Scaling Up Clustered Network Appliances with ScaleBricks

Dong Zhou, Bin Fan, Hyeontaek Lim, David Andersen
Michael Kaminsky†, Michael Mitzenmacher**
Ren Wang†, Ajaypal Singh‡

Carnegie Mellon University, †Intel Labs
**Harvard University, ‡Connectem, Inc.

http://www.istc-cc.cmu.edu/
Scaling Up Clustered Network Appliances

Throughput scaling
• Higher bandwidth & more ports

FIB scaling
• More endpoints & flows

Updating scaling
• Higher update rates

Previous cluster architectures (e.g., RouteBricks)

ScaleBricks

Focus of this talk
Motivation: Network Appliance in LTE

Base station

LTE-to-Internet gateway

Cluster interconnect

Internet
Motivation: Network Appliance in LTE

Base station

Upstream

LTE-to-Internet gateway

Handling node for X

Cluster interconnect

Internet

Possible downstream

Ingress node

Traffic is forwarded to handling node

Externally imposed requirements!
Fully Duplicated FIB

FIB (forwarding table)

<table>
<thead>
<tr>
<th>Label</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>X→B3</td>
<td>X→B3</td>
</tr>
<tr>
<td>Y→C1</td>
<td>Y→C1</td>
</tr>
<tr>
<td>Z→B2</td>
<td>Z→B2</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

Node D

Node B

Node C: Port 1

Node C: Port 2

We want: FIB scales with more nodes

Full FIB on every node

Cluster Interconnect

Node A

Node C

Node B

Node D

We want: FIB scales with more nodes

+ 1-hop forwarding
- No FIB scaling
Hash Partitioned FIB

Random partition of FIB determined by hashing
E.g., hash(Label X) = Node C

Node C may have FIB entries whose handling node is not Node C

We want: FIB scales with more nodes without indirection

+ FIB scaling
- Indirection
FIB Scale-Out on Cluster Architectures

- Does the cluster provide **FIB scaling** with more nodes?
- Does the cluster require **indirection** that adds overhead?

<table>
<thead>
<tr>
<th>Architecture</th>
<th>FIB Scaling</th>
<th>Indirection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Duplication</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Hash Partitioning</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ScaleBricks</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Scaling through 4-32 nodes
10% lower latency & 23% higher throughput

34% lower latency
ScaleBricks

Global Partition Table (GPT)
& Deterministic partition of FIB that matches handling node

+ FIB scaling
+ No indirection

? How to build GPT

Handling node of Node B’s FIB entries is also Node B

Node A

Cluster Interconnect

Node B

Node C

Node D

FIB (forwarding table)

<table>
<thead>
<tr>
<th>Label</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GPT</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Y</td>
<td>GPT</td>
</tr>
<tr>
<td></td>
<td>Node C: Port 1</td>
</tr>
<tr>
<td>Z</td>
<td>Y→C1</td>
</tr>
<tr>
<td></td>
<td>Node B: Port 2</td>
</tr>
<tr>
<td></td>
<td>X→B3</td>
</tr>
<tr>
<td></td>
<td>Z→B2</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Designing Global Partition Table (GPT)

- GPT should be very small
  - Every node has GPT containing every FIB entry’s handling node info.
- Strawman solution: Hash table

<table>
<thead>
<tr>
<th>Label</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Node B: Port 3</td>
</tr>
<tr>
<td>Y</td>
<td>Node C: Port 1</td>
</tr>
<tr>
<td>Z</td>
<td>Node B: Port 2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Most table space is taken by keys
- E.g., 104-bit keys (5-tuple labels) vs. 2-bit values (4 cluster nodes)

Is there a way to remove keys while avoiding collisions?
Our Solution for GPT: SetSep

- Practical set separation data structure
  - Do not store keys
  - Brute force to avoid “value” collisions (instead of key collisions)

FIB (forwarding table)

<table>
<thead>
<tr>
<th>Label</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Node B: Port 3</td>
</tr>
<tr>
<td>Y</td>
<td>Node C: Port 1</td>
</tr>
<tr>
<td>Z</td>
<td>Node B: Port 2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

SetSep-based GPT

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>X,Y,...</td>
<td>B,C</td>
</tr>
<tr>
<td>Z,...</td>
<td>B</td>
</tr>
</tbody>
</table>

hash := $H_1$

Value collision → Try next hash function

hash := $H_2$

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>X,...</td>
<td>B</td>
</tr>
</tbody>
</table>

hash := $H_{372}$

No value collision → Use this hash function (“$H_{372}$”) & value array as GPT
No Key Existence Test in SetSep

<table>
<thead>
<tr>
<th>Data structure</th>
<th>Existent key</th>
<th>Nonexistent key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash table</td>
<td>Correct value</td>
<td>“Key not found”</td>
</tr>
<tr>
<td>SetSep</td>
<td>Correct value</td>
<td>Arbitrary value</td>
</tr>
</tbody>
</table>

- Mitigating arbitrary return values
  - Tolerate arbitrary values for nonexistent keys; or
  - Use additional data structures to detect nonexistent keys

- ScaleBricks uses partial FIB to detect nonexistent keys
Making SetSep Fast

- Construction time problem
  - Exponentially increasing # of trials with more entries and wider values
  - 16→32 entries, 1-bit values: Up to $2^{16}$ times slower
  - 16 entries, 1→2-bit values: Up to $2^{16}$ times slower

- SetSep solutions to achieve linear construction time
  - Two-level hashing to divide entries into small, evenly-sized sets
  - Separate hash functions to encode individual value bits

See our SIGCOMM 2015 paper for more details
- Trading space for faster construction by using sparser value array
- Fast generation of many hash functions
- Fast batched lookups with memory prefetching
Main Properties of SetSep

• Compact size
  – $0.5+1.5\log_2(\text{node count})$ bits/entry
  – E.g., 3.5 bits/entry for 4 nodes

• Reasonably fast construction
  – 0.24 million entries/sec (1 thread)

• Fast lookup
  – 520 million lookups/sec (16 threads)
Evaluation Overview

• Full-system forwarding performance
• Scalability analysis

• Setup
  – Modified Connectem’s LTE Evolved Packet Core stack
    • Using Intel DPDK
  – Traffic generated by Spirent SPT-N11U Ethernet testing platform
  – 4x commodity server nodes
    • 2x Intel Xeon E5-2697 v2 (30 MiB L3 cache)
    • 2x Intel 82599ES (dual-port 10 GbE NIC)
  – 10 GbE hardware switch as cluster interconnect
End-to-End Latency with 4 Nodes

GPT avoids indirection → Low latency

Why is ScaleBricks also faster than full duplication? (both avoid indirection)
Per-Node Throughput

ScaleBricks’s partial FIB is smaller than full FIB
→ More FIB entries fit in CPU L3 cache
→ Higher throughput & lower latency

ScaleBricks exhibits graceful throughput decrease
ScaleBricks begins to slow with larger FIB than full duplication does
Scalability Analysis

Aggregate FIB size when each node uses **16 MiB** of memory

- **Hash partitioning** shows very good FIB scaling (at the cost of latency & throughput)
- **ScaleBricks offers FIB scaling through 4-32 nodes**
- **Full duplication** provides no FIB scaling
Conclusion

• ScaleBricks: Scalable cluster architecture for middleboxes
  – Global Partition Table + Partial FIB: FIB scaling without indirection
  – 23% higher throughput, 34% lower latency, FIB scaling through 4-32 nodes

• SetSep: Compact key-value mapping for small value space
  – Skip storing keys, brute force to avoid value collisions
  – Small memory overhead, fast lookup, good construction speed

• Applications
  – Clustered network appliances with flow pinning
  – We are looking for other cool applications of ScaleBricks and SetSep!