Balancing the QOS and Security in Dijkstra Algorithm by SDN Technology

Zhao JinJing[1](✉), Ling Pang[1], Xiaohui Kuang[1], and Rong Jin[2]

[1] National Key Laboratory of Science and Technology on Information System Security, Beijing 100101, China
misszhaojinjing@hotmail.com, lingpang313@yahoo.com

[2] Beijing Space Information Relay and Transmission on Technology Centre, Beijing 100094, China

Abstract. Dijkstra algorithm is widely used in a lot of common network routing protocols. We consider the problem of quality of service (QoS) and the Security features of the network routing area using software defined networks (SDN). The SDN framework enables an efficient decoupled implementation of dynamic routing protocols which could aware the communication network status. In this work we consider the varying delay status of the communication network along with other network security parameters. The routing problem is formulated as a multi-constrained shortest path problem. A new improved Dijkstra algorithm is presented named as QS-Dijkstra. The implement and experiment show that QS-Dijkstra algorithm is able to minimize traffic routing through vulnerable links while satisfying the QoS constraints of the network.

1 Introduction

Dijkstra algorithm is widely used in a lot of common network routing protocols, like OSPF and IS-IS. The main idea of Dijkstra algorithm is how to find a shortest path from a source node to a destination node in a network. So each network link has a cost value to present its status, and this cost is used to calculate the shortest path. In the practice, the link cost is defined as a static cost value in OSPF protocol, as the reference bandwidth divided by interface bandwidth or simply as 1 to reduce the shortest path weight to a hop count. The reason is that it's a very easy way in practice. But as the value of the link cost, it could not cover the feathers and status of the link.

In this work we present a practical way to calculate a more reasonable link cost in Dijkstra algorithm and consider the problem of QoS and the security features of the network routing procedure using SDN technology [1–3]. The SDN framework provides an approach to calculate the shortest path between source and destination based on dynamic link statuses through SDN's high network monitoring capability. A lot of useful link information, like link type, link ownership, interface bandwidth, transition delay and historical record, can be collected and computed by the SDN controller to enable more safe, reliable and efficient paths. In this way, we can consider the varying delay status of the communication network along with other network security parameters and get a presence of a passive/active adversary in the network routing area.

F. Zhang et al. (Eds.): NPC 2018, LNCS 11276, pp. 126–131, 2018.
https://doi.org/10.1007/978-3-030-05677-3_11

The remainder of this paper is organized as follows: double constrained shortest path problem is discussed and the derivation of QoS constraints and related cost metrics are presented in Sect. 2, the implementation details are provided in Sect. 3, Sect. 4 investigates the performance of the proposed framework. Conclusions and final remarks are discussed in Sect. 5.

2 System Model

Consider a graph representation of the communication network. *G(V, E, ω)* is a weighted undirected graph model and describes an N nodes and E links network. The node set is $V = \{v_1, \ldots, v_N\}$, and the edge set is $E = \{e_{ij} | i, j = 1, 2, \ldots, N\}$. The weight ω_{ij} on the edge e_{ij} is defined as the cost of the link. In this article, the interplaying between QOS and security features is concerned in the network routing process. The security metrics of the link between nodes i and j could include these features as:

(1) History $L_{ij}H$: a link that was previously targeted by an attacker in a particular time could be more likely to be attacked again.
(2) Security installed measures $L_{ij}S$: a link with high encryption is typically hard to be listened or hijacked. So $L_{ij}S$ values are dependent on the pre-installed and pre-configured security measures of nodes of the link.
(3) Bandwidth $L_{ij}B$: A link with high bandwidth is more difficult to be congested by data flow.
(4) Ownership $L_{ij}O$: a self-owned or in the same domain channel is more secure than a shared or leased channel by other domains.

The vulnerability metric $L_{ij}M$ should reflect the attributes that make a link more security.

$$L_{ij}M = L_{ij}H * \left(\alpha L_{ij}S + \beta L_{ij}B + \gamma L_{ij}O\right) \tag{2}$$

Where α, β, and γ are the weights of $L_{ij}S$, $L_{ij}B$ and $L_{ij}O$ depending on the impact importance of the considered security parameters.

Assume every link $e_{ij} \in E$ has two weights $c_{ij} > 0$ and $d_{ij} > 0$ (c_{ij} is cost and d_{ij} means delay). For source and destination nodes *(s, t)*, let P_{st} denote the set of paths from s to t. Further, for any path p define

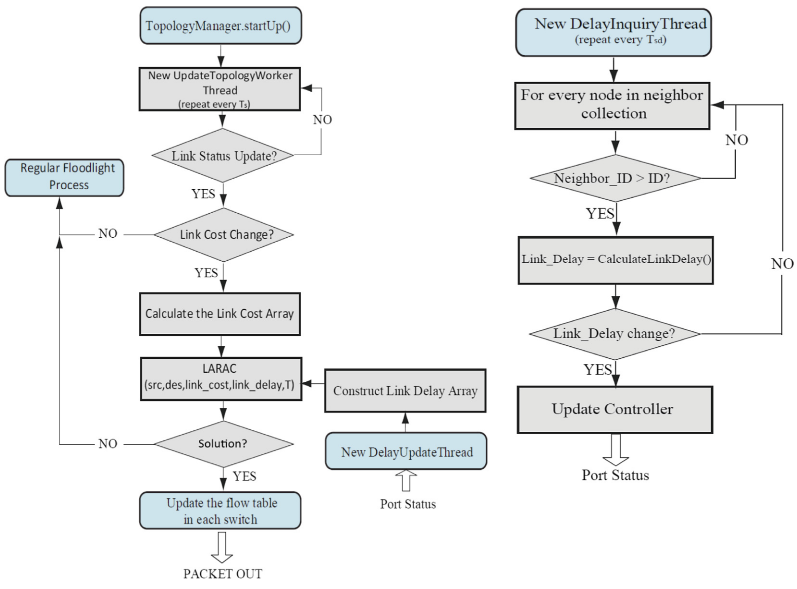
$$\mathrm{c(p)} = \sum\nolimits_{(i,j) \in p} L_{ij}M \tag{3}$$

$$\mathrm{d(p)} = \sum\nolimits_{(i,j) \in p} d_{ij} \tag{4}$$

The routing problem seeks to find the paths between s and t nodes with minimum link cost $c\ (p_{st})$, which satisfies $d(p_{st}) \leq T_{max}$. This is a typical NP problem named constrained shortest path (CSP) [4, 5], which can be solved by the Lagrangian Relaxation Based Aggregated Cost (LARAC) algorithm [6].

3 Implementation

The architecture of SDN network comprised of Floodlight controller and Mininet switches. In the floodlight controller, applications can be written in Java and can interact with the built-in controller modules via a JAVA API. Other applications can be written in different languages and interact with the controller modules via the REST API. And the controller allows the implementation of built-in modules that can communicate with their implementation of the OpenFlow controller (i.e. OpenFlow Services). The controller, on the other hand, can communicate with the switches via the OpenFlow protocol through the abstraction layer present at the forwarding hardware.


TopologyManager.startUp()
New UpdateTopologyWorker Thread (repeat every T_s)
NO
Link Status Update?
YES
Regular Floodlight Process
NO
Link Cost Change?
YES
Calculate the Link Cost Array
LARAC (src,des,link_cost,link_delay,T)
Construct Link Delay Array
NO
Solution?
YES
New DelayUpdateThread
Port Status
Update the flow table in each switch
PACKET OUT
New DelayInquiryThread (repeat every T_{sd})
For every node in neighbor collection
NO
Neighbor_ID > ID?
YES
Link_Delay = CalculateLinkDelay()
NO
Link_Delay change?
YES
Update Controller
Port Status

a) The Controller Side b) The Switches Side

Fig. 1. QS-Dijkstra algorithm implementation. The algorithm is separated into two parts; a controller function which is implemented in Floodlight using Java, and a switch function implemented in Mininet using Python.

We propose a Vulnerable-Link Avoidance Dijkstra (QS-Dijkstra) algorithm to capture the problem of best-effort avoiding vulnerable links while maintaining the delay constraint. QS-Dijkstra algorithm uses the previously-defined vulnerability metric in Eq. (2) to arrive at a set of feasible paths between source node s and destination node t.

The flowchart of the QS-Dijkstra algorithm that is implemented is shown in Fig. 1. The algorithm is separated into two parts, the switch side and the controller side. The algorithm of the controller side performs the following tasks:

(1) Listening to messages from switches and calculating *link-delay* value of each link, and then constructing the *link-delay* cost matrix.
(2) Calculating the *link-vulnerability* cost matrix according to the metric developed in formula (2); this matrix can be modified and calibrated by network operators or managers.
(3) Running a *topology-update* thread, and checking the *link-vulnerability* cost matrix updates every T_s; if a change is detected, the controller recalculates the routing paths.
(4) Calculating the routing paths based on the link cost metrics of interest, and updating the flow table of each switch by advertise a PACKET OUT message to switches.

The main function of the algorithm in the switches side is to collect the values of *link-delays* for the directly connected switches. This is done through an independent thread responsible for periodically testing the link between that switch and all connected switches with higher ID. The sampling time is parametric and is tuneable by the network managers; in our simulation environment, *Tsd* is set to 60 s. Link delay testing is done 3 times every *Tsd* and the average value is then compared with the last known value. If the new delay is significantly different from the previous value, the switch updates the controller accordingly.

4 Simulation and Results

We build two large scale network environment with the same topology and route information. One is running the Dijkstra routing protocol, the other is for QS-Dijkstra. In order to reach a high performance, in each environment, we use the high-performance workstation with 10 Intel Xeon Westmere EP six-core processors. Whose maximum process speed could reach 11.251Tflops. Thus, the whole network includes 260 routers and a controller. For every node, we pick a random number from [1, 10] for its connection number. And the commercial network flow generator Spirent TestCentre is chosen to generate some popular network application data, like http, IP, TCP, UDP. And it sends the same packets to the two networks synchronously.

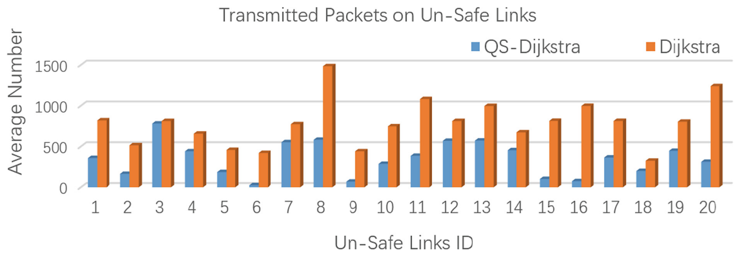


Fig. 2. The number of transmitted packets on un-safe links. There are 20 links which have very high vulnerable level.

In this test case, the link cost and link delay is randomized in the [0–100]. The maximum path delay constraint T still set as 1000 s. After calculating the link vulnerability metric, there are 20 links which have very high vulnerable level. We sampled the packets number transmitted through these links every 200 s and calculate the average value on each links. The result is shown in Fig. 2, which shows that the packets transmitted on these un-safe links in QS-Dijkstra are much less than in Dijkstra.

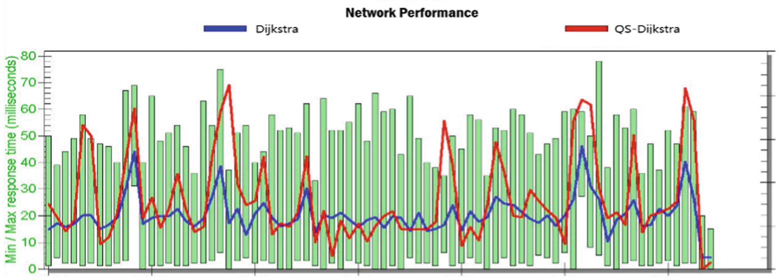


Fig. 3. The average response time of routers. In order to evaluate the network performance in the two networks, the average response times are recorded every 15 s.

From the results shown in Fig. 3, the conclusion could be proved that the network performance in QS-Dijkstra does not lost much except for a few short intervals, and the maximum responds time in these intervals is still could be acceptable.

5 Conclusion

In this paper, we consider the varying delay status of the communication network along with other network security parameters. Our approach capitalizes on the SDN framework and technology. The implement and experiment show that QS-Dijkstra algorithm is able to minimize traffic routing through vulnerable links while satisfying the QoS constraints of the network.

In the future work, the algorithm could consider more security and performance features of links and routing nodes, to make a more effective routing protocol.

References

1. Nunes, B., Mendonca, M., Nguyen, X.-N., Obraczka, K., Turletti, T.: A survey of software-defined networking: past, present, and future of programmable networks. IEEE Commun. Surv. Tutor. **16**, 1617–1634. https://doi.org/10.1109/surv.2014.012214.00180.pdf
2. Software-Defined Networking. http://en.wikipedia.org/wiki/Software-defined_networking
3. Open Networking Foundation: Software-defined networking: the new norm for networks. ONF White paper (2012)

4. Xiao, Y., Thulasiraman, K., Xue, G., Juttner, A.: The constrained shortest path problem: algorithmic approaches and an algebraic study with generalization. AKCE Int. J. Graphs Comb. **2**, 63–86 (2005)
5. Kuipers, F., Van Mieghem, P., Korkmaz, T., Krunz, M.: An overview of constraint-based path selection algorithms for QOS routing. IEEE Commun. Mag. **40**, 50–55 (2002)
6. Jüttner, A., Szviatovski, B., Mécs, I., Rajkó, Z.: Lagrange relaxation based method for the QOS routing problem. In: Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 2, pp. 859–868 (2001)