# IS-IS Multicast Synchronization Digest

Jérôme Chiabaut, chiabaut@nortel.com

Don Fedyk, dwfedyk@nortel.com

## 1. Introduction

Shortest Path Bridging (SPB) requires that Loop Prevention be performed before installing multicast state in the Filtering database (FDB). SPB uses IS-IS [1][2] to distribute topology information. in the form of IS-IS TLVs that contain multicast and topology attributes. IS-IS ensures that this information is delivered in a timely manner, is accurate and is synchronized as part of the existing operation, but IS-IS does not monitor the population of forwarding information in the fast path. In this paper, the term “forwarding information” or “forwarding state” is used to signify the mapping of the control plane FDB into the fast path configuration installed on each port.

In the normal mode of operation artifacts of IS-IS LSP exchange may create small temporal forwarding loops. Therefore some form of loop mitigation is typically applied to forwarding that is controlled by a link state database protocol such as IS-IS. While temporal forwarding loops can be mitigated by various methods, the SPB design goal is the elimination of forwarding loops for multicast (emulating the defined behavior of Spanning tree protocols). Mitigation techniques [1] do not completely remove loops for multicast and broadcast in all cases so loop prevention is the proposed approach in SPB.

Loop prevention requires agreement between nodes both on the contents of the topology database and that a set of loop prevention actions have been performed. These actions are in response to changes to the database that have occurred since the last time the topology between two neighbors was synchronized. This agreement must occur prior to refreshing potentially unsafe multicast state in the FDB. Such a mechanism does not currently exist in IS-IS.

To introduce a rapid verification of synchronization of loop prevention procedures, it is desirable to augment routing exchanges with a mechanism that required a minimum number of transactions. This implies an exchange of some form of digest of database and an associated semantic that the message confirms that a defined set of actions had been performed. This digest could be a complete list of the LSPs in the node’s link state database (LSDB) or a more compact, but lossy, representation of that information.

This paper describes an algorithm for computing a small digest of the IS-IS LSDB and highlights some implementation considerations. This algorithm is easy to implement, is computationally efficient, and supports incremental updates of the digest. The digest produced can detect loss of synchronization between a pair of SPB nodes with a very high probability.

## 2. SPB Synchronization

SPB requires that each and every node in the network compute and install multicast or broadcast forwarding state if, and only if, the node is on the shortest path tree between a given multicast source and one or more of the intended receivers. Conceptually, each node needs to do the following: for each pair of nodes, A and B, in the network 1) ask itself: “am I on the shortest path between nodes A and B?” and, if the answer is “yes”, 2) install multicast forwarding state for the I-SIDs that are common to nodes A and B.

IS-IS may create transient forwarding loops because the various nodes install forwarding information at different instants in time. Loops involving multicast traffic are potentially disruptive because, even when a data path ingress check prevents new traffic from entering the loop, copies of the multicast packets trapped in the loop can be replicated. These replicated packets could appear outside the loop at a rate that is only limited by the loop round trip delay, possibly flooding the rest of the network with duplicate traffic.

In order to eliminate the risk of multicast and broadcast loop formation for all practical purposes, SPB nodes must verify that their view of the network topology is synchronized with their neighbors' before installing any new multicast forwarding state. Topology synchronization is a normal function of a routing system, however, while routing systems are very efficient and reliable in synchronizing the exchange of network topology information, the routing system does not cover the actual synchronization of the forwarding database.

The proposed synchronization is tied to the FDB update process and signals the removal of any multicast forwarding information that has become unsafe as a result of a topology change and ensures that adjacent bridges are completely in synchronization before repopulating any potential loop forming FDB entries.

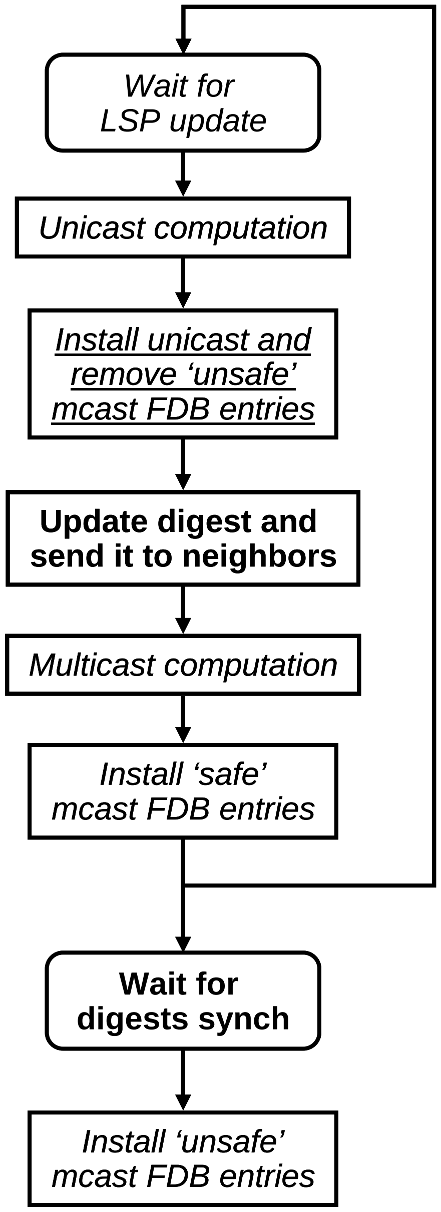
Wait for LSP update
Unicast computation
Install unicast and remove 'unsafe' mcast FDB entries
Update digest and send it to neighbors
Multicast computation
Install 'safe' mcast FDB entries
Wait for digests synch
Install 'unsafe' mcast FDB entries

Figure 1 Digest Synchronization Flow Chart

The simple abstract state machine in Figure 1 illustrates a possible modification to the FDB computation that uses the proposed digest to prevent loops. This logic is triggered whenever IS-IS detects an actual change in its LSDB, but it does not change the IS-IS operation. This mechanism ensures that in no circumstance can a bridge populate a multicast forwarding entry that could cause a loop to form. By synchronizing in this manner, a bridge is sure that each neighbor bridge has also removed any forwarding entries that may have caused a loop before repopulating its own unsafe multicast tree FBD entries. All bridges follow this procedure in parallel allowing rapid and safe repopulation of multicast entries.

## 3. Make-Before-Break Digest Requirements

When considering the requirements for a digest it is useful to form a perspective on the frequency and occurrence of loops. Although conditions for loops can occur in SPB networks in the absence of explicit loop prevention, the loops are infrequent and are unlikely to last for any noticeable length of time in a well engineered deployment. The IS-IS flooding mechanism combined with the SPB multicast capabilities ensures that the nodes' LSDBs will be synchronized within a few tens of milliseconds at the most. Also because of the strict SPB rules – multicast forwarding state is not installed unless required – and their enforcement by a data path ingress check, the formation of a loop requires some rather contrived circumstances involving at a minimum 4 nodes with inconsistent views of the network topology.

Because loops are expected to be the exception rather than the norm, the mechanism used to prevent them should be as lightweight as possible, reducing the likelihood that it interferes noticeably with normal network operation. In particular, the LSDB digest should be easy to compute and small enough to be easily exchanged in messages.

A large SPB deployment could easily have hundreds of nodes each with tens of Link State Protocol Data Units (LSPs) and their LSDB could, therefore, have upwards of thousands of LSPs in it. The IS-IS protocol has rules to ensure that, under normal circumstances, a simple topology change, such as a link failure or repair, will cause two, and only two, of these LSPs to change. Therefore, in a large network, the number of LSPs affected by a topology change will be a very small fraction of the total. Even a complete nodal failure will only affect a small number of LSPs, one for each of the failed node's neighbors.

An efficient digest method should allow for the incremental update of the digest based on the changed information alone so that the computational burden is roughly proportional to the number of LSPs that changed, not the total number of LSPs in the LSDB.

The normal IS-IS LSP refresh process causes new versions of LSPs to be flooded through the network approximately every 15 minutes by default. Therefore an LSDB with upwards of thousand LSPs will receive a new LSP every second or so – (the IS-IS 20 minute MaxAge architectural 'constant' is sometimes increased to reduce the traffic and churn induced by this refresh process). These LSPs are simply copies of the ones they are replacing with a new lifetime and a new sequence number (and a new checksum since the LSP sequence number is included in the checksum computation).

An efficient digest method should be immune to the apparent endless churn caused by the IS-IS refresh process for the following two reasons. Most importantly, when an actual topology change occurs, two nodes may appear to be de-synchronized because they have different versions of an unrelated LSP that is in the process of being refreshed. This apparent lack of synchronization caused by the normal IS-IS LSP refresh process will unnecessarily slow down convergence. Secondly, the high rate of LSP refreshes would impact the computational efficiency of such a digest: incrementally updating the digest each time a new version of an LSP is received implies needlessly wasting computing resources. Alternatively, computing the digest only when needed (i.e. when an actual topology change is detected by the node) means scanning the entire LSP database at that time.

It should be noted that, for loop avoidance purposes, the digest only needs to consider the topology information, and not the multicast service information (SPB) or the service information – the I-SIDs – (SPBB), possibly present in the LSDB. However, extracting and normalizing this information is, in general, a complex and compute-intensive process. Also the definition and implementation of such a digest would have to be revised each time the IS-IS protocol is extended, resulting in compatibility issues in both standards and implementations. The importance of stability and simplicity cannot be overstressed: a faulty digest implementation will do a lot more harm than good, given the odds of loops forming in SPB networks. Therefore, it is recommended that the digest does not require parsing the LSPs' content.

## 4. Proposed Make-Before-Break Digest

The proposed digest is based on a two step process: firstly, a high quality digest is used to normalize the LSP content to a small, fixed-size, quantity. Then, the digests of all valid LSPs are combined to form the LSDB digest. This two-step process limits the computational impact of the digest: a topology change will typically cause the re-computation of a couple of LSP digests and of the LSDB digest based on the LSP digests. The unchanged LSPs aren't even considered.

The proposed LSP digest includes the LSP identifier and the LSP payload but excludes the LSP remaining lifetime, sequence number, and checksum. The digest of an LSP only needs to be re-computed when the node's IS-IS implementation detects an actual change in the LSP payload – something that is an integral part of the normal IS-IS decision process and therefore does not increase the computational load.

The LSDB digest is based solely on the digests of the LSPs in the LSDB. Naturally, invalid and purged LSPs must be excluded from the LSDB digest computation. Invalid LSPs are not only LSPs with invalid checksum but also expired or purged LSPs and LSPs with an invalid sequence number (zero) or remaining lifetime (greater than MaxAge). Also excluded are LSPs for which LSP zero is not present or has a remaining lifetime of zero.

This combination of the LSP digests to form an LSDB digest involves some form of computation. This could be accomplished, for instance, by putting the LSP digests in a list and computing a digest of that list. However, to produce a consistent result, such an approach requires normalizing the order of the LSP digests in the list – not a trivial task when thousands of LSPs are involved – and computing a digest over a list that could be many times the size of an LSP.

In order to sidestep these issues, we recommend the use of an order-independent way of combining the digests: the LSDB digest is simply the exclusive-or of the digests of the valid LSPs. This has the advantage that the LSDB can be updated very simply and quickly: when an LSP digest changes, the old LSP digest is xor'ed out of the LSDB digest and the new LSP digest is xor'ed in. The appearance or disappearance of LSPs is even simpler, necessitating only one xor operation.

However this choice of combining function puts some additional constraints on the choice of LSP digests. Because exclusive-or is a linear operation, the LSP digests should not themselves also be linear operations (an example of which is a cyclic redundancy check (CRCs)). Otherwise simple changes in a pair of LSPs could cancel each other out in the LSDB digest. Consequently, the current proposal uses a secure hash function, SHA-256 [4][5], to produce a more randomized, 32-byte, LSP digest. These 32-byte LSP digests are then xor'ed together to produce a 32-byte LSDB digest.

A secure hash is slightly more expensive to compute than, for instance, a CRC, but the much simpler xor used for combining LSP digests into an LSDB digest more than makes up for this. Close inspection of a non-optimized software implementation of SHA-256 shows that, despite its many rounds, it performs about five times more ALU operations than a straightforward CRC implementation and, therefore, should only be about five times slower on most general purpose CPUs. Furthermore, regardless of the choice of hash function, the cost of computing the proposed digest is insignificant compared to the other processing functions triggered by LSP changes so the secure hash with its much lower probability of false positives (256-bit secure hash vs. 64-bit linear CRC) suggests itself as the natural choice.

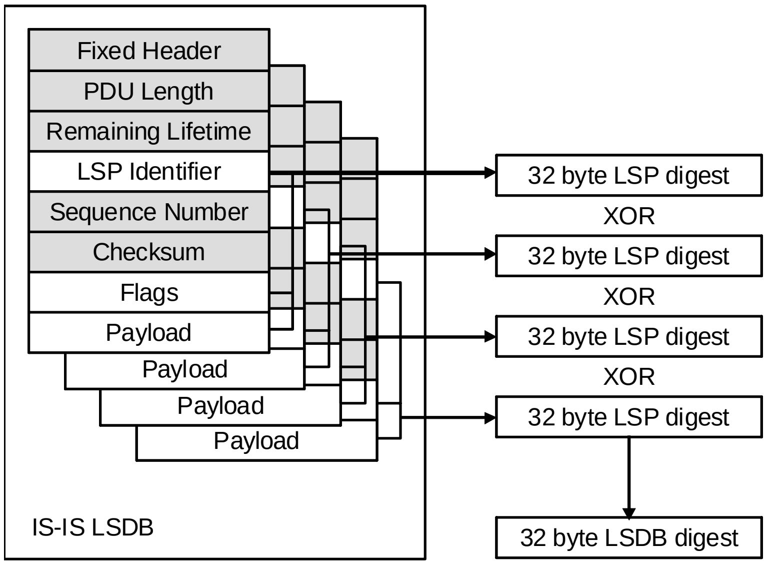

Figure 2 Link State Database Digest

## 5. Summary & Conclusions

This paper has presented a practical synchronization mechanism that can be used in Shortest Path Bridging to provide multicast loop prevention. While the method is targeted at shortest path tree algorithms it works for all distributed tree algorithms.

The proposed LSDB digest algorithm is easy to implement, is computationally efficient, and supports incremental updates. A 32-byte LSDB digest is formed by xor'ing together the digests of the relevant LSPs. The LSPs digests are computed by applying SHA-256 to the LSP identifiers and their payload. An LSP digest only needs to be re-computed when an actual change in the content of the LSP is detected by the IS-IS implementation. Updating the LSDB digest when an LSP digest is re-computed is a trivial operation that involves at the most a couple of xor operations. The 32-byte LSDB digest produced in this fashion can detect loose synchronization between a pair of SPB nodes with a very high probability.

## 6. Acknowledgements

The authors would like to thank Nigel Bragg and Dave Alan for their input and comments on this paper.

## 7. References

[1] ISO/IEC 10589, *Intermediate System to Intermediate System intra-domain routeing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service (ISO 8473)*, 2nd edition, November 15, 2002.

[2] *The Complete IS-IS Routing Protocol*, Hannes Gredler and Walter Goralski, 1st edition (December 16, 2004), ISBN 1852338229.

[3] Exact Hop Count, Mick Seaman. http://www.ieee802.org/1/files/public/docs2006/aq-seaman-exact-hop-count-1206-01.pdf

[4] FIPS 180-3, *Secure Hash Standard (SHS)*, United States of America, National Institute of Standards and Technology, Federal Information Processing Standard (FIPS) 180-3, October 2008. Available online at http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

[5] RFC 4634, *US Secure Hash Algorithms (SHA and HMAC-SHA)*, D. Eastlake and T. Hansen, July 2006.