Scalable Enterprise Networks with Inexpensive Switches

Minlan Yu
minlanyu@cs.princeton.edu
Princeton University

Joint work with Alex Fabrikant, Mike Freedman, Jennifer Rexford and Jia Wang
Enterprises are Hard to Manage

• Large number of hosts
  – Tens or even hundreds of thousands of hosts
• Dynamic hosts
  – Host mobility, virtual machine migration
• Complex management tasks
  – Access control: Block malicious traffic, rate-limiting
  – Resource allocation, path selection for applications
  – Measurement: monitoring for troubleshooting
New Management Techniques

• Flat addresses
  – Easy for self configuration
  – No change when host moves

• Shortest path routing
  – Disseminate end host information (no flooding)
  – Efficient routing (no spanning tree)

• Flow-based switches
  – Flow rules for access control, measurement, etc.
  – Support flexible policies

There are still scalability problems.
Scalability Challenges

- From enterprise networks
  - Network keeps growing
    - Lots of state in switches
  - Large number of policies
- From new management proposals
  - Flat addresses
    - Cannot aggregate addresses
  - Flow switches
    - Lots of multi-dimensional rules
- Memory constraints for packet processing

How to scale enterprises with many hosts, switches, and flow rules while reducing the cost and power consumption of the switch memory?

<table>
<thead>
<tr>
<th></th>
<th>Lookup</th>
<th>Speed</th>
<th>$$/MB</th>
<th>Power</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCAM</td>
<td>Wildcard rules</td>
<td>Fast</td>
<td>expensive</td>
<td>more</td>
<td>Flow rules</td>
</tr>
<tr>
<td>SRAM</td>
<td>Exact match</td>
<td>Slow</td>
<td>cheap</td>
<td>less</td>
<td>Forwarding</td>
</tr>
</tbody>
</table>
Scalability via Indirection

• Minimize state required at each switch
  – Small memory doesn’t always have complete or precise information
  – Redirect to other switches

• Traffic indirection in the data plane
  – Without sacrificing flexibility in the control plane
## Improving Scalability in Data Plane

<table>
<thead>
<tr>
<th></th>
<th><strong>BUFFALO</strong></th>
<th><strong>DIFANE</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scalability problem</strong></td>
<td>Packet forwarding in SRAM</td>
<td>Access control in TCAM</td>
</tr>
<tr>
<td><strong>Indirection</strong></td>
<td>Occasionally send traffic to random ports</td>
<td>Redirect some packets to other switches</td>
</tr>
<tr>
<td><strong>Algorithmic contribution</strong></td>
<td>Hashing instead of caching</td>
<td>Partition wildcard rules</td>
</tr>
<tr>
<td><strong>Switch modification</strong></td>
<td>Only require hash support</td>
<td>Software modification in some switches</td>
</tr>
</tbody>
</table>
BUFFALO: Bloom Filter Forwarding Architecture for Large Organizations
Scaling Packet Forwarding

• **Enterprise networks**
  – Flat addresses (e.g., MAC addresses)
  – Shortest path routing (link state, distance vector)

• **Scalability challenges of packet forwarding**
  – Forwarding table growth (in # of hosts and switches)
  – Increasing link speed
State of the Art

- Hash table in SRAM to store forwarding table
  - Map MAC addresses to next hop
  - Hash collisions:

  00:11:22:33:44:55
  00:11:22:33:44:66
  ... ...
  aa:11:22:33:44:77

- Overprovision to avoid running out of memory
  - Perform poorly when out of memory
  - Difficult and expensive to upgrade memory
Bloom Filters

- Bloom filters in fast memory (SRAM)
  - A compact data structure for a set of elements
  - Calculate s hash functions to store element x
  - Easy to check membership
  - Reduce memory at the expense of false positives
BUFFALO: Bloom Filter Forwarding

- One Bloom filter (BF) per next hop
  - Store all addresses forwarded to that next hop

![Diagram showing Bloom Filters for different nexthops]

Packet destination -> query

Bloom Filters

- Nexthop 1
- Nexthop 2
- ......
- Nexthop T

hit
Comparing with Hash Table

• Save 65% memory with 0.1% false positives

• More benefits over hash table
  – Performance degrades gracefully as tables grow
  – Handle worst-case workloads well
False Positive Detection

- Multiple matches in the Bloom filters
  - One of the matches is correct
  - The others are caused by false positives
Handle False Positives

• Design goals
  – Should not modify the packet
  – Never go to slow memory
  – Ensure timely packet delivery

• When a packet has multiple matches
  – Exclude incoming interface
    • Avoid loops in “one false positive” case
  – Random selection from matching next hops
    • Guarantee reachability with multiple false positives
One False Positive

• Most common case: one false positive
  – When there are multiple matching next hops
  – Avoid sending to incoming interface

• Provably at most a two-hop loop
  – Stretch $\leq \text{Latency}(AB) + \text{Latency}(BA)$
Multiple False Positives

• Handle multiple false positives
  – Random selection from matching next hops
  – Random walk on shortest path tree plus a few false positive links
  – To eventually find out a way to the destination
Stretch Bound

• Provable expected stretch bound
  – With k false positives, proved to be at most $O(3^{k/3})$
  – Proved by random walk theories
• However, stretch bound is actually not bad
  – False positives are independent
  – Probability of $k$ false positives drops exponentially
• Tighter bounds in special topologies
  – For tree, expected stretch is $2(k - 1)^2$
Stretch Evaluation

- **Evaluation method**
  - A campus network of 1500 switches
  - Given false positive rate of Bloom Filters
  - Measure the stretch each packet experience

<table>
<thead>
<tr>
<th>False positive rate</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Shortest path length</td>
</tr>
<tr>
<td>0.001%</td>
<td>99.9% packets: 1 (no stretch)</td>
</tr>
<tr>
<td></td>
<td>0.0003% packets: 2</td>
</tr>
<tr>
<td>0.5%</td>
<td>70% packets: 1</td>
</tr>
<tr>
<td></td>
<td>0.0002% packets: 7</td>
</tr>
</tbody>
</table>
Practical Issues

• Minimize false positive
  – Input: Forwarding table (FIB), # of hash functions
  – Output: BF size for each next hop
  – E.g., FIB 200K entries, 10 next hops, 8 hash func
  – 600 KB SRAM with 0.01% false positive rate

• Quickly adapt to routing changes
  – Use counting Bloom filter in the slow memory
  – Re-optimize BF sizes under significant routing changes
  – 10.7 µs to update a route
BUFFALO Switch Architecture

Control Plane (Routing Protocols)

FIB updates

Slow memory:
Counting
Bloom Filters

Nexthop0
Nexthop1
... ...
NexthopT-1

BF updates

Fast memory:
Bloom Filters

Nexthop0
Nexthop1
... ...
NexthopT-1

Matching next hop set
False Positive Handler

routing updates
packets

routing updates
packets
Prototype Evaluation

• **Environment**
  – Prototype implemented in kernel-level Click
  – 3.0 GHz 64-bit Intel Xeon
  – 2 MB L2 data cache, used as fast memory size $M$

• **Forwarding table**
  – 10 next hops, 200K entries

• **Peak forwarding rate**
  – 365 Kpps, 1.9 µs per packet
  – 10% faster than hash-based *EtherSwitch*
BUFFALO Conclusion

• Indirection for scalability
  – Send false-positive packets to random port
  – Gracefully increase stretch with the growth of forwarding table

• Bloom filter forwarding architecture
  – Small, bounded memory requirement
  – One Bloom filter per next hop
  – Optimizing of Bloom filter sizes
  – Dynamic updates using counting Bloom filters
Scalable Flow-Based Networking with DIFANE
Enterprise Network Management

• Network Management Policies
  – Access control, customized routing, measurement

• Support flexible policies *inside* the network
  – No control on the hosts

• High-level policies managed by the controller
  – E.g., deny the engineering group access to the human-resource database
Flow-based Networking

- Low-level rules stored in the TCAM of switches
  - Substituting high-level names with network addr.
  - E.g., (source =10.1.2.0/24, destination =10.1.4.0/24, port = 288) \( \rightarrow \) Drop
  - Wildcard, more complex than forwarding table
Pre-install Rules in Switches

- Problems:
  - No host mobility support
  - Switches do not have enough memory
Install Rules on Demand (Ethane)

Buffer and send packet header to the controller

First packet misses the rules

Install rules

Forward

Controller

- Problems:
  - Delay of going through the controller
  - Switch complexity
  - Misbehaving hosts
## DIFANE: Combining Proactive & Reactive

<table>
<thead>
<tr>
<th>Features</th>
<th>Install rules</th>
<th>Proactive</th>
<th>Reactive (Ethane)</th>
<th>DIFANE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host mobility</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Memory usage</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Keep packet in data plane</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
DIFANE Design Decisions

• Reducing overhead of cache misses
  – Handle “miss” packets efficiently
    → Keep them in the data plane (TCAM)
  – Reduce the *number* of “miss” packets
    → Caching wildcard rules

• Scaling to large networks and many rules
  – Partition and distribute the flow rules
  – Link-state protocol between the switches
DIFANE architecture (two stages)

Distributed Flow Architecture for Networked Enterprises
Stage 1

The controller *proactively* generates the rules and distributes them to authority switches.
Partition and distribute the flow rules

Just a group of coarse-grained rules
X:0-1 Y:0-3 \(\rightarrow\) A
X:2-5 Y:0-1 \(\rightarrow\) B
X:2-5 Y:2-3 \(\rightarrow\) C
Stage 2

The authority switches keep packets always in the data plane and *reactively* cache rules.
Packet redirection and rule caching

A slightly longer path in the data plane is faster than going through the control plane
Packet redirection and rule caching

- **First packet**
  - Ingress Switch
  - Redirect
  - Cache Rules
  - Forward

- **Following packets**
  - Ingress Switch
  - Partition Rules
  - Authority Switch
  - Auth. Rules
  - Feedback: Cache Rules
  - Egress Switch
  - Hit cached rules and forward

35
Three sets of rules in TCAM

<table>
<thead>
<tr>
<th>Type</th>
<th>Priority</th>
<th>F1</th>
<th>F2</th>
<th>Action</th>
<th>Timeout</th>
<th>No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache rules</td>
<td>210</td>
<td>00**</td>
<td>111*</td>
<td>Encap, forward to B</td>
<td>10 sec</td>
<td>R</td>
</tr>
<tr>
<td>Authority rules</td>
<td>110</td>
<td>00**</td>
<td>001*</td>
<td>Encap, forward to D, trigger ctrl. plane func.</td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Partition rules</td>
<td>15</td>
<td>0***</td>
<td>000*</td>
<td>Encap, redirect to B</td>
<td>∞</td>
<td>Prim</td>
</tr>
</tbody>
</table>

- **Cache rules are installed by authority switches**
- **Authority rules only exist in authority switches**
- **Partition rules are coarse-grained rules**
DIFANE Switch Prototype
built with OpenFlow switch

Just software modification for authority switches.
Caching Wildcard Rules

• Overlapping wildcard rules
  – Cannot simply cache matching rules

• Multiple authority switches
  – Cache rules in one ingress switch simultaneously
Partition Wildcard Rules

• Partition rules
  – Minimize the TCAM entries in switches
  – Decision-tree based rule partition algorithm

Cut B is better than Cut A
## Handling Network Dynamics

<table>
<thead>
<tr>
<th>Network dynamics</th>
<th>Cache rules</th>
<th>Authority Rules</th>
<th>Partition Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule changes at the controller</td>
<td>Timeout</td>
<td>Change</td>
<td>No change (most of the time)</td>
</tr>
<tr>
<td>Topology changes at the switches</td>
<td>No change</td>
<td>No change</td>
<td>Change</td>
</tr>
<tr>
<td>Host mobility</td>
<td>Timeout</td>
<td>No change</td>
<td>No change</td>
</tr>
</tbody>
</table>
Prototype Evaluation

• Evaluation setup
  – Kernel-level Click-based OpenFlow switch
  – Generate flows each with one 64 Byte packet
  – Traffic generators, switches, controller run on separate 3.0GHz 64-bit Intel Xeon machines

• Compare delay and throughput with NOX
Comparing NOX and DIFANE

NOX

Controller

Switch 1

Switch n

DIFANE

Authority switch

Switch 1

Switch n
Delay Evaluation

• Average delay (RTT) of the first packet
  – NOX: 10 ms
  – DIFANE: 0.4 ms

• Reasons for performance improvement
  – DIFANE always keeps packets in the kernel
  – DIFANE can be easily implemented in hardware to further improve performance
  – Rule caching does not block traffic in DIFANE
Peak Throughput

Local controller at the switch is the bottleneck

NOX is the bottleneck

DIFANE further increases the throughput linearly with the number of authority switches.
DIFANE Conclusion

• Indirection for scalability
  – Redirect miss packets to authority switches
  – Always keep packets in the data plane

• A distributed flow architecture for enterprises
  – React fast to rule, topology changes, host mobility
  – Caching and partition wildcard rules
  – Small modification in the control plane of authority switches
Conclusion

- Scaling with lots of hosts, switches and rules
  - Indirection to reduce memory usage
  - BUFFALO: Scaling packet forwarding (CoNEXT’09)
  - DIFANE: Scaling flow management (in submission)

- Combining BUFFALO and DIFANE
  - BUFFALO: Destination-based forwarding in SRAM
  - DIFANE: ACLs, monitoring in TCAM
Thanks!

- Questions?