Efficient LFA Implementation Method Based on Incremental Shortest Path First Algorithm Post-print

Authors: Geng Haijun, Guo Xiaoying, Yin Xia

Date: 2019-01-03T00:00:00+00:00

Abstract

To address the problems of high computational overhead and deployment difficulty associated with existing LFA implementation methods, we propose an LFA implementation method based on the incremental shortest path first algorithm (LFA implementation method based on incremental shortest path first algorithm, ERPISPF). Firstly, the problem of achieving rapid LFA implementation is transformed into the problem of efficiently computing the minimum cost from all neighbor nodes of a computing node to all remaining nodes in the network on a shortest path tree rooted at the computing node. Subsequently, a theorem for computing this cost is proposed and its correctness is rigorously proven. Finally, the time complexity of the algorithm is analyzed theoretically. Simulation results indicate that ERPISPF not only exhibits low computational overhead, but also provides the same failure protection rate as LFC.

Full Text

Efficient LFA Implementation Method Based on Incremental Shortest Path First Algorithm

Geng Haijun[1], Guo Xiaoying[1], Yin Xia[2]
[1]School of Software Engineering, Shanxi University, Taiyuan 030006, China;
[2]Dept. of Computer Science & Technology, Tsinghua University, Beijing 100084, China

Abstract

To address the high computational overhead and deployment difficulty of existing LFA implementation methods, this paper proposes an efficient LFA implementation method based on the incremental shortest path first algorithm

(ERPISPF). First, the problem of rapid LFA implementation is transformed into how to efficiently calculate the minimum cost from all neighbor nodes to all other nodes in the network on the shortest path tree rooted at the computing node. Then, a theorem for calculating this cost is presented and its correctness is proved. Finally, the time complexity of the algorithm is theoretically analyzed. Experimental results demonstrate that ERPISPF not only incurs lower computational overhead but also achieves the same failure protection rate as LFA.

Key words: i-SPF; loop free alternates; network failures

0 Related Work

The literature [20] proposes first using loop-free conditions to compute all eligible next-hops between node pairs, then employing label switching techniques to select appropriate next-hops for packet forwarding. However, this approach suffers from high algorithmic complexity and requires modifications to packet headers, making it unsuitable for practical network deployment. FIR [21] suppresses failure notification messages by setting timing parameters and uses locally precomputed backup next-hops to forward affected packets. If the failure persists beyond the predetermined time, the failure information is flooded and routing reconvergence is triggered. FIR is effective for transient failures but performs poorly for prolonged failures, can protect against link failures but not node failures, and faces challenges in selecting appropriate suppression parameters. Literature [22] modifies traditional FIR to enable protection against node failures, while literature [23] extends FIR to support networks with asymmetric link weights. MRC [24] employs multiple topology configurations to protect against network failures.

The IETF proposes using the LFA method to address network failures. However, computing nodes must calculate multiple shortest path trees to implement LFA, which significantly increases router burden. To reduce the algorithmic overhead of LFA implementation, researchers have conducted extensive work, with representative studies including TBFH [25] and DMPA [26].

1.1 Network Model

Consider a network topology $G = (V, E)$ where V represents the node set and E represents the edge set. For any node $c \in V$, its neighbor set can be represented as $N(c)$, and the shortest path tree rooted at c can be represented as $spt(c)$. In $spt(c)$, $D(c, x)$ denotes all descendant nodes of node x (including node x itself). For any $(i, j) \in E$, $w(i, j)$ represents the link cost. For any $x, y \in V$ and $x \neq y$, $cost(x, y)$ is the minimum cost from node x to node y.

[Figure 1: see original paper] shows the shortest path tree rooted at node c. In this tree, node c' s neighbor nodes can be represented as $N(c) = \{a, b\}$. Node c' s child nodes can be represented as $child(c) = \{a\}$, and node c' s descendant nodes can be represented as $descendant(c) = \{a, d, e, f, g, h\}$. The cost of the

shortest path from node c to node x is $cost(c, x)$. The default next-hop from node c to node x is $dn(c, x)$. The backup next-hop from node c to node x is $bn(c, x)$.

1.2 Problem Description

To improve network availability and enhance user experience, the IETF proposes using LFA rules to handle network failures. Below we introduce three rules in LFA:

a) **LFC (Loop-Free Condition):** For any destination address d, node c can send packets to its neighbor node x if and only if $cost(x, d) < cost(x, c) + cost(c, d)$.

b) **NPC (Node-Protecting Condition):** For any destination address d, node c can send packets to its neighbor node x if and only if $cost(x, d) < cost(x, next^*(c, d)) + cost(next^*(c, d), d)$, where $next^*(c, d)$ represents node c' s optimal next-hop to destination d.

c) **DC (Downstream Condition):** For any destination address d, node c can send packets to its neighbor node x if and only if $cost(x, d) < cost(c, d)$.

To implement the LFC rule on node c, the node needs to know the values of $cost(x, d)$ for all $x \in N(c)$. To implement the NPC rule, node c needs to know the values of $cost(x, next^*(c, d))$. To implement the DC rule, node c needs to know the values of $cost(x, d)$ and $cost(c, d)$. Therefore, the problem this paper needs to solve can be described as: For node c, given $spt(c)$, can we find an algorithm to quickly calculate the minimum cost corresponding to all its neighbors for all other nodes in the network? Theorem 1 answers this question and provides a proof.

Theorem 1. Given computing node c and its shortest path tree $spt(c)$, for any node x in the network, let $spt'(c)$ denote the new shortest path tree rooted at node c after adjusting the cost of link (c, x) to $cost(c, x) - 1$. Then:

- If $d \in D(spt(c), x)$, then $cost'(c, d) = cost(c, d) - cost(c, x)$
- If $d \notin D(spt(c), x)$, then $cost'(c, d) = cost(c, d)$

where $cost'(c, d)$ represents the minimum cost from node c to node d in $spt'(c)$.

Proof. The proof is divided into two cases:

Case 1: If $d \in D(spt(c), x)$, then the shortest path from c to d in $spt(c)$ must pass through link (c, x). Since we only changed the cost of link (c, x) and this link lies on the path, the new shortest path cost becomes $cost'(c, d) = cost(c, d) - cost(c, x)$.

Case 2: If $d \notin D(spt(c), x)$, then the shortest path from c to d in $spt(c)$ does not pass through link (c, x). Since the cost change only affects link (c, x) and this

link is not on the path, the shortest path cost remains unchanged: $cost'(c,d) = cost(c,d)$.

2.1 Algorithm Description

Algorithm ERPISPF
Input: $G=(V,E)$, $spt(c)$
Output: $bn(c,d)$ for all $d \in V$

1: for each $x \in N(c)$ do
2: $weight \leftarrow cost(c,x)$
3: $cost(c,x) \leftarrow cost(c,x)-1$
4: Construct $spt'(c)$ using i-SPF algorithm
5: for each $d \in V$ and $d \neq c$ and $d \neq x$ do
6: if $d \in D(spt(c),x)$ then $cost'(c,d) \leftarrow cost(c,d) - weight$
7: else $cost'(c,d) \leftarrow cost(c,d)$
8: endfor
9: $cost(c,x) \leftarrow weight$
10: endfor
11: for each $d \in V$ and $d \neq c$ do
12: for each $x \in N(c)$ do
13: if LFA condition holds then $bn(c,d) \leftarrow bn(c,d) \cup \{x\}$
14: endfor
15: endfor
16: return $bn(c,d)$

Algorithm ERPISPF illustrates how node c implements LFA rules. The algorithm takes $G=(V,E)$ and $spt(c)$ as inputs and returns the LFA next-hops from node c to all nodes. It visits each neighbor node x of c, adjusts the cost of the link between them to $cost(c,x)-1$ (lines 2-3), uses the incremental shortest path first algorithm to compute the shortest path tree $spt'(c)$ rooted at node c (line 4), then calculates the new costs according to Theorem 1 (lines 5-8), and finally restores the original link cost (line 9). The LFA condition for computing backup next-hops from node c to all other nodes is evaluated (lines 11-15). The LFA condition in line 13 can represent LFC, NPC, or DC; modifying this condition allows computation of backup next-hops satisfying different rules.

2.2 Algorithm Performance Analysis

Theorem 2. The runtime complexity of ERPISPF is $O(|E|\log|V|)$.

Proof. Assume node c runs the ERPISPF algorithm. From the pseudocode, the node needs to execute the shortest path first algorithm $|N(c)|$ times, where $|N(c)|$ represents the degree of node c. Let M represent the number of nodes changed during algorithm execution, L represent the number of edges changed, and P represent the number of priority queue operations. The complexity of i-SPF is $O(M\log M + L\log M)$. Since $M \leq |V|$ and $L \leq |E|$, the total complexity is $O(|E|\log|V|)$.

Theorem 3. Algorithm ERPISPF can compute all next-hop sets that satisfy LFA rules.

Proof. Assume node c runs the algorithm. From lines 1-10, after this portion executes, the algorithm can calculate the minimum cost from all neighbor nodes of c to all other nodes. Once these minimum costs are known, the backup next-hops from node c to all destinations can be computed using LFA rules. Therefore, the theorem holds.

3 Experiments and Results

Since the algorithms for implementing the three LFA rules are similar, this paper only describes the algorithm for implementing the LFC rule, denoted as ERPISPF in the experiments.

3.1 Experimental Datasets

1) Real Topologies
a) **Abilene [27]**: This topology includes 11 routers and 28 links.
b) **Rocketfuel [28]**: Table 1 lists the real topologies published by the Rocketfuel project.

Rocketfuel topology parameters

2) Synthetic Topologies
This paper uses the Brite [29] software to generate synthetic network topologies. The parameters for this software are listed in Table 2.

Brite parameters for topology generation

3.2 Evaluation Metrics

The evaluation metrics primarily include computational overhead and failure protection rate.

1) Computational Overhead
Computational overhead = (Actual runtime of algorithm) / (Actual runtime of Dijkstra' s algorithm).

2) Failure Protection Rate
Failure protection rate = $\frac{\sum_{s,d\in V,s\neq d} k(s,d)}{\sum_{s,d\in V,s\neq d} |V|-1}$, where $k(s,d)$ represents the number of protected next-hops for source-destination pair (s,d).

3.3 Computational Overhead

Figure 2 depicts the computational overhead of ERPISPF, LFC, TBFH, and DMPA on Abilene and Rocketfuel topologies. According to Figure 2, ERPISPF achieves the highest execution efficiency across all tested topologies, followed by DMPA and TBFH, while LFC exhibits the worst performance. This is because

the computational overhead of ERPISPF, DMPA, and TBFH does not vary with network degree. As stated in Theorem 2, ERPISPF' s computational overhead is less than the time required to compute one shortest path tree. DMPA' s overhead equals the time for one shortest path tree computation, while TBFH' s overhead approaches the time for two shortest path tree computations.

[Figure 2: see original paper] Computational overhead of different algorithms on Abilene and Rocketfuel topologies

Figure 3 shows the relationship between network topology size and computational overhead in synthetic networks. The results indicate that the computational overhead of all four algorithms is independent of network topology size, with ERPISPF maintaining the smallest overhead among the four. This occurs because as network topology size increases, both the actual runtime of the algorithms and Dijkstra' s algorithm increase at nearly the same rate, making the ratio relatively constant.

[Figure 3: see original paper] Relationship between computational overhead and topology size

Figure 4 illustrates the relationship between average node degree and computational overhead in synthetic networks. LFC' s computational overhead shows a linear relationship with average node degree, while the other three algorithms are largely unaffected. The reason is consistent with that for Figure 2.

[Figure 4: see original paper] Relationship between computational overhead and average node degree

3.4 Failure Protection Rate

Figure 5 shows the failure protection rates of ERPISPF, LFC, TBFH, and DMPA on Abilene and Rocketfuel topologies. ERPISPF and LFC achieve the highest failure protection rates, followed by DMPA and TBFH. According to Theorem 3, ERPISPF can compute all next-hops that comply with LFC rules. Due to limitations in their implementation methods, DMPA and TBFH cannot completely compute all eligible next-hops.

[Figure 5: see original paper] Failure protection rate of different algorithms on Abilene and Rocketfuel topologies

Figure 6 shows the relationship between topology size and failure protection rate in synthetic networks. Figure 7 shows the relationship between average node degree and failure protection rate. The results demonstrate that LFC and ERPISPF always achieve identical failure protection rates. Figure 7 reveals that failure protection rate exhibits a near-linear relationship with average node degree. This is because as average node degree increases, the number of neighbor nodes per node grows, leading to more next-hops that satisfy LFC rules.

[Figure 6: see original paper] Relationship between failure protection rate and topology size

[Figure 7: see original paper] Relationship between failure protection rate and average node degree

4 Conclusion

To improve the computational efficiency of LFA algorithms, this paper proposes an efficient LFA implementation method based on the incremental shortest path first algorithm. The approach fully leverages the characteristics of incremental shortest path first to cleverly compute, on a single shortest path tree, the minimum costs from the root node' s neighbors to all other nodes, thereby enabling rapid LFA implementation. Simulations on Abilene, Rocketfuel, and Brite-generated topologies demonstrate that ERPISPF' s computational overhead is less than the time required to compute one shortest path tree and can compute all next-hops that satisfy LFA rules.

References

[1] Abishek Gopalan, Srinivasan Ramasubramanian. IP fast rerouting and disjoint multipath routing with three edge-independent spanning trees [J]. IEEE/ACM Trans on Networking, 2016, 24(3): 1336-1349.
[2] Antonakopoulos S, Bejerano Y, Koppol P. Full protection made easy: the dispath ip fast reroute scheme [J]. IEEE/ACM Trans on Networking, 2015, 23(4): 1229-1242.
[3] Xu An, Bi Jun, Zhang Baobao. Failure inference for shortening traffic detours [C]//Proc of International Symposium on Quality of Service. Piscataway, NJ: IEEE Press, 2016: 1-10.
[4] Krist P. Scalable and efficient multipath routing: complexity and algorithms [C]//Proc of IEEE International Conference on Network Protocols, Piscataway, NJ: IEEE Press, 2015: 376-385.
[5] Yang Yuan, Xu Mingwei, Li Qi. Tunneling on demand: a lightweight approach for IP fast rerouting against multi-link failures [C]//Proc of International Symposium on Quality of Service. Piscataway, NJ: IEEE Press, 2016: 1-10.
[6] Elhourani T, Gopalan A, Ramasubramanian S. IP fast rerouting for multi-link failures [C]//Proc of Conference on Computer Communications. Piscataway, NJ: IEEE Press, 2014: 2148-2156.
[7] Narváez P, Siu K Y, Tzeng H Y. New dynamic algorithms for shortest path tree computation [J]. IEEE/ACM Trans on Networking, 2000, 8(6): 734-746.
[8] Narváez P, Siu K Y, Tzeng H Y. New dynamic SPT algorithm based on a ball-and-string model [J]. IEEE/ACM Trans on Networking, 2001, 9(6): 706-718.
[9] Francois P, Filsfils C, Evans J, et al. Achieving sub-second IGP convergence in large IP networks [J]. Computer Communication Review, 2005, 35(3): 35-44.
[10] Narvaez P. Routing reconfiguration in IP networks [D]. Cambridge: Massachusetts Institute of Technology, 2000.
[11] Lakshminarayanan K, Caesar M, Rangan M, et al. Achieving convergence-

free routing using failure-carrying packets [C]//Proc of ACM Special Interest Group on Data Communication, New York: ACM Press, 2007: 241-252.
[12] Francois P, Bonaventure O. Avoiding transient loops during the convergence of link-state routing protocols [J]. IEEE/ACM Trans on Networking, 2007, 15(6): 1280-1292.
[13] Clad F, Merindol P, Vissicchio S, et al. Graceful router updates for link-state protocols [C]//Proc of IEEE International Conference on Network Protocols, Piscataway, NJ: IEEE Press, 2013: 1-10.
[14] Zhang Baobao, Wu Jianping, Bi Jun. RPFP: IP fast reroute with providing complete protection and without using tunnels [C]//Proc of International Symposium on Quality of Service. Piscataway, NJ: IEEE Press, 2013: 1-10.
[15] Enyedi G, Rétvári G, Szilágyi P, et al. IP Fast ReRoute: lightweight not-via without additional addresses [C]//Proc of Conference on Computer Communications, Piscataway, NJ: IEEE Press, 2009: 157-168.
[16] Xu Mingwei, Yang Yuan, Li Qi. Selecting shorter alternate paths for tunnel-based IP fast ReRoute in linear time [J]. Computer Networks, 2012, 56(2): 845-857.
[17] Joel Sommers, Paul Barford, Brian Eriksson. On the prevalence and characteristics of MPLS deployments in the open Internet [C]//Proc of ACM SIGCOMM Conference on Internet Measurement. New York: ACM Press, 2011: 445-462.
[18] Hao F, Kodialam M, Lakshman T V. Optimizing restoration with segment routing [C]//Proc of IEEE International Conference on Computer Communications. Piscataway, NJ: IEEE Press, 2016: 1-9.
[19] Rétvári G, Tapolcai J, Enyedi G, et al. IP fast ReRoute: loop free alternates revisited [C]//Proc of IEEE International Conference on Computer Communications. Piscataway, NJ: IEEE Press, 2011: 2258-2266.
[20] Yang Xiaowei, Wetherall D. Source selectable path diversity via routing deflections [C]//Proc of ACM Special Interest Group on Data Communication, New York: ACM Press, 2006: 159-170.
[21] Lee S, Yu Y, Nelakuditi S, et al. Proactive vs reactive approaches to failure resilient routing [C]//Proc of IEEE International Conference on Computer Communications, Piscataway, NJ: IEEE Press, 2004: 1-9.
[22] Zhong Z, Nelakuditi S, Yu Y, et al. Failure inferencing based fast rerouting for handling transient link and node failures [C]//Proc of IEEE Computer and Communications Societies. Piscataway, NJ: IEEE Press, 2005: 2859-2863.
[23] Wang Junling, Srihari Nelakuditi. IP fast reroute with failure inferencing [C]//Proc of the SIGCOMM workshop on Internet network management. New York: ACM Press, 2007: 268-273.
[24] Kvalbein A, Hansen A F, Čičić T, et al. Fast IP network recovery using multiple routing configurations [C]//Proc of International Conference on Computer Communications Barcelona, Piscataway, NJ: IEEE Press, 2006: 1-11.
[25] Markopoulou M P, Francois P, Bonaventure O, et al. An efficient algorithm to enable path diversity in link state routing networks [J]. Computer Networks, 2011, 55(5): 1132-1149.

[26] Geng Haijun, Shi Xingang, Wang Zhiliang, et al. A hop-by-hop dynamic distributed multipath routing mechanism for link state network [J]. Computer Communications, 2018, 116: 225-239.
[27] Advanced networking research and education, [EB/OL]. [2017-10-28] https://www.internet2.edu/products-services/advanced-networking.
[28] Spring N, Mahajan R, Wetherall D, et al. Measuring isp topologies with rocketfuel [J]. IEEE/ACM Trans on Networking, 2004: 2-16.
[29] Brite, [EB/OL]. [2017-10-20]. http://www.cs.bu.edu/brite/.

Note: Figure translations are in progress. See original paper for figures.