Source Flow: Handling Millions of Flows on Flow-based Nodes

Yasunobu Chiba, Yusuke Shinohara and Hideyuki Shimonishi
System Platforms Research Laboratories, NEC Corporation
1753 Shimonumabe, Nakahara, Kawasaki, 211-8666, Japan
{y-chiba@bq, y-shinohara@cb, h-shimonishi@cd}.jp.nec.com

ABSTRACT
Flow-based networks such as OpenFlow-based networks have difficulty handling a large number of flows in a node due to the capacity limitation of search engine devices such as ternary content-addressable memory (TCAM). One typical solution of this problem would be to use MPLS-like tunneling, but this approach spoils the advantage of flow-by-flow path selection for load-balancing or QoS. We demonstrate a method named “Source Flow” that allows us to handle a huge amount of flows without changing the granularity of flows. By using our method, expensive and power consuming search engine devices can be removed from the core nodes, and the network can grow pretty scalable. In our demo, we construct a small network that consists of small number of OpenFlow switches, a single OpenFlow controller, and end-hosts. The hosts generate more than one million flows simultaneously and the flows are controlled on a per-flow-basis. All active flows are monitored and visualized on a user interface and the user interface allows audiences to confirm if our method is feasible and deployable.

Categories and Subject Descriptors
C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design C.2.6 [**Computer-Communication Networks**]: Internetworking

General Terms
Algorithms, Design, Experimentation, Performance

1. INTRODUCTION
Flow-based networks allow us to control traffic on a per-flow basis. In flow-based networks, a network node has capabilities that (1) identify a flow (based on programmed search criteria) and (2) execute appropriate actions for the matched flow. If we define flows with fine granularity, the number of flow entries in the node that holds “flow matching rule and actions” pairs might be huge. A flow table holding flow entries is usually implemented using a hardware device such as ternary content-addressable memory (TCAM) [4] for achieving high packet-per-second performance and low latency packet forwarding. However, a flow table that stores a large number of flow entries cannot realistically be implemented because of its maximum capacity, power consumption, and hardware cost. The maximum capacity of the TCAM generally available in the market today is 36 Mbits, and it only accommodates 64K to 128K flow entries if the node looks up multiple header fields from layer 2 to layer 4 headers at once as flow matching rules [2]. For handling short-term flows such as HTTP flows in high switching capacity nodes, the maximum capacity might not be enough. The power consumption of typical 18 Mbits TCAM is about 15 watts [7], and its market price is typically more than a hundred US dollars.

We propose a method named “Source Flow” that allows us to handle a huge amount of flows without changing the granularity of flows.

2. OUR METHOD
In our method, we use a typical characteristic of flow entries on a flow table where the number of various combinations of “actions” is usually less than the number of flow matching rules. In such condition, locating such long flow matching rules on all nodes on a selected path is redundant from the system point of view. For example with OpenFlow, [6] the length of flow matching rule is 264 bits fields plus additional 22 bits wildcards but the varieties of various combination of actions are usually small (e.g., 50 varieties) and can be represented by a small number of bits (e.g., 6 bits). We embed the actions for all intermediate nodes as a form of a list into a user packet. Each intermediate node identifies the actions listed in the user packet and simply executes them. This method makes flow lookup and identification on core nodes redundant. Core nodes just need to identify the list of actions that are represented by a smaller number of bits compared to the flow matching rule. The number of flow entries on edge nodes are not reduced but the number of entries on core nodes can be dramatically reduced. The idea looks similar to active networks [1] and IP source routing [5] but our method is simplified and specially designed for flow-based nodes. In addition, our method is actually implemented on today’s Ethernet switches without any hardware modifications. A detailed description of our method follows and is shown in Figure 1.

1. When an ingress edge node receives a packet from a host, it retrieves the list of actions (Figure 1-1) and then encodes the list, and an index counter that is used to identify a list element has to be referenced on each

SIGCOMM’10, August 30–September 3, 2010, New Delhi, India.
ACM 978-1-4503-0201-2/10/08.

intermediate node in the packet (Figure 1-2). Actions in the list can be represented as actual actions (e.g., output to port X and output to port Y) or as a pointer to an action table that stores possible actions on the node. The encoded list is embedded to either a newly defined header field or an existing one. After embedding the encoded list, the edge node sends the modified packet to the next hop node (Figure 1-3).

2. When a core node receives a packet with the list of actions, it identifies actions that have to be run on the node based on the index counter (Figure 1-4). Then, it increments the index counter and executes retrieved actions (Figure 1-5).
3. When an egress edge node receives a packet with the list of actions, it identifies actions that have to be run on the node based on the index counter (Figure 1-6). Then, it executes retrieved actions that may recover the modified packet to the original and output it to the destination host (Figure 1-7).

An action table that stores possible actions on a node can be implemented in the same way as a flow table, and it can be implemented as a part of a flow table. The list of actions might be retrieved from a centrally located controller (e.g., an OpenFlow controller) that governs the overall network topology and nodes.

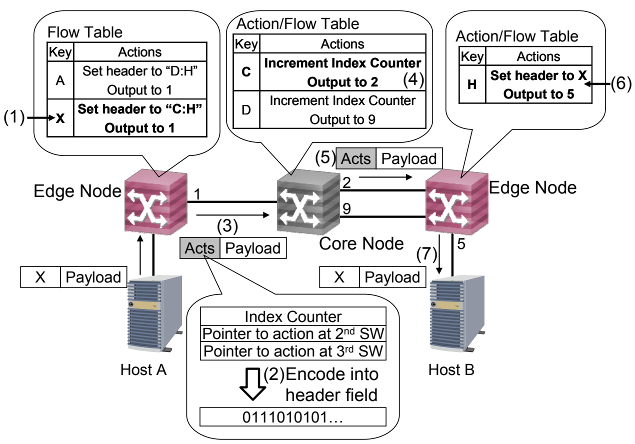

Figure 1. Details of our method

3. DEMONSTRATION

For demonstrating our method, we implemented the method as extensions to OpenFlow. Source Flow capable nodes were implemented based on Open vSwitch [3] and our prototype OpenFlow switch that is based on a commodity switch with small internal TCAMs. We used our prototype OpenFlow controller as a platform of the OpenFlow controller with our method.

We construct a small network for the demo. The network consists of hardware-based OpenFlow switches, x86 servers that accommodate software-based OpenFlow switches and virtual hosts, and a standard PC for an OpenFlow controller.

Virtual hosts that are located on physical hosts generate flows destined for any other virtual hosts. A path for each flow is calculated based on a multipath routing algorithm. Path selection happens for every newly appeared flow. The total number of active flows on the network is expected to be more than one million. All active flows are monitored and visualized in real-time by a user interface (Figure 2) and the user interface allows audiences to confirm if our method is actually working.

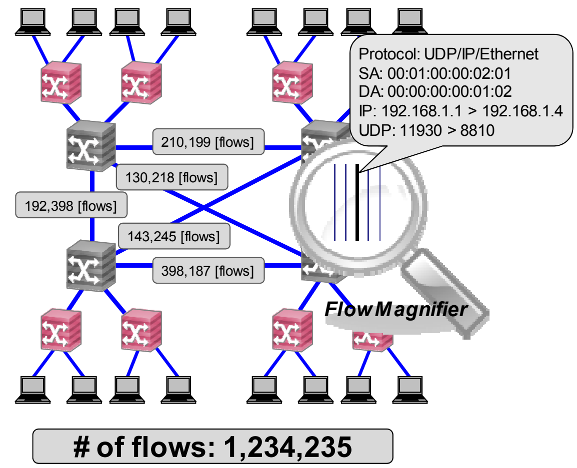

Figure 2. Flow Monitor

4. ACKNOWLEDGMENTS

This work was partly supported by the National Institute of Information and Communications Technology (NICT), Japan.

5. REFERENCES

[1] David, T. et al. 1997. A Survey of Active Network Research. *IEEE Communications Magazine vol. 35 no.1*, 80-86.

[2] NetLogic Microsystems, Layers 2-4 Knowledge-based Processors, http://www.netlogicmicro.com/Products/Layer4/Layer4.htm

[3] Open vSwitch – An Open Virtual Switch, http://openvswitch.org/

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

[5] Postel, J. 1981. Internet Protocol. *RFC791*.

[6] The OpenFlow Switch Consortium. 2009. OpenFlow Switch Specification Version 1.0.0, http://www.openflowswitch.org/documents/openflow-spec-v1.0.0.pdf

[7] Zane, F., Narlikar, G. and Basu, A. 2003. Coolcams: power-efficient TCAMs for forwarding engines. *INFOCOM 2003 vol. 1*, 42-52.