Exact Hop Count

Mick Seaman

In general network scenarios, hop count limits are a crude approach to mitigating the effects of temporary loops in the topology. In bridged networks in particular, there are parts of the topology where the bridged frame necessarily lack hop counts. Recent analysis of fast reroute techniques for IP has focused on other approaches to loop prevention and mitigation, and the RSTP and MSTP bridging protocols are specifically designed to prevent loops[1]. However, there is ongoing interest in applying routing experience to bridging and instituting a hop count to guard against faulty implementation. This note describes the use of hop count in encapsulating bridge backbones, where it can be made precise and effective, and the additional measures that are required to ensure that loops do not occur in the network as a whole - with frames potentially crossing the backbone multiple times.

It is not my intention, in this note, to argue for the inclusion of a hop count in a backbone encapsulation header[2]. Other loop prevention methods that do not require fields in the header, are available, and I know of no existing dedicated bridge hardware capable of performing the checks described. However I was surprised at the apparent lack of a description of the use of exact hop count checking, either for encapsulating bridging or for routed networks in general, and thought it worth writing down. In addition, the discussion serves to point out the relationship between any backbone specific loop prevention technique, and loop freeness in the bridged network as a whole.

These thoughts are largely a result of a conversation with Francois Tallet and of looking at the IP fast reroute framework[3]. They have benefitted from brief discussion and email exchanges with Mike Shand and Stewart Bryant, but all opinions, errors, and omissions are my own.

1. Introduction

The IP hop count is often checked, as part of crude but effective security measures, to ensure that the packet received is from an immediate neighbour (hop count 255) and has not be routed to the receiving interface. The injection of nuisance packets that might be an attempt at an ‘attack at a distance’ are thus prevented, unless the attacker is either local or can target a local decapsulation or packet creation service.

Within encapsulating backbone, or other core or specialized networks, all the immediate sources and destinations of traffic and their hop count distances may be known. Thus packets can be transmitted with a hop count that is just sufficient to reach the destination. In networks of small diameter this can be effective. Standards recommend a maximum bridge diameter of 8[4], the minimum loop hop count is 3, and most topology protocols and supporting mechanisms will not create a loop that includes either the source (backbone encapsulating) or the destination (backbone decapsulating) nodes. Even if the capsulating nodes are not included in the bridge diameter, this means that a simple loop can cause each node to be visited by a given packet no more than three times. Unfortunately the true maximum depends on the topology, on the topology protocol, and on the forwarding checks applied at each node.

This note considers the benefit not only of selecting an optimal hop count when an encapsulating header is first applied, but also of checking the value of the hop count at each intermediate node. In its simplest form this involves checking the hop count required to reach the decapsulating node, and discarding the packet if that is not exactly correct. The required count is stored with each destination in the forwarding database (the

[1] Attempts to improve the reconfiguration performance of the original spanning tree protocol by ‘tuning’ protocol parameters can create loops, as it lacks the ‘disputed’ Designated Bridge checks provided in RSTP and MSTP. Unfortunately it is possible to choose parameter values so that the loops only appear after the (relatively rare) loss of a BPDU or (in rare implementation dependent cases) when timers in different bridges beat across each other. These effects do not occur in reasonable sized networks with default parameter values.

[2] This note started as a serious attempt to argue the merits of hop count, given the continued contention by some that experience with routers showed that hop count really should be used, but concludes otherwise. However, to take the demand for hop count seriously, the development of the argument has been left as it occured. Read to the end for conclusions.

[3] draft-ietf-rtgwg-lf-conv-frmwrk-00.txt

[4] This number is somewhat out of date. It was selected to ensure that the DEC LAT protocol would operate satisfactorily.

'filtering database' for a bridge) and the database populated by the routing (or bridging) protocol.

2. What is 'a loop'

First it seems necessary to define what is (and possibly what is not) a data loop.

A network is instantaneously loop-free if a snap shot of the totality of data paths shows that if there were no transmission delays or other data buffering in the network, no packet transmitted through the network would be forwarded more than once by any node.

A network provides loop-free transmission if no packet is forwarded more than once by any node.

These two definitions are not the same, because a path within a network that is instantaneously loop-free can buffer a packet while the instantaneous loop-free topology is changing. The packet can be returned to a node after the change so that it is forwarded again.

3. Loop-free properties of exact hop count

A node that implements exact hop count decrements the hop count of each received packet and discards it if it is not exactly as required by the forwarding (filtering) database. A network of such nodes is instantaneously loop-free. If a node receives a given packet a second time, the hop count will be lower, and so cannot be forwarded to the same destination twice.

Such a network does not necessarily provide loop-free transmission, because the topology can change prior to the second reception of a given packet. Consider the network of Figure 1, which shows (symmetric) link costs, the path through the network taken by frames from X to Y, and the hop count at each node along that path.

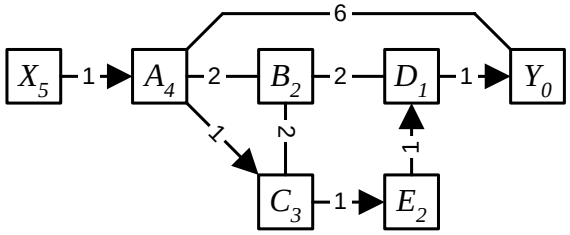

Figure 1—Example network

In Figure 2, the links from C to E and from B to D have failed. B is aware of both failures, C only of the C to E failure, and A of neither. Two link failures are shown because a link state protocol, such as IS-IS, will not create a loop[1] with a single failure if the link costs are symmetric.

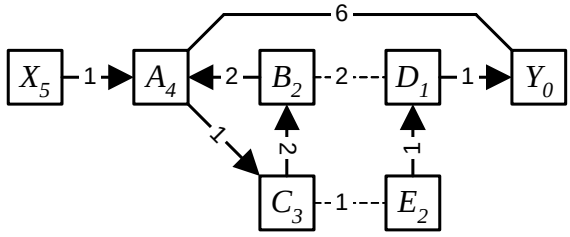

Figure 2—Loop suppressed by exact hop count

If it were not for the exact hop count check, the network in Figure 2 would exhibit a loop A-C-B-A-C-B- until A or C update their forwarding databases to reflect the failure of B-D or C-E respectively. With the exact hop count check, the looping frame is discarded as A attempts to forward it a second time.

If A updates its filtering database to reflect the loss of C-E and B-D while the frame is being transmitted from C to B, then that frame can loop A-C-B-A once., as shown in Figure 3.

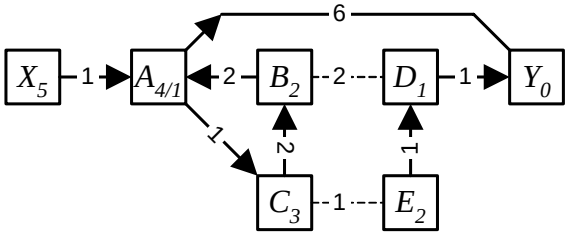

Figure 3—Packet looping in a loop-free network

With exact hop count, a frame can only be forwarded by a given node once plus the number of times as the topology (for that frame) is updated at that node while the frame is being buffered on a network path that returns to the node.

4. Alternatives to packet hop count

One alternative to carrying the hop count in a packet would be to carry the cost to the destination. In that case a frame routed or bridged on a shortest path would never loop (i.e. be forwarded by the same node twice) simply as a result of link failures, as no failure can result in lowering the cost to the destination from a node. The downside of this approach is the number of bits in the encapsulating header that would be required to carry the cost, and their inclusion in each filtering database entry together with the cost decrement required at each node. Moreover the difference between that approach, and the hop count, would only be seen in networks where link costs are such that a link failure can cause a node's shortest path hop count to the destination to *decrease* by at least three. The

[1]Other than loops across a single link, which are uninteresting for bridging application of this note, and loops involving ECMP, which this note does not attempt to discuss.

example network was deliberately constructed to have this property, which is rare amongst bridged networks.

If cost to the destination, rather than hop count, was used, then it makes sense for each node to check that the remaining cost in the received frame is greater than or equal to node's own cost calculation, and to set the cost in the transmitted frame to that calculated by the node. Using this check, rather than requiring strict equality means that the addition of links to the topology traversed by the frame before it reached the node would not cause the frame to be discarded. There is no degradation in loop prevention. The same technique could be applied to a hop count approach, by allowing a greater remaining count on receipt and setting the exact count on transmission.

Another alternative (to exact packet hop count checking) is not to put the hop count in the packet at all. Rather each node communicates with its downstream neighbour to check that that neighbour has a lower hop count to the destination, and temporarily blocks communication if that is not the case. Clearly the check is carried out recursively, so each node reports to its upstream neighbour the fact that either it and all its downstream successors have a given (or lower) hop count to the destination or will block traffic to that destination until that is true. If a node reports a given maximum hop count to a destination, and subsequently processes a topology update that increases that count, it blocks traffic to the destination until all its upstream neighbours indicate that they too have increased their hop counts (and so on recursively. This alternative thus reduces the per packet overhead at the expense of slightly longer failure recovery times, particularly in non-structured networks.

If, instead of using hop counts, the above node to node agreement approach uses the cost to the destination, then it becomes the loop prevention mechanism used in RSTP and MSTP today. In this case the 'destination' is actually the root of a spanning tree[1]. The disadvantage of the use of costs is the space they occupy in control packets when the node to node agreements are being revised, but they do prevent one time loops of the form shown in Figure 3. Moreover they do permit the use of ECMP (equal cost multi-path) with paths that do not have an equal hop count.

5. Multicast

The above discussion is couched in terms of unicast frames. To apply the same logic to multicast to different destinations, the source of that multicast needs to be known. In backbone bridged networks, the destination multicast address itself can be made source specific, or a VLAN Identifier (VID) can assigned to each encapsulating node.

When a source specific multicast is used, unicast and multicast frames are treated differently. The initial hop count of the multicast is set to a value sufficient for the packet to reach every node in the network. The exact hop count value at each node is counted down from source, rather than up from the destination.

If a source specific VID is used, it can be applied to both unicast and multicast in the same way, constraining each frame to a spanning tree rooted at the source node. Clearly the hop count could be counted up, rather than down, from the source, but counting down is compatible with nodes that cannot perform an exact match and is therefore to be preferred.

6. Ingress checking

As part of enforcing spanning tree active topologies, Bridges actually apply a forwarding check on interface reception (port ingress) as well as interface transmission (port egress). If the topology is calculated by a link state protocol, this makes loops even rarer, though not impossible. For the present we will ignore the fact that a bridge network comprises a bi-partite graph (one with two alternating types of nodes) of bridges and LANs with spanning trees reaching to and from every LAN[2] as well as every bridge, and simply

[1]More could be said on this subject, showing that loop free to the root implies loop-free from the root, and that communication between stations where neither is the root is still loop free because that communication is either a loop-free segment of communication to or from the root or is the loop free concatenation of such a segment with communication to or from the root. I assume that the required proofs are trivial for most readers of this note.

[2]One of the reasons that this matters is that the architecture has to allow management of a bridge directly from a directly attached LAN, even if the bridge is not forwarding frames to and from that LAN. Thus it is the case that a point-to-point LAN between two bridges can have attached end stations (management functions in each bridge) and frames from such an end station are forwarded by one bridge even if the other does not. This management requirement extends beyond traditional SNMP management to such capabilities as CFM (P802.1ag Connectivity Fault Management) which has fault coverage requirements, so it is not easily hand waved away.

consider trees of forwarding nodes. Consider the topology of Figure 4.

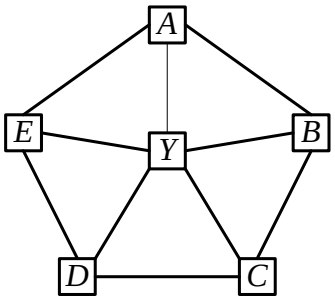

Figure 4—Example network

Each of the links shown is of equal cost, except that between A and Y, which has a somewhat higher cost. Assume that the links from Y to B, C, D, and E fail, that these failures are signalled to each of the nodes in separate link state updates (probably initiated by B, C, D, and E respectively) and that at some time each of the nodes is acting on a slightly different picture of the topology. See Figure 5.

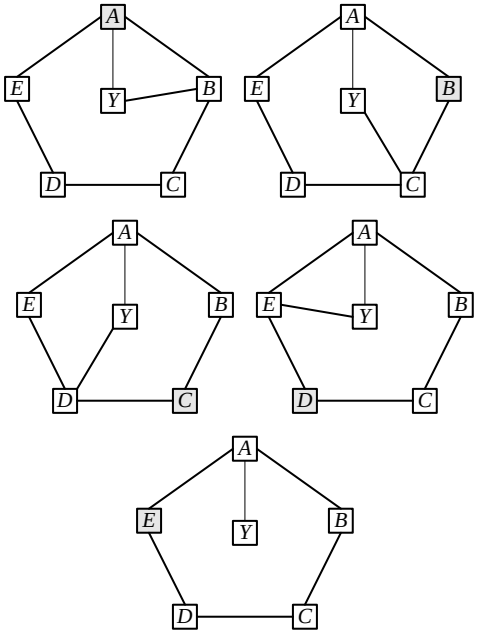

Figure 5—Different views of a changing topology

If A receives a frame destined to Y, it will forward it to B; B will forward the frame to C; C to D; D to E; and E to A once more. The frame will loop until at least one node updates its view of the topology again.

Note particularly that the frame is being forwarded in a loop even though each node checks that it is reasonable (given that node's view of the topology) for its predecessor to forward the frame to the node. The example uses five nodes in the loop to avoid any appeal to equal cost tie-breakers when making that assessment.

This example serves to illustrate looping for both unicast frames (with Y as their destination) and multicast frames (with Y as their source). In the latter case the ingress check serves to prevent Y from adding more frames to the loop once it has formed, so the only frames that are looping must have been buffered in a node that subsequently participates in the loop. This fact serves to further point up the difference between arguments that focus on the instantaneous topology of a network—which could have been used to show that no multicast frame can enter a loop when ingress checks are applied, and thus falsely conclude that looping multicast frames cannot occur—and arguments that focus on the path taken by a frame and include the buffering experienced by the frame. Unfortunately discarding of buffered frames by each node as it changes its view of the topology is not sufficient to excise frames that will subsequently loop, as the frames could be buffered in intermediate nodes whose local view of the topology remains unchanged, at least for long enough for them to host the frame.

The dynamics of packet looping in the presence of ingress checks is discussed further below (7).

A more aggressive form of ingress check is the complete reverse path forwarding check (RPFC). In the context of spanning trees, RPFC amounts to checking that a frame has been received on the tree rooted at its source, and will depart on the tree rooted at its destination—and is exactly the check that a bridge applies to a frame with a source specific VID and a known backbone destination. Examination of the set of instantaneous network topologies again shows (provided forwarding of packets between any pair of stations is symmetric, i.e. follows the same path in the reverse direction) that no frame can enter an existing loop, and no unicast frame can leave such a loop. However a unicast frame can loop, and each multicast frame has potentially many destinations and thus—unconstrained by a destination rooted tree—can both loop and spray copies of itself into the surrounding network.

A possible objection to the above analysis is that topological failures of the form illustrated by Figure 5 are contrived, will never happen in practice, and can be ignored. In this example four links fail, almost simultaneously, and each node processes updates from its peers in the potential loop in anti-clockwise order. The trouble with such can't happen arguments is that they are often exposed in practice, while the mere fear of a low probability network melt-down in a large and complex network can overshadow every attempt at fault finding. In this particular example, simply

imagine that the communication from Y to B, C, D, and E shares some physical transmission medium (e.g. a WDM fiber) at some point, and that each of the nodes are predisposed to forward link state updates out of a given interface, say port 1, more rapidly than through other interfaces, and that these ports happen to be arranged to point clockwise around a ring. Perhaps the failure is not so improbable!

Figure 5 does not, on its own, illustrate a loop for a unicast frame where full RPFC is used, as is the case in a bridged network with source specific VIDs. To construct a loop, for frames with destination Y and source X, add X connected to A thru E just like Y, and have the links from X to B thru E fail so that A believes E–X remains after the others have failed, B sees all failures, C believes B–X remains, D believes C–X remains, and E believes D–X remains. This is the same, in principle, as Figure 5 but with the failures reaching each of the nodes in clockwise rather than anti-clockwise order.

7. Catherine wheels and whirlpools

A multicast frame, looping as just described, can exhibit 'catherine wheel'[1] behavior. At each node around the loop the multicast is flooded on other links, so the catherine wheel sprays the multicast frame into the rest of the network. Once a multicast frame is in such a loop, it can circulate indefinitely. If the forwarding nodes do not implement 'cut through' (rare at high speeds) a single looping frame will occupy at most two buffers at an instant - one at the node transmitting the frame and one at the receiving node. Unless traffic from other sources congests a link in the loop and chance to cause discard of the looping frame, it is likely that the frame will loop until the nodes synchronize their view of the topology. The bandwidth within the loop, occupied by such a looping frame, will (in the absence of traffic shaping or metering within the network as opposed to admission control to the network) be in proportion to that taken by other frames on the most heavily loaded link within the loop. So the net effect of the looping frame on other traffic on a link within the loop is simply to delay that other traffic by the transmission time of that single frame. Much more serious is the effect of the frames that spin off into the rest of the network, as multiple copies of the original looping frame can be added to a single transmission buffer, thus forcing congestion collapse.

Unicast frames, given source based ingress checking, only exhibit a whirlpool (for want of a better name) effect. The looping frame remains within the loop[2], simply delaying traffic of the same priority by at most its own transmission time. A single looping frame will not completely starve lower priority traffic[3], though three or more looping frames might do so. The temporary effect of the looping unicast may be acceptable—provided that priority transmission scheduling has some round robin element within the network and is not simply dependent on admission control to prevent starvation.

8. Loop preventing combinations

As already mentioned, it is not possible to add a hop count to an unencapsulated bridged frame, as the absence of any changes to the frame is what distinguishes basic bridging from other techniques. One of the alternative loop prevention methods has to be used: probably the RSTP/MSTP mechanisms that ensure that the costs to (or from) the root of a spanning tree decrease (or increase) at each node as the frame proceeds to (or from) that root. This method can be combined with a hop count (or other loop prevention mechanism) within the encapsulating backbone of a bridged network. The entire backbone is treated as if it were at the same distance from the root, and frames only allowed to progress from one edge to another if those two edges are indeed at the same distance (and share the same root). To avoid having to add further fields to frames traversing the backbone, each node simply checks the identity of, and distance from, the root with its neighbour, discarding frames after any change until synchronization has been achieved. If the root of the tree is within the backbone, which is usually the case for simple networks, the identity of the root can be omitted—a check that the external distance to the root is zero is sufficient.

9. Conclusions

This note has discussed the utility of hop counts for loop prevention, with particular reference to backbone encapsulating networks, and has shown how loop prevention performance can be improved by requiring an exact hop count match at each node.

At the same time, the correspondence between a per packet hop count and an agreement between neighbouring nodes on the number of hops from each

[1] Otherwise referred to as a pinwheel firework by non-English speakers.

[2] And no other frames, at least between the same source-destination pair, are added.

[3] Unless some inappropriate traffic delay or shaping algorithm prevented link use by some priority on account of a recent higher priority transmission. I don't believe any such algorithm has been contemplated or proposed.

to the destination (or from the source) of the packet has been noted, together with the additional improvement derived from agreeing destination or source) cost rather than hop count. The use of cost also facilitates the use of equal cost unequal hop multipath, though whether this is important is unclear. The use of hops can lessen the communication requirement where large numbers participate in the backbone. In either case, the use of such an agreement mechanism still appears more attractive than per packet hop count.

If the each backbone edge is identified by a VID (VLAN Identifier), the use of source rooted trees as the basis of loop prevention appears a natural choice for bridged networks. Both source and destination rooted trees can be used for unicast, so that the ingress VID check is equivalent to a reverse path forwarding check. While this does not prevent loops, it does prevent unicast traffic from entering or leaving an existing loop, so the excess resource consumed by looping unicast packets may be acceptable—allowing omission of the neighbour to neighbour hop count or distance agreement following link failure updates. A destination rooted tree is not available for multicast, and the effects of a looping multicast packet can be far worse, so there seems little dispute that hop count or distance agreement needs to be applied to multicast.

Finally, the use of distance based agreement can be combined with a per packet hop count approach to a backbone core, simply by providing a neighbour to neighbour tree root identification and external distance check within the core. If the tree roots lie within (or at the encapsulating edge of) the core, as would be natural for a network based on a single backbone, a simple zero external distance check (to confirm that root positioning) is sufficient.