Towards SmartFlow: Case Studies on Enhanced Programmable Forwarding in OpenFlow Switches

Felicián Németh, Ádám Stipkovits, Balázs Sonkoly and András Gulyás[*]
HSN Lab, Department of Telecommunications and Media Informatics
Budapest University of Technology and Economics
{nemethf,stipkovits,sonkoly,gulyas}@tmit.bme.hu

ABSTRACT

The limited capabilities of the switches renders the implementation of unorthodox routing and forwarding mechanisms as a hard task in OpenFlow. Our high level goal is therefore to inspect the possibilities of slightly smartening up the OpenFlow switches. As a first step in this direction we demonstrate (with Bloom filters, greedy routing and network coding) that a very limited computational capability enables us to natively support experimental technologies while preserving performance. We distribute the demos[1] in source files and as a ready-to-experiment VM image to promote further improvements and evaluations.

Categories and Subject Descriptors

C.2.6 [**Computer-Communication Networks**]: Internetworking—*routers*

Keywords

SDN, OpenFlow, Bloom filters, Greedy routing, Network Coding

1. INTRODUCTION

Since its first proposition, OpenFlow (OF) [1] has become the key enabler and driver of innovation in networks. By now, numerous innovative applications have appeared on top of OF, targeting mostly the well-known issues of packet switched networks. The spectrum is spanning over protocols, routing, traffic engineering technologies and security. These sets of applications rely on the standard philosophy of OF, namely to execute forwarding-related tasks in the switches and move the control-specific operations completely into the controller. While such a scheme is satisfactory for a whole bunch of applications, we demonstrate that this view is somewhat limited when the support for unorthodox network technologies are considered.

In our demo we present OF testbeds for experimental technologies with non-negligible future potential. These testbeds will require increasingly more functionality from the OF switches. First we show that *Bloom filter* based forwarding [2] can be implemented in a *standard manner* over OF version 1.1[2], if the flow entries of the switches are interpreted in an *unorthodox* way. Secondly, we introduce the OF implementation of *"greedy" or geographical* [3] routing and show that although its implementation can be done in a standard fashion (using OF controllers), the beneficial properties of this basically distributed technology can *only be exploited by adding some computational capability to the switches.* Finally we present a testbed for inter session *network coding* [4] and demonstrate that if we want to apply this technology in OF then the usage of *smarter switches is imperative.* Our case studies support that introducing only lightweight and low-level *enhancements*, numerous networking operations may become available in OF. For the universal treatment of such enhancements we envision a generic language for switch computations similar to the vertex shading languages of video cards.

[*]MTA-BME Future Internet Research Group
[1]Available from http://github.com/nemethf/sigcomm2012

2. CASE STUDIES

Stateless multicast with Bloom filters: We assign Bloom IDs to switch ports and put a Bloom filter containing the packet's path into the packet's destination Ethernet address field. The Bloom filter is generated by OR-ing the Bloom IDs of the ports that the packet needs to pass through.

Our testbed shows that a switch can be "forced" to follow a Bloom filter based forwarding mechanism relying solely on standard OF 1.1 protocol[3], however we need to slightly deviate from the regular usage of flow entries: normally, flow entries correspond to specific set of *flows*, whereas we proactively configure a separate flow table for each *port* as shown in Table 1. The first entry matches if the packet's Bloom filter contains the port's Bloom ID. In case of a match the switch forwards the packet via the corresponding port and independently of the matching result, the packet is matched against the next port's flow table hereby providing the stateless multicast switching capability.

Fix sized flow tables with Greedy routing: Greedy routing relies on embedding of the switches by assigning them coordinates in a predefined metric space. A switch then forwards packets towards its neighbor closest to the packets' destination[4]. Knowing the topology and the coordinates of each node, the controller can configure the flow tables of the switches (see Table 2). Afterwards, routing operates without controller supervision.

[2]Note that this is not true for OF 1.0
[3]Currently, commercial OF switches support ver. 1.0 only.
[4]We store 2-dimensional coordinates in Ethernet addresses.

Table 1: Bloom filter in OF. Flow entries in the ith flow table.

match condition	associated actions

addr & BloomID$_i$ =BloomID$_i$	output(port$_i$)
	goto-table(i+1)

(always match)	goto-table(i+1)

Table 2: A single flow entry implementing greedy routing. address-n1...nN stands for the coordinates of the switch's neighbors.

match condition	associated actions

(always match)	write-metadata(max. number, invalid port)
	update-distance(address-n1, port-to-n1)
	update-distance(address-n2, port-to-n2)
	...
	update-distance(address-nN, port-to-nN)
	output-by-metadata

For enabling greedy routing we need only two simple extra actions. These experimenter actions assume that the metadata register holds two values: a distance and a port number. The action *update-distance* calculates the distance between its first argument and the (geographical) destination address of the packet being processed. If the distance is smaller than the one stored in the metadata register, then it saves the distance and its second argument in the metadata register. Our second general purpose action, *output-by-metadata* forwards packet to the port stored in the metadata.

Obviously, the routing decision can be implemented as a controller application without any need for protocol extensions. However, controller based greedy routing results in slower connection setup and error recovery, and more importantly the flow table size is proportional to the actual number of flows. On the other hand, by using enhanced switches, the table size is proportional only to the number of ports and the forwarding speed is independent of the traffic load.

200% link utilization with Network Coding: Network coding (NC) may mix or encode packets before forwarding, hereby improving resource efficiency and robustness [4]. However, implementing NC as an OF controller logic is practically impossible since every packet would have to be sent back and forth to the controller for en-/decoding. We extended the OF protocol with three simple actions to support XOR-based mixing of two flows. NC metadata is sent in three MPLS labels following the format proposed in [5] (Fig. 1). The outermost label is a flow identifier that can be used as a matching condition in the flow table. Sequence numbers can be prepended to packets by the action *set-mpls-label-from-counter* and two flows can be combined with the *XOR-encode* action. XOR-encode does not alter the packet in the OF processing pipeline, it creates a copy of the packet and puts it in the encoding queue or if there is a pending packet from the other flow, then it mixes them, replaces the MPLS flow-id, and sends the resulting packet through the pipeline. This final step ensures general applicability because further processing of the newly created packet can be specified with standard flow table entires. The *XOR-decode* action works similarly, but it searches match-

MPLS label: flow-id	MPLS label: seq.no 1	MPLS label: seq.no 2	data

Figure 1: NC packet format. If seq. numbers are both non-zero, then the original packets are *xor*ed in data.

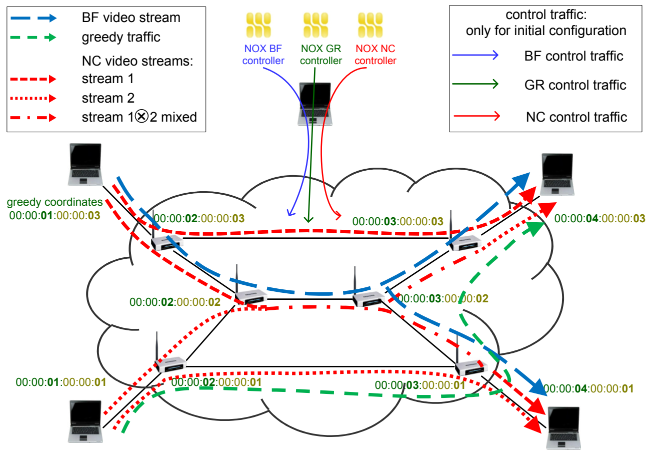

Figure 2: Demonstrated topology.

ing sequence numbers when it dequeues a packet from the decoding queue.

These three new OF actions are quite simple, but in the meantime they are general enough to support several NC use-cases defined in [4], e.g., intersession multicasting, delay or energy minimalization. To the best of our knowledge this will be the first NC implementation in OF and we hope that our case study will trigger further developments and discussions about the NC-OF interworking.

3. DEMO SCENARIO

For demonstration we have ported our extensions into OpenWRT firmware and built a testbed with TP-Link devices.[5] The demo topology shown in Fig. 2 can accommodate all of our testbeds. Video streams are sent using different forwarding schemes in a configurable manner. Changing the active controller results in switching among the three network operations.

4. REFERENCES

[1] N. McKeown et al. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM CCR*, 38(2), 2008.

[2] P. Jokela et al. LIPSIN: Line speed publish/subscribe inter-networking. *ACM SIGCOMM CCR*, 39(4), 2009.

[3] M. Boguna, F. Papadopoulos, and D. Krioukov. Sustaining the internet with hyperbolic mapping. *Nature Communications*, 1(6), 2010.

[4] P. A. Chou and Y. Wu. Network coding for the internet and wireless networks. *IEEE Signal Processing Magazine*, 24(5), 2007.

[5] T. Biermann, A. Schwabe, and H. Karl. Creating butterflies in the core – a network coding extension for MPLS/RSVP-TE. In *Proc. of Networking*, 2009.

[5]Our implementation extends OF 1.1 software switch by TrafficLab, Ericsson Research.