Research on Intra-domain Routing Protection Schemes Based on Software-Defined Networking: Postprint

Authors: Zhang Ju, Geng Haijun

Date: 2018-05-18T00:00:00+00:00

Abstract

Software Defined Networking (SDN) is a novel network architecture that separates the control plane and the forwarding plane. It has been favored by the industry due to its flexibility and controllability. However, current SDN employs shortest-path forwarding for packets, making it difficult to cope with frequently occurring node or link failures in the network. Therefore, to improve the availability of SDN networks, this paper proposes an intra-domain routing protection scheme based on software defined network (RPBSDN). This scheme can compute multiple backup next-hops for each source-destination pair in the network, utilizing the partial order relationship of nodes joining the shortest path tree to ensure that forwarding paths are free of routing loops. Experimental results demonstrate that the scheme not only exhibits low computational complexity, but also significantly improves network availability.

Full Text

Preamble

Title: Research on Intra-Domain Routing Protection Scheme Based on Software Defined Network

Authors: Zhang Ju, Geng Haijun (School of Software Engineering, Shanxi University, Taiyuan 030013, China)

Abstract: Software Defined Networking (SDN) is a novel network architecture that separates the control plane from the forwarding plane. SDN has gained industry favor due to its flexibility and controllability. However, current SDN implementations employ optimal path forwarding for packets, making it difficult to cope with frequent node or link failures in the network. Therefore, to improve the availability of SDN networks, this paper proposes an intra-domain routing

protection scheme based on Software Defined Network (RPBSDN). This scheme can calculate multiple backup next-hops for each source-destination pair in the network, utilizing the partial order relationship of nodes joining the shortest path tree to guarantee loop-free forwarding paths. Experimental results demonstrate that the scheme not only exhibits low computational complexity but also significantly enhances network availability.

Keywords: software defined network; open shortest path first; backup next hop; partial order

0 Introduction

Internet applications are becoming increasingly pervasive, with network scale expanding continuously. Various end-system software applications emerge endlessly, and society' s dependence on the Internet grows deeper. The Internet has been deeply integrated into daily life. However, these end-system applications rely on the underlying network layer and operate above IP, making IP layer routing design critically important. Particularly for certain real-time applications [1,2] such as VoIP, stock trading software, and online gaming, which have stringent requirements for network availability, even brief disruptions can cause significant losses. Therefore, how to protect routing during network failures to prevent service interruption has become a scientifically significant and practically relevant problem.

For network layer routing protection, the academic community has proposed several schemes. ECMP (Equal Cost Multiple Paths) achieves backup path protection by having administrators adjust link costs to create multiple equal-cost shortest paths between source and destination nodes, but research has proven this to be an NP-Hard problem [4]. LFA (Loop-Free Alternates) is a fast reroute scheme proposed by IETF, documented in RFC 5286. Due to its simplicity and ease of deployment, LFA has achieved some commercial adoption, but its protection capability is limited, with studies showing a protection rate of approximately 50% [4]. The Multiple Routing Configurations (MRC) scheme [5] provides routing protection by adding redundant routes directly to routing devices, which can handle single-node routing failures without needing to know the cause of network state changes. The Not-Via non-hop-by-hop forwarding approach uses two types of addresses: when a packet encounters a link failure, it is encapsulated according to the Not-Via address to bypass the failed node/link, then decapsulated at a predetermined node and forwarded according to IP network rules. This scheme involves substantial computation and is not conducive to practical deployment.

Among the aforementioned research schemes, some are complex and difficult to deploy, some require manual configuration by administrators, and while others are easy to deploy, their protection effectiveness is limited. Moreover, all these schemes are distributed, with each node computing routes independently.

SDN [6] provides greater flexibility for network designers and facilitates network configuration by administrators. However, existing SDN implementations still primarily use shortest-path forwarding. When network failures occur, they still require reconvergence and new routing table computation, during which packets may be dropped. As SDN technology evolves, it offers new solutions for network fault handling. Compared with traditional network architectures, SDN possesses stronger advantages for implementing routing protection [7-9] and has promising application prospects. European research has deeply investigated failure recovery techniques in SDN networks. The authors in [10] proposed a controller-scheduled recovery scheme, but this approach has high latency. The authors in [11] proposed adding Fast Failover to OpenFlow for network failure recovery. Related work has formulated the single-failure recovery problem in SDN networks as an integer programming model [12], thereby using heuristic methods to obtain approximate solutions.

Therefore, the algorithm proposed in this paper is implemented at the SDN central node, with the application environment being intra-domain routing protection. Innovatively swapping source and destination nodes, the algorithm achieves routing protection through a specific approach. While the communication between the SDN central node and data planes and flow table generation are not studied in this paper, the proposed scheme requires only minor adjustments to existing SDN protocols to implement routing protection without imposing additional burden on the control node. The core idea is to assign sequence numbers to nodes as they join a destination-rooted tree during construction, forming a partial order relationship among nodes. This partial order is then utilized to construct multiple loop-free paths from source to destination, thereby ensuring correct packet forwarding.

[Figure 1: see original paper] SDN Architecture

2 Intra-Domain Routing Protection Scheme and Algorithm

2.2 Constructing Shortest Path Tree with Destination as Root

This paper utilizes the Shortest Path First (SPF) algorithm to construct a shortest path tree rooted at node v, where node v is the destination node. The algorithm proceeds as follows: starting from the root node, find the shortest paths and adjacent nodes, recording them in a set called the selected path set. Then, using newly selected nodes as starting points, add new paths and nodes to form an expanded set called the candidate path set. From the candidate path set, find the shortest path and add it to the selected path set, then update the candidate path set. Repeat this process until all nodes have been added to the selected path set.

2.3 Constructing Protection Paths

We illustrate the basic concept of constructing protection paths using a simple example. Consider a network topology with destination address c as shown in

[Figure 2: see original paper], containing 6 nodes and 8 links, where the numbers on links represent link weights. Using the algorithm described in Section 2.2, we can construct a shortest path tree rooted at node v. If we record the order in which each node joins the tree during construction, we obtain [Figure 3: see original paper]. The numbers in [Figure 3: see original paper] represent the sequence numbers of nodes joining the shortest path tree, numbered in ascending order of their joining time.

When node e receives a data packet destined for node v, it looks up the next-hop forwarding address from the shortest path tree rooted at v. The lookup rule selects neighboring nodes with sequence numbers smaller than its own sequence number. In this case, these would be nodes d, b, and a, as shown in [Figure 4: see original paper], where dashed lines represent protection paths other than the optimal path. This constructs multiple paths to the destination node. In ascending order of sequence numbers, these are <e,d,v>, <e,b,v>, and <e,a,v>. Since the sequence numbers represent the order of node insertion into the shortest path tree, according to the tree construction method described earlier, this order also corresponds to increasing path cost, i.e., decreasing path priority.

From the actual topology, besides the three paths <e,d,v>, <e,b,v>, and <e,a,v>, there exist other paths from node e to node v, such as through node c. If this were allowed, node c executing the same algorithm might forward the packet back to node e, creating a routing loop. The following proves that the proposed method does not generate routing loops.

2.4 Loop-Free Proof

Theorem: When a packet' s destination address is d, node c can forward the packet to any neighbor node x if and only if nodes c and x satisfy sequence(d,x) < sequence(d,c). If forwarding follows this condition, the packet forwarding process will be loop-free, where sequence(d,x) represents the order of node x joining spt(d), sequence(d,d) = 1, and spt(d) denotes the shortest path tree rooted at node d.

Proof: Assume $p = (u , u \ , \cdots, u)$ is the packet' s forwarding path. Applying the above rule yields sequence(d,u) > sequence(d,u) > $\cdots$> sequence(d,u). Therefore, we can conclude that any two adjacent nodes maintain a strict partial order relationship, and consequently, the packet forwarding process will not contain loops.

2.5 Algorithm Implementation (Pseudocode)

This section details the execution process of the RPBSDN algorithm. Lines 1-4 initialize all nodes' visited attribute to false and cost to infinity, where $C(c,v)$ represents the minimum cost from node c to node v. Lines 5-7 set node c' s visited attribute to true, cost to 0, and initialize the sequence number. Line 8 enqueues node c. Line 10 extracts a node from the queue. Line 11 calculates the node' s sequence number in the tree, where sequence(c,v) represents the

sequence of node v joining spt(c). Lines 13-14 update node v' s visited attribute and cost. Lines 15-22 update the attributes of node v' s neighbors. Lines 24-30 compute the next-hop set for all nodes to node c, where $bn(v,c)$ represents the next-hop set from node v to node c.

Algorithm RPBSDN:

```
Input: Network topology G = (V, E)
Output: Next-hop sets for all source-destination pairs

1:  for each v   V do
2:    C(c,v) ← ϖ
3:    v.visited ← false
4:  endfor
5:  c.visited ← true
6:  C(c,c) ← 0
7:  seq ← 0
8:  Enqueue(Q, <c, 0>)
9:  while Q is not empty do
10:   <v, tc> ← ExtractMin(Q)
11:   sequence(c,v) ← seq
12:   seq ← seq + 1
13:   v.visited ← true
14:   C(c,v) ← tc
15:   for each neighbor u of v do
16:     if u.visited = false then
17:       newdist ← C(c,v) + L(v,u)
18:       if newdist < C(c,u) then
19:         Enqueue(Q, <u, newdist>)
20:       endif
21:     endif
22:   endfor
23: endwhile
24: for each v   V do
25:   for each neighbor u of v do
26:     if sequence(c,v) > sequence(c,u) then
27:       bn(v,c) ← bn(v,c)   {u}
28:     endif
29:   endfor
30: endfor
```

[Figure 3: see original paper] Shortest Path Tree Rooted at Node v
[Figure 4: see original paper] Example of Constructing Protection Paths

2.6 Algorithm Complexity Analysis

The RPBSDN algorithm primarily involves three priority queue operations: Enqueue, ExtractMin, and Decrease-Key. We denote the time required for these operations as T , T , and T respectively, using these operations to maintain priority queue Q. During algorithm execution, Enqueue and ExtractMin are called at most $|V|$ times each, while Decrease-Key is called at most $|E|$ times. Therefore, the algorithm' s complexity is $O(|V| \cdot T + |V| \cdot T + |E| \cdot T)$. When using a Fibonacci heap to implement the priority queue, $T = O(1)$, $T = O(\lg|V|)$, and $T = O(1)$. Thus, the complexity of RPBSDN is $O(|V| \cdot \lg(|V|) + |E|)$.

3 Experiments

This chapter presents experimental simulations of the proposed RPBSDN algorithm, first introducing the experimental methodology and then analyzing results to demonstrate algorithm performance.

3.1 Experimental Method

1) Experimental Topologies: To evaluate the algorithm comprehensively and realistically, we employed multiple topologies:
(a) **Abilene** [13], the real topology of the US education and research network, containing 14 edges and 11 nodes.
(b) **Rocketfuel** project topologies [14], from which we selected five topologically distinct cases: Telstra (108 nodes, 153 links), Ebone (87 nodes, 162 links), Tiscali (161 nodes, 328 links), Exodus (79 nodes, 147 links), and Abovenet (141 nodes, 748 links).
(c) **Brite-generated** topologies [15], using the open-source Brite tool to generate additional topologies with 20-200 nodes for more comprehensive evaluation.

2) Evaluation Metrics: Based on the algorithm' s objectives, we designed two metrics: average next-hop count and computational efficiency. We compared RPBSDN against SDN-SPF [9], which implements OSPF using SDN with shortest-path forwarding.

3) Experimental Setup: We used one PC running the OpenDaylight controller to obtain network topology and compute routes, and another PC running Mininet with Open vSwitch to generate diverse topologies.

3.2 Average Next-Hop Count

Average next-hop count is defined as total next-hop count divided by $(n \cdot (n-1))$, where n is the number of nodes in the topology. By this definition, higher average next-hop count indicates greater network availability.

[Figure 5: see original paper] and [Figure 6: see original paper] show results using Brite-generated topologies. In [Figure 5: see original paper], the average node degree is fixed at 4 while varying node count. In [Figure 6: see original

paper], node count is fixed at 200 while varying average degree. The results show that the average next-hop count is similar to the average node degree, primarily because Brite generates relatively regular topologies with good connectivity. Compared to OSPF-SPF, RPBSDN provides more routing choices, achieving the goal of routing protection.

[Figure 7: see original paper] presents results for Abilene and Rocketfuel topologies. Abilene shows the smallest average next-hop count at 1.4, while Abovenet shows the largest at 5.21, reflecting differences in network topology connectivity. Greater connectivity yields more next-hop options, consistent with RPBSDN' s design objectives.

3.3 Computational Efficiency

To eliminate uncertain factors, computational efficiency is measured by the number of shortest path tree constructions. As shown in [Figure 8: see original paper], OSPF-SPF and RPBSDN exhibit identical computational efficiency across all topologies. This is because in SDN, the controller must compute routing tables for all nodes. Deploying RPBSDN requires executing the RPBSDN algorithm $|V|$ times, yielding complexity $|V| \cdot O(|V| \cdot \lg(|V|) + |E|)$. Similarly, OSPF-SPF requires executing SPF $|V|$ times, with identical complexity. Therefore, implementing both OSPF-SPF and RPBSDN in SDN results in the same computational complexity.

[Figure 8: see original paper] Computational Efficiency of Different Algorithms in Abilene and Rocketfuel Topologies

4 Conclusion

In the Internet, intra-domain routing protocols predominantly use OSPF, which employs Dijkstra' s Shortest Path First (SPF) algorithm. Consequently, for any given destination, only the shortest path is used as the next-hop. When link failures occur, network communication remains impaired until new routes are computed. RPBSDN can generate multiple routes for each destination, increasing network reliability and providing routing protection.

Furthermore, given the rapid development of SDN applications, this paper implements the algorithm at the SDN central node. Leveraging the powerful computational capability of the SDN controller enhances RPBSDN' s advantages. However, due to environmental constraints, the SDN environment simulation in our experiments lacks full realism, which represents future work.

References

[1] Li Qing. Research on Weak Forwarding-Based Internet Routing Availability and Scalability [D]. Beijing: Tsinghua University, 2013.
[2] Zheng J, Xu H, Zhu X, et al. We' ve got you covered: failure recovery with backup tunnels in traffic engineering [C]// Proc of IEEE International

Conference on Network Protocols. 2016: 1-10.
[3] Geng Haijun. Research on Intra-Domain Multi-Path Routing Based on Routing Metrics [D]. Beijing: Tsinghua University, (year missing).
[4] Sridharan A, Guerin R, Diot C. Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks [J]. IEEE/ACM Transactions on Networking, 2005, 13(2): 234-247.
[5] Kvalbein A, Hansen A F, Cicic T, et al. Fast IP network recovery using multiple routing configurations [C]// Proc of IEEE International Conference on Computer Communications. 2007: 1-11.
[6] Zhang Chaokun, Cui Yong, Tang Heyi, et al. Research progress on Software Defined Networking (SDN) [J]. Journal of Software, 2015, 26(1): 62-81.
[7] Sharma S, Staessens D, Colle D, et al. Fast failure recovery for in-band OpenFlow networks [C]// Proc of the 9th International Conference on Design of Reliable Communication Networks. 2013: 52-59.
[8] Hartert R, Vissicchio S, Schaus P, et al. A declarative and expressive approach to control forwarding paths in carrier-grade networks [J]. ACM SIGCOMM Computer Communication Review, 2015, 45(5): 15-28.
[9] Kreutz D, Ramos F M V, Veríssimo P E, et al. Software-defined networking: a comprehensive survey [J]. Proceedings of the IEEE, 2015, 103(1): 14-76.
[10] Sharma S, Staessens D, Colle D, et al. Enabling fast failure recovery in OpenFlow networks [C]// Proc of the 8th International Workshop on Design of Reliable Communication Networks. 2011: 164-171.
[11] Sharma S, Staessens D, Colle D, et al. OpenFlow: Meeting carrier-grade recovery requirements [J]. Computer Communications, 2013, 36(6): 656-665.
[12] Hao F, Kodialam M, Lakshman T V. Optimizing restoration with segment routing [C]// Proc of the 35th Annual IEEE International Conference on Computer Communications. 2016: 1-9.
[13] Advanced Networking for Research and Education [EB/OL]. https://www.internet2.edu/products-services/advanced-networking.
[14] Spring N, Mahajan R, Wetherall D, et al. Measuring ISP topologies with Rocketfuel [J]. IEEE/ACM Transactions on Networking, 2004, 12(1): 2-16.
[15] http://www.cs.bu.edu/brite [EB/OL].

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