ISBN 978-981-11-0008-6
Proceedings of 2016 6th International Workshop on Computer Science and Engineering (WCSE 2016)
Tokyo, Japan, 17-19 June, 2016, pp. 236-242
doi: 10.18178/wcse.2016.06.037

Real-Time Packet Traceback Based on Intent for Software-Defined Networks

Chen Yunfang+, Jiang Kuan, Zhang Wei and Liu Qi

Nanjing University of Posts and Telecommunications, China

Abstract. Tracing back packets intentionally is a sub-issue of packet traceback, which is useful for network debugging, network forensics and performance testing. However, current mechanisms for classifying packets with IP addresses, port numbers and other low-level features of Software-Defined Networking (SDN)flows, rather than with abstractions map to the intent, such as people, applications and devices. In this paper, we present the basic idea for intentional packet traceback that satisfies users' intents or goals and a preliminary design and implementation that can work in a real-time environment. A significant improvement is designing an interface that defines an SDN controller as intent, and using the intent to collect packet as input variantfor computing back policy on the controller. We present an example that show how these goals can be achieved from modular component to iteratively traceback and discuss the real-time feature of our proposed.

Keywords: real-time intent, softwaredefined networking, traceback.

1. Introduction

With recent worsening of denial of service (DoS) attacks, packet traceback technology has developed rapidly. According to the tracing level of packets, we can divide packet traceback into two methods, one bases on network layer and the other bases on data link layer [1]. Traceback bases on IP layer is the most studied can be divided among four categories: link test method, log record method, ICMP-based method and packet marking method. The top three methods have too many constraints, and the last method by marking packets take advantages over tracing afterwards and in-band transmission, which is the main current method within the field of packet trackback. However, the traditional packet traceback also has a lot of problems [2], [3]. Reconstructing path or locating the source IP address in network attack slowly.

Software-defined Networking (SDN) is a new network architecture [4]-[6]. By separating the forwarding plane and control plane, SDN is designed to simplify network management, support network innovation and evolution. The logically centralized controller has complete knowledge of the global network, supporting flexible software programming, making an unprecedented improvement in automated management and control capacity.

In SDN networks, a variety of controller applications have been developed to take network management tasks because the controller can get current network policy in realtime and supply an unified interface for programmers. For SDN-based traceback, we have also made several different attempts in section 2. This method not only increases the processing time of packets, but also wastes lots of computing resources of switches in section 3. We also reconstruct traceback path by computing forwarding policy reverse, and from the starting point of tracing, each computing step leads to several next-hop addresses until back to the source. These traceback results depend on the particular input packet, a clear intent (e.g. applications or services) needs to be defined in section 4. Our attempt is to filter packets with the user intent, and tracing the packet

+ Corresponding author: Chen Yunfang Tel.: + 8613913869037.
E-mail address: chenyf@njupt.edu.cn.

that meets the specific intent, which is an important improvement over the current SDN traceback method in section 5.

2. Related Work

In research of SDN traceback, early work has been done and achieved some results. NetSight [7] proactively collects historical metadata information of packets, and the information can be sent to the network operators for network debugging, packet traceback, network monitoring, hierarchical network profiler. NDB[8] provides a SDN debugging tool, creates a postcard for each packet, which contains the packet header, matched flow rules, switch information and forwarding ports. Another packet traceback way just bases on the control plane, we use the forwarding policy to calculate "back policy"[9] and reconstruct the traceback path by iterative applying the back policy. Here we introduce the definition and use cases of back policy. Given a policy P and any packet x and y, we define P^{-1} formally as the policy for which

$$\mathrm{y} \in P(x) \Leftrightarrow x \in P^{-1}(y)$$

It means that for any packet y in the set of packets output when the policy P is applied to input packet x, it must hold that x is also in the set of packets output when P^{-1} applied to packet y.

For sequential (M·N) and parallel (M + N) components，their back policy are $N^{-1} \cdot M^{-1}$， $M^{-1} + N^{-1}$.

SDN monitoring is another hot issue. Techniques on monitoring can also be used in packet traceback. The conventional concept of network monitoring collects network traffic based on the low-level features (e.g. ports, IP addresses). NetAssay[10] introduces intents (people, applications, devices) to SDN monitoring system and solves the problem of conventional monitoring by using high-level abstraction intents. Each intent corresponds to a metadata engine running on the controller, by which we can take actions (e.g. create, update, maintain) on the intent. The system can collect packets based on specified high-level feature's intent.

In this paper, we introduce intentional naming technology into the traceback system, by matching intents at the observed node, so that the controller can identify the intents [11](e.g. peoples, devices, applications) which defined by the user, and compared traceback system with no intent's ones, users do not require knowledge of packet analysis capabilities.

3. Design Framework

The current SDN traceback system provides the idea of reconstructing a packet's path back to the source address based upon the forwarding path, which is limited to an inefficient way and does not meet the needs of programmers and operators. The intentional goal of traceback system design is to provide an interface to customers and enable them to extend the model based on intent to achieve customized tracing purposes. Then, we present how to define a general intent and explain that several aspects need special consideration.

3.1. Design Goals

Our design goal of traceback systems is intentional, lightweight and real-time. When a packet enters the traceback node, we do not immediately start tracing back. With the automatic packet header matching of intent policy, network operators can filter the packet, and determine whether it is should be traced. While in a real network, a large number of packets pass through the traceback node, it is necessary to ensure efficiency and accuracy, the difficulty of manual matching of packet IP addresses, port number is unimaginable.

Next we will divide intentional tracing into three processes and describe them in detail one by one, (a) the controller creates the intentional policy and installs the equal flow rules onto the switch (b) the packet passes through the traceback node, if matched to the intentional policy, the packet forwarded to the target server and sends to the controller as well (c) the controller comes out the packet path by iterative computing back policy.

In the process one, the network operator defines the intent which bases on customer’s purposes, such as network service's providers and content providers, which provide customers with a variety of services, so that for service means the customer's different intent. Because the network resources have Universal Resource Locator (URL) to identify, it is convenient to make the user's intent define as a set of URLs. After the intent being created, we need to use Domain Name System (DNS) module for resolving to IP addresses, define intent on network devices, resources and services. By using the DNS module we have created intention Policy (e.g. people, services, devices) into IP addresses, port numbers, and transform to flow rules. DNS resolution module is responsible for the intents creating, updating and maintaining, not require user intervention.

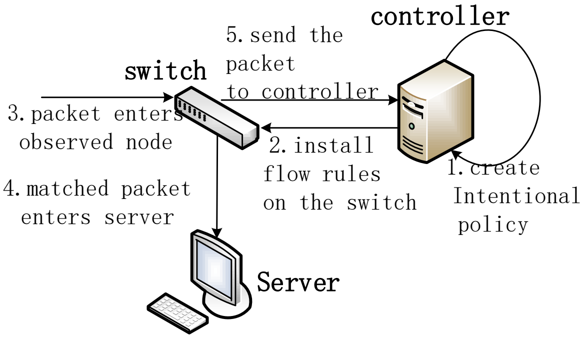
controller
5. send the packet to controller
switch
3. packet enters observed node
2. install flow rules on the switch
1. create Intentional policy
4. matched packet enters server
Server

Fig. 1: Framework of intentional traceback.

In the process two, DNS resolution module makes the flow rules install onto the switch. Therefore, all packets once pass through the switch and matched by intentional policy one by one, if success matched, then the packet is forwarded to the target server. At the same time, the switch will send the matched packet to the controller.

In the process three, the controller calculates the network topology policy, which is the whole links policies' set. After packet arrivals at the controller, and as an input parameter for calculation of back policy, and obtain location information of a previous hop, so by iterative calculating until the packet is discarded or drawn the entire path.

3.2. Interface of Intentional Traceback

Traceback application of the controller can also be provided as a service to the network operator, added intent is used to give network operators the advanced feature's abstraction of packet matching. In order to meet the diverse needs of network operators and adapt to the complex network environment, we add the intentional defined policy so that network operators can use to express people, application, devices on the intentional level according to customize needs, not just for the underlying network IP addresses, port numbers.

We implements the interface based on Pyretic [12]-[14] runtime environment provides network operators with the syntax framework for taking direct action on the virtual header of packets [15]. It is also convenient to program by extending a modular interface, so that network operator uses primitives (e.g. drop, fwd, flood) to change packet forwarding behaviors. Located packet [16], [17] enables operators to do something like the database query "groupby " operation, for that we make the runtime system compile operations into flow rules that switches understand and install them onto switches.

Table 1: Intentional Interface

Intent feature	**domain name**	**IP Address**
Controller	c0.controller.org	10.10.1.9
Application/Service	video.baidu.com	180.149.133.169, 180.149.133.159 180.149.133.139
Device Type	m.baidu.com	180.149.133.169

Controller intent feature. In the intentional traceback system, controller is the most important role, responsible for creating intentional policy and computing traceback policy. In a single controller environment, the problem is relatively simple. We make the intent of the policy create correctly, accurately transform to flow rules, communicate properly between switches and controller can enable network operators to complete the intentional traceback. We resolve unified identity issues by controller programmable interface. The intentional policy can distinguish the characteristics of the controller, and certification center can solve the authentication problem as well. So we can give the controller an identity that a certified IP address with the domain name "c0.controller.org". Supposing that the controller has authenticated network operators, so we believe that the controller is safe not only determined by the IP address, but also by the domain name. Therefore, by the definition of such intention "tracing the packet from controller C0", we can achieve intentional traceback goal. Tracing the packet from other controllers will help operators quickly debug networks, this tracing is limited in the range of known topology of the controller, tracing across the controller requires co-sponsored by different operators.

Application or service intent feature. Compared with controller features, the application features more diverse. For different services, such as video, music, file download, we need more detailed classification. However, emerging network applications still need to be distinguished. For some services, such as video service, how to define it as intentional policy, is it true that whoever visits video.baidu.com equals to use the video service of the website of baidu? Of course not, potential users may be search, download files. We can’t determine their behavior at a moment. But the fact is that the user who visited the website of video.baidu.com, we know that the main intent of the user is about the video, but not for the file searching, because if we just want to check, clearly search engine is more useful. So that video applications intent is defined as "packet requests the website of video.baidu.com" in this broader sense.

Device Type intent feature. The diversity of device type in networks exists is in any case. Users may own equipments such as a mobile phone and carry with them anywhere, and then connect to the Internet and use different services. Most mobile network devices (e.g. mobile phones, tablet, PDA) can request services through the web browser if they have one. For the service's server, which determines whether the packets from the mobile device, by analyzing clients’ requested http header and responses with web pages displayed to the user. Here, if we analyze the results of the visit, it can be inferred "packet from the URL of mobile.baidu.com the same as from the mobile device." We define a mobile device feature as "From URL mobile.baidu.com the data packet from the mobile device". Although we do not know a certain mobile device, whether belongs to a user or not, we still can define mobile device intent broadly as "the packet responses from URL of mobile.baidu.com". As mentioned above, if from the user's perspective, tracing packet from the mobile device to the source user is very difficult. However, from the perspective of the network operator, only tracing at observed node for "tracing packet's response from the website of mobile.baidu.com" is relatively easy to implement, studying the behavior in a wider range of packet traceback is more valuable, also for the intentional traceback.

4. Implementation of Intentional Traceback Interface

According to the goal and interface of intentional traceback described above, we need to design a runtime environment to support our defined intentional policy. Runtime environment [17] is also responsible for compiling high-level policy and transforming to flow rules, which installed onto the switch. Fortunately, Pyretic [13], [14] provides us with a runtime environment. We test our intentional packets traceback policy with a variety of topologies in Mininet [18]. This chapter describes from modular programming, topology strategy to be iterative back with implementation details of intentional traceback.

4.1. Modular Component

According to the previous definition, we present that people visit video website “video.baidu.com” as application's intent, or open port number 80 as http service intent. It is clearly that video website not contains video.baidu.com only, but here we suppose that video application intent specifies video.baidu.com, so we can consider packets that matched source IP address correlative with video.baidu.com as a video intent “packet requests the website of video.baidu.com”.

Depending on DNS resolution modular of netAsssy[10], our components also divided into two parts, the metadata engine and control application's component. Added metadata engine component is managed by the control application's component, extending pyretic runtime environment to support a variety of intentional policies. For example, assume that we have three servers to provide video services: 180.149.133.169, 180.149.133.159, 180.149.133.139, and they share the same URL website in order to provide high-quality services.

Table 2: Flowrules Translate

Intent policy	Pyretic policy
Match(service=’video’)>>Route ()	(match(srcip=180.149.133.169)+ match(srcip=180.149.133.159)+ match(srcip=180.149.133.139)) >> forward(3)

First, we extend metadata engine to identify the intent, specify the collection of the video URL website to support the creation of video type of intent and maintain intentional policy. Principle is to create a video intent to support metadata engine. Policy will be transformed into flow rules sent to the designated switch, later when the user makes access to video URL, and DNS request's packet to arrive at the switch. It will automatically match a defined action carried out.

Then modify the control application to support the expansion of the metadata engine. When the intentional policy is created, the switch can use the advanced features within the intentional definitions to match and forward packets, while physical conditions change, such as IP addresses, port numbers, the metadata engine is responsible for updating the current advanced features in real-time.

4.2. Topology Policy

Three elements of the network objects: topology, policy, and derived network mapping. By using function sets of topology and policy provided by pyretic, we can easily simulate Ethernet gateways, routers and other network middleware. In the intentional traceback model, we use the convenient generation network functionality [19], test the model in different network environments.

Using link policy, we distinguish forward policy with topology policy easily, packet forwarding on the link from the switch b to as a, so we can define link policy:

$$L_{ba} := [s = b] \cdot [o = 2] \cdot (s \leftarrow a) \cdot (i \leftarrow 2) \cdot (o \leftarrow \phi)$$

The topology policy is the summation for all network link’s policy:

$$T := \sum_{(a,b)\in S} L_{ab}$$

Relations between policy and topology can be expressed as the P applied to the T (as P·T). It means that a packet leaves one switch port, the port of the next hop switch for the packet to go, which based upon the policy. So that we can control not only for packet forwarding at switches but also for packet transmitting on the link.

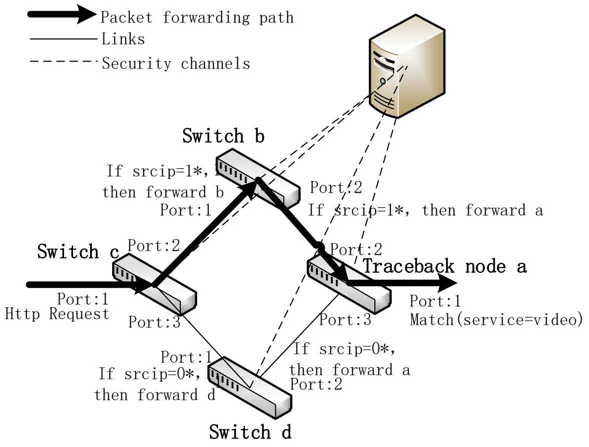

Fig. 2: Topology of intentional traceback.

4.3. Iteratively Traceback

From the Figure 2, according to the forwarding policy on each switch, you can quickly obtain a global forwarding P, global forwarding policy is to use the parallel composition operator "+" to all the single forwarding policy. For example, if a packet enters the switch c, according to the forwarding policy of switch c, there are two corresponding entries, assume that the source IP address of the packet is 1 *, it is forwarded to the port number of 2 of the switch c.

Back policy P^{-1} determines the traceback forwarding rules, given by any intentional policy matched packet x, into a port of a switch will trace back on all ports of the same switch. The next observation status of forwarding packet x on port number 2 of the switch c, because the link (a, b) has link policy [9] defined, when the packet passes through the link (a, b), left from the port number 2 of switch a enters port number 1 and number 2 the switch b. This is because the switch b has two ports, and link policy is not clear enough to determine either port, two ports need to be traced both.

$$L_{ba}^{-1} \triangleq [s = a] \cdot [i = 2] \cdot (s \leftarrow b) \cdot (o \leftarrow 2) \cdot \big((i \leftarrow 1) + (i \leftarrow 2)\big)$$

In this iteration on the packet switch b has two copies of packet x, then forwarding policy based on back policy P^{-1}, matching source address of 1 * After only from port number 2 of the switch c incoming packets will be from the port number 1 of the switch c continues back down, and from port number 1 incoming packets because there is no match and is discarded. For the same reason, L_{bc}^{-1}will also continue to replicate packet x with three copies, switch c back to port number 1, 2, 3 cannot match because the port forwarding rules to be discarded, back to the port number 1 of the switch.

5. Conclusion

In this paper, we have designed and implemented an intentional traceback system. We define different intents and tracing back by the controller to improve the practicality. The design goal of traceback system is the definition of intent and encapsulates the process as an interface to provide the convenience for programmers to extend. Customers' intent is defined as the set of websites' URL. Using intentional abstraction of matching the websites, the controller creates intentional policy and installs flow rules onto switches, and the users don't have to trace packet directly by IP addresses, port numbers. According to the inverse computing of forwarding policies, calculation of back policy without need to waste data plane resources. The intentional policy is created and maintained by the controller application, and corresponding flow rules installed to switch, while the network topology changing, intentional policy is in real-time been maintained, matched packet after successfully sending to the controller, automatically starts tracing process. The whole process does not require users' intervention, has the feature of real-time.

In our ongoing work, we will focus on three areas. First, we implement intentional traceback by definition of the simple intent, how to achieve more sophisticated functions of intentional traceback, such as "tracing malicious traffic", the need to combine the use of detection component of malicious traffic. Secondly, the intent require the users defined themselves, and network websites set can be changed arbitrary. Malicious traffic will be detected automatically and categorized by the controller application, the component with self-learning function can better help users complete the definition of the intent. Simple tasks can be done automatically depends on needs of users'. Finally, when the user's needs change, how to make sure the newly changed intent could work together with previously defined ones. Effective mechanism of collision detection of intent can help users solve the problem of changing an intent. In summary, our future is to achieve a practical, intelligent, scalable traceback program.

6. References

[1] A. Belenky, N. Ansari. On IP traceback[J]. Communications Magazine, IEEE, 2003, 41(7): 142-153.

[2] S. Savage, D. Wetherall, A. Karlin, et al. Practical network support for IP traceback[C].ACM SIGCOMM Computer Communication Review. ACM, 2000, 30(4): 295-306.

[3] A. C. Snoeren, C. Partridge, L. A. Sanchez, et al. Hash-based IP traceback[C].ACM SIGCOMM Computer Communication Review. ACM, 2001, 31(4): 3-14.

[4] D. Kreutz, F. M. V. Ramos, V. P. Esteves, et al. Software-defined networking: A comprehensive survey[J]. Proceedings of the IEEE, 2015, 103(1): 14-76.

[5] W. Xia, Y. Wen, C. H. Foh, et al. A survey on software-defined networking[J]. Communications Surveys & Tutorials, IEEE, 2014, 17(1): 27-51.

[6] W. John, K. Pentikousis, G. Agapiou, et al. Research directions in network service chaining[C].Future Networks and Services (SDN4FNS), 2013 IEEE SDN for. IEEE, 2013: 1-7.

[7] N. Handigol, B. Heller, V. Jeyakumar, et al. I know what your packet did last hop: Using packet histories to troubleshoot networks[C].Proc. USENIX NSDI. 2014.

[8] N. Handigol, B. Heller, V. Jeyakumar, et al. Where is the debugger for my software-defined network?[C].Proceedings of the first workshop on Hot topics in software defined networks. ACM, 2012: 55-60.

[9] H. Zhang, J. Reich, J. Rexford. Packet Traceback for Software-Defined Networks[J]. Technical Report TR-978-15, Princeton University, 2015: 1-7.

[10] S. Donovan, N. Feamster. Intentional Network Monitoring: Finding the Needle without Capturing the Haystack[C].Proceedings of the 13th ACM Workshop on Hot Topics in Networks. ACM, 2014: 5.

[11] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, et al. The design and implementation of an intentional naming system[C].ACM SIGOPS Operating Systems Review. ACM, 1999, 33(5): 186-201.

[12] C. Monsanto, N. Foster, R. Harrison, et al. A compiler and run-time system for network programming languages[J]. ACM SIGPLAN Notices, 2012, 47(1): 217-230.

[13] C. Monsanto, J. Reich, N. Foster, et al. Composing Software Defined Networks[C].NSDI. 2013: 1-13.

[14] N. Foster, A. Guha, M. Reitblatt, et al. Languages for software-defined networks[J]. Communications Magazine, IEEE, 2013, 51(2): 128-134.

[15] C. J. Anderson, N. Foster, A. Guha, et al. NetKAT: Semantic foundations for networks[J]. ACM SIGPLAN Notices, 2014, 49(1): 113-126.

[16] N. Foster, R. Harrison, M. J. Freedman, et al. Frenetic: A network programming language[C].ACM SIGPLAN Notices. ACM, 2011, 46(9): 279-291.

[17] J. Reich, C. Monsanto, N. Foster, et al. Modular sdn programming with pyretic[J]. Technical Reprot of USENIX, 2013.

[18] B. Lantz, B. Heller, N. McKeown. A network in a laptop: rapid prototyping for software-defined networks[C].Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. ACM, 2010: 19.

[19] N. Kang, Z. Liu, J. Rexford, et al. An efficient distributed implementation of one big switch[J]. open Networking Summit, April, 2013.