IEN 187

ISSUES IN INTERNETTING

PART 2: ACCESSING THE INTERNET

Eric C. Rosen

Bolt Beranek and Newman Inc.

June 1981

Bolt Beranek and Newman Inc.
Eric C. Rosen

ISSUES IN INTERNETTING

PART 2: ACCESSING THE INTERNET

2. Accessing the Internet

This is the second in a series of papers, the first of which was IEN 184, that examine some of the issues in designing an internet. Familiarity with IEN 184 is presupposed. This particular paper will deal with the issues involved in the design of internet access protocols and software. The issue of addressing, however, is left until the next paper in this series. Part of our technique for exposing and organizing the issues will be to criticize (sometimes rather severely) the current protocols and procedures of the Catenet, even though we do not, at the present time, offer specific alternatives in all cases.

In IEN 184, section 1.4, we outlined four steps in the operation of a Network Structure. Let's now look closely at the first step, viz., how the source Host actually submits a message to the source Switch. In general, a Host will need to run three separate protocols to do this:

- a protocol to utilize the electrical interface between the Host and the initial component of the Pathway it uses to access the source Switch.

-a protocol to govern communication between the Host and the Pathway (PATHWAY ACCESS PROTOCOL).

-a protocol to govern communication between the Host and the source Switch (NETWORK ACCESS PROTOCOL).

We can make this point more concrete by giving some examples. Consider the case of an ARPANET host which wants to access the Catenet via the BBN gateway (which is also a Host on the ARPANET). Then the ARPANET is the Pathway the host uses to access the source Switch (the gateway). If the host is a local or distant host, the electrical interface to the Pathway is the 1822 hardware interface. If it is a VDH host, the electrical interface is whatever protocol governs the use of the pins on the modem connectors. If it were an X.25 host, the interface might be X.21. The PATHWAY ACCESS PROTOCOL is the 1822 protocol, which governs communication between the host and the first IMP on the Pathway. The NETWORK ACCESS PROTOCOL in this case would be the DoD standard Internet Protocol (IP), which governs communication between the host and the source Switch (gateway).

If, on the other hand, we consider the case of an ARPANET host which is communicating with another host on the ARPANET, and whose data stays purely within the ARPANET, 1822 becomes both the NETWORK ACCESS PROTOCOL (since the source Switch is now identical to the source IMP), and the PATHWAY ACCESS PROTOCOL, since the Pathway is now the 1822 hardware connection.

We will have nothing further to say abthe electrical interface, since that is really just a straightforward hardware matter. (However, such characteristics of the electrical interface as error rate, for example, might have to be reflected in the design of the Pathway Access Protocol.) The design of both the Pathway Access Protocol and the Network Access Protocol do raise a large number of interesting issues, and that shall be the focus of this paper.

We believe it to be very unlikely that Host software (or gateway software) can utilize the internet efficiently unless it takes the idiosyncrasies of BOTH the Pathway Access Protocol and the Network Access Protocol into account. A gateway or host software implementer who spends a great deal of time carefully building his IP module, but who then writes a "quick and dirty" 1822 module, is likely to find that his inefficient use of 1822 completely sabotages the advantages which his carefully designed IP is supposed to have. Experience with the ARPANET has shown many times that poorly constructed host software can create unnecessary performance problems. It seems, for example, that many 1822 modules completely ignore the flow control restrictions of the ARPANET, thereby significantly reducing the throughput that they can obtain over the ARPANET. We have even encountered many hosts which cannot properly handle some of the control messages of the 1822 protocol, which also leads to a very inefficient use of the ARPANET.

It is not difficult to understand why a host (or gateway) software implementer might overlook the issues having to do with the proper use of the Pathway Access Protocol. There are a number of pressures that, if not dealt with properly at a management level, lead naturally to the neglect of Pathway Access Protocol issues. An internet implementer might want to concentrate on the "new stuff", viz., the Network Access Protocol, IP, and may not be at all interested in the idiosyncrasies of the older Pathway Access Protocol (1822). He might be misled, by the belief that the packet-switching networks underlying the internet should be transparent to it, into believing that those packet-switching networks can be treated as simply as if they were circuits. He might also be under pressure to implement as quickly as possible the necessary functionality to allow internet access. While this sort of pressure is very common, the pressure to make the internet PERFORM well (as opposed to the pressure simply to make it work at all) is generally not felt until much (sometimes years) later. The tendency to neglect performance considerations while giving too much attention to simply obtaining the needed functionality in the quickest way is also reinforced by such "modern" design procedures as top-down design, and specification of protocols in formal languages. While these procedures do have a large number of advantages, they also serve to obscure performance issues. If the researchers and designers of protocols, following modern

design methodologies, do not give adequate consideration to performance at the time of protocol design, one can hardly expect the implementers to do so. Yet ARPANET experience has shown again and again that decisions made at the level of implementation, apparently too picayune to catch the attention of the designers, can be important determinants of performance. Still another reason why protocol software implementers might tend to disregard the niceties of the Pathway Access Protocol is the lack of any adequate protocol software certification procedure. An ARPANET host could be connected to an IMP for months, transferring large amounts of traffic, without ever receiving certain 1822 control messages. Then some sort of change in network conditions could suddenly cause it to receive that control message once per hour. There really is no way at present that the implementer could have possibly run tests to ensure that his software would continue to perform well under the new circumstances. This problem is somewhat orthogonal to our main interests, but deserves notice.

One of the most important reasons why protocol software implementers tend to ignore the details of the Pathway Access Protocols is the "philosophical" belief that anyone working on internet software really "ought not" to have to worry about the details of the underlying networks. We will not attempt to refute this view, any more than we would attempt to refute the view of a person who claimed that it "ought not" to rain on his

day off. We emphasized in IEN 184 that the characteristics of a Network Structure's Pathways are the main thing that distinguish one Network Structure from another, and that the problems of internetting really are just the problems of how to build a Network Structure with Pathways as ill-behaved as packet-switching networks. Thus building a successful internet would seem to be a matter of dealing specifically with the behavior of the various Pathways, rather than ignoring that behavior. We assume that that our task is to create an internet which is robust and which performs well, as opposed to one which "ought to" perform well but does not. It is true, as we have said, that within the Network Structure of the Catenet, we want to regard the ARPANET as a Pathway whose internal structure we do not have to deal with, but that does NOT mean that we should regard it as a circuit. Any internet Host or Switch (gateway), TO PERFORM WELL, will have to have a carefully designed and tuned Pathway Access Protocol module geared to the characteristics of the Pathway that it accesses.

The relationship between the Pathway Access Protocol and the Network Access Protocol does offer a number of interesting problems. For one thing, it appears that these protocols do not fit easily into the OSI Open Systems model. If we are accessing a single packet-switching network, the Network Access Protocol appears to be a level 3 protocol in the OSI model, and the Pathway Access Protocol appears to be a level 2 protocol.

However, if we are accessing an internet, we still need the level 3 Network Access Protocol, but now the Pathway Access Protocol also has a level 3 component, in addition to its level 2 component. So the Host is now running two different level 3 protocols, although the Network Access Protocol appears functionally to be in a layer "above" the level 3 part of the Pathway Access Protocol. Perhaps the main problem here is that the OSI model has insufficient generality to capture the structure of the protocols needed to access an internet like the Catenet.

It is interesting to see how some of these considerations generalize to the case of a Host which needs to access an internet (call it "B") through a Pathway which is itself an internet (call it "A"). Then the Host needs a Network Access Protocol for the internet B, a Network Access Protocol for the internet A (which is also its Pathway Access Protocol for internet B), and a Network Access Protocol for the actual network to which it is directly connected, which is also its Pathway Access Protocol for internet A. As we create more and more complicated Network Structures, with internets piled on top of internets, the Hosts will have a greater and greater protocol burden placed upon them. Ultimately, we might want to finesse this problem by removing most of this burden from the Hosts and putting it in the Switches, and giving the Switches knowledge of the hierarchical nature of the (internet) Network Structure. For

example, a Host on the ARPANET might just want to give its data to some IMP to which it is directly connected, without worrying at all about whether that data will need to leave the ARPANET and travel via an internet. The IMP could decide whether that is necessary, and if so, execute the appropriate protocol to get the data to some internet Switch at the next highest level of hierarchy. If the data cannot reach its destination within the internet at that level, but rather has to go up further in the hierarchy to another internet, the Switch at the lower level could make that decision and execute the appropriate protocol. With a protocol structure like this, we could have an arbitrarily nested internet, and the Switches at a particular level, as well as the Hosts (which are at the lowest level), would only have to know how to access the levels of hierarchy which are immediately above and/or below them. This procedure would also make the host software conform more to the OSI model, since only one Network Access Protocol would be required. However, this sort of protocol structure, convenient as it might be for the Hosts, does not eliminate any of the issues about how to most efficiently use the Pathways of a Network Structure. Rather, it just pushes those issues up one level, and makes the Switches correspondingly more complicated. A proper understanding of the issues, therefore, is independent of what sort of protocol structuring we design.

Having emphasized the need for hosts and gateways to take account of the details of particular Pathway Access Protocols, we must point out that this is not always a simple thing to do. If the Network Structure underlying a Pathway is just a single network, like the ARPANET, this problem is not so terribly difficult, since one can expect that there will be available a lot of experience and information about what a host should do to access that network efficiently. If, on the other hand, the Pathway is really an internet itself, the problem is more difficult, since it is much more difficult to say anything substantive about its characteristics. This is a point we must keep in mind as we discuss specific issues in access protocol design.

In the remainder of this paper, we will attempt to deal with a number of issues involved in the design of robust, high-performance Network and Pathway Access Protocols. We will not attempt to cover every possible issue here. In particular, the issue of how to do addressing is important enough to warrant a paper of its own, and shall be put off until the next paper in this series. We will attempt throughout to focus on issues which particularly affect the reliability of the internet configuration (as perceived by the users), and on issues which affect the performance of the internet (as perceived by the users). Wherever possible, we will try to exhibit the way in which the reliability and performance of a protocol trade off against its

functionality. If protocol designers concentrate too heavily on questions of what functionality is desired, as opposed to what functionality can be provided at a reasonable level of performance and reliability, they are likely to find out too late that the protocol gives neither reasonable performance nor reliability.

2.1 Pathway Up/Down Considerations

In general, a Host will be multi-homed to some number of Switches. In fact, it is easy to imagine a Host which is both (a) multi-homed to a number of IMPs, within the Network Structure of the ARPANET (this cannot be done at present, but is planned for the future), and also (b) multi-homed to a number of gateways (namely, all the gateways on the ARPANET) within the Network Structure of the Catenet. Whenever a Host is multi-homed to a number of Switches in some Network Structure, it has a decision to make, namely, which of those Switches to use as the source Switch for some particular data traffic. In order to make this choice, the very first step a Host will have to take is to determine which Switches it can reach through operational Pathways. One thing we can say for sure is that if a Host cannot reach a particular Switch through any of its possible Pathways, then it ought not to pick that Switch as the source Switch to which to send its data. In a case, for example, where the ARPANET is partitioned, a Host on the ARPANET which needs to send

internet traffic will want to know which gateways it can reach through which of its ARPANET interfaces. To make this determination possible, there must be some sort of "Pathway Up/Down Protocol", by which the Host determines which of its potential Pathways to gateways are up and which are down. This is not to say, of course, that the Hosts have to know which gateways are up and which are down, but rather, they must know which gateways they can and cannot reach. Of course, this situation is quite symmetric. The Switches of a Network Structure (and in particular, the gateways of an internet) must be able to determine whether or not they can reach some particular host at some particular time. Otherwise, the gateway might send traffic for a certain Host over a network access line through which there is no path to that Host, thereby causing unnecessary data loss. Apparently, this problem has occurred with some frequency in the Catenet; it seems worthwhile to give it some systematic consideration.

The design of reliable Pathway Up/down protocols seems like something that "ought to be" trivial, but in fact can be quite difficult. Let's begin by considering the case of an ARPANET host which simply wants to determine whether it can reach some IMP to which it is directly connected. The first step for the host to take (if it is a local or distant host) is to look at the status of its Ready Line. If the Ready Line to some IMP is not up, then it is certain that communication with that IMP is not

possible. If the host is a VDH host, then there is a special up/down protocol that the host must participate in with the IMP, and if that fails, the host knows that it cannot communicate with the IMP. Of course, these situations are symmetric, in that the IMP has the same need to know whether it can communicate with a host, and must follow the same procedures to determine whether this is the case. However, even in these very simple cases, problems are possible. For example, someone may decide to interface a host to an IMP via a "clever" front-end which hides the status of the Ready Line from the host software. If a host is multi-homed, and has to choose one from among several possible source IMPs, but cannot "see" the Ready Lines, what would stop it from sending messages to a dead IMP? Eventually, of course, a user would notice that his data is not getting through, and would probably call up the ARPANET Network Control Center to complain about the unreliability of the network, which, from his perspective, is mysteriously dropping packets. From the opposite perspective, one must realize that such a front-end might also hide the status of the host from the IMP, so that the network has no way of knowing whether a particular host is currently capable of communicating with the network. This is especially likely to happen if the "clever" front-end takes packets from the network which are destined for a particular host, and then just drops them if the host is down, with no feedback to either IMP or host. If a host is multi-homed, but one of its access lines is down,

this sort of configuration might make it quite difficult for the network to reach a reasonable decision as to which access line to use when sending data to that host. The lesson, of course, is that the status of the Ready Line should never be hidden from the host software, but it is hard to communicate this lesson to the designers of host software. Again, the issue is one of performance vs. functionality. A scheme which hides the status of the Ready Line from IMP or host may still have the required (minimum) functionality, but it will just perform poorly under certain conditions.

This may seem like a made-up problem which probably would never occur, but in fact it has occurred. We once had a series of complaints from a user who claimed that at certain times of certain days he had been unable to transmit data successfully over the ARPANET. Upon investigation, we found that during those times, the user's local IMP had been powered down, due apparently to a series of local power failures at the user's site. Of course, the IMP will not transmit data when it is powered down. But it was somewhat mysterious why we had to inform someone of a power failure at his own site; surely the host software could have detected that the IMP was down simply by checking the Ready Line, and so informed the users. When this user investigated his own host software (a very old NCP), he found that it would inform the users that the IMP was down ONLY if the IMP sent the host a message saying that it was going down. Since the IMP does not

send a message saying that it is about to lose power, the host software, which apparently did not check the Ready Line as a matter of course, did not detect the outage. It looked to the user, therefore, as though the network had some mysterious and unreliable way of dropping packets on the floor. It seems that many hosts presently exist whose networking software is based on the assumption that the IMPs never go down without warning. Hosts do sometimes have difficulty determining whether their Pathway to an IMP is up or down, even when it seems like this should be totally trivial to determine. Reliable network service requires, however, that host software and hardware designers do not hide the status of the IMP from the host, or the status of the host from the IMP. This will become increasingly important as more and more hosts become multi-homed.

Of course, this is only a first step in a proper up/down determination. It is not impossible for a Ready Line to be up but for some problem either in IMP or host to prevent communications from taking place. So some higher level up/down protocol is also necessary. Some protocol should be defined by which Host and Switch can send traffic to each other, and require the other to respond within a certain time period. A series of failures to respond would indicate that proper communications is not possible, at least for the time being. It is important to note, though, that the need for a higher level up/down protocol does not obviate the need for the lower level procedure of

monitoring the Ready Line. If the higher level procedure fails, but the Ready Line appears to be up, knowledge of both facts is needed for proper fault isolation and maintenance. Also important to notice is that if the lower level procedure indicates that the Pathway is down, the higher level procedure should not be run. This might not seem important at first glance, but in practice, it often turns out that attempting to send traffic to a non-responsive machine results in significant waste of resources that could be used for something more useful.

In the more general case, where a Host's Pathway to a source Switch may include one or more packet-switching networks, it is far from trivial to determine whether the Switch can be reached from the Host via the Pathway. Consider, for example, how a given ARPANET host could determine whether a given Catenet gateway on the ARPANET can be accessed via some given ARPANET source IMP. Of course, the first step is to determine whether communication with that source IMP is possible. Even if it is, however, the gateway might still be unreachable, since it may be down, or the network may be partitioned. ("Officially", every ARPANET Host is supposed to be reachable from any other ARPANET Host. However, the average connectivity of the ARPANET is only 2.5, which means that only a small number of line or node failures are needed to induce partitions. Furthermore, a few ARPANET sites are actually stubs, which means that a single failure can isolate them from the rest of the ARPANET. As often

seems to happen in practice, the sites that are stubs seem to be attached by the least reliable lines, so that partitions are not infrequent. At any rate, there will probably be networks in the internet that partition more frequently than the ARPANET does. Internet protocols must detect and react to network partitions, instead of simply disregarding them as "too unlikely to worry about.")

In the special case where the Pathway between some Host and some Switch is the ARPANET, the ARPANET itself can provide information to the Host telling it whether the Switch is reachable. If the Switch is not reachable, and a Host attempts to send an ordinary data packet to it, the ARPANET will inform the Host whether or not that packet was delivered, and if not, why not. Unfortunately, the current ARPANET does not provide this information in response to datagrams. However, we have already seen the need to provide such information in the case of logically addressed datagrams (see IEN 183), and plan to implement a scheme for doing so. An ARPANET Host which is also an internet Host can implement a low level Pathway up/down protocol simply by paying attention to the 1822 replies that it receives from the ARPANET. There are hosts which seem to disregard these 1822 control messages, and which seem to continue to send messages for unreachable hosts into the ARPANET. Of course, this is a senseless waste of resources which can severely degrade performance. Indeed, it may look to an end-user, or even

a gateway implementer, as though the ARPANET is throwing away packets for no reason, when the real problem is that the host software cannot respond adequately to exceptional conditions reported to it by the network.

We have spoken of the need for Host and Switch to run a higher level up/down protocol, to take account of the conditions when one of them seems reachable to the network, but still will not perform adequately when another entity attempts to communicate with it. Switch and Host must run some protocol together which enables each to validate the proper performance of the other. The Catenet Monitoring and Control System (CMCC), currently running on ISIE, runs a protocol of this sort with the gateways. The CMCC sends a special datagram every minute to each gateway, and expects to receive an acknowledgment (or echo) for this special datagram back from the gateway. After three consecutive minutes of not receiving the echo, the CMCC infers that the gateway cannot be reached. After receiving a single echo, the CMCC infers that the gateway can be reached. (Gateways run a similar protocol with their "neighboring gateways".) A Pathway up/down protocol which does not rely on the intervening network to furnish the information would certainly have to involve some such exchange of packets between the Host and the Switch, but it would have to be rather more complex than this one. One of the problems with this protocol is that it is incapable of detecting outages of less than three minutes. This

may be suitable for the CMCC's purposes, but is not generally suitable for a Host which wants to know which source Switch to send its traffic to. We would not want some Host to spend three full minutes sending data to a Switch which cannot be reached; the effect of that could be many thousands of bits of data down the drain. (Of course, higher level protocols like TCP would probably recover the lost data eventually through the use of Host-Host retransmissions, but that involves both a severe drain on the resources of the Host, which ought to be avoided whenever possible, and a severe degradation in delay and throughput.) Another problem with this particular protocol is that it uses datagrams, which are inherently unreliable, and as a result, the inference drawn by the CMCC is unreliable. From the fact that three datagrams fail to get through, it is quite a big jump to infer that no traffic at all can get through. Another problem is the periodicity of the test packets. If they get in phase with something else which may be going on in the network, spurious results may be produced.

The design of a Pathway up/down protocol must also be sensitive to the fact that some component network of a Pathway may be passing only certain types of packets and not others. For example, at times of heavy usage, certain networks may only be able to handle packets of high priority, and lower priority packets may either be refused by that net (at its access point), or, even worse, discarded internally by the net with no feedback.

The Pathway up/down protocol must be sensitive to this, and will have to indicate that the Pathway is only "up" to certain classes of traffic. If a Pathway is really a Network Structure which will inform its Hosts when it cannot accept certain traffic types, then this information can be fed back into the up/down protocol. (Note however that this might be very difficult to do if the Pathway consists of not a single network, but of an internet). Alternatively, a Host may have to rely on its higher level Pathway up/down protocol to determine, for several classes of traffic, whether the Pathway is up to members of that class. Apart from the inherent difficulty of doing this, it may be difficult to map the traffic classes that a given component network distinguishes into traffic classes that are meaningful to a Host, or even to the Switches of the internet. Yet we wouldn't want traffic to be sent into a network which is not accepting that particular kind of traffic, especially if there are alternative Pathways which would be willing to accept that traffic.

Many of these considerations suggest that the higher level up/down protocols could turn out to be rather intricate and expensive. Remember that a gateway may have many many hosts "homed" to it, and must be able to determine, for each and every one of these hosts, whether communication with it is possible. Yet it probably is not feasible to suppose that each gateway can be continuously running an up/down protocol with each potential

host, and still have time left to handle its ordinary traffic. This suggests that the primary up/down determination be made from the low-level protocol, i.e., that the Switches should rely on the networks underlying the Pathways to inform them whether a given Host is up or down, and the Hosts should similarly rely on the networks underlying the Pathways to pass them status information about the gateways. It would be best if the higher level up/down protocol only needed to be run intermittently, as a check on the reliability of the lower level protocol. Unfortunately, the use of low level up/down protocols is not always possible. Many networks, unlike the ARPANET, do not even gather any information about the status of their hosts, and hence cannot inform a source Host that it is attempting to send data to a destination Host which is not reachable. (SATNET is an example of a network that does not pass "destination dead" information.) In the case where a particular Host-Switch Pathway is itself an internet, the problem is even worse. Unless the component networks of that internet can be made to cooperate in obtaining RELIABLE up/down information and passing it back to the source Host, it will be very hard for a Host to make any reasonable determination as to whether a particular Switch is reachable. We would strongly recommend the incorporation of low level up/down protocols in ALL component networks of the internet.

There is another important problem in having a Host determine which of its potential source Switches on the internet

are up and which are down. In order to run a protocol with the Switch, or even to query the lower level network about the Switch, the Host must have some way of identifying the Switch. It is not so difficult for a Host on the ARPANET to identify the IMPs that it is directly connected to, since it is quite simple to devise a protocol by which a Host can send a message down each of its access lines, asking who is on the other end. It is rather more difficult for a Host to find out which gateways it is homed to (i.e., which gateways are on a common network with it). There is no easy way for an ARPANET Host to find out which other ARPANET hosts are Catenet gateways. There is no "direct connection" at which to direct protocol messages. In the current Catenet, hosts have to know in advance how to identify the Catenet gateways on their networks (although there are certain restricted circumstances under which a host can obtain the name of another gateway from a gateway about which it already knows). Yet it does not seem like a good idea to require a Host to know, a priori, which other Hosts on its network are also internet Switches. This makes it difficult to enable Hosts to take advantage of newly installed gateways, without making changes by hand to tables in the Hosts (a procedure which could require weeks to take effect). There is a rather attractive solution to this problem. If each component network in the internet can determine for itself which of its Hosts are also internet Switches (gateways), then the Switches of that network can

provide that information to the Hosts. This would require the existence of a protocol which the gateways run with the Switches of the individual component networks, by means of which the gateways declare themselves to be gateways. Each individual network would also have to have some internal protocol for disseminating this information to other Hosts, and for keeping this information up to date. If the network allows GROUP ADDRESSING, further advantages are possible. The network could maintain a group address (called, say, "Catenet Gateways") which varies dynamically as gateways enter and leave the network. Hosts could find out which gateways are reachable over particular network access lines by sending some sort of protocol message to the group address, and waiting to see who replies. Hosts would thet have to have any ori knowledge of the gateways on their home networks.

One very important though often neglected aspect of up/down protocolshe way in which the up/down protocol interacts with the ability to perform adequate maintenance of the Network Sure. It is teg to think that a Pathway up/down protocol ought to declare a Pathway "down" only if it is totally dead or otherwise totally unusable. But in fact, a pathway should be declared down before it becomes totally dead, if its packet "non-delivery rate" exceeds a certain threshold. (We use the term "non-delivery rate" where the term "error rate" is more commonly used. We are trying to emphasize that it is important

to detect not only errors, in the sense of checksum errors, but rather any circumstances, including but not limited to checksum errors, which prevent the proper delivery of packets.) There are two reasons for this:

1) The existence of a non-zero non-delivery rate on a Pathway implies that some packets placed on that Pathway will not make it through to the other end. In most applications, these packets will have to be retransmitted at some higher level of protocol, or else by the end user himself (packetized speech is one of the few exceptions to this). As the number of retransmissions increases, the delay also increases, and the throughput decreases. So when the non-delivery rate reaches a certain point, the Pathway should be removed from service, in order to improve delay and throughput. Of course, this assumes that an alternate Pathway is available with a lower non-delivery rate. Also, other things being equal, removing bandwidth from a Network Structure will also tend to increase delay and reduce throughput, so we really want the up/down protocol to pick out the proper cross-over point.

2) It is often better to fix a Pathway at the first sign of trouble than to wait for it to fail totally. One implication of this is that the up/down protocol should

perform equally well whether or not the Pathway is heavily loaded with traffic. We would not want to use a protocol which made its determination solely by making measurements of user traffic, since that protocol would not function well during periods when user traffic is very light. That is, a faulty Pathway with no user traffic would not be detected. Yet if repair work has to be done on a Pathway, we would most like to find out about it during lightly loaded hours, so that a fix can be effected with minimal disruption, possibly before the heavily loaded hours begin.

Another important characteristic for a Pathway up/down protocol to have is the ability to determine the nature of the Pathway "outage". This is quite important for fault isolation, but is easy for a host software person to overlook, since he may not be aware of such issues. If a cannot get its packets to a Switch over a certain Pathway, it will want to regard that Pathway as down, and will want to use an alternate Pathway. From the Host perspective, it doesn't care whether the reason it can't use the Pathway is because of a network partition, or because of network congestion, or because of some other reason. However, if the Host personnel want to be able to call up the Pathway personnel and request that the problem be fixed, it's not enough to say, "Your network isn't working; call me back when it's fixed." The more information the Pathway up/down protocol can

gather, the quicker a fix can be effected. In the case where the Pathway is the ARPANET, quite a bit of information can be gathered from proper instrumentation of the 1822 module, and proper attention by the host software to the 1822 replies; this will be discussed further in section 2.6.

The design of the ARPANET's line up/down protocol might be a good model for the design of a general Pathway up/down protocol. The design of the ARPANET protocol was based upon a mathematical analysis of the probabilistic error characteristics of telephone circuits, and the protocol is intended to bring a line down when and only when its error rate exceeds a threshold. However, the error characteristics of Pathways in general (i.e., of packet-switching networks) are not well understood at all, and there is no similar mathematical analysis that we can appeal to. At present, we can offer no ready answer to the question of how a Host can tell which of several possible source Switches is reachable, if the Switches are accessed via a network (or sequence of networks) which will not even inform the Host whether or not its traffic even gets delivered. This is an important question which will require further thought, and considerable experimentation.

2.2 Choosing a Source Switch

Once a Host has determined which source Switches it can reach over which of its interfaces, it still has to determine which one to use for sending some particular packet (unless the Host is "lucky" enough to find out that only one source Switch is reachable). Making the proper choice can be quite important, since the performance which the Host gets may vary greatly depending upon which source Switch it selects. That is, some source Switch might be much closer to the destination, in terms of delay, than another. It then might be quite important to choose the proper one. To make things a bit more concrete, consider the case of a Host which is multi-homed (via several distinct 1822 links) to several ARPANET IMPs, and whose traffic can be handled entirely within the ARPANET. There are several things a host might want to take into account in choosing the best source IMP to use for a particular packet, including:

1) The loading on the 1822 access line to each possible source IMP.

2) The distance between each source IMP and the destination Host, for some notion of "distance."

The first of these two quantities is relatively easy to obtain, since all the Host need do is monitor its own 1822 lines; it should be possible to devise a monitoring scheme which

indicates which of the 1822 lines is providing the best service to its IMP, perhaps simply by measuring the queuing delay experienced in the Host by messages queued for that line. (Any such measurement would have to take into account some of the niceties of the 1822 protocol, though.) Obtaining information about the second quantity is more difficult. The Host might try to keep some measurement of round-trip delay (delay until a RFNM is received) between itself and each destination Host. However, in order to do this, some traffic for each destination Host would have to be sent over each access line, so that the delay could be measured. This means that some traffic has to be sent over a long delay path, simply in order to determine that that is a long delay path. A simpler scheme might be for the Host to get delay information from the IMP. A Host could ask each potential source IMP what its delay to the destination Host is. By using this information, plus the information it gathers locally about the loading of its access lines, the Host could determine which source IMP provides the shortest path to the destination.

This would require that we define a protocol by which a Host can ask the IMPs to which it is homed to provide their delays to a destination Host. The Host could make these requests periodically, and then change its selection of source IMPs as required in order to react to changes in delay. There are a few subtle protocol issues to be considered here, though. We would have to make sure that a Host cannot beat a Switch to death by

constantly asking it what its delays are; probably we would have to give the Switch the option of not replying to these requests if it is too busy with other things (like ordinary data traffic). A bigger problem lies in the assumption that the Switches will even have this data to provide. The routing algorithm used by the ARPANET IMPs does, in fact, provide each IMP with a value of delay, in milliseconds, to each other IMP in the network. There is no reason why this information could not be fed back to the hosts on request. Note, however, that while a source IMP knows its delay to each possible destination IMP, it does not know its delay to each potential destination HOST over each possible access line to that Host, since the routing algorithm does not maintain measurements of delay from an IMP to a locally attached host. Yet this latter delay might be quite significant. Still, the information that the ARPANET IMPs could provide to the Hosts should enable them to make a better choice than they could make without this information.

Another problem with this idea of having the Switches feed back delay information to the Hosts is the proper choice of units. If a Host is going to take the delay information provided by the network and then add some locally measured delay information to it, it is important for the Host to know what units the network is using to measure delay. Yet we also have to ensure that the network developers and maintainers are free to change the way in which the network does measurements, and the

units in which the measurements are taken, WITHOUT NEEDING TO COORDINATE SUCH CHANGES WITH ALL HOST ADMINISTRATIONS. That is, we don't want further development of the network, and further refinements in the way network measurements are done, to be overly constrained by the fact that the Hosts demand measurements in a certain unit. We also want to ensure that host software implementations are not invalidated by a decision to change the units that the network uses for its internal measurements. So the protocol would have to enable the Switch to tell the Host what units it is providing; the Host would then make any necessary conversions. (Alternatively, the Host could tell the Switch what units it wants, and the Switch could do the conversion before sending the information to the Host.)

In the internet environment, the situation is more complicated. An ARPANET Host which is also an internet Host would have to (a) figure out its delay to each of its source IMPs, (b) query each source IMP for its delay to each source gateway, and (c) query each source gateway about its delay to each destination. There is no straightforward way to gather the rest of the needed delay information, however, namely the delay from the destination gateway to the destination Host. In more complex Network Structures, with internets nested on top of internets, this problem becomes increasingly more complex. It seems that the only really reliable way, and the most straightforward way, for the source Host to gather information

about the delays via various source Switches to a destination Host, is for it to do the measurements itself. This is the recommended solution. Delay information should also be made available from the component networks for Hosts which cannot do this, but it should be understood that those hosts cannot expect to get as good a quality of service as the hosts which go to more trouble to do their own measurements.

2.3 Type of Service

One very important piece of information that a Host must specify to the source Switch through the Network Access Protocol is the "type of service" desired. To quote from the DoD standard Internet Protocol (IP) specification [1, p. 15], "The Type of Service is used to indicate the quality of the service desired: this may be thought of as selecting among Interactive, Bulk, or Real Time, for example." This seems to make sense, since one does have the feeling that different types of applications will fall into different categories, and information about the categories may help the Switches of the Network Structure through which the data is moving decide how best to treat it. However, choosing just the right set of categories of service is quite a complex matter. For example, both a terminal user of a time-sharing system, and a user of a query-response system (like an automated teller) fall under the rubric of "interactive", but that doesn't mean that the service requirements are the same.

Both Remote-Job-Entry and File Transfer fall under the rubric of "bulk", but it is not obvious that they have the same requirements. Both real-time process control and packetized voice fall into the category of "Real Time", but the requirements of these two applications seem to be very different. A very real issue, which has not yet been given adequate consideration, is the question of just how many categories of application type there really should be, and just what the implications of putting a packet into one of these categories ought to be. As we go on, we will see a number of problems that arise from failure to carefully consider this issue.

It is rather difficult to find examples of Network Access Protocols which have really useful class-of-service selection mechanisms. The 1822 protocol allows the user to select from among two priorities: it allows the choice of single-packet or multi-packet messages: it allows the choice between "raw packets" and "controlled packets." It is up to some user (or more realistically, up to some host software implementer who may have only a vague and limited understanding of the applications which his software will serve, and of the network that he is accessing) to map his application characteristics onto these three choices. Unfortunately, it is doubtful that there is anyone outside of the ARPANET group at BBN with any clear understanding of the implications of making the various choices. The task of making the optimum choice for some application is further complicated by

the fact that the effects of making the various choices can be very dependent on the network load. For example, it is often possible to get more throughput from single-packet messages than from multi-packet messages. This will happen if the destination IMP has several different source Hosts sending multi-packet messages to it, but is short on buffer space (as many of the ARPANET IMPs are), and if the multi-packet messages contain only two or three packets per message. Not only is this sort of thing very difficult for an arbitrary user to understand (to a naive network user, it must seem ridiculous), it is also subject to change without notice. Although users can vary their service significantly by sending optimum size messages, the principles governing the "optimum" size are very obscure, and we cannot really expect users to map their application requirements onto this network feature in any reasonable manner.

A similar problem arises with respect to the priority bit that the 1822 protocol allows. Basically, a priority packet will get queued ahead of any non-priority packets on the queues for the inter-IMP links and on the queues for the IMP-Host access lines. However, priority packets receive no special preference when competing with non-priority packets for CPU cycles or for buffer space. Also, there is no notion at all in the ARPANET of refusing to accept low priority packets because the network is already too heavily loaded with high priority packets. Although someone who has carefully studied the ARPANET might be able to

say what the effect of setting the priority bit is under some particular set of circumstances, some user who is wondering whether his application requirements are best served by setting the priority bit really has no way of answering that question. The actual effect of the priority bit does not fully correspond to any intuitive notion of priority that an arbitrary user is likely to have. Another problem: although it is presently allowed, it is not really a good idea to let the users choose whether to set the priority bit or not. Fortunately, most hosts do not submit packets with the priority bit on. It wouldn't be terribly surprising, though, if some host software implementer decided that he would always set the priority bit, in order to get faster service. Of course, overuse of the priority bit just means that it will have no effect at all, and that seems to mean that its use must be controlled in some way, and not simply left up to each user, as in the 1822 protocol.

The IP offers even worse problems than 1822 in these respects. Like 1822, the IP does not really allow the user to classify his traffic according to application type. Rather, it forces him to pick one of 5 possible precedence values (from highest to lowest precedence, whatever that means, exactly), to pick one of 4 reliability values (from most to least reliable), to indicate whether he wants his data to be stream data or datagram data in component networks for which this distinction is meaningful, to indicate whether he wants high or low speed, and

to indicate whether speed is more important to him than reliability is. The idea here, apparently, is that any user can map his application requirements into certain abstract properties, and the information which the IP passes from the Host to the source Switch is supposed to indicate which of these abstract properties the user needs. At each internet hop, these abstract properties are supposed to be mapped to particular properties that are meaningful to the network in question. The Pathway Access Protocol for that network would then be used to indicate to the Switches of that component network what particular properties the data transfer should have within that network. In fact, the only apparent use of the "type of service" information in the internet Network Access Protocol (IP) is to carry information to be passed to the individual Pathway Access Protocols.

This all sounds reasonable enough when considered in the abstract, but it gives rise to a large number of vexing problems when we attempt to consider particular ways in which this "type of service" information is to be used. Empirically, it seems that few current gateway implementations take any notice of this information at all. We suggest that the problem is not that the individual implementers have not had time to write the code to take account of this information, but rather that it is far from clear how this information should be handled, or even that this information is really meaningful. We suggest further that an

internet user would also have a great deal of difficulty deciding how to specify the "type of service" information in order to get a specific quality of service needed by his application.

Suppose a user needs the maximum possible speed for his application, so he uses IP to indicate that he values speed above all else. What would the current Catenet do? For concreteness, suppose there is a choice of sending this user's data either via a sequence of 4 low-delay terrestrial networks, or through three satellite networks, each of which contains two satellite hops. The current implementation of the Catenet would send the data through the three satellite networks. However, since the user indicated that he values speed above all else, he will get the fastest service that each of the satellite networks can provide! Of course, this may not be what the user will have expected when he asked for speed, since the fastest service through a satellite network is not fast. A user may well wonder what the point of specifying speed is, if his data is going to traverse some sequence of satellite networks, even if a much faster path is available. Furthermore, it is not correct to assume, in general, that a user who values speed will really want the speediest service through every network. If traffic must go through a satellite network, it may be important to try to get one-hop rather than two-hop delay, if this is possible. Yet it may not be economical to also try to get the speediest service through all terrestrial networks; the difference between high and low

speed service through a terrestrial network might be "in the noise", even when compared to the shortest delay through the satellite network. It is not impossible, or even unlikely, that better overall service (or more cost-effective service) can be achieved by using the fastest possible service through some networks, but less than the fastest through others. There are two immediate lessons here. First, the characteristics that a user specifies in the Network Access Protocol may require some interaction with routing, since the characteristics he desires simply cannot be provided, in general, by sending his traffic through a random series of networks, and then mapping information he specifies in the Network Access Protocol into information carried in the individual Pathway Access Protocols. Second, what a user means intuitively by "speed" just may not map into what some particular component net means by "speed". Once again, we see that the basic problem stems from the differing characteristics of the Pathways in the Network Structure.

Another peculiar feature of the IP is the mysterious "S/R bit", which a user is supposed to set to indicate whether he prefers speed over reliability, or vice versa, should these conflict. One unsuitable aspect of this is the apparent assumption that it even makes sense to prefer either speed or reliability over the other, without specifying more detail. It is easy to imagine that some user is willing to accept reliability of less than 100% if he can increase his speed

somewhat. It is also easy to imagine that a user would be willing to accept somewhat slower service if it gives him higher reliability. But there will always be a range that the user wants to stay within. If his reliability must be moved below a certain threshold in order to get more speed, he may not want this, even if he would be willing to say that he prefers speed to reliability. Similarly, if his delay must go above a certain threshold to gain more reliability, he may not want this, even if, when talking in general terms, he says that he needs reliability more than speed. It really doesn't make any sense at all to try to map a particular application type into "speed over reliability" or "reliability over speed", unless ranges and thresholds are also specified. What this means in practice is that a user will not be able to make a reasonable choice of how to set this bit in the IP header; whatever he sets it to is bound to produce results other than those he expects under some not too uncommon set of circumstances.

We do not want to leave unquestioned the tacit assumption that speed and reliability are opposing virtues, so that increasing one must be expected to decrease the other. To quote again from the IP spec, "typically networks invoke more complex (and delay producing) mechanisms as the need for reliability increases" [1, p 23]. This reasoning is somewhat superficial. It may be true that in some networks, the less reliable kinds of service are speedier, but this is not invariably the case. To

see this, consider the following (fictitious) network. This network allows the user to request either "reliable" or "unreliable" data transfer. Reliable packets are controlled by a set of protocols, both at the end-end and hop-hop level, which ensure delivery. Unreliable packets are not under the control of any such protocols. Furthermore, reliable packets go ahead of unreliable ones on all queues, in particular, the CPU queue. In addition, unreliable packets can be flushed from the net at any time, if some resource they are using (such as buffer space) is needed for a reliable packet. These latter two measures are needed to ensure that the net does not become so heavily loaded with unreliable packets that there is no room for the reliable ones. (It would not make much sense to advertise a "reliable" service, and then to allow the unreliable packets to dominate the network by using most of the network resources. If unreliable packets could grab most of the resources, leaving the "reliable" ones to scavenge for the left-over resources, then it would be virtually inevitable that the service received by the "unreliable" packets would appear, to the users, to be more reliable than the service received by the "reliable" packets. To achieve a true dichotomy between reliable and unreliable service, the reliable packets must be given priority in all respects over the unreliable ones. We should also remember, by the way, that although many protocols combine features of reliability, sequentiality, error control, and flow control, these are not the

same, and there is no reason why a network might not offer a reliable but unsequenced service). This sort of network design seems quite reasonable, perhaps more reasonable than the design of any existing network. It would allow for a (presumably inexpensive) class of service ("unreliable") which would be able to use only those network resources not needed by the more reliable (and expensive) class of packets, and which would not suffer any additional delay due to the presence of the protocols which would be needed to ensure reliability. In such a network, unreliable packets might well experience less delay than reliable ones, WHEN THE NETWORK IS LIGHTLY LOADED; WHEN IT IS HEAVILY LOADED, HOWEVER, RELIABLE PACKETS WOULD TEND TO EXPERIENCE THE SMALLER DELAY. If this is the case, it is hard to see how a user could be expected to make a reasonable choice of IP service parameters at all. He may know what his needs are, but we can hardly expect him to know how to map his needs onto particular aspects of the behavior of a particular network component of an internet, especially when the behavior determined by that mapping will vary dynamically with the network loading, and hence with the time of day.

Two other peculiarities of the "type of service" feature of the IP are worth mentioning. First, there seems to be no notion of the relation between speed and priority, though in many networks, the priority of a message is the major determinant of its speed. (There are, to be sure, networks which attempt to

treat priority solely as "acceptance class", differentiating it completely from considerations of speed. However, we know of no network implementation which has been shown to differentiate SUCCESSFULLY between these two concepts, and there is reason to doubt that this differentiation is even possible in principle.) Second, one of the choices to be made is whether to prefer stream or datagram service. This is a clear example of something that is not based on "abstract parameters of quality of service", but rather on a particular feature of one or two particular networks. Requesting stream service will NOT do what a user might expect it to do, namely set up a stream or virtual circuit through the entire internet. This would require a lengthy connection set-up procedure, involving reservations of resources in the gateways, which resources are to be used only for specific connections. If we are really serious about providing stream service, this is just as important as obtaining stream service within the component networks serving as the Pathways of the internet. Indeed, it is hard to imagine any real use for an internet "stream service" which treats packets as datagrams during most of their lifetime in the internet, and then treats them as stream packets in one or two component networks. It must be remembered that the sort of stream service provided by a network like SATNET is only useful to a user if his data appears at the SATNET interface at fixed periods, synchronized with the scheduling of the stream slots on the satellite channel. If the data must

first travel through several datagram networks before reaching SATNET. IT IS VIRTUALLY IMPOSSIBLE THAT THE DATA WILL ARRIVE AT SATNET WITH THE PROPER PERIODICITY to allow it to make proper use of the SATNET stream. Now there are certain specific cases where it might be possible to provide some sort of stream service, say if some data is going from a local network through SATNET to another local network and thence directly to the destination Host. (Though even in this case, some sort of connection set-up and reservation of resources in the gateways between SATNET and the local networks would probably be necessary.) Note, however, that if a user requests this type of service, he is also constraining the types of routes his data can travel. If SATNET is not available, he might not want to use the internet at all at that time. Or he might be willing to tolerate a less optimal route ("half a loaf is better than none"), but might not want "stream service" if the less optimal route has to be used. In no case can a type of service like "stream" be obtained simply through the mapping of "type of service" in the internet onto "type of service" in the component networks.

We do not want to have a Network Access Protocol that will need to be infinitely expandable, so that the user can indicate the type of service he wants in each particular network that his data may eventually travel through. For one thing, as the internet becomes larger, so that there are more paths between each possible source and destination, the users will not

generally know what set of networks their data will travel through. Since the number of component networks in the internet may be continually increasing, and since we cannot anticipate in advance the features that each new network may offer, it does not really seem reasonable to have to keep adding fields to the IP, to account for particular characteristics of each new component network. Yet this seems inevitable with the current approach. That is, we do not agree with the claim in the IP spec that the type of service field in the IP indicates "abstract parameters". Rather, we think the type of service field has been constructed with certain particular networks in mind, just those networks which are currently in the Catenet, and that the various service fields have no meaning whatsoever apart from the particular "suggested" mappings to protocol features of specific networks given in the spec. (And since these mappings are only "suggested", not required, one might wonder whether the type of service field really has any consistent meaning at all). This situation is perhaps tolerable in a research environment, where most of the users of the internet are explicitly concerned with issues of networking, and willing to try a large number of experiments to see what sort of service they get. One must remember, however, that in a truly operational environment, the average user will not be concerned at all about networking, will not know anything about networking, will not care about networking, and will only want the network to appear transparent

to him. In order for such a user to make successful use of the type of service field in a Network Access Protocol, the parameters of the field must be meaningful to him. If they are only meaningful to network experts, the user will never be able to figure out how best to set these parameters.

Rather than providing a type of service specification which is nothing but a sort of "linear combination" of the types of service provided by the component networks, the internet ought to offer a small, specific number of service types which are meaningful at the application level. . The possible values of internet service type might be "interactive session," "transaction," "file transfer", "packetized speech," and perhaps a few others. The categories should be simple enough so that the user can figure out which category his particular application falls into without needing to know the details of the operation of the internet. The Switches of the internet should take responsibility for sending the data on a route which is capable of providing the requested type of service, and for sending the data through component networks of the internet in a way which maximizes the possibility that the type of service requested will actually be achieved. Of course, in order to do this, we must first answer a couple of hard questions, such as "Exactly what characteristics of service do users want and expect for particular applications?", and "What features must the internet Switches have, and what features must the component networks

have, in order to provide service with the necessary characteristics?" In order to give adequate communications service in an operational environment, however, these questions must be given careful consiion by internet ders. To some extent, these questions are difficult research issues, and answering them will require doing some systematic experimenn and instrumentation in the internet. The problem is hard, but unavoidable. The IP's current approach seaimed at siepping these issues, since it places the burden entirely on the user. It tends to give users the illusion that, by properly specifying the bit fields in the IP header, they can tune the internet to provide them with the specific type of service they find most desirable. This is, however, only an illusion. The perspective taken by the current IP seems to be not, "How should the internet be designed so as to provide the needed characteristics of service while providing a simple interface to the user?", but rather, "Taking the current design of the internet as a given, how can we give the user the ability to massage, bend, and twist it so as to get service characteristics which might be close to what he wants?" The former perspective seems much more appropriate than the latter.

Although we are not at present prepared to offer an alternative to IP, there are several lessons we would like to draw from this discussion:

1) While an internet Network Access Protocol really does need to contain some field which indicates the desired type of service in a manner which is abstract enough to be mapped to particular protocol features of particular networks, the proper specification of a sufficiently abstract set of parameters is an open and difficult research issue, but one which needs to be studied if an operational internet configuration is ever to give really adequate service to a relatively naive end-user.

2) Providing the requested type of service may require cooperation from all the Switches (perhaps through the routing algorithm), and involves more than just mapping fields from the internet Network Access Protocol to the particular access protocols used by the component networks. If the type of service requested by the user is to be consistently meaningful, then his request must be given UNIFORM treatment by the internet Switches. Different gateways must not be allowed to treat the request differently.

2.4 Special Features

The DoD Standard Internet Protocol contains a number of features which, while not strictly necessary in order for a user to get his data delivered, and distinct from the type of service field, do affect to some extent the service a user gets from the

internet. Some of the features are worthy of comment, and that is the purpose of this section.

2.4.1 Time to Live

The presence of the "time-to-live" field in the Catenet IP seems like a clear example of something that has no place in an access protocol. The IP specification [1] has some contradictory things to say about time-to-live. The user is supposed to set this field to the number of ss after which he no longer cares to have his information delivered, or something like that. It's far from clear how some user is supposed to make a decision as to what value to set this to. For one thing, although this value is supposed to be represented in units of one second [1, p. 24], there does not appear to be any requirement for the gateways to figure out how many seconds to decrement this value by. The spec actually says that each gateway should decrement this field by at least one, even if it has no idea how much time has actually elapsed [1, p. 40]. Well, a user might ask, is this field represented in seconds or isn't it? What is the point of saying in the spec that it is in seconds, if it is not necessarily in seconds; this will only result in confusion. That is, any attempt by a user to set this field to a reasonable value is likely to have unanticipated consequences. Any attempt to make inferences about internet behavior from the effect that various settings of the time-to-live field will necessarily be unreliable.

At any rate, unless the Switches all keep a synchronized clock, there is no real way for them to determine how long a packet has been in the network (or internet), as opposed to how much time it has spent in the Switches, and this difference may be significant if a packet is sent over several long-haul networks with long-delay lines but fast Switches. It's hard to see the point of requiring a user to specify, in the Network Access Protocol, a value which cannot be assigned any consistent meaning. (It's not clear what value this information has anyway; according to the IP spec, "the intention is to cause undeliverable datagrams to be discarded" [1, p. 24]. But a reasonable routing algorithm should cause undeliverable datagrams to be discarded anyway, no matter what value is specified for time-to-live).

It seems plain in any case that over the years, Host personnel will begin to tend to set this field to its maximum value anyway. In most implementations, the setting of this field will not be left to the end-user, but will be in the code which implements the IP. Several years from now, no one will remember the importance of setting this field correctly. Eventually, someone will discover that the data he sends to a certain place does not get through, and after months of intensive investigation, it will turn out that his IP is setting too small a value in the time-to-live field, and his packets are dying just before they reach their destination. This will make people tend

to use the maximum value as a default, reducing the utility of the information to almost nil. (No one will want to spend the time re-tuning this value to the optimum as the internet configuration expands, causing real packet delays to become longer and longer. In fact, at many Host sites there may not be anyone who can figure out enough of the Host code to be able to re-tune this value.)

Time-to-live, while useful for debugging purposes (perhaps), has no real place in an operational system, and hence is not properly part of a Network Access Protocol. If the Switches of a Network Structure want to perform packet life timing functions, in a way which is under the control of a single network administration, and easily modified to reflect changing realities, that is one thing. It is quite a different thing to build this into a Host-level protocol, with a contradictory spec, where it will certainly fall into disuse, or misuse. Protocol features which are only useful (at best) for network experimenters and investigators are bound to cause trouble when invoked at the Host level, as part of a protocol which every Host must implement, and whose implementers may not fully understand the implications of what they are doing.

Some of these difficulties have, as their basic cause, the old implicit model of the internet that we discussed in IEN 185. The IP conflates protocol features that properly belong to the

Network Access Protocol with features that properly belong to the protocol used internally among the Switches. This sort of conflation, and consequent violation of protocol layering, are inevitable if the gateways are seen as hosts which patch networks together, rather than as Switches in an autonomous Network Structure.

2.4.2 Source Routing

The current IP has a feature known as "source routing," which allows each user to specify the sequence of networks that his internet packet is to travel. We mention this primarily as an example of something that a Network Access Protocol in a truly operational environment ought not to have. An acceptable internet routing algorithm ought to distribute the traffic in order to achieve some general goal on an internet-wide basis, such as minimizing delay, maximizing throughput, etc. Any such routing algorithm is subverted if each user is allowed to specify his own route. Much of the routing algorithm's ability to prevent or avoid congestion is also compromised if certain packets are allowed to follow a route pre-determined by some user, even if the routing algorithm determines that best service (either for those packets themselves, or for other packets in the internet) would be obtained if those packets followed a different route.

To a certain extent, the presence of the source routing option in the IP is probably a result of the rather poor routing strategy in the present Catenet, and a way of attempting to obtain better service than the routing algorithm can actually provide. The long-term solution to this problem would be to improve the routing algorithm, rather than to subvert it with something that is basically a kludge. We would claim that the existence of any application or service that seems to require the use of source routing is really an indication of some lack or failure in the design of the internet, and a proper long-term solution is to improve the situation by making basic architectural changes in the internet, rather than by grafting on new kludges.

Source routing also has its use as an experimental device, allowing tests to be performed which might indicate whether it is really worthwhile to add some new feature or service to the internet. (Although the way in which source routing subverts the basic internet routing algorithm can have disturbing side-effects on the experimental results, which must be properly controlled for.) However, we doubt that any truly useful experiments requiring source routing can be performed by individual users in isolation. Rather, useful experiments would seem to require the cooperation and coordination of the participating users as well as those who are responsible for controlling and maintaining the internet. So it is not clear that there is any true utility to

having a source routing option at the level of the Network Access Protocol, thereby giving each and every user the option of using it. In an operational environment, this feature should either be eliminated, or controlled through the use of authorizations, which would cause gateways to discard source-routed packets which lack proper authorization.

2.4.3 Fragmentation and Reassembly

One of the few problems which is really specific to an internet whose pathways consist of packet-switching networks is the fact that it is difficult to specify to the user a maximum packet size to use when giving traffic to the internet. If a user's traffic is to go through EVERY component packet-switching network, then the maximum packet size he can use is that of the component network with the smallest maximum packet size. Yet it seems unwise to require that no user ever exceed the maximum packet size of the component network with the smallest maximum packet size. To do so might lead to very inefficient use of other component networks which permit larger packet sizes. If a particular user's traffic does not happen to traverse the component network with the smallest maximum packet size, the restriction really does no good, and only leads to inefficiency. Since, in a large internet, most traffic will probably traverse only a small subset of the component networks, this is quite important. In addition, some Hosts with limited resources might

have a high overhead on a per-packet basis, making it quite important to allow them to put larger packets into the internet.

This gives rise to the question of, what should an internet Switch do if it must route a packet over a certain Pathway, but that packet is larger than the maximum size of packets that can be carried over that Pathway? The solution that has been adopted in the current Catenet is to allow the internet Switches to "fragment" the packets into several pieces whenever this is necessary in order to send the packet over a Pathway with a small maximum packet size. Each fragment of the original packet is now treated as an independent datagram, to be delivered to the destination Host. It is the responsibility of the destination Host to reassemble the original packet from all the fragments before passing it up to the next highest protocol layer. (If the destination happens to have a high per-packet overhead, too bad.)

The IP has several features whose only purpose is to enable this reassembly. These features are extremely general, so that fragments can be further fragmented, ad infinitum, and correct reassembly will still be possible. However, it seems that this feature has not had very much operational testing in the Catenet; gateway implementers seem to be as reluctant to actually implement fragmentation as Host implementers are to implement reassembly. If at least one gateway does do fragmentation, then if some Host does not do reassembly, it cannot, in general, talk

to any other Host on the internet. If a source Host knows that a destination Host does not do reassembly, then it can, through IP, indicate to the gateways that they ought not to fragment. However, in that case, any datagrams that are not fragmentable but which must be transmitted over a Pathway with a smaller maximum packet size are simply lost in transit.

It should be noted that the procedure of doing reassembly in the destination Host violates the precepts of protocol layering in a basic way. The internet is not transparent to protocol modules in the Hosts, since a datagram put into the internet by a protocol module in the source Host might appear at the destination Host in quite a different form, viz., as a set of fragments. One might try to avoid this conclusion by claiming that what we have been calling "the Host software modules" are really part of a Switch, rather than part of a Host, so that no transparency is violated. One could also claim that a dog has five legs, by agreeing to call its tail a leg. But this would no more make a tail a leg than calling a Host software module "part of the network" makes it so. One of the main advantages of properly layered protocols is the ability it provides to change the network without having to change the Hosts. This is needed if changes to the network are even to be possible, since any change that requires Host software to change is, for all practical purposes, impossible. This suggests that the boundary of the network be drawn at the boundary where changes are

possible without coordination among an unlimited number of Host administrations, and the natural place to draw this boundary is around the Switches. While the Switches of a Network Structure can all be under the control of a common administration, the Hosts cannot. This suggests that any violation of protocol layering that is as gross as the need to have Hosts do reassembly is a problem that is to be avoided whenever possible.

The problems of writing Host-level software to do reassembly in a reliable manner do not seem to have been fully appreciated. If a Host's resources (such as buffer space, queuing slots, table areas, etc.) are very highly utilized, all sorts of performance sub-optimalities are possible. Without adequate buffer management (see IEN 182), even lock-ups are possible. One must remember that reassembly is not a simple matter of sending the fragments to the next higher level process in proper sequence. The situation is more complex, since the first fragment of a datagram cannot be sent up to the next higher protocol level until all the fragments of that datagram are received. If buffers are not pre-allocated at the destination Host, then fragments of some datagrams may need to be discarded to ensure that there is room to hold all the fragments of some other datagram; otherwise "reassembly lockup" is possible. If the internet gateways really did a large amount of fragmentation, so that Hosts needed to do a large amount of reassembly, this would almost certainly give rise to a variety of peculiar performance

problems and phasing effects which could make the recently discovered "silly window syndrome" look quite benign. Unfortunately, it is hard to gain an appreciation of these sorts of problems until one has personally encountered them, at which point it is often too late to do anything about them.

Performance considerations (as opposed simply to considerations of functionality) would seem to indicate that fragmentation and reassembly be avoided whenever possible. Note that performance problems associated with reassembly might crop up suddenly at any time in the life of the internet, as some Host which rarely received fragments in the past suddenly finds itself bombarded with them, possibly due to a new application. Since this sort of effect is notoriously difficult to test out in advance, one would expect potential problems to be lying in wait. Problems like these tend to crop up at a time when the Host administration has no one available who understands and can modify the Host software, which means that such problems can be very intransigent and difficult to remedy. Of course, problems in Host networking software are usually blamed on the network (i.e., on the Switches), which also does not help to speed problem resolution.

One way to remove this sort of problem from the Host domain is to have the destination Switches themselves do any necessary reassembly before passing a datagram on to its destination Host.

This has the advantage that problems which arise will fall under the domain of the Network administration, which is more likely to be able to deal with them than are the various Host administrations. However, this really does not simplify the situation, or reduce the amount of performance sub-optimalities that we might be faced with; it just takes the same problems and puts them somewhere else. ARPANET IMPs do fragmentation (though only at the source IMP) and reassembly at the destination IMP, and this has turned out to be quite a tricky and problem-strewn mechanism. Other approaches should be investigated.

Of course, one possible way around fragmentation is to adopt a policy of not routing any packets over Pathways which cannot handle packets of that size. If there are several possible routes between source and destination, which have similar characteristics except for the fact that one of them has a maximum packet size which is too small, the most efficient means of handling this problem might just be to avoid using the route which would require fragmentation. Even if this means taking a slightly longer route to the destination, the extra delay imposed during internet transit might be more than compensated for by the reduction in delay that would be obtained by not forcing the destination Host to do reassembly. Of course, this scheme requires interaction with routing, but as long as there are a small number of possible maximum packet sizes, this scheme is not difficult to implement (at least, given a reasonable routing algorithm).

Unfortunately, it might be the case that there just is no route at all to a particular destination, or else no reasonable route, which does not utilize a Pathway whose maximum packet size is "too small." In this case, there seems no way around fragmentation and reassembly. However, a scheme which is worth considering is that of doing hop-by-hop fragmentation and reassembly within the internet. That is, rather than having reassembly done at the destination (Switch or Host), it is possible to do reassembly at the Switch which is the exit point from a component network which has an unusually small packet size. Datagrams would be fragmented upon entry to such networks, and reassembled upon exit from them, with no burden on either the destination Switch or the destination Host. The fact that fragments would never travel more than one hop without reassembly ameliorates the performance problems somewhat, since the amount of time a partially reassembled datagram might have to be held would be less, in general, than if reassembly were done on an end-end basis.

A strategy of doing hop-by-hop reassembly and fragmentation also allows more efficient use of the internet's Pathways in certain cases. One problem with the end-end strategy is the essential "randomness" of its effects. Consider, for example, a large packet which must traverse several networks with large maximum packet sizes, and then one network with a small maximum packet size. The current method of doing fragmentation and

reassembly allows the packet to remain large throughout the networks that can handle it, fragmenting it only when it reaches its final hop. This seems efficient enough, but consider the case where the FIRST internet hop is the network with the smallest maximum packet size, and the remaining hops are networks with large maximum packet sizes. The current strategy then causes a very inefficient use of the internet, since the packet must now travel fragmented through ALL the networks, including the ones which would allow the larger packet size. If some of these networks impose constraints on a per-packet basis (which might either be flow control constraints, or monetary constraints based on per-packet billing), this inefficiency can have a considerable cost. Hop-by-hop reassembly, on the other hand, would allow the large packet to be reassembled and to travel through the remaining networks in the most cost-effective manner. Such a strategy is most consonant with our general thesis that an efficient and reliable internet must contain Switches which are specifically tuned to the characteristics of the individual Pathways. It also removes the problem from the Host domain, making the system more consonant with the precepts of protocol layering.

There is, unfortunately, one situation in which hop-by-hop fragmentation cannot work. If the Pathway between some destination Host and the destination Switch has a small maximum packet size, so that the destination Switch must fragment

datagrams intended for that Host, then reassembly must be done by the Host itself, since there is no Switch at the other end of the Pathway to do the reassembly. This seems to mean that Hosts whose "home networks" have unusually small maximum packet sizes will be forced to implement the ability to perform reassembly, and must tolerate any resultant performance disadvantages.

2.5 Flow Control

The topic of "flow control" or "congestion control" (we shall be employing these terms rather interchangeably, ignoring any pedantic distinctions between them) breaks down naturally into a number of sub-topics. In this section we shall be concerned with only one such sub-topic, namely, how should the Switches of the Network Structure enforce flow control restrictions on the Hosts? We shall not consider here the issue of how the Switches should do internal flow control, or what protocols they need to run among themselves to disseminate flow control information, but only the issue of how the results of any internal flow control algorithm should be fed back to the hosts. The IP is a rather unusual Network Access Protocol, in that it does not have any flow or congestion control features at all. This makes it very different from most other Network Access Protocols, such as 1822 or X.25, which do have ways of imposing controls on the rate at which users can put data into the network. The IP, on the other hand, is supposed to be a

"datagram protocol", and therefore (?) is not supposed to impose any flow or congestion control restrictions on the rate at which data can be sent into the internet. In this section, we will discuss whether this is appropriate, and whether the "therefore" of the previous sentence is really correctly used.

The issue of how flow or congestion control restrictions ought to be passed back to a Host, or more generally, how a Network Structure ought to enforce its congestion control restrictions, is a tricky issue. Particularly tricky is the relation between datagram protocols and flow control. Datagrams are sometimes known (especially with reference to the ARPANET) as "uncontrolled packets," which tends to suggest that no flow control should be applied to them. This way of thinking may be a holdover from the early days of the ARPANET, when it was quite lightly loaded. In those days, the flow control which the ARPANET imposes was much too strict, holding the throughput of particular connections to an unreasonably low value. Higher throughput could often be obtained by ignoring the controls, and just sending as much traffic as necessary for a particular application. Since the network was lightly loaded, ignoring the controls did not cause much congestion. Of course, this strategy breaks down when applied to the more heavily loaded ARPANET of today. Too much uncontrolled traffic can cause severe congestion, which reduces throughput for everybody. Therefore

many people now tend to recognize the need to control the uncontrolled packets, if we may be forgiven that apparent contradiction. Clearly, there is some tension here, since it makes little sense to regard the same traffic as both "controlled" and "uncontrolled." If a Network Access Protocol is developed on the assumption that it should be a "datagram protocol", and hence need not apply any controls to the rate at which data can be transferred, it will not be an effective medium for the enforcement of flow control restrictions at the host-network access point. If congestion begins to become a problem, so that people gradually begin to realize the importance of congestion control, they will find that the Network Access Protocol gives them no way to force the Hosts to restrict their traffic when that is necessary. The probable result of this scenario would be to try to develop a scheme to get the congestion control information to the Hosts in a way that bypasses the Network Access Protocol. This is our "logical reconstruction" of the current situation in the Catenet. When gateways think that there is congestion, they send "source quench" packets to the Hosts themselves, and the Hosts are supposed to do something to reduce the congestion. This source quench mechanism should be recognized for what it is, namely a protocol which is run between EVERY host and EVERY Switch (including intermediate Switches, not just source Switches) within a Network Structure, and which completely bypasses the

Network Access Protocol (IP). This violates protocol layering in a very basic way, since proper layering seems to imply that a source Host should have to run a protocol with a source Switch only, not with every Switch in the network.

Of course, the fact that some mechanism appears to violate the constraints of protocol layering is not necessarily a fatal objection to it. However, given the present state of the art of flow control techniques, which is quite primitive, flow control procedures must be designed in a way that permits them to be easily modified, or even completely changed, as we learn more about flow control. We must be able to make any sort of changes to the internal flow control mechanism of a Network Structure without any need to make changes in Host-level software at the same time. ARPANET experience indicates quite clearly that changes which would be technically salutary, but which require Host software modifications, are virtually impossible to make. Host personnel cannot justify large expenditures of their own to make changes for which they perceive no crucial need of their own, just because network personnel believe the changes would result in better network service. If we want to be able to experiment with different internal flow control techniques in the internet, then we must provide a clean interface between the internal flow control protocols, and the way in which flow control information is fed back to the Hosts. We must define a relatively simple and straightforward interface by which a source

Switch can enforce flow control restrictions on a Host, independently of how the source Switch determines just what restrictions to enforce. The way in which the Switches determine these restrictions can be changed as we learn more about flow control, but the Host interface will remain the same.

It is not clear that the source quench mechanism has been generally recognized as a new sort of protocol, which bypasses the usual Network Access Protocol for the internet (IP). One reason that it may seem strange to dignify this mechanism with the name of "protocol" is that no one really knows what a source quench packet really means, and no one really knows what they are supposed to do when they get one. So generally, they are just ignored, and the "procedure" of ignoring a control packet seems like a very degenerate case of a protocol. Further, the source quench mechanism is a protocol which Host software implementers seem to feel free to violate with impunity. No implementer could decide to ignore the protocols governing the form of addresses in the internet, or he would never be able to send or receive data. Yet there is no penalty for ignoring source quench packets, although violating the flow control part of the internetting protocol seems like something that really ought to be prohibited. (We have even heard rumors of Host software implementers who have decided to increase their rate of traffic flow into the internet upon receiving a source quench packet, on the grounds that if they are receiving source quench packets, some of their traffic

is not getting through, and therefore they had better retransmit their traffic right away.)

We have spoken of a source Switch needing to be able to ENFORCE flow control restrictions, by which we mean that when a source Switch determines that a certain source Host ought to reduce its rate of traffic, the Switch will REFUSE to accept traffic at a faster rate. Proper flow control can never be accomplished if we have to rely either on the good will or the good sense of Host software implementers. (Remember that Host software implementations will continue for years after the internet becomes operational, and future implementers may not be as conversant as current implementers with networking issues). This means a major change to the IP concept. Yet it seems to make much more sense to enhance the Catenet Network Access Protocol to allow for flow control than to try to bypass the Network Access Protocol entirely by sending control information directly from intermediate Switches to a Host which is only going to ignore it.

We will not discuss internal flow control mechanisms here, except to say that we do not believe at all in "choke packet" schemes, of which the source quench mechanism is an example. Eventually, we will propose an internal congestion control scheme for the internet, but it will not look at all like the source quench mechanism. (Chapters 5 and 6 of [2] contain some

interesting discussions of congestion control in general, and of choke packet schemes in particular.) It appears that some internet workers are now becoming concerned with the issue of what to do when source quench packets are received, but this way of putting the question is somewhat misdirected. When you get some information, and you still don't know what decision to make or what action to take, maybe the problem is not so much in the decision-making process as it is in the information. The proper question is not, "what should we do when we get source quench packets?", but rather "what should we get instead of source quench packets that would provide a clear and meaningful indication as to what we should do?

Does this mean that the internet Network Access Protocol should not really be a datagram protocol? To some extent, this is merely a terminological issue. There is no reason why a protocol cannot enforce congestion or flow control without also imposing reliability or sequentiality, or any other features that may unnecessarily add delay or reduce throughput. Whether such a protocol would be called a "datagram protocol" is a matter of no import. It is worth noting, though, that the Network Access Protocol of AUTODIN II (SIP), while officially known as a datagram protocol, does impose and enforce flow control restrictions on its hosts.

The only real way for a source Switch to enforce its flow control restrictions on a source Host is simply for the Switch to REFUSE packets from that Host if the Host is sending too rapidly. At its simplest, the Switch could simply drop the packets, wi no further action.omewhat more complex procedure would have the Switch inform the Host that a packet had been dropped. A yet more complex procedure woull the Host when to try again. Even more complex schemes, like the windowing scheme of X.25, are also possible. To any of these whowever, it seems that a source Switch (gateway) will have to maintain Host-specific traffic information, which will inevitably place a limit on the number of Hosts that can be accessing a source Switch simultaneously. Yet this seems inevitable if we are to take seriously the need for flow control. At any rate, the need for flow control really implies the need for the existence of such limits.

2.6 Pathway Access Protocol Instrumentation

Fault isolation in an internet environment is a very difficult task, since there are so many components, and so many ways for each to fail, that a performance problem perceived by the user may be caused by any of a thousand different scenarios. Furthermore, by the time the problem becomes evident at the user level, information as to the cause of the problem may be long gone. Effective fault isolation in the internet environment will

require proper instrumentation in ALL internet components, including the Hosts. We will end this paper with a few remarks about the sort of instrumentation that Hosts should have, to help in fault-isolation when there is an apparent network problem. We have very often found people blaming the ARPANET for lost data, when in fact the problem is entirely within the host itself. The main source of this difficulty is that there often is no way for host personnel to find out what is happening within the host software. Sometimes host personnel will attempt to deduce the source of the apparent problem by watching the lights on the IMP interface blink, and putting that information together with the folklore that they have heard about the network (which folklore is rarely true). Our ARPANET experience shows quite clearly that this sort of fault-isolation procedure just is not useful at all. What is really needed is a much more complex, objective, and SYSTEMATIC form of instrumentation, which unfortunately is much more difficult to do than simply looking at the blinking lights.

Some sorts of essential instrumentation are quite specific to the sort of Network Access Protocol or Pathway Access Protocol that is being used. For example, users of the ARPANET often complain that the IMP is blocking their host for an excessive amount of time. By itself, this information is not very useful, since it is only a symptom which can have any of a large number of causes. In particular, the host itself may be forcing the IMP to block by attempting to violate ARPANET flow control

restrictions. One sort of instrumentation which would be useful for the host to have is a way of keeping track of the total time it is blocked by the IMP, with the blocking time divided into the following categories:

1) Time blocked between messages.

2) Time blocked between the leader of a message and the data of the ge.

3) Time blocked between packets.

4) Time blocked while attempting to send a multi-packet message (a subset of 2).

5) Time blocked during transmission of the data portion of a packet.

6) Time blocked while attempting to transmit a datagram (a subset of 2).

While this information might be very non-trivial for a host to gather, it does not help us very much in fixing the problem just to know that "the IMP is blocking" unless we can get a breakdown like this. In addition, it is useful to have those categories further broken down by destination Host, in case the blocking is specific to some particular set of hosts.

Additional useful information has to do with the 1822 reply messages. What percentage of transmitted messages are replied to with RFNMs? with DEADs? with INCOMPLETEs? This should also be broken down by destination host. In fact, it would be useful to keep track of the number of each possible 1822 IMP-host control message that is received. When problems arise, it may be possible to correlate this information with the problem symptoms.

The basic idea here should be clear -- besides just telling us that "the network isn't taking packets fast enough", host personnel should be able to tell us under what conditions the network is or is not taking packets, and just what "fast enough" means. If a host is also running an access protocol other than (or in addition to) 1822, there will be specific measurements relevant to the operation of that protocol, but in order to say just what they are, one must be familiar with those particular protocols. (Again we see the effects of particular Pathway characteristics, this time on the sort of instrumentation needed for good fault isolation.) In general, whenever any protocol module is designed and implemented, the designer AND implementer (each of whom can contribute from a different but equally valuable perspective) should try to think of anything the protocol or the software module which implements it might do which could hold up traffic flow (e.g., flow control windows being closed, running out of sequence number space, failing to get timely acknowledgments, process getting swapped out, etc.).

and should be able to gather statistics (say, average and maximum values of the amount of time data transfer is being held up for each possible cause) which tell us how the protocol module is performing.

If a protocol requires (or allows) retransmissions, rate of retransmission is a very useful statistic, especially if broken down by destination host.

Hosts should be able to supply statistics on the utilization of host resources. Currently, for example, many hosts cannot even provide any information about their buffer utilization, or about the lengths of the various queues which a packet must traverse when traveling (in either direction) between the host and the IMP. Yet very high buffer utilization or very long queues within the host may be a source of performance problems. When a packet has to go through several protocol modules within a host (say, from TELNET to TCP to IP to 1822), the host should be able to supply statistics on average and maximum times it takes for a packet to get through each of these modules. This can help in the discovery of unexpected or unanticipated bottlenecks within the host. (For example, packets may take an unexpectedly long amount of time to get through a certain module because the module is often swapped out. This is something that is especially likely to happen some years after the host software is initially developed, when no one remembers anymore that the host

networking software is supposed to have a high priority. This sort of instrumentation can be quite tricky to get just right, since one must make sure that there is no period of time that slips between the time-stamps). The offered and obtained throughputs through each protocol module are also useful statistics. In addition, if a host can ever drop packets, it should keep track of this. It should be able to provide information as to what percentage of packets to (or from) each destination host (or source host) were dropped, and this should be further broken down into categories indicating why the packets were dropped. (Reasons for hosts' dropping packets will vary from implementation to implementation).

Note that this sort of instrumentation is much harder to implement if we are using datagram protocols than if we are using protocols with more control information, because much of this instrumentation is based on sent or received control information. The less control information we have, the less we can instrument, which means that fault-isolation and performance evaluation become much harder. This seems to be a significant, though not yet widely-noticed, disadvantage of datagram protocols.

Host personnel may want to consider having some amount of instrumentation in removable packages, rather than in permanently resident code. This ability may be essential for efficiency reasons if the instrumentation code is either large or slow. In

that case, it might be necessary to load it in only when a problem seems evident. Instrumentation should also have the ability to be turned on and off, so that it is possible to gather data over particular time windows. This is necessary if the instrumentation is to be used as part of the evaluation of an experiment.

REFERENCES

1. "DOD Standard Internet Protocol," COMPUTER COMMUNICATION REVIEW, October 1980, pp. 12-51.

2. "ARPANET Routing Algorithm Improvements," BBN Report No. 4473, August 1980.