IEN 189

ISSUES IN INTERNETTING

PART 4: ROUTING

Eric C. Rosen

Bolt Beranek and Newman Inc.

June 1981

Bolt Beranek and Newman Inc.
Eric C. Rosen

ISSUES IN INTERNETTING

PART 4: ROUTING

4. Routing

This is the fourth in a series of papers that discuss the issues involved in designing an internet. Familiarity with the previous papers (IENs 184, 187, and 188) is presupposed.

The topic of the present paper is routing. We will discuss the issues involved in choosing a routing algorithm for the internet, and we will propose a particular algorithm. The algorithm we propose will be based on the routing algorithm currently operating in the ARPANET, called "SPF routing." This algorithm is described in [1] and [2], which interested readers will certainly want to look at. Although we will try to make this paper relatively self-contained, we will of course focus our discussion on those aspects of the algorithm which might have to be modified to work in the internet.

Any discussion of the proper routing algorithm to use in a particular Network Structure must begin with a consideration of just what characteristics we want the routing algorithm to have. That is, we must decide in advance just what we want the routing algorithm to do. Everyone will agree that the routing algorithm ought to be able to deliver data from an arbitrary source Switch to an arbitrary destination Switch, as long as there is a

physical path between them. Or at least, the routing algorithm should make the probability of being able to do this arbitrarily high. However, this is a very minimal criterion (as indicated by the fact that everyone would agree to it). There are many other requirements we must place on the routing algorithm if we intend to design a robust and high performance Network Structure. We will present some requirements and some possible routing algorithms which fulfill the requirements to a greater or lesser degree. We hope that by the end of this paper, we will have made a case that our proposed routing algorithm does a better job of meeting more of the desired requirements than does any other that we know of.

4.1 Flexibility and Topological Changes

One extremely important, though little noticed, feature that we should require of a routing algorithm is that it enable us to make arbitrary changes in the topology of the Network Structure, without the need to make manual changes in the internal tables of the Switches. This is a capability that has always existed in the ARPANET. IMPs can be added, removed, or moved around arbitrarily, and the routing algorithm automatically adapts to the new topology without any manual intervention. This seems simple enough, but it does place some significant constraints on the nature of the routing algorithm. For example, it immediately rules out fixed routing. By "fixed routing," we refer to any

scheme where a set of routes to each destination Switch is "compiled into" each Switch. In fixed routing schemes, there is generally a "primary route", to be used when the Network Structure is not suffering from any outages, and a set of alternate or secondary routes to be used if some component of the primary route should fail. We know of one network which does use this sort of fixed routing, and as a result, they are forced to adhere to a very strict rule which allows them to add or remove Switches only once every six months. Certainly, we would not want to build such a restriction into the internet.

Fixed routing also prohibits certain important day-to-day operational procedures that are often used in the ARPANET. For example, it is quite common, when an IMP is brought down for preventive maintenance, to "splice" that IMP out of the network by wiring together two of its modems. This causes two IMPs that ordinarily have a common neighbor to suddenly become direct neighbors of each other. (A similar function can also be performed by the telephone company, in case the power to the modems is shut off, or if the site cannot be reached.) This ability to preserve network bandwidth even when a site is down is quite important to robust network performance. Yet it is very difficult, if not impossible, to do this if the network has a fixed routing algorithm. It is not yet clear to what extent such day-to-day "firefighting" techniques will be applicable in the internet, but it certainly does not seem wise to design an

internet routing algorithm which would be too inflexible to permit the use of such techniques.

Another very useful capability which is difficult to combine with fixed routing is the ability to create arbitrarily configured test networks in the lab, and then to connect them to the real network. This is something that is done quite often in the ARPANET, usually for the purposes of testing out new software, and we will definitely need this capability in the internet in order to test out new gateway software (as well as to test out patches and bug fixes to the old).

It is also worth noting that implementing a scheme of fixed routing with a primary route and alternates to be used in case of outages is not nearly as trivial as it may seem. Remember that it is not enough for each individual Switch, when its Pathway to a particular neighbor fails, to pick an alternate neighbor as its next hop to some destination. Rather, any outage requires ALL the Switches to pick alternates in a COORDINATED MANNER, so that the routing produced by the use of the alternate paths is loop-free. This is quite a difficult problem, and if there are a large number of Switches and Pathways, any combination of which could fail, this means that a very large number of alternate paths must be maintained, requiring a consequently large amount of table space.

We will not be giving much serious consideration to the use of fixed routing in the internet. We mention it largely for the sake of completeness, and because there is a natural tendency, which we wish to oppose, to suppose that fixed routing must be simpler, cheaper and more reliable than dynamic routing. This tendency ignores the day-to-day operational problems involved in the use of fixed routing, as well as the difficult technical problems involved the the creation of fixed routing tables.

Preserving maximum flexibility to make topological changes requires the Switches to be able to determine, dynamically, just who their neighbors are. (Remember that two Switches of a Network Structure are neighbors if and only if they are connected by a Pathway, i.e., by a communications path containing no intermediate Switch of the same Network Structure.) In the ARPANET, each IMP is initialized to know how many modem interfaces it has, and does not determine that dynamically. However, initialization only tells the IMP how many interfaces it has; it does not tell the IMP who its neighbor is over each interface. The IMPs determine who their neighbors are dynamically, via the line up/down protocol, and a line between two IMPs cannot come up unless and until each of the IMPs knows the identity of the other.

The situation in the present Catenet gateways is quite different. Each gateway has a table of potential neighbors

"assembled in." When a gateway comes up, it attempts to communicate (via a special gateway neighbor protocol) with each of the gateways in its pre-assembled neighbor table. Two gateways are considered neighbors only if this communication is successful. Gateways will also consider themselves neighbors of other gateways that communicate with them according to the gateway neighbor protocol, even if the other gateway is not in the pre-assembled neighbor table. This means that two gateways, G1 and G2, cannot become neighbors unless either G1 is in G2's pre-assembled neighbor table, or G2 is in G1's pre-assembled neighbor table.

Of course, in a real operational environment, it is very important to ensure that site-dependent information is not assembled or compiled in. Rather, it must be separately loadable (over the network itself) by the Network Control Center, or whatever equivalent organization we create for operating the internet. In fact, site-dependent information ought to be preserved over reload of site-independent information, and vice versa. (This discipline is followed in the ARPANET.) Designing the gateways according to this discipline is a very non-trivial task, which must be planned for by the gateway designers at the earliest stage of gateway design. Otherwise, we will build for ourselves a very difficult set of unnecessary operational problems.

However, it is not a very good idea to have a fixed table of neighbors in each Switch, even if this table is separately loadable. This just does not give us the flexibility we desire for making arbitrary topological changes. If there has not yet been any difficulty with the Catenet's current scheme, that is probably because of the small number of gateways and component networks in the current internet environment. As the number of gateways increases, the need to have them dynamically determine who their neighbors are becomes increasingly more important.

However, having gateways discover (dynamically) who their neighbors are is a more difficult problem than having IMPs discover who their neighbors are. The interfaces on the IMPs function as point-to-point lines, so there can be at most one other IMP on the other end of a line, and any data sent out that line can be expected to reach just that IMP. Therefore it is not very hard for an IMP to discover which IMP is at the other end. An IMP simply sends its identity (a unique number which it reads from its hardware configuration cards) down the line in a message, and if the line is operational, the message must reach the IMP on the other end. For two gateways connected by a packet-switching network, the problem is more complicated, because, unlike telephone circuits, a packet-switching network is not a point-to-point line with a relatively transparent interface. In order for one gateway to identify itself to another, it must be able to address the other, using the Access

Protocol of the packet-switching network which serves as the Pathway between them. This seems to mean that for a gateway to be able to send its identity to a neighbor, it must already know the neighbor's name. This seems like Catch-22 -- there is no way to determine dynamically who your neighbor is, unless you can address him, but there is no way to address him unless you already know who he is.

This problem can be made more tractable through the cooperation of the packet-switching networks underlying the Pathways which connect the gateways. A packet-switching network could recognize that certain of its own components (which might be either Switches or Hosts within its own Network Structure) are also Switches within a Network Structure which is one level higher in a hierarchy. For example, in the ARPANET, there might be some special protocol (call it the "gateway discovery protocol"), carried out on the host-IMP level, by which certain hosts identify themselves as internet gateways. Whenever a gateway connected to a particular IMP comes up or goes down, this information could be broadcast to all other IMPs. Whenever a gateway comes up, the IMP it is connected to could tell it which of the other hosts are internet gateways. In this way, the IMPs could keep the gateways informed as to which other gateways are up or down at any particular time. This sort of scheme eliminates the need for the gateways to know in advance who their neighbors might be, and moves the responsibility for keeping

track of the gateways and their up/down status to the packet-switching network itself, which is better equipped to carry out this responsibility.

Such a scheme would not be very difficult, in principle, to build into the ARPANET. Information about gateways could be subsumed into the routing information. That is, an IMP connected to a gateway could represent the gateway as a stub node, and report on it as such in its ordinary routing updates. (Of course, this is only feasible if the number of gateways is relatively small when compared to the number of IMPs. Otherwise the additional overhead this would add to the ARPANET's internal routing algorithm would make the scheme infeasible. However, it does seem likely that the number of gateways on the ARPANET will always be much smaller than the number of IMPs.) This scheme would automatically cause the information about the gateways to be broadcast to all IMPs as part of the routing updates. (See section 4.5 for a description of the routing update procedure.) Each IMP which is connected directly to a gateway could forward information about other gateways to its own gateway as the information is received. The most difficult problem might be to get enough "security" in the gateway-to-IMP protocol so that only real gateways could declare themselves to be gateways. (Some of the issues involved in preventing a host from "fooling" the network into thinking it is a different host than it really is are discussed in IEN 183. See the discussion of LAD messages.

However, that note does not consider the real issue of security that arises here.)

This scheme for having gateways dynamically discover their neighbors through the cooperation of the networks underlying the internal Pathways of the internet is an important step towards the solution of the "flying gateway" problem. The flying gateway problem is the following. Suppose that N is a packet-switching network which is one of the component networks of the internet. Now suppose that due to some sort of emergency or natural disaster, N becomes partitioned into two "pieces", call them N1 and N2, and that this partition is expected to last for a significant amount of time. If H1 is a Host in N1, and H2 is a Host in N2, then H1 and H2 will no longer be able to communicate through the network N. (Of course, H1 and H2 might still be able to communicate though the internet, if there is an internet gateway on N1 and an internet gateway on N2, and a route between these two gateways other than the "direct" route via N. In fact, the addressing scheme proposed in IEN 188 will automatically cause traffic from H1 to H2 to be delivered over this alternate route, AS LONG AS H1 SUBMITS THIS TRAFFIC TO ONE OF THE INTERNET GATEWAYS CONNECTED TO N1, RATHER THAN TRYING TO SEND IT DIRECTLY TO H2 OVER THE NETWORK N.) However, in some cases, there may be no such alternate route, or else its characteristics might be unsatisfactory. In addition, it must be remembered that the partition of network N might actually result in the partition of

the internet itself, so that some pairs of Hosts which ordinarily communicate over the internet can no longer reach each other. In such cases, it might be desirable, at the level of the internet, to treat N1 and N2 as separate component networks, and to place an internet gateway between them so that internet traffic can flow from N1 to N2. One possible scenario is for this new gateway to be an airborne packet radio, hence the name "flying gateway."

If a flying gateway can be connected to both N1 and N2, and if the network N has a gateway discovery protocol of the sort we have been advocating, then the flying gateway need merely come up on N1 and N2, declaring itself to be an internet gateway. The gateway discovery protocol run in the network pieces N1 and N2 will cause the other internet gateways in N1 and N2 to become aware that they have a new neighbor, the flying gateway. Once the gateways in N1 and N2 become aware of their new neighbor, it automatically begins to participate in the routing algorithm (see section 4.5 for details of the routing updating algorithm that brings this about), and routing automatically begins to use the flying gateway for store-and-forwarding internet traffic. Thus any partition of the internet is automatically brought to an end.

In addition to using the flying gateway as a transit or intermediate gateway for internet traffic, it may also be desirable to use it as a destination Switch in the internet.

That is, it may be desirable to allow the other internet Switches (gateways) to use the flying gateway as the address to which they route traffic for Hosts in N1 or N2. This is slightly more complicated than simply using the flying gateway as an intermediate Switch. The logical-to-physical address translation tables in the gateways (we are assuming the addressing scheme proposed in IEN 188) will not, in general, map any Host logical addresses into the address of the flying gateway, which after all is not ordinarily on the internet. However, as long as the flying gateway indicates that it is a special, flying, gateway, and as long as this information is made known to all the other gateways, this problem is simple enough to solve. If F is a flying gateway, and G is an ordinary gateway, and F and G are neighbors, then any logical address which maps to G but cannot currently be reached through any ordinary gateway should be mapped to F. (As we shall see, the routing algorithm we propose makes available to each Switch all information about which pairs of Switches are neighbors.) Attempting to reach the destination Host via the flying gateway F will either be successful, or else should result in the return of a DNA message, which would indicate that the Host cannot be reached from the flying gateway either. The only remaining problem is for the flying gateway itself to determine which of the two pieces of the partitioned network contain some particular Host for which it is the destination Switch. Any data for destination Host H which

arrives at Switch F can potentially be sent to either piece of the partitioned network. The situation is no different than the problem of how an ordinary gateway, which has two Pathways to a particular Host, one of which is non-operational, decides which one to use. Note that the individual Hosts do not need to be aware at all of the existence of the flying gateway, since the logical addressing scheme automatically finds the right physical address. Of course, for this mechanism to be at all effective, there must be a robust and efficient Host-Switch up/down protocol, which works through the cooperation of the network underlying the Pathway between Host and Switch.

Unfortunately, not every component network of an internet can be expected to cooperate this way in a "gateway discovery protocol." In fact, if two Switches of the internet Network Structure are connected by a Pathway which is itself an internet, rather than a single packet-switching network, then this sort of cooperation in the "gateway discovery protocol" might be extremely difficult if not impossible. It seems though to be quite important to get the communications media which underlie the Pathways to participate in such a protocol, for that significantly increases both the reliability and the flexibility of the internetting scheme. It does not seem possible for Switches which are connected by uncooperative Pathways to determine dynamically who their neighbors are. In such cases, we may then have to live with hand-built neighbor tables (as in the

present Catenet), and a protocol which the Switches attempt to carry out with their neighbors to see which potential neighbors are really reachable. Networks which do not provide a gateway discovery protocol, however, cannot be patched together with a flying gateway if they should partition.

Even for Switches which are connected by cooperative Pathways, it is desirable to have a protocol which the Switches attempt to run with each one of their neighbors, to see whether they really can send and receive data to or from each neighbor. Suppose, for example, that two Switches are connected by a Pathway which is a very congested network. In such a network, the messages which are used to tell the internet Switches who their "neighbors" are might well be flowing, even though the congestion prevents ordinary (user) data from flowing. This is not at all unlikely, if the gateway discovery protocol makes use of the network's routing updates, which would probably be of much higher priority than ordinary data packets. Since we don't want to use this Pathway for internet traffic unless it can carry data, some independent means of determining this may be needed. The situation is somewhat more complicated if the Pathway is a packet-switching network with different "acceptance classes", so that only certain classes of traffic are accepted at any given time, depending perhaps on the internal loading conditions of the network. If a Pathway is only accepting a certain sub-class of data traffic, any internet Switches which are connected to that

Pathway must be able to determine which classes are being accepted (presumably the network underlying the Pathway will inform the Switches as to any access restrictions), and this information will have to be fed back into the internet routing algorithm, so that traffic which cannot be placed on a certain Pathway is not routed there nonetheless.

The reader will doubtless have noticed that these considerations, of determining who one's neighbors are, and of determining whether the Pathway to each neighbor is operational, are quite similar to the considerations adduced in IEN 187 in the discussion of Pathway up/down protocols to be run between a Host and a Switch. What we have been discussing is really an inter-Switch Pathway up/down protocol. The gateway discovery protocol corresponds to what we called a "low-level up/down protocol", and the type of protocol discussed in the previous paragraph corresponds to what we called the "higher-level up/down protocol."

4.2 Why We Cannot Require Optimality

What else would we like the routing algorithm to do, besides giving us the maximum flexibility to make topological changes? Generally, we tend to feel that a really good routing algorithm should optimize something, delay or throughput, for example. However, true optimality is really not possible. If we are given a complete description of a network, including its topological

structure, and the capacities and speeds of all its lines and Switches, and if we are also given the traffic requirement (as a Switch-Switch traffic matrix which tells us how much traffic each Switch will originate which is destined for each other Switch), and if the packet inter-arrival rates and sizes vary according to certain specific probabilistic distributions, and if the traffic is in a steady-state condition, it is just a mathematical problem to devise a set of routing tables for the Switches which will minimize the network average delay. Applied mathematicians have devoted a great deal of effort to devising algorithms to produce this optimal solution. There are a large number of problems with attempting to use this sort of "optimal routing algorithm" as the operational routing algorithm of a network:

1) Packet arrival rates and sizes do not necessarily vary according to the probabilistic distributions which are assumed by optimal routing algorithms.

2) Optimal routing algorithms are ALWAYS based on mathematical models of the relationship between delay and throughput which are not supported by empirical data.

3) Actual traffic requirements are quite variable, and may not really approach a steady-state for a long enough period of time to enable true optimization. Traffic requirements are also generally unknown, and difficult to predict or measure.

4) Most algorithms to compute the optimal routes are real number crunchers, and require large floating point computers. These algorithms would have to be run in a central location, producing routing tables for all Switches, and then distributing them somehow (centralized routing), with consequent problems of robustness and overhead.

5) There are distributed optimizing algorithms (e.g., Gallager's algorithm), but they are not implementable. That is, the proofs of these algorithms make assumptions which could not be made to hold in the real software and real hardware of a real network. Hence the algorithms would not be expected to give optimal results (or even anything close to optimal) in real networks. Furthermore, such algorithms seem to rely on updating protocols which are insufficiently robust in the operational environment. These algorithms also seem to contain parameters whose precise settings are quite important to proper performance, but whose most appropriate values are unknown and quite difficult to determine.

We realize that these rather brusque comments may make it seem like we are giving short shrift to the consideration of optimizing algorithms. We have made these comments simply in

order to state our reasons for not giving further consideration to such algorithms. Arguing in support of these reasons, however, would require another paper.

Another problem with optimal routing algorithms which is more specific to the internet environment has to do with the requirement that the capacities of the network components be known. With telephone circuits as the "links", it is possible to assign a fixed capacity and fixed propagation and transmission delays to each link. With packet-switching networks as the "links", it is doubtful that this even makes sense. If two gateways are connected by the ARPANET, there is no number we can assign as the capacity of the "link" connecting the gateways! The amount of throughput that can be sent between two gateways via the ARPANET is a highly variable quantity, with dependencies on hundreds of other things going on within the ARPANET. It is hard enough to get a handle on just what other things the throughput of a given connection depends on; we certainly can't express this dependency as a function, or assign numerical values to the "capacity." This seems to mean that currently known optimal routing algorithms are really quite useless within the context of the internet. Of course, they are not too useful even in individual networks, when considered as the operational routing algorithm of the network. They are, however, sometimes useful as a benchmark to which the operational routing algorithms can be compared. That is, it is a meaningful question to ask,

"how close does SPF routing in the ARPANET come to optimal?", where "optimal" is defined as the result produced by some optimal algorithm, run off-line. Within the context of the internet, it is difficult even to give meaning to this question. There is no mathematical model of the internet to which we can appeal.

This also raises an interesting question about the design of the internet topology, i.e., where to place the gateways and how best to interconnect them. The usual mathematical techniques for trying to optimize network topological design also assume some fixed assignment of capacity to the links; it's not obvious how such techniques can be extended to the internet.

4.3 Some Issues in SPF Routing

Even if we give up the quest for optimal routing, there are still a number of substantive things we can require of a routing algorithm. For example, we would like to have some form of distributed routing, rather than centralized routing, simply for reasons of robustness. ("Distributed routing" refers to any routing scheme in which each Switch computes its own routing table.) What this means basically is an algorithm based more or less on the routing algorithm of the ARPANET, i.e., an algorithm which runs in each Switch and computes the shortest path to each other Switch, based upon (dynamically determined) knowledge of the connectivity of the internet Network Structure, and an assignment of "length" to each Pathway that connects two

Switches. Routing algorithms of this sort can be characterized by three separable components: (a) the algorithm used to compute the shortest path, given the assignment of lengths to Pathways, (b) the algorithm used to assign a length to a given Pathway, and (c) the protocol used by the Switches for sharing routing information.

The most efficient shortest path algorithm that we know of is the SPF algorithm of the ARPANET [1,3] (which is basically just a modification of Dijkstra's shortest path algorithm), and we propose to base an internet routing algorithm on this. There are other algorithms for performing a shortest path computation, but the SPF algorithm seems to dominate them. One possible alternative to SPF would be something based on the distributed computation of the original ARPANET routing algorithm (which is the basis for the current Catenet routing), but we have studied that algorithm at great length and in great detail and it is inferior to SPF in a large variety of ways [3]. There are many other shortest path algorithms (such as Floyd's algorithm, or the algorithm advocated by Perlman in IEN 120), but the efficiency of these algorithms does not compare with that of SPF. We will not consider the issue of choosing a shortest path algorithm any further.

In the ARPANET, the "length" assigned to a line is just the average per-packet delay over that line during a preceding period

of ten seconds. The current Catenet routing algorithm assigns a length of 1 to each Pathway, irrespective of the delay. Other possible assignments of lengths to Pathways are also possible. We will recommend the use of measured delay as the best metric for the internet routing algorithm to use, and we argue for this proposal in sections 4.3.1 and 4.3.3. Section 4.3.2 covers the related topic of "load splitting." (One purpose of that section is to show that the two topics are indeed related, and in ways more subtle than generally realized.) In section 4.4, we discuss some of the issues in the design of an algorithm to measure the delays.

In the ARPANET, a routing update generated by an IMP A specifies the average per-packet delay on each of A's outgoing lines. Every update generated by an IMP is sent to every other IMP in the network, not just to the neighboring IMPs, as in the Catenet routing algorithm. This updating protocol, and its applicability to the internet, are discussed in section 4.5.

Although a routing scheme can be divided into a number of separable components, it is important to keep in mind that the ultimate characteristics of the routing scheme will result from the combination of the components. A routing scheme must be judged as a whole. The reader should try to focus throughout on how the components work together, and resist the temptation to judge each component separately.

4.3.1 Min-Hop Routing (Why Not to Use it)

The simplest routing scheme which is based on having each Switch compute its shortest path to each other Switch is "min-hop" routing. In min-hop routing, all Pathways are assigned unit length, so that the shortest path between two Switches is just that path which has fewer Pathways than any other. (Generally, ties are broken arbitrarily.) This sort of routing is used in the current Catenet, where traffic is routed through the fewest possible number of intermediate networks (or equivalently, through the fewest number of intermediate gateways.) This form of routing is quite simple, and does not require us to worry about anything as complicated as detecting changes in load or delay in remote components of the Network Structure. Such changing conditions within the Network Structure have no effect at all on the routing. This form of routing can be done with the minimal amount of overhead (in terms of the need to send routing updates from Switch to Switch). Updates need to be sent only when the Pathways go down or come up. Any algorithm which attempts to be more responsive to changing conditions in the Network Structure than min-hop routing still needs these up/down updates, plus more besides. Min-hop routing is definitely what one would use if one wanted to put in a "quick and dirty" routing algorithm, and put off worrying about complexities until some unspecified later time. It is also possible to argue for min-hop routing in the internet on more principled grounds, as follows:

> "In general, it is not unreasonable to expect that the more component networks an internet packet goes through, the less likely it is to get to its destination, and the longer its delay is likely to be, if it does reach its destination. We might expect that the number of component networks a message goes through would generally correlate fairly high with the delay of the message, and would generally correlate fairly low with the obtainable throughput of a host-host transfer."

Unfortunately, this sort of reasoning is only valid when applied to a Network Structure consisting of homogeneous Pathways, which have similar characteristics with respect to delay, throughput, and reliability. This is rather unlikely to be the case in the internet, whose distinguishing characteristic is the heterogeneity of its Pathways. Where the Pathways of a Network Structure have widely varying characteristics, delay and throughput are not very likely to correlate well simply with the number of hops.

It is true that the delay-oriented routing of the ARPANET generally gives the min-hop paths. (Remember, though, that the ARPANET, unlike the internet, has generally homogeneous Pathways.) Min-hop routing is all right for the "normal" case, where there are no areas of congestion in the network or internet, no areas where the delay is unusually high compared to other areas. Routing, however, is no different from other computer system applications, in that a scheme that works well only in the normal case just is not robust enough to be satisfactory. (Think of a magnetic tape driver which works in the normal case, where no tape errors are encountered, but which

crashes the system in the presence of "unusual" events, like errors on the tape. Such a driver may be acceptable if one accesses one tape a month, but not if one needs to read or write ten tapes a day. The analogy is that min-hop routing may perform acceptably in an experimental network with little traffic, but is much less likely to be acceptable in a heavily loaded operational network.) It is extremely common for some area of the network to be much more congested than another, so that traffic flows which traverse a particular area experience a very much longer delay (and lower throughput) than traffic flows which avoid that area. Significant imbalances in load cause significant reductions in the correlation between hop-count and performance. Such imbalances may not be present in a network initially, but if the ARPANET experience is any indication, imbalances start to occur with increasing frequency as network utilization grows. If the routing algorithm cannot account for such imbalances, network performance problems will start to occur with ever-increasing frequency as the network gains more users. This was our experience with the original ARPANET routing algorithm. For all its widely publicized faults, it provided generally acceptable performance as long as the network was very lightly utilized, but its failures became more and more evident as the ARPANET shifted from a research prototype to a communications utility. If we expect our network or internet to be heavily used by real users who are sending real data that they really need for their

applications, OUR ROUTING ALGORITHM WILL HAVE TO BE ROBUST ENOUGH TO DETECT EXCEPTIONAL CONDITIONS AND TO ROUTE THE TRAFFIC IN SUCH A WAY AS TO MINIMIZE THE EFFECT OF THE EXCEPTIONAL CONDITIONS. IF AREAS OF THE NETWORK BECOME CONGESTED OR EXPERIENCE UNUSUALLY LONG DELAYS, THEN WE HAVE TO BE ABLE TO ROUTE THE TRAFFIC AROUND THESE AREAS, instead of blindly sending traffic into congested areas. At a certain level of congestion, sending traffic into a congested area is like sending it into a black hole: the traffic will never leave the area to progress to its destination. Sending traffic into a congested area also induces a feedback effect, causing the congestion to spread farther than it otherwise would, and making it that much less likely that the congestion will dissipate. Any routing algorithm which cannot take this into account will not be robust enough to survive in a real operational environment.

Min-hop routing also has another disadvantage which is more specific to the internet environment. Let N1, N2, and N3 be three networks, and suppose we have to get some traffic from N1 to N3 by using N2 as a transit network. Let G12 be a gateway connecting N1 and N2, let G23 be a gateway connecting N2 and N3, and let G2X be a gateway which connects N2 to some other unspecified network. If we use min-hop routing, then any traffic which must go from G12 to G23 must go "directly", through network N2, without stopping at G2X, because the path G12-G2X-G23 has one more hop than the path G12-G23. Perhaps this doesn't seem like

much of a restriction; why would one want to have traffic stop at the intermediate gateway G2X when it could go directly from G12 to G23? Actually, two possible reasons come to mind immediately. The first reason has to do with the possible effects of the network's end-end protocol. In the ARPANET, for example, a source host is allowed to send only 8 messages to a given destination host before receiving the RFNM for the first of the 8 messages. Hence the throughput obtainable on a host-host connection is inversely related to the amount of time it takes to get a RFNM from the destination host to the source host. It follows that higher throughputs are obtainable between hosts that are "near" each other than between hosts that are "far" from each other. It is also possible that G12 and G2X will be near to each other, and that G2X and G23 will be near to each other, but that G12 and G23 will be far from each other. So the throughput obtainable in a transfer between G12 and G23 may be less than that obtainable in a transfer between G12 and G2X, and less than that obtainable in a transfer between G2X and G23. It follows that the throughput obtainable between G12 and G23 via G2X may be higher than the throughput obtainable between G12 and G23 directly. Basically, by using an additional gateway hop, the ninth message from G12 can be put into the network while the first message is still in transit from G2X to G23, while without the intermediate hop, this is not possible. Of course, the best solution to this sort of problem would be to fix the end-end

protocol so that it does not impose this sort of restriction. Our present point, however, is that our routing algorithm should not rule out the possibility of this sort of strategy. Note that by using an intermediate gateway hop, we might not only increase throughput, but also decrease the delay (since a ninth message would not be blocked as long.)

(It is interesting to think about whether this sort of strategy might not be useful entirely within the ARPANET.)

Another possible scenario in which an intermediate gateway hop might be useful occurs if the intermediate gateway is multi-homed. It is possible that an intermediate gateway will be homed to two IMPs which are distant from each other within the network. If so, the intermediate gateway may be used as an "expressway" around a congested area of the network.

If we replace the intermediate gateway G2X with two gateways G24 and G42, we also have the possibility of sending traffic from N1 through G12 into N2 to G24 through N4 to G42 into N2 to G23 and thence into the destination network N3. This is akin to the oft-discussed expressway problem, but cannot be handled within the framework of min-hop routing. Of course, it might be very difficult to take account of such factors, but one would not want to have a routing scheme which makes it absolutely impossible.

Still another disadvantage of min-hop routing in the Catenet is the following. The current Catenet routing algorithm, when faced with three gateways on the same network, considers the three to be equidistant. However, the delay and throughput obtainable from gateway A to gateway B may be very much different than the throughput obtainable from gateway A to gateway C. In a large distributed network like the ARPANET, some pairs of hosts are connected by high-performance paths, and some by low-performance paths (either because they are separated by many hops, or because the path between them is under-trunked, etc.) Allowing the routing algorithm to be sensitive to this could potentially have a large impact on the internet performance.

There may not be any network that actually uses min-hop routing, except for the Catenet. There are, however, networks that use a variant of it, which we might call "fixed cost" routing. In fixed cost routing, each Pathway is still assigned a constant length, but not all Pathways are assigned the same length, and some Pathways have a length which is not equal to 1. In a scheme like this, one attempts to assign values of length so that slow-speed lines appear longer than high-speed lines, reliable lines appear shorter than unreliable ones, and lines with high propagation delays appear longer than lines with low propagation delays. This sort of routing is used in DATAPAC and in DECNET. Both those network architectures have routing algorithms based on the original ARPANET routing algorithm. The

designers of those architectures apparently realized that min-hop routing is not very satisfactory if the links are not of relatively homogeneous quality, but were probably wary of the problems that the ARPANET's original algorithm had in adapting to changing traffic conditions. They avoided these problems by not adapting at all to changing traffic conditions. Of course, this is the weakness in fixed cost routing. It may be better than min-hop routing in a lightly loaded Network Structure with heterogeneous Pathways, but in a heavily loaded Network Structure with unbalanced load it really is no better than min-hop routing, and will still send traffic right into congested areas.

We have been emphasizing the claim that routing ought to be able to detect congestion and route traffic around it. Some may wonder whether we are confusing the proper functions of routing with the proper functions of congestion control. That is not the case. Congestion control schemes generally try to limit the amount of traffic entering a network so as to prevent or to reduce the overloading of some resource or of the whole network. When congestion actually exists in the network, however, it is the job of routing to try to send traffic around the congested areas; otherwise the routing actually causes the congestion to increase. Of course, one might attempt to design the routing algorithm under the assumption that there will be a congestion control scheme that will make congestion impossible. However, such a design could not be very robust. If we want to build a

robust Network Structure which will continue to operate under a variety of unforeseen conditions. then we want each software module or protocol to be designed with the assumption that the other modules or protocols will be less than optimal. The resulting system will be much less prone to system-wide failure than one which is designed so that no part of it will work at all unless every part of it works perfectly. Although we will not be discussing explicitly, in this paper, any schemes for controlling the amount of traffic which is input to the internet, that doesn't mean that we can ignore the way in which the routing algorithm affects and is affected by the existence of congestion. Particular problems related to overload of network resources should be discussed in whatever context they arise in, without worrying about whether the problem is properly called "congestion control" or "routing." There is in general no way of telling in advance whether the best solution to a particular problem is a routing solution or a congestion control solution, and putting labels on the problems just restricts our thinking.

4.3.2 Load Splitting

Routing in the ARPANET has always been "single-path routing." We mean by this that at any given moment, the ARPANET's routing algorithm provides only a single path between each pair of IMPs. All traffic which enters the network at some particular time, originating at IMP A and destined for IMP B,

will travel over the same path. Actually, this statement is somewhat oversimplified, since there might be a change of routes while some traffic is already in transit. The point, however, is that at any given time, each source or intermediate IMP will send all traffic for a particular destination IMP to a unique neighbor; it cannot split the traffic among several neighbors.

Routing in the Catenet is currently somewhat different. Suppose gateway A has two neighbors, B and C, and has some traffic to send to gateway E. The routing algorithm run in A assigns a distance value to the path to gateway E via neighbor B and a distance value to the path to E via neighbor C. If the distance from A to E via B is the same as the distance from A to E via C, then gateway A will alternate between use of B and C when sending traffic to E. That is, A makes simultaneous use of two distinct paths to E. Such a scheme would be somewhat more difficult to put into SPF routing, because in SPF routing, no assignment of distance values from A to E via each of the two neighbors is generated. Rather, only one path is computed, via one of the neighbors, and only the distance on that one path is known. Distance on other paths is not computed by the SPF algorithm. (On the other hand, the SPF algorithm generates the entire path, so that each Switch knows which other Switches its traffic will be routed through on the way to the destination. The original ARPANET algorithm does not do this, but only tells each Switch which of its neighbors to use when sending traffic to the destination.)

What is the significance of this? It seems to be commonly regarded as obvious that multi-path routing, or load splitting, is an important advantage, so that routing algorithms that permit it are better than routing algorithms that do not. However, when one asks advocates of multi-path routing why it is better than single-path routing, a very common answer seems to be, "Multi-path routing is better because it provides multiple paths." This sort of answer is rather superficial. Multiple-path routing is NOT a goal in and of itself; IT IS IMPORTANT ONLY INSOFAR AS IT SERVES SOME MORE FUNDAMENTAL GOAL. If a multi-path routing algorithm results in smaller delays or larger throughput than some other algorithm, then that is a good reason for favoring it over the other algorithm. Now, it is certainly true that any routing algorithm which OPTIMIZES network delay or throughput will be a multiple-path algorithm. THE CONVERSE, HOWEVER, IS NOT TRUE. A routing algorithm which provides multiple paths does not necessarily optimize delay or throughput. In fact, merely because a routing algorithm provides multiple paths, it does not follow that it provides better performance in any respect than some other routing algorithm which provides only a single path between a pair of Switches. An algorithm which provides a single good path may be far superior to an algorithm which provides several poor ones.

To see this, let's look at some possible effects of the load splitting in the Catenet routing algorithm. Let A, B, C, D, and

E be five gateways, and suppose that there are two possible paths from A to E, namely ABDE and ACDE. The Catenet routing algorithm would regard these two paths as equidistant, since that algorithm regards two paths as equidistant if they contain the same number of intermediate gateways. Therefore gateway A would perform load splitting on its traffic to E, sending half of the traffic to neighbor B and half to neighbor C. Does this provide more throughput than the use of a single one of these paths? Not necessarily. If the bottleneck on the paths from A to E is the Pathway DE, then the use of these two paths provides no more throughput than the use of either one alone. In fact, if DE is the bottleneck, the use of the two paths will probably result in lower throughput than the use of a single path. The use of several paths increases the likelihood of the packets from A to E arriving out of order at the destination host. Yet as more packets arrive out of order, more TCP resources are needed to handle them, and the TCP just has that much more work to do. TCP buffers that are occupied by out-of-order packets cannot be "allocated" for receiving more packets, so acknowledgments must be delayed, and windows must be kept smaller. The result of all this will be higher delays and lower throughputs. This was probably not the intention of load splitting, but is a likely consequence of it.

Suppose there really are two independent paths from A to E which are "equidistant", say ABDE and ACFE. Even here, sending

half the packets on each path may only degrade performance. To see this, suppose each of the Pathways AB, BD, DE, AC, and CF has a capacity of 50 kbps, but that link FE has a capacity of 10 kbps. Suppose also that we want to send 50 kbps of traffic from A to E. If we alternate packets between these two paths, by trying to send 25 kbps of traffic each way, we will be able to get at most 35 kbps of traffic through to the destination, and we will cause severe congestion on link FE (which will probably result in its being able to carry even less than the rated 10 kbps, further lowering the network throughput.) Had we used only the single path ABCD, we would have been able to pass more traffic. Again, we see a situation where the use of load splitting can reduce throughput and increase delay.

This sort of problem might at first appear to be too unlikely to be worth worrying about. However, it has already occurred in the Catenet, and has caused a significant problem. In fact, in the Catenet's actual problem, half of the traffic was sent on a path whose capacity was sufficient to handle all the traffic, and the other half of the traffic was sent on a path whose capacity was essentially zero (because a network partition made the destination host unreachable on that path). In this case, load splitting resulted in the throughput being cut in half, as half the traffic was routed down a black hole! The problem was "solved" by eliminating one of the two possible paths, thereby eliminating the possibility of load splitting.

However, this does not seem like a proper way to deal with this problem in the general case.

The Catenet's load splitting has been defended from this latter objection as follows: "If there were no load splitting, maybe all the traffic would have been sent into the black hole, not just half." This is less a defense than a sad commentary on the state of the Catenet routing; to accept this sort of defense is just to give up entirely on the problem of internet routing.

Someone may reply to our first criticism of load splitting by saying "maybe the bottlenecks will be Pathways AB and AC, rather than DE, in which case the use of two paths does increase the throughput." This reply is correct, but not very important. The sort of load splitting done in the Catenet might, by pure chance, increase throughput in some particular case. The point though is that it is no more likely to increase throughput than to decrease it. Certainly there is no reason to suppose that the cases in which it might help are any more likely to occur than the cases in which it hurts. In our experience with the ARPANET, schemes that seem a priori as likely to hurt as to help always end up hurting more than helping. (In networking, Murphy's law is more than just a joke.) Choosing equidistant paths for load splitting will generally result in paths which are only small variants of each other (if it results in any paths at all, since there are not necessarily several equidistant paths between a

pair of Switches), and there is no reason to suppose that the bottleneck will not be common to each path. Even if we do get two paths which do not share a bottleneck, unless we try explicitly to apportion the flows to the relative capacities of the two paths (rather than just dividing the traffic 50-50), we will not, in general, gain any increase in throughput.

In chapter 4 of [6], we actually devised a multiple-path routing scheme, based on SPF, whose purpose was to maximize throughput. In this scheme, we make sure that any set of simultaneously used paths between two Switches are "bottleneck-disjoint", (i.e., they don't share a bottleneck), so that we know that we can get more throughput by use of several paths. We also devised a flow apportionment scheme which attempts to match flows (or parts of flows) to the available capacity of each path. Anyone interested in seeing what it really takes to do multi-path routing should look at that chapter. The scheme proposed there is quite complex, however, and it is not obvious that it will work. Some simulation work will eventually be done on it. Until that sort of algorithm is much better understood, it would not be very wise to use the internet to experiment with it. It will be difficult enough to adapt a well-understood and much-used routing algorithm (like that currently in the ARPANET) to the internet environment. The internet is certainly not a place for experimenting with new and untried routing algorithms.

Although it is quite difficult to design a multi-path routing procedure that results in significant improvements of delay or throughput over single-path routing, there are other reasons for requiring multiple paths between a pair of Switches that are more easily dealt with. For example, we may be required to have different paths between a pair of internet gateways because of ACCESS CONTROL RESTRICTIONS. That is, certain classes of packets may not be allowed to traverse certain classes of networks, so that different routes would be required for the different classes of traffic. We may also decide that different types of service that may be requested by the user should travel over different paths, even if the source and destination gateways are the same for the different traffic classes (e.g., maybe we don't want to use multi-hop satellite networks for interactive traffic.) This is easily handled within the framework of SPF routing. Remember that the SPF algorithm produces the shortest path to a destination, based on an assignment of lengths to the Pathways. Rather than simply assigning a unique length to each Pathway, we can assign a set of lengths, indexed by traffic classes. We can then produce a set of routing tables, indexed by traffic type, such that the routing table for a given traffic type contains the "shortest" path, based on the length assignments for that traffic type. For example, if traffic class C is not permitted to traverse Pathway P, the length of P, indexed by C, can be set to infinity. This ensures that that

Pathway will not be part of any path found in the routing tables indexed by C. We even have the flexibility to assign to P a length which, while not infinite, is much larger than the length of any other Pathway. In this case, that Pathway will be used for traffic of class C only if EVERY path to the destination includes it (i.e., only if it can't be avoided). This sort of load splitting might be quite important in the internet, and is also quite simple to handle.

4.3.3 Delay vs. Throughput

In the ARPANET, each IMP measures the average delay per packet on each of its outgoing lines. This average delay is assigned as the "length" of the line, and shortest paths are computed on that basis. We have studied the performance of this algorithm a great deal [5]. It tends to use min-hop routes under conditions of light or of uniform load. However, it does seem to take account quite well of the varying delays that are produced by lines of different transmission or propagation delay characteristics. Since congestion causes large increases in the delays, congestion is generally detected by the routing algorithm, and traffic really is routed around congested areas when that is possible. While we cannot claim that our routing algorithm gives the optimal delay, the characteristics that it does have seem to be the characteristics that we would really like to see in any robust, operational network, and particularly

in the internet. The routing tends to be stable on what are intuitively the best paths, except when exceptional conditions arise which make it clear that some other path is likely to provide better performance. It is this sort of routing which we propose for the internet.

Before discussing further the use of delay-oriented routing in the internet, we would like to briefly consider the issue of throughput-oriented routing. In the previous section, we argued against the use of multi-path routing as a means of optimizing throughput, largely on the grounds that doing it right is extremely difficult (much more so than one might at first think), that the ways of doing it right are quite poorly understood, and that the internet is not a good testing ground for new and untried algorithms. However, one often hears that there are high throughput applications (bulk traffic) for which delay doesn't matter, and one may wonder whether there is not some kind of single-path routing which is more appropriate for such applications than is delay-oriented routing. One scheme that is very commonly suggested is that of routing traffic on the path of maximum excess capacity, instead of on the path of least delay.

Given an algorithm for determining the amount of excess capacity on each Pathway (which could be quite difficult to design for the internet environment -- how do we know what the excess capacity of a packet-switching network is?), it is no

difficult matter to modify the SPF algorithm to produce the paths of maximum excess capacity. However, it would not be a good idea to use the resultant routes for bulk traffic. For one thing, we must understand that such a routing algorithm would not maximize total network throughput. (By "maximizing total network throughput", we mean maximizing the amount of traffic that the network can handle.) Suppose, for example, we wanted to send 40 kbps of traffic, and had the choice of using a one-hop path with excess capacity of 50 kbps, or a 10-hop path, each of whose links had an excess capacity of 100 kbps (so that the total composite path has an excess capacity of 100 kbps). By using the shorter path, we use up a total of 40 kbps of network capacity, capacity which is now unavailable for other traffic. By using the longer path (which is the path of maximum excess capacity), we use up a total of 10x40 kbps (40 kbps per hop), thereby using up a total of 400 kbps which is no longer available for other traffic. In terms of maximizing the total network throughput, we do better by using the one-hop path, rather than the path of maximum excess capacity.

Maybe we are less interested in maximizing total network throughput than in finding a path for some particular traffic flow which has enough capacity to handle the required throughput of that flow. We still would not want to use the path of maximum excess capacity, for that path might have a delay which is much too long. Although we often hear that certain classes of traffic

(e.g., file transfer) care only about throughput, not delay, this is really a gross oversimplification. In file transfer, we don't care how long it takes for the first packet to reach its destination, AS LONG AS ALL THE FOLLOWING PACKETS FOLLOW IMMEDIATELY, WITH NO DELAYS BETWEEN THE ARRIVALS OF SUCCESSIVE PACKETS. Of course, if there are long delays between the packets of a file transfer, the throughput will be very low. Hence it is not quite true to say that file transfers and the like are unconcerned with delay. If higher level protocols like TCP are being used, then routing over a path of long delay will certainly result in lower throughput. The reason is as follows. A TCP sender will only send a certain amount of data, until he fills the window specified by the TCP receiver. The size of the window is very likely to depend on such network-independent things as the amount of resources (e.g., buffers) in the destination host. If the path between source and destination host is very long, then the sending TCP will fill the window, and then have to wait, idly, for some period of time while his data gets to the destination, and while the message indicating the re-opening of the window is transmitted from the receiving TCP. Since this network-imposed long delay causes the sending TCP to have to be idle for some period of time, it holds down the throughput. So it seems that all things considered, simply routing high-throughput application traffic on the path of maximum excess capacity is unlikely to actually result in high throughput.

If we really wanted to do single-path throughput-oriented routing, we would need something like the following. We would want to route traffic on the shortest path (i.e., the path of least delay) which does not contain any components whose available capacity is too small to handle the needed throughput. This would prevent us from choosing a path with arbitrarily long delays, or a path with too little capacity. Unfortunately, it is almost impossible to find out either what throughput is needed by an application, or to find out just what the capacity of particular components of the internet is. We might want to consider some strategy such as not sending batch traffic on paths which include components which are very heavily loaded. This is fertile ground for experimentation. Our present point, however, is that the delay-oriented SPF routing of the ARPANET already provides the basic structure that we need to accommodate this sort of strategy. If we knew that we wanted bulk traffic to avoid certain Pathways (e.g., Pathways with too little bandwidth), we could have SPF routing compute the shortest routes that did not include those Pathways, by using the "indexed length" scheme described in section 4.3.2. There is no need to consider different sorts of routing schemes.

4.3.4 Knowing the "Whole Picture"

The use of the SPF algorithm requires that every Switch know the complete topology of the Network Structure. That is, every

Switch must know of all the other Switches, must know which Switches are "directly connected" to which other Switches, and must know the "length" of each Pathway. This is not to say that this information is "compiled in", or even loaded in manually. Rather, it is determined dynamically, in real-time, through interpretation of the routing updates (see section 4.5). It is this uniform global knowledge of the topology and the Pathway lengths that enables each Switch to run a shortest path algorithm, while producing routes which are consistent with the routes produced by other Switches, so that routing loops do not form. The SPF algorithm does not merely tell a Switch to which of its neighbors it should send packets for destination D. Rather, it computes the entire path to the destination Switch. However, when a packet is routed, it does not carry with it the identity of the entire route, as computed by its source Switch. Each Switch just forwards the packet to the next "hop" along its route. The fact that all Switches have the same information about the topology is what ensures that this routing will be free of loops.

Since each Switch performs its routing based on a complete picture of the topology of the Network Structure, we can call this sort of routing scheme a "whole picture" scheme. In this section, we will compare "whole picture" schemes with some other schemes which do not require the Switches to have uniform global knowledge of the topology. We argue that "whole picture" schemes are always superior.

The original ARPANET routing scheme, and the current Catenet routing scheme, are not "whole picture" schemes. In these routing schemes, no Switch need have any knowledge of the topology, other than who its own immediate neighbors are, and the lengths of the Pathways to its immediate neighbors. These algorithms function as follows. When a Switch first comes up, it forms a hypothesis as to the best neighbor to which to send data for each possible destination Switch. This initial hypothesis is based only on its own local information about the lengths of the Pathways to its neighbors. It then informs its immediate neighbors of its hypotheses, and is informed of their hypotheses. It then forms a new hypothesis, based on its own local information AND the hypotheses communicated to it by its neighbors. It then exchanges hypotheses with its neighbors again, and again, and again, until its own hypotheses are in complete agreement with those of its neighbors, at which point stability is reached.

To see the difference between this sort of routing scheme and the "whole picture" scheme, consider the following situation. Suppose we have 100 people in a room, sitting in chairs which are properly lined up so that we can talk of each person's having two immediate neighbors. We also have a picture of an object, and our goal is to have ALL the people agree on the identity of the depicted object. Now we have a choice of two different procedures for bringing this about:

Procedure 1: Cut the diagram into 100 pieces, and give one piece to each person. Each person is now allowed to look at his one piece, and then form a hypothesis as to what is depicted in the full picture. Then each person can exchange hypotheses only with his immediate neighbors. Then each person can form a new hypothesis and exchange that with his immediate neighbors. The procedure terminates when all 100 people agree on what is depicted.

Procedure 2: Make 100 Xerox copies of the diagram, and distribute the copies to each person.

If we really think it is important for each person to know what is depicted in the picture, then we will certainly follow procedure 2, which will make the whole picture immediately available to all participants. Procedure 1 would only be useful as a party game. It would be quite amusing to see all the ridiculous hypotheses that are formed before all participants converge to the correct one, IF they ever do manage to converge. Even if they do converge, it might take quite a long time. We must remember that different people form hypotheses at different rates, and can communicate them at different rates. Some people may simply refuse to talk to certain neighbors at all. If one's left-hand neighbor has formed a good hypothesis, but one's right-hand neighbor has not, one's own hypothesis is likely to be thrown off the track, which in turn is likely to mislead one's left-hand neighbor into a poorer hypothesis during the next "iteration." This is not a very optimal procedure for bringing about convergence of opinion.

However, this situation is really too simple and straightforward to be truly analogous to routing. To improve the analogy, we must suppose that the picture is constantly changing, even as the people are still forming hypotheses. In procedure 2, this change is accounted for by simultaneously giving each participant a new copy of the picture. In procedure 1, changes in the picture are accounted for as follows: if the part of the picture originally given to person P has changed, then give him the corresponding piece of the same picture; he can now use this piece when forming his hypotheses, and should forget about the previous piece. When the procedures are thus modified to take account of changes in the picture, the situation described is more analogous to routing, and the advantages of procedure 2 over procedure 1 are even more pronounced.

The ARPANET's current routing algorithm is similar to procedure 2, since the whole picture is made available to each Switch. The ARPANET's original routing algorithm, and the Catenet's current one, are more similar to procedure 1; perhaps they should be called "jigsaw puzzle" algorithms. All of the problems of procedure 1 have their analogies in those routing algorithms. It should be obvious that in terms of responsiveness, accuracy, and consistency, whole picture algorithms are superior to jigsaw puzzle algorithms. Many of the problems of the original ARPANET routing algorithm, such as looping and very slow response to topological change, can be attributed to its "jigsaw puzzle" nature.

Even if one agrees that we ought to avoid "jigsaw puzzle" algorithms, one might still claim that we need not have a "whole picture" algorithm. One might wish to argue that a given Switch needs to know only the topology of a "region" which contains it. This region would be larger than a single Switch, but smaller than the set of all Switches. A region would also be geographically contiguous, so that if two Switches are in the same region, then there is a path between them which is entirely within the region. Then traffic which does not need to leave a region to get from its source to its destination is in effect routed by a "whole picture" scheme. Traffic which must leave the region, however, does not have its whole route preplanned. Switches within one region will know only how to get traffic out of the region. Other Switches in the next region will know how to get the traffic through that region, etc. It seems, one might argue, that this sort of regionalized routing scheme ought to be possible. After all, consider the analogy with ordinary road travel. If one wants to travel from Boston to Los Angeles, one need not preplan the entire route. One can just head in the general direction of Los Angeles, with no need to know anything about the roads which are close to Los Angeles until one actually gets close. A similar scheme ought to work with data.

One problem, however, with the suggested analogy, is that it does not even hold in the case of ordinary automobile travel. If one were planning an automobile trip to LA, one would want to

know about any record-setting blizzards in the midwest long before one actually approached the midwest. One would want to know about the status of Mt. St. Helen's volcano long before one approaches Oregon. One might try not to be passing through Chicago at rush hour. Avoiding any of these potential disaster areas could require quite a bit of advance planning. Of course, the amount of advance planning that one performs when travelling is a matter of personality; some people are more adventurous than others, and might actually enjoy a disaster or two along the way. Users of a data communications utility, however, whatever personality traits they may have, generally do not want their data to be sent on an adventure. Rather, they want their data to be treated with a conservatism and caution which require considerable preplanning.

In any case, the analogy between the road system and a data communications network is very misleading because of the very rich interconnectivity of the road system. No matter how many problems an automobile driver encounters as he approaches Los Angeles, he still has a large number of choice points, in that he can take any number of relatively short detours around problem areas. In data networks, however, the connectivity is much less rich, and the closer the data gets to its destination, the fewer choice points there are. With a sufficiently sparse connectivity, the entire path could even be determined by the very first routing choice that is made, so that no detours around

problem areas are possible once the "trip" begins. The situation is as if someone drove from Boston to Nevada, then found that all roads from Nevada to California were closed, and that he then had to drive all the way back to Boston to start on a new route to California. This sort of sub-optimality is inherent to any regionalized routing scheme for data communications networks.

In fact, the situation could be even worse. If Switches in Boston know nothing about what is happening between Nevada and California, then data for California which arrives at Nevada and then is sent back from Nevada to Boston for alternate routing will just loop back to Nevada. The data will be stuck in an infinite loop, never reaching its destination. In IEN 179, Danny Cohen proposes a regional routing scheme like this, apparently not realizing that it suffers from loops. His proposal also includes a form of hierarchical addressing which is closely bound up with routing, so that a Switch in Boston might not even be able to distinguish data for Nevada from data for California. That is, in Cohen's scheme, data for Nevada and data for California would be indistinguishable at the Boston Switches; all such data would appear to be addressed to Nevada. Only the Switches at Nevada would look further down the address hierarchy to determine whether the data needs further forwarding to California. Any such scheme is hopelessly loop-prone, except in a Network Structure whose connectivity is extraordinarily rich, much more so than the Catenet's will ever be.

It might seem like these objections would also have to apply to the internet, since a gateway does not know all about IMPs and packet radios and SIMPs, etc., in the component networks. However, the looping problem is avoided in the internet since it is organized in a strict hierarchy of Network Structures. Switches in one Network Structure need not know anything about Switches in any other Network Structure, but they must have complete information (Whole Picture) about Switches in the same Network Structure. All (source or intermediate) Switches in a particular Network Structure always route data to a Switch in that same Network Structure. This imposition of strict hierarchy prevents looping, as long as the lower levels of hierarchy are controlled by the higher levels. In the internet, this means that, e.g., if a gateway hands a packet to an ARPANET IMP for delivery to an ARPANET Host or to another internet gateway, the ARPANET is required to deliver the packet as specified by the gateway, or to say why not. It must not simply pass the packet back to the gateway, or a loop will form. (This sort of looping has been frequently noticed between IMPs and port expanders.) This does not imply that an ARPANET IMP cannot pass a packet to an internet gateway for delivery (through an "expressway network") to another ARPANET IMP, but only that once an internet gateway decides to send a packet into the ARPANET, the ARPANET must get that packet to the intended destination, or else inform the gateway that it cannot do so.

It is also important to note that the hierarchical levels in the internet tend to be "horizontal", rather than "vertical". That is, in an internet spanning North America, there would be internet gateways located all across the continent, as well as IMPs and packet radios and PSATs located throughout the continent. This is quite different from regionalization, in which Switches which are close geographically are in a common region. This distinction is very important if we are to avoid such problems as looping.

Although building the internet as a strict hierarchy of Network Structures avoids the problems of looping, there is always some degree of sub-optimality introduced whenever the topological knowledge of the Switches is restricted in any way, even if the restriction is just to Switches within the same Network Structure. This is a point to which we return in section 4.6. where we discuss some of the basic limitations of internetting.

4.4 Measuring Pathway Delay

One of the most basic problems in devising a scheme to do delay-oriented routing is to figure out a way to determine the delay. In the ARPANET, the delay measurement algorithm is quite straightforward. When a packet arrives at an IMP, it is stamped with its arrival time. When it is transmitted to the next IMP, it is stamped with the time of transmission. ARPANET packets are

buffered in an IMP until acknowledged by the next IMP; if a packet has to be retransmitted, its transmission time stamp is overwritten with the time of latest transmission. When the packet is acknowledged by the receiving IMP, the arrival time is subtracted from the transmission time, yielding the total time the packet spent in the IMP. The propagation delay (i.e., the speed of light delay along the phone line from one IMP to the next) is then added in to compute the total amount of time it took to get the packet from one IMP to the next. There are three important aspects of this delay measurement algorithm:

1) It is necessary to measure the amount time each packet spends within the Switch. This should be as easy to apply to a gateway as to an IMP.

2) It is necessary to determine how long it takes a packet to travel from one Switch to another over the Pathway connecting them. If the Pathway is a telephone line, as in the ARPANET, this is just the propagation delay, and is a constant which can be separately measured and then stored in a table. On the other hand, if the Pathway is a packet-switching network, or even an internet, this is much more difficult to determine, and is certainly no constant.

3) There must be some way to account for packets that don't get through, or don't get through immediately, due either

to errors or to congestion. In the ARPANET, if a packet doesn't get through on its first IMP-IMP transmission, and has to be retransmitted 200 ms. later, this 200 ms. gets added into the packet's delay. This is a very important feature, since it enables the delay measurement to reflect the effect of congestion or of a very flakey line. But unless the gateways run a reliable transmission protocol among themselves, it will be difficult to make sure that our delay measurement really reflects these factors. If we are trying to send data through a network which is dropping most of the data we send it, we want to make sure that our delay measurement routines produce a high value of delay, so that traffic will tend to be routed around this very flakey and unreliable Pathway. (Remember that if too much traffic is dropped, some (higher) level of protocol will have to do a lot of retransmissions, resulting in very high delays and low throughputs.)

The problem of how to measure delay is more tractable in the case of AREA ROUTING than in the more general internet case. Recall that by "area routing," we mean a sort of internet all of whose component networks are basically identical (see IEN 184). For example, we might at some future time decide to divide the ARPANET into areas, connected by gateways, so that the ARPANET itself turns into a hierarchical network. If we decide to use

the same routing algorithm at the high level (i.e., among the intra-ARPANET gateways) as we use at the lower level (i.e., among the individual IMPs in a particular area), then the gateways could obtain the delay measurement information directly from the routing updates sent by the individual IMPs. That is, the lower level routing algorithm could provide information to the gateways enabling them to deduce their delay to other gateways. If the gateways are also ordinary IMPs, this information is automatically available. If the gateways are hosts on the low level ARPANET, a special protocol would have to be developed to enable the IMPs to transmit the routing updates to the gateways they are connected to (though this wouldn't be much different from the protocol that IMPs now use to transmit routing updates to their neighboring IMPs). Of course, if we were to implement a scheme like this, we would still want to make the ARPANET appear as a single Pathway (with no intermediate Switches) at the level of the Network Structure of the Catenet. That is, the Catenet would be a third hierarchical layer over the two hierarchical levels of the ARPANET, which would be transparent to it.

In the more general internet case, we cannot rely on the component networks to provide us with the sort of delay information we would like to use for the internet routing algorithm; the internet Switches will have to have some way of gathering this information themselves. In general, it will not be possible for a Switch to measure the one-way delay from itself

to its neighbors. (We wouldn't want to rely on the radio clocks that are now beginning to be deployed at the gateways; while these might be useful for doing measurements, we wouldn't want the reliability of the entire operational internet system to depend on a radio broadcast over which we have no control.) It is possible, however, to measure round-trip delay between each pair of neighboring gateways. In the ARPANET, for example, round-trip time is easily measured by keeping track of when a message is sent to a neighboring gateway, and then noting the time when the RFNM is received. One-way delay would be approximated by dividing the round-trip delay in half.

It is certainly true that the round-trip delay is not, in general, exactly twice the one-way delay. However, it seems like a good enough approximation to use in the internet routing algorithm. All we really require is that it be roughly proportional to the one-way delay, in that both one-way and round-trip delays tend to rise and fall together, and that congestion in the Pathway (component network) tends to make both increase. Of course, before designing the precise delay measurement scheme that we would want to use in the internet, we would have to run a series of tests and experiments to see which of several possible delay measurement algorithms gives us the results we want. This would be similar to the extensive testing of the ARPANET's delay measurement algorithm that is documented in [4].

Unfortunately, there are many networks which do not return anything like RFNMs that could be used to gauge even the round-trip delay. (Many networks, e.g., SATNET and the forthcoming wideband network, do not even tell you whether you are sending traffic to a host which is down.) So we will need a gateway-gateway protocol in which gateways receiving data from other (neighboring) gateways send back replies which can be used for timing.

This does not mean that every packet sent from one gateway to another must be acknowledged by the receiving gateway. Rather, we would propose something like the following. Suppose we have, as part of the gateway-gateway protocol, a bit that a sending gateway can set which requires the receiving gateway to acknowledge the packet. The sending gateway can have a random number generator, which lets it select packets at random in which to set this bit. These packets will have their round-trip delay measured, and will constitute a random (and hopefully a representative) sample. The packets need not be buffered in the sending gateway pending acknowledgment, but they will need to have unique identifiers so they can be kept track of. The round-trip delay of each packet is then easily determined when the acknowledge is received. (This probably implies though that gateways will have to run a protocol with their neighbors when they first come up in order to synchronize sequence numbers to use for identifying packets uniquely.) There will also have to

be a time-out, so that a packet which is not acknowledged within a certain amount of time (perhaps dependent on the expected delay of the packet, based on previous measurements) will be considered to have been lost on the Pathway between gateways (or in the receiving gateway). Packets which have been lost should be assigned a very high delay, so that the routing algorithm assigns a very high delay to Pathways which lose a lot of packets. This will tend to cause internet traffic to avoid such Pathways. There doesn't seem to be any problem in principle with a scheme like this, but we will probably need to do some statistical analysis in order to determine the best random sampling technique, and to figure out how many packets we might need to keep track of during some period of time (i.e., how big a table do we need to keep track of packets which are awaiting acknowledgments?).

This sort of random sampling can also be used as part of a Pathway up/down protocol. If a certain percentage of the sampled packets do not get through, it might be good to assume that the Pathway is not of sufficient quality to be operational, and should appear to be down as far as the internet routing algorithm is concerned. In the absence of real data traffic, we could run the up/down protocol with randomly generated test packets. Randomly generated test traffic or randomly sampled data traffic will give us a better result than periodic test traffic, since measurements based on random sampling are less likely to be correlated with other network phenomena.)

After we compute the delay for individual packets, we still face the following two questions:

1) The delay which the routing algorithm assigns to a particular Pathway will be a function of the measured delays of the individual packets sent on that Pathway. But what function should it be?

2) Once a Switch determines the delay on the Pathways emanating from itself, it must inform all other Switches of these values (in routing updates). What protocol should it use for disseminating these updates?

The second question will be discussed in section 4.5. The remainder of this section will deal with the first question.

After measuring the delays of individual packets, the individual delays must be put through some sort of smoothing function before they can be used as input to the routing algorithm. For example, in the ARPANET, we take the average, every 10 seconds, of the delays experienced by all the packets traversing a particular line in the previous 10 seconds. This average is used as input to the routing algorithm (i.e., it is assigned as the "length" of the line when the shortest-path computation is run.) We didn't choose this smoothing function at random; we chose it because it meets certain desiderata. Our real purpose in measuring delay on a particular line is to enable

us to predict the delay that will be seen by packets which are routed over that line in the future. Knowing the average delay during some period in the past is of no value except insofar as it enables us to make predictions about the future. We found in the ARPANET that for a given level of traffic, the delays experienced by the individual packets would vary quite a bit, but the delay when averaged over 10 seconds stayed relatively constant. (It is interesting that everyone who does measurements of individual packet delay always discovers this large variance, and always expresses great surprise. This "surprising" result is so often re-discovered that it should cease to be a surprise.) When designing the delay measurement routines for the ARPANET, we investigated some other smoothing functions (everyone seems to have his own favorite), but none gave more reasonable results than the simple average we adopted (which is not a running average, but rather starts over again from scratch every 10 seconds). We also tried averaging periods of less than 10 seconds, but found what we regarded as too much variation, even when the traffic load was stable.

Note that if we take an average every 10 seconds, we cannot react to a change of conditions in less than 10 seconds. and we are often criticized by people who claim that it is important for routing to be able to react more quickly. Our reply, however, is simply that it takes 10 seconds to be able to detect a significant change in delay. Averages taken over smaller periods

show too much variation under constant load to be useful in predicting the future delay, and hence are not useful in routing. In other words, averages taken over smaller periods give spurious results, "detecting" changes when in fact there are none. We want to change routing in response to real changes in network conditions, but not in response to the normal range of stochastic variations in delay. Any change in routing made on the basis of a shorter-term average is at least as likely to be harmful as to be helpful. That is, if we attempt to make routing changes based on delay data which is not sufficiently smoothed, we are really making changes at random, since we have left too much random variation in the delay data. And it seems that a good routing algorithm should not make changes at random. Of course, it would be nice if we could make routing changes instantaneously based on instantaneously detected changes in real network conditions, but this is not possible simply because there is no instantaneous way of detecting important changes in network conditions.

It is important to realize, however, that the measurement periods in the various IMPs are not synchronized. Although a given IMP generates updates no more often than every 10 seconds, some IMP or other is generating an update about every 500 milliseconds. Mathematical analysis indicates that synchronized measurement and updating periods should be avoided, since they give worst case performance [4].

There are other important reasons for not making routing changes too often. During the lifetime of a single packet in the network, we want the routing to be relatively constant, so that the packet can get to its destination without having to take too many detours. If we changed the routing every millisecond, for example, a single packet in transit though the network would experience many routing changes while in transit, which would probably cause it to have a longer delay than necessary. The rate at which we change routing should be low relative to the average transit time of a packet through the network. Another reason for not making routing changes too frequently has to do with the time it takes routing updates to travel around the network. We want to make sure that the information carried in a routing update is not totally obsolete by the time the update is received. This implies that the smoothing interval for delay measurements has to be long relative to the time it takes updates to traverse the network.

In the ARPANET, 10 seconds is much longer than the amount of time it takes to get updates around, or the amount of time a packet spends in transit in the network. We chose 10 seconds as the averaging interval because it seemed to be the shortest period that was long enough to give us a reasonable amount of smoothing. If we think that in the internet, however, average transit times might be measured in the tens of seconds, we may have to make our smoothing interval considerably longer than 10

seconds, perhaps as long as a minute. This could seriously limit the responsiveness of the routing algorithm to changing network conditions. However, there is nothing we can do about this. THE LONGER IT TAKES PACKETS TO TRAVEL AROUND A NETWORK, THE LESS RESPONSIVE THE ROUTING ALGORITHM OF THAT NETWORK CAN BE, for the simple reason that it will just take longer to disseminate the information needed for routing around the network. The transit time of a network places an upper limit on the responsiveness of that network's routing algorithm. Any attempt to exceed this upper limit (with kludges or heuristics) will just be futile, and will result only in unstable and mysterious behavior on the part of the routing algorithm, reducing, rather than increasing, performance.

This is not to say that each Switch must generate routing updates as often as every 10 seconds. If there is no change in delay from one 10-second period to another, then there is no reason to generate an update. Or if there is a change, but it is not "significant", then there is no reason to generate an update. In the ARPANET, a delay change is considered to be significant if it exceeds a certain (parameterized) threshold. We devised a scheme wherein the threshold decreases with time, so that a very large change is always "significant", but a small change is significant only if it persists for a long time. Of course, routing updates must be generated not only in response to measured changes in delay, but also if a line goes down or comes up.

We would expect that the details of the delay measurement and smoothing algorithms will have to be different in the internet than in the ARPANET, but the principles outlined above would seem to apply in the internet environment also. WE WILL HAVE TO DO SOME CAREFUL EXAMINATION OF THE DELAY-THROUGHPUT CHARACTERISTICS OF EACH OF THE INDIVIDUAL NETWORKS THAT ARE USED AS PATHWAYS IN THE INTERNET, and it may be that somewhat different smoothing algorithms will have to be used for the different kinds of Pathways. However, there doesn't seem to be any problem in principle with doing this sort of delay measurement.

An interesting issue arises if a given pair of gateways is connected by two or more distinct Pathways. For example, two gateways might both be connected to ARPANET and SATNET, so that each can be reached from the other by either of those two networks. Or, a gateway might be multi-homed on the ARPANET, so that it has two distinct access lines over which it can reach all the other ARPANET gateways. In such cases, do we want to separately report the delay on each of the distinct Pathways, or do we want (at the level of routing) to represent the connection between each pair of gateways as a single, unique line, whose delay is some function of the delay of the distinct Pathways which really exist? This issue is a generalization of an issue we have been looking at in the context of the ARPANET, which we call "parallel trunking." In parallel trunking, a single pair of

IMPs is connected by two or more trunks, and the same issue of how to represent them in routing (as individual trunks, or as a single, composite, trunk) arises. When the trunks are telephone lines, the problem is relatively easy to deal with. Routing can treat them as a single trunk, with a delay which is the average delay of all packets sent over the composite trunk. The actual decision as to which particular component trunk to use for transmitting a particular packet can be made locally, by the IMP to which the parallel trunks are connected; there is no need for routing to play a role in this decision.

In the case where the parallel trunks are of comparable lengths (so that there is not much difference in the propagation delays), the trunks can serve a common queue according to the standard FIFO single-queue multiple-server discipline. If the trunks are more heterogeneous, say one is a terrestrial line and one is a satellite line, a somewhat more complex queuing discipline is required. We would like to avoid using the satellite line until the load is such that if only the terrestrial line were used, packets would experience a delay comparable to that they experience over the satellite line. With this sort of queuing discipline, packets sent to the composite line experience a delay which is independent of the particular component (land-line or satellite line) that they use. That is, no packet is forced to suffer the quarter-second satellite delay unless the terrestrial line is so backed up that the delay for

packets sent over it is comparable to the delay of packets sent over the satellite line. This sort of scheme seems to ensure the best delay performance for the composite trunk. (Actually, the mathematics of queuing theory suggests that a smaller average delay for the composite trunk might be achieved by starting to use the satellite line sooner. That is, a somewhat smaller average delay might be achievable if a few packets are given a much longer delay by being forced over the satellite line sooner than they would be with the queuing discipline we suggested. Considerations of fairness would seem to rule that out, however; how would you like it if your data got a much higher delay so that someone else's could get a slightly smaller one? In addition, the queuing discipline we suggested would seem to produce a smaller variance in delays, thereby making the measured average delay on the composite trunk a better predictor of future performance, and the better we can predict future performance, the better performance our routing algorithm can provide.)

Basically, there is no reason for routing to be aware that a particular line consists of several parallel components rather than a single component, because, if the argument above is right, any decision as to which component to use can be best made locally, at the IMP from which the parallel lines emanate. That is, the global routing algorithm cannot really make effective use of information about which lines consist of parallel components, and should not be burdened with information that it cannot use.

This is good, since the SPF algorithm cannot really handle parallel lines between a pair of Switches except by representing them as a single line. (A careful study of the algorithm would show that much of the algorithm's space and time efficiency would be sacrificed if it had to be modified to handle parallel trunks as separate trunks. Since this efficiency is the main thing that recommends the SPF algorithm over other shortest-path algorithms, we must be sure that we don't destroy the effectiveness of the algorithm by making poorly thought-out changes to it.)

In the internet environment, however, we have a more complex problem with parallel trunks than in the ARPANET. The scheme we outlined for using parallel trunks in the ARPANET depends on our being able to know when the load on the composite trunk is such that exclusive use of the faster component would cause delays that are just as high as we get when we use the slower component. This is not difficult to know if the components are phone lines of one sort or another, since the relation between load and delay is pretty well-defined if we know the length of the lines and their capacity. If the components of a parallel "trunk" are really packet-switching networks, however, it is much harder to figure out which components are slow and which are fast, and it is hard to figure out when the load on the fast component is such that we have to start using the slow one.

It seems that by separately measuring the delays obtained over the "parallel trunks" in the internet case, we ought to be

able to devise some algorithm for splitting the traffic among the parallel components in a way which gives reasonable delay/throughput performance. However, we don't yet have a solution to this problem, which we will put aside for the present. Whatever scheme we eventually decide on, however, should be compatible with treating the parallel components as a single line at the level of routing. Of course, if we decide to have different routes for different traffic types (say, excluding satellite networks for interactive traffic, but using them for batch traffic), then the problem is eased somewhat since we partially solve the problem a priori. There would still be no need to represent the parallel lines as separate lines. Rather, we would represent them as a single line, with different delay characteristics for different traffic types.

4.5 Routing Updates

4.5.1 Overhead

Everyone seems to be in agreement that the overhead due to routing updates should be kept low. At least, no one seems to advocate that the overhead should be made high. Unfortunately, "apple pie" pronouncements like this aren't much help in actually designing a routing scheme. In evaluating a routing algorithm from the perspective of overhead, one must understand the way in which overhead is traded off against functionality.

One advantage of the SPF routing algorithm is that it provides a lot of handles that can be used to control overhead. In SPF routing, a routing update generated by a particular Switch identifies each neighbor of that Switch, and gives the delay over the Pathway to that Switch. Thus the size of an update generated by a particular Switch is proportional to the number of neighbors that the Switch has, generally a fairly small number (no more than 5 in the ARPANET, and probably of a similar magnitude in the internet).

In the current Catenet routing algorithm, the size of the routing updates is a function of the total number of gateways (or equivalently, of the total number of component networks), a number which can increase by a great deal over the years. In the SPF algorithm, the size of the updates is a function of the connectivity of the internet, which could not increase anywhere near as much or as rapidly as the number of gateways. (In the two years that SPF has been running in the ARPANET, the number of IMPs has increased by a third, with another similar increase expected in the next several months, while the connectivity, and hence the average update size, has remained relatively constant.) This is important, since we wouldn't want to get ourselves into a situation where the update size eventually becomes so big (due to network growth) that we can no longer fit a whole update into a single packet (a situation that was imminent during the last days of the original ARPANET routing algorithm.) In the internet, the

maximum size of an update packet is constrained by the component network which has the smallest maximum packet size. It seems likely that any component network whose packets are large enough to carry the enormous TCP and IP headers should have no trouble carrying the routing updates.

The amount of overhead due to routing updates is not only a function of the update size, but also of the rate at which updates are generated. In the ARPANET, since each IMP averages the delay on its outgoing lines over a period of 10 seconds, changes in delay on the lines emanating from a particular IMP cannot occur, by definition, more often than once every 10 seconds. In addition to generating updates when the delay changes, updates must also be generated when lines go down or come up. In the ARPANET, a line which goes down cannot come up for at least 60 seconds. So in an IMP with 5 neighbors, the most updates that can be generated in a minute is 11 (due to each of the lines either going down or coming up during the minute, for 5, and a delay change every 10 seconds, for 6). It is important to note that this is the maximum rate at which updates can be generated, not the average rate. Since IMPs need not generate routing updates unless they have a "significant change" in delay to report, the average rate can be much lower. In the ARPANET, the average rate for generating updates is actually about one per IMP per 40 seconds. This is a very limited amount of overhead. Of course, the overhead will increase as the number of IMPs

increases, because there are just more IMPs to generate updates. However, the amount of overhead is always under our control, since we can always alter the averaging interval, or the threshold of significant change in delay, to force updates to be generated less frequently and thereby to reduce overhead. These same principles apply to the internet also, so it doesn't seem as if we will be generating enormous amounts of routing overhead.

There are some things we might want to do which would tend to make the routing updates longer than so far indicated. For example, if we defined several priorities of traffic at the internet level, and mapped these priorities to different priorities of some particular component network, we might want to separately measure the delay across that network for each priority. We might also want to compute a separate set of routes across the internet for each priority. If we adopted some such scheme, we would need to report in each update several different delays for each Pathway, indexed by priority. These indexed delays could then be used for computing a set of routing tables indexed by priority, allowing traffic of different priorities to use different routes. Of course, this would lengthen the updates, adding more overhead. Part of the decision as to whether to adopt such a scheme would involve an evaluation of the trade-offs between the cost of this increased overhead and the benefit of the expected improvement in performance. The issues, however, are clear, and there are enough handles controlling the

amount of overhead so that we can put into effect any decision we make.

It is important to understand that the number of routing updates generated by a single internet event (such as the outage of a gateway access line) is much less with SPF routing than with the current Catenet routing algorithm. In SPF routing, a given event causes the generation of ONE routing update, which must then be sent to every gateway (thereby giving each gateway an up-to-date copy of the "whole picture"). On the other hand, in the current Catenet routing algorithm, a single internet event causes a flurry of updates, as all gateways send and receive updates repeatedly to and from each neighbor, until the routing tables stabilize and the process settles down. This can take quite a long time and quite a few updates, particularly if the number of gateways is large.

In addition, in an internet with a large number of gateways, the updates for the current Catenet routing algorithm are very much larger than the SPF updates would be. It is clear that the routing overhead due to a single network event would be much less with SPF than it currently is. However, if we plan to send routing updates when delay changes, as opposed to just when a gateway access line comes up or goes down (as at present), then we will be generating updates in response to more network events. This tends to drive the overhead up. Again, the trade-offs are

relatively clear here; the amount of overhead simply trades off against the responsiveness of the routing algorithm to changing network conditions. The decision as to how to draw this trade-off can be made as a policy decision, and can be changed if performance considerations warrant it. The situation with the current Catenet routing algorithm is quite different, since the amount of overhead that it generates is almost impossible to compute. In that algorithm, the number of routing updates generated in response to a particular event depends on the order in which the updates are processed by the individual gateways, something that is essentially random and hence hard to predict. The SPF algorithm has no such dependency.

The need for hysteresis in the Pathway up/down protocol run between neighboring gateways is worth emphasizing. If connections between neighboring gateways are allowed to come up and go down with great frequency, causing a constant flurry of routing changes, packets in transit will bounce around a lot. Putting a limit on the frequency with which a gateway-gateway connection can change state is needed not only to limit the amount of overhead generated, but also to give some stability to the routing. It is worth noting that the ARPANET, although providing hysteresis in its own line up/down protocol, does not provide any hysteresis in host up/downs. Hosts are allowed to go down and come up repeatedly many times a minute, and this does

result in problems, causing congestion and instability. Hysteresis in the gateway's Pathway up/down protocol will have to be ensured explicitly; we cannot rely on the ordinary host access protocol of the component networks to do the right thing. That is, if a network interface goes down, we must keep it down for a period of time, even if the network itself allows the interface to come back up immediately.

4.5.2 Protocol

We turn now to the problem of how to disseminate the routing updates around the Network Structure. Remember that the updates generated by a particular Switch will contain information about the delays to the neighbors of that Switch. When a Switch generates an update, it must broadcast that update to ALL other Switches. As a result, every single Switch will know the values of delay between every single pair of neighboring Switches. It is then straightforward to have each Switch run a shortest-path algorithm which determines the shortest path from itself to each other Switch. The basic idea is for each Switch to know the entire topology of the Network Structure, so that the shortest paths can be determined by a localized shortest path algorithm, with no need for a distributed computation. In the ARPANET, the IMPs do not start out with any knowledge of the topology. They determine who their own neighbors are, and they reconstruct the rest of the topology from the routing updates they receive.

It is possible to prove that, as long as all Switches have the same information about the topology and the delays, then they will produce routes which are consistent and loop-free. (That is, the situation in which Switch A thinks its best path to B is through C, and C thinks its best path to B is through A, can never arise.) However, if some routing updates somehow get lost before being received by every single Switch, then there is no guarantee of consistent loop-free routing. In fact, if routing updates get lost, so that different Switches have different information about the topology or the delays, we would expect long-term routing loops to arise, possibly making the Network Structure useless for some period of time. So the protocol used to broadcast the routing updates needs the highest possible reliability. Of course, it will always take some amount of time for an update to be broadcast around the Network Structure, and during that time, some Switches will have received it and some not. This means there will always be a transient period when routing loops might arise. So another aim of the routing updating protocol must be to keep this transient period as short as possible. In the ARPANET, we have an updating protocol which seems to provide these characteristics of extremely high reliability and low delay. Some of its aspects adapt readily to the internet, but others are more difficult to adapt. In what follows, we first describe the ARPANET's routing updating protocol, and then discuss its applicability to the internet.

Suppose IMP A has to generate a routing update, either because of some "significant" change in the measured delay, or because of a line up/down state change. Each update generated by A has a sequence number, which is incremented by 1 for each new update. (In the ARPANET, we use 6-bit sequence numbers, which wrap around after 63.) After creating the update, IMP A sends it to each of its neighbors. The update is transmitted as a packet of extremely high priority; only the packets used in the line up/down protocol are of higher priority. We use the notation "A(n)" to refer to the update generated by IMP A with sequence number n. Now let's look at what happens when a copy of update A(n) is received by an IMP B. (IMP B is intended to be an arbitrary IMP somewhere in the network, possibly identical to A or to one of A's neighbors, but not necessarily so.) If B has never received an update from A before, it "accepts" A(n), by which we mean that it (a) remembers in its tables that the most recent update it has seen from A is A(n) (i.e., the sequence number n, the list of neighbors of A, and the delays from A to each neighbor are stored in B's tables), (b) it forwards A(n) to each of its neighbors, including the one from which it was received, and (c) the SPF algorithm is run to produce a new set of paths, given the new delay and topology information contained in A(n). If B has received an update from A before, it determines whether A(n) is more recent than the update it has already seen, and "accepts" it (as just defined) if it is;

otherwise it simply discards A(n). The determination as to whether A(n) is more recent than some previously received update A(m) is made by a sequence number comparison (which, of course, must account for the fact that sequence numbers can wrap around); A(n) is not considered to be more recent than itself.

If one thinks a bit about this inductive definition of the protocol, one sees that each IMP in the network will receive every update which is generated by any IMP, and further that it will generally receive a copy of each update on each of its lines. This means of broadcasting an update from one IMP to all other IMPs is called "flooding." It is highly reliable, since updates cannot be lost in the network due to IMP crashes or partitions. If there is any path at all between two IMPs, flooding will get the update from one to the other. (Of course, if there is no path at all from A to B, then updates cannot get from one IMP to the other. However, this is not a problem, since if traffic from A cannot even reach B, then it cannot use B's outgoing lines, so there is no need for A to know the delays of B's outgoing lines in this case. In saying that flooding prevents updates from getting lost due to network partitions, we are thinking of the case where an update is in transit from A to B when a partition forms, such that A and B are in the same partition segment, but the update is in a segment which is now isolated from either A or B. Flooding ensures delivery in this situation.)

Flooding also ensures that an update travels over the shortest (in terms of delay) possible path. Basically, every possible path is attempted, so the update necessarily gets through first on the shortest path, by definition. In addition, this means of transmitting routing updates does not depend in any way on the routing algorithm itself. Since routing updates are sent out all lines, there is no need to look in the routing tables to decide where to send the routing update. The transmission of routing updates is independent of routing, which eliminates the possibility of certain sorts of disastrous negative feedback.

One might think that a protocol which sends a copy of every update on every line creates a tremendous amount of overhead. In the ARPANET, however, the average update packet size is 176 bits, and the average number of updates sent on each line (in each direction) is less than 2 per second, for an average overhead of less than 1% of a 50 kbps line. And this is with almost 75 IMPs generating updates.

Of course, a protocol like flooding is only as reliable as are the individual point-to-point transmissions from IMP to neighboring IMP. We ensure reliability at this level with a positive acknowledgment retransmission scheme. Note, however, that no explicit acknowledgments are required. If IMP X sends update A(n) to neighboring IMP Y, and then X receives from Y an

update A(m), where A(m) is at least as recent as A(n), we consider that Y has acknowledged X's transmission of A(n). Since an IMP which accepts an update sends it to all neighbors, including the one from which it was received, in general, if X sends A(n) to Y, Y will send A(n) back to X, thereby furnishing the acknowledgment. We say "in general", since there is a little further twist. As another reliability feature, we make each update carry complete information, and forbid the carrying of incremental information in updates. That is, each and every update generated by an IMP A contains all the latest information about A's neighbors and its delay to them, so that each update can be fully understood in isolation from any that have gone before. This means that if update A(n+1) is received and processed by some IMP B, then the prior update A(n) is superfluous and can just be discarded by B. In particular, if IMP X sends A(n) to neighboring IMP Y while at the same time Y is sending A(n+1) to X, then X can interpret the receipt of A(n+1) from Y as an acknowledgment of the receipt of A(n); that is, X no longer has to worry about retransmitting A(n), since that update is no longer needed by Y. If no "acknowledgment" for an update is received from a particular neighbor within a specified amount of time, the update is retransmitted. Of course, it must be specially marked as a retransmission, so that the neighboring IMP will always "acknowledge" it (by echoing it back), even if the neighbor has seen it before. This is needed to handle the case

where the update got through the first time, but the acknowledgment did not. It should also be noted that all the information in a routing update must be stored in each IMP's tables in order to run the SPF computation. This means that if it is necessary to retransmit an update to a particular neighbor, the update packet can be re-created from the tables; it is not necessary to buffer the original update packet pending acknowledgment.

We must remember that if congestion forms in some part of the network, we want routing to be able to adapt in a way which can route traffic around the congestion. For this to have any hope of working, we must be sure that ROUTING UPDATES WILL BE ABLE TO FLOW FREELY, EVEN IF CONGESTION IS BLOCKING THE FLOW OF DATA PACKETS. Therefore, routing updates in the ARPANET are not sent by the ordinary IMP-IMP protocol, which provides only 8 logical channels between a pair of IMPs. That would be disastrous, since congestion often causes all 8 logical channels to fill up and remain filled for some time, blocking further data transmission between the IMPs. Transmission of routing updates must be done in a way that is not subject to this sort of protocol blocking during periods of congestion. (This sort of "out-of-band" signalling was quite easy to put into the ARPANET. However, it is worth noting that such protocols as HDLC make no explicit provision for out-of-band signalling, and it seems that many networks are being built in which the routing updates will

not be able to flow when the network gets congested. Designers of such networks will no doubt be quite surprised when they discover what is inevitable, namely that their routing algorithms break down completely in the face of congestion.) We also want to be sure that we have enough buffers available for holding routing updates, and that we process them at a relatively high CPU priority.

There is one more twist to the updating protocol, having to do with network partitions. A network partition is a situation in which there are two IMPs in the network between which there is no communications path. Network partitions, in this sense, may be as simple as the case in which some IMP is down (an IMP which is down has no communications path to any other IMP), or as complex as the case in which four line outages result in partitioning the network into two groups of 40 IMPs. When a partition ends, we have to be sure that the two (or more) segments do not get logically rejoined until routing updates from all IMPs in each segment get to all the IMPs in the other segments. That is, data packets must not be routed from one segment to the other until all IMPs in each segment have exchanged routing updates with all IMPs in the other segments. Otherwise, routing loops are sure to form. We must also remember that the sequence numbers of IMPs in one segment may have wrapped around several times during the duration of the partition. Therefore we must ensure that IMPs in one segment do not apply

the usual sequence number comparison to updates from IMPs in the other segment.

We have dealt with these problems by adding the following three time-outs to the updating protocol:

1) MAXIMUM INTERVAL BETWEEN UPDATES: Every IMP is required to generate at least one update every minute. whether or not there has been any change in delay or line state.

2) MAXIMUM UPDATE LIFETIME: If an IMP B has not received any updates generated by IMP A for a whole minute, then B will "accept" the next update it sees that was generated by A, regardless of the sequence number.

3) WAITING PERIOD: When a line is ready to come up, it is held in a special "waiting" state for a minute. While in the waiting state, no data can be sent on the line. However, routing updates are passed over the line in the normal way, as if the line were up.

Since the ending of a partition is always coincident with some line's coming up, these three features ensure that a partition cannot end until a full exchange of routing information takes place. They also ensure (given the facts that there is a 6-bit sequence number space and that IMPs can generate at most 11 updates per minute) that sequence numbers of updates generated after the end of the partition are not compared with sequence numbers of updates generated before the partition occurred.

The general idea of flooding the updates seems as important in the internet as in the ARPANET. In general, we can expect the internet to be subject to many more mysterious outages and disturbances than is the ARPANET, and the reliability and speed of flooding will be essential if an internet routing algorithm is to have any hope of working. The issue of overhead may be somewhat worrisome, though. If an IMP has to send each of 4 neighbors a copy of each update, it is just a matter of sending a copy of a small packet on each of 4 wideband lines. On the other hand, if a gateway has to send a copy of each update to each neighbor, this may mean that it has to send 4 copies into a single network, over a single network interface. This may be somewhat more disruptive. Of course, this problem only exists on networks which do not have group addressing. If a network allows the gateways to be addressed as a group, then each gateway needs only to place one copy of each update into the network, and the network will take responsibility for delivering it to each other gateway. (This might result in each gateway's receiving back its own copy of the update, since the sending gateway will also be part of the group, but that is no problem. As long as the gateway can identify itself as the transmitter, it can just throw away any updates which it transmitted to itself.) This idea of sending the updates to all neighbors on a particular network by using group addressing fits in well with an idea expounded in section 4.1, namely the idea that a network should be able to

tell which of its hosts are gateways, and should inform the other gateways when a new gateway come up. This same mechanism could be used by the network to augment its group addressing mechanism, to allow the group definition to change dynamically and automatically as the set of gateways connected to it changes. Unfortunately, few networks seem to have group addressing. Even SATNET has only a primitive group addressing feature, although it seems odd to have a broadcast network without full group addressing capabilities. (Group addressing is much more complex on a distributed network like ARPANET than on a broadcast network.) Perhaps as further internet development proceeds, more of the component networks will add group addressing, in order to make their use of the internet more robust and efficient.

Retransmission of routing updates on a gateway-to-neighboring-gateway basis, based on the scheme in the ARPANET, also seems to offer no problems in principle. However, the retransmission time-outs might have to be carefully chosen, and tuned to the characteristics of the network connecting the sending and receiving gateways. The retransmission time has to be somewhat longer than the average round-trip delay in that network, and this may vary considerably from network to network. In principle, however, this is no different from the ARPANET, where the retransmission timers for routing updates vary according to the propagation delay of the phone line connecting two IMPs.

There is a bit of a subtle problem that we discovered in the ARPANET, having to do with the scheme of using the updates themselves as acknowledgments. Suppose Switch A has two neighbors, B and C. A receives a copy of update u from B, and queues it for transmission to C. However, while u is still on the queue to C, A receives a copy of u from C. If A had already sent u to C, this copy from C would have served as A's acknowledgment that C had received the update. But now, with u on the queue to C, if we are not careful, A will send u to C after having received a copy of u from C. When C gets this copy of u from A it will not accept it (since it has already seen a copy of u and sent that copy on to A), which will cause A to retransmit u to C, resulting in an unnecessary retransmission.

In the ARPANET, we deal with this problem by turning on the retransmission timer as soon as an update is received, rather than when it is sent. That way, an update which is still queued for transmission when its "acknowledgment" is received will still get transmitted unnecessarily, but the retransmission timer gets shut off, causing only one, rather than two, unnecessary transmissions. A more logical scheme would be to check the transmission queue to a Switch whenever an update is received from that Switch. If a copy of the same update that was just received is queued for transmission, it should just be removed from the queue. This would prevent any unnecessary transmissions. In the ARPANET, a few unnecessary transmissions

don't really matter, but in the internet, if we really want to keep the overhead low, it is probably worthwhile trying to get this just right. We must remember that network access protocols may limit the number of packets we can get into the network during some period, which makes it all the more important to avoid sending unnecessary packets.

Suppose we find that for some reason or other, it is taking a very long time to get updates from some gateway to one of its neighbors. This would show up as an excessive number of retransmissions of updates. In such a case, we would probably have to consider that particular gateway-gateway Pathway to be down, irrespective of what our ordinary Pathway up/down protocol tells us. Remember that in order to ensure consistent and loop-free routing, we must get the updates around the internet as rapidly as possible. If updates cannot travel sufficiently rapidly on some Pathway, then we just cannot use that Pathway at all for transit within the internet. Attempting to keep that Pathway up for transit can result in relatively long-term routing loops, which could in turn cause a degradation in network performance which swamps the degradation caused by not using that Pathway at all. Especially disastrous would be a situation in which ordinary data packets could pass, but routing updates, for some reason, could not. It is hard to know what might cause such a situation (perhaps a bug in the component network that we are using as a Pathway), but it is certainly something we need to

protect against. (Note, however, that even if we declare some gateway-gateway Pathway down, it does not follow that the network underlying that Pathway cannot be used as a terminus network, to which data for Hosts can be sent and from which data from Hosts can be received. Even if some network is not usable for providing a Pathway between two gateways on it, it may still be useful for providing a Pathway between the gateways and some set of Hosts.)

We have emphasized the need to transmit routing updates as "out-of-band" signals, which bypass the ordinary communications protocols (such as the IMP-IMP protocol in the ARPANET), so that when congestion forms which causes those protocols to block, the routing updates can still flow. That is, we would like to have a protocol which is both non-blocking and non-refusing. This may be quite difficult to achieve in the internet environment, where sending an update from gateway to gateway requires us to use whatever network access protocol is provided by the Pathway network. Here our most difficult problem might be with the ARPANET's 1822 protocol, which can cause blocking of the network interface for tens of seconds. We really can't delay sending a routing update for 15 seconds or so while the IMP is blocking, so whenever this happens we would have to declare the pathway down.

In the ARPANET, we have two ways of trying to deal with this. One way would be to send all packets into the ARPANET as

datagrams, which cannot cause blocking. Another way would be to use the standard virtual circuit interface, but to obey the flow control restrictions of the ARPANET (i.e., to control the number of outstanding messages between a pair of hosts), and to avoid the use of multi-packet messages (which can cause blocking if the destination IMP is short of buffers, as ARPANET IMPs chronically are). There are other situations in which blocking can occur, but they all involve a shortage of resources at the source IMP, and in such cases declaring the Pathway to be down is probably the right thing to do. We do not want to be forced into declaring Pathways down simply because we have ignored some protocol restriction, but it seems much more sensible to declare a Pathway down if, say, the IMP to which a gateway is attached is too congested to provide reliable service for internet packets.

It is important to note that whatever restrictions we apply to our use of the network access protocol apply not only to routing updates, but also to all messages sent into the ARPANET from the gateway. It would do no good, for example, to send in routing updates as datagrams, while using non-datagrams for other packets, since this would allow the other packets to block the routing updates. At this point, it is not quite clear just what the best scheme would be. The use of datagrams enables us to get around the sometimes time-consuming but often unnecessary resequencing which the ARPANET performs before delivering packets to the destination host (it is neither necessary nor desirable

for the ARPANET to resequence routing updates before delivering them to a gateway), but it also reduces the reliability of transmission through the ARPANET, and it is not obvious how this trades off. For each network which we intend to use as a component of the internet, we will have to carefully study the details of its network access protocol, and possibly do some experiments to see how the various details of network access affect the performance, in terms of delay, throughput, and reliability of the network. Only by careful attention to the details of network access on each particular network, and by continuing measurements and instrumentation in the gateways to see if we are getting the expected performance from the component networks, can we hope to make the routing updating protocol quick and reliable enough to ensure consistent and loop-free routing throughput the internet. There are a few general principles we might appeal to, such as making routing updates be the highest priority traffic that we send into the component networks. However, it is difficult to be sure a priori what effect even so straightforward a principle might have. It's not hard to imagine a poorly designed network in which low priority packets receive better performance than high priority ones, under certain circumstances. To make the internet robust, we need to be able to detect such situations (and to gather enough evidence, via measurements, to enable us to point the finger convincingly), and we cannot simply assume that a component network will perform as advertised.

If we might digress a little, the considerations of the preceding paragraphs raise an interesting issue with respect to the use of fragmentation in the gateways. We raised the possibility of not using multi-packet ARPANET messages, and such a strategy would doubtless require more fragmentation than is presently done. Fragmentation in the gateways has long been thought of as a necessary evil, necessary because some networks have a smaller maximum packet size than others. If a gateway receives a packet from network A which is too large to fit into network B, then the gateway must either fragment it or drop it on the floor. However, perhaps fragmentation is sometimes useful as an optimization procedure. That is, some network may have a suitably large maximum packet size so that fragmentation is, strictly speaking, unnecessary. Nevertheless, the network might actually perform better if given smaller packets, so that fragmentation provides better performance. We see this in some current Catenet problems. It seems that the BBN-gateway between ARPANET and SATNET often receives packets from SATNET which are 2000 bits long, or twice the size of an ARPANET packet. The gateway then presents these messages to the ARPANET as two-packet messages. As it happens, two-packet messages generally give the lowest possible throughput on the ARPANET (a consequence of the limited buffer space at the destination IMPs and the fact that the ARPANET assumes that all multi-packet messages will contain 8 packets); the gateway could probably obtain better performance

from the ARPANET by fragmenting the two-packet message into two single-packet messages. Of course, the situation is a bit more complicated in general than this may make it seem. If messages are being sent from a source host through SATNET and then through ARPANET to a destination host, best performance might well be achieved by sending the messages as 2000-bit messages through SATNET, then fragmenting them and sending them as 1000-bit messages through ARPANET. However, what if the messages must go beyond ARPANET, through another network, which handles 2000-bit messages more efficiently than 1000-bit messages? This sort of strategy, if useful at all, is best done in combination with the hop-by-hop fragmentation/reassembly scheme suggested in IEN 187.

The part of the routing updating protocol which deals with recovery from partitions (including the degenerate case of initialization when a Switch comes up) is somewhat more tricky to apply to the internet environment. In the ARPANET, we have a number of one-minute timers. Each IMP must generate an update at least once per minute; a line that is ready to come up must participate in the updating protocol for a minute before being declared up; and an update that has been held for a minute in an IMP, with no later update from that update's source IMP having been seen, is regarded as "old", in the sense that its sequence number is no longer considered when the IMP is deciding whether the next update it sees (from the same source) is acceptable. In attempting to adapt these procedures to the internet, we must

take notice of the way in which these timers interact with each other and with other features of the internet. Consider, for example, the length of the maximum update lifetime, which determines how long an update's sequence number remains valid for the purposes of judging the acceptability of the next update. There are two restrictions on the length of this timer:

1) A Switch A should not time out an update whose source Switch is B unless there really is a partition which destroys the communication path between A and B (remember, this includes the degenerate case of a partition where B simply goes down). This means that the time-out period must be greater than the sum of the maximum interval between updates PLUS the maximum amount of time that an update from B could take to get to A.

2) The sequence numbering scheme used for the updates must be such that the sequence numbers cannot wrap around in a period of time which is less than the maximum update life-time.

In the ARPANET, the sequence numbers cannot wrap in less than a few minutes, each IMP generates an update at least once per minute, and the time to get that update to all other IMPs is negligible when compared to a minute, so a maximum update lifetime of one minute is fine. In the internet, however, we could not expect to measure transit times in the hundreds of

milliseconds; tens of seconds would be more like it. So even if we forced each gateway to generate at least one update per minute, we would still need a maximum update lifetime of several minutes. And the longer our maximum update lifetime, the larger our sequence number space must be (to prevent wrap-around), which means additional overhead (memory and bandwidth) to represent the sequence numbers.

A similar constraint applies to the "waiting period". The purpose of the waiting period is to ensure that when a gateway-gateway Pathway is ready to come up, it is not permitted to carry data until an update from each other gateway traverses it. Clearly, for this to have the proper effect, the waiting period must be longer than the sum of the maximum transit time plus the maximum interval between the generation of updates from a single gateway. We would probably also have to set this to several minutes. This does have a serious operational consequence, namely that no outage will persist for less than several minutes. This can be an inconvenience, lengthening the time it takes to put out a new software release to all the gateways, for example, and possibly affecting the MTTR statistics, but it is something we just have to live with. Note, by the way, that as long as the waiting period is at least as long as the maximum update lifetime, a gateway that restarts after a failure (or a reload) can start generating updates with sequence number 0, irrespective of what sequence numbers it was

using before, since all its prior updates will have timed out (if the timers are set right).

4.6 Limitations of Internetting

This discussion of routing in the internet points out some of the inherent limits of internetting. Good performance requires the use of a routing updating procedure which broadcasts the updates in a very reliable and quick manner. Anything that delays the routing updates, or makes their transmission less than reliable, will lengthen the amount of time during which different Switches have a different "picture" of the Network Structure, which in turn will degrade performance. We believe that the updating protocol we developed for the ARPANET solves these problems in the context of the ARPANET. It seems clear, however, that broadcasting routing updates in the internet is just going to be slower and less reliable than it is in the ARPANET. Although the same principles seem to apply in both cases, the characteristics of the internet Pathways are not sufficiently stable to ensure the speed and reliability that we really would like to have. It is going to be very hard to ensure that we can get our routing updates through the various component networks of the internet in a timely and reliable manner, and it may be hard to get the component networks to handle the internet routing updates with enough priority to prevent them from being blocked due to congestion. This is going to place a limit on internet

performance which we cannot avoid no matter what architecture we choose.

The only way to eliminate this sort of problem would be to have the component networks themselves give special treatment to internet control packets, such as routing updates. Currently, the component networks of the internet treat internet control packets as mere data. We have suggested that in some cases, it is impossible to meet certain of our goals without special help from the underlying networks. For example, in our discussion of the "gateway discovery protocol", we argued that preserving the maximum flexibility for making topological changes in the internet requires cooperation from the underlying networks. This point can be generalized, though. The more cooperation we can get from the underlying networks, the better we can make our internet routing algorithm perform, and the better we can make the internet perform. We would recommend therefore that serious consideration be given to modifying the component networks of the Catenet to maximize their cooperation with the internet.

Even if the component networks of the internet cooperate to the fullest, there is another problem which may limit the responsiveness of the internet routing algorithm. If there are very long transit times across the internet, much longer than we ever see in individual networks like the ARPANET, then the responsiveness of routing is necessarily held down. This factor

will place a natural restriction on the growth of the internet. At a certain point, it will become just too big to be treated as a single Network Structure, so that further growth would make routing too non-responsive to provide good service. That is, eventually we reach a point of diminishing returns, where adding more Switches, or even adding more levels of hierarchy, begins to significantly degrade service throughout the internet by making the routing algorithm too non-responsive. It is important to understand that the notion of "big" here has nothing to do with the number of Switches, but rather with the transit time across the internet.

If there are two Hosts which cannot, for reasons like this, be placed on the same internet, it may still be possible for them to communicate, though at a somewhat reduced level of efficiency. Each of the Hosts would have to be on some internet, but not necessarily on the same one. Suppose, for example, that there are two different internets, internet A and internet B, which cannot be combined into one larger internet because the resultant internet would be too large to permit a reasonably responsive routing algorithm. However, it is still possible for each internet to model the other one as an Access Pathway. Suppose that Host H1 on internet A needs to communicate with Host H2 on internet B. Then if a Switch SA of internet A can be connected to a Switch SB of internet B, the internet A can represent Host H2 as being homed to its Switch SA, via a Pathway (of whose

internal structure it is unaware) which is actually internet B. A corresponding mapping can be made in the other direction, permitting full-duplex communication. However, neither internet could use the other as an internal (i.e., Switch-Switch) Pathway, or the resulting configuration would be insufficiently responsive. (This may seem akin to the regionalization against which we argued in section 4.3.4. However, since neither internet uses the other as an internal Pathway, there are no problems of looping.) Naturally, just as Hosts on a common network can expect to get more efficient communications than can Hosts which must communicate over an internet, Hosts on a common internet will get more efficient communications than will hosts on different internets.

There are other reasons besides non-responsiveness which may make it imperative to have separate internets which cannot use each other as internal Pathways. For example, two internets might cover the same "territory," geographically speaking, but may be under the control of two different organizations, or may use essentially different algorithms or protocols. In fact, several different internets might even cover the same set of Hosts, and consist of the same set of component packet-switching networks. (It is important to remember that it is the set of gateways which constitute the internet, not the set of component networks. Imagine if every ARPA-controlled network had a Brand X gateway and a Brand Y gateway. Then there would be two separate

internets, Brand X and Brand Y, which are logically rather than physically separate.) Our procedure of having each internet regard the other as an Access Pathway to a set of Hosts, but not as an Internal Pathway, allows communication among Hosts on the different internets, without introducing problems of looping, and while preserving the maintainability of the individual internets. Of course if the two internets have different access protocols, then the Switches of one or the other internet (or both) must be prepared to translate from one protocol to the other, but that is a simpler problem than the ones we have been dealing with.

REFERENCES

1. J.M. McQuillan, I. Richer, E.C. Rosen, "The New Routing Algorithm for the ARPANET," IEEE TRANSACTIONS ON COMMUNICATIONS, May 1980.

2. E.C. Rosen, "The Updating Protocol of ARPANET's New Routing Algorithm," COMPUTER NETWORKS, February 1980.

3. J.M McQuillan, I. Richer, E.C. Rosen, ARPANET ROUTING ALGORITHM IMPROVEMENTS: FIRST SEMIANNUAL TECHNICAL REPORT, BBN Report No. 3803, April 1978.

4. J.M. McQuillan, I. Richer, E.C. Rosen, D.P. Bertsekas, ARPANET ROUTING ALGORITHM IMPROVEMENTS: SECOND SEMIANNUAL TECHNICAL REPORT, BBN Report No. 3940, October 1978.

5. E.C. Rosen, J.G. Herman, I. Richer, J.M. McQuillan, ARPANET ROUTING ALGORITHM IMPROVEMENTS: THIRD SEMIANNUAL TECHNICAL REPORT, BBN Report No. 4088, March 1979.

6. E.C. Rosen, J. Mayersohn, P.J. Sevcik, G.J. Williams, R. Attar, ARPANET ROUTING ALGORITHM IMPROVEMENTS: VOLUME 1, BBN Report No. 4473, August 1980.