Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: 4 September 2020

S. Chen
T. Li
Arista Networks

3 March 2020

An Algorithm for Computing Dynamic Flooding Topologies
draft-chen-lsr-dynamic-flooding-algorithm-00

Abstract

Link-state routing protocols suffer from excessive flooding in dense network topologies. Dynamic flooding [I-D.ietf-lsr-dynamic-flooding] alleviates the problem by decoupling the flooding topology from the physical topology. Link-state protocol updates are flooded only on the sparse flooding topology while data traffic is still forwarded on the physical topology.

This document describes an algorithm to obtain a sparse subgraph from a dense graph. The resulting subgraph has certain desirable properties and can be used as a flooding topology in dynamic flooding.

This document discloses the algorithm that we have developed in order to make it easier for other developers to implement similar algorithms. We do not claim that our algorithm is optimal, rather it is a pragmatic effort and we expect that further research and refinement can improve the results.

We are not proposing that this algorithm be standardized, nor that the working group use this as a basis for further standardization work, however we have no objections if the working group chooses to do so.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 4 September 2020.

Table of Contents

1. Introduction

In [I-D.ietf-lsr-dynamic-flooding], dynamic flooding is proposed to reduce the flooding of link-state protocol packets in the network. The basic idea is to find a sparse flooding topology from the physical topology and flood link-state protocol data units (LSPDUs or LSPs) only on the flooding topology. The flooding topology should have the following properties:

1. It should include all nodes in the area. This ensures that LSPs can reach all nodes.

2. It should be biconnected if possible. This ensures that the LSP delivery is resilient to a single node or link failure.

3. It has a limited diameter. Being a subgraph, the flooding topology often has a larger diameter than the physical topology.

A larger diameter indicates a longer convergence time. The tradeoff between flooding reduction and convergence should be considered during the flooding topology computation.

4. It has a balanced degree of distribution. The degree of a node on the flooding topology indicates its burden in flooding LSPs. It is desirable to balance this burden across multiple nodes. Hence, the degree of each node should also be considered during flooding topology computation.

With the above properties in mind, we propose an iterative algorithm to compute the flooding topology.

2. Problem Statement

We model the physical topology as an undirected graph. Each system or pseudonode in the area is represented by a node in the graph. An edge connects two nodes who advertise each other as neighbors. Given the set of the nodes and the set of edges, we propose an algorithm to compute a biconnected (if possible) subgraph that covers all nodes. The subgraph is computed with consideration of diameter and node degree.

3. Algorithm Outline

A simple cycle that covers all nodes is a biconnected subgraph with balanced node degrees. While it has some desirable properties, a simple cycle is not suitable as a flooding topology at large scale. With N nodes in the area, a link-state update has to take N/2 hops to reach all nodes. The undue propagation delay causes a long convergence time.

The proposed algorithm constructs a subgraph composed of small overlapping cycles. The base graph is denoted by G(V, E), where V is the set of all reachable nodes in this area, and E is the set of edges. The subgraph to be computed is denoted by G'({}, {}), which

starts with an empty set of nodes and an empty set of edges.

1. Select a subset of nodes V(0) from V and a subset of edges E(0) from E to form an initial cycle. This cycle is added to the subgraph: G'(V(0), E(0)).

2. Select a subset of nodes V(i) and a subset of edges E(i), that are not included in the current subgraph. That is, V(i) is selected from V - V(0) - ... - V(i-1) and E(i) is selected from E - E(0) - ... - E(i-1). These nodes and the edges are selected to form an arc path whose two endpoints are included in the current

subgraph. This arc path is added to the subgraph: G'(V(0) + V(1) + ... + V(i), E(0) + E(1) + ... + E(i)).

3. Repeat step 2 until all nodes are included in the subgraph G'.

The subgraph constructed by this algorithm has the following properties:

1. It covers all nodes in the area.

2. It is biconnected. This can be easily proven by induction. Specifically, the initial cycle is biconnected. Adding an arc path, whose two endpoints differ, to a biconnected subgraph maintains the biconnectivity of the subgraph.

3. It has a limited diameter. By selecting small cycles, the subgraph will have a smaller diameter. More specifically, the implementation can pick endpoints of each arc path to reduce the diameter of the subgraph. The degree of a node in the subgraph is determined by the number of arc paths it is on. By carefully selecting the arc endpoints, we may balance the node degrees.

Together with the encoding scheme in [I-D.ietf-lsr-dynamic-flooding], this algorithm can be used to implement centralized dynamic flooding. The area leader can build the base graph from its link-state database (LSDB), apply this algorithm to compute the flooding topology, and then encode it into the Dynamic Flooding Path TLVs specified in [I-D.ietf-lsr-dynamic-flooding]. In a topology change event, the area leader can repeat the above process and send out the new

flooding topology.

4. Algorithm Details

The outlined algorithm allows for different approaches to find the initial cycle and subsequent arc paths. We do not intend to find the theoretically optimal solution. Our aim is to find a practical approach that works for any connected base graph, and is also easy to implement.

4.1. Initial Cycle Setup

The initial cycle forms a base of the subgraph computation. Intuitively, we would like to place the initial cycle around the centroid of the base graph, and gradually expand it outwards. Complicated graph analysis can help, but is not desired.

We propose to select a starting node and then search a path that ends at this node. We suggest selecting the node with the highest degree

as the starting node. The degree of each node can be easily determined when the base graph is constructed from the LSDB. Starting from this node, we perform a depth-first search (DFS) for a limited number of steps, and then a breadth-first search (BFS) to find the shortest path back to the starting node. The restriction of the DFS depths and the use of BFS effectively help limit the diameter of the initial cycle. Below is a summary of the procedure. We omit the details of the well-known DFS and BFS algorithms.

1. Let V(0) = [] be the list of nodes on the initial cycle.

2. Find the starting node, denoted by n0. V(0) = [n0].

3. Perform DFS starting from n0 until either the first leaf is reached or the depth exceeds a preset limit. The visited nodes are denoted by n1, n2, ..., ni, where i < DFS depth limit, and added to the list V(0) in order.

4. Mark nodes in V(0) as visited to avoid BFS visiting these nodes. Then perform BFS to find the shortest path from ni to n0. Append nodes on the shortest path to V(0). Since this is a cycle, node n0 appears twice in V(0): the first and the last places.

Note that since DFS and BFS do not process a node more than once, we are ensured to obtain a cycle (if one exists) from the above procedure.

4.2. Arc Path Selection

After obtaining an initial cycle, we recursively add arc paths to the subgraph until all nodes are included. Each arc path's two endpoints are chosen from the current subgraph. This ensures that the resulting subgraph remains biconnected. To limit the diameter of the resulting subgraph, we select an arc path with limited length and attach it closer to the initial cycle.

In each iteration, we have a starting subgraph, which includes the initial cycle and the arc paths obtained in earlier iterations. We first select a node from the starting subgraph, that has at least one neighbor that is not in the starting subgraph. To balance the degree distribution, we prefer to select a node that has the least degree in the starting subgraph. Meanwhile, we intend to place the new arc path closer to the initial cycle, which will help reduce the diameter of the resulting subgraph. As the number of iterations increases, it becomes hard to find such nodes that meet both conditions. A tradeoff between the node degree and the node distance to the initial cycle has to be made.

Starting from the selected node, we perform a DFS for a limited number of steps in the base graph to include nodes and edges that do not belong to the starting subgraph. Then BFS is performed to find the shortest path back to any node in the starting subgraph except the starting node of this iteration. The resulting path is combined with the starting subgraph to generate a new subgraph, which serves as the starting subgraph in the next iteration. The iteration is repeated until all nodes in the base graph are included in the subgraph.

The procedure in each iteration is very similar to the one used to find the initial cycle, except that the two endpoints of the new path do not match:

1. Let V(i) = [] be the list of nodes found in the i-th iteration.

The starting subgraph includes nodes in V(0) + ... + V(i-1), and edges between each pair of adjacent nodes in each node list V(j).

2. Select a starting node n0 for this iteration from the starting subgraph. V(i) = [n0].

3. Mark all nodes in V(0) + ... + V(i-1) as visited. Then perform DFS from n0 until either the first leaf is reached or the depth exceeds a preset limit. Nodes visited in this iteration are denoted by n1, n2, ..., ni in order, and appended to the list V(i).

4. Mark all nodes in V(0) + ... + V(i-1) + V(i) as visited. Then perform BFS to find the shortest path from ni to any node in V(0) + ... V(i-1) - [n0], where n0 is the starting node in this iteration. If a path is found, append its nodes to V(i) and repeat the iteration from step 1.

By correctly marking the visited nodes before DFS and BFS, we ensure that the obtained arc path (if it exists) has two endpoints and only these two points on the starting subgraph.

4.3. Exceptions

If the base graph is biconnected, there exists a simple cycle between any two nodes. We are thus ensured to find one arc path in each iteration and the algorithm described above will yield a biconnected subgraph that covers all nodes in the base graph. Otherwise, however, we may not be able to find an arc path with both endpoints belonging to the starting subgraph in that iteration. When this happens, we know that the edge between the last node found by DFS and its parent is a cut edge in the base graph. For connectivity, this edge must be included in the resulting subgraph. Hence, when step 4

(the BFS stage in earlier procedures) fails, we should amend it with the following:

5. If BFS does not find a shortest path from ni, and V(i) contains only two nodes, i.e., the cut edge case, then go back to step 1 to continue the iteration.

6. If BFS does not find a shortest path from ni, and V(i) contains

more than two nodes, then remove the last node from V(i), and go back to step 4.

Similarly, we might face the same problem when selecting the initial cycle. We can apply step 6 until we find a cycle. However, if we happen to find a cut edge, we can change the first neighbor of the starting node that is visited by the DFS and repeat the procedure. If all edges connecting to the starting node are cut edges, we can change the starting node. If an initial cycle is not found after all the above efforts, indicating that the base graph does not have a cycle, then we will return the base graph as the result.

4.4. LANs

We model a pseudonode as a node in the base graph. The proposed algorithm can be applied as-is. There are, however, possible optimizations for the multi-access LAN case. First, a pseudonode is not required to be on the flooding topology. The algorithm can thus be terminated as soon as all real nodes are included in the subgraph. Second, if a pseudonode is included on the flooding topology, all nodes connecting to this LAN will have to flood their LSPs to this LAN (see [I-D.ietf-lsr-dynamic-flooding] Section 6.6). Hence, if a pseudonode is included in the subgraph, then it will automatically provide uni-connectivity to all its neighbors that are not yet included. The algorithm can take advantage of this LAN property to reduce the edges in the subgraph.

5. Example

The proposed algorithm can be applied to any connected base graph. For ease of explanation, we consider a complete graph of 10 nodes and 45 edges. To limit the diameter of the resulting subgraph, we pre-set the maximum steps in the DFS to 3.

1. Let ni, i = 0, 1, ... 9, denote nodes in the base graph.

2. Find an initial cycle.

    a. Without loss of generality, we select node n0 as the starting point.

    b. Perform DFS from n0 for 3 steps. We obtain a path n0 - n1 -

n2 - n3.

c. Perform BFS from n3 to n0. Since this is a complete graph, where every node is directly connected to any other node, the shortest path is only one hop away.

d. The initial cycle is found n0 - n1 - n2 - n3 - n0.

3. Find the first arc path.

a. Select a node on the initial cycle, say n0.

b. Perform DFS from n0 for 3 steps. We obtain a path n0 - n4 - n5 - n6.

c. Perform BFS from n6 to any node on the initial cycle except n0, i.e., {n1, n2, n3}. These three nodes have the same degree. We may select any one of them as the endpoint. Suppose that n1 is selected.

d. The first arc path is found n0 - n4 - n5 - n6 - n1.

4. Find the second arc path.

a. When selecting the starting node for this path, we may consider the current node degree as well as the node distance to n0 (the starting point of the initial cycle). We notice that both n0 and n1 have a degree of 3 while other nodes have a degree of 2. Nodes n1, n3, n4 are the closest to n0. We select a node with a lower degree and closer to n0. Suppose n3 is selected.

b. Perform DFS from n3 for 3 steps. We obtain a path n3 - n7 -n8 - n9.

c. Perform BFS from n9 to any node except n3. Using the same criteria in a. to select the endpoint, we select n4.

d. The second arc path is found n3 - n7 -n8 - n9 - n4.

5. The subgraph has included all nodes. The iteration ends.

The subgraph found by the proposed algorithm can be represented by three paths:

n0 - n1 - n2 - n3 - n0

n0 - n4 - n5 - n6 - n1

n3 - n7 -n8 - n9 - n4

The subgraph has 12 edges, significantly reduced from 45 in the base graph. The highest node degree is 3 and the lowest node degree is 2. The diameter of the subgraph is 4, increased, as expected, from that of the base graph.

6. Security Considerations

This document introduces no new security issues. Security issues within dynamic flooding are already discussed in [I-D.ietf-lsr-dynamic-flooding].

7. Informative References

[I-D.ietf-lsr-dynamic-flooding]
Li, T., Psenak, P., Ginsberg, L., Chen, H., Przygienda, T., Cooper, D., Jalil, L., and S. Dontula, "Dynamic Flooding on Dense Graphs", Work in Progress, Internet-Draft, draft-ietf-lsr-dynamic-flooding-04, 26 November 2019, <https://tools.ietf.org/html/draft-ietf-lsr-dynamic-flooding-04>.

Authors' Addresses

Sarah Chen
Arista Networks
5453 Great America Parkway
Santa Clara, California 95054
United States of America

Email: sarahchen@arista.com

Tony Li
Arista Networks
5453 Great America Parkway
Santa Clara, California 95054
United States of America

Email: tony.li@tony.li