Floodless in SEATTLE:
A Scalable Ethernet Architecture for Large Enterprises

Changhoon Kim, Matthew Caesar,
and Jenifer Rexford
Presented by Chi-Yao Hong
Adapted from slides by Changhoon Kim
Oct. 1, 2009

In ACM SIGCOMM, Seattle, WA, Aug. 2008
Challenges of Layer-2 networks

• Facing unprecedented scale

• Highly requirements in terms of efficiency and availability

• Large data centers might comprise 100,000+ computers within a single facility

• What’s the possible solution?
Ethernet has substantial benefits

- Because of its simplicity!
  - Plug-and-play via flat addressing
  - Simplifies the handling of topology changes and host mobility
- IP networks require massive effort to configure and manage
  - Why do we still use IP routing inside a single network?
What’s Wrong With Ethernet?

- Ethernet is not scalable!
  - Network-wide flooding
  - Frequent broadcasting
  - Unbalanced link utilization due to tree-based forwarding
- Scalability requirement is growing very fast
Current Practice

- Multiple small Ethernet-based IP subnets interconnected by routers

  - Loss of self-configuring capability
  - Complexity in implementing policies
  - Limited mobility support
  - Inflexible route selection

Scalable but...

- Scalable but sacrifices Ethernet's simplicity
- Loses IP's efficiency
# SEATTLE: The best of IP and Ethernet

<table>
<thead>
<tr>
<th>Feature</th>
<th>Ethernet Bridging</th>
<th>IP Routing</th>
<th>SEATTLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ease of configuration</td>
<td>🟢</td>
<td>🟥</td>
<td>🟢</td>
</tr>
<tr>
<td>Optimality in addressing</td>
<td>🟢</td>
<td>🟥</td>
<td>🟢</td>
</tr>
<tr>
<td>Host mobility</td>
<td>🟢</td>
<td>🟥</td>
<td>🟢</td>
</tr>
<tr>
<td>Scalability</td>
<td>🟥</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Path efficiency</td>
<td>🟥</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Tolerance to loop</td>
<td>🟥</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Objective</td>
<td>Approach</td>
<td>Solution</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------</td>
<td>-----------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Avoiding flooding</td>
<td>Never broadcast unicast traffic</td>
<td>Network-layer one-hop DHT</td>
<td></td>
</tr>
<tr>
<td>Restraining broadcasting</td>
<td>Bootstrap hosts via unicast</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reducing routing state</td>
<td>Populate host info only when and where it is needed</td>
<td>Traffic-driven resolution with caching</td>
<td></td>
</tr>
<tr>
<td>Shortest-path forwarding</td>
<td>Allow switches to learn topology</td>
<td>L2 link-state routing (switch-level topology)</td>
<td></td>
</tr>
</tbody>
</table>
Network-layer One-hop DHT

• Switches maintain <key, value> pairs by commonly using a hash function $F$
  – $F$: Consistent hash mapping a key to a switch
  – Link-state routing ensures each switch knows about all the other live switches, thus enabling one-hop DHT operations

• Benefits
  – Reducing lookup complexity
  – Fast and efficient reaction to changes
Network-layer One-hop DHT

Host discovery or registration (e.g., Gratuitous ARP request)

Traffic to \(x\) Hash \(F(x) = B\)

Tunnel to egress node, \(A\)

Optimized forwarding directly from \(D\) to \(A\)

Tunnel to relay switch, \(B\)

Notifying \(<x, A>\) to \(D\)

Store \(<x, A>\) at \(B\)

Entire enterprise (A large single IP subnet)

Switches

End-hosts

Control flow

Data flow
Handling host dynamics

- Events **NOT modifying the set of live switches**
  - E.g., most link failure/recovery
  - LS routing
- Events **modifying the set of live switches**
  - E.g., switch failure/recovery
  - $F$ works differently after a change
  - Two simple operations
    - If $F_{\text{new}}(k) \neq F_{\text{old}}(k)$, owner re-publishes to $F_{\text{new}}(k)$
    - Remove any $<k, v>$ published by non-existing owners
Handling host dynamics

when shortest-path forwarding is used

Old Dst

< x, G >

Src

y

A

B

Relay (for x)

G

New Dst

< x, G >

D

< x, A >

< x, G >
Handling ARP requests

- **Host discovery**: Owner of \((IP_x, mac_x)\)
- **Hashing**: \(F(IP_x) = B\)
- **Unicast ARP request to** B
- **Unicast ARP reply** <\(IP_x, mac_x, A\)> to ingress
- **Switches**
- **End-hosts**
- **Control flow**
  - Control flow (red dotted line)
  - ARP msgs (blue line)
Enhancements

• **Goal:** Dealing with switch-level heterogeneity
  • **Sol:** Virtual switches

• **Goal:** Attaining very high availability of resolution
  • **Sol:** Replication via multiple hash functions

• **Goal:** Dividing administrative control to sub-units
  • **Sol:** Multi-level one-hop DHT
Performance Evaluation

• **Packet-level Simulation**
  – Synthetic traffic based on real traces from LBNL
    • Inflated the trace while preserving original properties
  – Real topologies from campus, data centers, and ISPs

• **Emulation with prototype switches**
  – Click/XORP implementation
Amount of Routing State

![Graph showing the amount of routing state with and without caching in SEATTLE, with a factor of 20 difference.]
Control Overhead

Ethernet

a factor of 1000

SEATTLE w/ caching

SEATTLE w/o caching
Latency Stretch

![Graph showing latency stretch for ROFL and SEATTLE with varying maximum cache size per switch. The graph displays a downward trend as the cache size increases.]
Conclusion

• SEATTLE is a plug-and-playable network architecture ensuring both scalability and efficiency
• One-hop DHT coupled with LS routing
• Reactive location resolution and caching
• Shortest-path forwarding
• Discussion
  – Higher delay stretch
  – Limited switch-level scale
  – Failure