Tunneling over IP Based on Match-Action Table in Software Defined Networks

Keyao Zhang, Jun Bi, Yangyang Wang, Yu Zhou, Zhengzheng Liu
Institute of Network Science and Cyberspace, Tsinghua University
Department of Computer Science, Tsinghua University
Beijing National Research Center for Information Science and Technology(BNRist)
zhang-ky15@mails.tsinghua.edu.cn,junbi@tsinghua.edu.cn

ABSTRACT

Tunneling over IP has been widely used in the field of network virtualization, overlay network, heterogeneous network, and so on. Nonetheless, there exists maintenance difficulty, management complexity, low efficiency in tunneling. SDN provides open and unified APIs, which greatly enhances the network management efficiency. However, as a significant southbound interface, OpenFlow does not primitively support the establishment of IP tunnels, therefore it still relies on traditional manual configurations. In this paper, we adopt Match-Action Table (MAT) programming model and propose a new IP tunnel mechanism, called MAT tunnel. The MAT tunnel can encapsulate and decapsulate packets directly by installing flow rules instead of manually configuring tunnel ports. We implement the MAT tunnel prototype based on Open vSwitch and Floodlight. We also construct a simulation environment based on a real topology. Comparing traditional tunnels, the MAT tunnel can reduce the average delay while improving the programmability and flexibility. In addition, the tunnel path switching tests suggest the MAT tunnel can significantly decrease the delay jitter and throughput loss.

CCS CONCEPTS

• **Networks → Programmable networks**;

KEYWORDS

Software Defined Networks, OpenFlow, Tunneling over IP, Overlay Network, Open vSwitch

ACM Reference Format:
Keyao Zhang, Jun Bi, Yangyang Wang, Yu Zhou, Zhengzheng Liu. 2018. Tunneling over IP Based on Match-Action Table in Software Defined Networks. In *CFI 2018: The 13th International Conference on FutureInternet Technologies, June 20–22, 2018, Seoul, Republic of Korea.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3226052.3226054

1 INTRODUCTION

Tunneling over IP is an important networking approach and has lots of significant usages in various networking environments. For example, it is used for network virtualization in data centers in order to isolate network resources and tenants [1]. We can also construct overlay network via IP Tunneling on the Internet for performance improvement [4, 5]. In addition, Tunneling over IP has been widely used to connect heterogeneous network domains (e.g., ICN, IPv6). However, these tunnels are usually created and maintained via manually configurations on network devices, which is complex, inconvenient and error-prone.

With decoupled control and data plane, Software Defined Networks (SDN) enable a flexible and efficient paradigm of network management [3]. In the last few years, SDN have been widely applied in campus, enterprise, and data center networks. The establishment and management of tunnels is an important requirement of many applications in SDN (e.g., SD-WAN). However, OpenFlow [7], which is the most influential instance of the SDN architecture, only supports tag-based tunneling (e.g., MPLS). Tunneling over IP is not defined in the specification of Openflow switch. As a result, OpenFlow has many restrictions on network application, function and scalability in terms of tunneling.

In order to provide the function of IP tunneling, data plane which supports OpenFlow usually adopts additional programming interfaces. For example, in Open vSwitch (OVS) [9], a well-known software switch, OVSDB [8] is proposed for managing tunnels. But these interfaces are different on different targets. So tunneling over IP is not actually simplified in SDN, suffering from maintenance difficulty, management complexity, and low flexibility.

Inspired by the Match-Action Table (MAT) programming models in OpenFlow, we argue that expressing tunneling logic with the MAT model could improve the programmability and flexibility. We propose a mechanism of tunneling over IP based on MAT in SDN, called **MAT tunnel** and implement the MAT tunnel prototype, including VxLAN [6] and GRE [2] tunnels.

Our key contributions are as follows:

- We introduce a mechanism of tunneling over IP based on Match-Action Table. Our MAT tunnel allows controller to install table entries specifying the encapsulation/decapsulation actions and parameter of the tunnel rather than any explicit tunnel configuration interface. (Section 2)
- We implement the MAT tunnel prototype based on Open vSwitch and Floodlight. (Section 3). And we also construct a simulation environment based on a real topology and compare the MAT Tunnel with tunnels configured by OVSDB in terms of the delay and throughput (Section 4)

2 DESIGN OF MAT TUNNEL

The overall design of MAT Tunnel is illustrated in Figure 1. The data plane consists of OpenFlow switches which need to be extended to

CFI 2018, June 20–22, 2018, Seoul, Republic of Korea

ACM ISBN 978-1-4503-6466-9/18/06…$15.00
https://doi.org/10.1145/3226052.3226054

support the MAT tunnel function. We extend the *Match* and *Action* Fields in *OpenFlow Flow_Mod* messages so that the controllers can install flow entries about MAT tunnel on the switches. We also provide RESTful API on controllers for network applications and administrators, which makes it easier to create or remove the MAT tunnel.

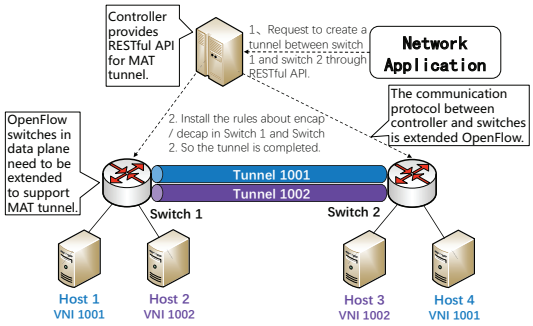


Figure 1: Overview of MAT tunnel.

However, there are some challenges in the design:

(1) How can we implement tunneling over IP through Match-Action Table? And how does it work in the switches? Moreover, we should explore how to complete tunnel encapsulation and decapsulation according to the flow entries.

(2) In most OpenFlow implementations (hardware or software), the interfaces attached to an OpenFlow instance in the switches will work as dumb layer-2 ports, for which you can't configure IP address. Packets from these interfaces will not be handled by layer-3 network stack. So we need to deal with ARP messages of the MAT tunnel.

2.1 Match-Action Extension

This section will introduce our Match-Action extension in OpenFlow to express tunneling logic. In OpenFlow, network data structure is reconstructed with Type-Length-Value (TLV) format to support user-defined combination and extension of header fields and actions. As shown in Figure 2, we add new *Match* and *Action* types in the MAT tunnel.

For instance, as for VxLAN, the new field *VXLAN Network Identifier (VNI)* in packets needs to be matched. We also introduce 2 new types of *Action* (*PUSH_VXLAN_TUNNEL* and *POP_VXLAN_TUNNEL*) for tunneling encapsulation and decapsulation. The *Value* of encapsulation action (e.g., *PUSH_VXLAN_TUNNEL*) will have the necessary parameters, such as *source IP address (src_ip)*, *destination IP address (dst_ip)* and *VXLAN Network Identifier (vni)*. After that, controllers can specify the new *Match* field and *Action* type in OpenFlow *Flow_Mod* messages.

Controllers can install flow entries for tunneling with the MAT tunnel rather than any explicit configured tunnel interface. When we want to create a MAT tunnel between two switches, 4 flow entries need to be installed in all, one flow entry for encapsulation and one for decapsulation on each switch. The switches work as the Tunnel Endpoints (TEP) according to the Match-Action Table. The control information and policy can be managed by the controllers, so the MAT tunnel decouples the control and forwarding of tunneling.

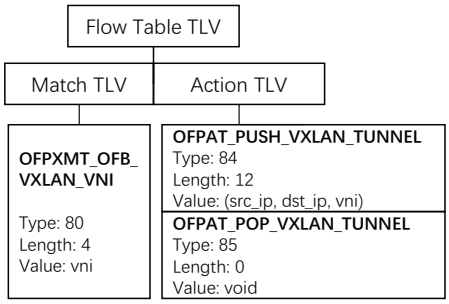


Figure 2: Match-Action Extension in the MAT tunnel.

Table 1 shows the flow entries in one direction (Host 1 -> Host 4, in Figure 1). Controllers will install different flow entries with different identifiers (e.g., vni for VxLAN) on the tunnel endpoint switches for different virtual networks. With the global view, controllers know the relationship between identifiers and ports on the switches, so the MAT tunnel can also support the isolation of virtual networks.

Table 1: Flow Entries of MAT tunnel on the switches in one direction.

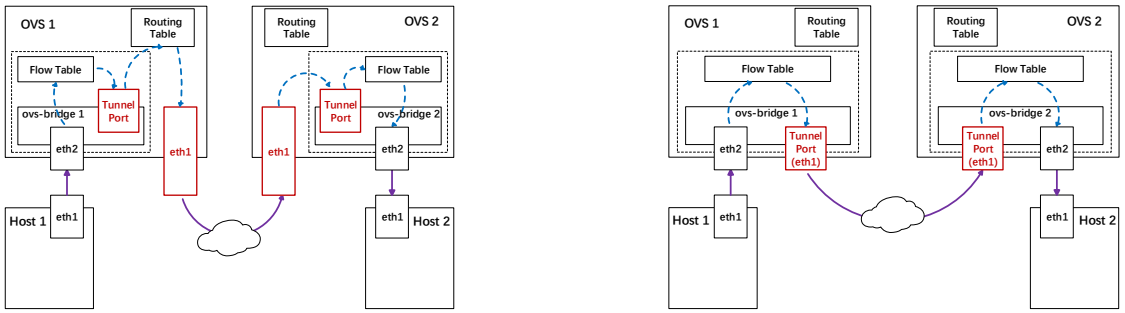
Switch	Match	Action
Ingress Tunnel Endpoint	inport = from host 1 src_mac = host 1 MAC dst_mac = host 4 MAC	push_vxlan_tunnel = (src_ip = switch 1 IP dst_ip = switch 2 IP vni = 1001) output = outer port
Egress Tunnel Endpoint	inport = outer port src_ip = switch 1 IP dst_ip = switch 2 IP udp_port = 4789 MAC vni = 1001	pop_vxlan_tunnel output = to host 4

2.2 Workflow in the Switch

In this section, we will introduce how to handle tunnel packets in the switches. In the traditional tunnel implementation of OpenFlow switches, you should create a tunnel interface and attach it to an OpenFlow instance. This tunnel interface can complete encapsulation and decapsulation functions with your tunnel configurations. But just as mentioned before, the interface in the OpenFlow instance does not have the capability of layer-3. So encapsulated tunnel packets will be forwarded to other layer-3 interface outside the OpenFlow instance according to the routing table in the switch.

We will take Open vSwitch, the software switch as an example. Illustrated in Figure 3(a), the ovs-bridge can be considered as an OpenFlow instance. We have already added a tunnel port to the ovs-bridge. When a packet from eth2 arrives at the ovs-bridge, it is matched against the flow table and forwarded to the tunnel port. The tunnel port is responsible for packet encapsulation. The encapsulated packet will match the longest prefix in routing table and be sent to the physical port (eth1) outside the ovs-bridge. The decapsulation process is similar.

As shown in Figure 3(b), the MAT Tunnel does not rely on any explicit tunnel port. Instead, we should add the physical egress port



(a) Tunnel implemented by Open vSwitch.

(b) MAT Tunnel.

Figure 3: Tunnel workflow in the switch.

(eth1) to the ovs-bridge. It must be noted that, when the port is attached, it will work on layer-2 with no IP address. So controllers should assign routable *virtual IP addresses* for tunnel endpoints in encapsulation actions to ensure the reachability on the Internet. In tunnel ingress endpoint, the packet is matched against the flow table, encapsulated according to the actions and then forwarded directly to the physical interfaces (eth1). In egress endpoint, the virtual switch can match the tunnel packet and decapsulate the packet. The original configuration will be the parameters of the encapsulation actions, which means this schema can work in equal.

From the workflow in the switch, we can expect a simplified logic to handle tunnel packets with the MAT tunnel, which can help reduce the cost of network stack.

2.3 ARP Proxy on the Controller

This section will introduce our solutions to handle ARP messages of the MAT tunnel. As mentioned before, the tunnel interface attached to an OpenFlow instance in the MAT tunnel works on layer-2 with no IP address. The controller allocates a routeable virtual IP address and virtual MAC address for the tunnel interfaces and maintains an IP-MAC binding table. While encapsulating, the controller tells the OpenFlow instance about the MAC addresses of ingress and egress tunnel endpoints, so it will know how to encapsulate outer MAC header.

If ingress and egress tunnel endpoints are in the same subnet, it works well. But if they are in different subnets, the tunnel interface won't handle any ARP request, and the OpenFlow instance won't generate the ARP response by itself as well. Thus, the network gateway which the interface is connected to can' t obtain the MAC address of tunnel endpoint. To solve the problem, we design an ARP proxy on controllers to deal with the ARP requests of tunnel interfaces. When the encapsulation packet arrives at the gateway which the egress tunnel endpoint is connected to, the ARP request for egress tunnel endpoint will be sent to the controller via *Packet_In* message and the controller replies via *Packet_Out* message. Then the tunnel packet can be transmitted to egress tunnel endpoint and decapsulated.

3 IMPLEMENTATION

We implement the MAT Tunnel prototype based on Open vSwitch including GRE and VxLAN tunnels. We also provide REST API on floodlight controller. Source code is at https://github.com/mat-tunnel.

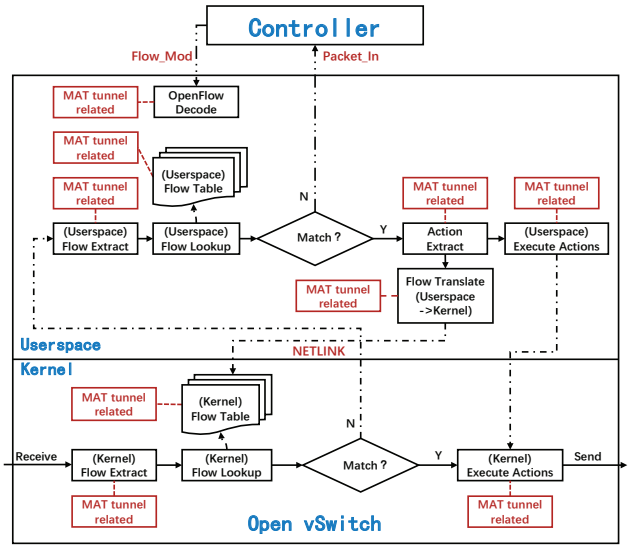


Figure 4: MAT tunnel implementation in Open vSwitch.

As shown in Figure 4, there are kernel and user space in the Open vSwitch. The first packet of a new flow will be sent to user space, which is the *slow path*. And the following packets of the flow will just be handled in kernel, not going through user space. This path is called *fast path*. We have added operations about the MAT tunnel in both slow path and fast path. For example, our implementation will decode MAT tunnel related *Flow-Mod* messages and save the rules in the flow table. When a tunnel packet arrives, it can be matched against its tunnel header fields. Regarding different tunnel protocols, the encapsulation and decapsulation actions are different. For UDP Tunnel like VxLAN, the switch pushes tunnel header (e.g., VxLAN Header), outer UDP header, IP header and MAC header in encapsulation actions.

4 EVALUATION

As shown in Figure 5, we construct a simulation environment based on the real topology of Sprint in Topology Zoo dataset [10], which consists of 11 nodes and 18 links. We create tunnels between all neighbor switches (OVS). We measure the Ping latency between

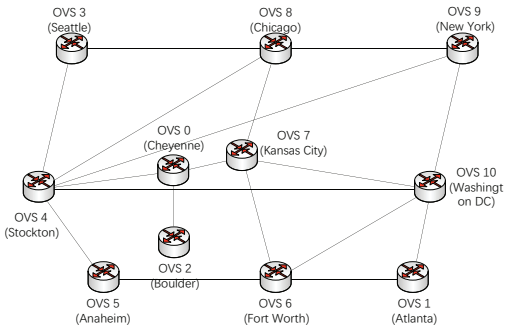


Figure 5: Topology for evaluation.

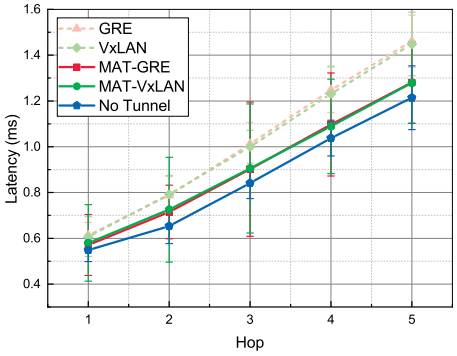


Figure 6: Ping latency between the two hosts.

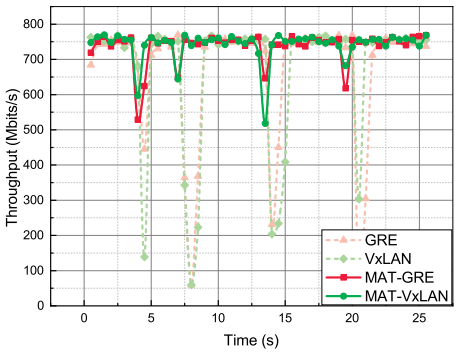


Figure 7: Throughput fluctuation in path switching.

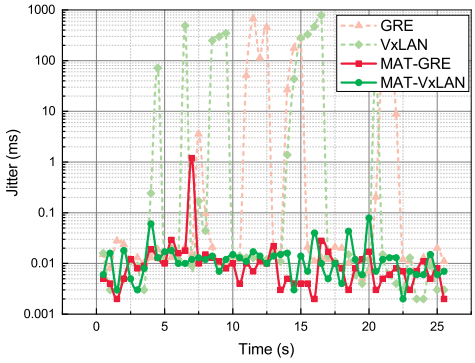


Figure 8: Jitter fluctuation in path switching.

the two hosts passing through one to five hops of different tunnels. Figure 6 shows the result of standard OVS GRE, OVS VxLAN, MAT GRE and MAT VxLAN tunnels. We find our MAT tunnels have less latency in general, compared with the same type of OVS implemented tunnels.

We also conduct the measurement of path switching via tunnels. Two hosts, HOST1 and HOST2, are connected to OVS1 and OVS3. The default path of HOST1 to HOST2 is HOST1-OVS1-OVS10-OVS4-OVS3-HOST2. Making the link between neighbor switches down or up may cause the path switching. The path switching will result in the creation and removal of tunnels. For OVS implemented tunnels, we send the tunnel configurations via OVSDB. For MAT tunnels, we install the flow entries through OpenFlow. We measure the throughput and jitter of HOST1 and HOST2 by iPerf tools. It is illustrated in Figure 7 and Figure 8 that we can expect less fluctuation in throughput and jitter with MAT tunnels, because configuring virtual tunnel interfaces via OVSDB needs to create a network device in linux kernel, which will result in higher overhead compared with installing rules in MAT tunnel. Obviously, MAT Tunnel is more flexible and efficient.

5 CONCLUSIONS

This paper proposes the MAT tunnel, a mechanism of tunneling over IP based on Match-Action Table in SDN and implements a MAT tunnel prototype. The result shows that the MAT tunnel has lower latency in communication, less fluctuation in throughput and jitter in path switching while improving the programmability and flexibility of tunneling.

ACKNOWLEDGMENTS

This work is supported by National Key R&D Program of China (2017YFB0801701), and National Natural Science Foundation of China (No. 61472213). Jun Bi is the corresponding author.

REFERENCES

[1] NM Mosharaf Kabir Chowdhury and Raouf Boutaba. 2010. A survey of network virtualization. *Computer Networks* 54, 5 (2010), 862–876.
[2] Dino Farinacci, P Traina, Stan Hanks, and T Li. 1994. *Generic routing encapsulation (GRE).* Technical Report.
[3] Open Networking Fundation. 2012. Software-defined networking: The new norm for networks. *ONF White Paper* 2 (2012), 2–6.
[4] Jinu Kurian and Kamil Sarac. 2010. A survey on the design, applications, and enhancements of application-layer overlay networks. *ACM Computing Surveys (CSUR)* 43, 1 (2010), 5.
[5] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. 2005. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys & Tutorials* 7, 2 (2005), 72–93.
[6] Mallik Mahalingam, Dinesh Dutt, Kenneth Duda, Puneet Agarwal, Lawrence Kreeger, T Sridhar, Mike Bursell, and Chris Wright. 2014. *Virtual extensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks.* Technical Report.
[7] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38, 2 (2008), 69–74.
[8] Ben Pfaff and Bruce Davie. 2013. The open vSwitch database management protocol. (2013).
[9] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. 2015. The Design and Implementation of Open vSwitch.. In *NSDI*. 117–130.
[10] Topology Zoo 2012. The Internet Topology Zoo. (2012). Retrieved July 18, 2012 from http://www.topology-zoo.org/dataset.html