FLYING PACKET RADIOS AND NETWORK PARTITIONS

IEN #146
PRTN #292

Radia Perlman
Bolt Beranek and Newman, Inc.
June 1980

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.

Another problem is known as the flying packet radio problem, in which there are two ground PR nets and an airborne PR, potentially in radio contact with either or both ground nets. The problem is to route internet packets to that airborne PR.

In IEN #120 I presented a design for network partitioning and for the flying packet radio problem. This paper differs from IEN #120 in several ways:

1) In this paper there is a simpler solution proposed for finding the host/partition correspondence.

2) In this paper an argument is made for doing the link state routing algorithm in a straight per gateway (rather than per net) computation. This is more costly, since there are more gateways than nets, but it is more straightforward to implement, and is more flexible.

3) A different solution is proposed for the flying packet radio problem. The solution in IEN #120 was an easily implementable one that could be immediately implemented. The solution in this paper depends on the rest of the design being implemented, but is a less costly solution.

4) In IEN #135, Carl Sunshine and Jon Postel present an alternative approach to the flying packet radio problem. A comparison of the two approaches is made in this paper.

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. The necessity for a link state routing scheme was demonstrated in IEN #120. In this paper I will merely present the design.

II. LINK STATE ROUTING

A "link state" routing scheme is one in which the nodes computing the routing have complete knowledge of the state of all the links in the network. All nodes monitor the state of their links to their neighbors, and report this information to the nodes that compute routing. In a totally distributed algorithm, all nodes compute routing, so that means that all nodes must broadcast the state of their links to every other node.

A link state scheme is currently in operation in the ARPANET. An alternative to a link state scheme is the algorithm that used to be in operation in the ARPANET, and is currently implemented in the gateways. In the old-style ARPANET routing algorithm, nodes give to their neighbors a vector of their distance to all destinations, and a node compiles its own distance vector by taking the minimum distance of its distance to a given neighbor, plus that neighbor's distance to the destination. The advantage of a link state scheme over the old-style ARPANET scheme is that a link state scheme is more flexible, since nodes have more information.

The most straightforward link state scheme would be one where each gateway computes routes from itself to all other gateways. This design was specified in IEN #24 (also known as PRTN #242). Let us call that scheme the per-gateway scheme.

In IEN #120 (PRTN #279) I proposed a modification to the per-gateway design, wherein gateways computed routes to destination networks rather than destination gateways. Let us call that scheme the per-network scheme. The per-network scheme is computationally less costly for the gateways, since there are more gateways than nets. However, there is a problem with the per-net scheme. The problem is that in the per-net scheme, different costs cannot be assigned to different pairs of gateways on the same network. And in networks like the packet radio net, or the ARPANET, the delay between very distant gateways on the same net can be very different from the delay between close gateways on that net. Currently there is no mechanism for measuring delays between neighbor gateways, and the cost function is the simplest possible -- number of hops. However, at some point in the future we might want to use a more sophisticated cost function. Thus I recommend abandoning the per-network approach and going to the straightforward per-gateway approach.

Currently there are few enough gateways so the per-gateway approach would not be a problem. If in the future there are too many gateways to make this approach feasible (more than 100), there are other approaches that can be taken. For example, instead of a totally distributed algorithm, there can be a few "routing centers" distributed around the catenet. These routing centers would be large enough machines so that they would not have space problems with computations involving hundreds of

nodes, and do not have to be gateways, so the time involved in computing routes would not degrade gateway forwarding performance. This would make the link state scheme less costly, since gateways would only have to report the state of their links to the routing centers, not to all gateways.

If the number of gateways was truly huge (more than a few hundred), it would not be practical even for a large routing center to compute routes for a network that large. In that case a heirarchical approach, of breaking the net into subnetworks and treating the network of subnetworks as an approximation to the entire network should be used. This approach has been taken in the multistation packet radio design, which is a design to accomodate a very large network of PRs.

If it is decided that the capability of assigning different costs to different pairs of gateway links is not essential, the per-network scheme might be adopted, so I will include the description here.

III. 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

IV. 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 V.

V. ROUTING COMPUTATION

5.1 Per-Network Scheme

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, for forwarding packets to that destination network.

If the cost function assigns different costs to different networks, then instead of merely putting a "1" in the connectivity matrix where there is connectivity, the gateway does the following. If the assigned cost (a constant) of network A is C(a) and the assigned cost of network B is C(b), then in the connectivity matrix for the entry [A,B], deposit C[a]. In the entry [B,A] deposit C[b]. In other words, assign the cost of the network you are leaving.

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 an entry 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, IEN #24). 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.

5.2 Per-Gateway Scheme

This scheme is more straightforward. The rows and columns in the connectivity matrix represent gateways. If different costs are assigned to different gateway links on the same network, gateways would report the cost of their links to their neighbors in their link state reports, and this cost would be deposited into the entries in the connectivity matrix.

As in the per-net scheme, the connectivity matrix could be *'ed, or the Dijkstra algorithm could be applied.

VI. 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).

VII. 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.

VIII. FIGURING OUT WHICH PARTITION A HOST IS IN

This is the aspect of the design for which I did not find the design presented in IEN #120 completely satisfying. Here is a better approach.

Important goals are to:

1) Shield gateways from state information such as which hosts are in which partitions.

2) Shield hosts from the necessity of knowing much about the structure of the catenet. In particular, since hosts do not receive gateway link state reports, they do not know which gateways and links are up, do not dynamically discover new gateways and networks, and thus cannot intelligently provide a complete source route on a packet. And requirements for sophistication on the part of the host means adding that sophistication to many different implementations.

The proposed solution is that:

1) If a gateway receives a packet for a partitioned destination network, with no partition name filled in in the packet header, that gateway duplicates the packet, sending a copy of the packet to each partition. (Subsequent gateways will not duplicate the packet because the first gateway would have supplied partition names on the packets it sent out.)

2) Gateways on a partitioned network fill in their IDs in packets leaving the partitioned network.

3) Hosts communicating with a host on a partitioned network can either ignore the whole network partitioning issue, or copy the partition name from packets returning from the host on the partitioned network. If hosts ignore the partitioning issue, the cost is duplicated packets. If hosts choose to copy the information, they must keep state information per host on that partitioned network, and they must notice when that information becomes out of date (if packets fail to reach their destination, the host should erase its knowledge of which partition the packet was routed to, since the other host might have moved to a different partition).

One advantage of this design is that gateways can be completely sheltered from per-host state information. They already detect partitioned networks, so the only added work is duplicating packets and filling in partition names. Another advantage is that hosts can either be totally oblivious of the whole issue, at

the expense of duplicated packets, or they can, without much work, obtain the information of the proper partition for a given host. And the decision as to which course to take can be taken independently by each host.

IX. 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.

X. FLYING PACKET RADIOS

In IEN #110, Dr. Vinton Cerf raises the following problem. An airborne PR flies above two ground nets, A, and B. At times the airplane PR is in radio contact of A, B, both nets, or neither ground net. The problem is for hosts in the catenet to send packets to the airplane PR. They cannot simply fill in "A" or "B" as the destination network because that might be incorrect. And if they did somehow know the proper net at the time, higher level protocols would get confused by a changing net number, and be unable to match packets to an existing connection.

In IEN #120 I presented a scheme for solving this problem that could be implemented without the rest of the network partitioning design. That scheme involved assigning a virtual net number to each airplane PR. Gateways on the ground nets A and B would have half gateways associated with each virtual net, that "pinged" the associated airplane PR occasionally to see if it was reachable. The cost of this solution is traffic overhead in the ground nets, from all the pinging, and extra net numbers for each airplane PR. However, it is easy to implement.

I will present here a solution that is much cheaper, but depends on the rest of the design in the paper being adopted.

The solution is that:

1) A single virtual net number, P, is assigned to include all airborne PRs.

2) Gateways on A and B always report that they are connected to "net P" and that their links to their "neighbors" on the other ground net, via net P, are always down. (They could report their link via net P to be up if some airplane PR connects the two ground nets so that they actually can reach the other ground gateways, but it is simpler to declare that link always down, and it will avoid forcing the airborne PRs to forward much traffic.) Gateways on A report their links to the other gateways on A, via net P, to be in the same state as the actual links via net A. The gateways on net B do likewise.

3) Thus P will look to the rest of the catenet like a partitioned net. Consequently, gateways receiving packets for P with a blank partition name will duplicate the packet, sending a copy to each "partition" of P, i.e., a copy to net A and a copy to net B. And gateways on nets A and B receiving packets from "P", will fill in their IDs as the "partition name" of P.

4) Hosts talking to an airborne PR can either ingore the whole problem and let its packets get duplicated, or copy the proper "partition name" in packets it receives from the airplane. However, since airplanes are liable to switch nets quickly, the information is liable to become quickly out of date. Thus it is probably better (assuming there are only 2 ground nets) to live with the duplicated packets, ensuring delivery (assuming the airplane PR is reachable via one of the ground nets).

The cost of this approach is the implementation of all the rest of the network partitioning design presented in this paper, plus a single virtual net number and duplicated packets to the airplane PRs (or hosts copying the information provided in packets from the airplane PRs).

XI. COMPARISON WITH IEN 135

In IEN #135, Carl Sunshine and Jon Postel present an alternative approach to the flying packet radio problem. Their approach is:

1) A virtual net number is assigned for all airplane PRs.

2) Airplane PRs have the responsibility for ascertaining their current network location, and reporting this information to a global database. (There can be multiple global databases for reliability.)

3) Hosts wishing to communicate with an airplane PR request their location from the global database, which furnishes them with a single level source route to write into the packet header. The source route consists of both a net number (as in ground net A or ground net B, depending on which net the airplane in question is currently in radio connectivity of) and a local address on that net, called a "forwarder". The purpose of the "forwarder" is merely to fit this into the already existing mechanism of source routing, which requires a full internet address. It would be most convenient to have as the ID of the "forwarder" the same ID as the destination, so that the destination would be net P, host XXX, and the source route would be net A (or B), host XXX.

The authors propose this scheme over the one presented in IEN #120 (and the one presented here) in order to save the gateways from dealing with the problem.

Costs of the scheme in IEN #135 are:

1) Maintaining a global database is complex and costly.

2) The airplane PR must ascertain its true net and report this information to the global database. There is no mechanism currently designed into packet radio networks to make this easy, and certainly no mechanism for alerting the airplane PR that it is "about to leave" a net, as suggested in IEN #135.

3) Hosts wishing to communicate with an airplane PR must first contact the global database. This is extra code that must be implemented in order for the host to communicate at all with the airplane PR. And it must be implemented in every host that might be in contact with an airplane PR.

4) Airplane PRs are liable to change nets so quickly that by the time the airplane discovers it has changed nets, contacted the global database, the host has queried the database, and the host has received a reply from the database, the airplane PR has very likely changed nets.

The costs of the scheme presented in this paper are:

1) Implementing a link state routing algorithm for gateways. A link state algorithm requires more control traffic and more computation than the old-style ARPANET algorithm. It requires a recoding of the gateways, since they currently have implemented the old-style ARPANET algorithm. However, the extra overhead of the link state scheme is not that bad, and there are possibilities for decreasing the overhead further (for instance, by declaring a set of gateways all connected to the same networks and all in contact with each other as a "group" and having a single member of the group report status information for the entire group). And the link state scheme gives needed flexibility for other internet routing problems, for instance extended routing (including access control).

2) The rest of the partitioning design, all of which is minor, including:

a) having gateways detect a partition and compute routing accordingly
b) having gateways receiving packets for a partitioned network duplicate the packets
c) having gateways on the partitioned net fill in the partition name on outgoing packets
d) extra packets or having hosts communicating with a host on a partitioned net copy the partition name from incoming packets

3) For the flying packet radios, a single virtual net number.

XII. 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 duplicates the packet for each partition, and sends a copy to each partition (filling in the partition name on each copy).

When a host receives packets with a partition name filled in, it can copy that information in a per host table, being careful to erase that information if packets fail to reach the destination. Hosts that choose not to implement that will cause nothing more serious than duplicated packets.