On IPv6 support in OpenFlow via Flexible Match Structures

Rodrigo R. Denicol,
Eder L. Fernandes,
Christian E. Rothenberg
CPqD - R&D Center for Telecommunications
Campinas - SP - Brazil
{rdenicol,ederlf,esteve}@cpqd.com.br

Zoltán Lajos Kis
Ericsson Research
Budapest - Hungary
zoltan.lajos.kis@ericsson.com

1. INTRODUCTION

As is so common in technologies, low cost/performance and flexibility conflict with each other. While software-only routers lack of performance, network processors are regarded as too complex to program, and hardware-based designs (including commodity silicon) have been too inflexible [4]. Recognizing the different tradeoffs between these goals, recent efforts [4, 7] advocate for a path towards flexible and efficient networking gear based on co-evolving router hardware, extensible router software, and a clean interface between them. For the latter, OpenFlow [3] is considered the best candidate API available to unveil true innovation in networking.

Up to recently, the OpenFlow specification has worked with fixed tupples (10 in v1.0 and 12/14 in v1.1) limiting the available match fields to a hardwired subset of fields in the protocol stack. The rigid flow match structure specification is considered a risk to the protocol success and endangers the promise of future-proof independent evolvability of the data and control planes. To turn this over, OpenFlow 1.1 introduces experimenter features that allow extensible matches, actions, messages, and errors. The ability of extending the flow match structure by defining new types of Flow Match Structures (see A.2.3 [3]) allows controllers and switches to agree on any flow matching syntax.

IPv6 [6] was designed with extensibility in mind. In addition to more efficient forwarding,the improved support for extensions and options in IPv6 allows a greater flexibility for introducing new options in the future. Basically, the functionality of options and other rarely used fields is removed from the main header in IPv4 and implemented in IPv6 through a set of additional headers called extension headers (EH), some of them based on a variable number of "options" encoded in TLV format.

When IPv6 EHs are to be supported in OpenFlow, a packet field above the base IP header may then occur at different bit positions. The extra switch complexity required for IPv6 EH has two main disadvantages: One is an obvious increase in switch production costs. The other is a higher probability that switch functionality becomes obsolete due to the introduction of new, or changes to existing IPv6 EHs. The required switch upgrades may jeopardize the cost savings that OpenFlow aims to enable.

In this poster, we show our ongoing work towards solving part of one OpenFlow limitation: lacking of a flexible, extensible flow match structure. Our work takes on the TLV-based structure NXM (Nicira eXtended Match) [1]. IPv6 traffic with multiple EHs is used as the test case to validate the proposed extensible flow structure. Our software-based prototype switch and NOX controller are based on OpenFlow 1.1 and leverage the Network Protocol Description Language (NetPDL) [9] to implement the parsing engine. NetPDL provides a standardized syntax based on XML upon which new protocol encapsulations, headers and fields can be easily introduced.

2. DESIGN AND IMPLEMENTATION

Figure 1 shows the basic building blocks of our OpenFlow 1.1 enabled setup used to validate NXM as a means to extend OpenFlow to support IPv6 traffic. Both the OpenFlow 1.1 NOX controller and the software switch [2] implement the library (oflib) that holds the internal representations for each OpenFlow entity (messages, actions, etc.) and provides the required conversion functions. Functions like `ofl_msg_unpack` and `ofl_msg_pack` have been extended to support the newly typed flow match structures.

The second important block to be enhanced is the parsing engine that decodes the protocol stack, extracts the packet header fields and stores them in an internal data structure. Parsing needs to be done not only in the switch datapath but also in the controller to extract the flow fields from packet-in events and handle them to the application. Due to its extensibility, parsing IPv6 traffic can be specially cumbersome,[1] and the complete process becomes more challenging as the number of supported fields and encapsulations increases.

In the following, we detail the focus of this poster, namely the two pieces of work that contribute to protocol flexibility and extensibility.

2.1 Flexible Flow Match Structure

While OpenFlow 1.0 used a fixed-length `struct ofp_match` for specifying flow matches, version 1.1 of the protocol introduces a `ofp_match_type` field in the flow match structure that allows introducing new extensions by defining new

[1]Because of this, IPv6 with EHs are commonly handled in the software-based slow path of commercial equipments.

CHANGE & OFELIA Summer school '11 Nov 7-11, 2011, Berlin, Germany

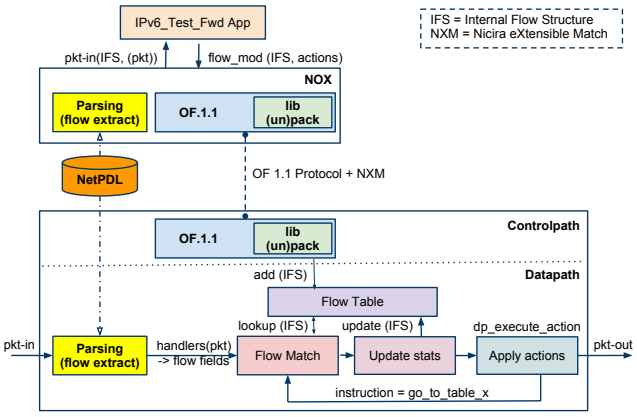

Figure 1: Building blocks of the NXM-extended OF.1.1 switch and controller proof of concept implementations supporting IPv6 traffic handling.

```
struct ofp_ext_match {
    uint16_t type;                  /* One of OFPMT_* */
    uint16_t length;                /* Length of ofp_ext_match in bytes */
    uint8_t pad[4];                 /* Align to 64 bits */
    struct flex_array *match_fields; /* Match fields. 64-bit aligned */
};
struct flex_array {
   uint16_t size;  /* Length of the entries array in bytes */
   uint16_t total; /* Number of entries */
   uint8_t pad[4]; /* Align to 64 bits */
   uint8_t entries[]; /* TLV entries. 64-bit aligned */
};
```

Figure 2: Extensible flow match structure.

match types. Figure 2 shows the proposed extensible flow match structure, called `ofp_ext_match` and based on a flexible array structure. Each element in the array contains a variable-length TLV entry that follows the NXM format. In NXM, a 32-bit header (Fig. 3) acts as the TL part that identifies the vendor/experimenter, the packet field, the presence of a mask (hm – has mask), and the length in bytes of the value plus optionally mask fields. Figure 4 exemplifies one NXM entry encoding a masked MAC address.

2.2 NetPDL

The Network Protocol Description Language (NetPDL) [9] is an extensible, XML-based language for describing the format of protocol headers and the encapsulation rules between different protocols. NetPDL supports a myriad of protocols and cyclic encapsulations (e.g., an IPv4 packet tunnelled within another IPv6 packet). As many other technologies (e.g., XDR, ASN.1, IDL), NetPDL can be used to generate C code implementing packet processing [8], thus being able to run this code (natively) at very high speed.[2] By including the NetPDL-based packet decoder C application [5] as an independent parsing module in the architecture (both the controller and the switch), new protocols can be easily supported by updating the XML-based protocol database. Since an IPv6 header possibly consists of various optional headers; the number and type is not known in advance. NetPDL has shown to be very effective in extracting fields from complex IPv6 header combinations by defining `<loop>`, `<switch>`, `<block>` and `<includeblk>` elements.

[2]For hardware environments, an interesting line of research would be the automatic generation of NP or VHDL code.

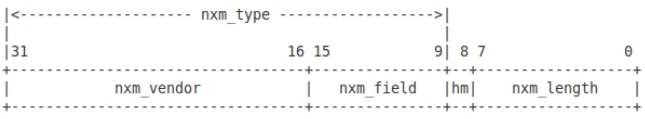

Figure 3: The NXM header is interpreted as a 32-bit word in network byte order.

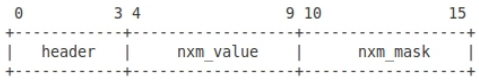

Figure 4: NXM-encoded Ethernet address with mask (48-bit nxm_value, hm=1, nxm_length=12).

3. CONCLUSIONS AND PATH AHEAD

IPv6 extensibility itself brings OpenFlow limited extensibility under the spotlight. We evaluated NXM as a valid approach to have switches and controllers agree on the flow match syntax. Moreover, a protocol database like NetPDL makes this coordination simpler if not automatic. While the approach works fine in software, further considerations are required in hardware (e.g., "extensible parsing", TCAM). Fixing the hardware forwarding logic with true independence of the protocol definitions is as ambitious as challenging. If achievable at least to some extent, then controllers could specify bit positions (i.e., "flexible fields"), instead of packet fields, for conditions and actions, effectively moving to the controller any extra functionality required to parse IPv6 EHs and other protocols. Closer in the OpenFlow protocol roadmap seems the adoption of new flexible data structures such as TLVs, which may be adopted in further aspects of the protocol for consistency and maximum extensibility.

4. REFERENCES

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

[2] Openflow 1.1 SoftSwitch. https://openflow.stanford.edu/display/of11softswitch/Home.

[3] OpenFlow Switch Specification v1.1.0 Implemented. http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf.

[4] M. Casado, T. Koponen, D. Moon, and S. Shenker. Rethinking packet forwarding hardware. In *HotNets*, October 2008.

[5] Computer Networks Group (NetGroup) at Politecnico di Torino. The NetBee Library. http://www.nbee.org/.

[6] S. Deering and R. Hinden. *RFC 2460 Internet Protocol, Version 6 (IPv6) Specification*. IETF, Dec 1998.

[7] J. C. Mogul, P. Yalag, J. Tourrilhes, R. Mcgeer, S. Banerjee, T. Connors, and P. Sharma. API design challenges for open router platforms on proprietary hardware. In *HotNets*, October 2008.

[8] O. Morandi, F. Risso, M. Baldi, and A. Baldini. Enabling flexible packet filtering through dynamic code generation. In *ICC*, pages 5849–5856. IEEE, 2008.

[9] F. Risso and M. Baldi. Netpdl: An extensible xml-based language for packet header description. *Comput. Netw.*, 50:688–706, April 2006.