AVA

Universität Rostock

Traditio et Innovatio

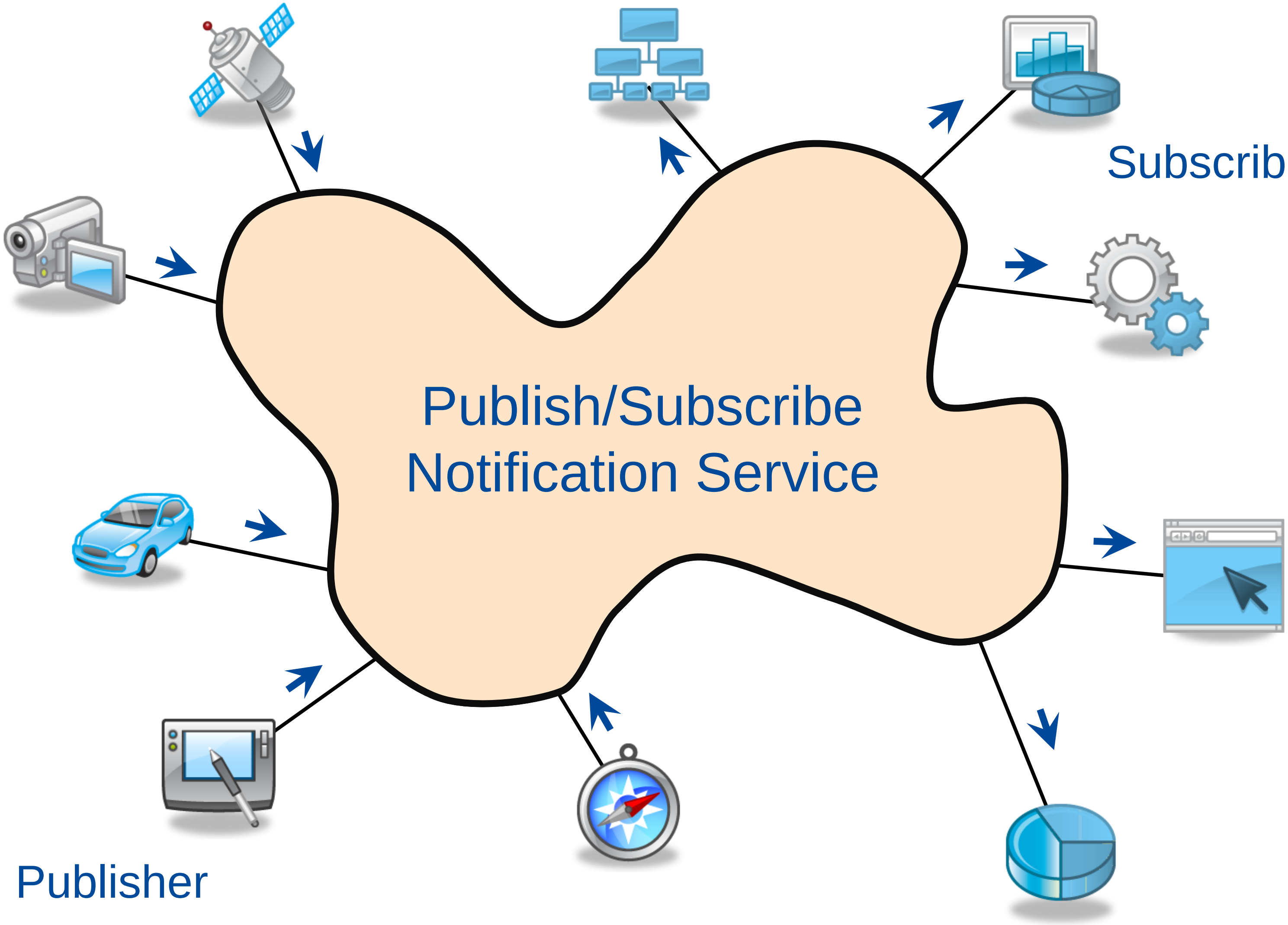
P4-programmable Data Plane for Content-based Publish/Subscribe

Christian Wernecke, Helge Parzyjegla, Gero Mühl

Architecture of Application Systems (AVA)
Faculty for Informatics and Electrical Engineering
University of Rostock

Agenda

- Publish/Subscribe
- Motivation
- Pub/Sub with P4
  - Source-based Routing
  - Referring Forwarding Trees
  - Stitching Forwarding Trees
- Evaluation
- Conclusions


Subscriber
Publish/Subscribe Notification Service
Publisher

Content-based Pub/Sub

- Brokers enable flexible filtering and forwarding
  - Implementation on application layer
  - Support arbitrary filter expressions
- Brokers increase latency
  - Forwarding latency between switch and broker host
  - Delay for traversing the OS network stack
  - Scheduling delay on broker host
  - Costs for serialization, deserialization, and filtering

Possible data plane programming for pub/sub?

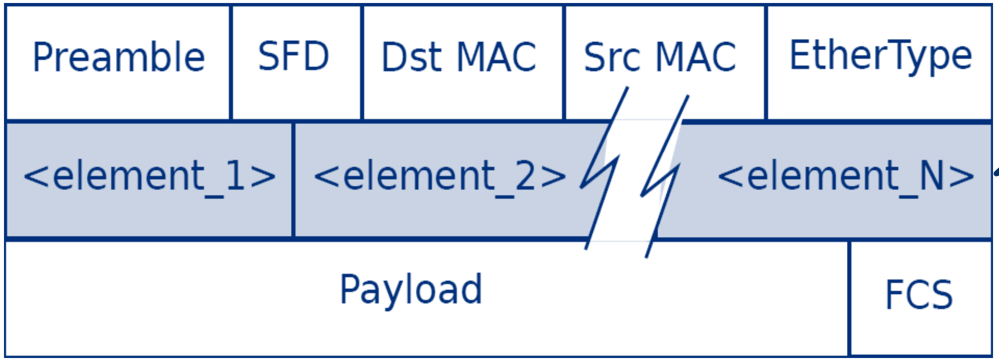
Motivation and Approaches

> Challenges for content-based publish/subscribe
  > Different set of receivers for each notification
  > Exponential number of receiver combinations
  > Difficult to predict efficient multicast groups

1. Source-based Routing
   > Whole distribution tree encoded into packet header
2. Referring Forwarding Trees
   > Stored distribution tree for stable subscribers of a publisher
   > Additional information in packet header to extend stored tree
3. Stitching Forwarding Trees
   > Forward along multiple distribution trees
   > Information in packet header truncates or bridges/extends trees

How to encode distribution trees?

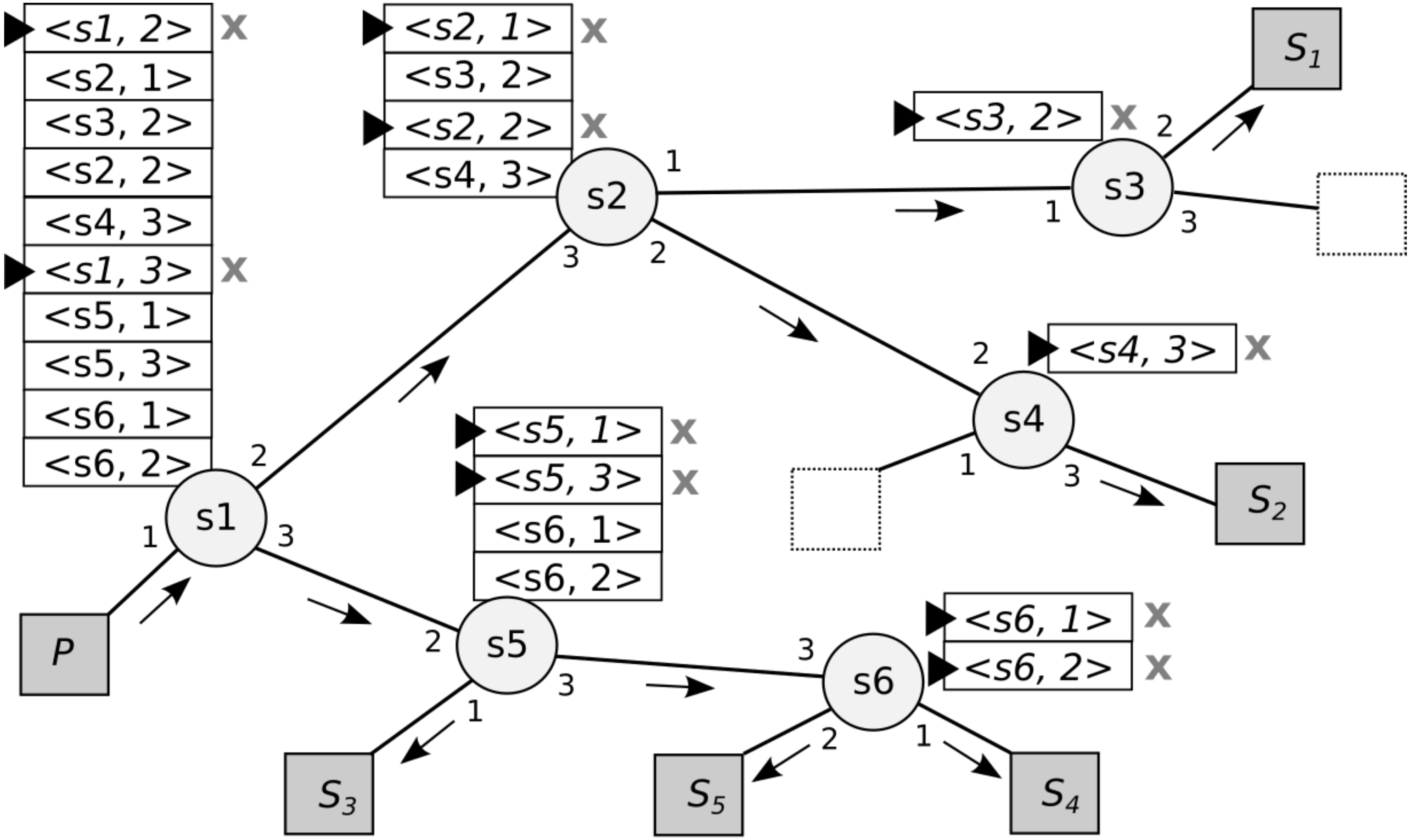
P4 – Programming Language



Custom defined header fields between data link and network layer.

- Imperative, protocol-independent, target-independent
- Provides packet parsing and processing mechanisms
- Allows definition of own parsers and header fields
- → specification of custom protocols for pub/sub
  1. Routing information in header stacks
  2. Stored distribution trees as flow rules in switches

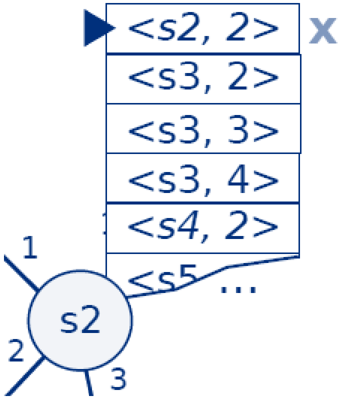
Source-routed Multicast



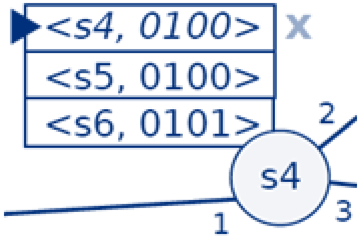
▶...entry used for forwarding x...entry removed from stack

Source-routed Multicast Strategies

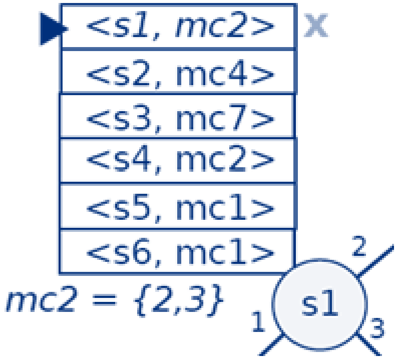
1. Output-ports



2. Bitmasks



3. Multicast groups



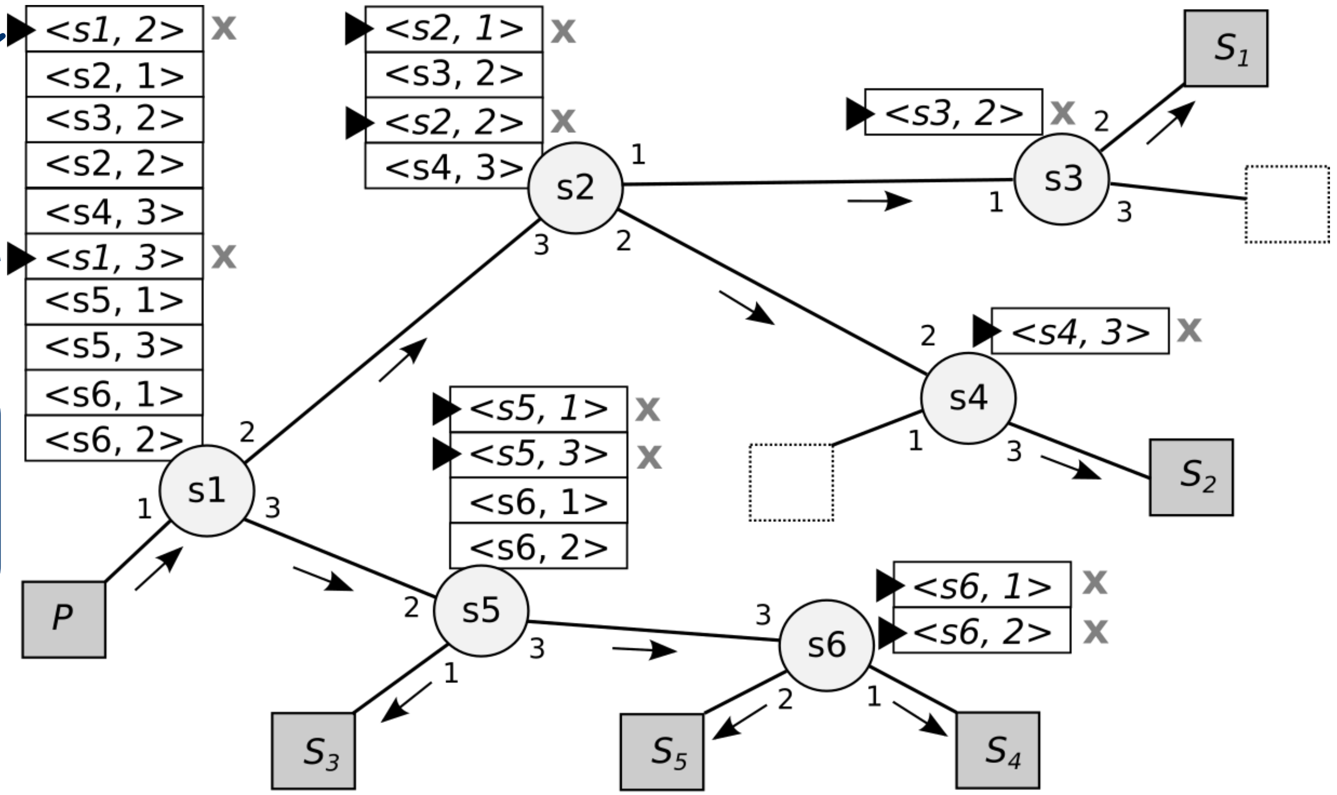
Distribution by Output Ports

Stack contains whole distribution tree at first switch

Stack contains only information for subtree s2

No more stack elements at final destination

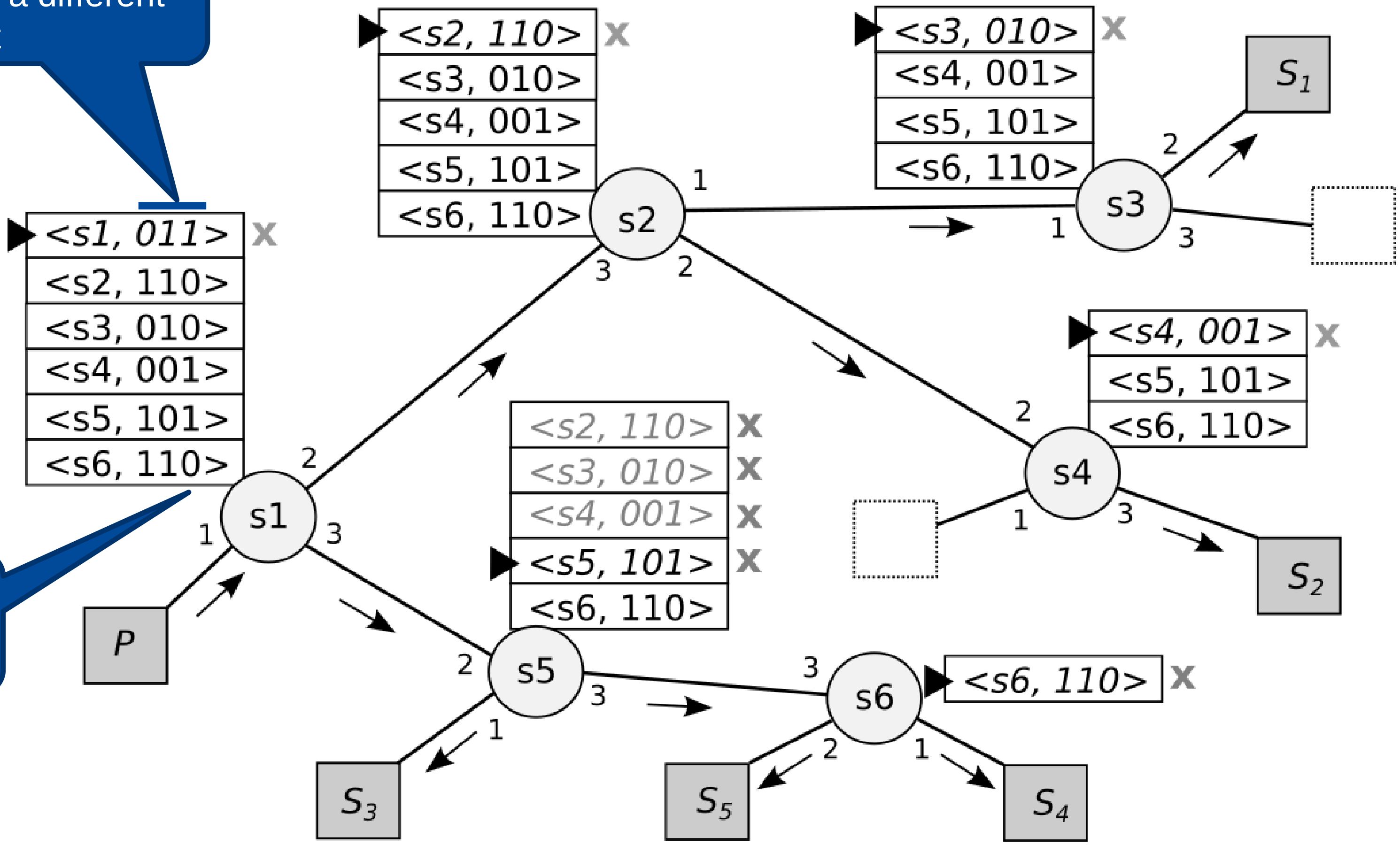
Forwarded duplicates only contain entries for their subtrees



▶...entry used for forwarding x...entry removed from stack

Distribution by Bitmask

Each bit of bitmask represents a different output port

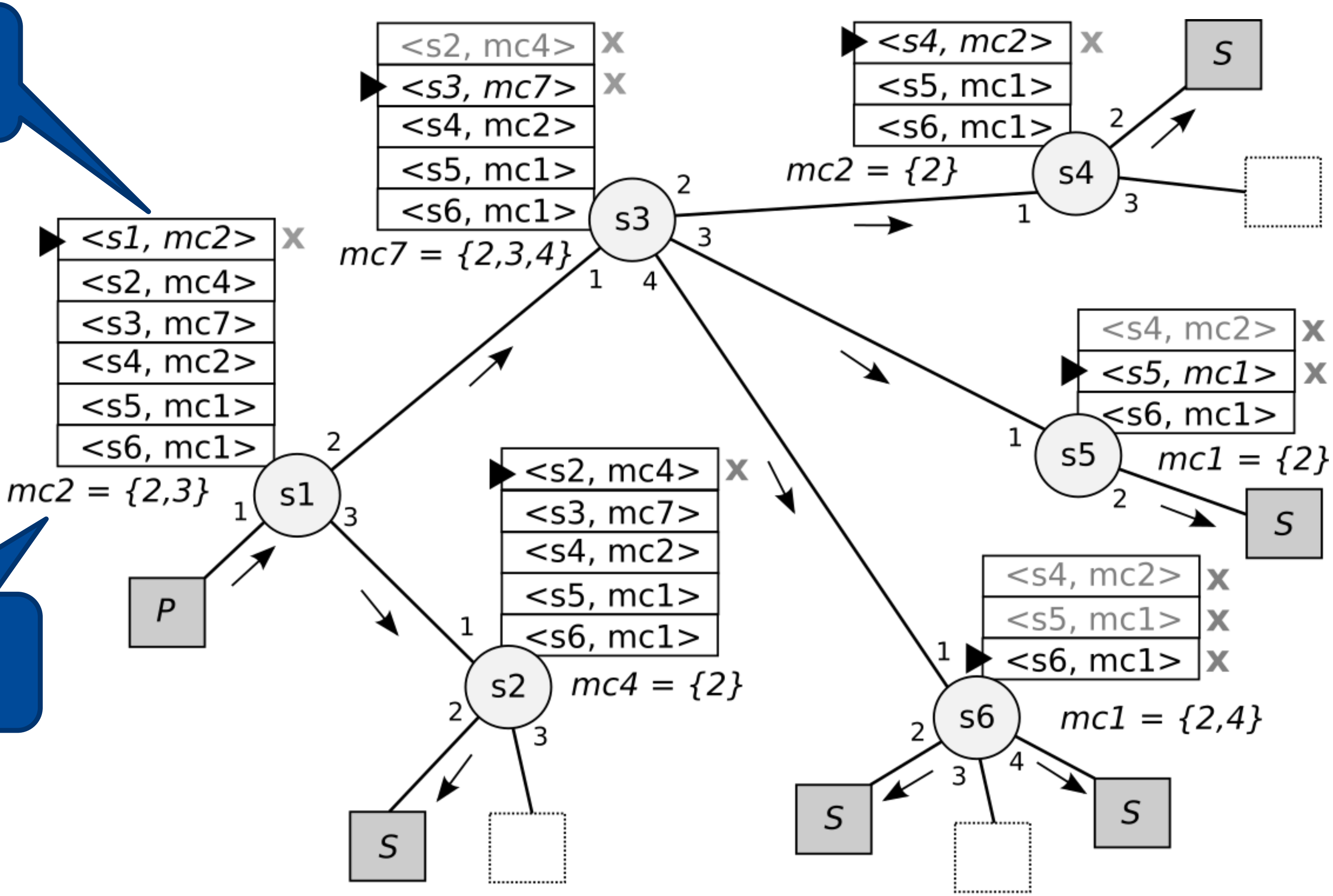


Fast skipping of zero bits

▶...entry used for forwarding x...entry removed from stack

Distribution by Multicast Groups

One stack entry for each switch



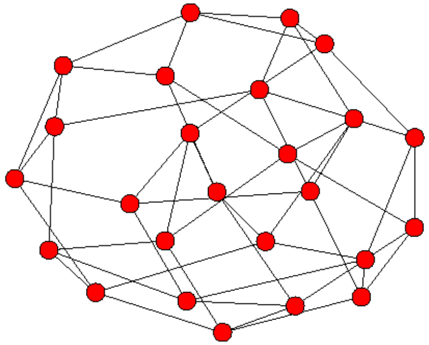
Preinstalled local multicast groups

▶ ... entry (multicast-group) used for forwarding

x ... entry removed from stack

Evaluation of source-routed Multicast

- Jelly-Fish topology with 255 switches (each connected to a host)
- Mininet with the P4 software switch behavioral model version 2

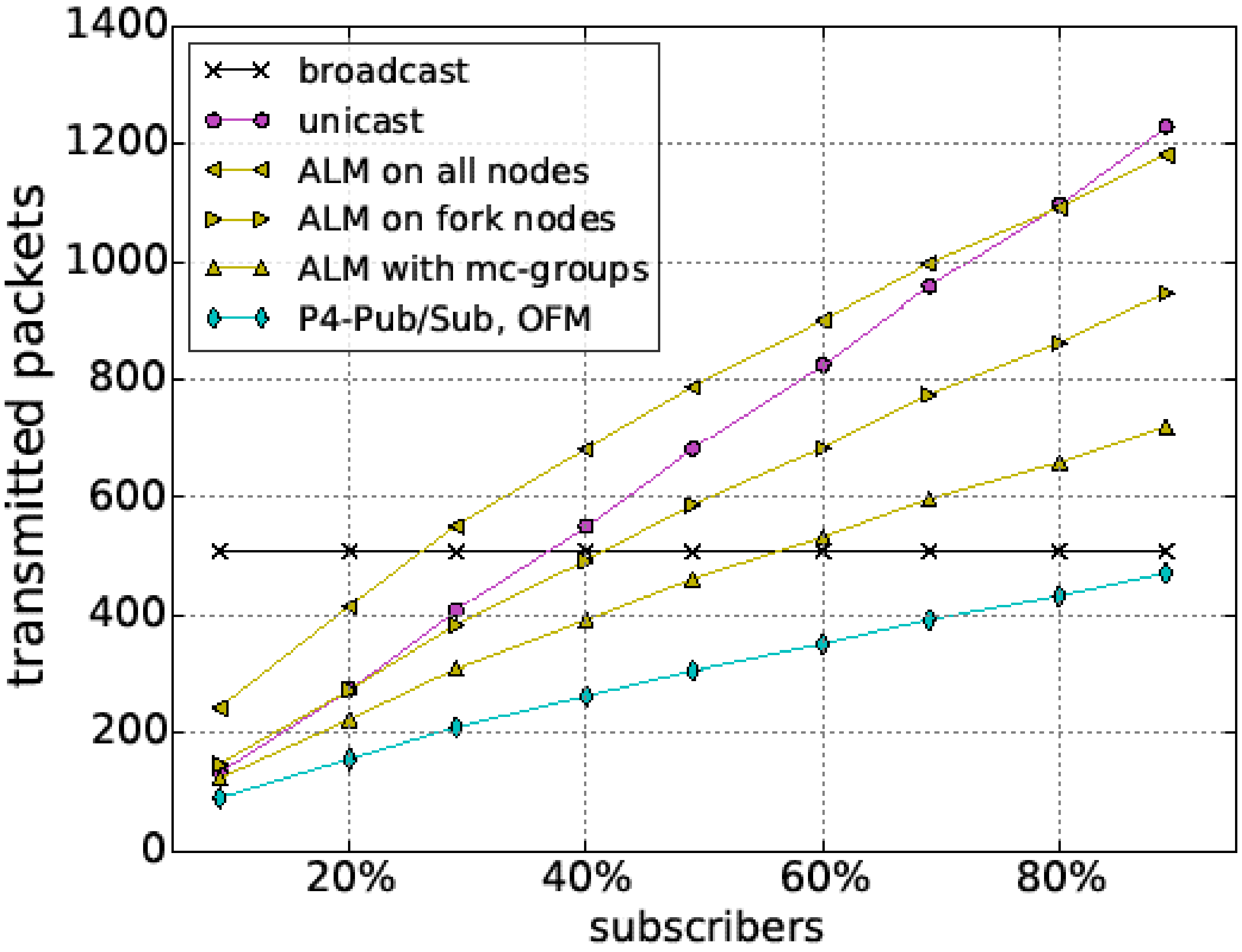


- Comparison against
  - Broadcast / unicast
  - Application layer multicast
  - Network layer mutlicast (OpenFlow multicast)
- ... by measuring

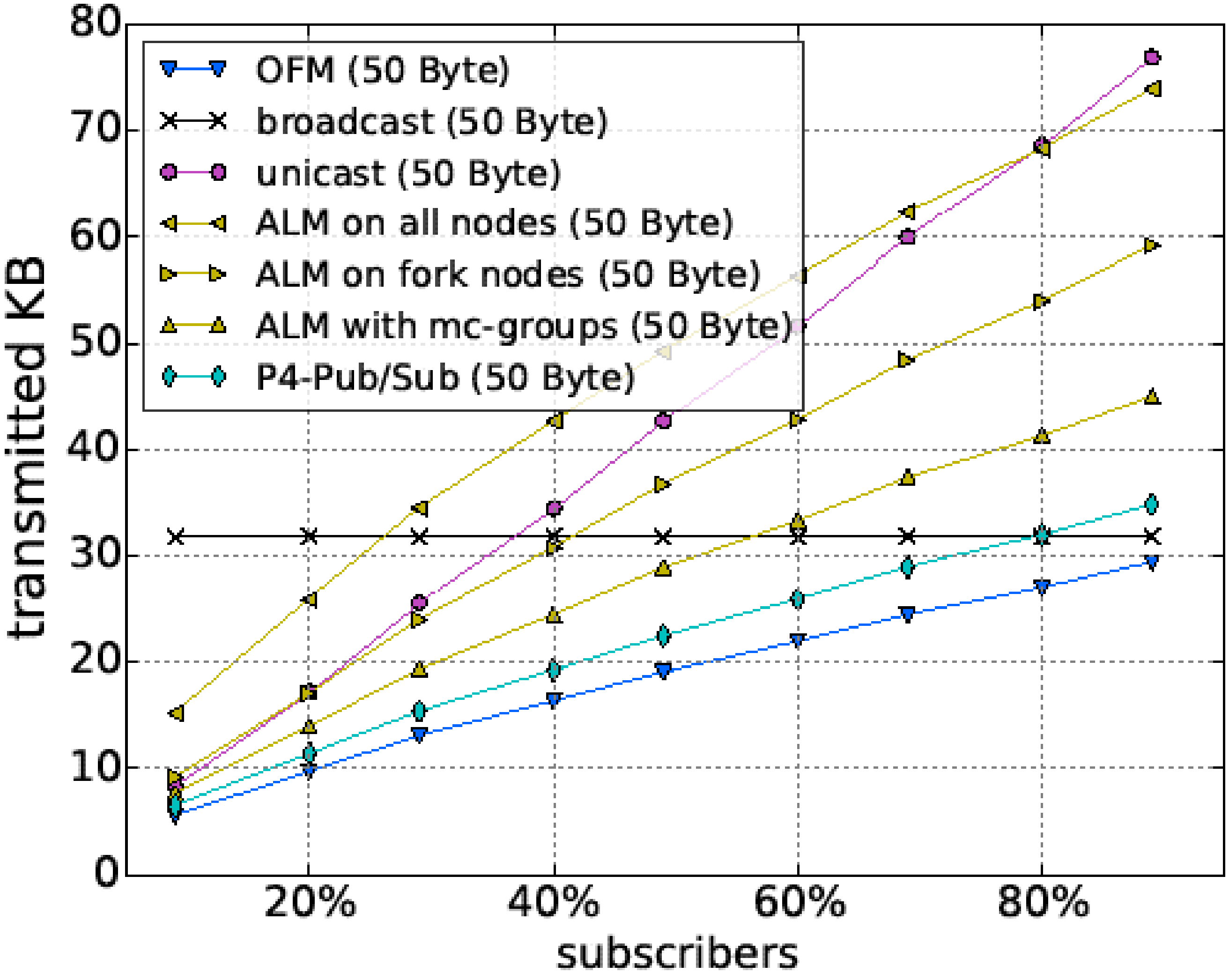
  (a) initial header size, (b) table lookups, (c), transmitted packets,

  (d) transmitted bytes, (e) worst-case delay
- ... under variation of
  - subscribers with matching subscriptions (10% - 90%)
  - notification payload (50 Byte, 500 Byte)

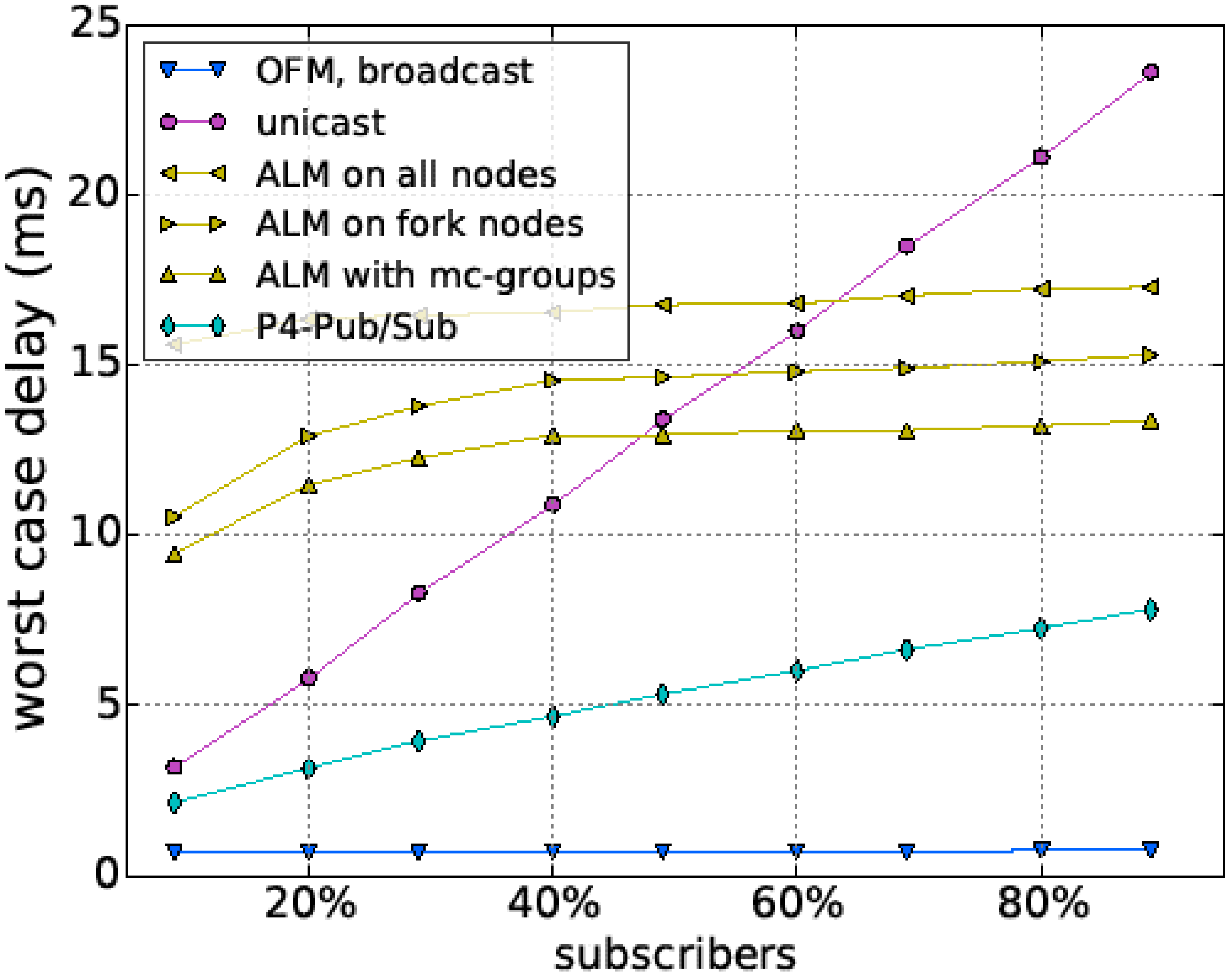
P4-Pub/Sub vs. competing Approaches



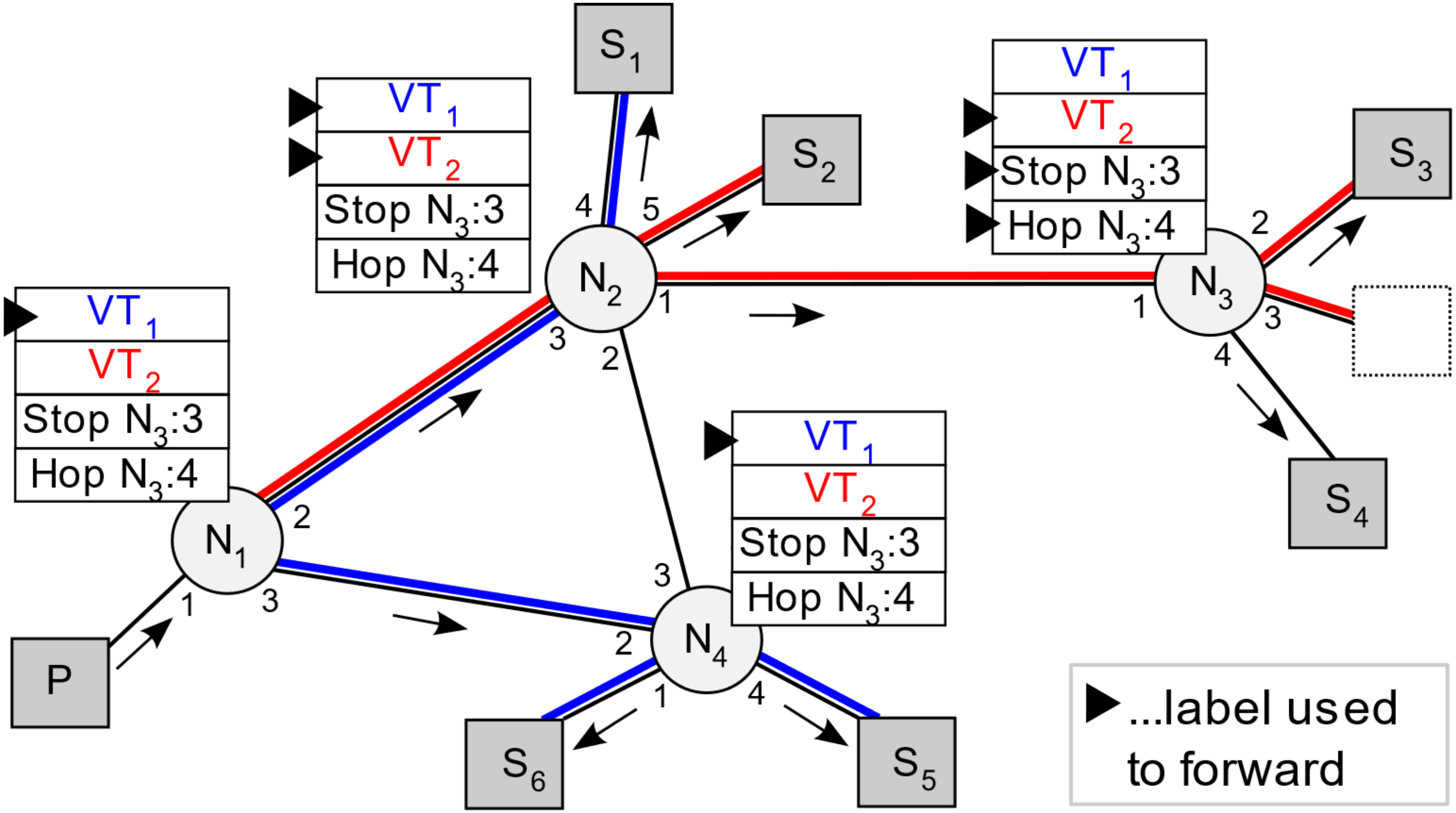
P4-Pub/Sub vs. competing Approaches



P4-Pub/Sub vs. competing Approaches

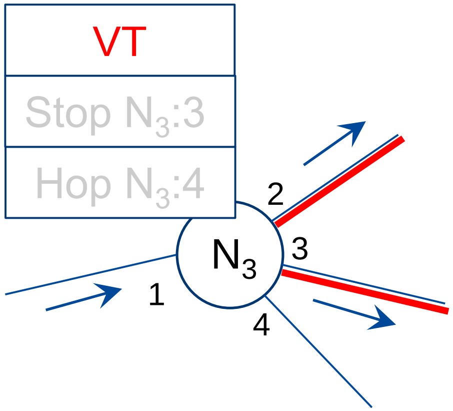


Referring Forwarding Trees



Virtual Trees

1. Virtual Tree

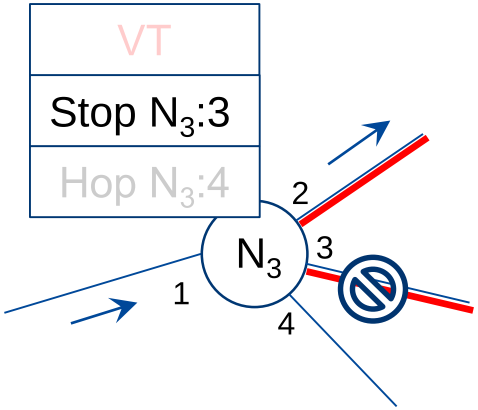


VT

Stop N_3:3

Hop N_3:4

2. Stop Entry

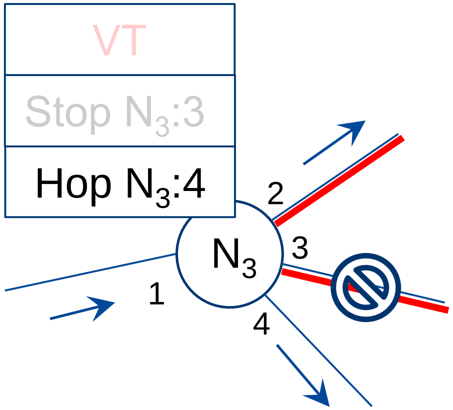


VT

Stop N_3:3

Hop N_3:4

3. Hop Entry



VT

Stop N_3:3

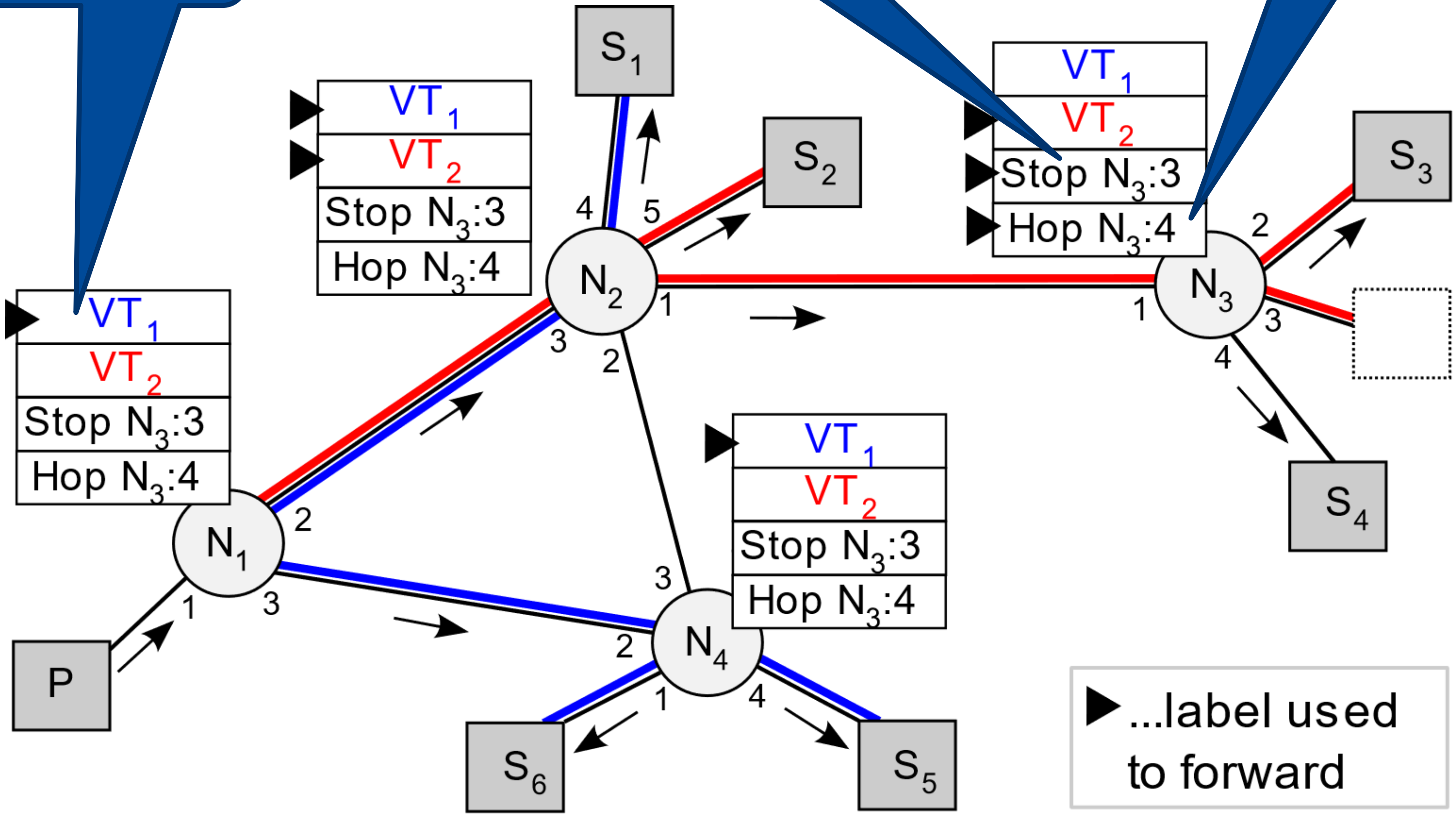
Hop N_3:4

Stitching Forwarding Trees

Stack describes whole distribution tree by using 2 VTs

Stop entry to cut off edges that leave the distribution tree.

Additional hop to extend the distribution tree.

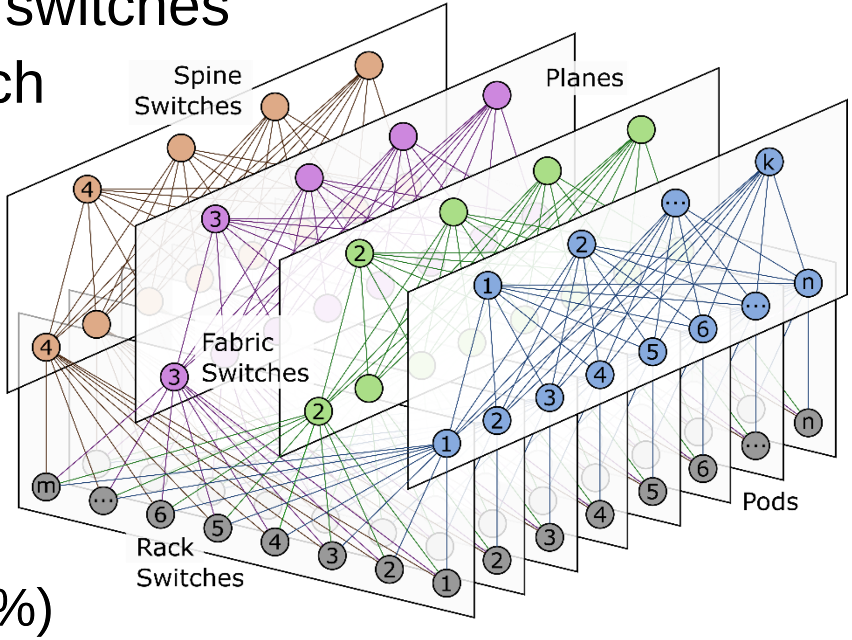


Deriving beneficial Virtual Trees

- Quality of header savings depends on stored virtual trees
- Beneficial virtual tree creation requires knowledge about pub/sub network
- Different levels of pub/sub system knowledge possible
  1. topological properties of the physical network
  2. publisher/subscriber relationships
  3. notification statistics

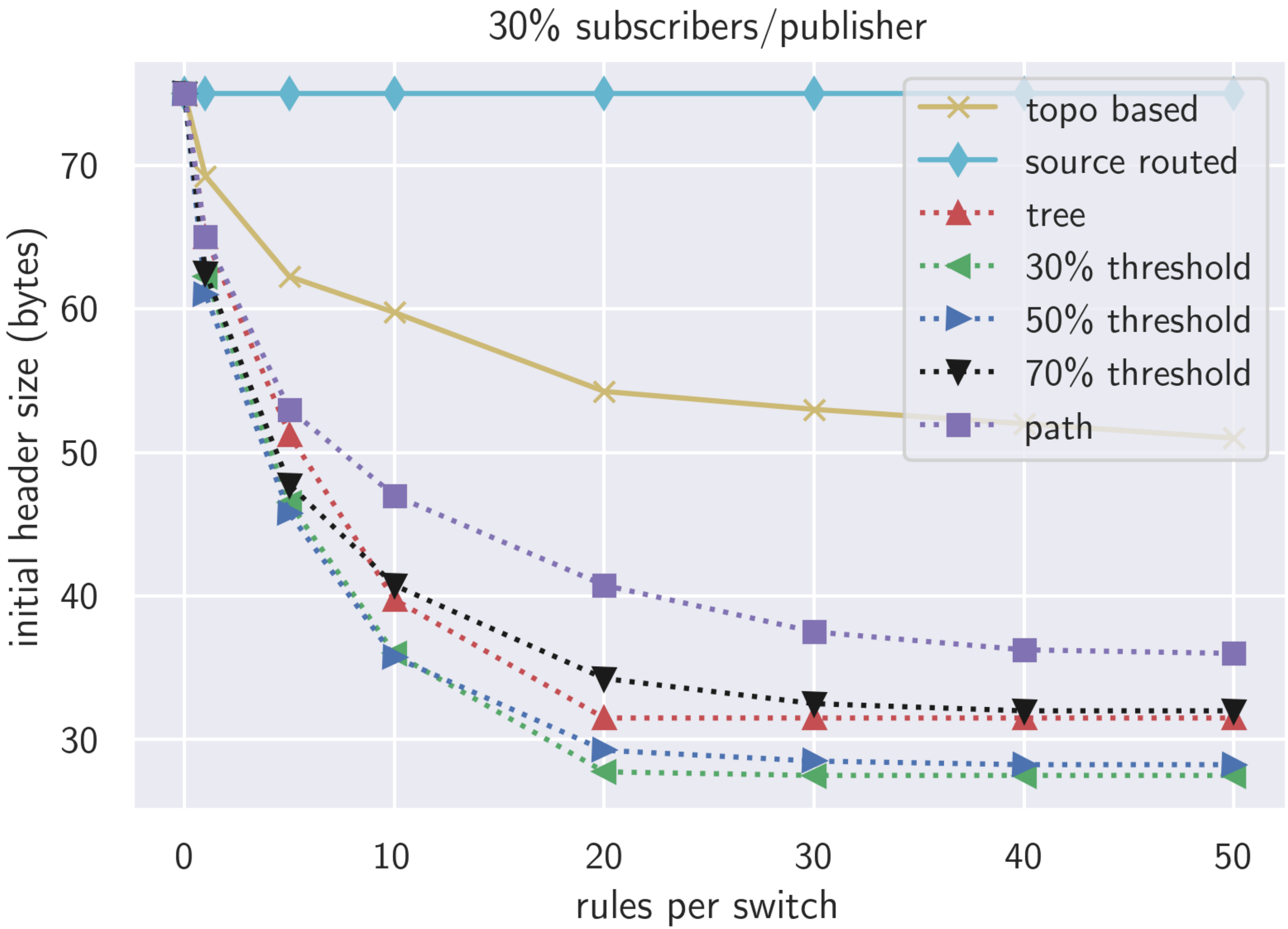
Evaluation of Multicast by Virtual Trees

- Facebook fabric topology with 80 switches
- Mininet with the P4 software switch behavioral model version 2

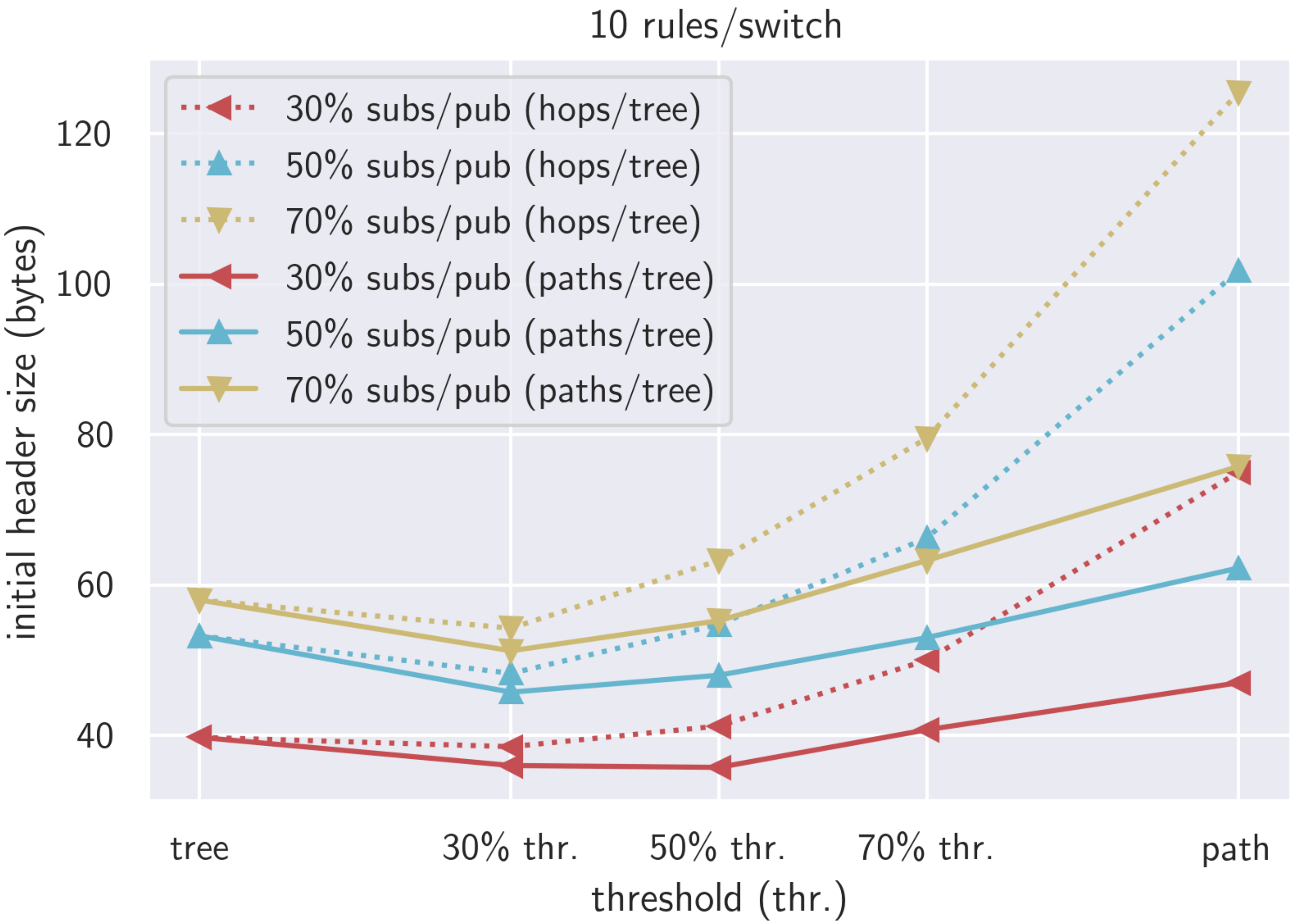


- Evaluation by measuring
  - initial header size
  - network load
- ... under variation of
  - number of subscribers (30% - 70%)
  - max. number of flow rules per switch
  - virtual tree installation strategies
- … by setting following fixed values
  - random filter matching probability for each subscriber
  - notification payload (50 Byte)

Evaluation – Publisher‘s Header Size



Evaluation – Hops vs. individual Paths



Conclusion

- Notification distribution strategies for content-based pub/sub
- All forwarding decisions are made on network layer
- Full control over a notification‘s distribution tree
- Flexibility to define custom pub/sub protocols
- Not bound to protocols of the IP suite
- Multicast by source routing
  - 1) Output-ports, 2) Bitmasks, 3) Multicast groups
- Multicast by referring stored distribution trees
  - i) Virtual Trees, ii) Hops, iii) Stops
- Evaluation against unicast, broadcast, ALM and OFM
- Outlook
  - study different real-world networks
  - develop different strategies for deriving forwarding rules

Thank you for your kind attention!

M.Sc. Christian Wernecke

christian.wernecke@uni-rostock.de

https://www.ava.uni-rostock.de