Realizing Source Routed Multicast Using Mellanox’s Programmable Hardware Switches

Matty Kadosh\textsuperscript{1}, Yonatan Piasetzky\textsuperscript{1}, Barak Gafni\textsuperscript{1}, Lalith Suresh\textsuperscript{2}, Muhammad Shahbaz\textsuperscript{3}, Sujata Banerjee\textsuperscript{2}

\textsuperscript{1}Mellanox, \textsuperscript{2}VMware, \textsuperscript{3}Stanford
Group Communication in Public Clouds

10,000s of tenants
→ 100s of workloads

Millions of distinct group communications

- Replication for databases and state machines
- Publish-Subscribe like ZeroMQ and RabbitMQ
- Infrastructure Apps like VMware NSX and OpenStack
Limitations of Native Multicast

- Excessive control churn due to membership and topology changes
- Limited state in switches for group entries < 10K

Data Center Controller
Processing overhead
Restricted to Application-Level Multicast

Traffic overhead

Processing overhead

Low throughput

P Publisher  S Subscriber
Elmo: Source Routed Multicast \rightarrow SIGCOMM’19

Group Table Entry:
ID \rightarrow Ports
Elmo: Source Routed Multicast

Packet Rules or p-rules:
List of \((\text{Switch ID} \rightarrow \text{Ports encoded as bitmaps})\) tuple as well as a default p-rule.

P Publisher  S Subscriber
Elmo: Source Routed Multicast
Elmo: Source Routed Multicast

- Read a p-rule
- Read the default p-rule

Diagram: Upstream and Downstream data flow with Publisher (P) and Subscribers (S) connected through routers.
Mellanox Programmable switch model

- Hybrid – Integration between legacy (switch router) and programmable pipeline
- NOS (ONYX / SONiC) and user applications run in parallel
Options parser

- Options are common among network protocols (IPv4, TCP, etc.)

- Options follow some common structure
  - Base header has a known length
  - Total header length (computed)
  - Total options length
  - Options are built in a TLV fashion:
    - Type (self-indicator)
    - Length (some granularity)
    - Type and Length fields are fixed
  - This structure mainly exists to support unknown options
  - State transition is defined in the base header

- In Elmo:
  - Downstream P-rules are options
  - Unknown switch ID
  - Default p-rule – common Switch ID
Single switch Functionality

- Upstream bitmap
- Downstream bitmap
- Default p-rule
  - Increase scale on the expense of excess traffic
- Normal forwarding by the legacy pipeline
Mcast groups:

1. H1 transmits to H2:
   - UP bitmap: [000, 010]
   - no DP header

2. H1 transmits to H3, H4 and H6:
   - UP bitmap: [001, 001]
   - DP: [S1: 001, L2: 101]

3. H1 transmits to H2, H4, H5:
   - UP bitmap: [001, 010]
   - DP: [S1: 001, L2: 110]

4. H1 transmits to H5, H6:
   - UP bitmap: [001, 000]
   - DP: [S1: 001, default: 011]
**Mcast groups:**

1. H1 transmits to H2:
   - UP bitmap: [000, 010]
   - no DP header

2. H1 transmits to H3, H4 and H6:
   - UP bitmap: [001, 001]
   - DP: [S1: 001, L2: 101]

3. H1 transmits to H2, H4, H5:
   - UP bitmap: [001, 010]
   - DP: [S1: 001, L2: 110]

4. H1 transmits to H5, H6:
   - UP bitmap: [001, 000]
   - DP: [S1: 001, default: 011]
Demo
Challenges

- In the following slides, we’ll share our experience from this work.
- Challenges encountered during this work:
  - Multicast
  - Options parsing
  - Extraction
Multicast

- Multicast is not handled by the PSA model (extern)

- Hard for stateless switch multicast

- This work - directly expose MC bitmap to the dataplane:
  - metadata.egress_ports = headers.elmo_downstream_default_p_rule.bitmap;

- Multicast group table can be easily supported

- Hybrid architecture - support non-physical ports as well (e.g., router interface etc..)
Options parsing

- Options current implementation in P4
  - possible but not trivial
  - Not easily offloadable

- Common use case
  - Worthwhile to have standard fashion of defining
  - Easily HW offloaded by the different vendors.

- Build a sub-parser prototype which follows the observed structure
Extractions

- Extract methods act on entire headers – consumes them and advances the cursor.

- It is further assumed that HW will extract all the fields of the accepted header.
  - Could be costly.

- What if you require a subset of the fields?

- May prevent HW optimizing by selective.
  - Dynamically loaded control.

- Advanced field extraction features like:
Extraction - proposal

- We implemented field extraction primitives on our architecture:
  - Void extract_field<T>(out T headerLvalue.field);
  - void extract_field<T>(out T headerLvalue.field, in bit<32> variableFieldOffset);
  - void extract_field<T>(out T headerLvalue.field, in bit<32> variableFieldSizeInBits, in bit<32> variableFieldOffset);

- Extract a single field and advance the cursor,
  - Adds to current header primitives (not replace)

- Useful also for:
  - variable offset fields
  - more than one variable length field in a header

- Another option - Usage analysis in the compiler backend
  - Sufficient for monolithic P4 executables

  - Problematic for target architectures which allow dynamic insertion of control pipelines (which share the same parser)
Conclusions

- Elmo compactly encodes multicast policy inside packets

- Designed for multi-tenant data centers scales

- Demonstrated, for the first time, Elmo implementation with wire speed performance using hybrid programmable dataplane

- All legacy forwarding and control plane is intact
Thank You

Matty Kadosh (Mellanox), mattyk@mellanox.com
Lalith Suresh (VMware), lsuresh@vmware.com

Acknowledgement
Jen Rexford¹
Nick Feamster²
Ori Rottenstreich³,⁵
Mukesh Hira⁴, Mihai Budiu⁴, Ben Pfaff⁴
Alan Lo⁵, Aviv Kfir⁵, Jose Yallouz⁵