Cohen

INTERNET ROUTING AND THE NETWORK PARTITION PROBLEM

IEN #120
PRTN #279
Radia Perlman
Bolt Beranek and Newman, Inc.
October, 1979

I. INTRODUCTION

As described in IEN 110, "Internet Addressing and Naming in a Tactical Environment", a network can become partitioned into two or more pieces. Assuming some of these pieces are still connected to the catenet, we would like the catenet to be able to efficiently deliver packets to a host in any such piece. Such a capability in the catenet could additionally be utilized by a scheme for delivering intranet traffic across partitions in a partitioned network.

There are four parts to the solution:
1) detecting that a network is partitioned
2) deriving a name for each partition
3) figuring out which partition a host is in
4) routing packets to the correct partition

The currently implemented gateway routing algorithm is based on the original ARPANET algorithm. To efficiently provide for routing to network partitions, routing must be based on a link state routing scheme. I will demonstrate this after first presenting the design (parts II-VIII), then showing what would be involved in modifying the original ARPANET algorithm for that purpose (part IX), and then comparing the two approaches (part X).

II. TERMINOLOGY

1) neighbor gateways--two gateways attached to the same network

2) functioning neighbor gateways--neighbor gateways able to communicate with each other over their common network

3) attached network--a network physically attached to a gateway, and with which the gateway can communicate directly (not through another gateway)

4) neighbor network of gateway G--an attached network of a functioning neighbor gateway of G, excluding attached networks of G.

III. TABLES TO BE MAINTAINED BY EACH GATEWAY

1) a list of attached networks--This list is relatively constant and is updated by a gateway when it notices a network interface is down or for some other reason the gateway is incapable of communicating with an attached network. Keeping this table updated is solely the responsibility of each gateway, and does not require intergateway communication.

2) a table of all gateways and their attached networks--This table is maintained by intergateway communication -- gateways give copies of their table 1 to all other gateways. The table of all gateways never shrinks (a down gateway is assumed to exist but be unreachable).

3) a table of link states to neighbor gateways--This table in gateway G specifies, for each neighbor gateway G1, over which common networks G and G1 can communicate. This table is updated by G periodically bouncing packets off each neighbor gateway from which it has not recently received traffic. Note that I refer to two gateways as neighbor gateways even if they cannot (temporarily, hopefully) communicate with each other.

4) a list of neighbor networks--This list is derived from the table of link states to neighbor gateways and the list of gateways with attached networks (tables 3 and 2).

5) total link state--This is a table of all gateways and the state of their links to their neighbor gateways. This table is compiled from intergateway communication. When a gateway notices that its table of attached networks, or its table of link states to neighbor gateways (tables 2 and 3) changes, that gateway efficiently broadcasts this information to all other gateways in the catenet. To minimize numbers of reports when a link is flaky, a link on an attached network must be up continuously for some amount of time before its state is considered to change from down to up and trigger a link state report.

6) shortest distance matrix--This is a data structure from which routing decisions can be made directly. It is computed from the other tables. It is described more fully in part IV.

IV. ROUTING COMPUTATION

A gateway, using the tables described above, constructs a connectivity matrix whose rows and columns represent networks, and whose entries are 1 if any gateways claim to be attached to both networks, and infinity otherwise. Then the gateway *'s that matrix to construct a shortest distance matrix. (The operation "*" consists of "multiplying" a matrix by itself, using the operations min and plus instead of plus and times, until the result stabilizes. This is a well-known algorithm.) The gateway

then looks in the shortest distance matrix for the neighbor network (or set of such) closest to the destination network, and chooses a functioning neighbor gateway (or set of such) attached to that neighbor network, to forward packets to for that destination network.

When a link state report changes the state of an entry in the connectivity matrix (remember, all gateways connecting two networks have to go down before a 1 changes to infinity), a gateway must recompute the distance matrix.

This design is a slight modification of the design presented in "Gateway Routing", by Radia Perlman (PRTN #242, PSPWN #99). The modification is that the indices of the matrix are networks, not gateways. The purpose of this modification is to make the size of the matrix smaller, an important modification given that in the catenet there are many more gateways than networks. There are aspects to the scheme that are irrelevant to a discussion of how to solve the network partition problem, such as sequence numbers for link state reports, etc. The purpose of this paper is to direct a correct approach to the design, and not to present an implementation specification. Thus an implementer should read PRTN 242 to discover the details of a link state algorithm that were not relevant for presentation here.

Note that an alternative to *'ing the matrix is to use the scheme that the ARPANET has switched over to, which is a link state scheme in which a shortest path routing tree is constructed from the connectivity information. The new ARPANET scheme is less costly to maintain as links change state. Its disadvantages are that it precludes load splitting, probably a very important problem in the case of the catenet, and is probably a little harder to implement. Since links will not change state very often, the author favors the overhead of the matrix *'ing scheme over the disadvantages of the ARPANET scheme. However, this decision is separable from the rest of the design and can be decided either way at a later time.

V. DETECTING THAT A NETWORK HAS PARTITIONED

Now we look at the problem of network partitions. In the design presented so far there is enough information for any gateway to detect a partitioned network and to isolate groups of gateways on each partition: A gateway G knows that network N is partitioned if there are two sets of gateways, set Q and set R, such that all gateways in both sets report they are attached to network N, but there are no two-way links between a member of set Q and a member of set R via network N. This information is derived independently by each gateway from the table of all gateways and their attached networks, and from the table of total link state (tables 2 and 5).

VI. DERIVING A NAME FOR EACH PARTITION

It is necessary to expand the internet header to allow a field for identifying a network partition. The reason for this is to avoid the necessity for every gateway on a packet's route to discover to which partition the packet should be sent.

The partition name must give sufficient information so that every gateway can make the proper routing decisions to send a packet to that partition, based on its tables of total link state and gateways/attached nets (tables 5 and 2).

The following schemes for naming a partition are all done independently by all gateways, as opposed to having some central authority choose a name and inform all gateways, or having a group of gateways decide on a name "by committee".

One method of identifying a partition is to use the name of any member gateway of the partition. It will not matter if two gateways choose different names for the same partition. Since the sets of gateways involved in the network partitions are disjoint, any member of the set identifies the set.

Another method is to list (either by an explicit list or a bit table) the set of gateways that make up that partition. This is unnecessarily descriptive, since the list of gateways is derivable from a single member of the set. And it is a less robust scheme, because any change to the partition (a gateway going down, coming up, or the net partitioning into more pieces) can confuse a gateway trying to route to that set of gateways. In the first method, if the partition changes, the packet will be routed unambiguously to whatever partition the named gateway is in. Of course, if the named gateway goes down, the packet becomes undeliverable, but that is easier to deal with than trying to deliver a packet to a set of gateways that overlaps two partitions.

A third method is for each gateway to number partitions from 1 to the number of partitions, ordered by, say, the highest numbered gateway in each partition. This method uses fewer bits in the packet header but is a much less robust scheme. With gateways having slightly differing information, partition names have different meanings. Also, partitions can switch names suddenly. For instance, a net can be partitioned into 2 pieces, numbered 1 and 2, and, assuming the highest numbered gateway was down, and comes up in partition 2, partitions 1 and 2 now switch identities.

Thus the recommended method of identifying a partition is the first method.

VII. FIGURING OUT WHICH PARTITION A HOST IS IN

Now we will examine several schemes for having the correct partition identified in a packet. It is the responsibility of either the source host or first gateway to do this. By examining the alternative schemes we can also determine whose responsibility it should be.

a) Source host determines correct partition by trial and error -- The source host does not know about the structure of the catenet and does not know that the destination net is partitioned. When it sends a packet to that net with no partition name filled in, the first gateway to receive the packet sends back a message that that network is partitioned, and lists the partition names. Assuming there are k partitions, the source host sends k packets requiring ACKs to the destination, each packet addressed to a different partition. The packet that receives an ACK is the one addressed to the correct partition.

If a gateway receives a packet with an incorrectly filled in partition name field, that gateway will send back the same kind of notification as for a packet with a blank field -- it will notify the host that the net is partitioned and list the partition names, or if the net is no longer partitioned, give that information.

If the source host is sending packets that require acknowledgments, it will notice quickly if its packets stop getting successfully delivered to the destination. Then it can redetermine the host's partition.

b) The first gateway, using trial and error -- If it is the first gateway that has the responsibility, it can do the same thing as the source host in scheme a, sending packets to the destination addressed to each partition to discover from which partition it receives an ACK. Since a network is unlikely to be partitioned into very many pieces, it is not costly to try all partitions. Either the correct partition will be found or no ACK will return (in which case presumably the host is down or the network is partitioned in such a way that some hosts are unreachable from all gateways). The disadvantage of having the first gateway do the work in this scheme is that a gateway does not know whether packets it is forwarding successfully reach their destination. Thus it must either keep a cache of host/partition correspondence, which can be out of date for some amount of time during which the gateway will misaddress packets to a destination, or the gateway must rediscover the correct partition on a packet by packet basis, which is of course unacceptably expensive. Also, assuming it is common for a source host to split its traffic among several gateways on the source net, after a gateway discovers the correct partition for a destination host it should inform all other gateways on the source net of the correct partition, to prevent the necessity of them rediscovering that fact.

c) gateways on a partitioned net could keep track of host/partition correspondence for their net -- Another method is for gateways on a partitioned net to find out which hosts they can reach, and exchange that information with the other gateways on that partitioned network. Then a gateway could respond more intelligently to a packet addressed to the incorrect partition by sending back a message giving the correct partition (to the packet source if that is who fills in the partition field in the packet header, or to all gateways on the source net otherwise). In addition, a gateway on the partitioned network can forward the misaddressed packet to the correct partition.

This method requires gateways on the partitioned network either to keep a complete list of the hosts on the net, marked as to partition, or to keep a cache of hosts, adding hosts to the cache by querying the gateways on other partitions at the time the necessity of locating that host arises. In the complete list case, gateways on a partitioned net would periodically send packets requiring ACKs to all hosts on that net in order to keep their lists up-to-date. In the cache case, gateways would poke a host only when the need to know its location arose (when the gateway received a packet for that host, and the host was not already in its cache, or when a query from a gateway on a different partition of the net arrived, asking for that host's location).

This method suffers from the same problem as method b, with the first gateway having responsibility for determining host/partition correspondence -- the tables in the gateways on the partitioned net can become out of date, during which time they will misdirect traffic, and they cannot constantly be checking their tables.

Thus I recommend method a, having the source host fill in the partition field using the trial and error method of discovering host/partition correspondence.

VIII. ROUTING PACKETS TO THE CORRECT PARTITION

As stated above, a gateway G, distant from partitioned network N, must know which gateways are involved in a partition before G can correctly route a packet -- it might have to make a different routing decision for one partition than for another one.

When G detects a network has become partitioned into n pieces, G must add n-1 rows and columns to its shortest distance matrix, i.e., it treats each partition as a separate network. It is an implementation detail, and not a difficult one, to ensure that the gateway understands the meaning of each row and column. And given that the gateway understands the meaning of each row and column, it is easy for it to fill in the connectivity matrix from its table of total link state. The computation is done exactly as in the nonpartitioned case.

IX. MODIFYING THE ORIGINAL ARPANET ROUTING FOR PARTITIONS

The original ARPANET routing is the currently implemented routing algorithm in the gateways. The basic design is that gateways report their distance vector to all their neighbor gateways (their distance vector gives their distance to all destination nets). They derive their distance vector from their neighbors' distance vectors. (A gateway's distance to a destination net is 0 if the gateway is directly attached to the destination net. Otherwise, it is 1 hop further than the neighbor closest to the destination.)

The major modifications that are necessary to handle partitioning are:

1) Currently distance vectors are just a list of numbers, and gateways have an assembled-in offset/net number correspondence. Thus the vectors do not need labels for each entry. If networks became partitioned, more destinations would need to be reported in the distance vector. Either some (very complicated) negotiation process would need to be carried out so that all gateways would agree, when nets became more or less partitioned, on a new offset/net number correspondence, or the distance vectors would need labels identifying the destination whose distance is being reported. The problems associated with a negotiation process make that scheme unworkable. Thus we can assume the vectors would be expanded to have an identifying label for each destination. The label would include net number and partition name.

2) Gateways do not have global knowledge of the structure of the catenet, in contrast to a link state scheme. Thus it is the responsibility of the gateways on a partitioned network to notice that the net has become partitioned and start a routing update.

In the current implementation, there is no way for gateways on a partitioned net to tell the difference between having their net partitioned and having several gateways on their net go down, since they do not receive information about individual gateways -- they only receive distance vectors from their neighbors. They will no longer receive distance vectors from their neighbors on a partitioned net, or from neighbors who have gone down, so lack of response from neighbors does not distinguish between dead neighbors and a partitioned network.

Thus either distance vectors would have to contain information about all catenet gateways (which adds a great deal of overhead since there are many more gateways than nets, and the only purpose of doing that is to detect partitions) or gateways on a network would report that the network has become partitioned every time a gateway goes down.

3) Gateways in a partition must agree on a partition name, since if two of them started a routing update with two different names for the same partition, the rest of the catenet can draw no conclusion except that the two partition names refer to distinct destination partitions. Agreeing on a name is not that easy. If some simple algorithm is chosen, such as highest numbered gateway in that partition, the name of a partition can change. Suppose the old partition name was 5 and it changes to 12. A source host (or distant gateway) has gone through the overhead of determining that the proper partition for a destination host was 5. When the name of the partition changes, this overhead must be repeated. Also, when the name of a partition changes, the rest of the gateways on the catenet must be informed of that fact so that they will stop reporting about obsolete partition names in their distance vectors.

X. COMPARISON OF LINK STATE AND ORIGINAL ARPANET SCHEMES

The link state scheme is far more robust. Because gateways have global knowledge, routing is more likely to proceed calmly while routing updates are percolating throughout the catenet. Partition names are not as important in the link state scheme -- gateways do not have to agree on a single name for a partition.

As stated above, because in the currently implemented scheme gateways report only their distances to destination networks, and not to individual gateways, either gateways would report network partitions whenever gateways went down, or the distance vectors would have to be expanded to include reports about all gateways. This is a further disadvantage of the original ARPANET scheme to this application.

Another disadvantage of the original ARPANET routing, not related to partitioning, is that, because nodes do not have global knowledge of network connectivity, there are types of routing loops which they cannot distinguish from degradation of best routes due to connectivity changes. As currently implemented in the catenet, nodes report their distance to a destination as "infinity" (a number higher than the maximum possible distance in the catenet) when reporting to downstream neighbors. This fixes many kinds of routing loops. However, neither this scheme nor any variant (such as hold-down, the scheme chosen by the ARPANET as a modification of the original algorithm) can distinguish all kinds of routing loops from connectivity changes. Thus there are cases when a group of nodes will have to count up their distance to a destination until it reaches "infinity" before discovering the destination is unreachable. This does not make the scheme unworkable for the current catenet, since the longest possible path in the catenet is less than 10 hops. However, it is again a further disadvantage of the original ARPANET scheme.

Another important consideration is the link state scheme's flexibility. There are new features that the catenet is scheduled to provide, most notably extended routing, in which the functional differences between links are recognized and accounted for. As described in IEN #86 "Extended Internet Routing", by Radia Perlman, a link state scheme must be adopted eventually in order for the catenet to provide this service.

Thus the link state approach should be adopted to provide for network partitioning.

XI. CONCLUSIONS

A link state scheme, as originally presented in PRTN 242, modified as presented in part IV of this paper should be the basis of internet routing.

The internet header should include a field long enough for a gateway ID, for the purpose of specifying a partition name. A partition name is the ID of any member gateway on that partition.

The first gateway that handles a packet checks to see if it is addressed to a partitioned network. If so, and if the partition name field in the internet header is blank, the gateway sends back a special packet to the source host informing it that the network is partitioned and giving it a name for each partition of that network. When a gateway on the source net handles a packet for an unpartitioned network in which the partition name field is not blank, it erases that field and informs the source that that network is no longer partitioned.

When a source host receives notice that a network is partitioned, it stores the partition names for that network, and when it wishes to send a packet to a host on that net, it first tries all partitions to determine the correct one. It keeps a cache of host/partition correspondence. When packets for a host in its cache no longer reach the destination, the source host should again attempt to determine the correct partition for that host.

APPENDIX
COMBINING USUALLY SEPARATE NETWORKS

In IEN 110, Dr. Vinton Cerf raises the possibility of combining nets, given that the catenet could handle a partitioned network. In general if the networks in question are usually partitioned, this is a bad idea, since there is overhead involved in having a partitioned network. Every time a source wishes to send a packet to a destination, someone must discover which partition to send the packet to.

However, the specific example discussed in IEN 110 is an example where there is also a cost associated with not combining networks. In the example there are two ground PR nets, A and B. There are also a number of PRs on airplanes, call them P1, P2, ... Pn. When Pi is within range of a PR in net A, Pi automatically becomes a part of network A. When Pi is within range of both PR nets, the nets become a single PR net.

Keeping the two nets separate leads to problems of addressing the airplane PRs, since the net on which they reside changes. Combining the two nets into a single network has the overhead of introducing a usually partitioned network into the catenet.

There is a third solution to the particular case involved here. That is to keep networks A and B as separate logical networks, and to have P1, P2, ... Pn also as separate logical networks on the internet level. On the packet radio level there might be only one net, because one of the Pi connects nets A and B. But on the internet level there will be n+2 nets.

A gateway on net A, called G1, will have a half gateway associated with each of the nets it might be "directly connected to" in the internet sense. In other words, it will have a half gateway for A, P1, ... Pn. The half gateway associated with network A determines whether its interface to net A is up or down depending on the state of the hardware ready line, etc., as is now done. The half gateway associated with "network" Pi must determine whether it is "connected" to its "network" by some other means. One method is to have a special querying packet containing the number i. The packet would be addressed, with a local header only, to Pi, and sent out the interface to network A. Pi's responsibility, upon seeing this querying packet, is to send back a special answering packet, also containing the number i. The half gateway associated with network A, upon receiving one of these special answering packets, uses the number contained in the packet to dispatch the packet to the half gateway associated with Pi. The half gateway associated with Pi, upon receiving this special answering packet, knows that its "network" is up.

G1's list of neighbor gateways will include, besides all the gateways on net A, all the gateways on net B, since a gateway on

net B also has the Pi as potential attached networks. If some Pi connects nets A and B, then the gateways on A and B will all consider each other functional neighbors, and A, B, and the connected Pi, which have formed themselves into a single functional PR net, will function as a single net on the internet level, too. If one of the Pi is not within reach of either net A or net B, then all the gateways on nets A and B will report that they are not attached to net Pi, and all the gateways in the catenet will know Pi is unreachable. If A and B have not merged into one net (none of the Pi are in both nets), then the gateways on each will report which Pi are reachable from them, so the catenet will automatically route packets for Pi to the correct ground PR net.

[It would be reasonable to include, in gateway G1, a half gateway for net B also, since if nets A and B merged, G1 would be connected to net B. However, it is not necessary to and is slightly more efficient not to, since even if nets A and B are merged, PRs in B are probably physically closer to the gateways on net B, so the catenet should route packets for PRs in B to the gateways that "really" are on ground net B. The advantage of including a half gateway for B in G1 is that net B could potentially partition in such a way that some partition included no gateways from B, but was reachable in the catenet via net A and some Pi. It is not obvious, however, what algorithm a half gateway for B should use to determine whether its "network" is up.]

The airplane PR Pi does not think of itself as a network. From its point of view it is an ordinary PR. The only difference between Pi and an ordinary PR on net A is that Pi (or the TIU attached to Pi, if we want to strictly adhere to packet radio terminology) has stored as its internet address, Pi for its net number. It also has a list of possible gateways to use for internet packets. This list includes all the gateways on nets A and B. In the current PR net there is only one gateway, and all PRs know the ID of the gateway. This will change such that there will either be a special ID for an information service that will give out the ID of a gateway on the net (so that Pi, instead of keeping a list of gateways, could ask for a gateway address, as would the rest of the PRs on nets A and B) or all PRs will have assembled in a list of gateways, and they will need to probe each in turn until they find one that responds. Thus the only difference in Pi's finding a gateway and in an ordinary PR on net A finding a gateway, is that (assuming the assembled-in gateway list scheme is used) Pi's list will be longer, since it will also include the gateways on net B.

There is obviously a cost associated with this solution, too. If the number of Pi are small, then this is a reasonable solution. If there are enough Pi, then the cost of having all those logical nets becomes greater than the cost of having an often partitioned network, so the solution of combining A, B, and all the Pi into one logical net in the catenet is a more practical solution.