Fault-Tolerant Broadcast of Routing Information

Radia Perlman
Digital Equipment Corp., 1925 Andover Street, Tewksbury, MA 01876, USA

An algorithm is presented for the reliable broadcast of routing information throughout a network. The algorithm anticipates the possibility of long-delayed packets, line and node outages, network partitions, hardware failures, and a history of arbitrarily corrupted databases throughout the network. After any failure, the algorithm stabilizes in reasonable time without human intervention, once any malfunctioning equipment is repaired or disconnected. The algorithm also has the advantages of not requiring frequent control traffic in the absence of topological changes, not imposing artificial delays on nodes upon startup, and not relying on timers in ordinary operation. The algorithm is compared to a functionally similar algorithm in the ARPANET.

Keywords: Broadcast, Fault-tolerant, Routing, ARPANET, Synchronization.

Radia Perlman is a Network Architect for Digital Equipment Corporation, working on design of network protocols and algorithms for DECNET. She was formerly at Bolt Beranek and Newman, working on design and implementation of protocols and algorithms for the ARPA packet radio net, the ARPA internet, the ARPA satellite net, and the INTELPOST international facsimile mail network. She received the S.M. and S.B. degrees in mathematics from the Massachusetts Institute of Technology.

North-Holland
Computer Networks 7 (1983) 395–405

1. Introduction

This paper assumes the kind of routing scheme (such as currently in operation in the ARPANET) in which each node in the network ascertains the state of the links to its neighbors, and reports that information in a Link State Packet, which is broadcast to all the other nodes in the network. Each node in the network keeps in its database the latest Link State Packet from every node in the network. Thus each node's database contains a complete map of the network. With this complete map, a node can compute routes using an algorithm such as Dijkstra's algorithm [1]. Correct routing requires that all nodes agree on the map of the network.

This paper uses the ARPANET routing broadcast scheme as a comparison, and shows how certain modifications to that scheme can yield improvements of self-stabilization, decreased overhead, and higher reliability.

The principles behind the scheme in this paper are:

1. After any failure, including arbitrarily corrupted databases due to unknown or even malicious causes, the algorithm should stabilize in reasonable time to correct routes, without human intervention, once any malfunctioning equipment is repaired or disconnected.

Note that it is impossible to design an algorithm that functions correctly in the presence of continuous malfunction, because that implies an algorithm that functions correctly even if the algorithm is changed into a different algorithm. Thus, it is possible for a node to fail in such a way as to require human intervention to discover and disconnect the node. However, it is still important to design an algorithm that self-stabilizes once malfunctions cease. Otherwise, an intruder could inject a few well-chosen packets into a net and bring the net down long after he has left the scene. In the case of malfunctions, it is often the case (as in the ARPANET bug related in [3]) that malfunctioning equipment will crash completely soon after

acting erratically. Thus even though the algorithm cannot function during the presence of malice or certain types of malfunctions, an algorithm that recovers by itself once malfunctions cease is much more likely not to require human intervention.

2. "Low probability" events cannot be ignored. In a network, low probability events do occur. The ARPANET incident described in [3] was due to low probability events. This failure took $1\frac{1}{2}$ years of operation to occur in the ARPANET. However, the ARPANET is only a single network. With a design that is used in a network product, many networks will be running with the design. Any low-probability events that can cause disasters are much more likely to occur, of course, the longer a design is in operation, and the more places in which it is operating.

Low probability events should not cause permanent catastrophe, though it is tolerable for the network to require some amount of time to automatically recover from a low probability event.

3. Timers should be avoided, if possible, since they can be set incorrectly. Although the timer values might work at first, they can fail years later when the network has evolved with more nodes, different speed lines, etc. Debugging occasional timing problems that would then ensue would be very difficult, especially if enough time has elapsed that the network designers are no longer available.

4. Timers which cannot be eliminated from the design should have a large safe space from which to be chosen, so that the design can be exported to a network with different characteristics, so that the design will continue to work as the network evolves, and so that skews in local clocks will not impact performance.

5. Correct operation should not be delayed for timeout periods, except after a low probability event. For example, nodes should not be artificially constrained to wait a timeout interval upon restart before returning to operation.

6. The algorithm should minimize control traffic.

The algorithm does not assume the existence of special hardware such as globally synchronized clocks, or local clocks with battery backup.

In the section "Basic Scheme", we present the general idea behind the broadcast scheme. The basic scheme applies both to the ARPANET design, and to the design recommended in this paper. In the section "Design Details", we present the ARPANET scheme in detail, and describe modifications to that design to yield increased robustness, flexibility, and efficiency.

2. Basic Scheme

2.1. Propagation of Link State Packets

Propagation of routing information should not depend upon routing in a network being correct, for obvious reasons. A simple, fairly efficient method of distributing routing information is by "intelligent flooding". "Flooding" means that a node broadcasts a Link State Packet on all its links except for the one on which it was received. "Intelligent flooding" means that the node recognizes duplicates, and does not flood a packet unless it is a new packet.

With intelligent flooding, assuming no need for retransmissions, each packet will traverse each link at most twice, once in each direction. (Usually each packet will traverse each link once, but a packet can traverse a link between A and B twice if A sends the packet to B while B's transmission to A is in progress.)

Intelligent flooding is used in the current ARPANET scheme, and it is used in the scheme presented in this paper. However, the most difficult part of the scheme is determining whether a received packet is older or newer than the packet stored in the database.

2.2. Recognizing Most Recent Information

2.2.1. Globally Synchronized Clocks.

In a network with globally synchronized clocks, Link State Packets could be timestamped by the source upon generation. Then it would be easy to compare a received Link State Packet with a previously received Link State Packet stored in the database, and the one with the later timestamp would be the more recent packet.

The timestamp will be a finite length field, so eventually it will wrap around (unless it is many bits long). However, each node can periodically scan its database for Link State Packets that are very old, and purge them long before a new timestamp could look old because of wrap-around.

Global clocks can also safeguard against faulty timestamps. If a node (due to hardware fault or other causes) issued a Link State Packet with an

incorrect timestamp indicating the packet was created far in the future, the other nodes would be able to detect the problem and prevent such a Link State Packet from being propagated.

Most networks do not, however, have globally synchronized clocks.

2.2.2. Local Clocks.

If each node in the network had a local clock that was guaranteed to be monotonically increasing, even when the node was down, Link State Packets could be timestamped by the source with the local clock.

However, given that the timestamp is a finite length field, a node could be down for long enough for the timestamp to wrap around, so that when the node came up again, its new Link State Packets could look old. With local timestamps (as opposed to globally synchronized clocks), the timestamp alone would not be enough information for a distant node to determine how long ago the packet was generated, because the timestamp has meaning only to the source node.

The local timestamp could be a large enough field so that for all practical purposes it would never wrap around. However, a hardware fault or data corruption at the source node S could cause the high order bits to be set in S's timestamp. Also, a hardware fault, malice, or data corruption in a distant node could cause a faulty timestamp for S, with the high order bits set, to be propagated throughout the net. With a global clock, this would be easily detected and corrected, but this is not easily done with local clocks. Therefore, the possibility of wrap-around must be considered no matter how large the field. Thus local clocks are not much of a simplification over sequence numbers (to be described below), and require hardware that is not always available in packet switches. Thus a sequence number scheme, as described below, is assumed for the rest of the paper.

2.2.3. Sequence Numbers

Each node A maintains a counter (known as a sequence number) and increments the sequence number each time it generates a Link State Packet, marking the Link State Packet with the sequence number. When distant node B receives a Link State Packet from A, B compares the sequence number with the one it has stored in its database from the last Link State Packet received from A.

Assuming the sequence number field is large enough (e.g., 64 bits at one message/millisecond would take > 500 million years to wrap around), wrap-around would not occur under normal circumstances. However a hardware fault or other data corruption could set the high order bits. Also, a node (possibly maliciously) could inject a Link State Packet into the network with source ID A and a sequence number with the high order bits set.

Thus it is possible for the field to get incremented to the maximum value. When the field does increment to the maximum value, something must be done. Basically there are two choices:

1. attempt to reset, or
2. wrap around.

Resetting is very difficult. Assume some sort of "reset" packet. The reset packets must be flooded somehow to ensure that all nodes are reset. Old duplicate reset packets that might emerge after a reset would cause a whole new reset operation. And worse yet, a node that was isolated from the net at the time of the reset would still have the high value for the sequence number, so there is no way to ensure that the reset will reach all nodes.

Thus we assume nodes cannot make a simple arithmetic comparison of sequence numbers, but must treat the sequence number space as a circular space.

In a circular sequence number space, assuming the sequence number space to be of size n, and numbers a and b are being compared, the ordering LT (less than) is defined by:

a LT b if $a < b$, and $b - a < n/2$,

or

$a > b$, and $a - b > n/2$.

There are problems with this simple scheme.

1. Node A goes down and comes back up. Since A does not know which sequence number it was using before it crashed, it might choose a sequence number that looked old compared to the sequence number it was using before the crash. Then A's packets will not be believed until A's sequence number increments past the old sequence numbers issued before the crash.

2. The net partitions. Suppose the network partitioned into East and West, with node A a member of West. East would not receive any Link State Packets from A during the partition, and during that time A could issue $n/2$ Link State Packets, received by the nodes in West. When the network

reformed, the new Link State Packets from A would look old to the nodes in East.

3. A sequence number is corrupted by another node. Some node D in the network might, due to a hardware problem, corrupt the sequence number on A's Link State Packet, causing it to be greater than the true value, and then to nodes downstream from D relative to A, A's Link State Packets would look old, until A's sequence number incremented past D's corrupted value. This type of error might not be detected by Data Link Layer CRC, because it might have occurred in D's memory, and not during transmission of the packet.

4. A sequence number is corrupted by the source node. Node A, due to hardware problems, might issue Link State packets for itself with arbitrary sequence numbers. This problem actually occurred in the ARPANET [3]. The result was that after A was repaired there was no value A could choose for a sequence number that would look new to all nodes in the network.

Thus sequence numbers do not give sufficient information to determine which of two packets is newer. It is necessary to add an age field to the Link State Packet. The purpose of the age field is to give the network enough information to purge an old Link State Packet before the source node's sequence number can wrap around, and to ensure that any Link State Packet will eventually cease to exist.

3. Design Details

3.1. The ARPANET Approach

3.1.1. Age Field Rules

1. When a Link State Packet is created, the age field in the packet is set to a max value, MAX-AGE, by the source node S.

2. When S's Link State Packet is accepted by another node, T, T copies the age field from the received packet, and decrements it according to holding time at T. The clock resolution in the ARPANET is one tick every 8 seconds. Thus the age field is decremented by T after the packet has been held for 8 seconds.

3. When the age decrements to 0, the age field is no longer decremented, and a packet with an age field of 0 is considered to be "too old", and no longer propagated, although it is not purged from the database. A Link State Packet that has a nonzero age is always considered newer than a Link State Packet with a zero age, regardless of sequence number.

4. A source node must issue a new Link State Packet within MAX-INT after the previous one.

5. A node must wait an interval RESTART-TIME, for its old Link State Packets to expire, before it can restart. The parameters are:

1. MAX-AGE, the amount of time a Link State Packet is considered valid since it was created by the source node (64 seconds, for the ARPANET), and

2. MIN-INT, the minimum interval allowed between creation of Link State Packets by the source node.

3. n, the size of the sequence number space. Since it must not be possible for a source to issue Link State Packet numbered $K + (n/2)$ before Link State Packet $\#K$ has expired, $n/2 >$ MAX-AGE/MIN-INT.

4. MAX-INT, the maximum interval allowed between creation of Link State Packets by the source node (60 seconds, for the ARPANET). MAX-INT must be less than [MAX-AGE minus the propagation time required for a Link State Packet from the source to reach all nodes in the network], so that the new Link State Packet will reach all nodes before the old one expires. (Once A's packet expires, nodes no longer exchange A's information, so new nodes will have incomplete databases until A issues a new Link State Packet.)

5. RESTART-TIME, the interval a node must wait after restarting, in a dormant state, waiting for its old Link State Packets to expire. RESTART-TIME must be greater than MAX-AGE. In the ARPANET, RESTART-TIME is set at 90 seconds.

3.1.2. Flooding Rules

1. If the received packet is newer than the stored packet, flood.
2. If the received packet is the same as the stored packet, treat it as an ACK and then discard it.
3. If the received packet is older than the stored packet, discard it.

3.2. Problems And Fixes

In this section we describe potential problems in the ARPANET design, and give modifications

to the ARPANET scheme that would enhance its robustness and efficiency.

3.2.1. Immortal Packets

One problem with the ARPANET scheme described above is that it does not ensure that packets will age. If the clock resolution is coarse (8 seconds in the case of the ARPANET), and a packet is held for less than a clock period before being transmitted to the next node, the next node will receive the packet without the packet having been aged at all. This can happen over many hops, and the error can accumulate. Even if the clock resolution is not coarse, the age can be arbitrarily in error if the packet loops, being held a negligible time in each node, but being transmitted indefinitely between nodes.

The looping problem actually occurred in the ARPANET [3]. A source node, A, due to hardware problems, issued Link State Packets with three sequence numbers all around the sequence number circle, say 0, $n/3$ and $2n/3$. Since the circular sequence number space is not well-ordered (there is no smallest number), $0 < n/3 < 2n/3 < 0$. A node X, having A's Link State Packet #0 stored, would accept the Link State Packet #$n/3$ as newer (and flood it to its neighbors), and subsequently accept Link State Packet #$2n/3$ as newer, and subsequently accept Link State Packet #0 as newer, ... Thus all three packets circulated indefinitely, and without increasing their ages, since the holding time at each node was shorter than the clock resolution.

The solution is to *require* packets to age. If the clock resolution is sufficiently fine, it can be required that every node increase the age. Otherwise, if the clock resolution is so coarse that it is unreasonable to require the packet to age at each hop, each node can decrement the age field with pseudorandomly generated probability x, where x is the fractional number of units the packet was held.

If the age field in the ARPANET had been forced to decrease in this way, the packets would have died out by themselves without human intervention, especially since the malfunctioning hardware failed completely soon after generating the offending packets. Instead, human intervention with a lot of effort by people very familiar with the implementation was required to fix the problem, even after it was diagnosed.

3.2.2. Parameter Settings

Another possible problem with the ARPANET aging mechanism is the criticality of parameter settings. MAX-AGE must be enough larger than MAX-INT to ensure that the new Link State Packet will reach all reachable nodes before the old packet expires. Propagation time across a hop can be much longer than expected because of long queues, contention for shared (e.g., CSMA) links, or if a packet must be retransmitted several times because of transmission errors. Also, a packet might age more quickly than the source expects, due to a distant node having a faster-running clock. Thus [MAX-AGE minus MAX-INT] must be fairly large.

If MAX-INT is set very small, the overhead of having every node issuing Link State Packets so frequently might be excessive. This overhead becomes more severe as the network grows in size. Thus it is desirable for MAX-INT to be as large as possible.

If MAX-AGE is set very large, to give more room for setting the MAX-INT parameter, it will take longer to purge the net of old Link State Packets. This can be a problem in the ARPANET scheme because a node is not allowed to restart until all its old Link State Packets have timed out. Thus in the ARPANET scheme, since RESTART-TIME must be greater than MAX-AGE, MAX-AGE must be kept as small as possible.

Thus the goals are:

1. Minimize MAX-AGE (to enable nodes to restart in reasonable time).
2. Maximize [MAX-AGE minus MAX-INT] (to prevent routing disruption if a Link State Packet expires before the next one reaches all parts of the net).
3. Maximize MAX-INT (to cut down on control traffic overhead).

These are basically conflicting goals, making the parameter settings very critical, if there even are settings that are reasonable. Although there may be parameter settings that are satisfactory for the current ARPANET, the tight bounds and interrelationships of the various timers make it difficult or impossible to export the design to a network with different characteristics, and might even pose a problem as the ARPANET evolves.

The solution is to remove the restriction (to be described in the section "RESTART-TIME Elimination") that nodes must remain dormant for

RESTART-TIME after restarting.

Given that nodes do not have to wait RESTART-TIME, MAX-AGE now is needed just to recover from low-probability events, so it can be set to the maximum amount of time considered tolerable for the net to recover from a low-probability event. For most networks, a value on the order of an hour is quite reasonable for this purpose.

Given that the MAX-AGE parameter is much longer, the MAX-INT interval can be set much longer as well. A value of half MAX-AGE is very safe and reasonable, so MAX-INT can be on the order of a half-hour, as contrasted with being on the order of a minute under the ARPANET scheme.

Thus the amount of control traffic is vastly reduced, due to increasing MAX-INT, and there is a large safe space for setting the two parameters MAX-AGE and MAX-INT.

3.2.3. Premature Aging

Another potential problem is that a node D could corrupt the age field in S's Link State Packet, or age the packet incorrectly so that the age is almost zero. Then S's packet might not make it through the entire network before it expires. Then the network can disagree about the contents of S's packet until S issues a new Link State Packet, and routing can be incorrect during that time.

Also, since the aging mechanism is only an estimate, a large skew can develop among the estimated ages of a Link State Packet in different parts of the net.

Any problem that might occur because of different portions of the net disagreeing about when a Link State Packet has expired will be rectified once the source issues a new Link State Packet. However, it is safer and less timer-dependent to ensure that the nodes synchronize the event of packet expiration.

Synchronization can be accomplished with the same flooding mechanism used for propagation of routing information. When node D notices that its copy of S's Link State Packet has aged to zero, D refloods the packet, with age zero, to D's neighbors. If a node N receives S's Link State Packet with zero age, which has the same sequence number as the stored packet from S, N treats the received packet as a "new" packet, allows it to supersede the stored copy, and N floods the information to its neighbors. Thus the entire net will be forced to age the packet to zero virtually simultaneously. If S is indeed alive and well, it will discover that the network is impatient for new information (by receipt of its own zero-aged packet), and S will issue a new Link State Packet.

3.2.4. Old Packets

Another potential problem is that old Link State Packets are kept in the database, but not propagated. Since this information is not exchanged, nodes A and B can have different information in their databases about node S. This condition will remain until a new Link State Packet is received from S, which may not occur because S might be unreachable. If the information is used in routing calculation, incorrect routing can result. This will usually not be a problem because if a new Link State Packet has not been received from S, S is probably not reachable and its Link State Packet will not affect the routing. However, it is more consistent to either declare the information useless and erase it, or propagate it to ensure that all nodes agree.

An important reason for erasing the information is that a node, due to hardware fault or even malice, can issue Link State Packets with random source IDs. Unless some mechanism exists to purge bad data, these Link State Packets would remain forever, cluttering up the databases of the other nodes.

Expiration synchronization as described above requires a node D to hold onto an expired packet until it has successfully flooded the packet to its neighbors. However, it is safe for D to discard the packet after MAX-AGE time units after D decided the packet expired.

Thus the expiration rules are:

1. If the age field on S's Link State Packet expires while in memory, flood the packet, with age field zero, to all neighbors.
2. If S's Link State Packet is received with age zero, then:
    1. If the sequence number matches the sequence number of the stored and unexpired Link State Packet for S, accept the expired copy and flood it.
    2. If the sequence number does not match the sequence number of the stored Link State Packet, or if there is no Link State Packet stored for S, ignore the received Link State Packet

3. If MAX-AGE has elapsed since the packet expired in memory, or since an expired copy of the packet was accepted from a neighbor, discard the packet.

3.2.5. Negative Ages

Another potential problem is that if MAX-AGE is considerably less than the maximum value that can fit into the age field, a source, due to hardware error or other causes, could issue a bad Link State Packet with a value in the age field much larger than MAX-AGE. Then when the source was repaired, its old, bad Link State Packets would not time out within MAX-AGE. There are numerous methods of preventing this from happening, given that the problem is recognized. One method (used in the ARPANET) is to mandate that MAX-AGE be the largest value that can physically be expressed in the field.

3.2.6. RESTART-TIME Elimination

In the ARPANET, a node must wait RESTART-TIME upon restarting. Three modifications to the ARPANET scheme eliminate this restriction:

1. send a stored Link State Packet to neighbor N in response to receipt of an older Link State Packet received from N;
2. use a lollipop-shaped sequence number space instead of a circular space;
3. have a quick method of obtaining all current information upon restart.

First we discuss the response to an older Link State Packet. If a node restarts, and chooses a sequence number that looks older than the sequence number it had been using before, its new packets will not be believed until the old ones time out. The rule in the ARPANET is that a node must wait, upon restarting, the amount of time it takes for all old packets it had issued to time out before it is allowed to issue new packets.

This approach has the consequence that a node is forced to wait some amount of time before reentering the network. On the ARPANET, it was assumed (falsely) that after 90 seconds all old packets would have timed out. Even if the ARPANET design were fixed by forcing the age to increase at each hop as suggested (so that it could be assumed that all old packets would die out in some finite time), this solution would still be unacceptable in many networks because of the large amount of time that would be necessary to wait upon restart. In the ARPANET, with a relatively small diameter and low delay lines, the amount of time might be tolerable, but in other networks the amount of time could easily be on the order of a half hour. [As explained above, RESTART-TIME > MAX-AGE > (MAX-INT + Propagation Time + Maximum Possible Aging Error). If Propagation Time and/or Aging Error is large, and/or if the network cannot tolerate a small value for MAX-INT, then RESTART-TIME must be large.]

A method of eliminating the RESTART-TIME restriction is to modify the third flooding rule to:

3. If the received packet is older than the stored packet, send the stored packet back across the link upon which the older packet was received.

In this way, if node A is using a sequence number for itself that looks old to any portion of the net, A will be informed of the sequence number that the rest of the network believes. Then A should (sometimes) switch to the newer-looking sequence number.

We must guard against the case where, due to past history, three sequence numbers j, k, and l are extant in the network, with $j < k < l < j$. If A always switched to a newer-looking sequence number every time A heard on one, A would switch between j, k, l, j,..., and the numbers would never age out because A would be constantly issuing new packets with each sequence number. Node A must never issue sequence numbers differing by $n/2$ or more within MAX-AGE time. Thus within a time period of MAX-AGE, A may make jumps in the sequence number space totaling a parameter J, set to at most $n/2$ minus the number of Link State Packets A can issue within MAX-AGE. Thus $J < n/2 - (\text{MAX-AGE}/\text{MIN-INT})$.

Thus no matter what the previous history is, all A's old Link State Packets will die out within MAX-AGE, and A's real Link State Packets will then be believed.

And in the case that a previous untimed-out Link State Packet has sequence number s, which is closer than J to that which A is using, A can immediately switch to sequence number $s + 1$ and have its Link State Packets believed.

Next we discuss a lollipop-shaped sequence number space. With a circular sequence number space, there is no sequence number less than all other sequence numbers. Suppose there were a sequence number, s, less than all others. Then,

when a node A restarted, it could use sequence number s, and if any previously issued sequence number t was extant in the network, A would, by the scheme above, hear about it. A could safely switch to using sequence number $t+1$, and the packet it had previously issued with sequence number s would not be mistaken for more recent since s was assumed to be less than any other sequence number.

However, A might have time to issue several Link State Packets before hearing that it had been previously using sequence number t. The solution is a lollipop-shaped sequence number space, where sequence numbers start at some value, say minus k, and increment up to 0, and proceed from there to a circular space of non-negative numbers. The value k must be large enough so that A could not possibly issue k Link State Packets before any old Link State Packets it might have issued previously would age out. In other words, k must be more than the number of Link State Packets A can issue during MAX-AGE ($k >$ MAX-AGE/MIN-INT).

The rules for sequence numbers in a lollipop shaped space are: Sequence numbers start at some value, say $-k$, and proceed to a circular space where numbers vary between 0 and n. Sequence number a LT b if and only if:

1. $a < 0$ and $a < b$, or
2. $a > 0$, $a < b$, and $b - a < n/2$, or
3. $a > 0$, $b > 0$, $a > b$, and $a - b > n/2$.

Using a lollipop shaped sequence number space has the advantage that A can start with sequence number $-k$, and if A hears any other sequence number for itself A can immediately switch to that. The restriction in jump to J pertains only to the circular portion of the sequence number space. Thus the lollipop space makes it much less likely that a problem would occur in which A would have to wait for MAX-AGE to elapse, for its old Link State Packets to time out, before its new ones would be believed.

In fact, the only events which could cause A to have to wait MAX-AGE are:

A. Some node, due to incorrect operation, in the past issued Link State Packets with source ID A, and sequence numbers all around the circular space.

B. An absurd sequence of carefully timed restarts by A and network partitions occurred. An example:

1. The network partitions East-West after A issues sequence #0, with A in East.
2. A issues b more Link State Packets before MAX-AGE has elapsed, so that East has sequence #b for A, and West has sequence #0 for A, with no packets yet expired.
3. A restarts and the network partition ends at about the same time.
4. A is informed of West's sequence #0 first, and issues a Link State Packet with sequence 1.
5. The network again partitions.
6. A is then informed of the sequence #b, and issues a Link State Packet with #$b+1$, and subsequently issues c more Link State Packets before MAX-AGE has elapsed. Thus East will have # $b+c+1$ for A, and West will have #1 for A.

If this sequence of events repeats, the gap between sequence numbers on unexpired packets can increase arbitrarily. Anyway, both these events are very low-probability events.

Next we discuss obtaining current information upon restart. When a node restarts, it needs to reacquire the Link State Packets it had received and acknowledged before it crashed.

The ARPANET addresses this problem by requiring all nodes to issue Link State Packets fairly frequently (every 60 seconds). When a node restarts, it first waits 90 seconds, with the assumption that during that time it will hear Link State Packets from all other nodes in the network.

However, for some networks, the overhead of artificially requiring each node to frequently issue Link State Packets, even if their contents have not changed, may be excessive.

A simple solution is to have a special packet that a node can send to a neighbor that means, "Assume I have not acknowledged any Link State Packets, so send them all to me." Then a node will not need to wait upon restart, and nodes will not be required to send Link State Packets at such a frequent rate just to solve the restart problem.

An alternative to a special packet is to have node A mark node B as not having acknowledged any packets when the line to B goes down. Then A will automatically send all packets to B when the line comes back up. The only disadvantage of this scheme over the special packet scheme is that if it was really the line to B that went down, and not node B itself, then it would be unnecessary for A to resend all the information to B, that B had previously acknowledged, when the line came back up.

When the line from A to B comes up, it will be clogged for a time CLOG-TIME with routing messages while A sends one Link State Packet for each node in the network to B. This is not a problem because:

1. CLOG-TIME (the amount of time to send s packets, where s is the number of nodes in the net) will be orders of magnitude less than the amount of time a restarting node would be required to wait under the RESTART-TIME scheme. Depending on s and the speed of the link, CLOG-TIME should be at most on the order of a few seconds.

2. The link from A to B was down for some duration which would probably be orders of magnitude longer than CLOG-TIME, so the clog of routing messages upon restart of a link would go unnoticed.

3. CLOG-TIME is much less than the amount of time it takes for a Link State Packet to propagate throughout the entire net. Any traffic traversing link A-B is likely to encounter routing loops until the Link State Packets issued by A and B notifying the net of the newly recovered link reach all parts of the net. Thus it is probably desirable, and certainly not undesirable, for traffic to get held up for CLOG-TIME when the link recovers.

4. When a link recovers, it is desirable to wait some amount of time to ensure it stays up. CLOG-TIME is probably very close to that amount of time. It is beneficial to exercise the link, by sending lots of packets when it first comes up, before issuing a Link State Packet to the rest of the net declaring the link up.

3.2.7. Restarts

There is still a problem associated with restarting of a node. For a period of time MAX-AGE after node A restarts, there is the possibility that there are still packets around the network, with age less than MAX-AGE, issued by A before the restart. The sequence number negotiation decribed above does not solve the complete problem. There are three cases:

1. *The sequence number A is now using is greater than the sequence number used before the restart.* In this case, A's new packets will supersede the old ones, and there is no problem.

2. *The sequence number A is now using is less than the sequence number used before the restart.* In this case, by the third flooding rule above, A will hear back the old packet, and will issue a new Link State Packet with sequence number one greater than the one heard (except in the very rare case that A has already made jumps in the circular sequence number space totaling J, and MAX-AGE must elapse before this case is solved). Thus except after very low-probability events, A will immediately skip to a sequence number big enough to supersede the old packets, and this case is not a problem.

3. *The sequence number A is now using is the same as the sequence number used before the restart.* There are two possible causes:

i. A previously had time only to issue one Link State Packet (# $-k$) before crashing. When A restarts, it again restarts with sequence # $-k$.

ii. A previously issued two Link State Packets, with sequence numbers b and $b+1$, before it crashed. The $b+1$ packet did not yet propagate throughout the net. When A restarts, A is informed of sequence number b first, and switches to $\#b+1$.

If A uses the same sequence number that already exists in the net, A's new Link State Packets will be treated as duplicates and ignored.

If nothing is done about the third case, the network may have inconsistent databases until A issues a new Link State Packet (MAX-INT). Since this problem is a relatively low-probability event, it might be reasonable to allow MAX-INT to elapse before it is solved. However, there is a solution to the problem.

The solution is not to treat a Link State Packet from A as being a duplicate merely because the sequence number matches the stored Link State Packet in the database, but to compare the data to ensure that the Link State Packet is, indeed, a duplicate. If it turns out that the data does not match, but the sequence numbers do, A must be informed, so that A can issue a Link State Packet with the next sequence number, which will supersede the packets that the network disagrees about.

But informing A is not a simple matter of sending a message to A, since it is very likely that routing to A will not work. (The network disagrees about the contents of A's Link State Packet.) Thus some means of flooding must be used to get the information back to A.

The solution is to include a "confusion" bit in each Link State Packet. A Link State Packet with sequence number n and the confusion bit set is treated as having a sequence number between n and $n+1$.

If node D's stored Link State Packet for A has sequence number n, and another Link State Packet for A arrives with sequence number n and different data, D sets the confusion bit on A's Link State Packet, and floods the packet with the "newer" sequence number (n, plus confusion). If node D has a Link State Packet for A with sequence number n, and a Link State Packet for A arrives with sequence number n and the confusion bit, D accepts the packet as newer and floods it. If node D has a Link State Packet for A with sequence number n and the confusion bit, and a Link State Packet arrives with sequence number $n+1$, the arrived Link State Packet is treated as newer and flooded. Thus the confusion bit will get flooded in the same manner as a Link State Packet. When A gets the information that there is confusion about its sequence number n, A will issue a Link State Packet with sequence number $n+1$, unless the information is out of date, since the last Link State Packet A issued had sequence number greater than n. If the information is out of date, A simply ignores it.

Since the flooding scheme involves receiving many duplicates, it may be undesirable to require that every node check each Link State Packet for duplicates. There are two possible optimizations:

1. If a data checksum is included as recommended below, nodes need only compare the data checksum field, and do not have to compare the entire contents of the packets.

2. The problem can occur only for a period of MAX-AGE after a node restarts. A Link State Packet can contain a bit marking it as having been issued by a node which has been up for less than MAX-AGE. Link State Packets which are not so marked do not need to be compared.

3.2.8. Eliminating Corruption

To guard against fields in a Link State Packet from being corrupted in memory by distant nodes, a data checksum field can be added to the packet. This checksum should be computed by the source node and never recomputed by any other node. That implies that the age field and the confusion bit, which are modified by other nodes, must not be included in the checksum.

Luckily, corruption of the age field is not a very serious event. If protection is deemed important for the age field, it can be provided by including two copies of the age field. For added safety, one copy should be the one's complement of the other, making it extremely unlikely that any hardware fault would corrupt both copies in a consistent manner.

Luckily, corruption of the confusion bit is also not a serious event. If some node incorrectly set the confusion bit on S's packet, the Link State Packet with the confusion bit would propagate throughout the net, eventually reaching S, which would issue a new Link State Packet. Having a node incorrectly clear the confusion bit on S's packet is not a serious event unless it prevents S from receiving a copy of its own packet with the confusion bit set. Preventing S from receiving a copy is a very low-probability event since it requires all of the following to occur:

- S restarts with a sequence number still extant in the network. (This can happen easily if S just had time to issue one Link State Packet, $\# -k$, before going down. This can also happen, but is less likely, if S had issued two Link State Packets, $\#b$ and $\#b+1$, right before crashing, and S recovered before $\#b+1$ had time to propagate throughout the net, and when S restarts, it is informed of $\#b$, but not of $\#b+1$.)
- A node D corrupts the confusion bit, but nothing else in the packet that would cause the data-checksum to catch the problem.
- The corruption occurs before D has propagated the packet.
- D does not subsequently receive a duplicate of the packet with the confusion bit set.
- The network topology is such that S is reachable from the portion of the network that discovered the confusion, only through D.

This makes the event of sufficiently low probability that it is tolerable to wait MAX-INT (during which time S will issue a new Link State Packet), for the network to recover.

It is important to guard against a node A, due to hardware problems, issuing a Link State Packet claiming to be a different source node. Before issuing a Link State Packet A must check the ID in the Link State Packet it is about to issue against a copy of its ID stored elsewhere, to ensure that they match. In this way, if A has a mistaken notion of its own ID, or if A is pointing to the wrong place in memory for its packet, the mistake will probably be caught.

4. Summary

This paper presents a scheme for distributing routing information around a network. The scheme is compared to the current ARPANET scheme. Advantages of the scheme over the current ARPANET scheme are:

1. There are no artificial delays imposed on nodes upon restart.
2. Nodes do not need to frequently generate routing traffic in the absence of topological changes.
3. There is less reliance upon timers. Timers that do exist are invoked only as backup after very low-probability events. There is a large safe space from which to choose timer values.
4. Error conditions are less likely to occur, and those that do occur are fixed by the network within reasonable time without human intervention, no matter how low-probability the event was that caused the error, once the offending hardware/software is repaired or removed.

References

[1] J. McQuillan et al., "The New Routing Algorithm for the ARPANET", IEEE Transactions on Communications, May 1980.

[2] E.C. Rosen, "The Updating Protocol of ARPANET's New Routing Algorithm", Computer Networks, Vol. 4, Nr. 1 (1980), 11-20.

[3] E.C. Rosen, "Vulnerabilities of Network Control Protocols: An Example", Computer Communication Review, July 1981.