The FlowAdapter: Enable Flexible Multi-Table Processing on Legacy Hardware

Heng Pan
Institute of Computing Technology, Chinese Academy of Sciences, China
panheng@ict.ac.cn

Hongtao Guan
Institute of Computing Technology, Chinese Academy of Sciences, China
guanhongtao@ict.ac.cn

Junjie Liu
Institute of Computing Technology, Chinese Academy of Sciences, China
liujunjie@ict.ac.cn

Wanfu Ding
Huawei Technologies Co. Ltd.
dingwanfu@huawei.com

Chengyong Lin
Huawei Technologies Co. Ltd.
linchengyong@huawei.com

Gaogang Xie
Institute of Computing Technology, Chinese Academy of Sciences, China
xie@ict.ac.cn

ABSTRACT

OpenFlow is one of the most potential technique to enable innovation in network. To enable OpenFlow more flexibility and high-efficiency, multi-table pipeline has been introduced in OpenFlow. A HAL(Hardware Abstraction Layer) is proposed to address the incompatibility of flow table pipeline between legacy switch hardware and the controller. However, the burden of controller will be increased greatly. In this paper, an innovative middle layer called FlowAdapter is proposed. It converts flow entry rules from the controller flow table pipeline to switch hardware flow table pipeline, so that the same rules can be fitted into different types of hardware. With FlowAdapter, legacy OpenFlow hardware can be used to support multi-table pipeline rules. Located in switch, FlowAdapter is transparent to the controller. With a prototype implementation, we find that the FlowAdapter performs rules conversion effectively.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Store and forward networks; C.2.2 [**Network Protocols**]: Routing protocols; C.2.3 [**Computer-Communication Networks**]: Network Operation; C.2.6 [**Internet working**]: Routers

General Terms

Soft Defined Networking, Algorithms, Design, Experimentation

Keywords

OpenFlow, equivalent conversion, flow table, FlowAdapter, middleware

HotSDNqf13, August 16, 2013, Hong Kong, China.

1. INTRODUCTION

OpenFlow [1] is one of the most promising approaches to enable innovation in network and becomes a hot topic among industry and academia. A number of network switch vendors have great interest in OpenFlow, including Huawei, Cisco, IBM, and NEC. They also launched their own OpenFlow1.0-enabled switch and other SDN based solutions in data center network. The establishment of ONF (Open Networking Foundation) has helped the standardization of OpenFlow. In OpenFlow 1.1 specification [2], multi-table pipeline has been introduced to make the switch more flexible and extensible. However, the implementation of such a pipeline in hardware can be very complicated.

Even though TCAM (Ternary Content Addressable Memory) is the de facto standard in industry to achieve high performance FIB lookup [3,6], the flexibility of TCAM based multi-table pipeline is quite limited. The multi-table pipeline can be implemented with multiple TCAM chips or a shared TCAM chip, but both of which has either limited pipeline stages or uncertain pipeline delay. In addition, in OpenFlow 1.3 spectification, the length of all match fields exceeds 1000 bits, whereas the length of entries in a TCAM chip is relatively limited, usually ranging from 36 to 576 bit. Moreover, the hardware capabilities of diverse switches may be different. It also brings about a big challenge for the controller to put the same rule set into diverse hardware pipeline.

The controller-issued rules (also called user-defined rules), which are generated by user applications through a "network operating system" like NOX [7], do not consider switch hardware capabilities. Depending on user applications, the form of the user-defined rules can be various ranging from single table pipeline rule to multi-table pipeline rule. On the other way, the lack of flexibility in switch hardware is impossible to adapt different form of rules in hardware pipeline design. For example, suppose a user-defined rule requires to issue a flow entry to flow table 5 of an OpenFlow-enabled switch. Unfortunately, the hardware switch only contains three flow tables. For another example, suppose a user-defined rule issues a flow entry with IP_SRC field to flow table 1. Nevertheless, flow table 1 of the hardware switch may not support IP_SRC match field. How can rules be run on fixed switch hardware while maintaining their flexibility? How can the

same rule set be issued to switches with different capabilities?

The ONF Future Group tries to address this chanllage using a HAL (Hardware Abstraction Layer) [8], which gives the controller a unified interface, (i.e. HAL models), to negotiate pipeline capabilities with OpenFlow switch. Through the negotiation mechanism, the controller can issue rules fitting the switch pipeline capabilities. However, the controller already has many complex tasks to do, getting status, issuing rules, communicating with hundreds and thousands of switches. Managing rules for different HAL model switches can further burden the controller. Additionally, the HAL solution also requires both the controller and switches to be updated and aware of the HAL models. To reduce the complexity of the negotiation, the types and flexibility of HAL models will be limited which might compromise the flexibility of multi-table pipeline.

In this paper, we has proposed a three-layer architecture addressing this challenge. The architecture contains a flexible software data plane, a relatively fixed yet high performance hardware data plane and a middle layer in between called FlowAdapter. The software data plane stores controller-issued flexible rules. The hardware data plane is used to perform high-speed packet forwarding. And the function of the FlowAdapter is to convert M-stage multiple flow table rules into N-stage rules equivalently. So, those flexible rules stored in the software data plane can be converted and issued to the hardware data plane based on the capabilities of the hardware. Using this architecture, the switch hardware is transparent to the controller. Flexible user-defined rules can be arbitrary issued to any specified switch without worry about pipeline capabilities. This paper focuses on depicting the design and implementation of the FlowAdapter, an innovative middle layer, in detail.

To build up the three layer architecture, we implement a simple hardware data plane using PEARL platform [9] and run OpenFlow software switch in server as software data plane [10]. Above all, our key contribution is to present the first method to complete equivalent conversion of the rules that makes the OpenFlow network more flexible and available.

We organize the rest of the paper as follows. In Section 2 we discuss some related work. In Section 3 we show how the FlowAdapter is designed and implemented. In Section 4, we evaluate our FlowAdapter in latency of rules conversion. In Section 5, we discuss some future work. Finally, we conclude our work in Section 6.

2. RELATED WORK

As we mentioned in the Section 1, some people, in ONF Future Group, have proposed a HAL model to address the challenge of the controller adapting OpenFlow-enabled switches [8]. The HAL model defines three ways to configure itself. It offers a unified interface to communicate with the controller. The switch can negotiate with the controller using the HAL model, so that the controller can issue rules based on the capability of the switch. In other words, the model provides a mechanism to make it possible that the controller can detect the capability of a switch. Then the controller converts those flexible user-defined rules into the rules that fitting the capability of the switch. Finally, the controller issues the converted rules to the switch.

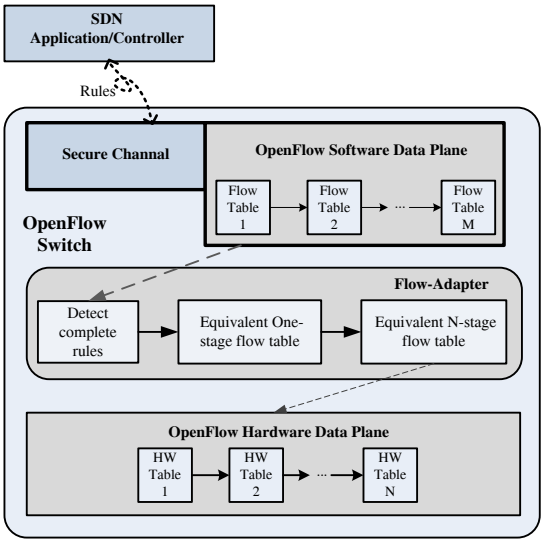

Figure 1: The architecture of a switch

Another work has been done by Alex X. Liu and Chad R. Meiners [11], although they were originally not intended to solve this problem. They use a FDD (Firewall Decision Diagram), a special decision tree, to let a d-dimensional packet classifier be split into multiple low dimensional classifiers. Then they use an equal number of TCAM chips to store those low dimensional classifiers. Accordingly, They let a d-dimensional lookup be replaced by a multi-stage pipeline lookup. It refers to how to convert one-stage flow table into M-stage multiple flow tables equivalently.

3. OUR FLOWADAPTER

3.1 The Architecture Overview

We propose a mechanism to address the mismatch between the controller-issued multi-stage rule tables and the hardware planes. Each OpenFlow-enabled switch contains a flexible software data plane which can be upgraded to support new protocols. The hardware data plane of the switch achieves high-speed packets forwarding. Our FlowAdapter is a middle layer between the software data plane and the hardware data plane to enable them work together effectively. Figure 1 shows the architecture of our proposed OpenFlow-enabled switch.

The FlowAdapter is designed to convert M-stage multiple flow tables issued by the controller into N-stage multiple flow tables implemented in the hardware data plane of each switch equivalently. Now, we will first describe our FlowAdapter at a high level here. We divide the FlowAdapter into two major stages - MTO (the conversion of M-stage multiple flow tables into a One-stage flow table), OTN (the conversion of a One-stage flow table into N-stage multiple flow tables). One might wonder that why you don't conduct the conversion of M-stage multiple flow tables into N-stage multiple flow tables directly. We think that the value of N may be 1. In addition, Our FlowAdapter should be compatible with the legacy hardware which only stores a single flow table. Besides, the One-stage flow table can act as a great bridge in the whole conversion process. It can simpli-

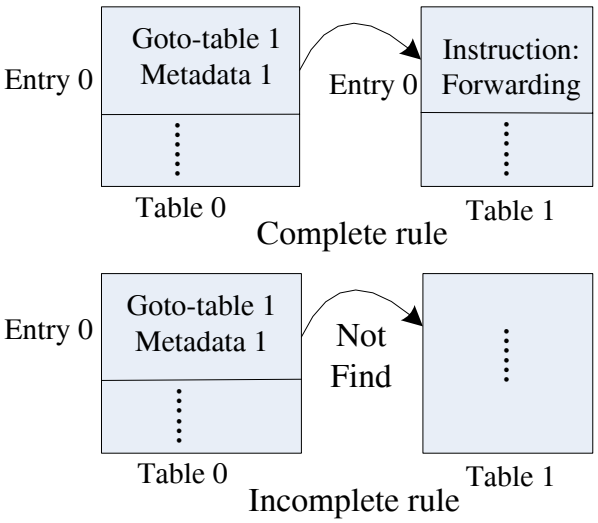

Figure 2: Complete rules and Incomplete rules

fy many complicated processes like eliminating redundant match fields or actions.

We define a *complete rule* based on the OpenFlow multi-table pipeline as follow. A rule consists of one or multiple flow entries in the OpenFlow multi-stage flow tables. If multiple flow entries, each flow entry should belong to different flow tables. In addition, those flow entries use the Goto instruction and the Metadata [2] to form a pipeline. Through the Goto instruction and the Metadata, the pipeline process can find the next flow entry. In fact, each flow entry has a Goto instruction except the last one in the pipeline. If and only if the pipeline process cannot find the next flow entry through the Goto instruction of the current flow entry, its rule is incomplete. Otherwise, the rule is complete which are illustrated in Figure 2. We only think about complete rules in the M-stage multiple flow tables which should be converted by our FlowAdapter. So, it is necessary to detect complete rules in the software data plane before the conversion.

3.2 Equivalence analysis

The FlowAdapter has two major conversion stages, MTO and OTN. One might argue that how the FlowAdapter can ensure its equivalence of conversion . It's well-known that the functions of flow tables are to match packets, modify packets match fields and execute actions. The flow tables can be abstract as a black box. Packets can be taken as inputs and outputs of the black box. So, two kinds of equivalent multiple flow tables mean that they have the same outputs if the same packets enter into the two black boxes respectively. We verify the equivalence of the conversion depended on three equivalence principles as follows - (1) the equivalence of packet match result: (2) the equivalence of packet match fields' modification, (3) the equivalence of packet execution actions.

MTO Equivalence. The MTO is to convert a complete rule from the M-stage multiple flow tables to the One-stage flow table. To meet the three equivalence principles, a

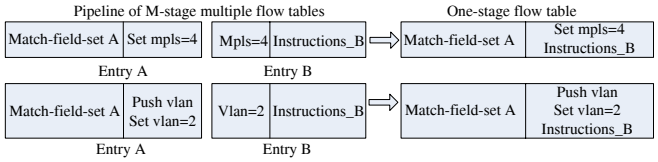

Figure 3: Redundant match fields

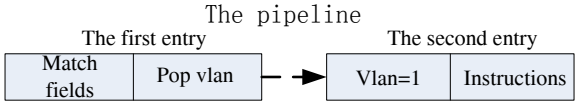

Figure 4: Inconvertible rule

flow entry of the One-stage flow table, representing a rule, should contain equivalent match fields and a instruction list. It means that the match fields of the flow entry should be constructed based on those flow entries in the corresponding pipeline of the M-stage multiple flow tables. It guarantees the packet match consequence equivalence. As for the instruction list, it consists of those instruction lists of the corresponding pipeline in order except pipeline-constructed instructions like the Goto instruction. It guarantees that the packet match fields modification and packet execution actions equivalence. Certainly, the One-stage flow entry also should eliminate its redundant match fields during the construction process.

There are two situations when FlowAdapter can eliminate redundant match fields equivalently, which are illustrated in Figure 3. Situation 1: Flow entry A is in front of flow entry B, which are in the same pipeline of the M-stage multiple flow tables. Entry A has a push-vlan/mpls action in its instruction list. Entry B has a vlan/mpls match field. In addition, there is not any pop-vlan/mpls action in the entries which are between entry A and entry B in the same pipeline. So the mpls/vlan match field is redundant, because this field is always matched. Otherwise the corresponding rule is a error rule, against which cannot be matched by any packet. Situation 2: If a flow table entry in the pipeline of M-stage multiple flow tables has a apply-action instruction which includes a set-field action for a specific type field, we shouldn't add the rest of match fields in the pipeline, which is the same type with the specific field of the set-field action, into the match fields of the One-stage flow table entry.

Unfortunately, we find that a type of rules can not be converted in the process of MTO equivalently, which are illustrated in Figure 4. It means that a packet has two vlan/mpls fields when it enters into the switch. And the pipeline of M-stage multiple flow tables wants to match against the second mpls/vlan match field of the packet. This is a very special situation when we can not convert this complete rule into a One-stage multiple flow table entry directly and equivalently. But, we also present a feasible strategy to address this problem in the architecture of our FlowAdapter. The strategy is that the FlowAdapter can detect the type of rules. And then it will upload them to the software data plane. In other words, those packets matched this type rules should be processed by the software data plane directly.

OTN Equivalence. The OTN also needs to make sure the conversion equivalence. It means that a One-stage flow

entry needs to be split into multiple flow entries, which should be used to construct a new pipeline of the N-stage multiple flow tables. Allowing for the equivalence of packet execution actions and match fields' modification, the One-stage flow entry's instruction list should be put into the last flow entry of the new pipeline. The others flow entries' instruction fields only need to be padded by those pipeline-constructed instructions like Goto and Write-Metadata instructions. For the equivalence of packet match consequence, the match fields of the One-stage flow entry only needs to be spilt into the new pipeline based on the match fields types of each flow table.

3.3 MTO conversion

In this section, we describe how we convert the M-stage multiple flow tables into a One-stage flow table equivalently. The algorithm for the MTO consists of the following three steps which are illustrated in Figure 5: (1) N-Tree Construction: convert the pipeline mapping relation in the M-stage multiple flow tables to its equivalent N-Tree representation. (2) Leaf-Node Obtainment: traverse the N-Tree to get its all leaf nodes which can represent the all complete rules. (3) One-stage flow table Generation: traverse those leaf nodes to the root node for getting the match fields and instructions of all complete rules, construct flow entries for each complete rule and then generate a One-stage flow table.

3.3.1 N-Tree Construction

OpenFlow uses metadata to achieve the pipeline of M-stage multiple flow tables. In addition, more than one flow entry in a flow table may contain the same metadata. So, we represent the pipeline process using a N-Tree data structure. Every node in the N-Tree represents a flow entry which is in the M-stage multiple flow tables. We use these entries which are in flow table 0 as the root node to construct N-Tree respectively. One might argue that why we don't choose those entries which are in other tables except table 0 as the root node. Because pipeline process of OpenFlow starts at the first flow table, flow table 0, and the other tables may be used depending on the outcome of the matched flow entry. If the behavior of table miss is continuing to next flow table of the switch, we'll conduct a default flow entry for each flow table to address this situation when the first flow entry of a complete rule is not in flow table 0. So, we define three types root nodes for constructing the N-Tree as follows. The process is illustrated in Figure 5(b).

For those flow entries, the first type, in flow table 0, We describe the process of N-Tree construction as follows. Firstly, we select a flow entry sequentially in the flow table 0, which has not been used, as the N-Tree root node. Secondly, we check whether the instructions of the entry have the action of Goto-Table. If the answer is no, this N-Tree construction will end. Otherwise, we go to the specified table to match its entries using metadata. And all matched entries are used as the root node's child nodes. Thirdly, we take these child nodes as new root nodes and then construct N-Trees recursively and respectively. Finally, we mark that the root node in flow table 0 has been used.

For the default flow entry in flow table 0, we also select it as a N-Tree root node, the second type. OpenFlow specification tells us that the first flow entry in a complete rule does not have the metadata. So, the N-Tree's child nodes consist of the default flow entry and these flow entries that have

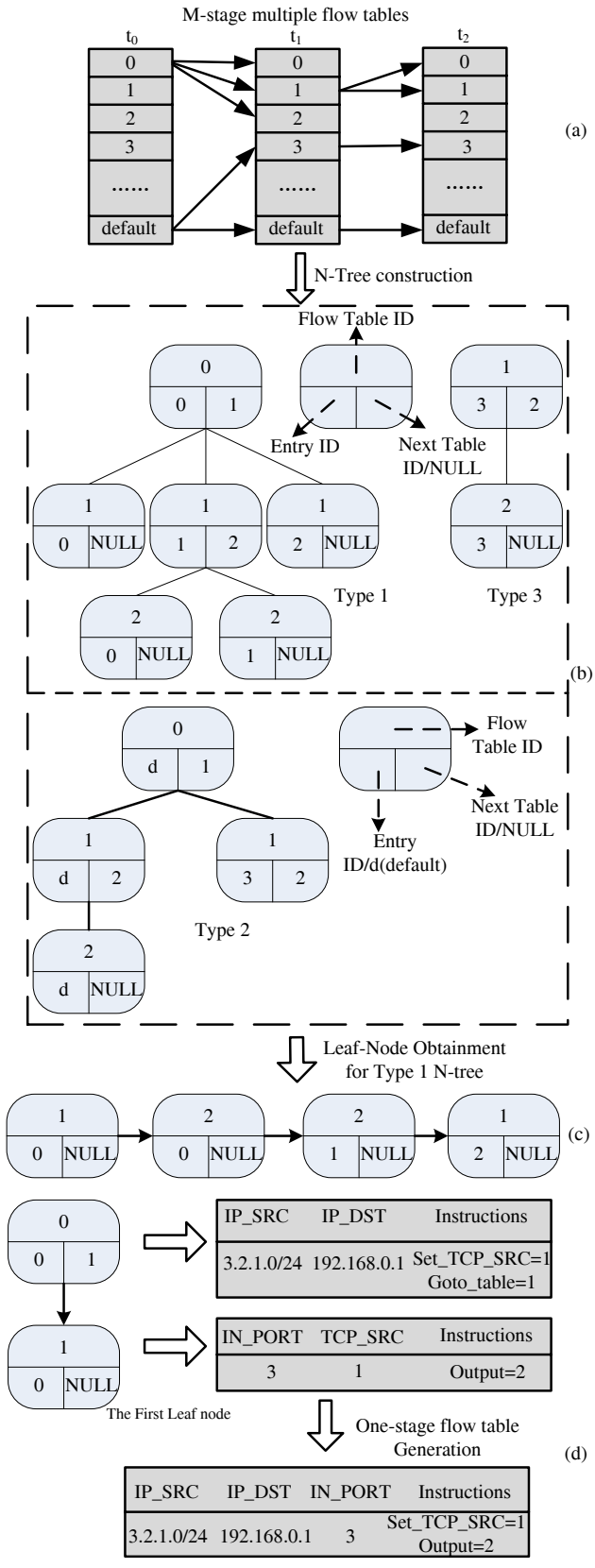

Figure 5: The MTO conversion

no metadata in the next table. And then these child nodes should be taken as new root nodes to construct N-Trees recursively and respectively until the last flow table. Though the above process, we will get a N-Tree that stores the first flow entries of all the complete rules in the M-stage multiple flow tables except starting from flow table 0. Then, we use those first flow entries as root nodes, the third type, to construct N-Tree respectively like the first type root nodes.

However, only the type one and type three root nodes can represent the rules in M-stage multiple flow tables. It's a remarkable fact that this N-tree construction process may have a pruning operation. It means that the node should be removed from the N-Tree when no matched entry is found in the next specific flow table based on the node's Goto-Table action.

3.3.2 *Leaf-Node Obtainment*

After N-Tree construction, the next step is to get all leaf nodes. A complete rule can be mapped as a path from the root to a leaf node. Getting the all leaf nodes means getting the all complete rules which use the root node of the N-Tree as their first entry. The process is illustrate in Figure 5(c).

3.3.3 *One-stage flow table Generation*

After Leaf-Node Obtainment, we have got the all complete rules. So, the next step is to construct a one-stage flow entry for each rule, whose match fields are filled with the corresponding complete rule's match fields in order. Their instruction lists also consist of the complete rule's instruction list in order. Finally, those one-stage flow entries should be inserted into the One-state flow table.

The process of conversion should also pay attention to some aspects as follows. Firstly, those instructions about indicating pipeline relation (e.g. Goto, Write-Metadata instruction) shouldn't be added into the instruction list of the one-stage flow table entry. Secondly, if a flow table entry in the pipeline also has a apply-action instruction which includes pop/push vlan/mpls action, we also should check the rest of flow entry in the pipeline including match fields and instructions based on the previous equivalence analysis. The process is illustrated in Figure 5(d).

3.4 OTN conversion

Above, we described how we convert M-stage multiple flow tables into a One-stage flow table equivalently. However, the switch often needs to contain multiple tables in the practical application scenarios. In addition, the OpenFlow-enabled switch also should support the multi-table pipeline feature to some extent. Next, we describe the conversion of the One-stage flow table into N-Stage multiple flow tables equivalently. The conversion process is made up of 4 steps, which are illustrated in Figure 6 based on previous illustration: flow table initialization, match fields padding, instructions padding and flow entry insertion.

Flow Table Initialization Stage. The construction of N-stage multiple flow tables should be so flexible that the value of N and the match field types of each flow table can be set based on the specific switches. It means that the FlowAdapter should first get the switch pipeline capabilities. This step only need to confirm the number of flow tables and the match types of each flow table. Our FlowAdapter provides a unified interface for programmers. So it can be aware of the pipeline capabilities by the programmers through the unified interface. We suppose that the value of N is 4 and match field types of each flow table are set as showed in Figure 6(a).

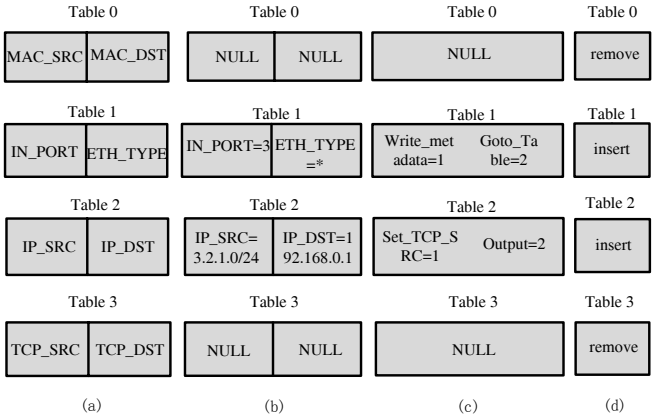

Figure 6: The OTN conversion

Match Fields Padding Stage. After flow table initialization, the next objective is converting One-stage flow entry into multiple flow entries. We don't know in advance the match fields types which the One-stage flow entry contains. So, we construct a initial flow entry for each flow table. The match field types of these flow entries are limited by its corresponding flow table. In addition, the initial flow entries have no any match field. Then we fill the corresponding match field value of the One-stage flow entry to those flow entries based on their containing match field types. The flow entry with no match field padded will be set NULL. The padding process is illustrated in Figure 6(b).

Instructions Padding Stage. In this stage, two kinds of instructions should be added into those entries of the N-stage flow tables. One is these instructions, which are used for supporting pipeline process, like Goto and Write-Metadata instructions. The other is the instruction list of the One-stage flow entry. It is remarkable that only those flow entries, which has been padded match fields, can be taken into consideration the instructions padding. The second type instruction list will be padded in the last N-stage flow entry. The last N-stage flow entry means that the table ID of the flow entry is the biggest among those flow entries which contain match fields. The others should be padded with the first type instructions. Moveover, the instruction fields of flow entries with no match field also be set NULL. The padding processing is illustrated in Figure 6(c).

Flow Entry Insertion Stage. The last step is to add the flow entry to its corresponding flow table. It is worth that the flow entry, whose match fields are NULL, can not insert its flow table. The Insertion processing is illustrated in Figure 6(d).

4. EVALUATION

In this section, we present performance results of our FlowAdapter implementation. It is crucial to complete the conversion process in real-time to enable the network performance not to be affected. Because of lack of OpenFlow rules set, we modify some ACL rule set [12] to fit OpenFlow. Our experiments were performed on a customized machine with

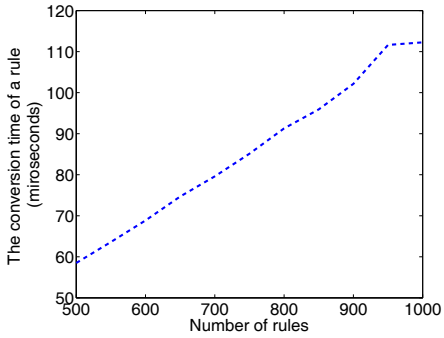

Figure 7: Total conversion time of FlowAdapter

an Intel(R) Xeon(R) CPU with 8 cores running on 64 bit Ubuntu Linux 10.04.3. The result is illustrated in Figure 7.

From Figure 6, we see that FlowAdapter can perform converting within hundreds of microseconds per rule conversion. By limiting the conversion latency with in hundreds of microseconds, it will not affect the network performance. For the HAL, it not only needs to implement rules conversion in the controller, but also achieves the implementation of the HAL in each switch. In addition, it needs to be aware of the HAL models. Our FlowAdapter shares the burden of rules conversion instead of the controller.

5. FUTURE WORK

The algorithm of N-Tree construction is not optimized. It only achieves it by force. So, the N-Tree construction of the FlowAdapter needs to be improved to add or delete nodes incrementally instead of rebuilding the N-Tree in the future. In addition, we need to improve the FlowAdapter to support the type rules of inconvertible by itself instead of the software data plane.

In addition, we will spend more time on stripping our FlowAdpater from the switch. Because the compute resources in the switch are typically scarce. The FlowAdpater will be presented as a strategy to implement an OpenFlow agent.

6. CONCLUSION

In this paper, we design and implement a novel middle layer called FlowAdapter to achieve the conversion of M-stage multiple flow tables to N-stage multiple flow tables. The FlowAdapter is a middle layer between the software data plane and hardware data plane in the switch. In view of the lack of flexibility of the hardware data plane, the FlowAdapter can adapt multi-table rules from the controller to different switch hardware capabilities. The application of the controller can construct and issue rules depending on its own requirement, despite of the various capabilities of flow table pipeline in switch hardware. In other words, the FlowAdapter can enable the flexible multi-table pipeline processing in legacy hardware.

7. ACKNOWLEDGMENTS

This work was supported in part by National Natural Science Foundation of China (NSFC) under Grants 61133015 and 61202411, National High-tech R&D Program of China under Grant 2013AA013501, Strategic Priority Research Program of CAS under Grant XDA06010303 and the Huawei Technologies Co. Ltd.

8. REFERENCES

[1] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review. 38, 69ÍC74 (2008)

[2] OpenFlow switch specification. http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf.

[3] F. Baboescu, S. Singh, and G. Varghese. Packet classification for core routers: Is there an alternative to CAMs? In Proc. IEEE INFOCOM, 2003.

[4] A. Bremler-Barr and D. Hendler. Space-efficient TCAM-based classification using gray coding. In Proc. 26th Annual IEEE Conf. on Computer Communications (Infocom), May 2007.

[5] Q. Dong, S. Banerjee, J. Wang, D. Agrawal, and A. Shukla. Network virtualization: Implementation steps towards the future internet. In Proc. ACM Sigmetrics, pages 311-322, 2006.

[6] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary. Algorithms for advanced packet classification with ternary CAMs. In Proc. ACM SIGCOMM, pages 193-204, August 2005.

[7] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martin Casadao, Nick McKeown and Scott Shenker. NOX: Towards an operating system for networks. In SIGCOMM CRR (2009).

[8] ONF OpenFlow Futures. https://www.opennetworking.org/.

[9] Xie, G., He, P., Guan, H., Li, Z., Xie, Y., Luo, L., Zhang, J., Wang, Y., Salamatian, K. PEARL: a programmable virtual router platform. Communications Magazine, IEEE. 49, 71–77 (2011)

[10] Of12softswitch: An open-source software data plane. https://github.com/CPqD/of12softswitch.

[11] Chad R. Meiners, Alex X. Liu and Eric Torng. TCAM SPliT: Optimizing Space, Power, and Throughput for TCAM-based Packet Classification Systems. Proceedings the 2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems, Pages 200-210.

[12] The rules set of Evaluation Packet Classification, http://www.arl.wustl.edu/ hs1/PClassEval.html