OpenFlow Networks with Limited L2 Functionality

Hiroaki Yamanaka, Eiji Kawai, Shuji Ishii, and Shinji Shimojo
Network Testbed Research & Development Promotion Center
National Institute of Information and Communications Technology
KDDI Otemachi Bldg. 21F, 1-8-1 Otemachi, Chiyoda-ku, Tokyo 100-0004, Japan
Email: {hyamanaka, eiji-ka, shuji, sshinji}@nict.go.jp

***Abstract*—OpenFlow enables flexible control of network traffic with arbitrary flow definitions. On carrier access networks, OpenFlow can be used to provide customized network settings for each client for virtual private network (VPN), Internet Protocol television (IPTV), and content delivery network (CDN) services, etc. Since there is a massive amount of traffic in carrier access networks, high performance switches are necessary. However, costs tend to increase due to the number of switches. An OpenFlow switch processes wide-range of header fields and supports wildcard matching. Large spaces for ternary content addressable memory (TCAM) and application-specific integrated circuits (ASICs) are used for high performance lookups. Currently proposed techniques reduce the amount of energy that is consumed by reducing the frequency of TCAM usage in switches. However, these techniques require complex functionality in order to manage matching in the switches. As a result, switches are still expensive. In this paper, we propose a technique that enables the construction of OpenFlow networks using switches that require little more than L2 switch functionality. The functionalities that are required for the switches include an OpenFlow interface for handling the flow table externally and a simple matching function for the MAC header. Arbitrary flow definitions from an OpenFlow controller are translated to the flow definitions by the MAC address at the external proxy.**

Keywords-OpenFlow; TCAM; L2 switch; carrier access network

I. INTRODUCTION

Software Defined Networking (SDN) technologies (e.g., OpenFlow [1]) ease network management, service management, and quality of service (QoS) provisioning. SDN technologies are being considered for use in carrier-grade networks. One possible model for applying SDN technologies in carrier networks involves the control of fined-grained flows using OpenFlow in carrier access networks and multi-protocol label switching (MPLS) tunnels (i.e., relatively simple logic for packet forwarding) in core networks [2].

When OpenFlow is deployed in carrier access networks, the costs for infrastructure are relatively high. There are two reasons for high costs. The first is that there are many OpenFlow switches in carrier access networks. The second reason is that hardware OpenFlow switches are costly. The ternary content addressable memory (TCAM) significantly increases the costs that are associated with a hardware-based OpenFlow switches. TCAM is a special type of memory that enables matching on the headers of received data packets during one clock cycle, regardless of the number of entries in memory. This capability of TCAM is preferable in carrier access networks because it enables high performance for the forwarding of high volumes of data packets. However, TCAM is power hungry and expensive [3]. It has been noted that TCAM is up to 80 times more expensive than static random access memory (SRAM) [4]. In an OpenFlow switch, the required TCAM space is large due to the wide range of header fields that are supported in OpenFlow [5].

Techniques have been proposed in the research community for reducing the amount of power that is consumed by TCAM in an OpenFlow switch. The basic idea is to decrease the frequency of the usage of TCAM. Only the first data packet in a flow is matched using TCAM and subsequent data packets that have the same header fields are matched using SRAM or binary content addressable memory (BCAM). In DevoFlow [6], SRAM is used in conjunction with the hash method in order to match the subsequent data packets. The hash method improves the performance as far as possible when SRAM is used. Generally, when the TCAM is avoided in a switch, it is necessary to include chipsets in the switch for complex packet processing (e.g., applying the hash function) in order to obtain high performance packet forwarding.

In this paper, we propose a technique that retains the high performance of during packet forwarding and lowers the cost of switches for OpenFlow infrastructure through the use of relatively simple and inexpensive devices. The idea is to restrict the matching fields in the memory of a switch to the source MAC header and allow the switch to perform matches in a simple manner using the single source MAC header. Meanwhile, the proposed technique enables an OpenFlow controller and the end-hosts to use all of the header fields that are supported in OpenFlow. The external proxy that is between an OpenFlow controller and the switches translates the matching fields that were originally defined by the controller to the matching fields for the source MAC addresses. Furthermore, the proxy manages the edge switches in order to modify the source MAC addresses for data packets and forward them to the network. In the network, the source MAC address of a data packet represents all of the original header fields. As a result, the switches in the network only need to match on the single source MAC header. The proposed technique enables the construction of an OpenFlow network with switches that are implemented using similar chipsets of L2 switches.

The remainder of this paper is organized as follows. Section II describes the mechanism and the limitations of

 ISBN: 978-1-61208-318-6

the current technique for reducing TCAM usage. Section III describes the concept and the architecture of our proposal. Section IV describes the detailed implementation of the proposal. Section V evaluates the overheads that are associated with the proposal. Section VI contains remarks about related work and Section VII presents the conclusion.

II. Reducing TCAM Cost in OpenFlow

A typical hardware-based OpenFlow switch contains TCAM for high performance data packet processing in a network. An OpenFlow [1] network is composed of a controller and a group of OpenFlow switches. The controller and the OpenFlow switches communicate through a control plane on the network in order to maintain flow entries in the switches. The OpenFlow switches transfer data packets on the data plane of the network based on their flow entries. A flow entry includes definitions of the flows that are referred to as the matching fields. The matching fields include the ingress port number and the header fields from layers 2–4 that are specified in the OpenFlow switch specification [5]. Wildcards are allowed for any of the matching fields. An OpenFlow switch searches the flow entries that need to be matched in the header fields of each data packet that is received. TCAM enables searching during one clock cycle, regardless of the number of the entries in the TCAM and regardless of whether wildcards are included or not included in the matching fields.

Since TCAM is power hungry and expensive, it increases the infrastructure costs for OpenFlow networks tremendously. State-of-the-art techniques have been proposed academic papers in order to reduce the frequency of the usage of TCAM (i.e., energy consumption) in switches. These techniques allow TCAM to only be used for matching for the first data packets that arrive, while subsequent data packets are matched using SRAM or BCAM. A switch sees all of the header fields for the first data packet that are matched using TCAM. Then, it sets up the matching fields for the subsequent data packets using SRAM or BCAM. When SRAM is used, matching can be implemented using the hash method for subsequent data packets in a constant time, which is not one clock cycle of central processing unit (CPU). When BCAM is used, matching can be implemented for subsequent data packets in one clock cycle of CPU.

In the section below, *"wildcard matching fields"* refers to matching fields in which at least one header field is a wildcard. For an IP header, it may be the IP prefix. *"Exact matching fields"*, on the other hand, refers to matching fields in which there is no header field with wildcards or IP prefixes.

A. Setting Exact Matching Fields

This section summarized the method for reducing the frequency of TCAM usage. This method is found in DevoFlow [6]. This method determines the exact matching fields inheriting from the wildcards using the header fields of the data packets that are arriving at the switch. The procedure for setting the exact matching fields and the data packet processing is as follows (Figure 1).

1) The wildcard matching fields that are originally set by the OpenFlow controller are memorized in TCAM in the switch.

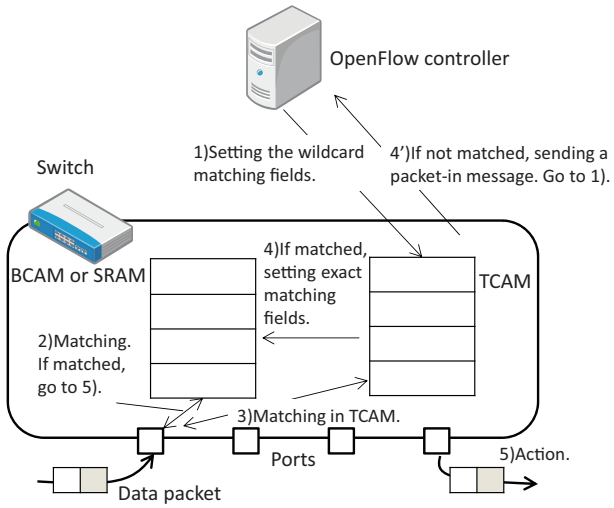

Figure 1. The basic procedure of setting exact matching fields.

2) When a data packet arrives, its header fields are matched to the exact matching fields in BCAM or SRAM. If the header fields are matched, the data packet is processed based on the flow entry (go to 5)).
3) If they are not matched to any of the exact matching fields in memory, then the header fields are matched to the wildcard matching fields in TCAM.
4) If the header fields are matched using TCAM, then the corresponding exact matching fields (Figure 2) are stored in BCAM.
4′) If they are not matched, then the data packet is forwarded as a packet-in message to the OpenFlow controller in order to query about the proper method for processing the data packet.
5) The data packet is processed using the action that is specified in the matching flow entry.

B. Limitations

Generally, there is a trade-off between the level of packet forwarding performance and the complexity of the chipsets for switches when TCAM is not used. Even if it is possible for a switch to process only the exact matching fields, BCAM is still necessary in order to obtain line-rate performance for packet forwarding. Because there are fewer circuits in BCAM than in TCAM [3], the prices for BCAM devices are lower and the devices also consume less energy. However, BCAM is still more costly than SRAM. Current techniques propose methods for obtaining high performance levels with limited BCAM space.

In DevoFlow [6], the exact matching fields are stored in SRAM and the hash method is used to search the flow entries that need to be matched for a data packet. The chipset for DevoFlow is relatively simple. However, the performance is limited because the matching process largely depends on the CPU.

Congdon et al. [7] utilized BCAM to match against the exact matching fields. BCAM stores the small size data of a partial header field or a hash value of the exact matching

 ISBN: 978-1-61208-318-6

The wildcard matching fields

src MAC	dst MAC	src IP	dst IP	src TCP/UDP	dst TCP/UDP
00:11:22:33:44:55	*	192.168.12.1	*	*	80

matched | compliment | matched | compliment | compliment | matched

src MAC	dst MAC	src IP	dst IP	src TCP/UDP	dst TCP/UDP
00:11:22:33:44:55	00:66:77:88:99:AA	192.168.12.1	192.168.12.100	2020	80

The header fields of the arriving data packet

src MAC	dst MAC	src IP	dst IP	src TCP/UDP	dst TCP/UDP
00:11:22:33:44:55	00:66:77:88:99:AA	192.168.12.1	192.168.12.100	2020	80

The corresponding exact matching fields

Figure 2. An example of the wildcard matching fields and the exact matching fields.

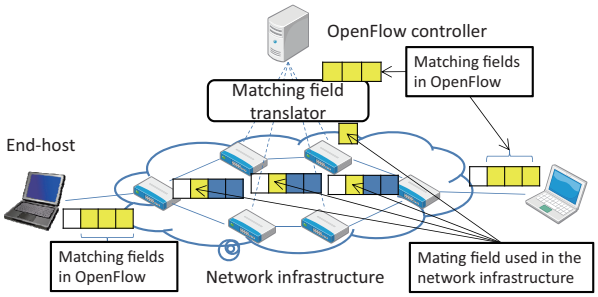

Figure 3. Limiting the matching fields in the network infrastructure.

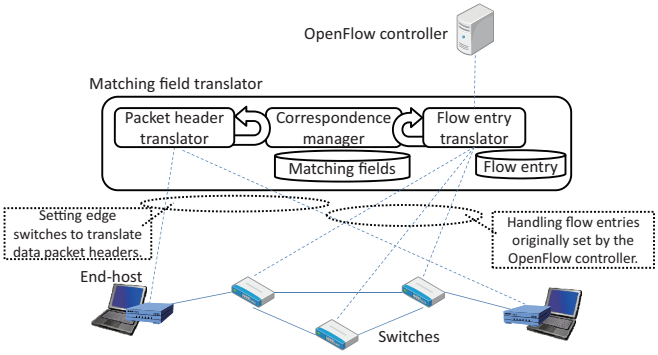

Figure 4. The architecture of the matching field translator.

fields. Along with the small data, BCAM stores the pointer to the original matching fields stored in SRAM. When the partial value or the hash value of the data packet header is matched to the data in BCAM, the correctness of the matching is confirmed by referencing the original exact matching fields in SRAM. This method requires a chipset that is capable of performing a certain amount of complex logic.

III. Proposal of OpenFlow Deployment using the L2-based Switches

We propose a technique that enables deployments of OpenFlow networks using simple, low-cost switches. The main idea of this technique is to limit the use of matching fields to the single source MAC header inside the network (Figure 3). Switches inside the network simply match on the source MAC header fields of the data packets. Meanwhile, the technique enables an OpenFlow controller and the end-hosts to use all of the matching fields that are normally supported in OpenFlow as the matching fields. As a result, this technique retains the programmability of OpenFlow for an OpenFlow controller. Because the single MAC header is the only matching field for single flow entries in a switch, more flow entries can be stored in BCAM. Furthermore, the switch simply matches on the MAC header. As a result, the chipset for the switch requires only minor extension beyond what is required in an L2 switch.

A. Summary of the proposed technique

The technique maps the matching fields that the OpenFlow controller and end-host manage to the corresponding MAC addresses that are manged by switches inside the network. The matching fields for the OpenFlow protocol messages are mapped for the OpenFlow controller. The header fields of data packets are mapped for end-hosts. In the section below, we refer to the translated MAC addresses that are managed by switches inside the network handle as *MAC ID address*.

There is a manager for the correspondence between the exact matching fields and the MAC ID addresses. The correspondence is global throughout the network. Based on this correspondence, the exact matching fields that are specified by the OpenFlow controller are translated into the source MAC ID addresses and the source MAC ID addresses are stored in the BCAM of the switch. For the wildcard matching fields, the exact matching fields are determined using the header fields in actual data packets that arrive, and then, the exact matching fields are translated into the MAC ID addresses.

At a network ingress switch, the source MAC header in the data packet is replaced with the MAC ID address, which corresponds to all of original header fields in the data packet. At a network egress switch, the original header fields from the data packet are recovered based on the correspondence with the source MAC ID address in the data packets and the data packet is transferred to the end-host.

B. Architecture

There is a proxy between the switches and the OpenFlow controller that is referred to as a “matching field translator”. For the OpenFlow controller, the matching field translator behaves like an OpenFlow switch. For the switches, the

 ISBN: 978-1-61208-318-6

TABLE I. AN SIMPLE EXAMPLE OF THE CORRESPONDENCE OF THE ORIGINAL MATCHING FIELDS AND THE MAC ID ADDRESSES.

Original L2–L4 headers			MAC ID address
L2	*L3*	*L4*	
00:11:BB:CC:DD:EE	192.168.1.1	80	00:00:00:00:00:01
00:11:BB:CC:DD:EE	192.168.1.1	22	00:00:00:00:00:02
00:01:BB:CC:DD:FF	192.168.1.100	80	00:00:00:00:00:03

matching field translator behaves like an OpenFlow controller. As shown in Figure 4, there are the correspondence manager, the flow entry translator, and the packet header translator in the matching field translator.

1) The correspondence manager: The correspondence manager handles the correspondences between the exact matching fields and the MAC ID address. For the exact matching fields, the correspondence manager allocates a 48-bit ID in the form of a MAC address. Table I shows a simple example of the correspondence between the matching fields and the MAC ID addresses. The matching fields and the IDs are in a one-to-one correspondence. There are two modules that search the correspondence: the flow entry translator and the packet header translator. If the correspondence manager has not allocated IDs to the matching fields yet, it allocates a new ID and returns it as the MAC ID address. This ID allocation can be implemented by allocating sequential numbers to the new exact matching fields. Searching can be implemented using the hash method in a constant time.

Note that the MAC ID address always serves as the basis for the matching fields that are specified by the OpenFlow controller or for the header fields in the data packets that are sent and received by end-hosts. As a result, when the original matching fields are searched using the MAC ID address, the corresponding matching fields always exist.

2) The flow entry translator: The flow entry translator modifies and relays the OpenFlow protocol messages between the OpenFlow controller and the switches. For a flow entry installation for the exact matching fields, the flow entry translator simply replaces them with the corresponding MAC address and forwards the message to the switch.

For a flow entry installation for the wildcard matching fields, the flow entry translator determines the corresponding exact matching fields when the new data packet arrives at the switch. Then, the flow entry translator obtains the corresponding MAC ID address, and sends the flow entry installation message for the MAC ID address.

For a flow statistics request, from the OpenFlow controller, for the flow entry of the wildcard matching fields, the flow entry translator collects the flow statistics from the switch, at first. The objectives of the collection are the flow entries of the exact matching fields that correspond to the wildcard matching fields. Then, the flow entry translator responds with the aggregated value as the statistics for the requested flow entry.

3) The packet header translator: The packet header translator modifies the header fields of data packets that are transferred through the edge ports (i.e., the ports that connect directly to the end-hosts) of the edge switches. For a data packet that is sent from an end-host, the packet header translator simply replaces the source MAC header with the MAC ID address corresponding to the original header fields in the data packet. For a data packet that is sent from the switch to the end-host, all of the header fields are set to the values in the exact matching fields that correspond to the source MAC ID address of the data packet. Note that header fields other than the source MAC header are modified at an egress switch because the OpenFlow controller may have modified other header fields.

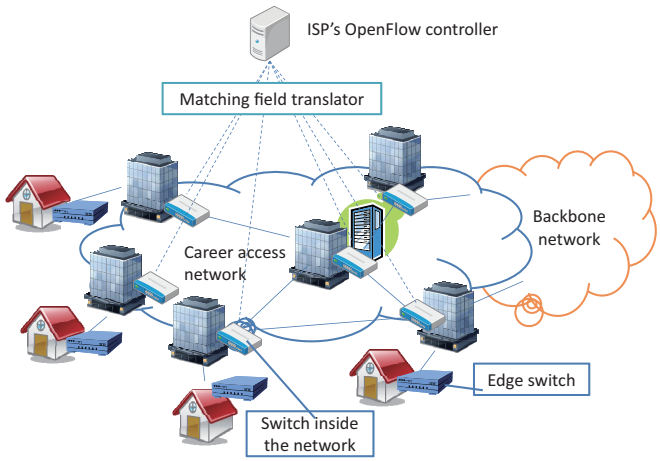

Figure 5. Deployment in a carrier access network.

C. Deployment Scenario

The proposed architecture can reduce the capital expenditure (CAPEX) and operational expenditure (OPEX) for infrastructure in a carrier access network deployment scenario [2]. On the carrier access network, an ISP's controller manages flows for clients in order to provide customized networks settings for virtual private network (VPN), Internet Protocol television (IPTV), content delivery network (CDN) services, etc. As shown in Figure 5, an ISP's controller manages switches in central offices and data centers in the carrier access network via the matching field translator. Edge switches for packet header translation exist in each homes. Although the edge switches need to match all of the header fields, the cost of edge switches is not high. Since the amount of traffic is relatively low in individual home, software-based switches are sufficient for meeting the demands that are replaced on edge switches.

IV. DETAILED IMPLEMENTATION

In this section, we describe the detailed implementation of packet header translation at ingress and egress switches, and the handling of flows (i.e., translations for flow entry installations, obtaining flow statistics, and managing packet-out messages). These scenarios involve translations of the matching fields into the MAC ID addresses.

A. Translation of data packet headers

Edge switches rewrite the headers of data packets that are sent directly from and to end-hosts. The edge switches include OpenFlow functionality, i.e., they can manage all of the supported header fields. An edge switch has at two ports. One is connected to an end-host in a home network. The other is

 ISBN: 978-1-61208-318-6

connected to the central office, i.e., the carrier access network. The packet header translator is configured in advance with information that specifies the port that connects to the end-host or the central office.

When an unknown data packet (i.e., no matching flow entry is set) arrives at the port in the edge switch that connects to an end-host, the switch sends a packet-in message to the packet header translator. Then, the packet header translator obtains the corresponding MAC ID address for the header fields in the data packet. Finally, the packet header translator sets the flow entry in the edge switch. The matching fields for the flow entry include the header fields of the data packet. The actions that are performed include modifying the source MAC header to the corresponding MAC ID address and sending out the data packet from the other port. Subsequent data packets for the same header fields are simply matched. Then, flow entry actions are applied and the data packets are transferred.

When an unknown data packet arrives at the port of an edge switch that connects to the central office, the switch obtains the flow entry from the packet header translator. The flow entry is used for recovering the original data packet header and transferring the data packet from the other port. The matching fields in the flow entry include the source MAC ID address, which corresponds to the original header fields. The actions that are performed include the modification of all of the header fields to the original header fields. Note that the OpenFlow controller may modify arbitrary header fields in data packets in the network. Since only the source MAC header is managed in the network, other header fields are left unchanged. However, the source MAC ID address of the arriving data packet corresponds to the header fields when the data packet arrives at the switch. This is guaranteed by the translation of the flow entry installation that is described in Section IV-B2.

B. Flow Entry Installation

In order to manage flows, the flow entry translator manages the correspondence of the original flow entries and the exact matching fields for the flow entries that have actually been installed (Table II). The original flow entries are for the wildcard or exact matching fields that were set by the OpenFlow controller. The installed flow entries are for the exact matching fields, which are the same as the original exact matching fields or the exact matching fields that were specified by the data packet that actually arrived. The flow entry installation process is described in the section below. In this process, the correspondence database (i.e., Table II) is updated when a flow entry is installed into, or removed from the switch.

For the installation of a flow entry whose matching fields are the exact ones, the flow entry translator simply translates the flow entry whose matching fields are the source MAC ID address, and installs ("FI-iii" in Figure 6). For the installation of a flow entry whose matching fields are the wildcard values, the flow entry translator installs possible flow entries in the switch first in order to preserve the consistency of the priorities in the switch's flow table ("FI-i" in Figure 6). Then, the flow entry translator determines the corresponding exact matching fields ("FI-ii" in Figure 6) and installs the flow entry ("FI-iii" in Figure 6) .

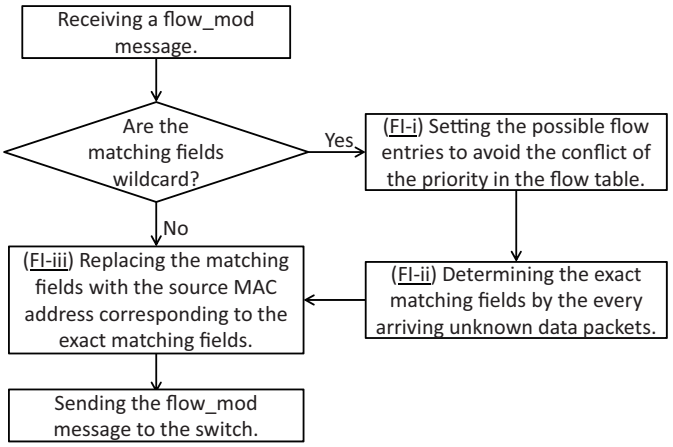

Figure 6. Summarized process of flow entry installation.

1) Maintaining the priorities in the flow table (FI-i): As described in Section IV-B2, for a flow entry for the wildcard matching fields, the flow entry translator does not immediately install the flow entry when it receives the flow entry installation message, i.e., a flow_mod message. Due to the time lag that is associated with the installation of a flow entry, an arriving data packet can be matched inappropriately to another flow entry whose priority is lower than that of the original one.

For instance, in Table II, let us assume that a flow entry for switch 0x1 (the priority is 65536 and the matching fields are "00:11:22:33:44:55, 192.168.10.2, 80") is not installed. Meanwhile, let us assume that another flow entry for switch 0x1 (the priority is 65534 and the matching fields are "00:11:22:33:44:55, 192.168.10.2, 80") is installed. Subsequently, a data packet that arrives with header field values of "00:11:22:33:44:55, 192.168.10.2, 80" should be matched to the flow entry with priority 65536. However, the data packet matches to the one with priority 65534, and the switch fails to send a packet-in message. This conflicts with the OpenFlow controller because the OpenFlow controller assumes that the data packet matches to the flow entry with priority 65536.

In order to avoid this conflict, the flow entry translator installs the possible flow entries for the exact matching fields corresponding to the flow entry (denoted by "E") for the wildcard matching fields. First, the flow entry manager compares the wildcard matching fields for E and the exact matching fields for the flow entries that have already been installed for the lower priorities in the flow table. Then, for the installed flow entries that are found, whose exact matching fields overlap with the matching fields of E, the flow entry translator immediately installs the flow entries that correspond to E. The exact matching fields of the installed flow entries are the same as the matching fields for the flow entries that were found. The priority of the flow entries are the same as the priorities for E. The actions for the installed flow entries are the same as the actions for E. The specified flow entries are translated and installed by the same process that is found in "install-iii" in Figure 6.

2) Determining the matching fields and installing the flow entry (FI-ii): The flow entry translator determines the corresponding exact matching fields based on the data packets that

TABLE II. AN EXAMPLE OF THE FLOW ENTRY CORRESPONDENCE DATABASE.

Switch ID	Original matching fields	Priority	Installed exact matching fields
0x1	00:11:22:33:44:55, ∗, 80	65536	00:11:22:33:44:55, 192.168.10.1, 80
		65536	00:11:22:33:44:55, 192.168.10.2, 80
0x1	00:11:22:33:44:55, 192.168.10.2, 23	65535	00:11:22:33:44:55, 192.168.10.2, 23
0x1	00:11:22:33:44:55, 192.168.10.2, 80	65534	00:11:22:33:44:55, 192.168.10.2, 80
0x2	00:11:22:33:44:55, 192.168.10.1, ∗	65536	00:11:22:33:44:55, 192.168.10.1, 23
		65536	00:11:22:33:44:55, 192.168.10.1, 80
0x2	00:11:22:33:44:55, 192.168.10.2, 23	65535	00:11:22:33:44:55, 192.168.10.2, 23

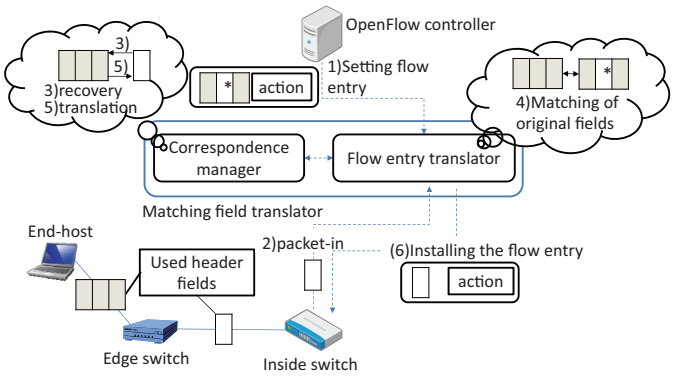

Figure 7. Installing a physical flow entry based on an arriving data packet.

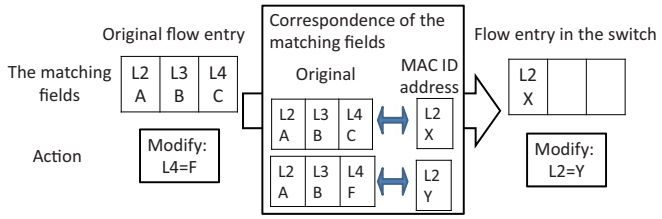

Figure 8. Translation of header modification.

are arriving. When the flow entry translator receives a packet-in message from the switch (#2) in Figure 7), it first recovers the original L2–L4 headers from the source MAC ID address of the data packet (#3) in Figure 7). Then, the flow entry translator compares the L2–L4 headers with the wildcard matching fields from the entries in the flow entry correspondence database (#4) in Figure 7). If there is the flow entry for the wildcard matching fields to be matched to, a flow entry translator generates a flow entry using the exact matching fields that correspond to the source MAC ID address in the packet-in message. Then, the flow entry is translated and installed into the switch by the same process that is described in "install-iii" (#5) and #6) in Figure 7). If there is no flow entry to be matched to, the header fields in the data packet are recovered using the original ones from the source MAC ID address and the packet-in message is transferred to the OpenFlow controller.

3) Translation of a flow entry (FI-iii): For an original flow entry whose matching fields are exact, the flow entry translator simply replaces the original matching fields with the source MAC ID address corresponding to the original matching fields, and installs the flow entry.

When the flow entry translator installs a flow entry into the switch, the action for modifying the header field is also replaced. When an OpenFlow controller modifies the header field of a data packet, only the partial header field that is to be modified is specified in the message. The flow entry translator translates the action field so that only the source MAC ID address is replaced (Figure 8). This translation enables switches to modify only the source MAC header while the source MAC ID address of the data packet in the network always represents the original header fields that should be at the given position in the network at that time. The translated source MAC ID address is the one that corresponds to the combination of the values of all header fields resulting from the modification. The combination of the resulting header fields can be determined by replacing the value of the partial header in the matching fields with the value that is specified in the action of the flow entry.

C. Other processes of flow handling

1) Obtaining flow statistics: If the OpenFlow controller sends a message to obtain the flow statistics from the flow entry for the exact matching fields, the flow entry translator simply replaces the original matching fields to the source MAC ID address. The MAC ID address corresponds to the original exact matching fields.

If the request is for the flow statistics for the wildcard matching fields, the flow entry translator collects the flow statistics from the switch, sums them up, and responds with the sum for the flow statistics from the original flow entry. Since the flow statistics represent the number of data packets that were matched to the flow entry, the sum of the flow statistics for the corresponding flow entries that were installed for the exact matching fields is returned.

2) Packet-out: A packet-out message is for sending out a data packet from the OpenFlow switch. In the message, the data packet that is being sent is specified by the buffer ID or by the data packet included in the message. If the data packet is included in the message, the flow entry translator replaces the source MAC header with the MAC ID address that corresponds to the original header fields in the data packet. Then, the message is forwarded to the switch and the switch sends out a data packet with MAC address that corresponds to the original header fields.

V. EVALUATION

The proposed technique enables simple, low-cost implementations of hardware-based switches for OpenFlow infrastructures. However, the matching field translator adds overheads. The overhead is especially critical for the data transmission performance. The overheads that are caused by the matching field translator are caused by the latencies for flow entry settings. In our proposal, the corresponding matching fields are determined for each new data packet.

 ISBN: 978-1-61208-318-6

In order to evaluate the latency from the flow entry installations, we measured the lost data size due to installations by the flow entry translator. For comparison purposes, we measured the lost data size on a network that was controlled directly by an OpenFlow controller. The parameter that affects the delay of the flow entry installation process is the number of original flow entries for the wildcard matching fields that are set by the OpenFlow controller. The number of original flow entries for the wildcard matching fields directly affects the delay because the matching to the wildcard matching fields is implemented by software, i.e., linear searching.

A. Settings

We assumed a simple network whose data plane was composed of two end-hosts, two edge switches, and a switch. On the control plane, there was an OpenFlow controller and a matching field translator. There was no matching field translator and the software edge switches functioned as the switching hubs when measurement were being performed in the network without the proposed technique. The OpenFlow controller and the matching field translator ran on systems with an Intel Xeon E5520 processor and 6GB RAM. The edge switches used systems with Intel Celeron Dual-Core T3330 processors with 2GB RAMs that ran Open vSwitch [8]. We used the USB Network Interface Cards (NICs) as the ports of the edge switches. The end-hosts ran on systems with Intel Celeron Dual-Core T3330 processors and 2GB RAMs. In the experiments, the operating system on the all hosts was Ubuntu 12.04 LTS.

The switch inside the network was NEC PF 5240 switch. Based on the switch specification that we confirmed, the switch supported a maximum buffer size of 544 packets, i.e., the maximum number of new data packets with the same header fields that can be buffered in the switch is 544. Note that our proposed technique does not require TCAM in a switch, but does require the BCAM space for the source MAC headers. Unfortunately, we did not have a hardware-based switch that incorporates the required and necessary functionality for the proposed technique. As a result, we used the hardware-based OpenFlow switch that includes TCAM when we ran the matching field translator. The exact matching fields of the source MAC ID addresses for the flow entries installed by the matching field translator were processed using TCAM. However, since the matching performance of TCAM and BCAM is same for the exact matching fields, the switch used in this experiment did not affect the results in terms of the packet forwarding performance of a switch in the network.

The OpenFlow controller set the number flow entries for the wildcard matching fields when it connected to the switch. When we used the matching field translator, the flow entries were stored in the matching field translator at first. However, if we did not use the matching field translator, the flow entries were installed immediately into the switch.

We used iperf to send data packets between the end-hosts. The UDP packets were sent at a rate of 75.40 Mbps because that is the maximum throughput for environment that was used in the experiment. The maximum throughput was low due to the USB NICs. However, it was sufficient because we assumed that the edge switches were home router-class switches. In the experiments, the size of a UDP datagram was 1.47 KB.

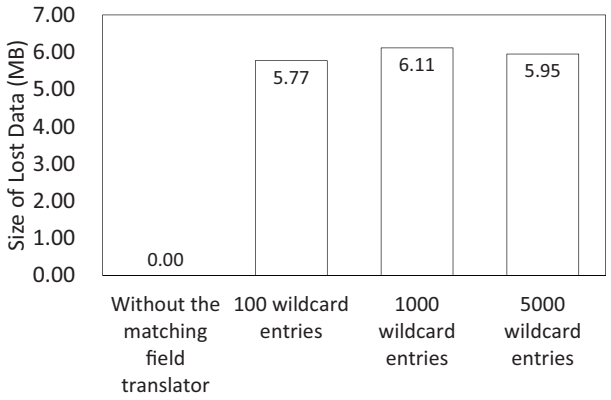

Figure 9. Lost data size vs. the wildcard entries in the flow entry translator.

B. Results

We measured the total lost data size for the first 1 seconds of sending UDP packets. Note that, regardless of the number of the wildcard flow entries, packet loss ended within 1 second in our experiment. Figure 9 shows the sizes of the lost data based on the numbers of wildcard entries in the flow entry translator and also for the cases where the OpenFlow controller was connected directly to the switch. The average for 10 experiments is shown.

As expected, a certain amount of data was lost when the flow entry translator was used. In this experiment, the size of the lost data was in the range of 5.77–6.11 MB. At most, the data for $1.47\text{KB} * 544\text{packets} = 799.68\text{KB}$ was buffered in the switch. Note that it was not possible to measure the precise size of the buffered data because the buffer was in the shared memory in the switch. The sum of the lost data and the rough estimate for the size of the buffered data is less than 7 MB. This is comparable with the buffer sizes in current top-of-rack (ToR) switches, e.g., the HP 5900 has 9MB of buffer space [9]. In general, the necessary buffer size depends on the throughput and the rate at which data packets with new header fields are arriving. Larger buffer sizes may be necessary. However, since buffers can be implemented using SRAM, we believe that the technique that we have proposed is capable of enabling low cost switches for the OpenFlow infrastructure. The cost is lower than the cost of using TCAM and complex chipsets.

VI. Related Work

Other than the approach that is described in Section II for reducing TCAM power, there are also approaches that reduce the cost of OpenFlow switches.

A. Software switch implementation

Software-based OpenFlow switch implementations uses general purpose processors (GPP) and DRAM or SRAM devices. The software implementation reduces the CAPEX and the OPEX of the infrastructure significantly. Open vSwitch [8] is the most widely used software for OpenFlow switch functionality on Linux systems. Dedicated hardware is not required when Open vSwitch is being used. It is designed for edge switches for virtual machines in data centers. Matsumoto et

 ISBN: 978-1-61208-318-6

al. [10] proposed a hash searching method for high speed matching to flow entries in RAM. For the exact matching fields for a flow entry, the hash value is calculated immediately when it is installed. For the wildcard matching fields for a flow entry, matching for the packet that arrives first is performed using one-by-one comparisons to flow entries in RAM. Then, the hash value of the header in the first data packet is calculated and the value is used for matching the subsequent data packets. A software switch is not suitable for forwarding the massive volumes of data packets in carrier access networks.

B. Product OpenFlow switches

Hybrid OpenFlow switches from NEC, HP, IBM, and Juniper use combination of general purpose and dedicated hardware devices. There are various patterns of combinations. Examples include combinations of application-specific integrated circuits (ASICs) and SRAM, combinations of TCAM for a limited number flow entries and SRAM for other flow entries, etc. The hybrid implementations can obtain high performance levels and reduce costs slightly. We propose a technique that would dramatically reduce the complexity of an OpenFlow switch implementation and also achieve line-rate packet forwarding performance levels.

C. IP source routing-like approach

Source flow [11] proposes an IP source routing-like approach. This technique embeds the action lists for data packets from intermediates switches in the network into the header fields at the ingress switches. Switches inside the network base their actions on the pointers in the header fields. As a result, the number of entries in the switch is the same as the number of actions in the switch. Because the number of flow entries is typically lower than of the number of actions that have been performed in the switch, the technique can reduce the number of entries in switches. The action list for the switch is retrieved from a centralized controller like an OpenFlow controller. The constraint on this approach is that all of the actions have to be supplied for the data packet at the ingress switch at once. Our proposal, on the other hand, virtualizes an OpenFlow network completely, i.e., an OpenFlow controller can see flows by obtaining the flow statistics at every hop in the network. As a result, the proposed functionality offers a higher level of control of the network than the source flow technique.

D. Label switching

The label switching techniques enable all switches in the network to match using only a single header that is referred to as the label. At the ingress switch of the network, the label for the data packet is added to the header fields of the data packet or the header fields are replaced using the value from the label. Inside the network, switches do not need to have TCAM. Furthermore, the length of the matching field is short. As a result, less BCAM space is sufficient. MPLS [12] is widely used in carrier core networks. The MPLS label is added at the ingress switch based on the destination IP address. In the network, switches forward data packets based simply on the label. However, MPLS does not have the same level of programmability as OpenFlow because it has two constraints: inflexible flow definitions (i.e., the label is based on the destination IP address) and a lack of a global view of the network.

PortLand [13] is designed to control routing for data center communications using the MAC header as the label in order to support the communications for many hosts at different sites using a large space for a MAC addresses. For every data packet that is destined for another site in the data center, the MAC header is replaced at the gateway. The new MAC address corresponds to the original MAC address and the ID of the destination site. This correspondence is managed globally. The MAC address is recovered at the gateway of the destination site. The difference between PortLand and our proposal is that PortLand is not designed to virtualize an OpenFlow network, but is designed for construction of a large-scale L2 network without the ARP broadcast overhead. On the other hand, our proposal supports arbitrary flow definitions in the network.

VII. Conclusions and Future Work

This paper proposed a technique that enables construction of OpenFlow networks using switches that have little more than L2 switch functionality. Arbitrary flow definitions from an OpenFlow controller are translated into flow definitions that are based on the MAC ID address at the external matching field translator and the flow entries are installed into the switch. For the wildcard matching fields in the flow entry, the corresponding exact matching fields are determined and the flow entry is installed into the switch after the first arrival of the matching data packet in order to avoid using TCAM. For translations in the data plane, the matching field translator manages the edge switches in order to modify the MAC headers of data packets. In our proposal, the OpenFlow interface is required in the switches inside the network in order to manage the flow table externally and enable simply matches on the MAC header. We believe that this type of switch can be assembled as an extension of an L2 switch.

A future work will focus on a distributed implementation for the matching field translator for scalability in carrier access networks. In the architecture of the proposed technique, the single matching field translator manages all of the switches in the network. The functions of the packet header and the matching field translators involve independent processes for individual switches in the network. As a result, they can be implemented easily in a distributed manner. Furthermore, the global correspondence of the original header fields and the MAC ID address can be managed easily by a distributed lookup system, e.g., a distributed hash table. However, we need to explore fast sharing schemes for the correspondence data because the distributed version of the matching field translator will be deployed at geographically remote sites. The time that is required in order to reference the correspondence is critical for the performance of the packet header and the matching field translators.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, et al., "OpenFlow: Enabling innovation in campus networks," in Proceedings of the ACM SIGCOMM 2008 Conference, vol. 38, no. 2, 2008, pp. 69–74.

[2] "SPARC deliverable d2.1: Initial definition of use cases and carrier requirements," 2010, [retrieved: Dec. 2013]. [Online]. Available: http://www.fp7-sparc.eu/home/deliverables/

 ISBN: 978-1-61208-318-6

[3] K. Pagiamtzis and A. Sheikholeslami, “Content-addressable memory (CAM) circuits and architectures: A tutorial and survey,” IEEE Journal of Solid-State Circuits, vol. 41, no. 3, 2006, pp. 712–727.

[4] J. Liao, “SDN system performance,” 2012, [retrieved: Dec. 2013]. [Online]. Available: http://pica8.org/blogs/?p=201

[5] “OpenFlow switch specification version 1.3.0 (wired protocol 0x04),” 2012, [retrieved: Dec. 2013]. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf

[6] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, “DevoFlow: Scaling flow management for high-performance networks,” in Proceedings of the ACM SIGCOMM 2011 Conference, 2011, pp. 254–265.

[7] P. Congdon, P. Mohapatra, M. Farrens, and V. Akella, “Simultaneously reducing latency and power consumption in openflow switches,” IEEE/ACM Transactions on Networking, 2013, to appear.

[8] “Open vSwitch,” [retrieved: Dec. 2013]. [Online]. Available: http://openvswitch.org/

[9] “QuickSpecs HP 5900 Switch Series,” [retrieved: Dec. 2013]. [Online]. Available: http://h18004.www1.hp.com/products/quickspecs/14252_div/14252_div.pdf

[10] N. Matsumoto and M. Hayashi, “Performance improvements of flow switching with automatic maintainance of hash table assisted by wild-card flow entries,” in Proceedings of the 10th International Conference on Optical Internet (COIN 2012), 2012.

[11] Y. Chiba, Y. Shinohara, and H. Shimonishi, “Source flow: handling millions of flows on flow-based nodes,” in Proceedings of the ACM SIGCOMM 2010 conference, 2010, pp. 465–466.

[12] “RFC 2031: Multiprotocol label switching architecture,” 2001, [retrieved: Dec. 2013]. [Online]. Available: http://tools.ietf.org/html/rfc3031

[13] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, et al., “PortLand: A scalable fault-tolerant layer 2 data center network fabric,” in Proceedings of ACM SIGCOMM 2009 Conference, 2009, pp. 39–50.

 ISBN: 978-1-61208-318-6