hierarchical item, but this is in fact just one possible way of naming attachment points; a very useful name, to be sure, but not the only one.

Whenever one has different classes of objects, and names for them, several questions immediately arise. The first is "do we have the right classes", second "what is the mapping between the classes of objects", third is "how many names, and of what form, do we have for each class of object", and fourth "what is the mapping between the names and objects".
The mappings form the heart of the power, and the problems. Two aphorisms of computer science illustrate this: the first (due to B. Lampson) is "All problems in Computer Science can be solved by adding another level of indirection"; the second (due to D. Clark) is "The function of an operating system is to establish many different names for the same object, so that it may busy itself keeping track of the mappings between the various names."

Do we have the right fundamental objects? Exactly which names and mappings are useful in this instance?

3.3.2 Names and Mappings

It is clear that the two object classes named are highly useful. Are there any remaining problems, the solution to which requires yet another object class (and associated mappings)? Only one appears possible; the problem of hierarchical addresses implying hierarchical routes. This topic cannot be treated in full at this point in the document; the architecture as given solves this problem, but perhaps not in an efficient enough way. It may be necessary to add an extra object class to deal with it. For the moment, however, the answer appears to be that the two mentioned are enough.

As to the mappings among the classes, this also appears simple enough. Just as in the real world, a single node may have several network interfaces, a single node identifier may map to more than one attachment point. Also, just as hosts may move, the mapping from a node identifier to a network attachment point may change.

The only namespace for nodes which appears to be useful is a relatively short, fixed length, pseudo-flat binary identifier. This name is intended for solely for globally unique identification of nodes, and for efficient processing by automatons; this dictates the form of the name. (Since the space will probably be split up to various number assignment authorities for each of administration, it only appears to be flat.) Since it appears (for reasons to appear below) that the most useful form of the network address is long and complex, having at least one name for the destination in the packet which is easy to manipulate is desirable. There does not appear to be a use for a different kind of node identifier at the moment, but it could be added if need be.

This is obviously in addition to the existing domain names, which are hierarchical character string names for hosts. These names are obviously intended to be useful to humans, and the structure in them is intended to make interpretation by a distributed database (the Domain Name System) easier. It is entirely feasible that domain names which represent, for example, services could map to more than one node identifier, but that is a different issue.

The names of network attachment points, the address, present a more interesting problem. An address usually does two things. First, it uniquely identifies a connection point to the network. Second, it does so in a structured manner, which allows something useful to be done with it. Exactly what that structure is depends on what other uses it is put to; most often the structure of the address is used in routing.

Normally, groups of attachment points which are topologically related are given related addresses. Usually, this allows the number of different destinations which must be tracked in the various routing devices throughout the network to be reduced (in its simplest form, this allows routing tables to be smaller). That is, rather than keep knowledge of all the individual network attachment points, a routing device can keep information on collections of network attachment points, at a considerable savings.

It may have other benefits; in this architecture, it also does several other very important things. First, the structured address allows the point on the topology graph which the address names to be found easily, without mapping through any additional object class, or other attachment point name. Second, it helps provides a representation by which topology information can be distributed. Third, the structure of the address space provides a framework
for the abstraction process by which simplified graphs of the topology are created.

Finally, in the current version of this architecture, the address (which is heirarchical) in involved in creating those routes which take the least amount of effort to compute (sometimes called "no-brainer" routes), that route being basically heirarchical. (It is this latter function which may prove inefficient to overload onto the address of a network attachment point, necessitating the creation of a different object class or naming space for network attachment points.)

Note that all these functions are usually performed by a single kind of name, a structured address. However, other possibilities have been suggested.

One proposal is that each network attachment point be given a unique binary flat identifier; this would allow the creation of multiple, independant, structured namespaces (i.e. address spaces), each of which have a separate mapping into the first namespace which identifies all attachment points. The claimed advantages are that it would be possible to experiment with new addressing schemes, or introduce a new one, or even operate with several different ones in use at the same time.

This possibility has been discarded in this architecture, however. The whole point of an addressing system is to allow traffic to flow between all the myriad parts of the Internet. To do that, it is necessary to pass around routing information, and the form in which this is passed around is inevitably that of addresses. A new address scheme, resulting in an address form which the old routing architecture did not understand, would inevitably interrupt this flow.

Moreover, should it be necessary to supersede the addressing scheme in this architecture (if adopted), it would be just as easy to create a new mapping from node identifiers to attachment points as it would to create a new mapping from attachment point names to attachment points. Alternatively, mappings could be created from old attachment point names to new ones directly. No good reason can be seen for the flat namespace for attachment points, so incurring the cost of it should clearly be avoided.

4 Routing and Addressing Architecture

It is necessary to consider some ramifications of some of the requirements in detail, and the steps taken to meet them, before the entire architecture can be outlined.

4.1 Requirement Ramifications

The two major ones have to do with the trust model requirement, and the ability to expand the attribute list.

Briefly stated, the trust model problems is that some users may have policies on which links they will and will not use that cannot be described in anything less than a general purpose computational notation. Moreover, in the most extreme cases, some users may not trust outside agents to pick a route for their data, even given a complete statement of their requirements. This means that computations in intermediate nodes cannot be used to route traffic. This flies in the face of the current architecture, which assumes traffic is routed on a hop-by-hop basis; i.e. each switching node makes an independant decision on what to do with the traffic.

The attribute list problem is a more general consequence of the specific observation above, in the discussion of the requirements, that it is probably not possible to completely specify what policy attributes will be needed at the moment. If the new system is to last, it must be able to change and expand its capabilities. To do this, it is necessary to be able to invent new types of attributes for links, and phase them in incrementally (since such changes cannot be coordinated instantaeously across a large system).
This presents a problem, since this seems to violate the restriction given about in the discussion of LS algorithms, namely, that all the nodes run the same route generating algorithm. If one node is taking certain attributes on links (and requests to use them in packets) into consideration when generating routes, and another is not, this can potentially form routing loops.

These two problems appear substantial, yet both can be solved by the same mechanism. In addition, that mechanism allows yet more problems to be solved, as detailed below.

4.2 Architectural outline

The general outline of the architecture is as follows (each element will be discussed in more detail below).

The routing architecture will be a multi-level area (i.e. heirarchical) LS system; in the topology graph representation of the network, links and nodes have attributes. The exact algorithm for generating routes, given the topology map, is not specified as part of the protocol (but a default algorithm can be provided as an appendix). Abstraction of data will be determined primarily by external mechanisms (although a simple default will be provided either as part of the protocol or as an appendix), and in any case clients are not bound to accept any abstraction unconditionally. Routes will be completely determined by the initiator of the traffic; i.e. all packets will be source routed. Packets will belong to ephemeral associations called ‘flows’. (There may be other forces acting on the architecture which make flows desirable; if so, several things can be done with this new mechanism.)

A new kind of address will be created, for use in making routing decisions. The existing IP address field will be retained as well (exactly as is in the short term), but the contents will be put to slightly different use as a node identifier.

In the short term, hosts will continue to operate with no changes at all, and use the existing packet formats for their interactions with routers, although there will be certain optional optimizations that speed things up, but are not necessary. In the long term, a transition to a longer version of the node identifier (the old IP address) will be necessary. A number of factors will probably eventually combine to cause the definition of a new IP packet format, but most of those forces are outside the scope of this document; however, phasing this longer version in will probably occur as part of that transition.

A key thing to notice about this architecture is what is not part of the architecture and protocol. Neither the algorithm to create routes from the topology database, nor the exact method by which abstraction occurs are part of the architecture. These are also the two hardest problems, and the two in which future work is most likely to bring improvements. It is a key feature of this proposed architecture that these two areas can be left outside the scope of the protocol; future improvements can be easily phased in incrementally.

Route generation in this architecture will need routing algorithms which have a different goal from most existing work, such as the Dijkstra algorithm. Most known routing algorithms create an entire routing table of optimal routes to all destinations; since this has been what was needed in the past, this is the area in which most research has occurred. The problem of finding the optimal (or a good approximation thereof, within limits) route between two points in a graph, ignoring other destinations, has not been a topic of much work. This only increases the need to leave flexibility in this area; improvements in algorithms and heuristics tend to be slow to appear.

The things which are part of the routing architecture and protocol are a way of representing and characterizing the network topology, a way to distribute this information, and a way to set up traffic flows. Everything
else can be left and specified later; a key to the power and flexibility, as well as the ability to last (gracefully), of this architecture.

Upon some reflection this seems likely to be a theoretical minimum for performing those tasks. It is not clear whether the fact that the two hardest problems are left outside is fortuitous or a reflection of some deeper correctness. In any case, the minimal design does limit the amount of work to be done, as well as limit the possibility that something will be done wrong or in an inflexible way.

"Perfection has been attained, not when there is nothing left to add, but when there is nothing left to take away." [Corbato]

4.2.1 Multi-level LS Algorithm

The LS class of algorithm is chosen, basically for the reasons listed above in the discussion of these algorithms. It is really the only practical way to handle a system with complex access restrictions on links which form a key part of policy considerations. The expandable attribute list allows future growth.

The algorithm has to be multi-level to handle the very large amount of link state in the system. Even if the system were restricted to distributing information about AS’s, the general consensus is that a single level will not be able to handle the number of AS’s we expect to be deployed in the reasonable future.

However, it is also felt that the time and need for AS’s, and the strict division of routing in inter- and intra-AS, is past. AS’s are a limited and simplistic mechanism that was created long ago to fulfill certain limited goals, primarily that of providing routing firewalls; since the new architecture can do all these, and better than the AS concept, it is now entirely appropriate to discard AS’s. (This does not mean that the concept of policy groupings would be discarded; far from it. What it does mean is that there will not be a particular layer which is called out for special treatment.)

The primary reason for the introduction of AS’s was to provide some routing firewalls in the system. Since the new architecture will problem more flexible and robust firewalls, retaining the archaic AS mechanism is pointless. An additional advantage of AS’s (and the division of the task of routing between IGP’s and EGP’s) was to allow different routing technologies to be used. The extreme power and flexibility of the new architecture makes this less useful, and in any case, the fact that the process by which the representation of an area is constructed (the abstraction process) is not in the scope of the protocol means it would still be possible to use other routing technology within an area, and pass the results of that process up as the representation of the topology of the area.

The new system would be responsible for all routing in the IP architecture; it would be able to depict, at its lowest level, the physical transmission assets of the system. This unification of routing would not only reduce the complexity (and likelihood of problems, implementation and configuration cost, etc), but bring the full power and flexibility of the new architecture to bear at all levels. A number of problems exist today because of the split between inter and intra-AS routing; removal of this artificial barrier allows greater flexibility.

The use of a link-state architecture does bring with it the attendant difficulties of how to abstract link-state data. The use of thinning, necessary for useful abstraction, brings with it the difficult task of choosing which data to discard, and the problems caused by loss of the discarded information. As noted above, this cannot be solved by purely technical means; value judgements must be made about which information to discard. It will be usually be necessary for the administrators of any given area to help decide what the representation of that area will be when the internal details are hidden.
Since this is now a problem of entering configuration information, it is desirable to do this in a way that allows no chance for inconsistent data. A default algorithm will also be provided, for those who do not care what the representation of their area is, or are unsophisticated and cannot participate in choosing the representation of their area. The fact that there is no specified algorithm for doing this also allows improved default algorithms to be incrementally deployed as they are conceived.

However, the problem of non-optimal or unfound routes remains. The solution to this part of the puzzle requires the next key component.

4.2.2 Source Routing

The source routing is a necessary consequence of both the trust model part of policy considerations, and the expandable attribute list, as outlined above. Having the source set the route avoids both of these difficulties. As mentioned, it also allows the incremental deployment of better algorithms for creating routes as ongoing research provides them, and also provides a means to attack the non-optimal route problem, as noted immediately above.

Note that the route (nor indeed, the new style address) is not necessarily carried in each and every packet; this is an engineering decision to be made on the basis of the costs and benefits of doing so. The costs are primarily the increased overhead of carrying around the extra data. There is another issue in that the existing IP packet format will probably not allow ‘on the fly’ modification to hold the source route, since the latter will probably be too large to fit into the IP header. Depending on the length of the addresses, these could be a problem as well. This could be solved by ‘wrapping’ the packets, but at considerable cost in complexity and switching speed (and some in line utilization).

The benefits of carrying the source route and/or address in each packet are that the routers may remain more stateless. It is anticipated that routes will not be carried in packets, and there will be a ‘route set-up’, as part of the ‘flow set-up’, which happens invisibly to the client of the packet layer. This flow will not be critical state, like a connection, but rather a hint. (A ‘hint’ is data which allows processing to be optimized, but which is does not have to be present to allow correct operation.) This may be recreated by the routers without reference to the source of the flow setup should any intervening node fail.

Obviously, since the source (or an agent close to it) choses the entire route, it is trivial for the source to enforce whatever constraints it wishes on the path that its traffic takes (even constraints which are not stateable in anything less than a general purpose computational language). Alternatively, if the agent computes routes using links with attributes it understands, but which the intervening switching nodes do not, the traffic will be routing correctly, their lack of comprehension notwithstanding.

Also, if in computing a route, the agent generating the route comes across an attribute it does not understand (either because it was recently introduced, and the agent does not yet know how to deal with it, or because it is a private attribute which the agent will never understand), the agent will still be able to compute a route. If all attributes carry some limited information about whether they are restrictive (certain types of traffic will not be allowed across) or informational (giving information about the link) it would be possible for a route to use links which have attributes the agent does not understand.

Since the entire route is calculated by one agent, it clearly does not matter if different agents use different algorithms for generating routes. Deployment of new algorithms, or a choice between several existing ones (for example, one which is fast but which does not produce good routes would be useful for one-shot traffic, whereas more complex and expensive ones would be appropriate for long-lived flows sending a lot of data)
thus becomes trivial. [SalzterSR]

4.2.3 Local Abstraction Control

As for the non-optimal route problem, it was noted above that the use of hop-by-hop routing required both a consistent algorithm and a consistent database. Discarding hop-by-hop routing allows relaxation of the latter requirement as well. This allows a powerful (and previously unconsidered) adaption of the typical operation of an LS algorithm. Canonical LS algorithms require everyone to have an equivalent map.

("Equivalent" is used instead of "identical" since in existing multi-level protocols such as OSPF, agents in different areas will each have (identical) top level maps, as well as maps of their own area, but of course these latter maps are different. All routers within a single area, however, will (and must) have identical maps.)

However, there is no requirement in this architecture that this be so. Neighbouring routers at the same level may have wildly different maps. One might have only a default representation of a neighbouring area; another, which is sending a lot of traffic into that area, might have detail on the internal connectivity of that area, to allow it to pick the optimal entry router into that area for the traffic it is handling. Any node may call for more detailed topology information about any part of the system it wants, provided that that information is accessible to it, and the cost of providing that information are borne somehow.

However, the implications of lifting this restriction are considerable. As noted above, the thinning problem is essentially a cost-benefit tradeoff; making this tradeoff at the area which is being described imposes the area’s idea of the correct cost-benefit tradeoff on everyone. However, allowing each route generator to make its own decision on how much detail to keep in its maps allows this cost-benefit tradeoff to be made at the client, i.e. with far greater flexibility.

It will be necessary for the costs of keeping maps with more detail to be shifted to those agents which wish to keep them; this will require attention at the time that accounting and resource usage algorithms are brought to the network.

The default abstraction algorithm is thus seen as that which will provide the theoretical minimum of data necessary to route traffic through the system. The actual routes taken by traffic under this algorithm will thus be strictly heirarchical. Looked at another way, local abstraction control thus allows a way to do non-heirarchical routing.

5 Details of the Architecture

Many of the components necessary to this routing architecture and protocol can be ‘borrowed’ from the work being done on the Open Routing system, and similar LS systems such as the ARPANet algorithm, OSPF [Moy], etc. These include the mechanisms to do reliable database distribution and flow setup. According, description of these solved problems will be skipped in this document.

This section includes more detail on certain key parts of the architecture, along with calling attention to those parts in which engineering questions remain.

5.1 Multi-Level Topological Representation

The first step is to consider the maps (graphs, actually) which represent the actual detailed topology. It is worth noting that in the graph representation of typical networks, there are usually two kinds of nodes; ‘network’ nodes and ‘router’ nodes. The arcs then represent network interfaces. It is possible to have only a single kind of node (either
networks or routers), but this is artificial, loses valuable information, and is more difficult to work with in practice than the more complex version. This architecture will almost certainly use this type of model.

In the maps, it will be possible to take an contiguous section of the graph (i.e. some number of nodes and their connecting arcs), and reduce it to a simpler representation. Such a reduced section is called an ‘area’. Areas may be nested, and areas have as an attribute a level; an area which contains another area has a level one higher than the contained area.

It is unclear whether an area will need to be represented as a section of graph, or just a node. In either case, a new type of node for an area is probably needed. The issue probably hangs on whether the rules for constructing topologies demand router nodes between area nodes or not. In the former case, the minimum representation of an area would be the collection of border routers plus a pseudo-node for all the rest of the topology of the area, to which the border routers are connected. If the area has complex transit policies which vary depending on which pairwise set of border routers are picked, a more complex representation allowing this to be expressed would of course be necessary.

It is not clear whether it is best to allow only ‘strict’ reduction, in which an area of level k would be allowed to contain only objects of level k-1, or whether ‘loose’ reduction, in which objects of varying levels less than k could be contained, is preferable. The latter is clearly more flexible, but might make the engineering more complex. This decision can probably be put off until the detail design happens. What is important is to realize that the division into areas provides the framework for the default abstraction algorithm; agents outside an area will not in general have detailed information on the internal topology of the area.

On the subject of border routers, it seems wisest to set the rule that all routers belong to a single area. This is a good match to reality, where most of the physical assets have an owner; the half-router model has proven to be less useful in practice.

One aspect of representation that needs to be addressed is the question of configuration errors; this is the most likely place for such errors. One technique that appears useful is to simply indicate to each router whether or not it is a boundary router for a k level area, not which k level area it belongs to [BBN]. Thus, misconfiguration of a router simply joins two k-level areas together, but has no other ill effects on the routing. Similar approaches must be taken throughout this aspect to minimize configuration and prevent configuration errors from being a major problem.

In any case, it is clear that some decisions about open questions need to be taken about the multi-level topology representation before any other parts of the design can be worked on in detail. Obviously, these decisions may need to be revisited if work elsewhere reveals that a poor choice has made the design of some other part more complex than need be.

5.2 Multi-Level Addresses

The address format chosen will need to map closely onto the representation of network topology, which in turn is chose to make the job of the routing simple. Remember, the address is simply a way of naming an attachment point to the network, in other words, specifying a place in the graph, and it is essential that given an address, it must be easy to correlate that address with its location in the topology graph easily. (It might be argued that the ‘address’ need not have that property, but it might map to something else which does. All that is saying is that terminology has been confused, and the wrong name for a network attachment point has been called the address. The final concept will inevitably have to be one which is easy to locate.)

In essence, what is proposed is a multi-level hierarchy, with a
variable number of levels, each of variable length. This may seem like overkill, but remember that these addresses are only needed to create flows, and since the cost is minimal it is better to have too much flexibility than too little. The mapping from addresses to the map is fairly simple; the first element of the address identifies which top level area an address is in, the next element which of the next level areas within that top level area, and so on.

In any event, the bottom level will be the level representing real physical assets, and numbering will proceed up from there. The advantage of this is that two different systems with differing address spaces, each allocated without reference to the other, can be joined by the simple expedient of creating a new level above the top of each existing system and making each system appear as a simple object in that level. Alternatively, a small independantly numbered system could be joined onto a large existing system as a branch at the appropriate level.

This will also avoid useless arguments about whether or not something deserves to be a 'top-level' area or not. Since there is no such thing as a top-level area, pointless fights over status should mostly be avoidable. Note that good functioning of the resulting system demands proper engineering of the topology, and of the areas; since the area definitions provide the framework for the abstraction process, which in turn affects the default heirarchical routing, a poor choice of area boundaries will cause the default abstraction process, and the default heirarchical routing, to work less well, although it will continue to function.

The analogue in the addressing area to the question of only allowing 'strict' areas is whether or not networks (and presumably hosts and routers as well) can have numbers at any but the lowest level. If a k level area is allowed to contain physical assets, presumably they can be addressed with k-1 level addresses, without going all the way down to the bottom level.

The question of exactly what form the addresses of networks and routers (if any) take is open. In the current architecture, no real provision was made for numbering networks, although the convention of a zero 'rest' field has come to mean the network. Routers are currently indistinguishable from hosts (which is good and bad). In the new systems, hosts will not have addresses per se; the lowest level item which can be addressed is a network interface, and presumably typical hosts will have one each.

Once again, choices have to be made, but with the understanding that they may have to be revisited in light of work elsewhere on the architecture.

5.3 Topology Information Flooding and Route Generation

It is next necessary to consider the question of which devices perform these functions. It seems most plausible that the first function is co-located in the routers themselves, for reasons of fate-sharing; connectivity information is flooded through the very connection elements that are being described. This also removes dependency loops where topology flooding agents which are not routers are required to make contact with neighbour topology flooding agents through routers.

Route generation could, however, quite easily be performed in separate devices, which communicate with routers (both local, and distant if they wish to have detailed information about distant areas). It might be useful to build this function into routers, but this is an implementation decision. Certainly, clients with complex policy requirements will probably desire to generate their own routes.

5.4 The Default Abstraction Algorithm and Local Abstraction Control

Given the discussion above, the default abstraction algorithm is very simple. As stated above, the division into areas provides the framework for
the default abstraction algorithm. The routers which are border routers for an area are responsible for doing the abstraction on the area topology when it is advertised outside the area.

Two possibilities exist for a simple algorithm to provide a default representation of an area. One, the \( N^2 \) model, maps the area as a complete set of connections between all the border routers. This model can fairly accurately represent the true characteristics of the connection between any two pairs of routers. (Of course, how to do so in the presence of policies and types of service, where a multitude of possible connections are possible, will require a bit of work; probably the least policy-limited path should be advertised.) In the other, the pseudo-node model, all the border routers are connected to a single node in the center. This cannot accurately represent the metrics between any two border routers, but an averaging system will give a good guess.

The option is always available to go to a representation with human input, or to a representation prepared by some other algorithm. Some control needs to be created to prevent areas from flooding the system with unnecessarily complex representations of themselves.

Of course, if an area is represented solely by a single pseudo-node, none of these problems arise, which is the attraction of this possible way of representing an area. The disadvantage is that then areas must have transit attributes which mirror the attributes (both policy and type of service) on the actual transmission links which connect the border routers.

The working of the local abstraction control (somewhat inelegantly named the "Blister Model") is simple to understand in this context. Generally, any routing agent outside an area is given, as a default, whatever representation that area has decided to purvey. However, should a routing agent wish a more detail of a non-local part of the map, nothing is to stop it (other than information-hiding access control, of course) getting the information and upgrading its map.

5.5 Virtual Links and Flow Repair

One concept which will probably appear as part of the representation of an area is that of a virtual link. Where it is desired to describe the connection between two border routers, without providing detail on the constituent links which actually connect the two, a virtual link (with the attributes of the complete path between the two) could be advertised.

One key aspect of this idea is the ability to do local repair on flows when topology changes happen. If a virtual link at level \( k \), which was advertised to a source route generator and which is used in creating a flow, suffers a component failure, two outcomes are possible. First, the virtual link may be reconstituted with the same attributes using other lower level assets. In this case, the repair would be local, with no necessity for intervention on the part of the source route creator. As an option it might be notified if some attribute, such as delay, increased more than an allowable amount on some component it used in setting up a flow. Second, if such a reconstitution cannot be made, the \( k+1 \) level virtual link of which the \( k \) level link is a constituent is notified, and the algorithm is then run recursively at a higher level. Only if some asset which the source route creator fails would it be necessary for the creator to select a new path for the flow.

5.6 Partitioned Areas

Handling of partitioned areas needs to be addressed. A number of methods are possible. One obvious method is to reconnect the two parts of the partitioned area by going through another area.

Another elegant method gives each level 1 area a globally unique identifier. A \( k \) level area is given the identifier (at \( k \) level) of the lowest (or highest, or some unique algorithm) numbered \( k-1 \) level area within it. Then, if a \( k \) level area partitions, it automatically creates a new \( k \) level
area, which is automatically numbered from its constituent k-1 level areas. The disadvantage of this is that the address of everything within the new k level area has changed, which is probably a substantial disadvantage (although not insuperable).

5.7 The Node ID

The node identifier (what used to be the IP address) is a key part of the system; not only does it provide new capabilities, but it makes the new system incrementally deployable. Instead of being a number with structure, as it is currently (which is causing inefficient use of the number space), it is simply a unique 32-bit (initially) number. That way, we put off the day when that address space runs out, since we can use it efficiently instead of wasting large gaps, as we do now.

A number of outstanding issues, particularly mobile hosts and multi-homed hosts, can be solved with this, with no change to the upper levels. An open TCP connection to a given node identifier will stay open even if the node moves and gets a new address. (The flow may need to be rehomed, but this will be invisible to the TCP, and perhaps to the host entirely.) If an explicit mapping exists from a node identifier to multiple addresses, many issues having to do with multi-homed hosts (both for reliability as well as bandwidth) become much simpler.

The source and destination of packets in the network will continue to be identified by these identifiers. If we chose not to include the new addresses in each packet (as seems likely), a packet arriving at a router is assigned to a particular flow by inspecting these numbers. This process will be at least as quick as current routing, and if the addresses are not included in the packets, there will be no extra line overhead either.

5.8 Mapping to the Address, from the Name and from the Node ID

Obviously, a system for providing the new addresses, and mapping between them and other identifiers, such as host names and identifiers, needs to be available. It clearly ought to be distributed and replicated. We already have a distributed replicated system for containing mappings; the DNS. It would probably be simple to add this mapping as well.

When going from the string name, the obvious thing is to return the new address as well. For clients which need to map from node identifiers to addresses, some analogue of IN-ADDR will have to be provided.

One thing to be careful of is that in the process no dependency loops are formed. For instance, error reporting might need to get an address if all that is on hand is a packet with the node identifier. The actual process of getting routing information around would have to be examined carefully to make sure it does not rely on the existence of this mapping system, otherwise dependency loops will certainly form.

5.9 A Sample Route Generation Algorithm

One simple way of generating a route in a complex policy environment is to run over the map, dropping all links which are unusable for policy reasons or because they do not match the type of service desired. It would then be possible to run the Dijkstra algorithm to create a routing tree, and thus a route, for a given metric which is it desired to minimize (probably delay or cost). To optimize several metrics at once, a formula for weighing the metrics together to create a composite metric needs to be provided; as each link is examined in the Dijkstra, the composite metric will be calculated from the exact metrics of the link, and the composite metric will be what the Dijkstra minimizes.

Metrics such as bandwidth, or MTU, where minimization over a path is inappropriate, would be handled by dropping inappropriate links in the first phase; if no route can be found, the actual requirement on this metric could be lowered and the process repeated to see if a route appears when a less
optimal value is allowed.

5.10 Authentication and Robustness

Making the system robust against attack is vitally necessary. One approach would be to tag all data with a private key system; this guarantees that topology information could not be modified as it is flooded through the system. The same approach could be used in flow setup; acknowledgements tagged with the private key would prevent the flow from being hijacked to a different destination by a corrupted switch (although it could be copied).

Protection against denial of service attacks are more difficult. In the above example, the corrupted switch could prevent the flow from being set up, although it cannot make it appear that the flow has been correctly set up when it cannot. Of course, if the switch refuses to set up the flow, an alternate path could be picked which avoids that switch.

Another difficult problem is bogus connectivity information. Some checks can be made, such as ensuring that both ends of a link agree that it exists, but this will still not catch collusion. Once again, it can be detected that this is happening if the flow does not set up correctly, and the errant switches avoided. However, if the switches claim a direct connection, which causes traffic to flow to them, and then route the traffic through, it will be difficult to detect except by measuring the service provided.

6 Possible Optimizations

Although this document is not an engineering design, there are a number of possible optimizations which are worth discussing here. Many other such optimizations are possible. They have not been considered in detail here, since they are generally low level engineering and depend on the exact details of the structure chosen. The key focus in this document is to discern the hard problems, and pick a broad line of attack on them.

6.1 Default Routing Tables

One useful optimization which might be included is to provide a facility for handling traffic without the overhead of calculating a route and setting up a flow. [Estrin] Needless to say, this could only be done for a limited number of classes of traffic, but it is possible. Basically, a way would exist to indicate that traffic was not part of any flow, and one (or a small number) of default routing tables for such traffic would be precomputed, in all routers, as is the current practise.

This would not be unreasonably expensive. For example, assume that the system contains 5 levels, and the fan-out at each level is on the order of 200. Thus, the system could handle up to 200^5 networks, or 3x10^10 networks; this should be large enough for the reasonable future. Such a system would only mean that the average router with a minimal map would need a map with 5*200*(3+1), or 4000, nodes (assuming that routers outnumber networks/areas by a 3:1 ratio, which seems about right from the current system; this gives triple connectivity to each network, or fairly redundant connectivity). Assuming each router is connected to 3 networks, then the map would contain 5*200*(3*3), or 9000, arcs. Such a map would be fairly easily handable by even the current generation of router hardware.

The problem with this idea (within this architecture) is that a mapping must still be created from the destination node identifier to the address. How does this mapping happen? One possibility is to have first hop router (or the host) add it to the packet. Alternatively, each router along the path would have to do the mapping. Clearly, the mapping in each intermediate router could be installed via a setup, but this would just get us exactly what we are trying to avoid!

Also, it is not clear if this optimizations will be useful in the
future. It remains to be see whether if, in a system in which policy configuration and access controls are much easier, and thus more common, much traffic is sent out in the default class, and if the free, general access model of network use we have enjoyed continues. If not, the concept of default routing tables is probably not a useful one, since almost all traffic would need a flow set up to handle it.

6.2 Map Discarding in Stub Areas

One additional idea, if default routing tables are included, is to support stub areas. The complaint might be raised that even supporting a map with 4000 nodes (from the example above) is pointless if an area is only singly connected to the outside world, or neither wants to support transit traffic nor pick optimal area exit routers. In these cases having the topology map for the rest of the system outside the area is unnecessary; the traffic is simply sent to any border router, and is routed from there.

(The former simplification is obvious; doing either of the latter two means that traffic must be able to differentiate as to which border router is the best one to head toward; i.e. traffic inside the system must have access to a full outside map to know which border router is the best exit router, if routing loops are to be prevented.)

6.3 Flexible "No-brainer" Routes

One major potential issue with the current architecture is the fact that the simplistic "no-brainer" route is that which is computed with the minimal map, and that minimal map implies strict hierarchical routing.

The problem with that is that a K level area, such as a campus, which in actual physical terms has connections to several long-haul networks, each of which is probably a separate K+1 level area, has an insufferable choice to make. Depending on which K+1 level area they associate themselves with, any traffic using the simple "no-brainer" routing will get to that K level area by means of the long-haul network with which the K level area is associated in its global address. This is the problem referred to some while above, where it was indicated that perhaps using the address to generate the "no-brainer" route is an overload of that name which finally breaks down.

One solution to this problem involves the use of what are called "route suffixes" [Clark]. When the host-name to address translation is performed, the address which comes back comes with several incomplete source routes, which represent potential useful paths, terminating at the destination, through the nearby topology. If the most detailed (last) item in the route suffix is not know to the source, it backs up and tries the previous (less detailed) item, and so on. Given several route suffixes, probably one for each major long-haul network which an area is attached to, the source can easily make a determination as to which long-haul network it prefers to use to get to the destination.

The major advantage of this scheme is that it is fairly efficient; a small amount of extra data is passed back at the time of the address lookup which allows a choice to be easily made. What mechanisms are available in this architecture to do this?

One rough equivalent in this architecture can easily be found by allowing each network attachment point to have multiple addresses; this would indicate that a destination could be reached via multiple long-haul networks. (This is equivalent to allowing K-level areas to overlap.) This solution is not preferred since it conflicts with a previous goal of the address, which is to provide a way of concisely describing the topology. If any network attachment point can have many names (addresses), this will be more difficult.

In effect, this approach overloads yet one more function onto the address, which is to optimize picking a slightly more optimal route; the
address is not really suited to having to support this one further function, however. There are a number of ways to do this which do fit well within the architecture, though.

The first is a slightly more general variant of the original scheme, which is to use the local abstraction control mechanism to discover some details of the topology around the destination. This is seems to be less efficient than the "route suffix" mechanism, but has the same effect; the source is given more information which allows it to make a choice.

The second, in some senses a variant of the first, is to remove from the map all areas which do not allow transit traffic; i.e. "stub" areas. This would greatly diminish the number of nodes in the map, and allow more detail to be kept. Whenever a source needed to send traffic to a destination in that stub area, it could pick up topology information about that area and enter it into the map.

Both of these represent a slightly more formal way of doing essentially what the route suffix does, which is provide more routing information specific to the destination. The difference is that the information in the route suffix is picked by the destination, as opposed to the source being given general information about the topology near the destination.

One final option is to provide essentially exactly the route suffix mechanism. As alluded to above in the discussion of fundamental object classes, it might be necessary, if this optimization is deemed important, to provide a new object class, that of route suffix. Rather than overload this function onto any of the existing object classes (and the names for those objects), it is perhaps best to simply bit the bullet and create the new object class.

7 Deployment

A great advantage of this architecture is that it can be deployed in an incremental fashion, and furthermore that it does not require any changes in hosts for fairly full operation (although minor changes would improve things somewhat).

The following section is a first crack at defining how the deployment would work.

7.1 Incremental Deployment in the Routers

The system can fairly obviously be interoperated with the existing architecture provided host identifiers continue to be allocated along the existing lines of network numbers. This will allow the system to be deployed and brought up incrementally in the routers.

In an old-style router, it will still be given lists of network numbers, and the metric to each. In new-style routers, areas of the network which are being serviced by old-style routers will be masked by 'transitional' routers which provide a mapping from the old routing protocol to the new system.

Of course, as the number of networks mounts, the existing routing will break down, and render the old-style routers non-functional. By that time, basically all the main routers should have been converted to use the new system, and the old routing mechanisms can be turned off. Stub routers which use default only can continue to operate as before, obviously. Once the old-style routing mechanisms are turned off, allocation of host identifiers can proceed in a more flexible and space efficient form.

Obviously, old-style routing could continue to be used once this happens, but hosts served by old-style routers would probably not be able to get access to the new hosts, since the new host identifiers would not contain any topology information the old-style routers could use to
route the traffic. One possible way around this is to have all traffic to what looks like new network numbers send to a new router which would perform the mapping in a similar way to the way hosts are supported.

7.2 Incremental Deployment in the Hosts

For the first stage, no changes would be absolutely necessary in the hosts at all. The host would translate from the name to the host identifier, and send out an existing packet to the first hop router.

The first hop router would then do the flow setup, using some default policy attributes. The destination network address would either be looked up separately, with attendant overhead, or, if the router overheard the name lookup, it might have cached it.

The issue of network masks and node identifiers needs to be thought about to make sure that getting from hosts to neighbouring hosts, as well as first hop routers, continues to work.

Clearly, if node identifiers are assigned without respect to topology, then the mask and match process can no longer be used to determine whether or not a given destination is on the local wire. Doing anything else means that the address space will not be allocated inefficiently, bringing us back to where we were! However, given the practical difficulties involved in true random allocation of node identifiers, it may be necessary to allocate in blocks.

One issue is involved in the translation of node identifiers to local hardware addresses. Some possibilities exist involving use of ARP, but these are nasty. For networks which depend on direct mapping from the node identifier to the network address, it is obviously unavoidable to allocate chunks of the node identifier space.

Another problem involves making sure that each host understands how to get to its first hop router for off-network traffic. If hosts exactly follow the Host Requirement RFC, and are set up with an all ones subnet mask, then any attempt to send traffic to another host will perforce go to a router. The default router table will have been filled in via the Router Discovery ICMP mechanism, and the router’s local network address will be resolved as discussed above. This is inefficient for traffic on the local network, however; if the all ones mask approach is used, routers will also have to stop sending Redirects!

If a proposed document on routing in the host is worked on (or alternatively a revision to the Host Requirement RFC is put in hand), this topic should be carefully considered to make sure the proposed mechanism will integrate well into this new approach.

Eventually, the process could be optimized by slight changes in the host. It would ask for and remember the new type address when it contacted the name server; if it had special policy requirements (or charge authorizations, or whatever) it would include these in the packet it sends out, either in the initial connection packet or in a special communication to the first hop router, or whichever box is doing the flow setup.

7.3 Upgrading to a Long Node ID

A 32-bit system wide node identifier is clearly unsuitable in the long run. One possibility for the latter is to make the UID’s locally (perhaps pairwise locally) significant only; however, this is complex, and loses some of the advantages of having a node identifier.

The other main possibility is a new IP packet format with 64 bit UID’s. A new packet format might be desirable anyway, for other reasons. If the 64 bit UID path is chosen, and the phaseover started before the 32 bit space is used (so that the UID of any node in the new 64 bit space is simply the zero extension of its UID in the 32 bit space), there will not be any cases of new and old style hosts which cannot communicate due to the inability
of the old address space to name hosts in the new space, which is the usual cause of problems in conversions. Once basically all hosts have been converted to the new format, use of the rest of the identifier space can proceed. This is probably the preferable path.

A number of ways of interoperating old and new style hosts exist. The first hop router might automatically convert old-style packets to new style. The advantage of this is that hosts can throw out the old code; the disadvantages are the cost of the conversion to and fro when two old-style hosts are conversing. Another possibility is that hosts might continue to understand both old and new.

In any case, there are four cases to be considered; old-old, old-new, new-old, and new-new (the first being the source and the second the destination of the packets). The tricky one is the new-old, since in any system something must convert the format; either the source host needs to find out that the host does not understand new-style (perhaps by trying a new-style first and seeing if it gets a response) and send an old-style, or the last-hop router, which needs to know which of its hosts are new and which are old, needs to do the conversion. A messy problem however it is sliced! One useful trick is to use spare bits in the headers to record packets which were sourced in the other format, or hosts which can generate and understand new format packets, etc.

8 Conclusion

The system proposed above consists of many elements, with the utility of each created and increased by the interaction with others. The way in which they were put together was not as straightforward as it now appears. A long process of revisiting of requirements and tinkering with the design has resulted in the complete structure now laid out, with each piece dependant on others and part of a deeply integrated whole.

The system also contains a number of old ideas, put together in a novel way, together with the addition of some new ideas. In both of these aspects (the extreme interdependence of a limited set of ideas, developed over time, together with the reuse of existing good ideas with some augmentation) it resembles the process which created Unix. [Thompson]

It is believed that the proposed architecture detailed above meets all the goals outlined in the requirements, and in particular the size, making possible a extremely long (essentially indefinite) use of the IP architecture.

Finally, it is worth noting again the many instances here in which the power and flexibility of the architecture are improved by leaving something out (route generation, data abstraction, etc), rather than by adding something.

References (Incomplete)

[BBN] ???; "Adding Areas to the ARPANet Routing Algorithm??"; 1982??.

[Chiappa84] J. Noel Chiappa; "??"; ‘dev-cgw’ mailing list; 1984?.


[Corbato] Fernando Corbato and Jerome H. Saltzer; "Multics: The First Seven Years?"; ??


[Reed] David Reed; "??"; CSR Group Talk; 1979??

[SaltzerSR] Jerome H. Saltzer; "Source Routing?"; ??

