IEN 24

Section 2.3.3.8

GATEWAY ROUTING

Radia Perlman

Bolt Beranek and Newman Inc.

January 27, 1978

PRTN #241

PSPWN #99

GATEWAY ROUTING

This paper proposes an algorithm to improve gateway routing. It is in two parts because the second part is an extension to the first. The first part can be implemented without the second, and is a scheme to route packets successfully around failed gateways. The second is a scheme to improve overall flow, both by distributing traffic more evenly throughout the catenet (the collection of nets connected by gateways) and by providing feedback to sources when they are producing traffic too quickly.

This paper is not a design specification -- there are some details yet to be worked out and some issues yet to be resolved. This is written to convey our current thinking on the topic.

Another paper has been released by Virginia Strazisar on gateway routing. It is a proposal to use Gallager's algorithm and is a different approach to the problem. This scheme is conceptually simpler. It does not attempt to achieve theoretical optimal limits of overall catenet traffic throughput or overall catenet traffic delay. Instead it achieves good results with less overhead.

It is not necessary to have read Strazisar's paper in order to understand this one.

1.1 Introduction

This algorithm is designed for finding reasonable paths, not optimal ones. It does not attempt to optimize either delay for a single packet, or the average delay throughout the catenet. It is designed for simplicity, low cost in terms of the information which must be exchanged between gateways, and workability in cooperating with gateways that do not implement this algorithm.

The basic idea is that each gateway cooperating in this scheme has a model of the connectivity of the catenet. As events occur which change the connectivity of the catenet, these events are reported and circulated throughout the catenet.

Each gateway uses this information to compute a distance matrix of shortest distance paths between each pair of nodes. Packets are routed on shortest length paths.

1.2 Terminology

catenet -- the collection of nets connected by gateways. For the purpose of describing this algorithm, gateways will be considered nodes in a graph, with networks considered to be links.

neighbors -- gateways on a common network, i.e., gateways that can potentially communicate without intermediate gateways. We will refer to gateways as neighbors even if communication between them is currently failing.

simple gateways -- gateways that do not implement this algorithm.

1.3 Connectivity Information

Connectivity information is passed in the form of link-state packets, in which a gateway reports which of its neighbors it can communicate with. (Gateways contain an assembled-in list of neighbors, and a new gateway starting up gets in touch with its neighbors so that they can add it to their list if it was not already there.) A gateway can report this information periodically but will definitely report it when the state of a link between itself and one of its neighbors changes. When a node A hears the packet generated by D about the state of the links to D's neighbors, A reports the information to each of its neighbors. In order to avoid transmitting duplicate information, a sequence number will be transmitted with the connectivity information. A node A remembers the last sequence number associated with D that it reported to its neighbors and will not re-report connectivity information for node D until it receives information with a higher sequence number.

Determining the status of a link is dependent on the networks involved. One way is to send a neighbor a self-addressed packet and see if it gets back. This works for simple gateways, too.

When a gateway comes up, it will not know what the sequence numbers are, either for information it must report or for information from other sources. For this reason and because of

the possibility of lost packets, neighbors will exchange sequence numbers packets, which contain the current sequence numbers for all nodes.

1.4 Routing Decision

There must be some sort of associative nonnegative distance function for routes through the catenet. A gateway G will route a packet destined f. ode D to a node B such that d(B,D) < d(G,D). Requiring the distance function to decrease at each hop assures no steady-state loops.

The simplest such distance function is the number of hops (i.e. nets). A gateway will always choose the route with the fewest number of hops. If there are two such routes, the gateway can load split, sending half its packets down each path.

It is important that all gateways use the same distance measure for choosing routes or it will be possible for a steady-state loop to form (A could send traffic for C through B because this route has the fewest number of hops, and B could send traffic for C through A because that path has the smallest delay). In this algorithm, temporary loops are not serious, because connectivity information is reported to all nodes as soon as received, so that deadlocks cannot occur. A packet might loop a few times but the loop will eventually go away when the connectivity information is received.

In the description of the algorithm we will use number of hops as the distance function. The algorithm generalizes to any

distance function, but we will use hop count in the initial implementation.

1.5 Sequence Numbers, Clocks, etc.

If some global clock or monotonically increasing number were kept between gateways, connectivity information could be stamped with the clock instead of a sequence number. This has the advantage that gateways would not have to store sequence numbers for all gateways, and would eliminate the need for sequence-numbers packets. However, it is undesirable to wait for a global clock to be implemented before implementing improved gateway routing.

Another possibility is having each gateway contain a hardware clock that remains running even when the gateway crashes, or even if the power fails. Such are available, but it seems undesirable to require all gateways to have one.

Therefore we will implement sequence numbers. When generating a link-state packet, a gateway simply increments the number used on the previous packet, modulo 16 bits. When hearing a link-state packet generated by gateway G, a gateway decides it is recent if the sequence number is higher than the last sequence number from G.

To keep neighbors up-to-date with each other, sequence-numbers packets will be exchanged. These packets contain the latest sequence numbers for each gateway. A neighbor

N, upon receiving a sequence-numbers packet from gateway G, checks to see if G has sequence numbers at least as recent as N has. For each node A for which N has a higher sequence number than G, N will send the most recent link-state packet from A that N knows about. Thus sequence-numbers packets serve as requests for latest connectivity information.

There will be one sequence number, say Ø, reserved to mean "sequence number not known". This will enable a gateway starting up to request complete information from a neighbor. When a gateway is not heard from for a long time, gateways should "time out" the old sequence number and flag the sequence number as old. Then, when the gateway comes back up, any sequence number will be accepted.

When a gateway G first comes up, it sends a sequence-numbers packet to a neighbor A, with all sequence numbers Ø. If A has a sequence number for G (because G was down for a sufficiently short amount of time that A did not time out the sequence number), G will use that sequence number. Otherwise G can use any sequence number.

1.6 Gateway internal algorithm

Ignoring any simple gateways, each gateway has a table of all gateways, in a matrix. Entry i,j=Ø means i=j; entry i,j=1 means i and j are neighbors and the link between them is up; and entry i,j=infinity means they are not neighbors or the link between them is currently down. (A link is down if either

neighbor thinks it is down.) To find out which neighbor to send a packet to for each gateway in the catenet, the gateway "squares" the matrix at most log n times (base 2), where n is the number of gateways. "Squaring" means performing matrix multiplication of the matrix by itself using the operations plus and min. If the connectivity matrix is represented as C and the square of the connectivity matrix as C2, this operation can be stated as:

$$C2(i,j) = \min \text{ over } k \; [\; C(i,k) + C(k,j) \;]$$

The matrix is squared repeatedly (by setting C = C2 and repeating the above operation) until the matrix resulting from the squared operation, C2, is the same as the previous matrix, C. The first time the matrix is squared, the entries i,j are either infinity or are path lengths for paths less than 2 hops in length. Squaring again yields all paths of length 4 or less, etc. In the final matrix, entry i,j is the length of the shortest path from i to j. If i is a neighbor of j, the shortest path will be with k equal to either i or j and with the entry i,j in the final matrix equal to 1.

As a result of this operation, the gateway has a matrix of distances between gateways. To decide to which neighbor to send a packet destined for j, the gateway scans the matrix entries k,j for each neighbor, k. It chooses the neighbor k with the smallest value for entry k,j. If desired the gateway can construct a table indicating to which neighbor to send a packet

for each destination in the catenet. The gateway can then simply look up to which neighbor to send a packet instead of checking each neighbor's entry in the distance matrix. Note that this table would have to be recomputed each time the matrix changes.

1.7 Noticing out-of-date information

If a sequence number is very different than expected, or if a gateway G receives a packet from a neighbor N, which G thinks is closer to the destination than G is, G can send N a sequence-numbers packet, which will cause N to send any link state packets G has missed, and which will cause N to send G a sequence-numbers packet if N has missed any link-state packets. Since the connectivity information for each node has its own sequence number (or timestamp), each of the gateways can benefit from this exchange (each might have more up-to-date information than the other on different pieces of information).

1.8 Format of Routing packets

There are two kinds of packets -- link-state packets and sequence-numbers packets. The link-state packet tells the state of links with all neighbors. The sequence-numbers packet contains sequence numbers for all gateways (from which can be deduced how up-to-date the gateway's information is).

Link-state packets get broadcast throughout the catenet. Sequence-numbers packets are sent only to the neighbors of the gateway generating the packet.

Link-state packets

These packets contain the state of all links from the reporting node, in the form:

node number (# of gateway reporting)
sequence number
neighbor number
link state
neighbor number
link state
etc.

Including all links has the advantage that gateways only need to remember as many sequence numbers as there are other gateways, not as many as there are links in the catenet.

When a gateway G hears a link-state packet from node A, G ignores it if the packet's sequence number is not greater than the last link-state packet it received from A. If the sequence number is greater, G sends the packet to all of G's neighbors.

Sequence-numbers packets

The sequence-numbers packet contains sequence numbers for all the nodes as understood by the sender. It is of the form:

node number (# of gateway reporting)
gateway #
sequence number
gateway #
sequence number
etc.

These packets are not passed around the catenet but can merely be used by gateways to make sure they have not missed the latest connectivity packets from each node. If a gateway sees that a neighbor has an out-of-date sequence number for any node, the gateway should send the latest link-state packet from that

node to the neighbor with out-of-date information. Thus it is important to save the latest link-state packet received from each gateway. These packets therefore serve two functions: keeping neighbors informed about each other's states, and requesting retransmission of specific link-state packets.

1.9 Undesirable Paths

If there are some paths that are undesirable for some reason, the undesirable link can be arbitrarily weighted against by assigning it a hop value greater than 1. All gateways would have a list of undesirable links, and when the link was reported as up, they would insert the weighted number rather than 1 in the distance matrix.

This is a way of generalizing the distance function slightly without making the algorithm more complex.

1.10 Simple Gateways

Simple gateways will not exchange information about themselves or their links. Thus gateways will not discover a new simple gateway automatically. Therefore the only simple gateways that will be known will be ones that at least one neighbor gateway knows about (either because it was assembled into the program or because some human put it into the table).

There are a number of ways the information of where a simple gateway exists can be reported and transmitted. One reasonable way is for neighbors that know about it to include information

about the simple gateway in their link-state packets, with link state numbers reserved for meaning "simple gateway with link up" and "simple gateway with link down".

Since the status of links between simple gateways can not be reliably determined, gateways will use paths through simple gateways only when no other path exists. It will be assumed that any path between two simple gateways is always up.

To assure paths through simple gateways are adequately weighted against, a hop value equal to the number of known gateways can be assigned to any link with at least one end being a simple gateway.

1.11 Traffic to a Net (not a gateway)

Most traffic is destined not to a gateway, but to a host on a network. There could be many gateways attached to the network, and a gateway G must choose which of the gateways attached to the relevant net to send a packet to. This can be done by considering networks as pseudo-nodes in the catenet. They are different from gateway nodes in that they don't report any connectivity information and in that connectivity between neighbor gateways should not be deduced from the fact that both gateways are attached to the same net. (Or if connectivity between them is deduced that way, delay or cost should not be decided by adding the delay from each gateway to the attached net.)

This can be accomplished by ordering the gateways as nodes 1 to n, and ordering the pseudo-nodes as nodes n+1 to n+m in the connectivity matrix (assuming there are n gateways and m nets). When calculating the distance matrix, ignore the pseudo-nodes (only calculate according to the n X n matrix). Then square the (n+m) X (n+m) matrix consisting of the n X n distance matrix and the connectivity information of links ending in pseudo-nodes once. (Actually only entries involving a pseudo-node need be calculated.) This assures that links ending in pseudo-nodes will not influence the calculation of the distance matrix.

1.12 Reliable Transmission of Link-State Packets

If a mechanism were provided to avoid loss of link-state packets, neighbors would remain up-to-date automatically. This could eliminate the need for sequence-numbers packets.

A reliable transmission scheme would require at least twice as much traffic, since every packet would need to be acknowledged, and would require gateways to constantly try to deliver the packets over failed links. This might be very desirable, however, since a recovered link would be quickly diagnosed.

Even if the implemented scheme did not require reliable delivery, gateways communicating over lossy networks would presumably require acknowledgments for each packet.

2.1 Introduction

This algorithm is designed for two purposes. One is to route traffic more efficiently throughout the catenet by "load splitting", i.e., not sending all traffic on one path. The other purpose is to give traffic sources feedback on when they are sending too much traffic so they can quench themselves. Like the "ROUTING AROUND FAILED GATEWAYS" algorithm, this is not an algorithm that produces the theoretically optimal traffic flow, but rather one that produces a reasonable traffic flow and is designed to be simple.

This algorithm builds on the "ROUTING AROUND FAILED GATEWAYS" algorithm. So we assume gateways are exchanging connectivity information and computing a connectivity matrix and a *'d connectivity matrix (the repeatedly squared result of the connectivity matrix which gives a matrix of minimum distances between nodes). I will refer to the connectivity matrix as CONN, and the *'d connectivity matrix as *CONN*. Likewise I will refer to a reliability matrix as REL and the *'d result as *REL*.

The basic idea of this algorithm is that reliability information is passed around with connectivity information. A gateway, instead of sending ALL its traffic for a destination to ONE of its neighbors that is the minimum distance from the destination, splits its traffic to the destination among the set of neighbors that are the minimum distance from the destination, in a proportion so as to optimize the reliability of the route to

the destination. All traffic to the destination is sent along the most reliable route until (and if) that amount of traffic causes the reliability of that route to drop below another route, at which time a small amount of traffic is diverted to the alternate route. In the steady state, traffic will be split among paths of approximately equal reliability.

The reason for using reliability in this algorithm is that it is an easily ascertainable number. Gateways already keep track of the percentage of packets they have had to drop. Assuming that most packets are lost because the gateway had to drop them, this is an adequate measure. If most packets were lost in the net between gateways, the gateways would probably require acknowledgments, so reliability could then be measured by the percentage of packets that got acknowledged.

It would be possible to use marginal delay rather than number of hops in the distance measure in the algorithm described in the first part of this paper, which would result in the same distribution of traffic as would result from the algorithm described in Strazisar's paper, and would make the algorithm described in this part of the paper unnecessary. However, marginal delay (or delay) is much harder to measure and is less likely to be immediately implementable. In addition, this scheme is less sensitive to incomplete information throughout the catenet. Since the number of hops in routes is liable to be a very constant number (as opposed to delay), slow or unreliable propagation of information will not be a serious problem.

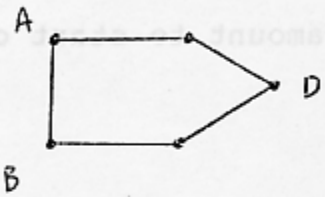
2.2 The algorithm

In addition to merely bouncing packets off its neighbors to determine connectivity, a gateway keeps statistics on the percentage of traffic that gets through to the neighbor. Instead of just reporting "the link between me and neighbor A is up", the gateway should report "the link between me and neighbor A is 92% up". If there is very little traffic between 2 gateways, but one packet is sent and it gets there, it is still reasonable to assume the path is 100% reliable (a low traffic path is desirable).

Now gateways can construct two matrices, a connectivity matrix, CONN (with entries of infinity for neighbors less than some reliability threshold apart) and a reliability matrix, REL.

If two neighbors report a link as having very different reliabilities, both numbers can be stored in the matrix REL, since entry (A,B) can mean the reliability of the link between A and B as reported by A, and entry (B,A) can mean the reliability of the link as reported by B. The reason in CONN that we assume a link to be down if either end declares it down is that it is the safe thing to do -- if one end thinks the link is down, something is wrong with the link and it is safer not to use it. If what is wrong with the link is that one of the gateways died, you will not get a report from the dead end that the link is no longer operational. Connectivity information would have to get timed out, adding needless complication when the simple ploy of declaring a link down unless both neighbors declare it up works.

Gateways still compute *CONN* for the purpose of determining the set of neighbors which are the minimum distance from the destination. Traffic will be split among this set of neighbors. The reason we restrict the next hop to this set is to avoid loops. An example of a loop that could otherwise form is the following:



Node A sees two paths to D, the direct one of length 2 and the one through B of length 3. B sees the same. If we allowed gateways to send traffic over a path of nonminimal length (without complicating the algorithm unreasonably or using a trick we have not seen) A would send some of its traffic destined to D through B and B would send some of its traffic to D through A, so a packet to D could get bounced back and forth between A and B indefinitely. This is not desirable.

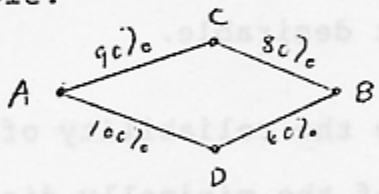
Then gateways compute the reliability of the path to the destination through each of the minimally distanced neighbors. (This will be explained in the next section; for now assume that number is known.) The gateway keeps a matrix of fractions, with entry (i,j) being the fraction of traffic destined to node j that the gateway sends to node i as the next hop. For each of the neighbors in the minimally distanced set the gateway increases by a little the traffic it sends to the neighbors with highest reliability to the destination, and decreases by a little the fraction of traffic it sends to the neighbors with lowest

reliability. The purpose of this is to maximize the reliability of the path from that gateway to the destination.

The amount to adjust the fractions by on each update cycle can be a parameter. If it is too small, this scheme will take too long to adjust to changes. If it is too large it will cause "thrashing". A reasonable amount to start out with is probably on the order of 10%.

2.3 Computing *REL*

Except for one problem which we'll fix up in the next paragraph, *REL* is computed from REL in the same way as *CONN* is computed from CONN (cf. section 1.6), repeatedly squaring the matrix until the same matrix stabilizes. Except instead of using the operations MIN and PLUS, use MAX and TIMES. To see this consider the example:



The reliability of the path between A and B is 90% times 80% if it goes through C and it is 100% times 60% if it goes through D. Thus the reliability of the path between A and B is MAX(72%,60%), since you can assume traffic is being sent along the most reliable path.

The problem is there might be some really long path using obscure nodes that has 100% reliability along each hop, so this method would tell you the reliability between A and B was 100% even though that would be false since A would never choose that

path. Thus we have to restrict the set over which we take MAX in squaring the matrix to minimal distanced paths. We can do this by computing both matrices simultaneously, and when we take MIN for a row in the CONN matrix, we use only those indices in the row equal to the MIN in considering the MAX in the REL matrix.

2.4 Measuring Reliability

Gateway reliability is a function of both the gateways' reliability and the reliability of the networks which the packet traverses. It is assumed that the network specific code in each gateway will guarantee a reasonable level of reliability for transmission of packets across that network. For some networks, such as the Arpanet, which provide highly reliable communications paths, the network specific code may be very simple. In other networks, such as the Packet Radio Net, a retransmission and acknowledgement scheme may be necessary to improve reliability on the network.

If an acknowledgement scheme is used for purposes of improving reliability on a network, then the percentage of packets acknowledged may be used by the gateway to measure reliability of the network. If such a mechanism is not used (presumably because the network is reliable), then it is reasonable to assume a fixed reliability for this network.

The reliability of the gateways is also a concern. The major source of unreliability in the gateways is the limitation on the gateway's resources. When the gateway's buffer pool is

exhausted, the gateway will drop packets it can no longer store and forward. Gateway reliability can be measured by counting the number of packets received versus the number of packets dropped due to lack of resources. (This is currently done in the gateways.)

Thus, the reliability figure computed by a gateway in the procedure outlined above is a function of both that gateway's reliability and the reliability of the network about which the gateway is reporting.

2.5 Source Quenching

If the reliability of a gateway's path to the destination drops below some threshold, and the gateway is the closest gateway to the source (it is on the same net as the source), the gateway can send a message to the source to cut back on the amount of traffic it is sending.

There could be cases in which this would not be appropriate, for instance in a net that was lossy for reasons other than too much traffic, but the gateway will know the characteristics of its attached nets and can act accordingly. Also, the message sent to the source is merely a warning and an opportunity to obtain feedback. The source can act on it however it thinks appropriate -- by ignoring the message, by stopping transmission altogether, or by cutting back just a small amount, awaiting further warning messages.

The format of the warning message has yet to be defined.

3.1 Traffic

When connectivity changes, a gateway sends a link-state packet. These link-state packets get reported once by each gateway to each of its neighbors. Thus if k is the average number of neighbors per gateway in the catenet, each connectivity change generates (k)(n) packets, where n is the number of gateways. Link-state packets are also reported periodically, so this results in another (k)(n)(n) packets every update interval.

Sequence-numbers packets need not be transmitted throughout the catenet. Thus each gateway will periodically send a packet to each of its neighbors, resulting again in (k)(n) packets.

Adding on the second part of the algorithm (the flow control part) does not increase the number of packets since the only change in information exchange is reporting the quality of links rather than just their existence. It could cause more traffic because the quality of links changes more quickly than their existence, but this should be minimized by using the reliability numbers as approximations, and only reporting changes above a certain threshold.

3.2 Computation time

The only part of this algorithm that takes significant computation time is the operation of *'ing the matrices REL and CONN. Each squaring step takes on the order of (n)(n)(n)

operations (since the matrices are n X n). It is never necessary to square the matrices more than log n (base 2) times. (Doing it j times tells you about paths of length 2^j or less and no paths are of length greater than n, since that would imply visiting a gateway twice, i.e., having a loop.) Thus *'ing the matrices is on the order of (n)(n)(n)(log n) steps. In a catenet with 30 gateways, and with the computer taking about 5 microseconds for each step, this will result in taking about .7 seconds. This need only be done when the matrix changes significantly.

If the .7 seconds seems unreasonably long there are many ways of making things faster. One is to notice that in the matrix REL, multiplications are only done to compare relative reliabilities of minimal length paths. Therefore for each entry (i,j) in *REL*, there are on the average only as many multiplies as there are different shortest length paths between i and j. This will be significantly less than n, and the .7 seconds was based on a factor of n. More likely than the number 30 for a 30 gateway catenet would be the number 3, so the estimate of .7 can really be as little as .07 seconds.

Another savings is to note that numbers in CONN never decrease twice. (This is assuming the distance function is the number of hops.) Thus on a given squaring step, you need only compute entries equal to infinity. Since entries in REL are computed in parallel with CONN, this reduces the computation time on REL also. Since the entries in *REL* are only computed when minimal length paths are found, and not on each pass, that reduces the estimate on the amount of work per entry in *REL* by a factor of log n.

3.3 Storage

A matrix of fractions must be kept, where the entry (i,j) is the fraction of traffic destined to node j that should be sent to neighbor i. The fractions need only be 8 bits or less, so this matrix could be stored in (1/2)(n)(k) 16-bit words, where k is the number of neighbors the gateway has.

The other matrices are each n X n. Each entry can certainly fit into an 8-bit byte, or less if storage is really a problem. There are four relevant matrices, CONN, *CONN*, REL, and *REL*. (4)(1/2)(n)(n) = (2)(n)(n) is about 2000 16-bit words for a 30 gateway catenet.

Note: It is not necessary to have a "scratch" matrix to store intermediate results while squaring. Overwriting entries in the matrix will not change the result. (Though if you want to use the trick that entries in CONN only decrease once, it is necessary to not use updated entries until you complete a pass.)

It is necessary to also store the latest link-state packet received from each gateway. These packets contain approximately 2 16-bit words for each of the gateway's neighbors (gateway number and link state). Thus storage required to store these packets is about (2)(n)(k) words, where k is the average number of neighbors per gateway.