# Taiji: Managing Global User Traffic for Large-Scale Internet Services at the Edge

2019

David Chou, Tianyin Xu, Kaushik Veeraraghavan, Andrew Newell, Sonia Margulis, Lin Xiao, Pol Mauri Ruiz, Justin Meza, Kiryong Ha, Shruti Padmanabha, Kevin Cole, Dmitri Perelman

Francis Rinaldi

# Motivation

- Facebook used a static edge-to-datacenter traffic mapping.
- As Facebook became more popular, the traffic became more variant.

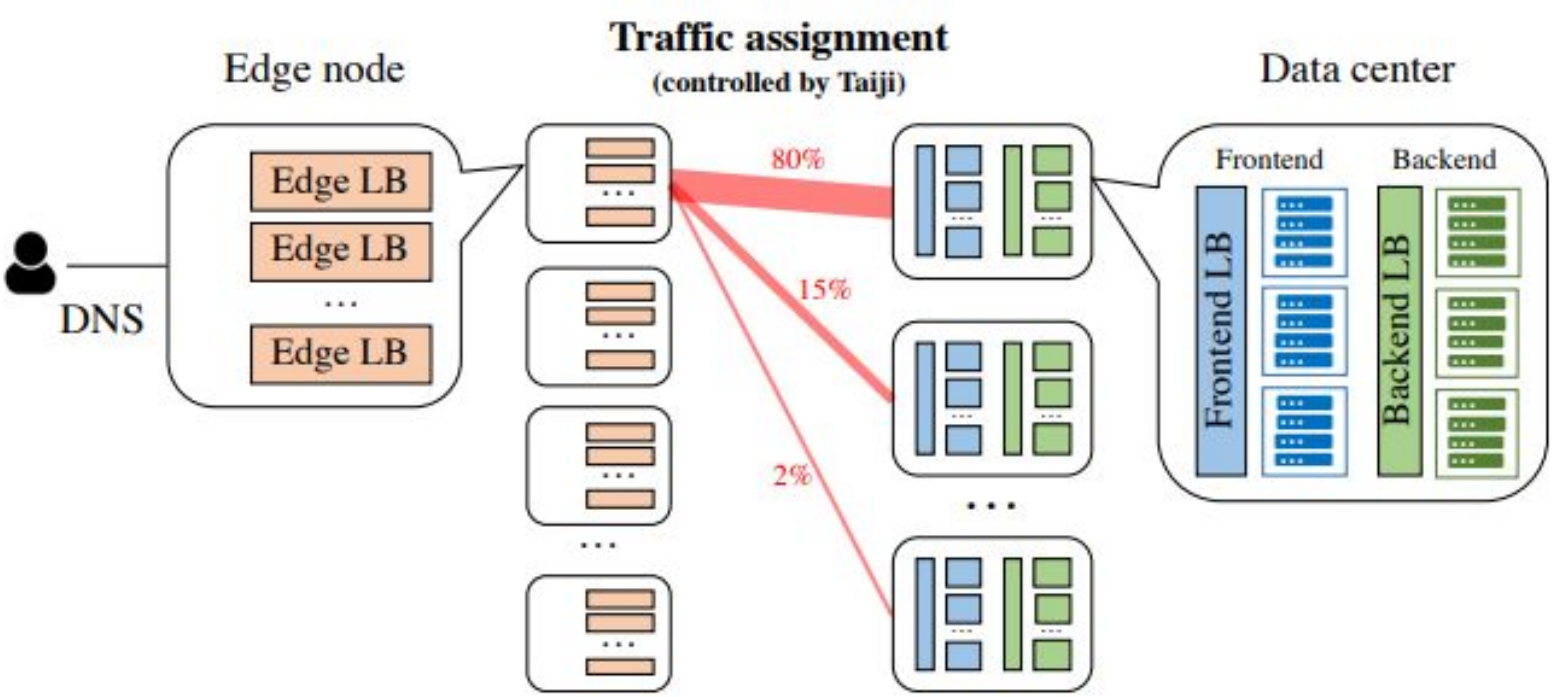



**Figure 3.** An overview of Facebook's infrastructure and the role Taiji plays. Taiji decides how dynamic user traffic at the edge nodes is routed to data centers by continuously adjusting the Edge LB's routing configurations.

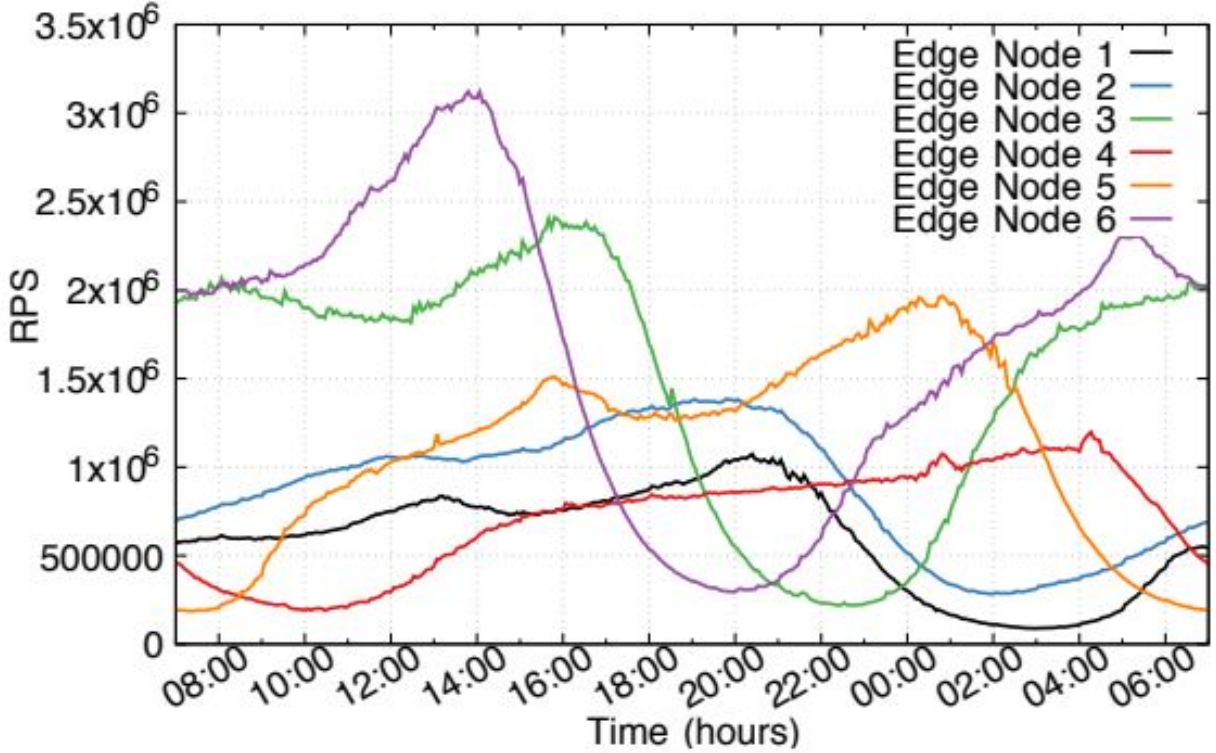



**Figure 1.** User requests per second (RPS) received at six edge nodes in different geo-locations over a 24-hour day. Observe that peak load differs substantially from trough at every edge node.

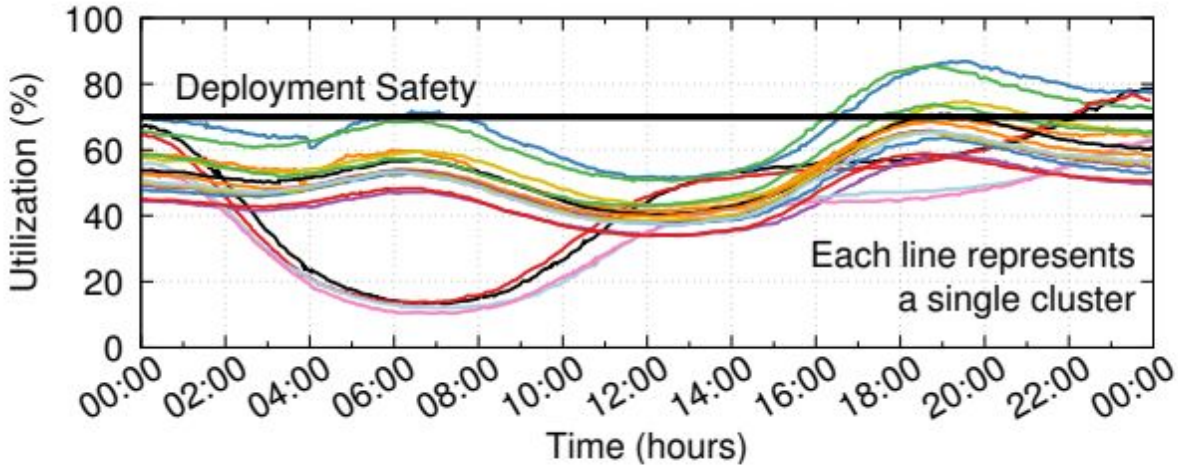



**Figure 2.** The normalized frontend utilization for Facebook's main web service on a day in April 2015 using static edge-to-datacenter traffic mapping. Each line denotes the utilization for one frontend cluster composed of several thousand web servers; each data center houses one or more frontend clusters. Observe that four clusters were nearly idle at 10% utilization, while seven were above 60% utilization. The black line denotes the "Deployment Safety" threshold above which we cannot safely rollout software updates that require restarting web servers.

# Motivation

Dynamic edge-to-datacenter traffic routing:

- Guards against Capacity Crunches
- Improves Product Heterogeneity
- Improves Hardware Heterogeneity
- Improves Fault Tolerance

# Background

- Large Backbone Network
- Limiting Factor is Backend Capacity
- Static vs Dynamic Content
- Stateless vs Sticky Traffic

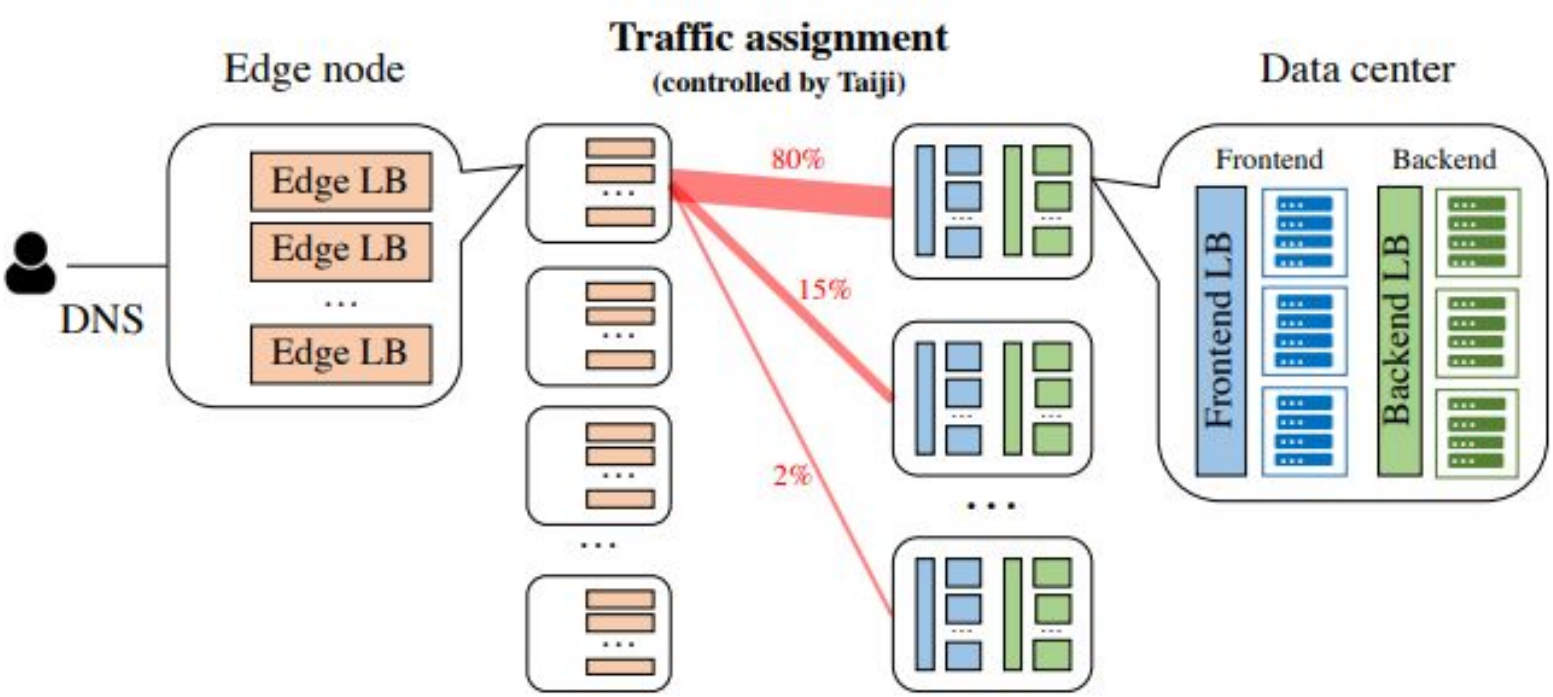



**Figure 3.** An overview of Facebook's infrastructure and the role Taiji plays. Taiji decides how dynamic user traffic at the edge nodes is routed to data centers by continuously adjusting the Edge LB's routing configurations.

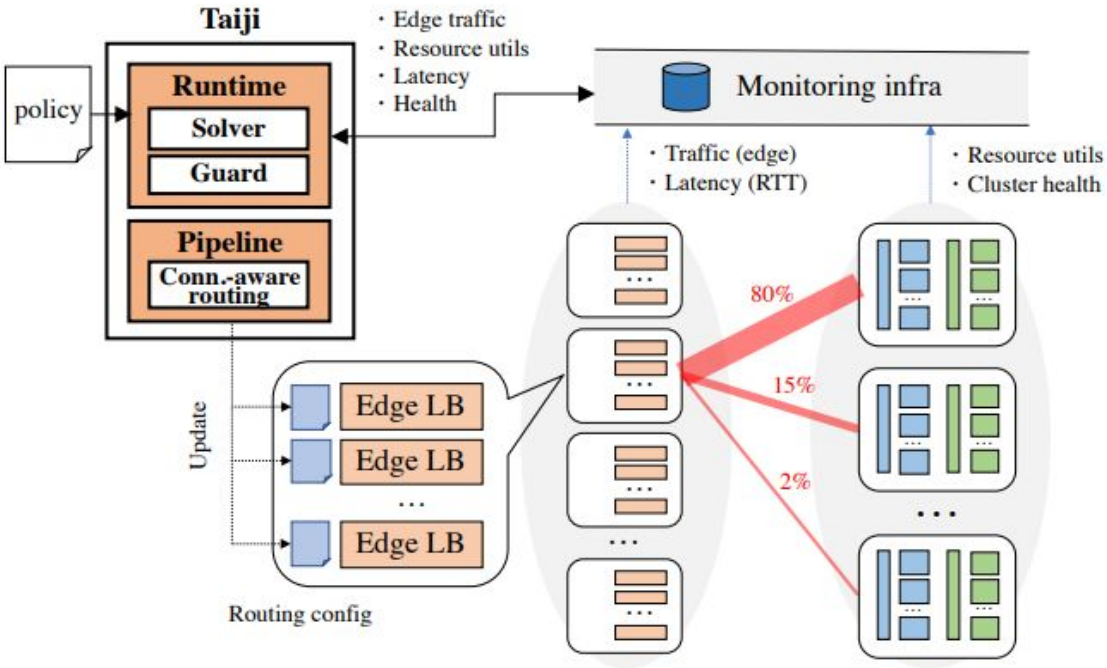



**Figure 4.** Taiji consists of two components: (1) a Runtime that generates a routing table based on various system inputs (traffic load, data center utilization, latency between each edge node and data center pair, etc.) and a service-level policy; (2) a Traffic Pipeline translates the routing table into fine-grained entries to instruct each Edge LB how to route traffic of a user to a particular data center.

# Taiji Runtime

- Generates routing table from policy and dynamic data every epoch (5 min)
- Contains a reader that reads, normalizes, and aggregates data

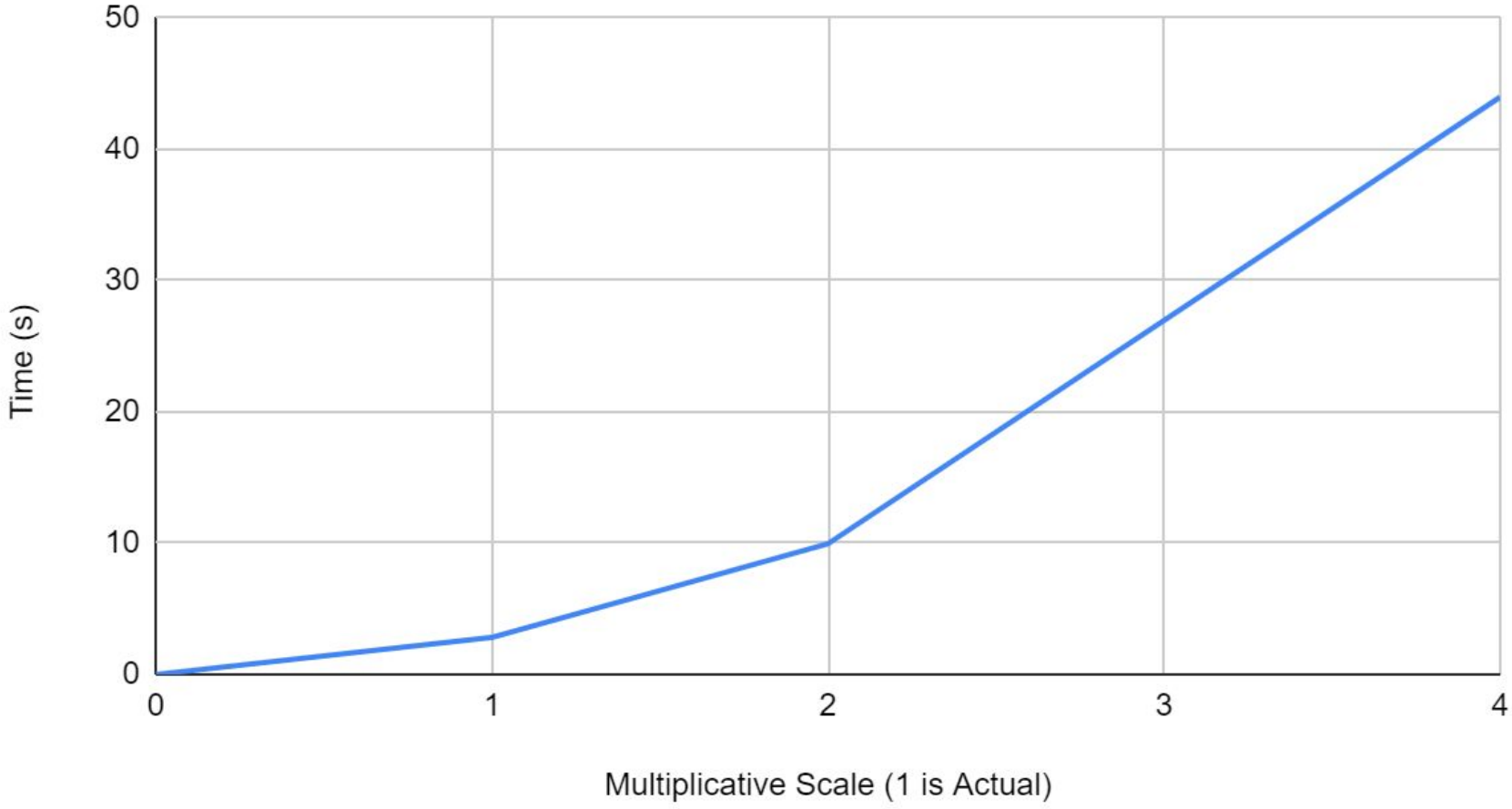
# Policy

- Specifies constraints and objectives for the assignment solver
- Facebook’s most used policy is:
  - Balance Data Center Utilization
  - Optimize Network Latency

# Assignment Solver

- Uses a local search algorithm
- Iterates the following:
  - Swaps a unit of traffic between two data centers
  - Updates model with the better data center
  - Done if no changes are made over all traffic
- Finding the local maximum
- Uses symmetry to reduce recalculation

# Average Time to Generate an Assignment Solution by Scale



# Safety Guards

- Onloading Safety Guard (0.04)
  - Constant upper bound on the increase of utilization of a data center per epoch
- Traffic Fraction Safety Guard
  - Upper bound on the fraction of traffic sent to a data center

# Safety Guards

- Minimum Shift Limit (~1%)
  - Minimum threshold for the percentage of traffic changes to cause an update
- Dampening Factor (80%)
  - Dampens the change in traffic in order to prevent overshooting

# Sensitivity Analysis

- Continually runs to tune the safety guard values
- Found:
  - Traffic shifts are more costly for stateful than stateless services
  - Traffic shifts for stateless services initially cause worse latency and throughput

# Taiji Traffic Pipeline

- Takes the routing table from the Taiji Runtime
- Groups users into buckets via connection-aware-routing
- Generates routing entries that specify which buckets go to which data centers
- Sends routing entries to local Edge LBs
- Takes about 1 minute

# Connection-Aware Routing

- Introduces locality in traffic routing
- Groups highly-connected users into buckets via classical balanced graph partitioning
  - Each bucket is roughly the same size
  - Maximizes connections within each bucket

# Bucket Size Issue

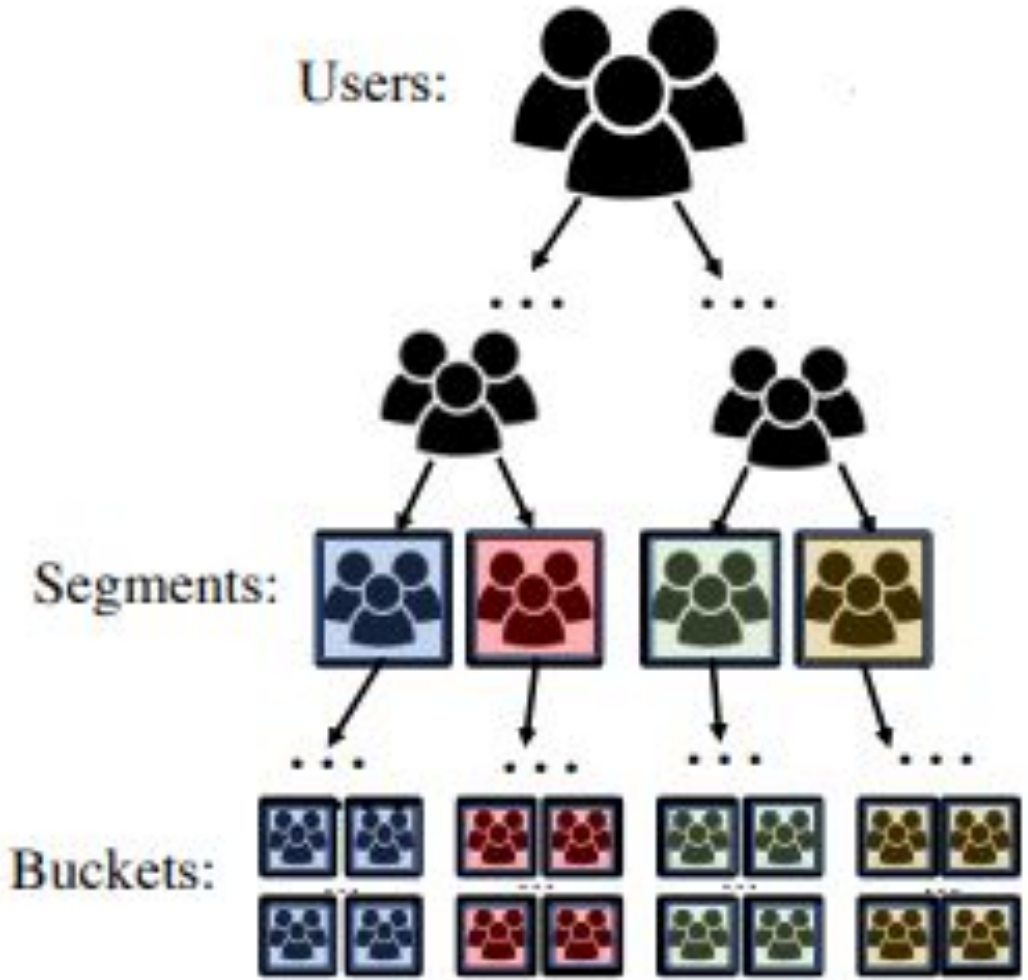
- Trade-off for the bucket size:
  - Increasing yields more cache efficiency
  - Decreasing yields more routing accuracy
- Taiji wants a fine granularity of traffic
  - Each bucket represents ~0.01% of global traffic
  - This can separate large communities, resulting in losing traffic locality benefits.

# Solution

- Combination of:
  - Offline User-to-Bucket Assignments
  - Online Bucket-to-Datacenter Assignments
- Increases traffic locality for community connections from 55% to 75%

# User-to-Bucket Assignment

- Partitions all users across a complete binary tree
- The root contains all users
- For each node, if its user size is not ~0.01%:
  - Perform a balanced bipartition of the node's users between two children that minimizes edge cuts


Users:
Segments:
Buckets:

# User-to-Bucket Assignment

- Performed offline once a week
- Limits total user movement across buckets per week to 5% to limit rerouting
- In practice, <2% of users are moved per week
- Two types of connections are ignored:
  - User to Highly-Connected Entity
  - One-Time Interactions

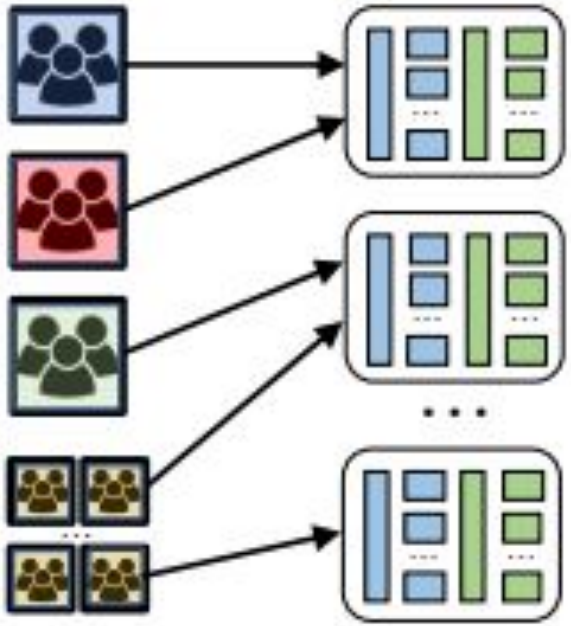
# Bucket-to-Datacenter Assignment

- Groups buckets that are part of same-sized subtrees into segments
  - Represented by the root of the subtree
- Strives to assign buckets in the same segment to the same datacenter
- Creates more traffic locality
- Wants to be stable: buckets are barely reassigned

## Routing Configuration

$E_1$: { $DC_1$: {$B_1$,$B_2$,...,$B_8$},
$DC_2$: {$B_9$,$B_{10}$,...,$B_{14}$},
$DC_3$: {$B_{15}$,$B_{16}$}}

...



# Bucket-to-Datacenter Assignment

- A level L is picked
  - The segments will be the nodes on level L
  - There will be $2^L$ segments
- Tradoff:
  - Smaller L increases traffic locality
  - Bigger L increases stability
- Empirically picked L to be 7

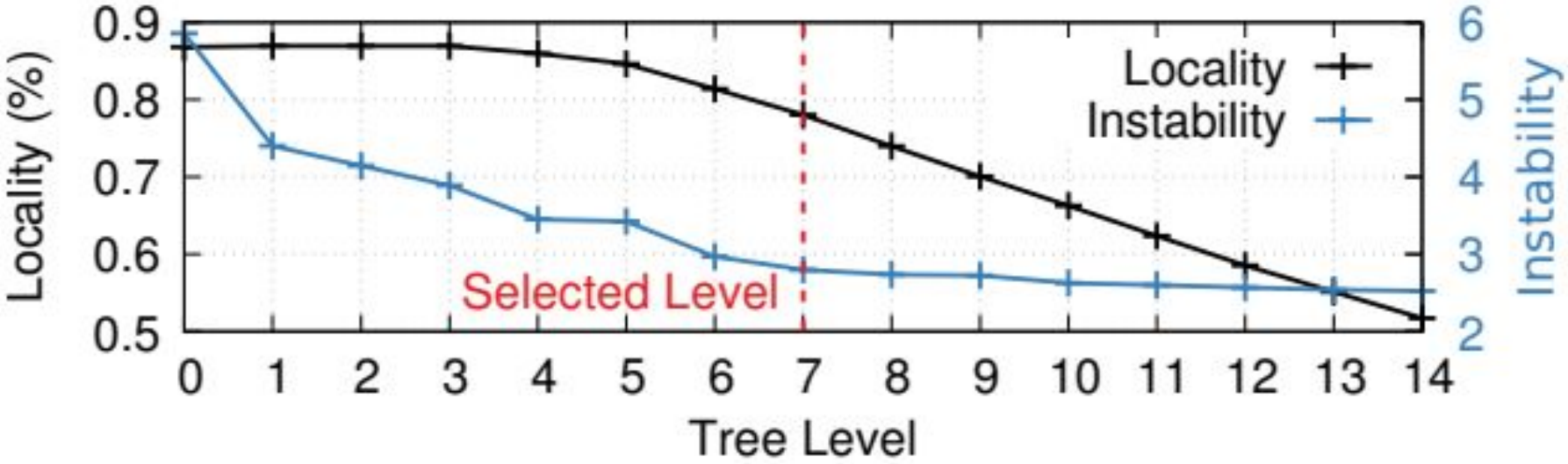



**Figure 10.** The tradeoff between locality and instability in selecting the tree level used by connection-aware routing. Locality is measured by the percentage of connections routed together, while the instability is measured by the average number of data centers a user bucket may be routed in a week.

**Algorithm 1** Stable Segment Assignment

$B \leftarrow$ list of buckets
$D \leftarrow$ list of data center names
$W_B \leftarrow$ bucket weights
$W_D \leftarrow$ data center weights
$S \leftarrow$ number of segments
$T \leftarrow \emptyset$
**for** $d \in D$ **do**
  $P \leftarrow \text{PermutationWithSeed}(S, d)$
  **for** $b \in B$ **do**
    $\text{segment} \leftarrow \lfloor \frac{b \cdot |B|}{S} \rfloor$
    $T \leftarrow T \cup \langle P[\text{segment}], d, b \rangle$
  **end for**
**end for**
SortLexicographically($T$)
$A \leftarrow \emptyset$
**for** $\langle \text{preference}, d, b \rangle \in T$ **do**
  **if** $b \notin A \wedge W_D[d] > 0$ **then**
    $A[b] \leftarrow d$
    $W_D[d] \leftarrow W_D[d] - W_B[b] \cdot \frac{\sum W_D}{\sum W_B}$
  **end if**
**end for**
**return** $A$

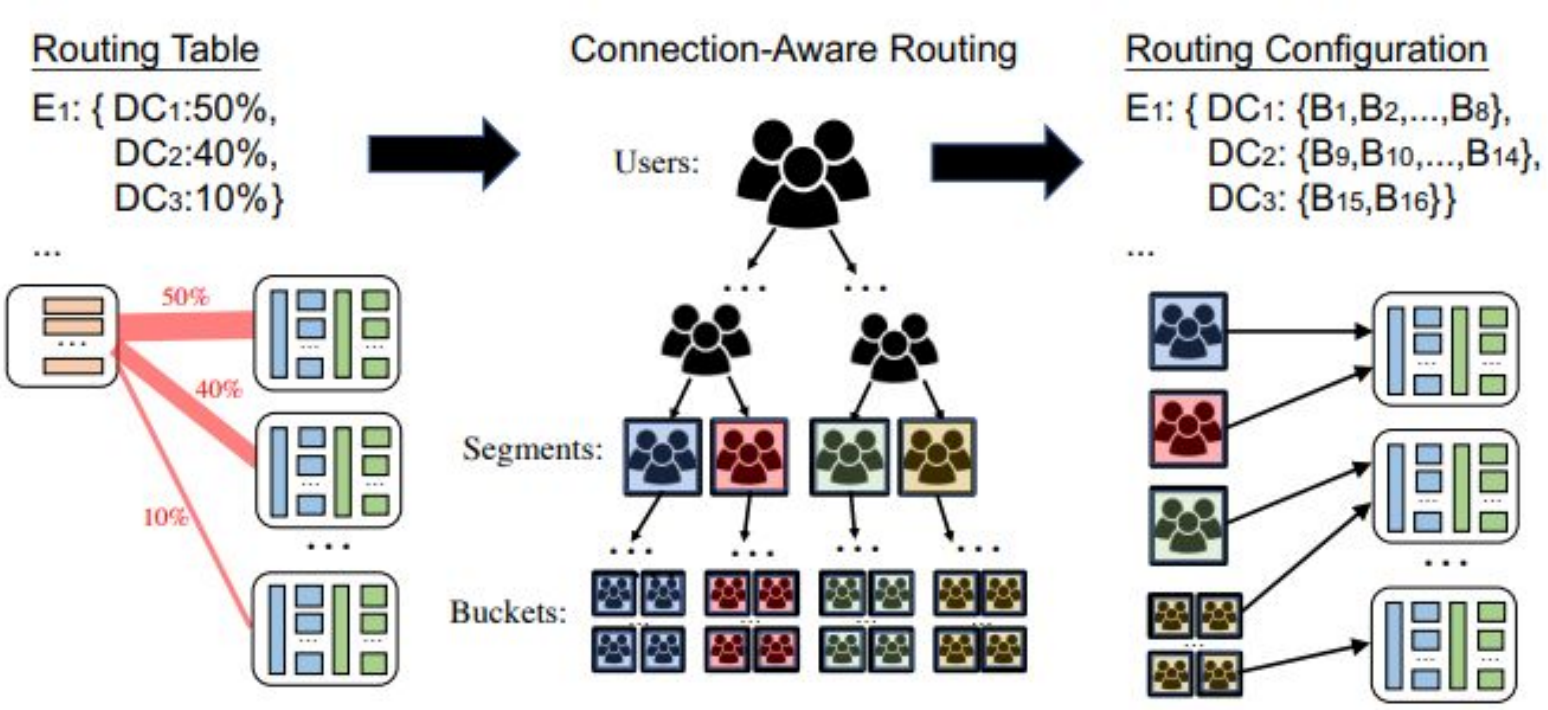



**Figure 5.** Connection-aware routing transforms a routing table into routing configuration entries that specify how to route user traffic based on user connections in a community graph. We do not depict the bucket weights (per-bucket traffic fraction) for clarity.

# Home for Mappings

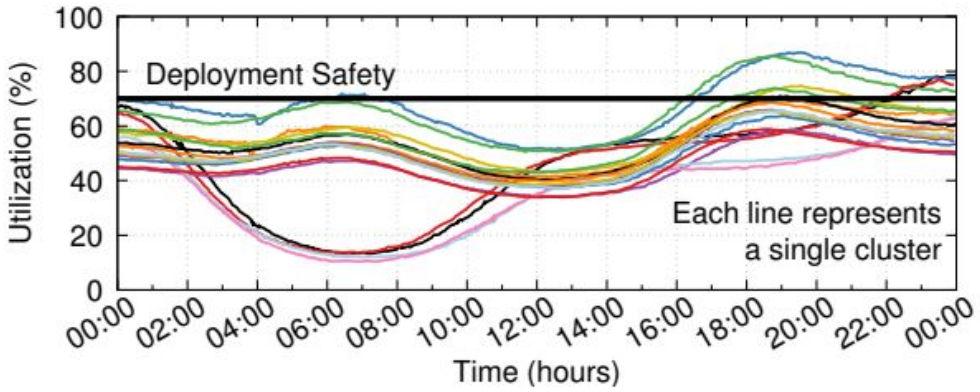
- Each Edge LB contains the bucket-to-datacenter mapping for their edge machine
- Each datacenter contains the user-to-bucket mapping.
- User has a cookie that adds the user’s bucket ID to that user’s requests

# Fault Tolerance

- If a datacenter fails or its traffic is drained, Taiji excludes that datacenter from traffic balancing.
- Backend safety is the primary concern

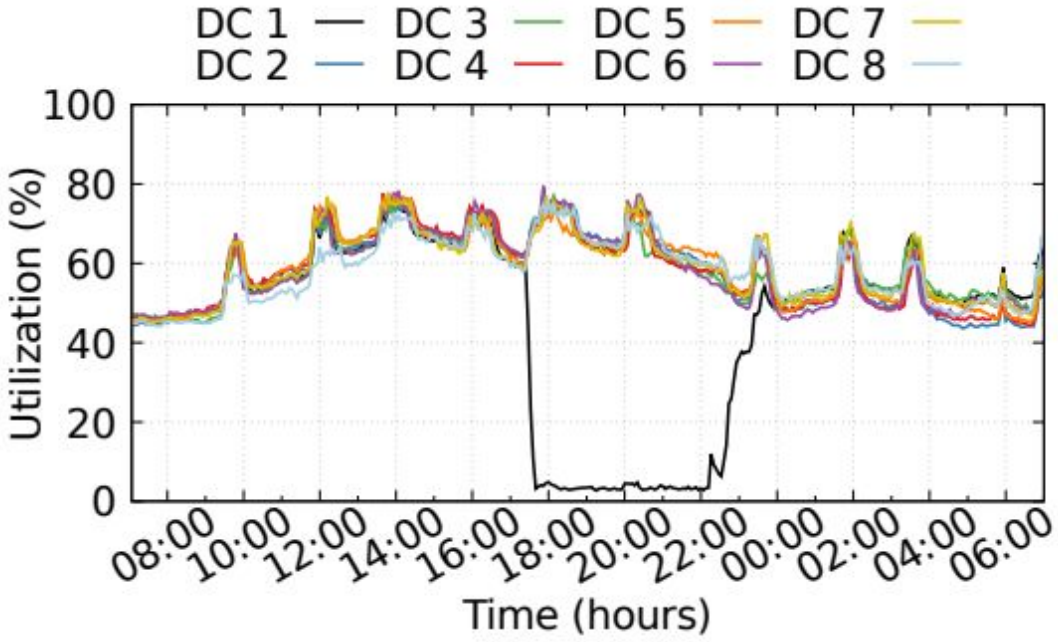
2015

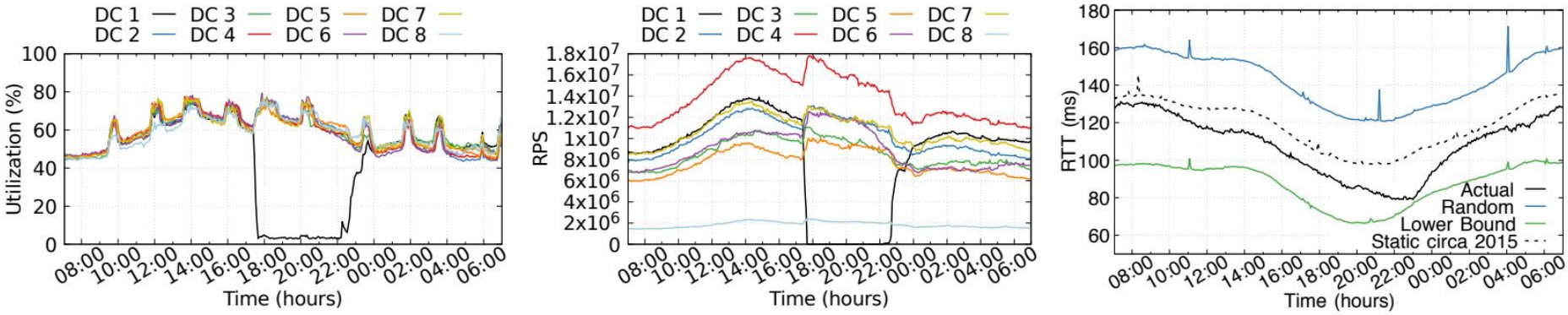
Static




2019

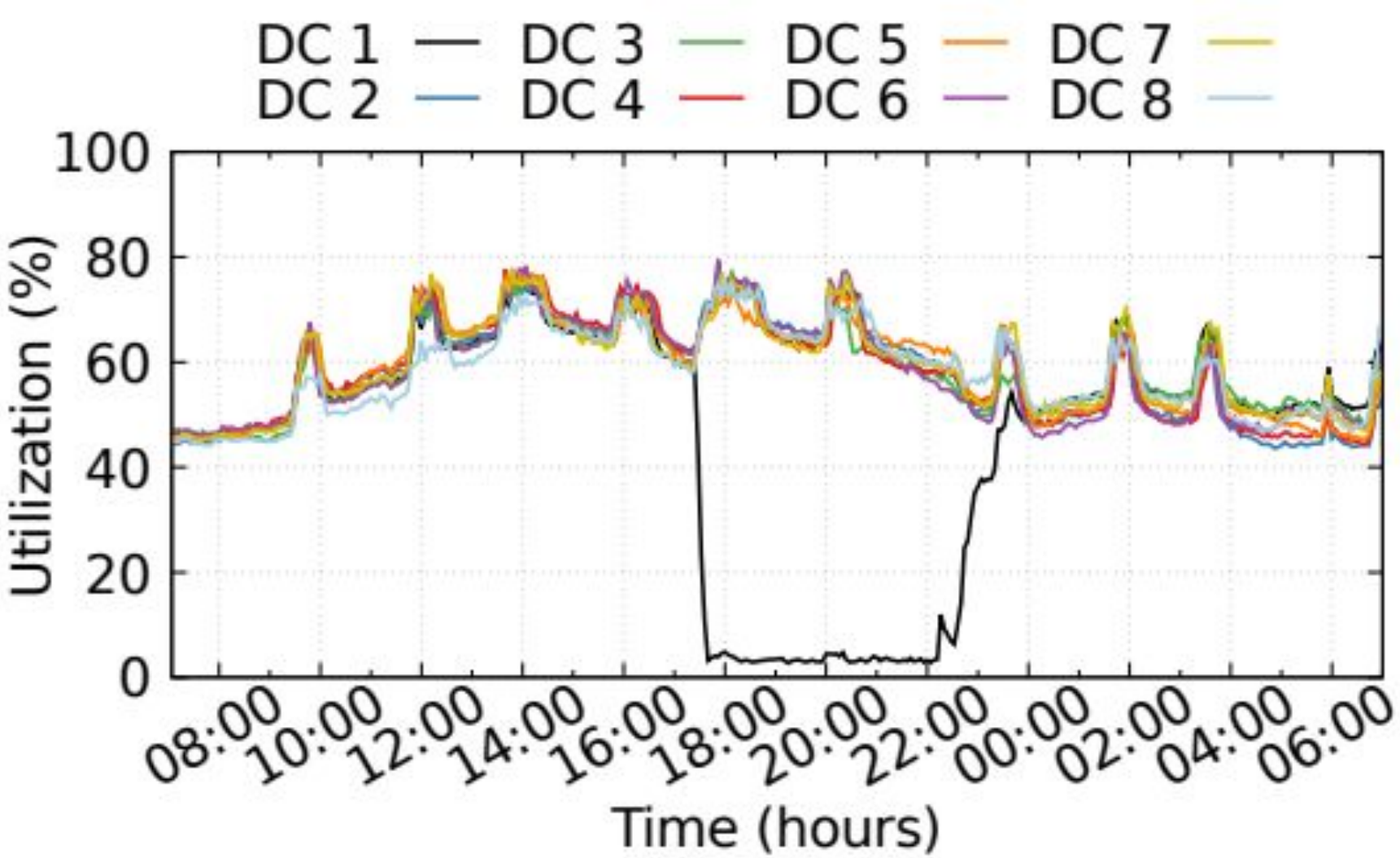
Taiji

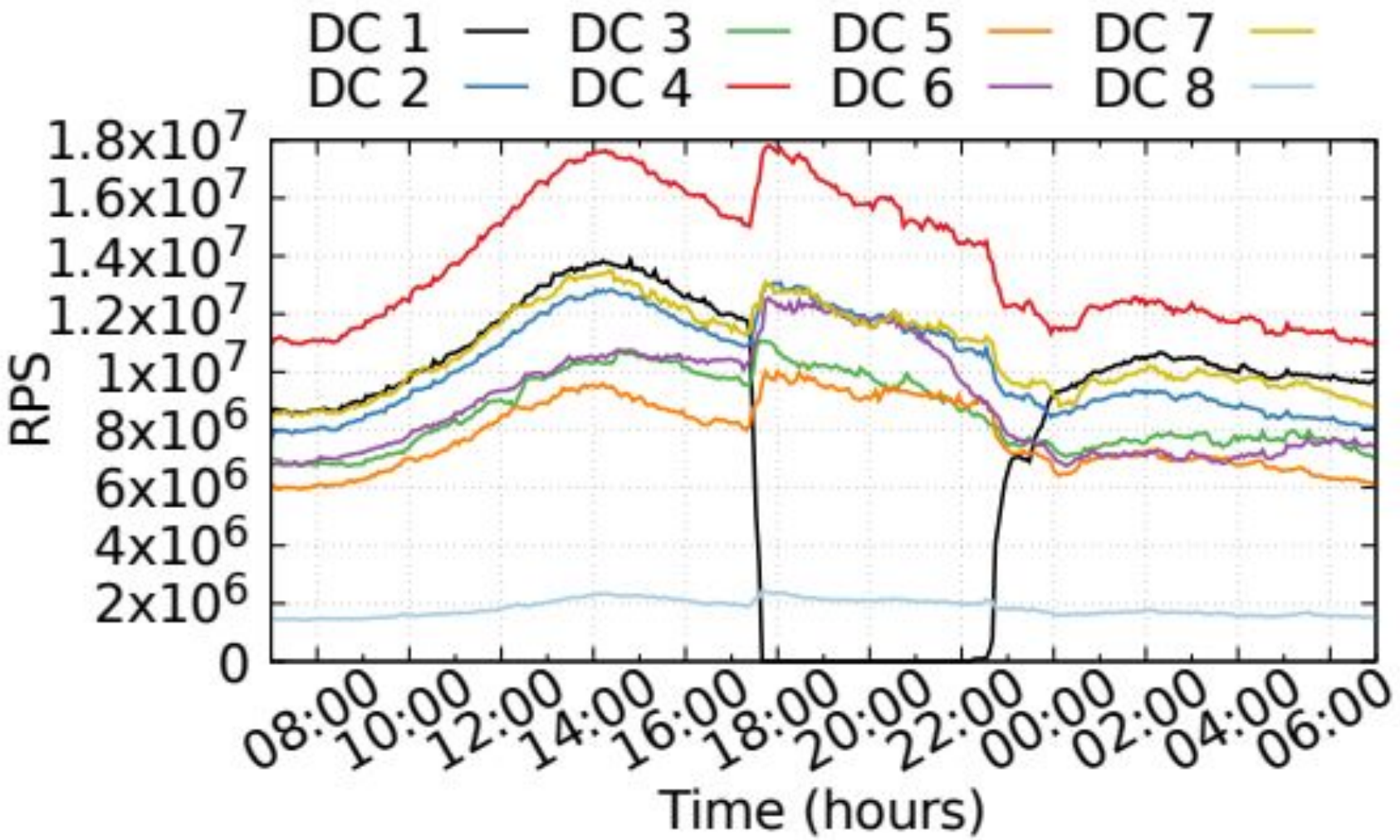


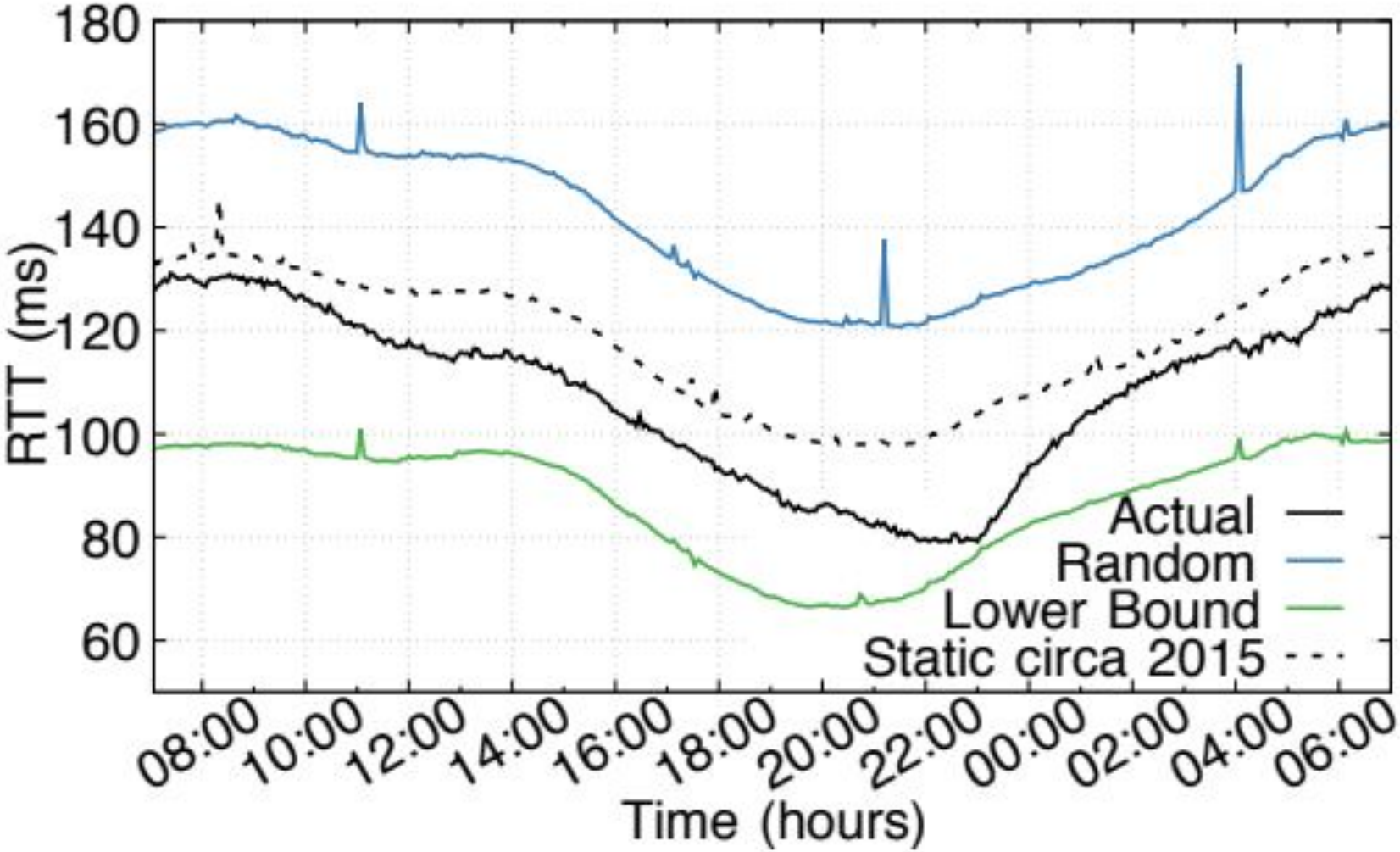



**(a)** Data center utilization **(b)** Requests Per Second **(c)** Round Trip Time

**Figure 6.** Taiji manages edge-to-datacenter traffic for Facebook's largest web service. This figure (from a day in 2019) depicts: (a) data center utilization, (b) traffic load (Requests Per Second) allocated to each data center, and (c) average network latency (Round Trip Time).


DC 1
DC 2
DC 3
DC 4
DC 5
DC 6
DC 7
DC 8
100
80
60
40
20
0
Utilization (%)
08:00
10:00
12:00
14:00
16:00
18:00
20:00
22:00
00:00
02:00
04:00
06:00
Time (hours)


DC 1
DC 2
DC 3
DC 4
DC 5
DC 6
DC 7
DC 8
RPS
1.8x10^7
1.6x10^7
1.4x10^7
1.2x10^7
1x10^7
8x10^6
6x10^6
4x10^6
2x10^6
0
08:00
10:00
12:00
14:00
16:00
18:00
20:00
22:00
00:00
02:00
04:00
06:00
Time (hours)


180
160
140
120
100
80
60
RTT (ms)
Actual
Random
Lower Bound
Static circa 2015
08:00
10:00
12:00
14:00
16:00
18:00
20:00
22:00
00:00
02:00
04:00
06:00
Time (hours)

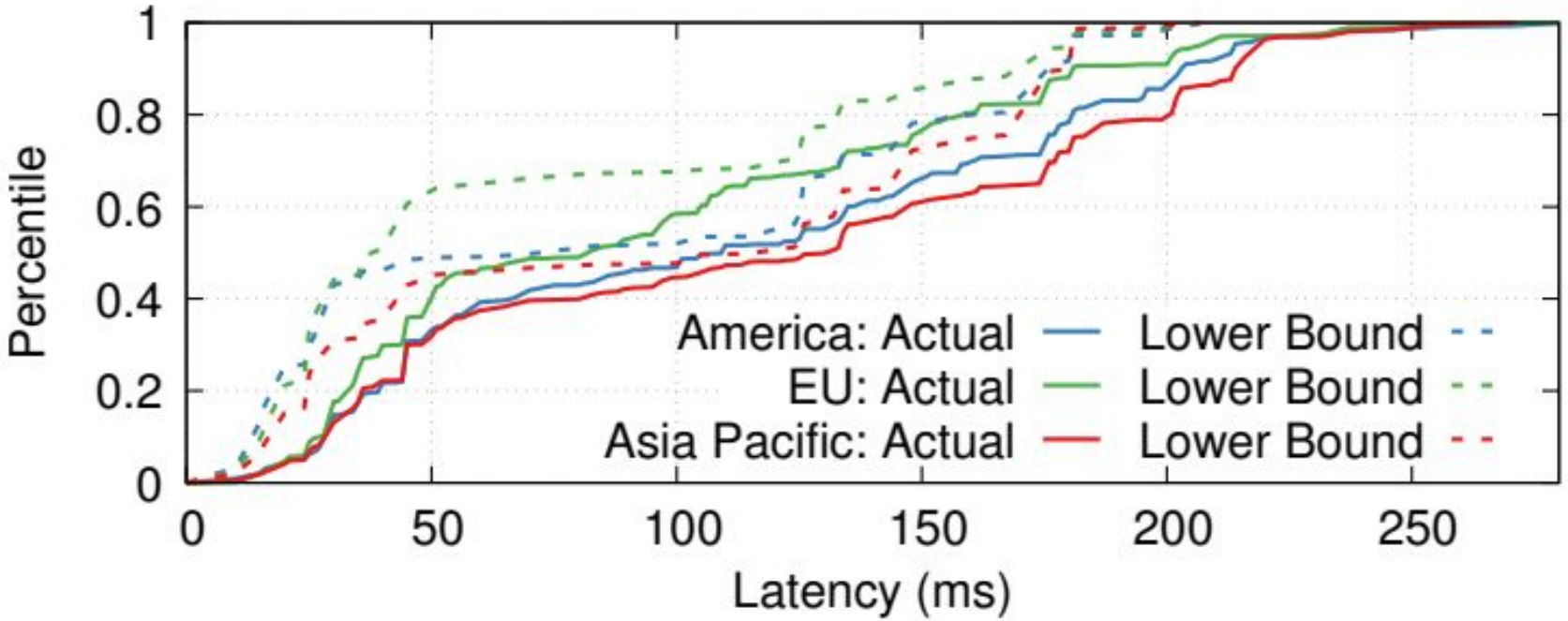
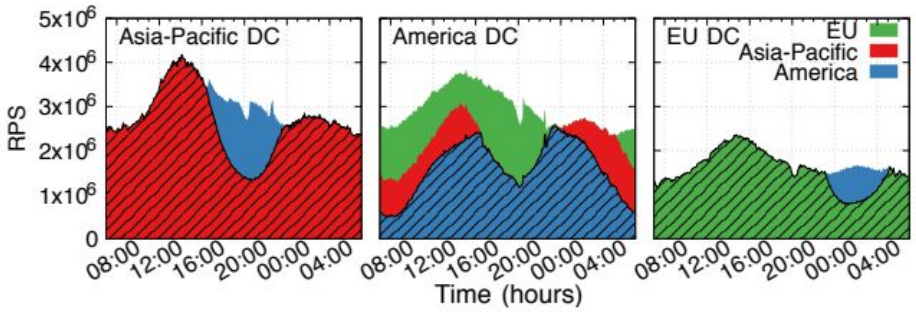
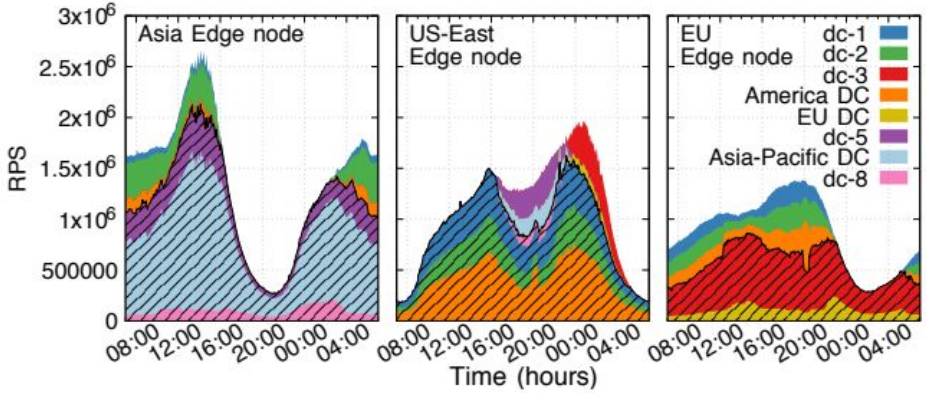



**Figure 7.** The CDF of the edge-to-datacenter latency (in terms of RTT) at the peak load of three different geographical locations.

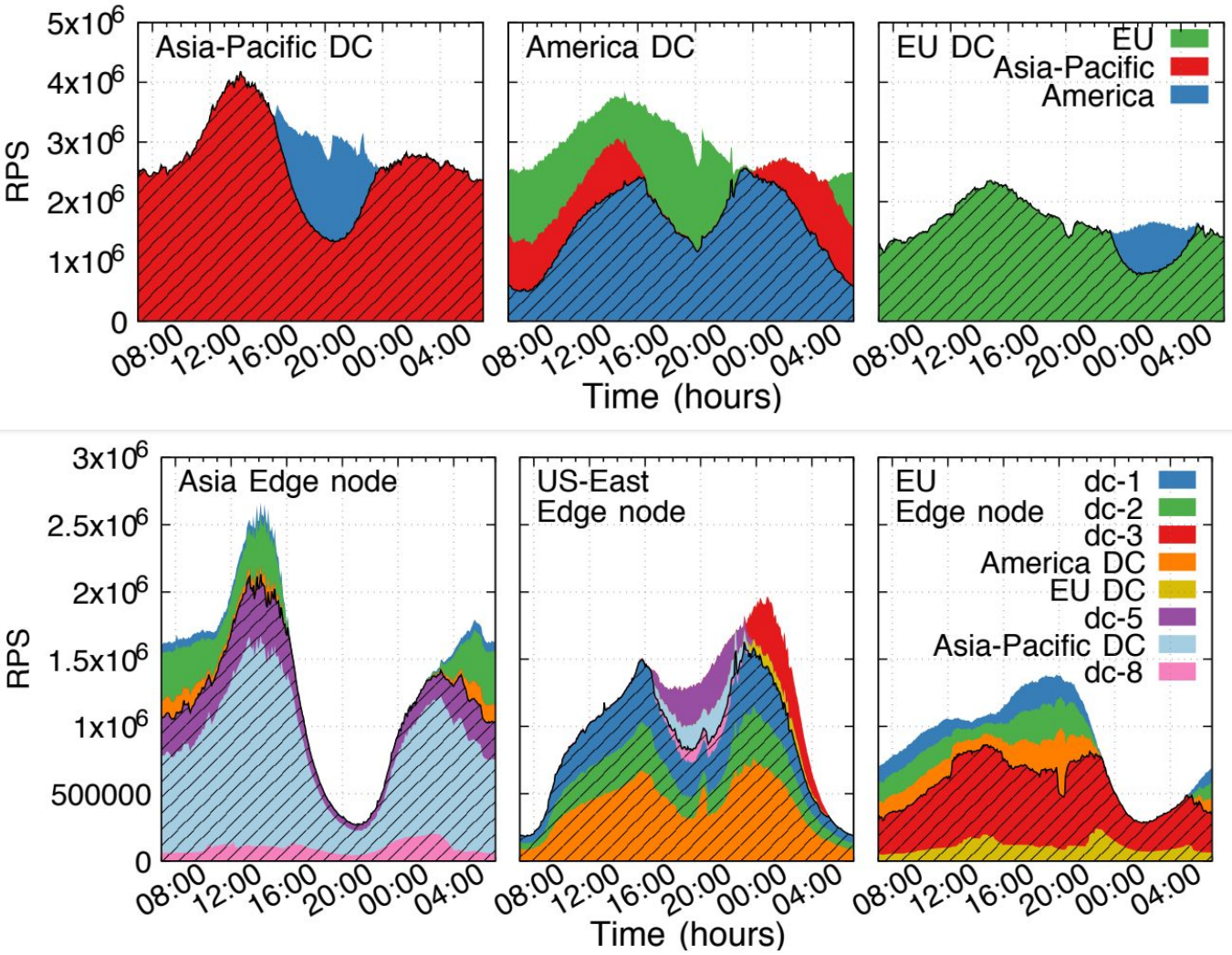



**(a)** Distribution of traffic originated from different edge nodes received at three data centers (Asia-Pacific, America, and EU DCs).




**(b)** Distribution of traffic destined for different data centers at three edge nodes (Asia, US-East, and EU edge nodes).

**Figure 8.** Daily edge-to-datacenter traffic distribution. The shaded area denotes the traffic sent from/to the same geographical regions.


Asia-Pacific DC
America DC
EU DC
EU
Asia-Pacific
America
RPS
5x10^6
4x10^6
3x10^6
2x10^6
1x10^6
0
08:00
12:00
16:00
20:00
00:00
04:00
Time (hours)
Asia Edge node
US-East
Edge node
EU
Edge node
dc-1
dc-2
dc-3
America DC
EU DC
dc-5
Asia-Pacific DC
dc-8
3x10^6
2.5x10^6
2x10^6
1.5x10^6
1x10^6
500000
0

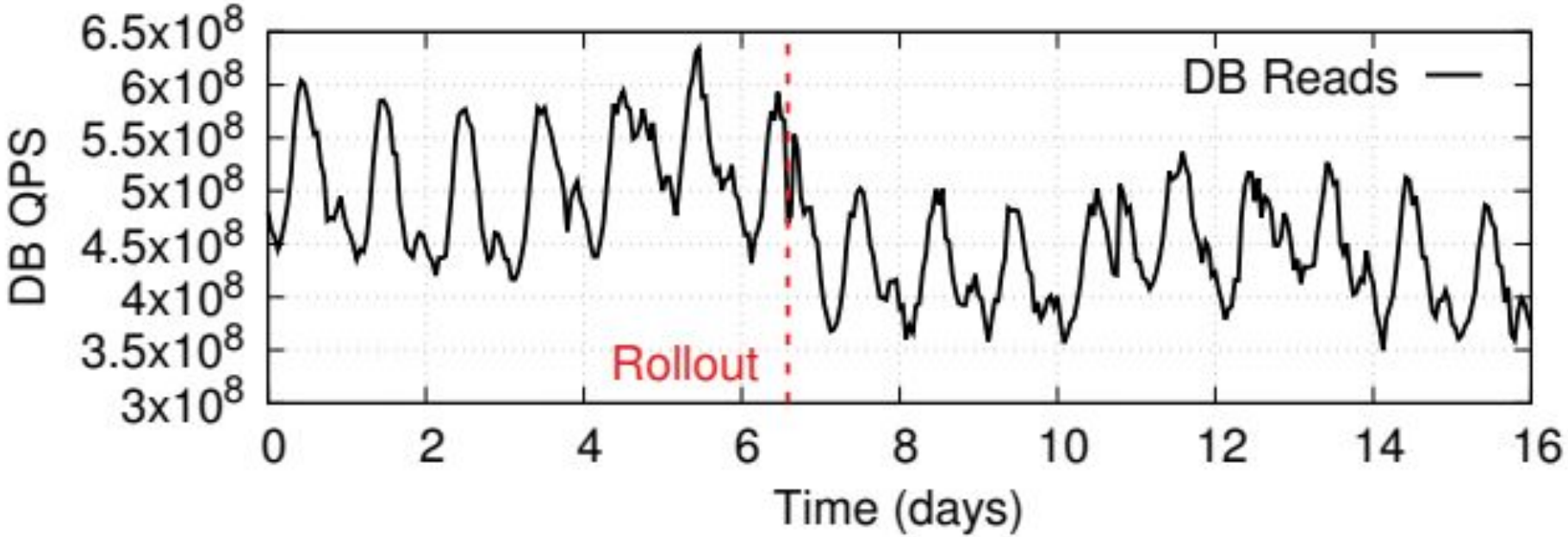



**Figure 9.** Reduction of database queries per second after the deployment of connection-aware routing which improves caching efficiency (the reduced queries are served at the cache layer).

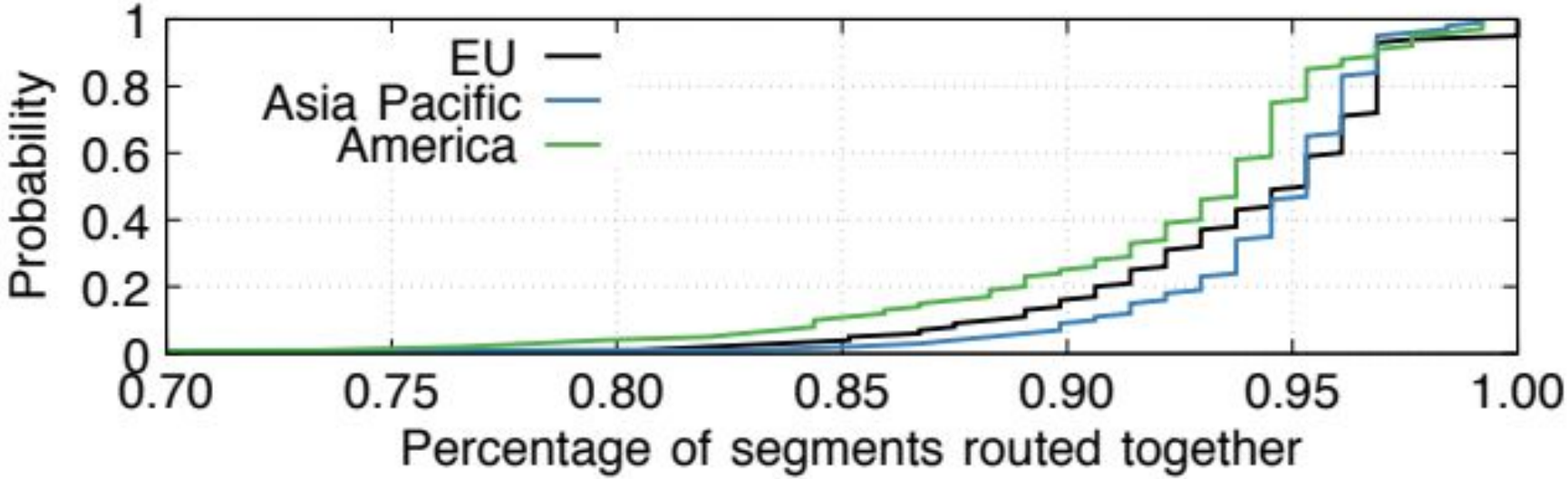



**Figure 11.** The probability of the percentage of segments routed together (to the same data centers) at three different edge nodes over the course of a day. Taiji routes 95+% of the segments together for 80% of the time (the other segments are further split up and routed at the bucket granularity).

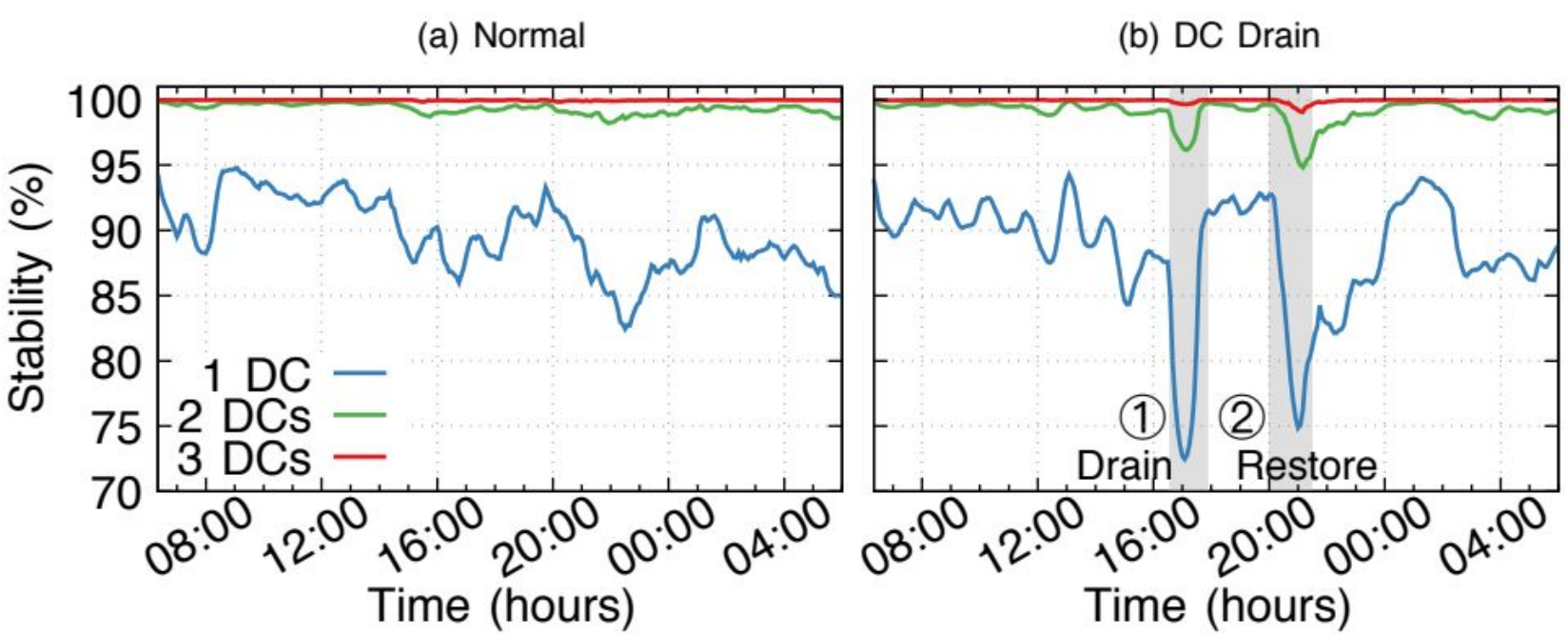



**Figure 12.** Stability of Taiji's routing (a) in a day with the normal state where 98+% users visit no more than 2 data centers, and (b) in a day with a datacenter-level drain events which increase oscillations during the traffic drain and restore periods.

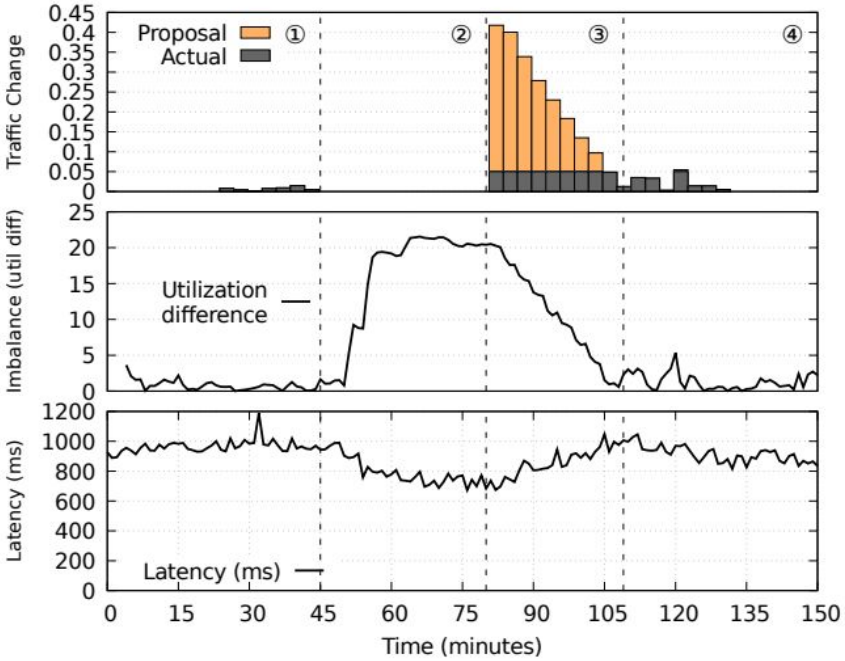
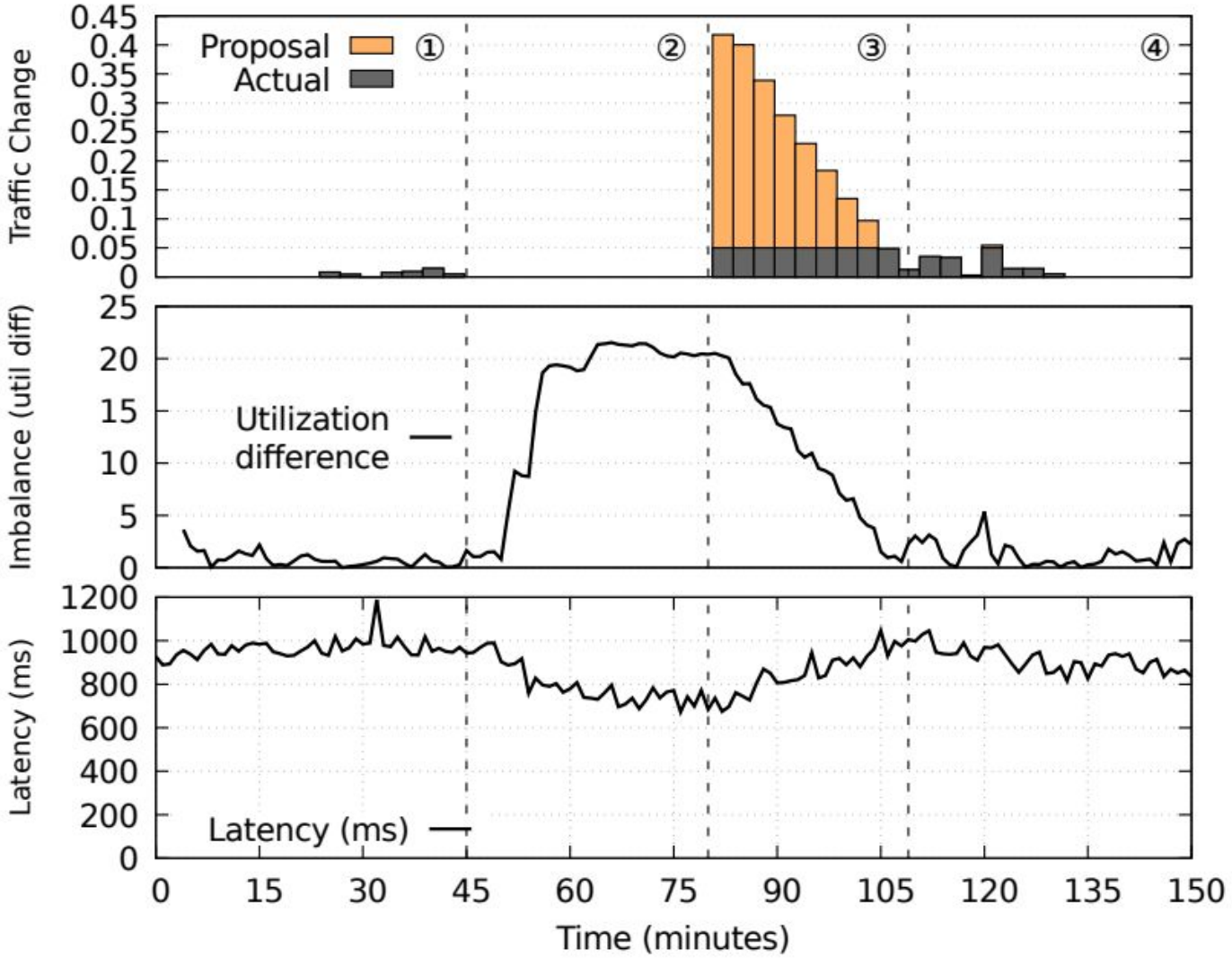



**Figure 13.** Taiji's pacing during datacenter-level drain and restore events. The uppermost subfigure shows the proposed traffic changes versus the actual changes after pacing. The middle subfigure shows the utilization divergence between the target data center and other data centers—the drain leads to significant divergence while Taiji gradually removes the divergence after the restoration. The bottom subfigure shows the 95th percentile of backend processing time—with Taiji's balancing strategy, the drain/restore events do not affect the backend processing latency.


Traffic Change
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
Proposal
Actual
①
②
③
④
Imbalance (util diff)
25
20
15
10
5
0
Utilization difference
Latency (ms)
1200
1000
800
600
400
200
0
Latency (ms)
0
15
30
45
60
75
90
105
120
135
150
Time (minutes)

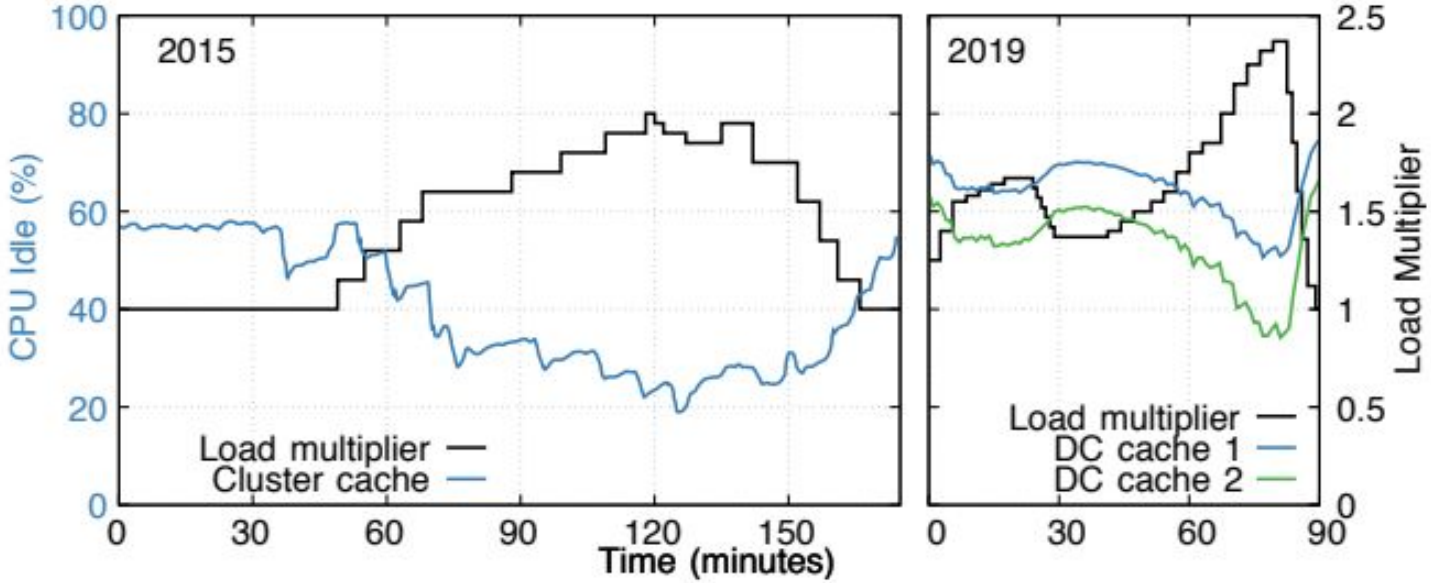



**Figure 14.** Sensitivity analysis for the same services in 2015 and 2019, respectively. The analysis in 2015 takes almost twice as long as the one in 2019. The impact on the backend systems is visible. Specifically, in 2015, there are icicles at the beginning of each step—the introduction of cold traffic leads to cache misses and results in CPU spikes at the backends.

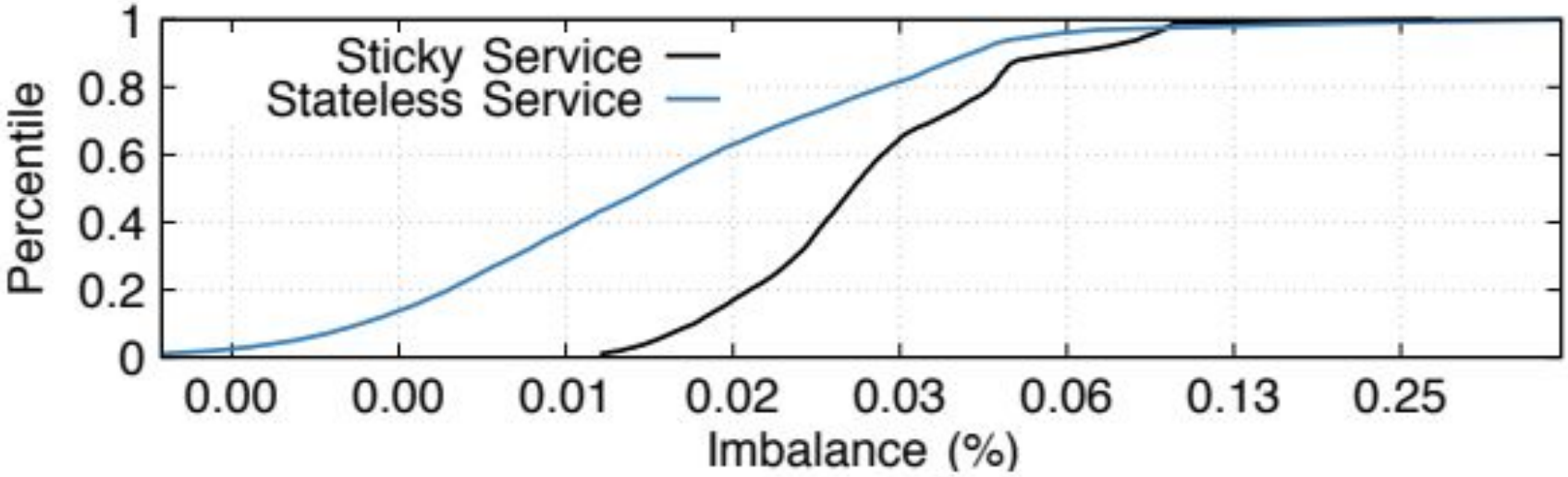
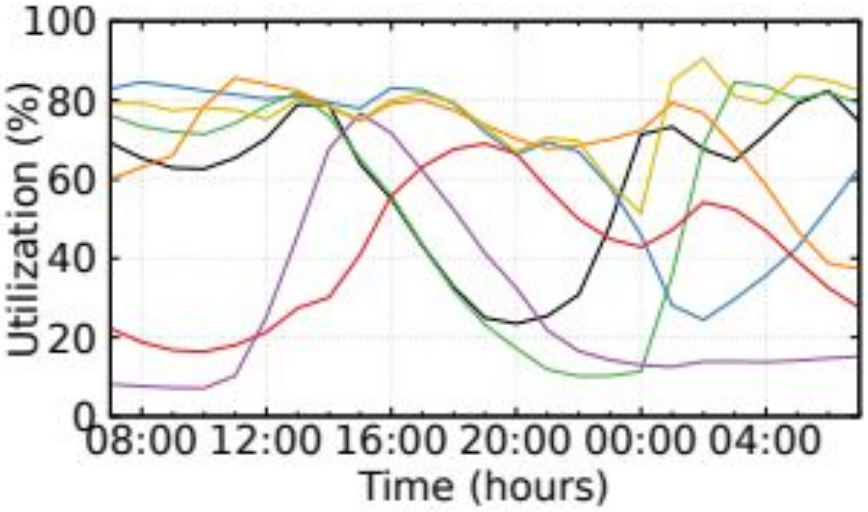
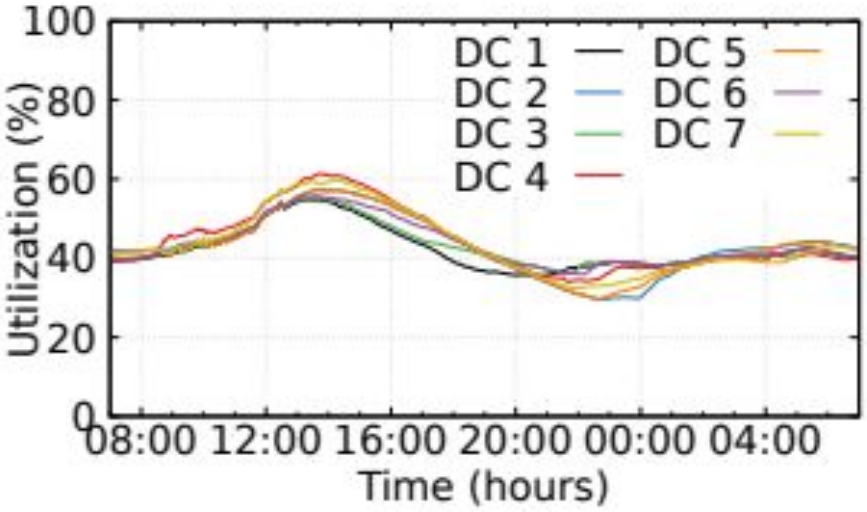



**Figure 15.** CDF of divergence from perfect utilization balancing for a stateless service and a sticky service over two weeks.




(a) Closest-datacenter-first




(b) Latency-aware balancing

**Figure 16.** Frontend utilization of different data centers for Facebook's mobile service, Facebook Lite, with (a) closest-datacenter-first and (b) balancing with latency optimization.

# Lessons

- Customizing load balancing strategy is key to managing infrastructure utilization
- Build systems that keep pace with infrastructure evolution
- Keep debuggability in mind
- Build tools to simplify operations

# Limitations

- Taiji might increase the latency for some users during peak load
- Taiji only considers edge-to-datacenter latency
- Taiji only controls the edge-to-datacenter routing