Scaling Up Clustered Network Appliances with ScaleBricks

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

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

Presentation by

Christopher Villalpando

Overview

- Intro
- Background
- ScaleBricks
- SetSep
- Implementation
- Evaluation
- Summary

Introduction

Introduction

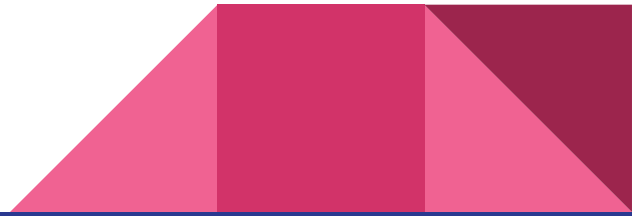
Paper Contributions:

- Present ScaleBricks Architecture.
- Use ScaleBricks to improve performance of a commercial LTE Cellular Network.

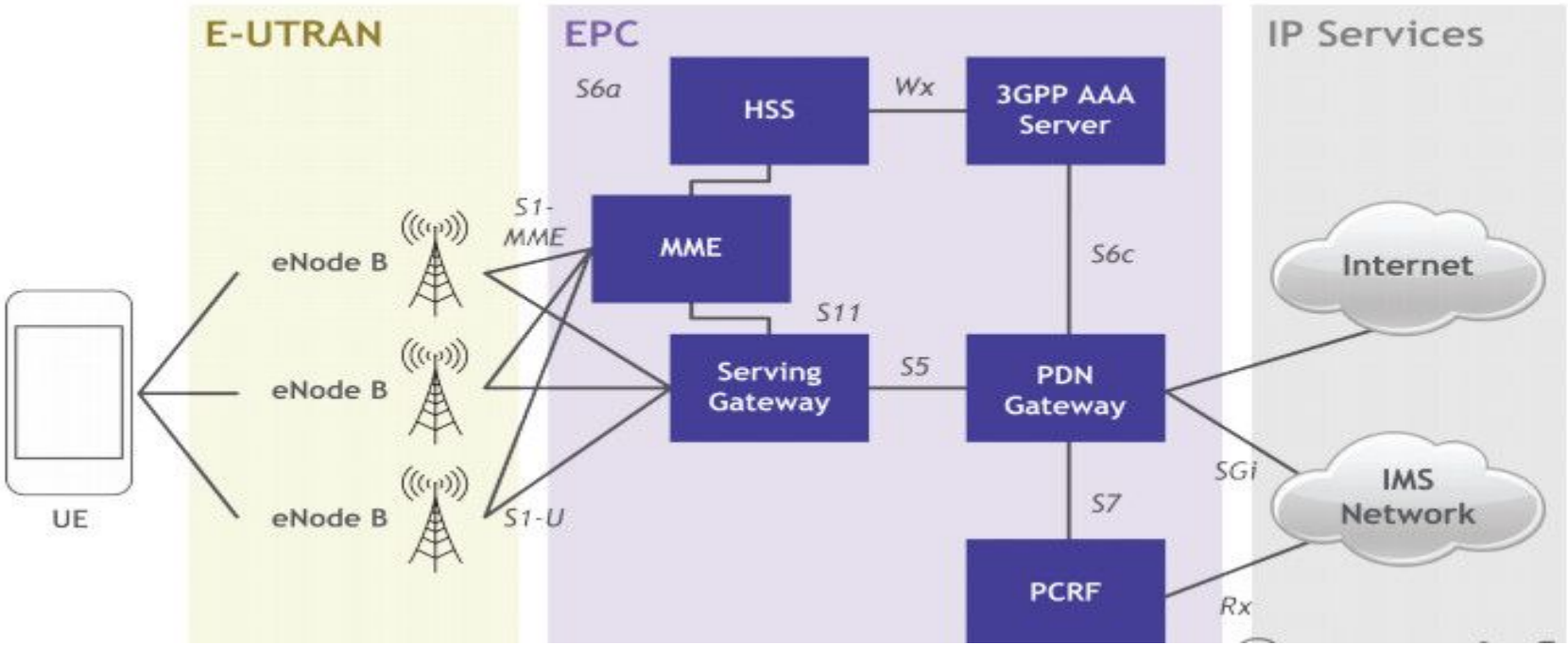
ScaleBricks Goal: To improve performance and scalability of a software based Evolved Packet Core (EPC) stack.

ScaleBricks Focus:

- Throughput scalability
- FIB scalability
- Update scalability



Background - LTE Cellular networks



https://www.linkedin.com/pulse/imsvolte-architecture-introduction-nitin-gupta/

Background - LTE Terms

- **EPC** - Evolved Packet Core
  - **HSS** - Home Subscriber Server - call and session setup, user authentication and access authorization.
  - **MME** - Mobility Management Entity - responsible for most control plane functions in EPC
  - **SGW** - Serving gateway - All uplink/downlink flow through here, responsible for handovers.
  - **PDN** - Public Data Network Gateway - responsible for allocating IP addresses to UEs
  - **PCRF** - Policy and Charging Rule Function - determines what traffic is allowed an how to account for it (billing)

- **eNodeB** - base station for LTE radio
- **UTRAN** - LTE access network
- **UE** - User Equipment

http://www.3gpp.org/technologies/keywords-acronyms/98-lte

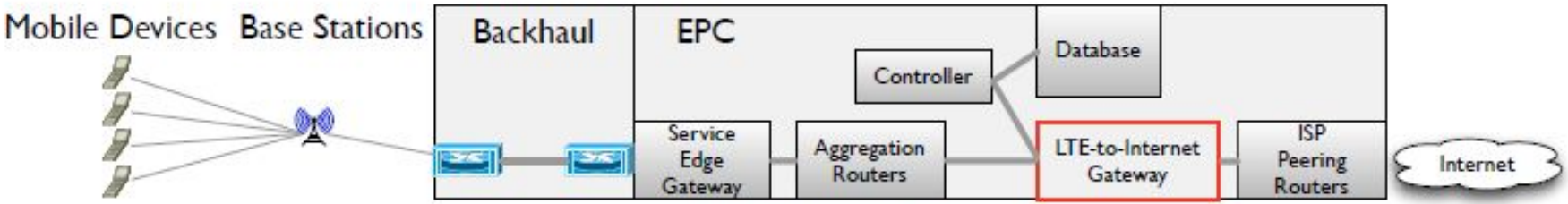
Background - LTE Service

Upstream traffic:

- App running on mobile initiates a connection.
- Controller assigns new connection tunnel and unique End Point ID.
- Upstream traffic sends packets through several middleboxes.
- LTE-to-internet Gateway performs administrative functions.
- Gateway encapsulates packets from tunnel, sends them to ISP peering routers.

Downstream traffic:

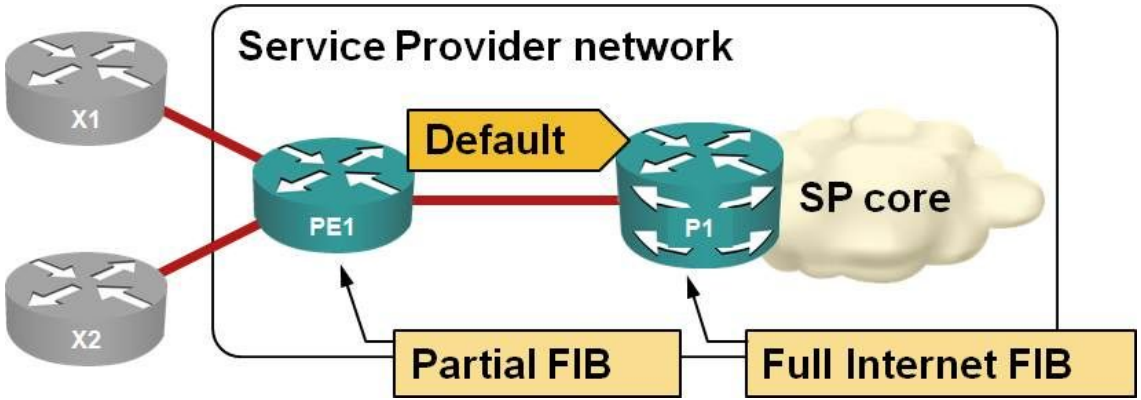
- Downstreams traffic follows reverse path.
- Packets are sent to correct base station, then are send to mobile.



(a) Simplified EPC Architecture

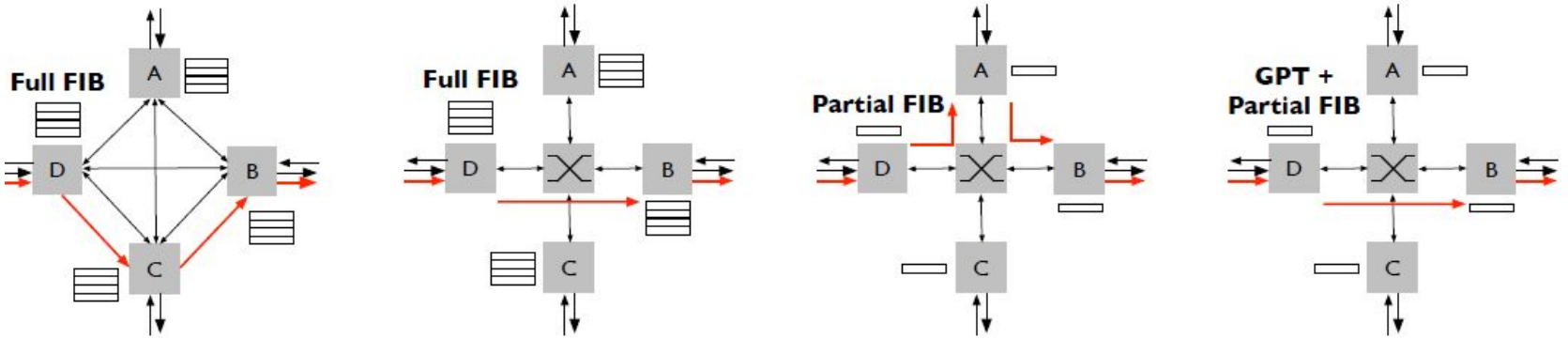
Background - Forwarding Information Base (FIB)

"... also known as a **forwarding table** or **MAC table**, is most commonly used in network bridging, routing, and similar functions to find the proper interface to which the input interface should forward a packet." - wikipedia



https://searchtelecom.techtarget.com/tip/Virtual-aggregation-Lifeline-for-exploding-Internet-routing-tables

Background - FIB Architectures



(a) RouteBricks (b) Full Duplication (c) Hash Partitioning (d) ScaleBricks

Figure 2: Packet forwarding in different FIB architectures

- ***Ingress node:*** node where packets enter the cluster
- ***Handling node:*** node where packet is processed within the cluster
- ***Indirect node:*** intermediate node touched by a packet
- ***Lookup node:*** a node storing forwarding entries associated with a packet

ScaleBricks

ScaleBricks - Overview

ScaleBricks applies a compact lookup structure to route packets directly to the appropriate handling node.

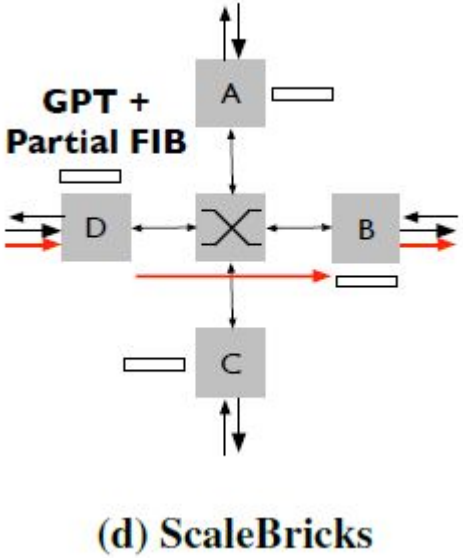
Distributes entire routing information using a hash partitioned design.

Its lookup structure is many times smaller than the alternative approach.

Able to improve throughput and latency while increasing the total number of flows handled by such a cluster.

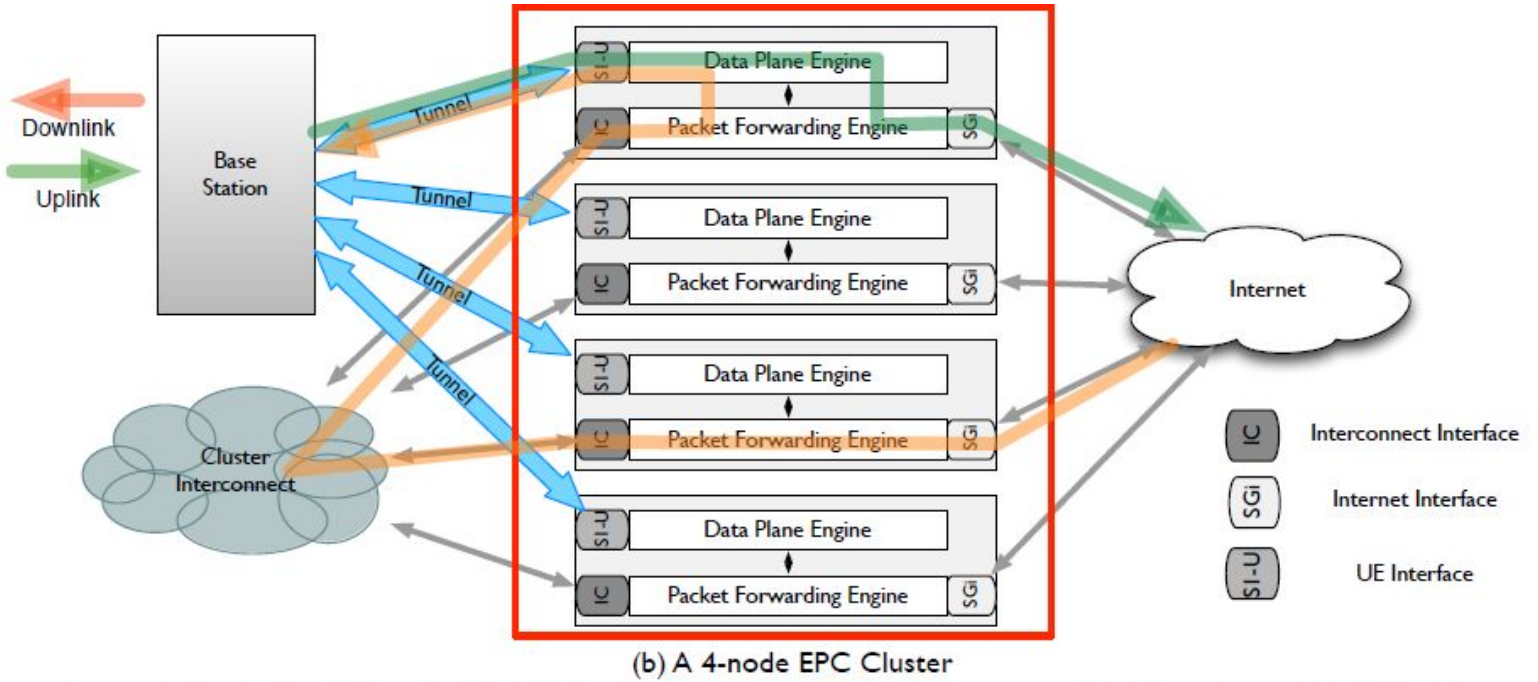
ScaleBricks - Architecture

- ScaleBricks connects servers using a middle switch.
- Topology reduces internal bandwidth requirement and processing latency.
- Scalebricks distributes the entire Routing Information Base (RIB) across the cluster.
    - Global Partitioning Table (GPT)
    - Forward information base (FIB)



(d) ScaleBricks

ScaleBricks - Application



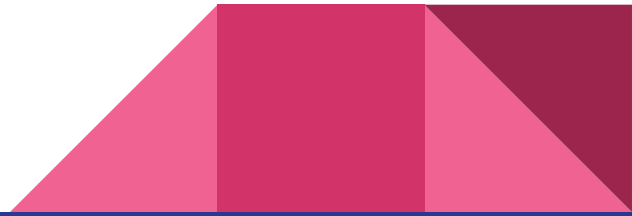
(b) A 4-node EPC Cluster

Figure 1: (a) Simplified Evolved Packet Core (EPC) architecture and (b) a 4-node EPC cluster

ScaleBricks - Components

Global Partitioning Table (GPT):

- Replicated in every ingress node
- This table maps keys to lookup/handling nodes.
- Must be compact for scalability
- Divides sets of keys into smaller subsets of disjoint sets
- Uses data structure ***set separation*** at the core,
    - Very fast lookups, and low memory usage.



ScaleBricks - Components

FIB Table:

- Partitioned tables are distributed among the handling nodes.
- A handling node stores only the FIB entries that point to it.
- Can handle configurable sized values with minima performance impact
- Based upon previous work on space efficient, high performance, hash tables.

ScaleBricks - Components

RIB Updates:

- Sent to the appropriate RIB partitioned node
- Generates GTP and FIB table entries
- GTP entries send to all ingress nodes
- FIB entries sent to appropriate handling node

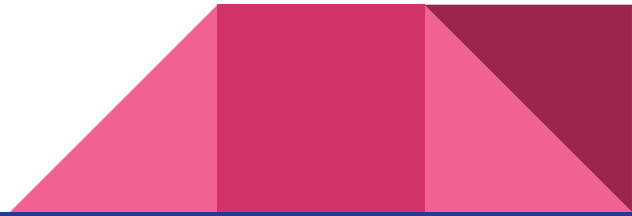
Set Separation

SetSep - Build GPT

The Idea: Use brute force computations to find a function that maps each input key to a value.

The process:

1. Divide input set into small groups.
2. Build high level structure to index groups.
3. For each group, use brute force to find hash function that produces correct outputs for each key.



SetSep

Set Separation resides at the core of the GPT. It leverages 3 properties of ScaleBricks:

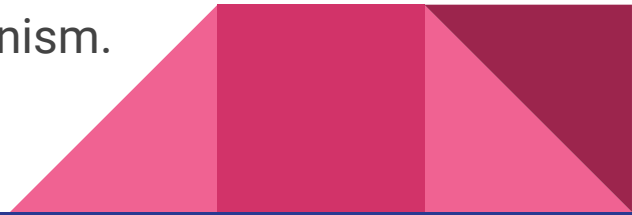
1. GPT returns integers between (0, N-1), where N is the number of servers in the cluster.
2. GPT may return an arbitrary value when queried by unknown destination key.
3. Lookups are frequent but updates less so. Prioritizes lookup performance.

SetSep

Again, first divide n keys into smaller subsets.

Search for SetSep function:

1. Maintain a set of hash functions (function family)
2. Iterate over the hash functions to test unique key-value pairs for every key in group
3. If a key-value fails, reject hash function, try next function.
4. If hash function works for all keys, store index of such function.
5. If no hash function found. Stop, use fall back mechanism.



SetSep

Storing SetSep

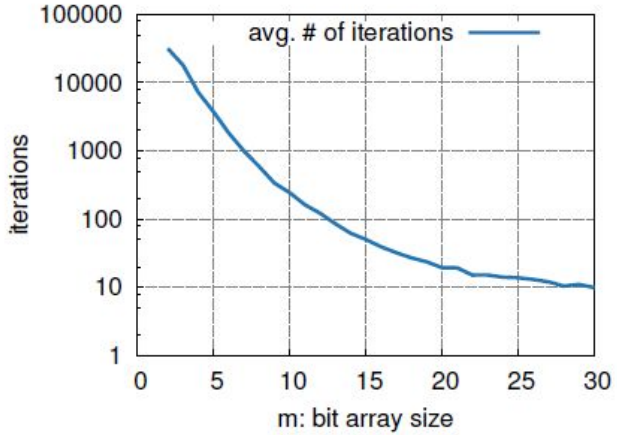
- The hash function index per group is stored in a variable length encoding.
  - The implementation consumes 1.5bits per key

Generating Hash functions:

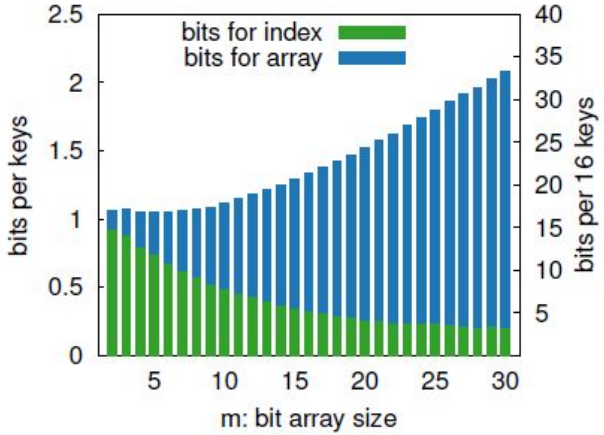
“...we draw inspiration from theoretical results that two hash functions can sufficiently simulate additional hash functions [23]”

$$H_i(x) = G_1(x) + i \cdot G_2(x)$$

SetSep - Space vs Speed



(a) Avg. # of iters. to get one hash func.



(b) Space cost breakdown

Figure 3: Space vs. time, as the function of bit array size m

SetSep

Scaling to billions of Items

Let each group of keys be small → 16 keys

We generate and store hash function index for respective group

1. Mapping must ensure low variance in group size.
2. Mapping should add little space.

Solution: 2 level hashing

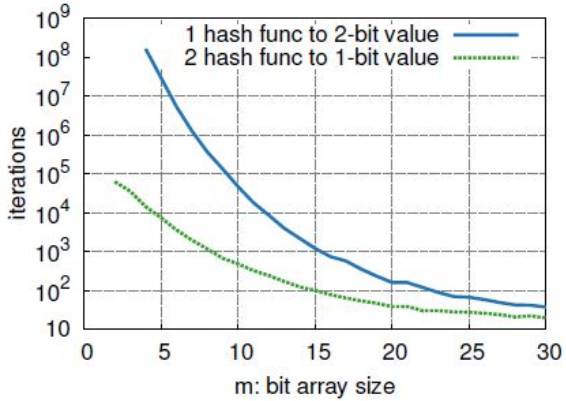


Figure 4: One hash func. vs. multiple hash func.

SetSep

Scalable Updates

“...number of flows the EPC can keep track of.”

Process:

- RIB entries are computed using a hash of the Key
- Keys in the same 1024 block are stored in the same node
- For construction, each node computes its own portion of SetSep
- Results are exchanged with other nodes.
- Similarly, when updating a key *k*, only node responsible for *k* computes value and broadcasts to all other nodes.

Implementation & Evaluation

Implementation

Implemented in C, uses Intel's Data Plane Development Kit (DPDK) for x86 platforms as a fast user-space packet I/O engine.

GPT using SetSep.

- Modification: Efficient use of Mem bandwidth and CPU cycles using memory prefetching to increase L1/L2 cache hits.

Partial FIB using Cuckoo Hashing

- Modification: Hash table initialized at run time(key/index), and additional value array created to allow for variable size values.

Algorithm 1: Batched SetSep lookup with prefetching

```
BatchedLookup(keys[1..n])
begin
    for i ← 1 to n do
        bucketID[i] ← keys[i]'s bucket ID
        prefetch(bucketIDToGroupID[bucketID[i]])
    for i ← 1 to n do
        groupID[i] ← bucketIDToGroupID[bucketID[i]]
        prefetch(groupInfoArray[groupID[i]])
    for i ← 1 to n do
        groupInfo ← groupInfoArray[groupID[i]]
        values[i] ← LookupSingleKey(groupInfo, keys[i])
    return values[1..n]
```

Evaluation

How fast can SetSep construct GPT?

How does ScaleBricks improve LTE EPC?

How does ScaleBricks scale with number of servers?

Evaluation - SetSep

Dual socket server with 2 Intel Xeon E5-2680 CPUs, 20MB L3 Cache, 64 GB RAM.

1st Experiment - Measure Construction Rate

Construction depends on 3 parameters:

- Number of bits to store hash index and bit array per group
- Number of possible values (sets)
- Number of threads used to parallelize construction.

Results: Construction time:

- increases linearly with num. Of keys
- decreases linearly with num. Of concurrent threads

Evaluation - SetSep

2nd Experiment - Lookup Rate

Test lookup throughput with different batch (entries) sizes.

Results:

- Batching improves performance
- Performance degrades after 32 Mil entries.
- FIBs smaller than 500k entries perform faster without batching.

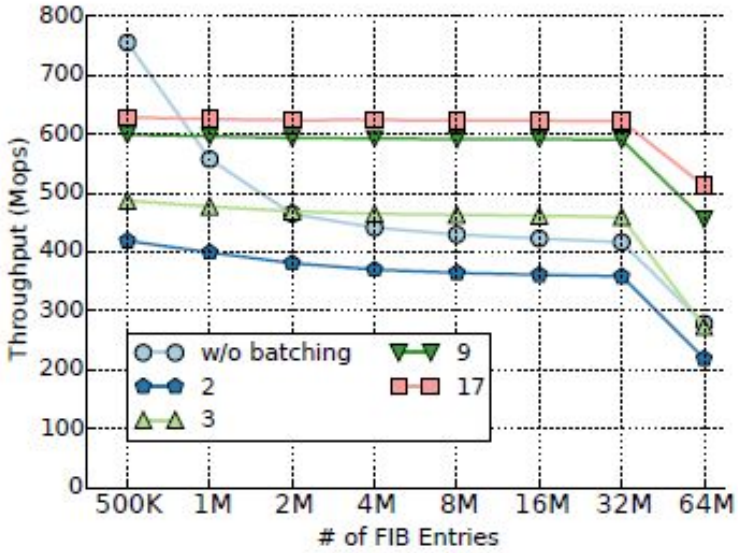


Figure 7: Local lookup throughput of SetSep (GPT)

Evaluation Results

Construction setting			Construction throughput	Fallback ratio	Total size	Bits/ key
$x+y$ bits to store a hash function, x-bit hash function index and y-bit array						
16+8	1-bit value	1 thread	0.54 Mkeys/sec	0.00%	16.00 MB	2.00
8+16	1-bit value	1 thread	2.42 Mkeys/sec	1.15%	16.64 MB	2.08
16+16	1-bit value	1 thread	2.47 Mkeys/sec	0.00%	20.00 MB	2.50
increasing the value size						
16+8	**2-bit value**	1 thread	0.24 Mkeys/sec	0.00%	28.00 MB	3.50
16+8	**3-bit value**	1 thread	0.18 Mkeys/sec	0.00%	40.00 MB	5.00
16+8	**4-bit value**	1 thread	0.14 Mkeys/sec	0.00%	52.00 MB	6.50
using multiple threads to generate						
16+8	1-bit value	**2 threads**	0.93 Mkeys/sec	0.00%	16.00 MB	2.00
16+8	1-bit value	**4 threads**	1.56 Mkeys/sec	0.00%	16.00 MB	2.00
16+8	1-bit value	**8 threads**	2.28 Mkeys/sec	0.00%	16.00 MB	2.00
16+8	1-bit value	**16 threads**	2.97 Mkeys/sec	0.00%	16.00 MB	2.00

Table 1: Construction throughput of SetSep for 64 M keys with different settings

Evaluation - ScaleBricks

Measure PFE performance using 4 servers, each with 2 Intel Xeon E5-2697 v2 CPU, 2.7Ghz, 30MB L3 cache, 128 GB DDR RAM. Intel 82599ES 10GbE NICS.

Simulates downstream traffic using Spirent SPT-N11U Ethernet testing platform.

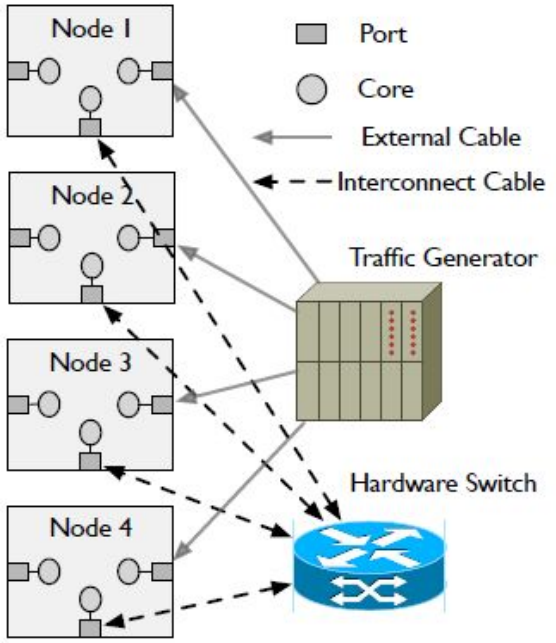


Figure 6: Cluster setup

Evaluation

- Compared to the baseline, ScaleBricks reduced average latency by 10%
- ScaleBricks improves end to end latency due to faster memory access and eliminating the extra hop.
- A single CPU core can handle 60K updates a second
- ScaleBricks cluster can scale up to 5.7 time more FIB entries compared with a cluster using Full FIB entries.

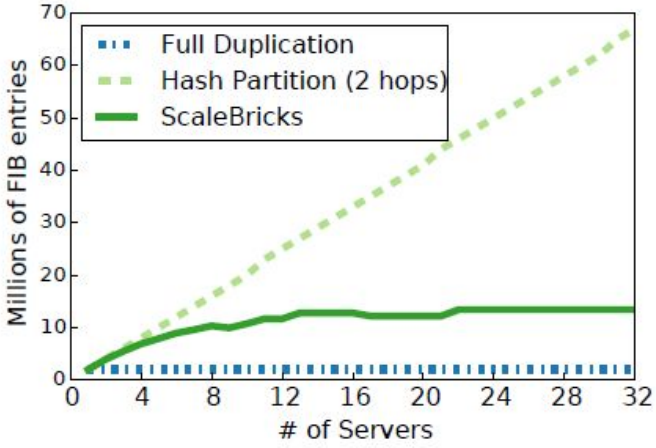


Figure 11: # of FIB entries with different # of servers

Summary

Summary

- ScaleBricks is a new approach for scaling up clustered network applications.
- Provides efficient mechanisms for partitioning full forwarding state, constructing, and updating a Global Partitioning Table.
- SetSep stores compact mapping for keys (flow IDs) to values (handling node IDs).
- SetSep requires 3.5 bits/key to store mapping from arbitrary keys to 2 bit values, along with extremely fast lookups
- Improves packet forwarding throughput of a 4-node LTE-to-packet-network gateway by 23% and reduce latency by 10%