Seer: Enabling Future-Aware Online Caching in Networked Systems

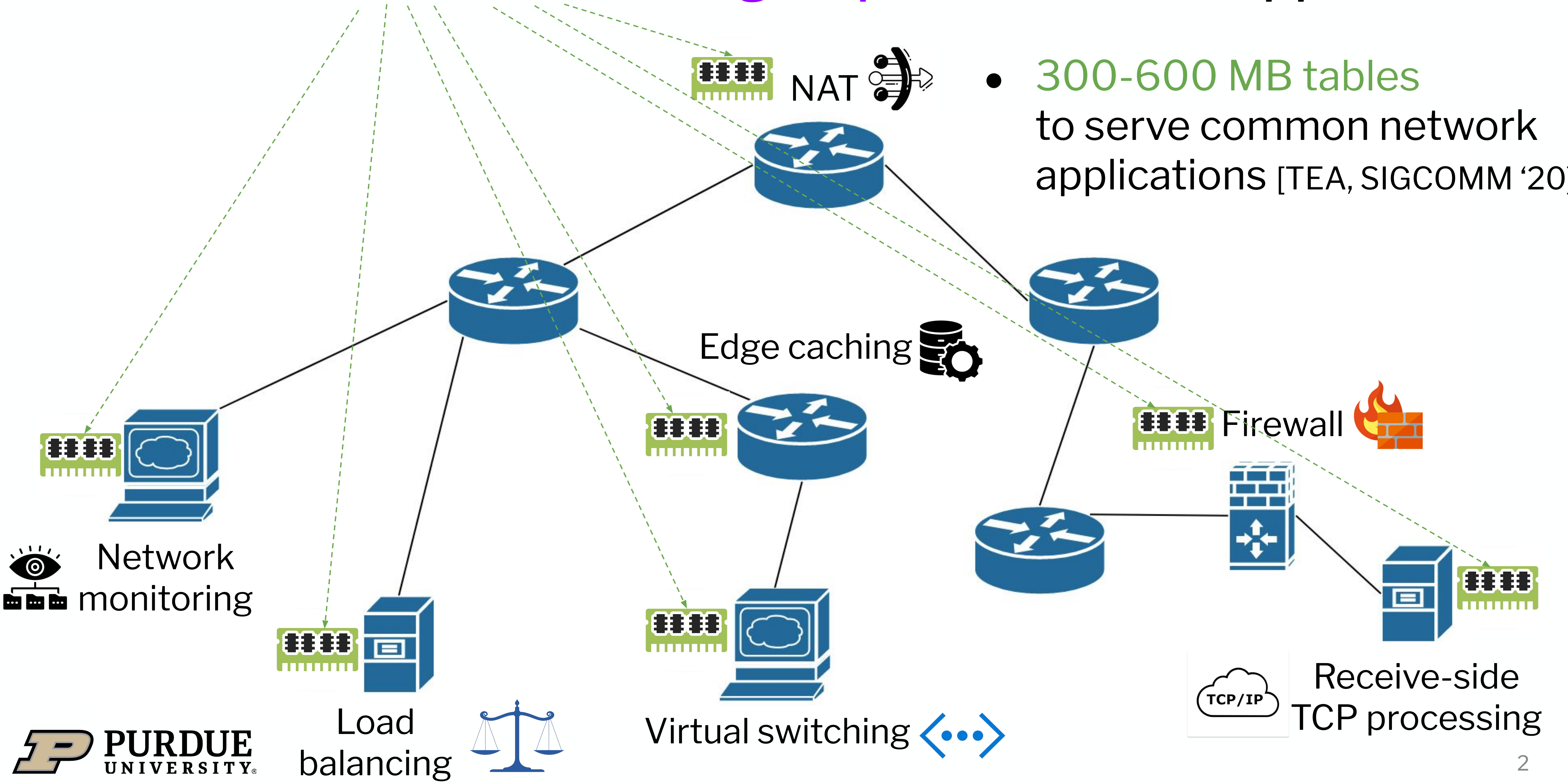
Jason Lei, Vishal Shrivastav

PURDUE UNIVERSITY

nsdi '24

Motivation: State-Intensive High-Speed Network Applications

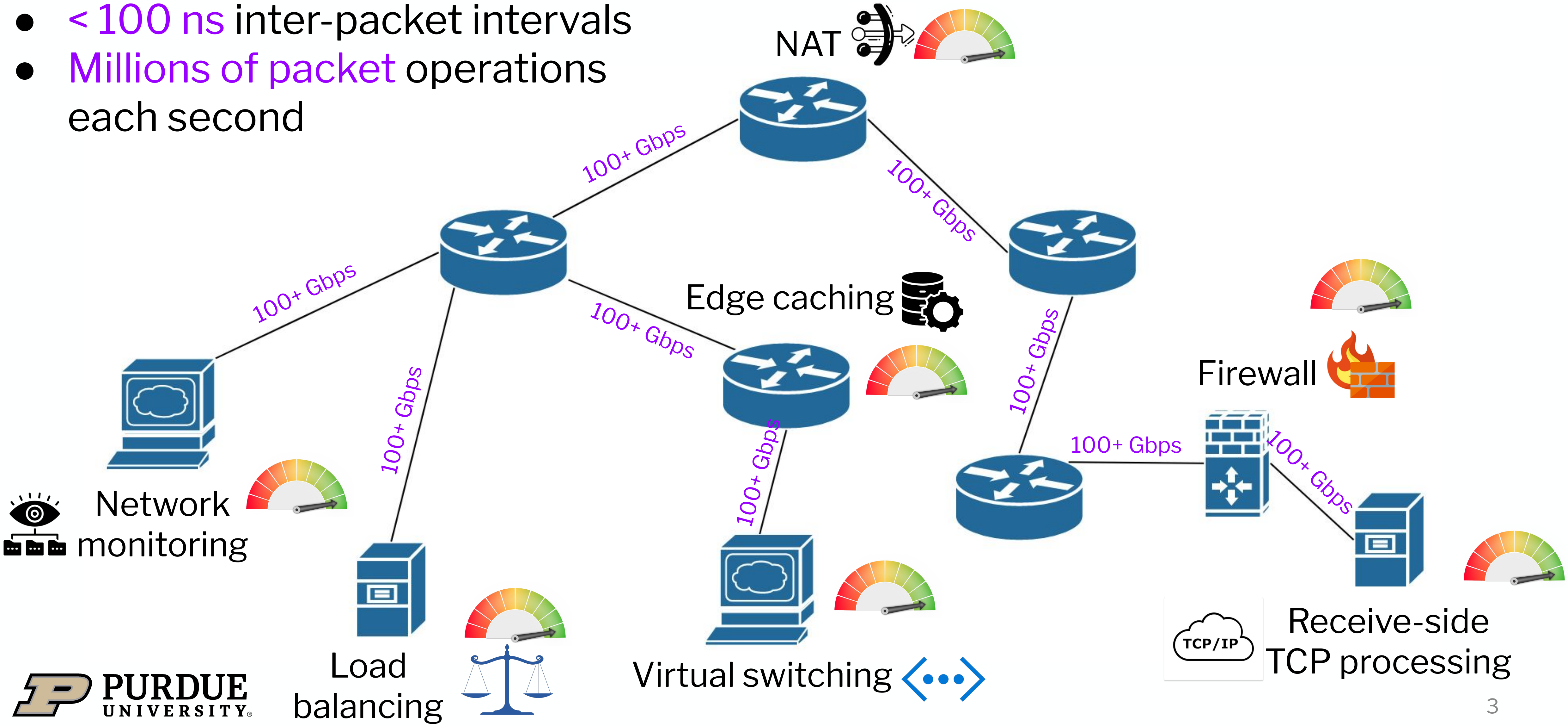
- 300-600 MB tables to serve common network applications [TEA, SIGCOMM '20]



PURDUE UNIVERSITY

Motivation: State-Intensive High-Speed Network Applications

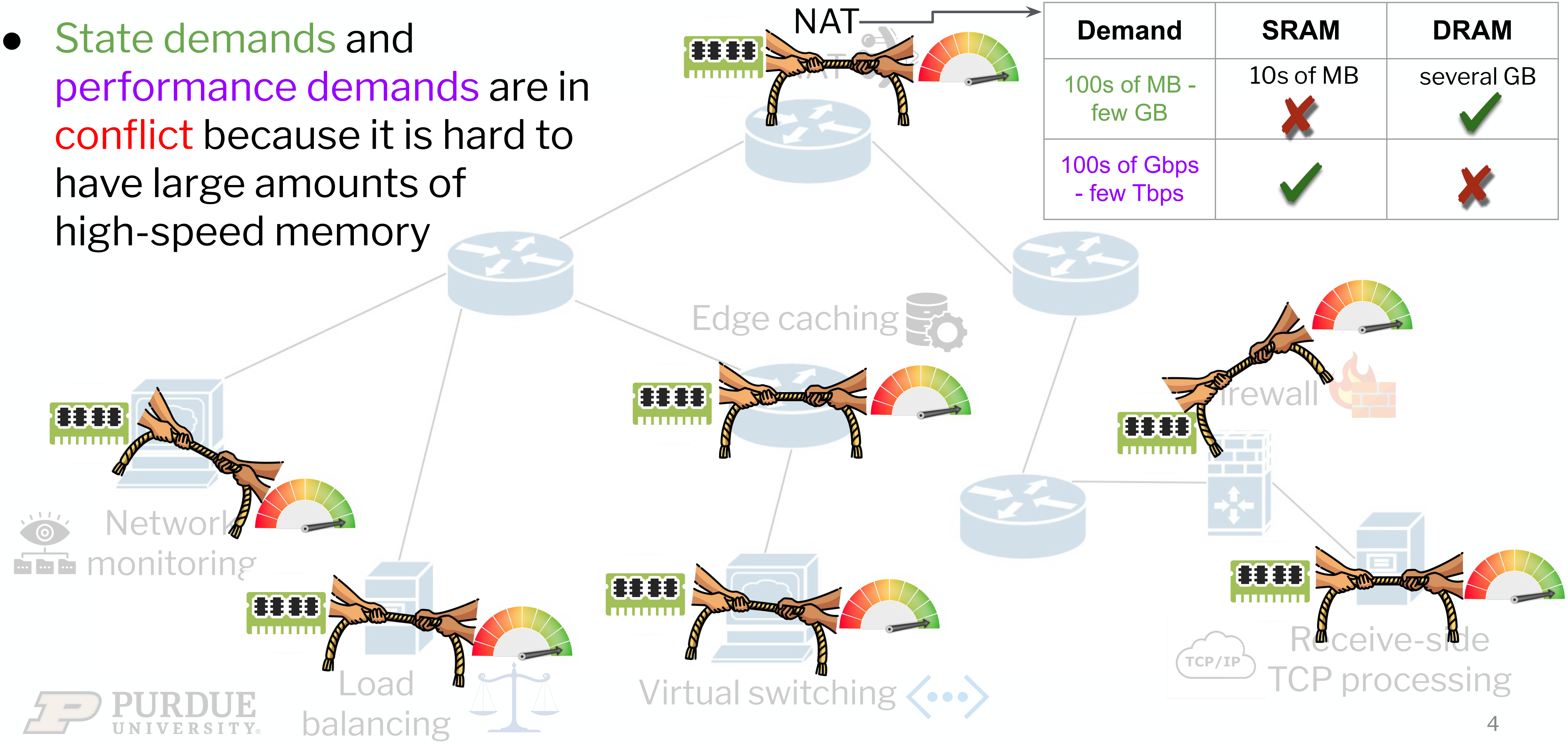
- < 100 ns inter-packet intervals
- Millions of packet operations each second



PURDUE UNIVERSITY

Motivation: State-Intensive High-Speed Network Applications

- State demands and performance demands are in conflict because it is hard to have large amounts of high-speed memory

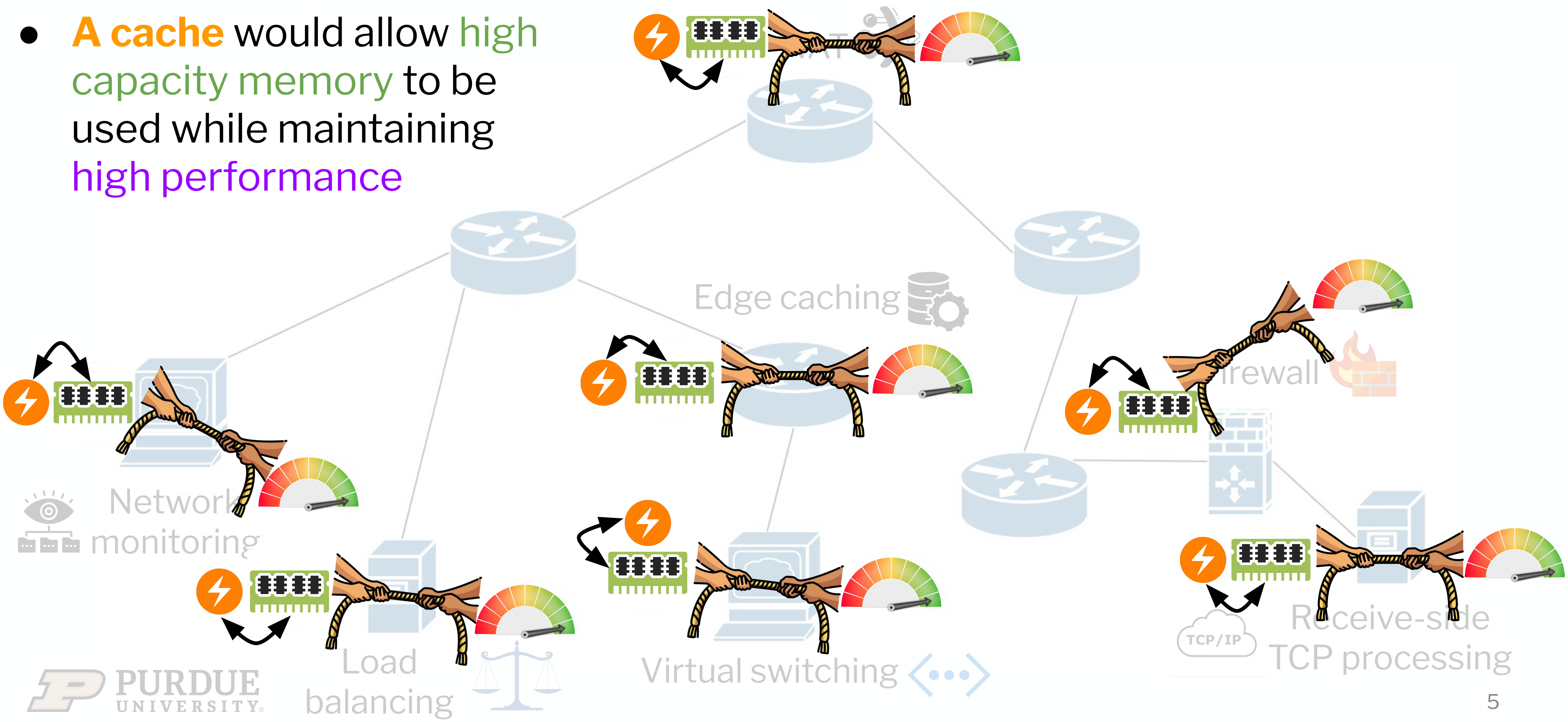


Demand	SRAM	DRAM
100s of MB - few GB	10s of MB ✗	several GB ✓
100s of Gbps - few Tbps	✓	✗

PURDUE UNIVERSITY

Motivation: State-Intensive High-Speed Network Applications

- **A cache** would allow high capacity memory to be used while maintaining high performance



PURDUE UNIVERSITY

Motivation: State-Intensive High-Speed Network Applications

- **A cache** would allow high capacity memory to be us…
  hi…

Takeaway:

High-speed state-intensive network applications require **efficient caching**

NAT

monitoring

Load balancing

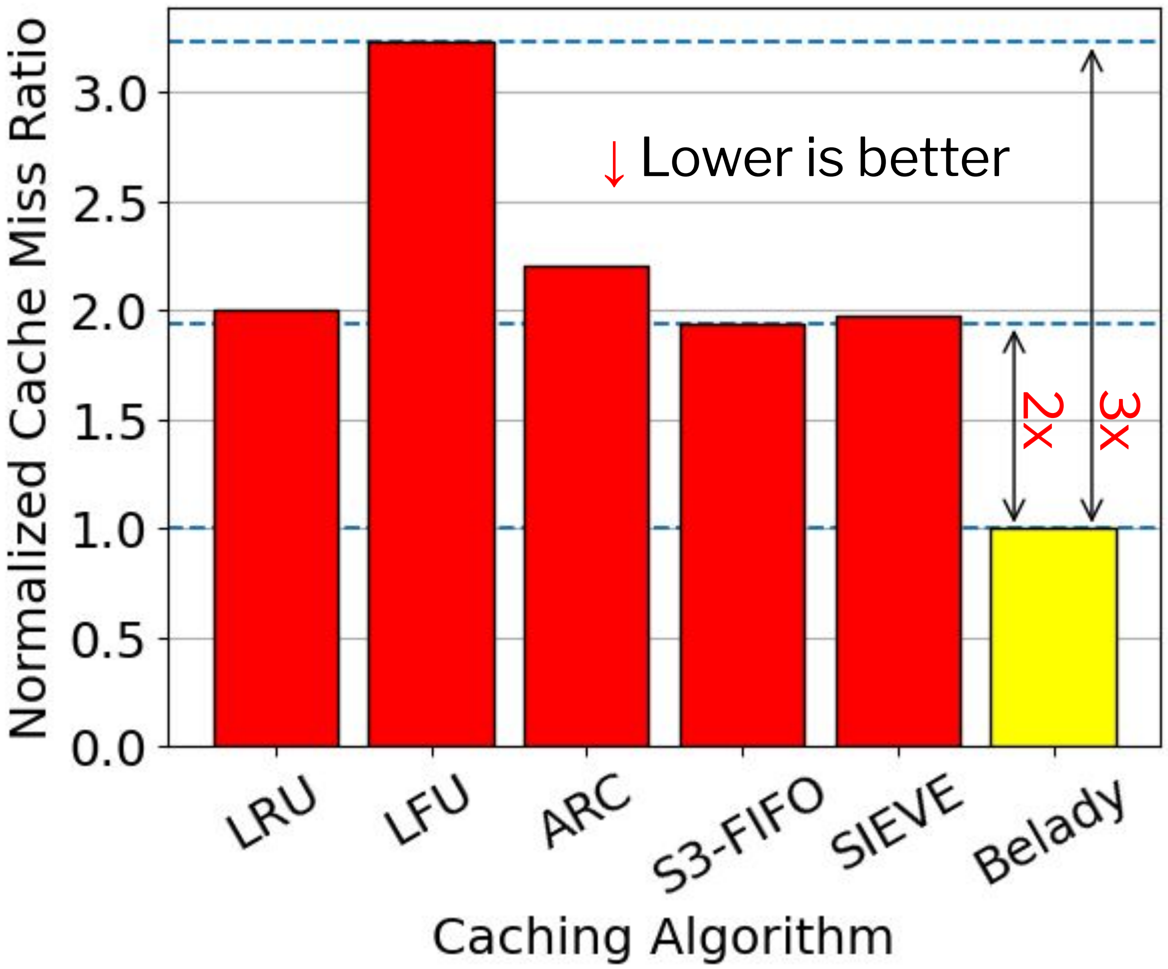
Virtual switching

TCP/IP

Receive-side TCP processing

State of Practice Falls Short of Ideal Case

- No shortage of online caching algorithms - LRU, LFU, ARC, CLOCK, S3-FIFO, SIEVE, etc.
- All fall short of optimal offline caching algorithm (Belady) by a significant margin
  - Ranging from 2-3x higher cache miss ratio

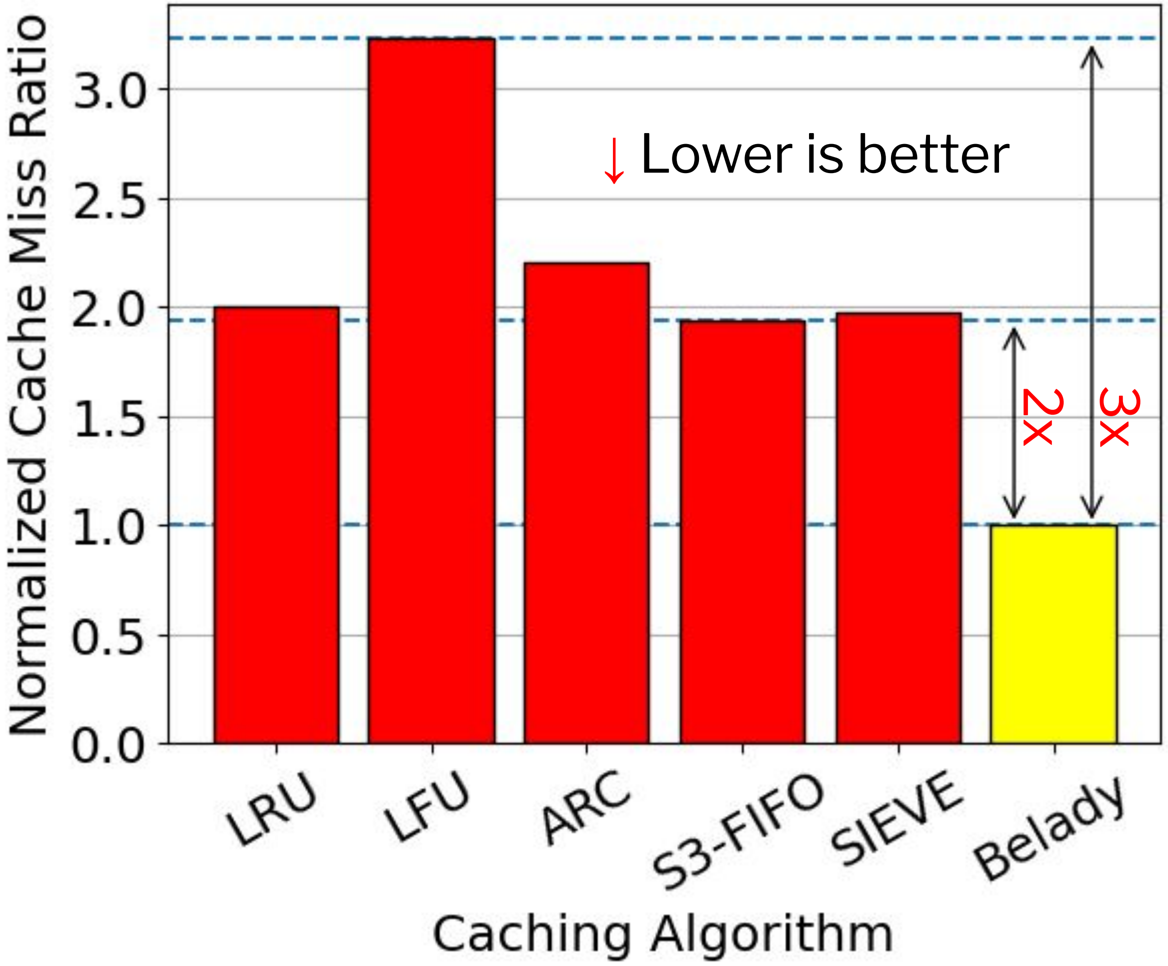


Setup:
2-tier fat-tree network with 144 nodes running in-network load balancing application with websearch workload

PURDUE UNIVERSITY

State of Practice Falls Short of Ideal Case

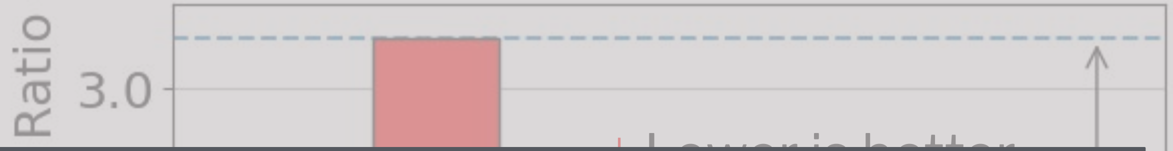
- No shortage of online caching algorithms - LRU, LFU, ARC, CLOCK, S3-FIFO, SIEVE, etc.
- All fall short of optimal offline caching algorithm (Belady) by a significant margin
  - Ranging from 2-3x higher cache miss ratio
- ML-based solutions are prone to mispredictions



Setup:
2-tier fat-tree network with 144 nodes running in-network load balancing application with websearch workload

PURDUE UNIVERSITY

State of Practice Falls Short of Ideal Case



Fundamental Cause of Performance Gap:

Offline algorithm (Belady) uses knowledge of future state accesses to make optimal caching decisions, but ...

Traditional online caching algorithms lack **accurate awareness of future state accesses**

cache miss ratio

Caching Algorithm

Setup:

2-tier fat-tree network with 144 nodes running in-network load balancing application with websearch workload

PURDUE UNIVERSITY

Key Research Question

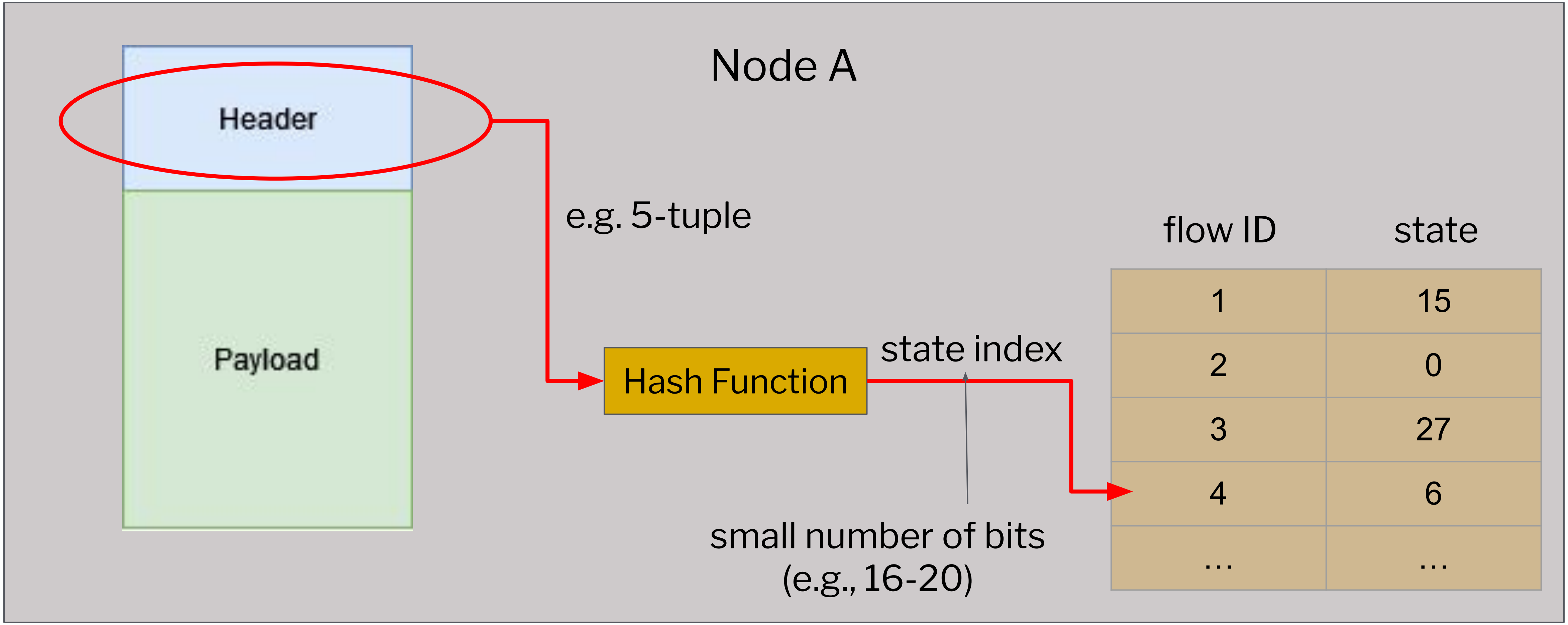
How can **future-aware caching** be realized accurately **in practice (online setting)?**

Key Insight

Traditional online caching assumes future-awareness is challenging.
However…

Networked setting presents unique opportunities to provide very accurate visibility into future state accesses!

Insight 1: Header-Based State Indexing



PURDUE UNIVERSITY

Insight 1: Header-Based State Indexing

Node A

Takeaway 1:

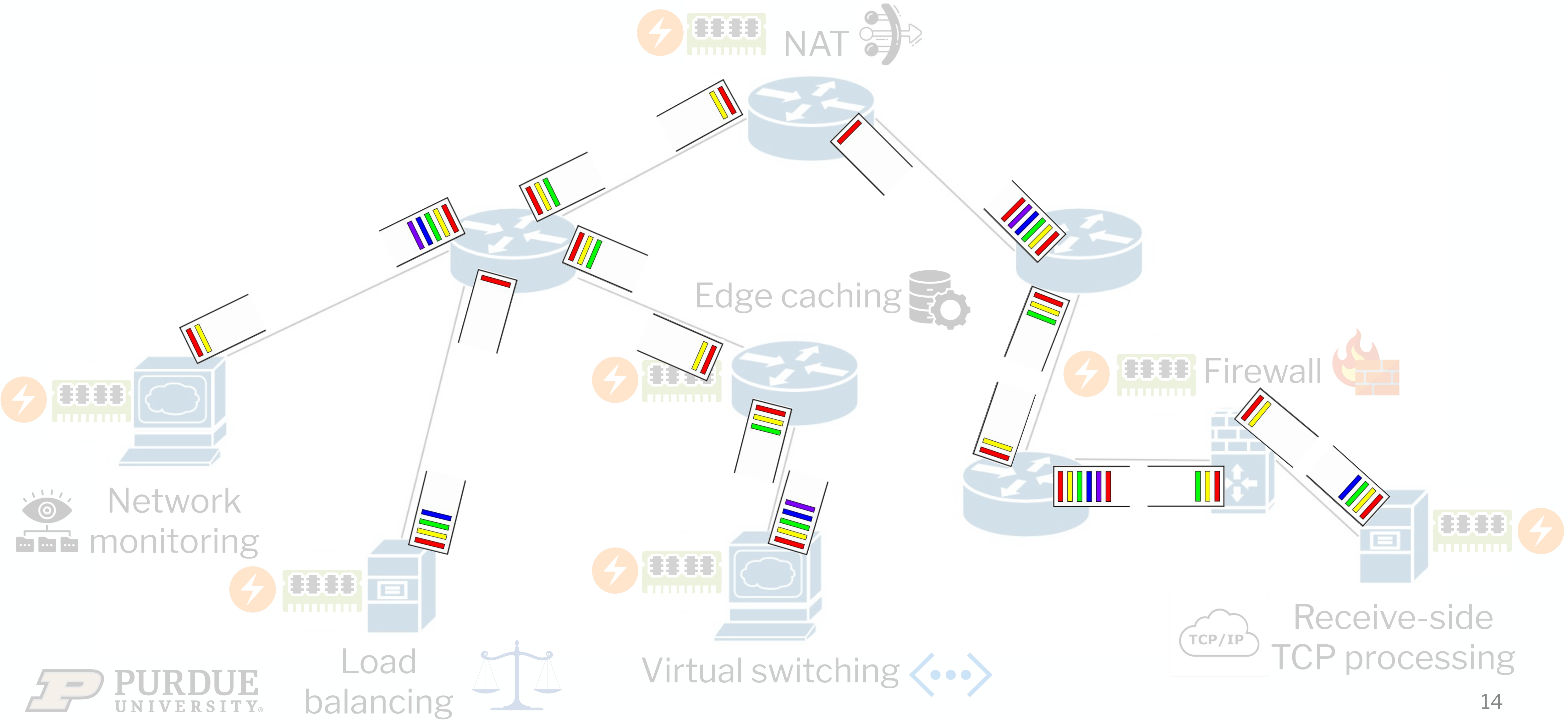
State access indices are carried in incoming packet headers, and can be encoded using **a small number of bits**

(e.g., 16-20)

… …

PURDUE UNIVERSITY

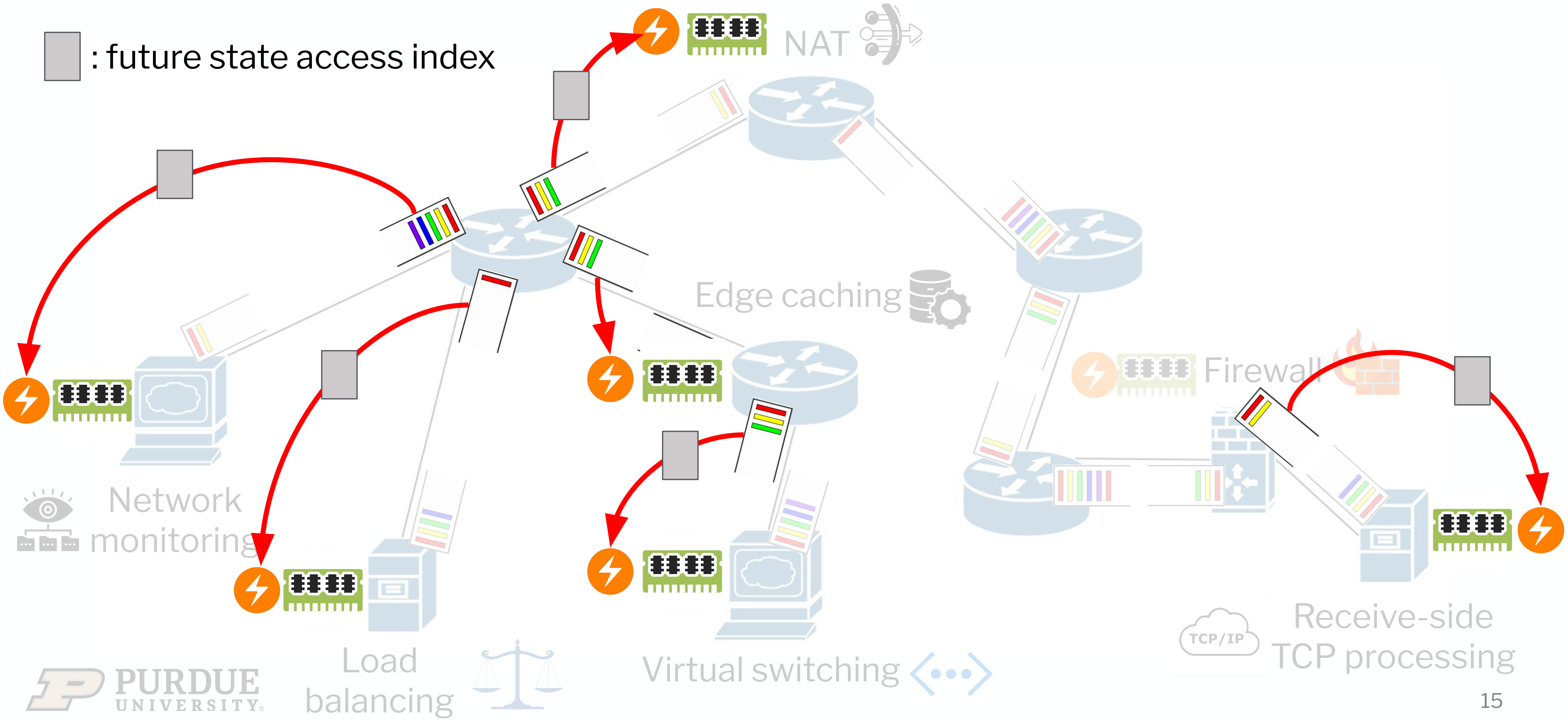
Insight 2: Network Delays Create Opportunities



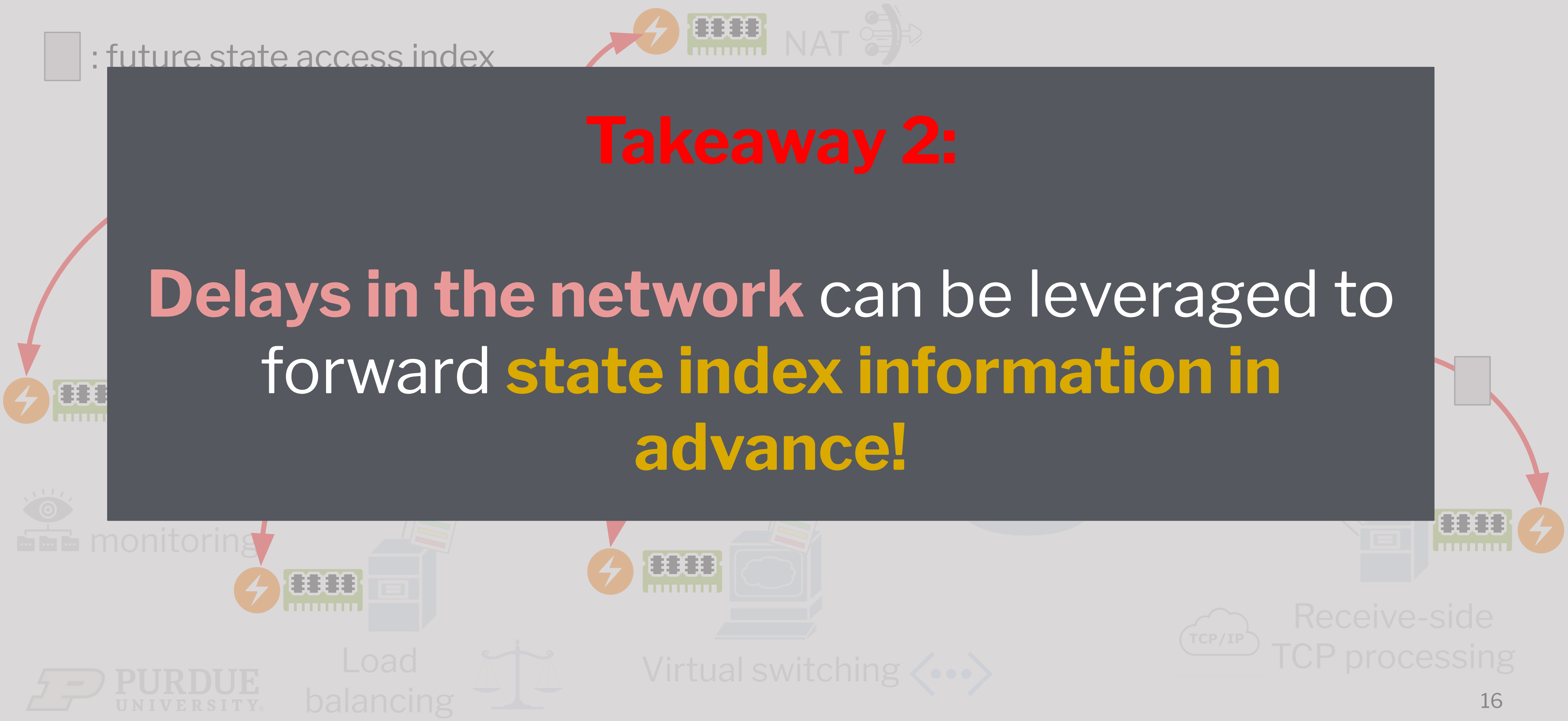
PURDUE UNIVERSITY

Insight 2: Network Delays Create Opportunities

: future state access index



Insight 2: Network Delays Create Opportunities



Takeaway 2:

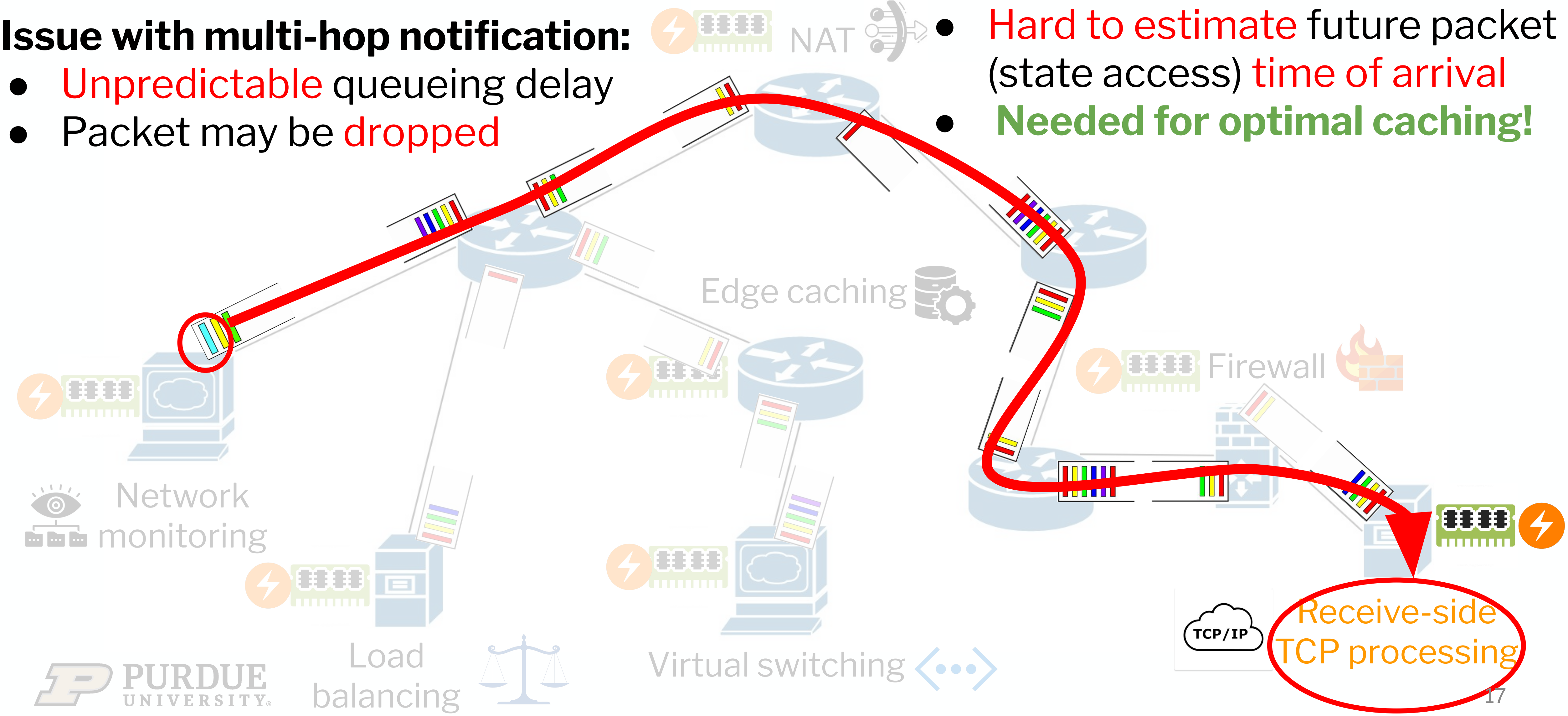
Delays in the network can be leveraged to forward **state index information in advance!**

Insight 3: Neighbors Are Most Accurate Notifiers

Issue with multi-hop notification:

- Unpredictable queueing delay
- Packet may be dropped

- Hard to estimate future packet (state access) time of arrival
- **Needed for optimal caching!**



Insight 3: Neighbors Are Most Accurate Notifiers

Issue with multi-hop notification:

- Unpredictable queueing delay
- Pa

NAT

- Hard to estimate future packet (state access) time of arrival
- Needed for optimal caching!

Takeaway 3:

Multi-hop notification provides **inaccurate estimation** of future state **access time**, but **optimal algorithm (Belady)** heavily relies on it

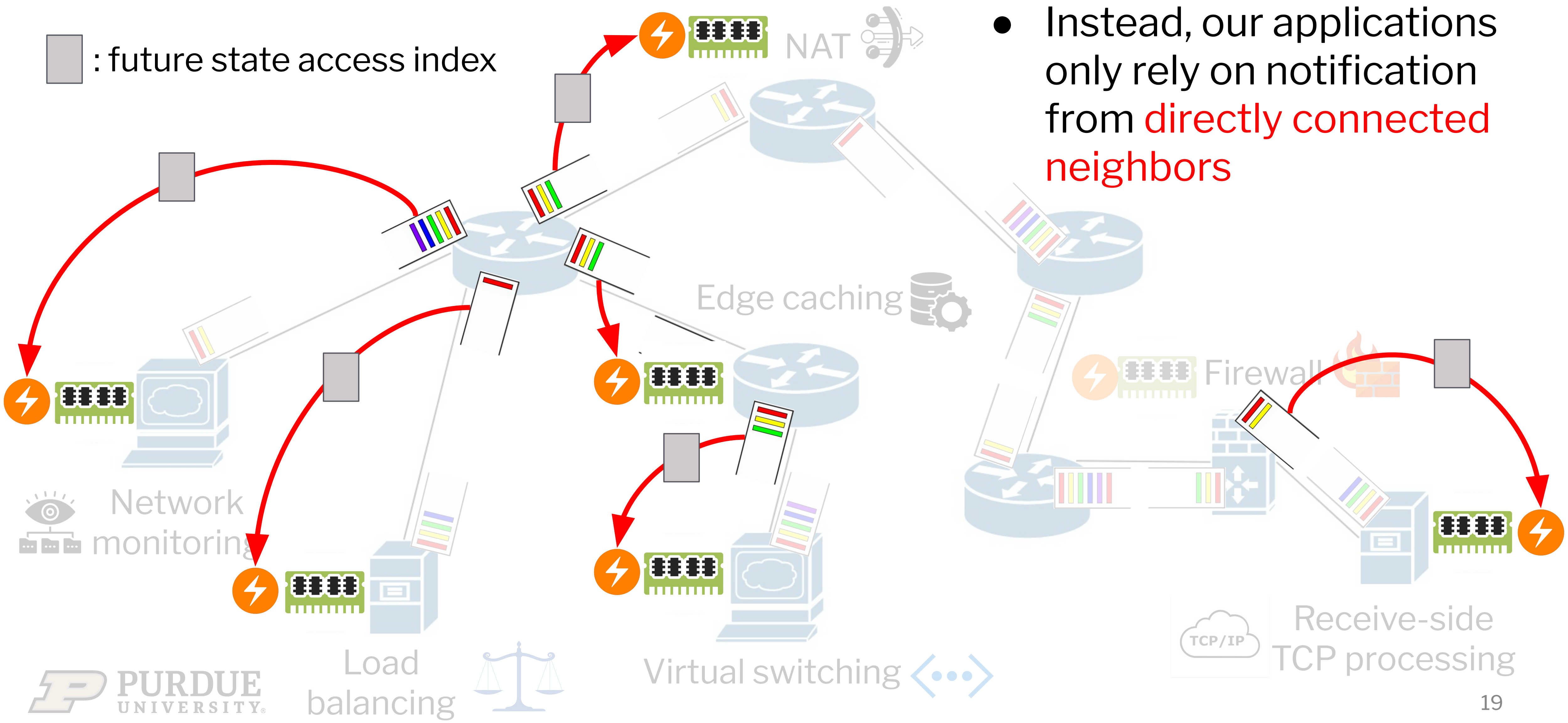
Load balancing

Virtual switching

TCP/IP

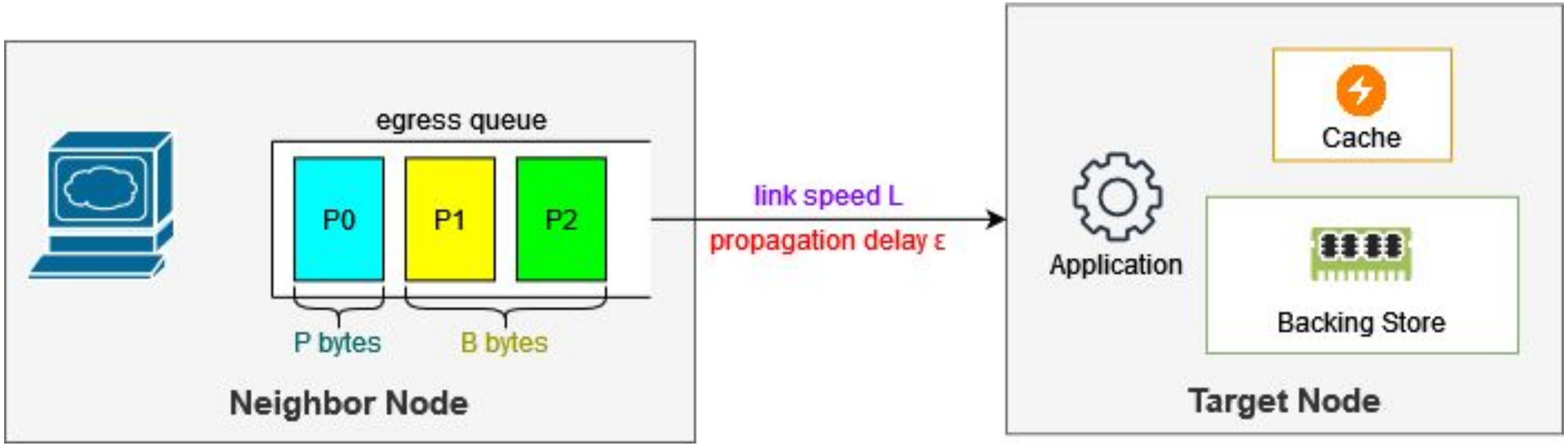
Receive-side TCP processing

Insight 3: Neighbors Are Most Accurate Notifiers

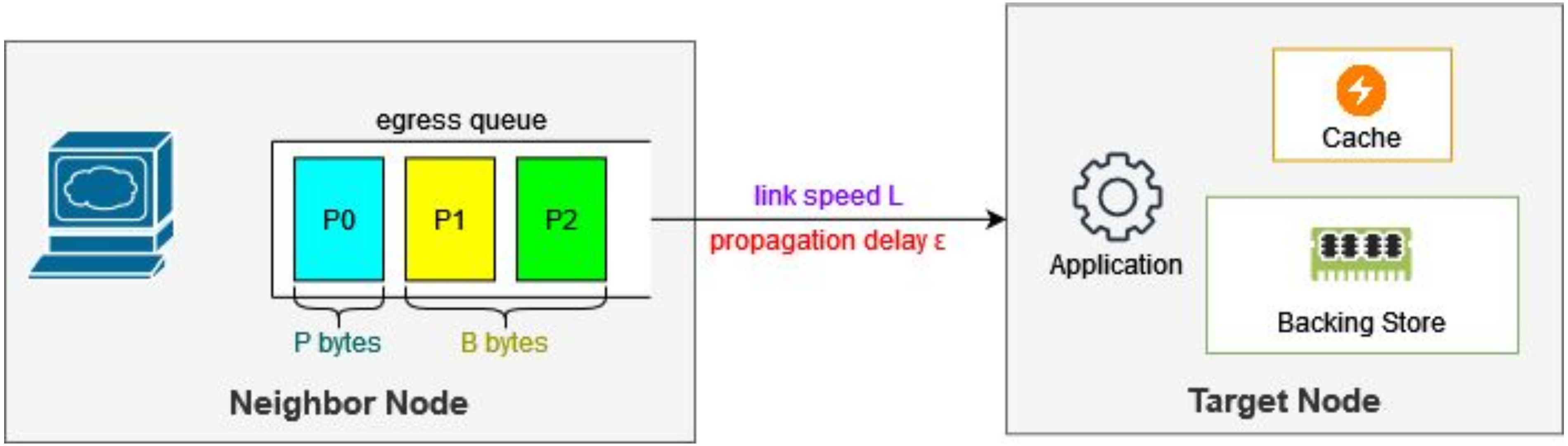


- Instead, our applications only rely on notification from directly connected neighbors

Insight 3: Neighbors Are Most Accurate Notifiers



Insight 3: Neighbors Are Most Accurate Notifiers



$$T = t_0 + (P + B)/L + \varepsilon$$

- **T**: time when packet will reach target node
- t_0: current time at target node
- **P**: size of packet in question P0
- **B**: bytes of queued data in front of packet P0
- **L**: link speed
- **ε**: link propagation delay

PURDUE UNIVERSITY

Insight 3: Neighbors Are Most Accurate Notifiers

Takeaway 4:

Directly connected neighbors provide a perfectly accurate **estimation of future state access times!**

- t_0: current time at target node
- P: size of packet in question P0
- B: bytes of queued data in front of packet P0
- L: link speed
- ε: link propagation delay

PURDUE UNIVERSITY

Putting It All Together

1. **State access indices** are carried in incoming packet headers, and can be encoded using **a small number of bits.**

2. **Delays in the network** can be leveraged to forward state index information **in advance.**

3. **Directly connected neighbors** provide a **perfectly accurate estimate** of future state access times.

Our Contributions

Seer: A Future-Aware Online Caching System

1. Low-Overhead **Future State Access Notification**
2. Future-Aware **Cache Manager**
3. Fast **Hardware Implementation**

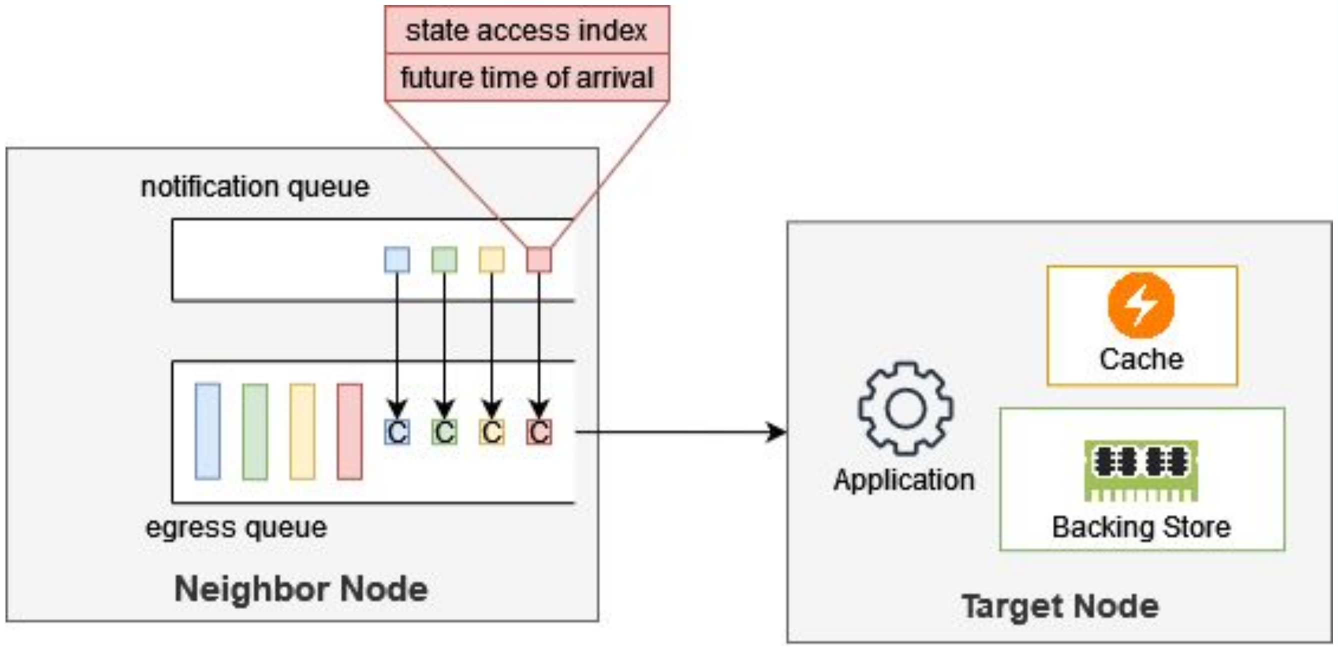
(1) *Low-Overhead Future State Access Notification*

Future State Access Notification for each packet contains:

- State access index
- Future time of arrival of corresponding packet

Naive solution for notification: control packets

- High bandwidth overhead – one control packet per data packet
  - If all pkts are minimum-sized, can consume half of total bandwidth!



State access notifications carried in control packets

(1) *Low-Overhead Future State Access Notification*

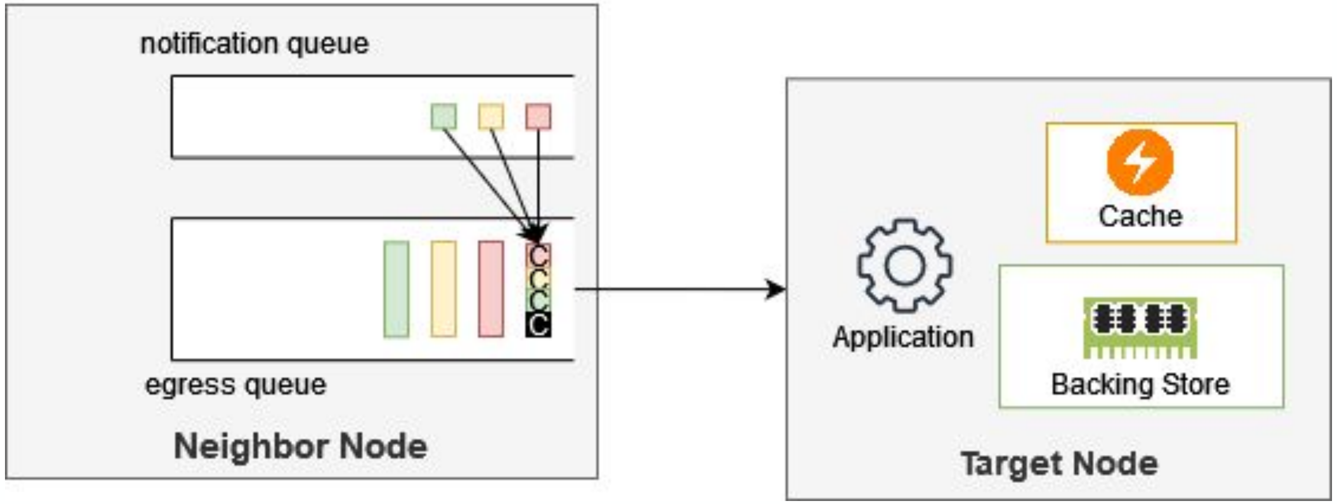
Future State Access Notification for each packet contains:

- State access index
- Future time of arrival of corresponding packet

Naive solution for notification: control packets

- High bandwidth overhead – one control packet per data packet
  - If all pkts are minimum-sized, can consume half of total bandwidth!

Batching reduces number of control packets but can delay notification while it waits for batch



PURDUE UNIVERSITY

(1) Low-Overhead *Future State Access Notification*

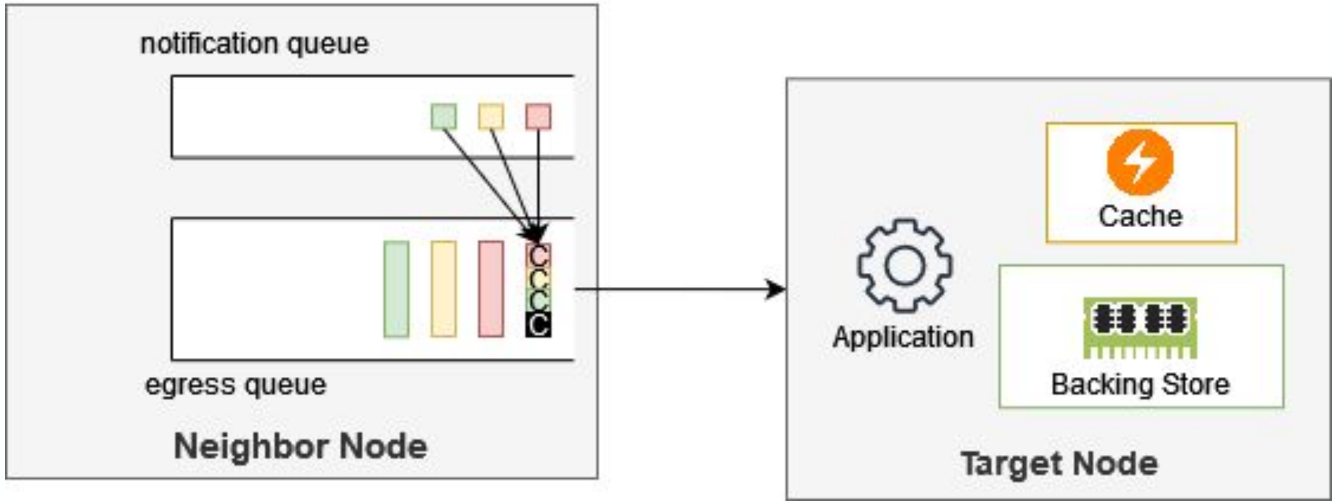
Future State Access Notification for each packet contains:

- State access index
- Future time of arrival of corresponding packet

Naive solution for notification: control packets

- High bandwidth overhead – one control packet per data packet
  - If all pkts are minimum-sized, can consume half of total bandwidth!

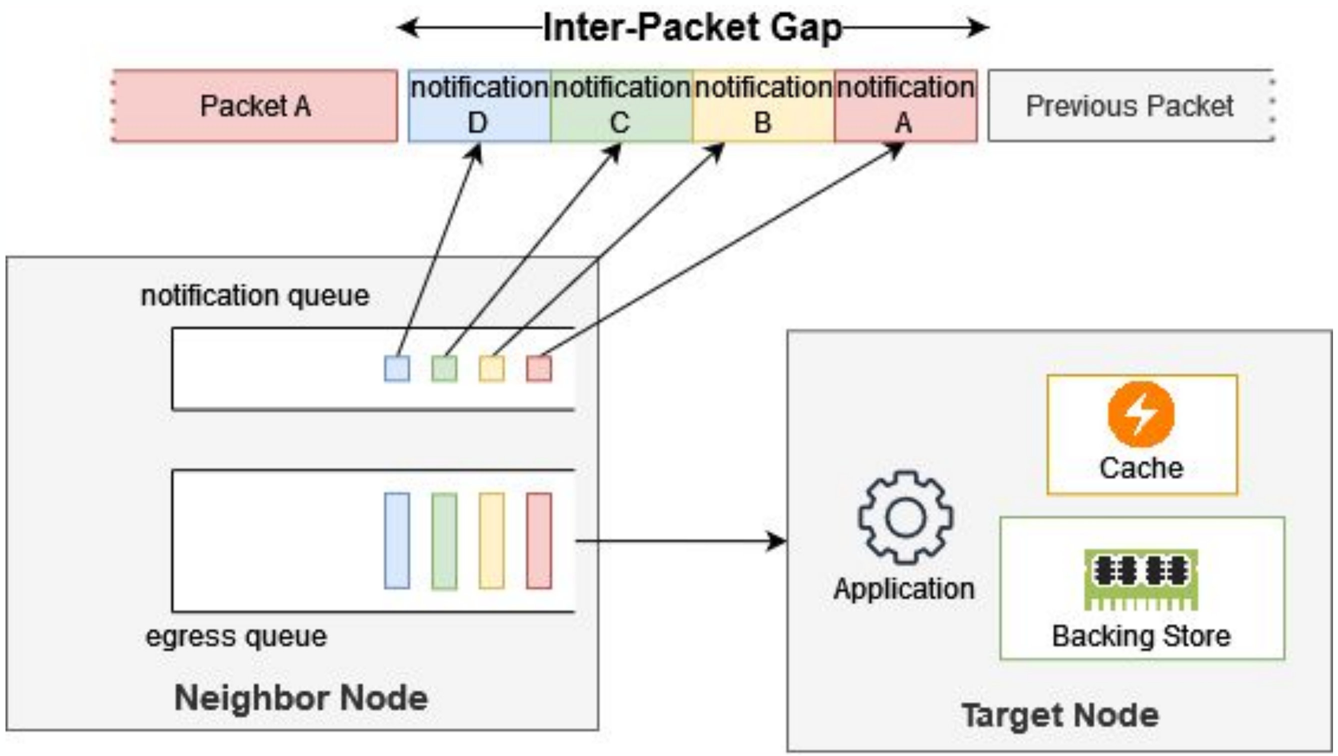
Batching reduces number of control packets but can delay notification while it waits for batch



How to send future state notifications in a timely manner and with low overhead?

(1) Low-Overhead Future State Access Notification

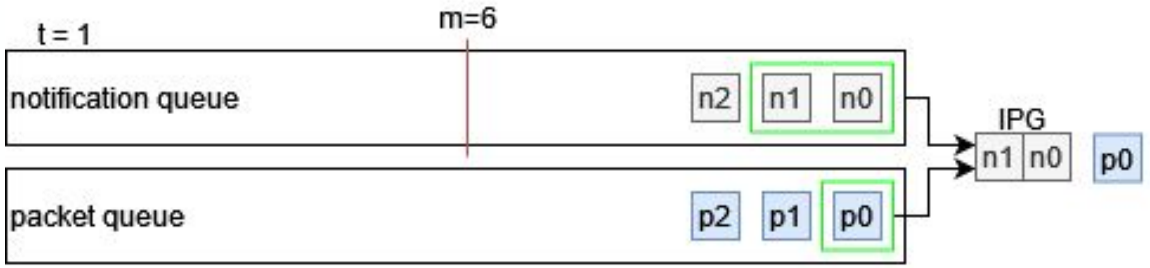
- **Send notifications in IPG**
  - Ethernet PHY enforces a minimum of 96 bit **inter-packet gap (IPG)** between packets
  - Can carry multiple packets' state access notification within a single IPG
  - **Zero bandwidth overhead**
  - **Limitation**: Limited # of bits for communication
    - Limits rate at which packet notifications can be sent



Send notifications using IPG between packets

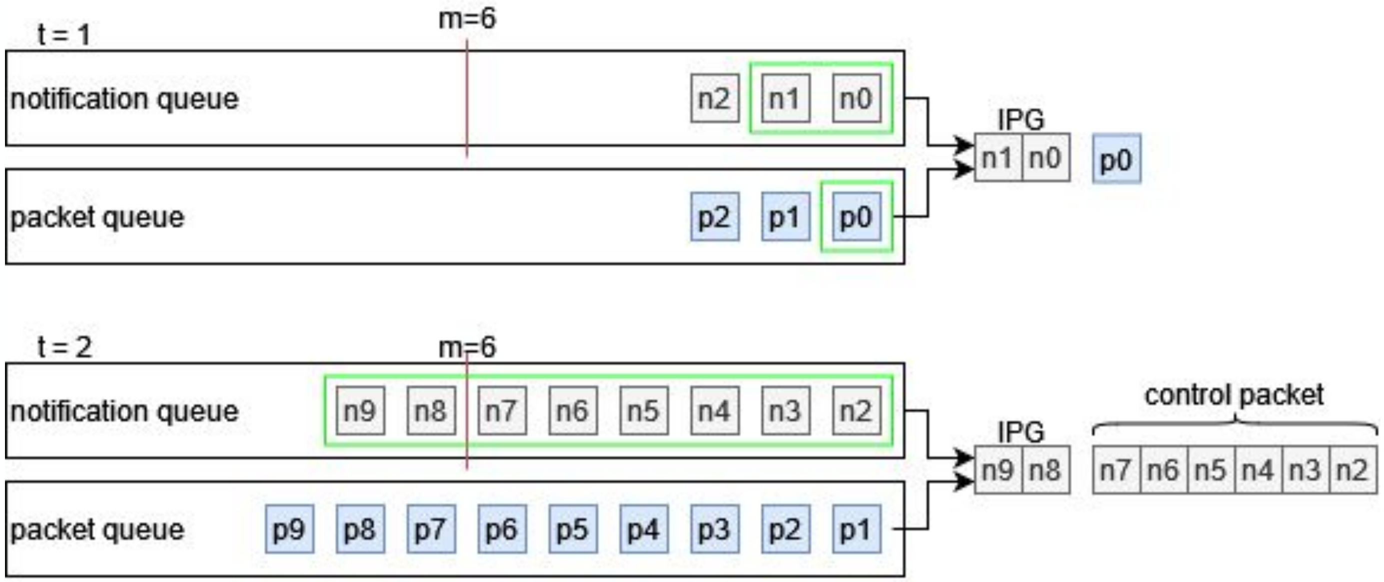
(1) *Optimization: Opportunistic Batching*

- Send notifications over IPG under normal scenarios
- Send a control packet when notification queue exceeds configurable parameter ***m***



(1) Optimization: Opportunistic Batching

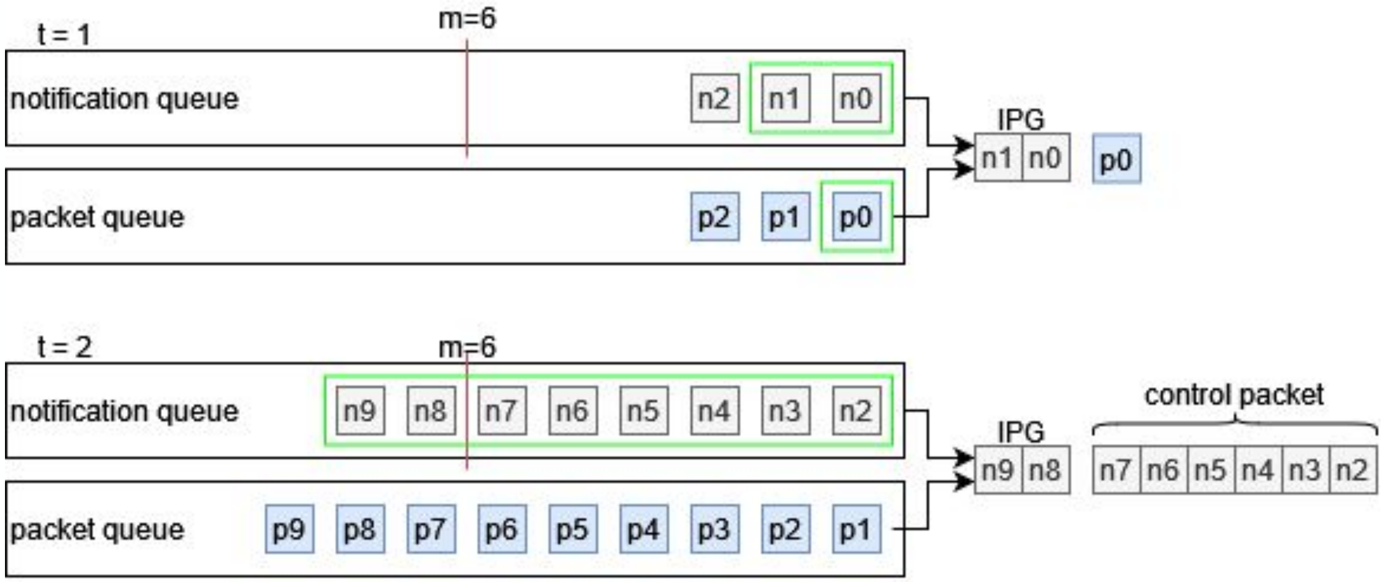
- Send notifications over IPG under normal scenarios
- Send a control packet when notification queue exceeds configurable parameter ***m***
  - By configuring *m*, we control bandwidth overhead of control pkts



PURDUE UNIVERSITY

(1) Optimization: Opportunistic Batching

- Send notifications over IPG under normal scenarios
- Send a control packet when notification queue exceeds configurable parameter ***m***
  - By configuring *m*, we control bandwidth overhead of control pkts



Best of both worlds:

Timely notification with low bandwidth overhead and no batching delay

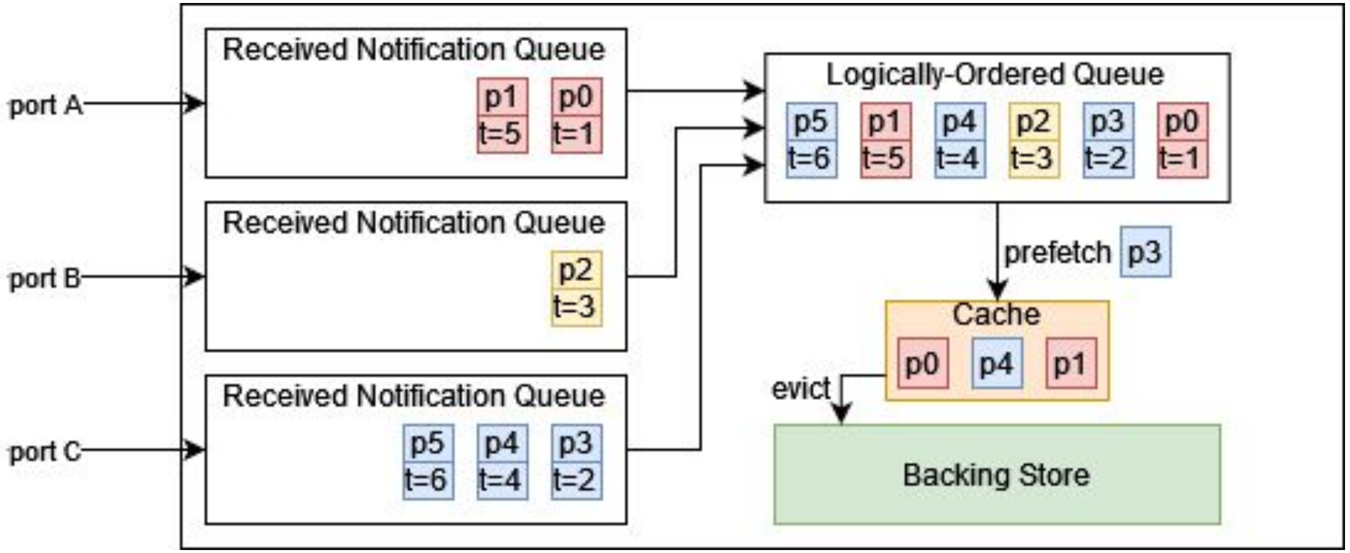
PURDUE UNIVERSITY

(2) *Future-Aware Cache Manager*

- Cache manager uses received future state access notifications to make smarter **prefetching** and cache **eviction** decisions
- Cache manager consists of two components:
    - Future-Aware **Prefetching**
    - Future-Aware **Cache Eviction**

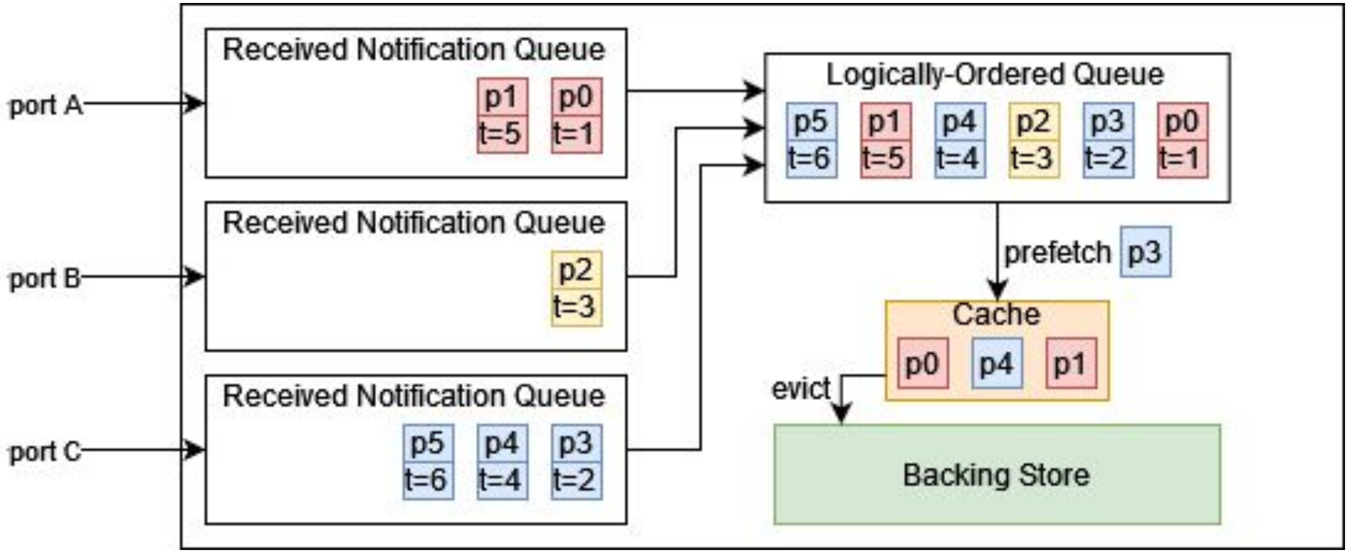
(2) *Future-Aware Prefetching*

- **Goal:** Fetch state in order of **predicted time of access**
  - One received notification queue per input port
  - Combine into **one logically sorted queue** based on future access time
  - Fetch soonest state not in cache



(2) *Future-Aware Prefetching*

- **Goal:** Fetch state in order of **predicted time of access**
  - One received notification queue per input port
  - Combine into **one logically sorted queue** based on future access time
  - Fetch soonest state not in cache



(3) ***Fast Hardware Implementation***

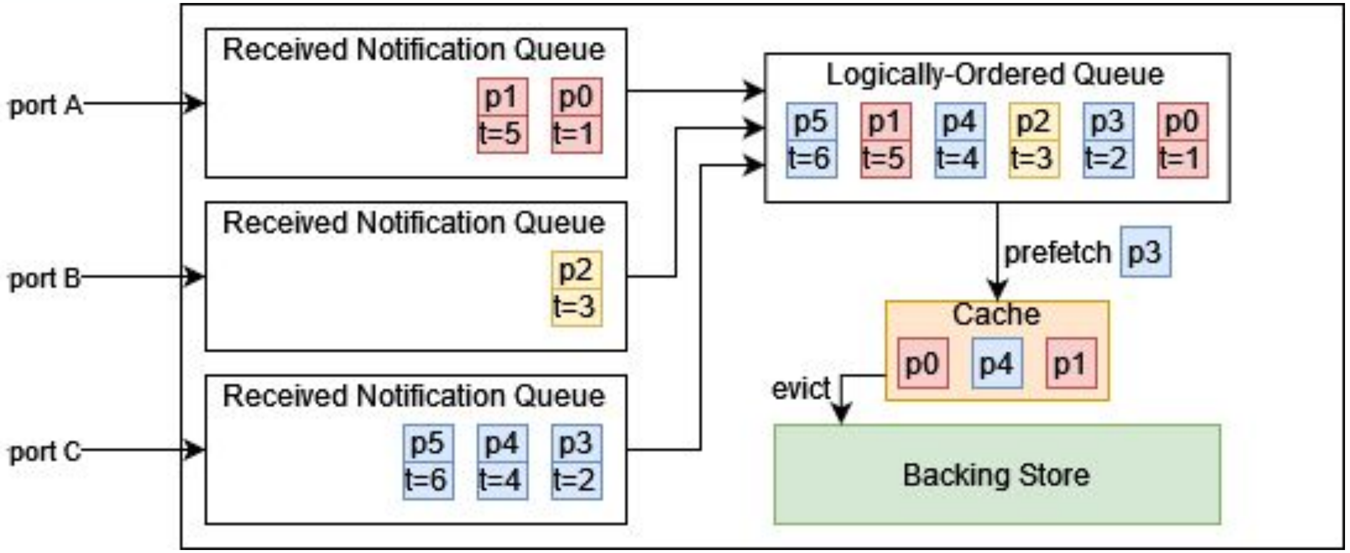
$$t_{prefetch} = 1 + \log(P) + k \text{ clock cycles}$$

P: number of ports

k: cache set size

(2) *Future-Aware Prefetching*

- **Goal:** Fetch state in order of **predicted time of access**
  - One received notification queue per input port
  - Combine into **one logically sorted queue** based on future access time
  - Fetch soonest state not in cache
  - If cache is full → **eviction algorithm**



(3) *Fast Hardware Implementation*

$$t_{prefetch} = 1 + \log(P) + k \text{ clock cycles}$$

P: number of ports

k: cache set size

PURDUE UNIVERSITY

(2) *Future-Aware Cache Eviction*

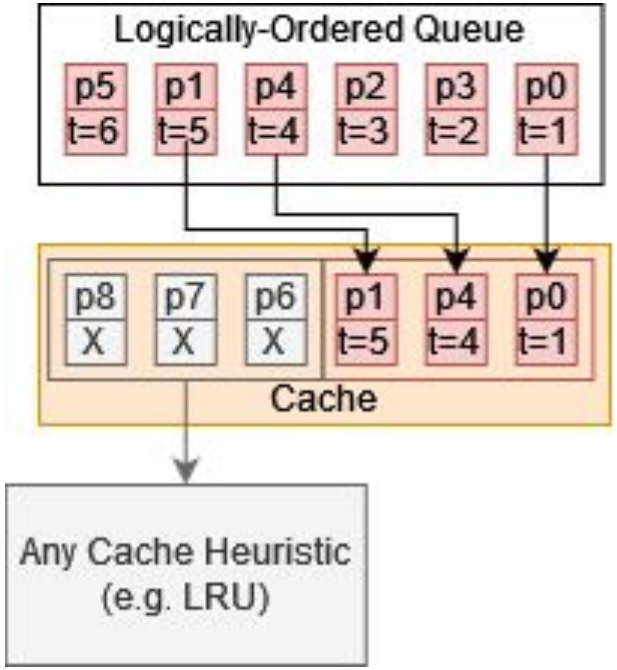
- **Goal:** Emulate **Belady's algorithm** as closely as possible:
  - Evict an entry that will be accessed furthest in the future

- **Challenge:** Knowledge of only a **partial set** of future state accesses

PURDUE UNIVERSITY

(2) *Future-Aware Cache Eviction*

- **Our solution:**
  - Split cache into two sets: objects with known access time vs unknown access time
  - Prioritize evicting objects with unknown access time using ***any*** cache heuristic

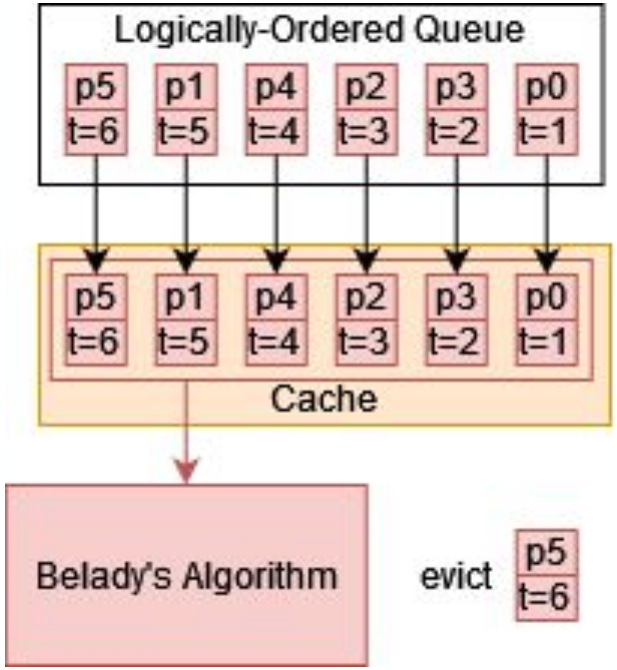


- **Bounded performance**
  - Worst case: **Caching heuristic**

PURDUE UNIVERSITY

(2) Future-Aware Cache Eviction

- **Our solution:**
  - Split cache into two sets: objects with known access time vs unknown access time
  - Prioritize evicting objects with unknown access time using ***any*** cache heuristic
  - When cache solely contains objects with known access time, evict according to Belady's algorithm

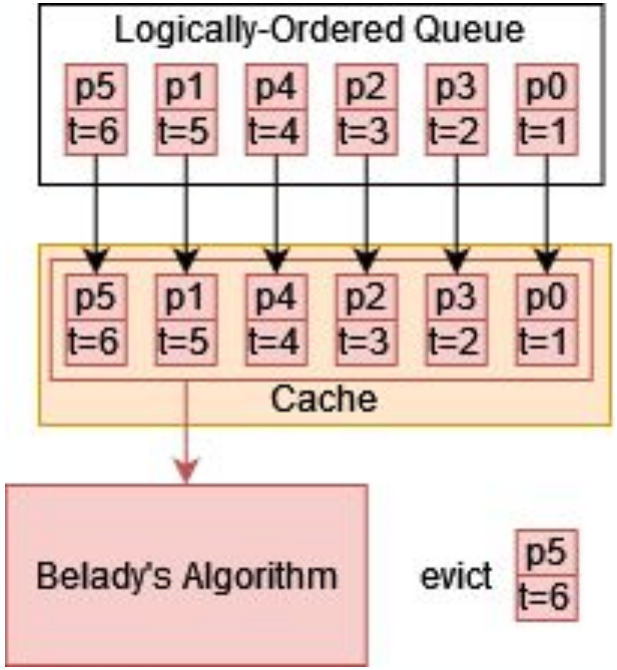


- **Bounded performance**
  - Worst case: **Caching heuristic**
  - Best case: **Belady's algorithm**

PURDUE UNIVERSITY

(2) Future-Aware Cache Eviction

- **Our solution:**
  - Split cache into two sets: objects with known access time vs unknown access time
  - Prioritize evicting objects with unknown access time using ***any*** cache heuristic
  - When cache solely contains objects with known access time, evict according to Belady's algorithm



- **Bounded performance**
  - Worst case: **Caching heuristic**
  - Best case: **Belady's algorithm**

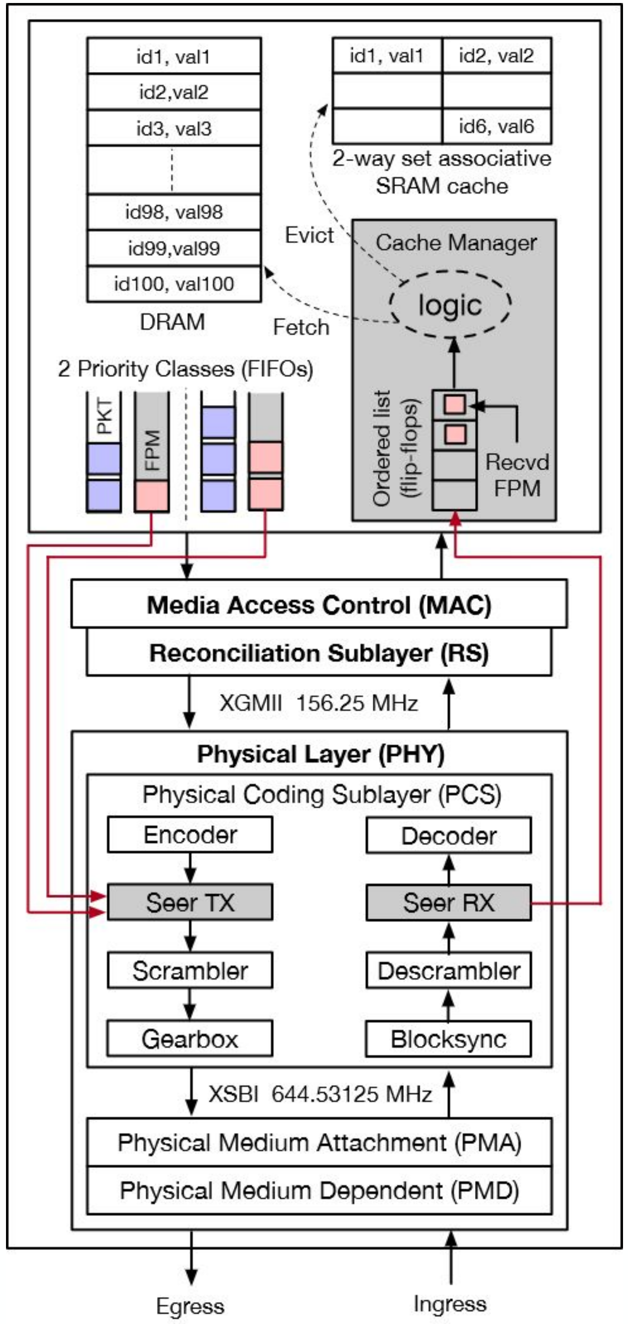
*(3) **Fast Hardware Implementation***

$$t_{evict} = \text{k clock cycles}$$

k: cache set size

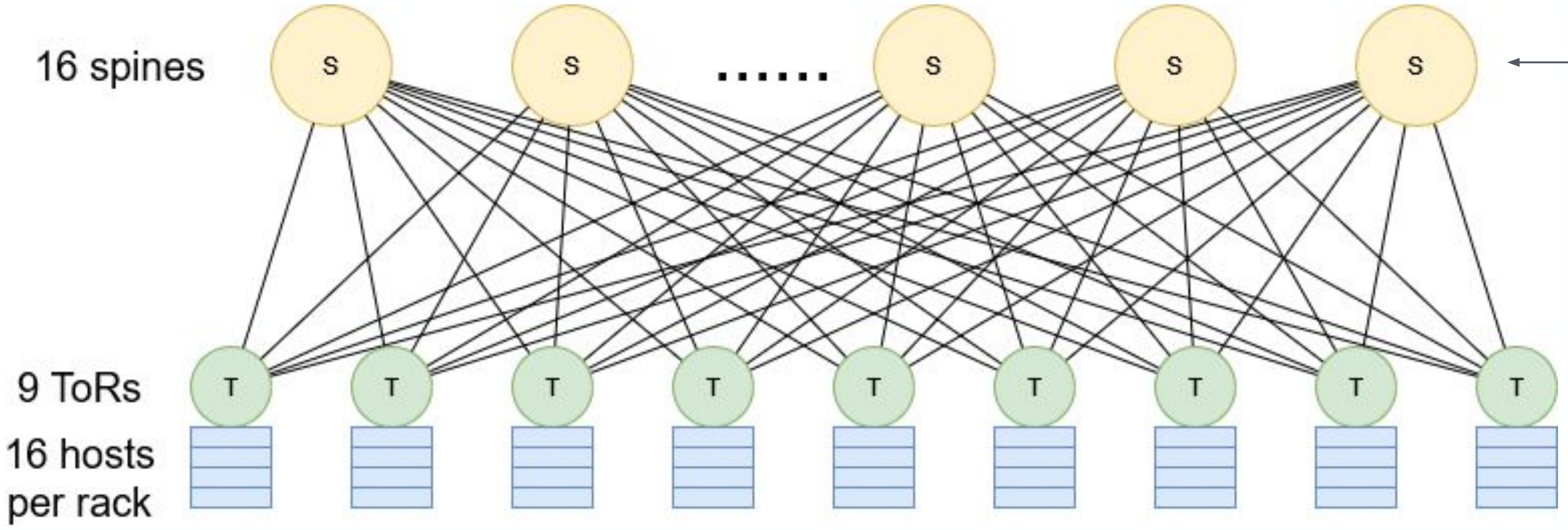
Prototype

- **FPGA prototype**
  - Altera Stratix V FPGA: 234 K adaptive logic modules, 52 Mbits SRAM, four 10 Gbps network ports
- **Seer modifies Ethernet physical layer (PHY) to access IPG**
  - Replaces default idle 0 values in IPG with state access notification



PURDUE UNIVERSITY

Evaluation Setup



Applications:

L4 Load Balancing
(per connection state)

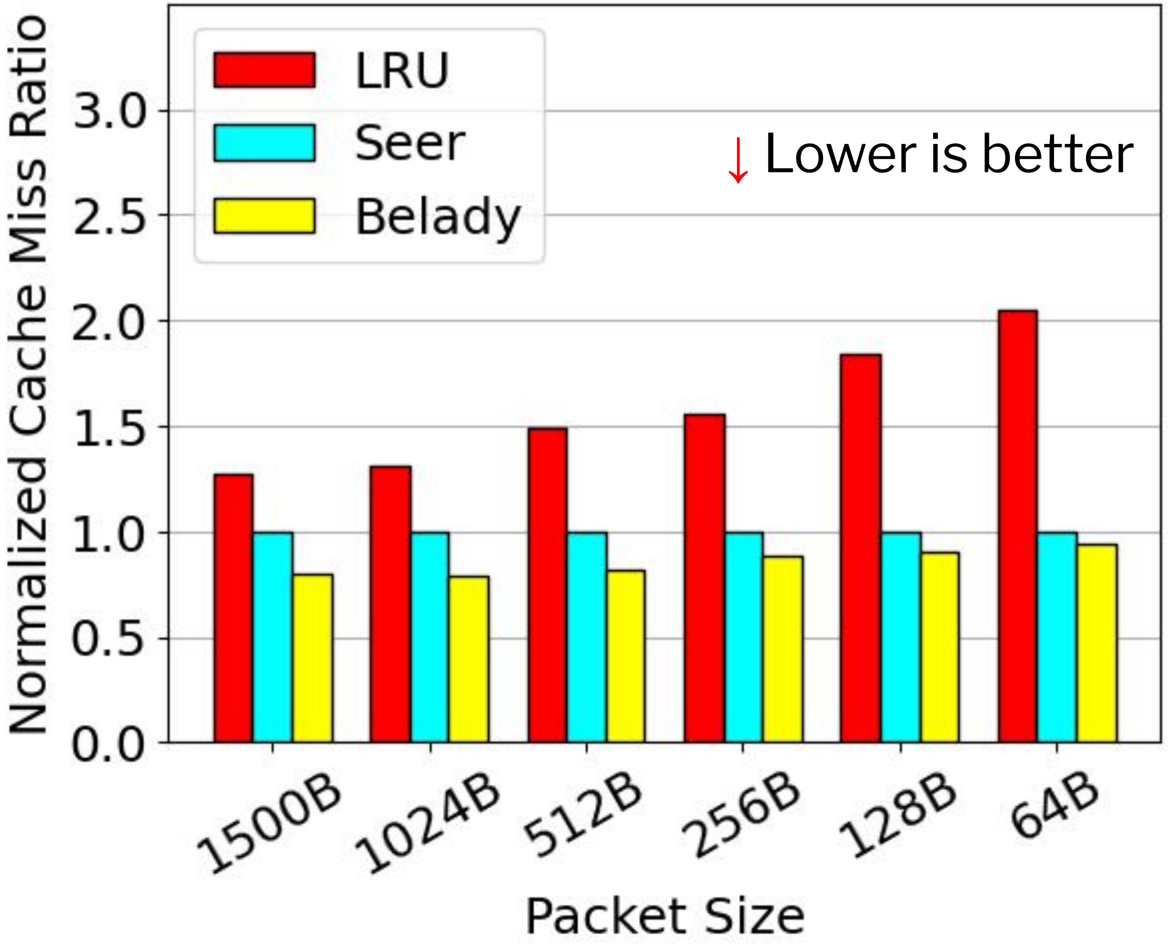
Intrusion Detection
(per flow state)

Packet-level simulator in C		
Packet-level simulator in C • Two-tier Fattree topology ○ 16 spine switches ○ 9 racks ○ 16 hosts / rack (total 144) ○ Full bisection bandwidth	• 100 Gbps links • 100 ns per-hop propagation delay • 100 ns backing memory access latency • 96 bit inter-packet gap	• DCTCP congestion control • ECMP load balancing • Switches support ECN • **Evaluation metric:** ○ **Cache miss ratio**

PURDUE UNIVERSITY

Evaluation: Good Case for Seer

- **Incast** Traffic Pattern:
  - Incast traffic results in most queueing at neighbor node
  - Provides furthest visibility into future state accesses
- Seer remains within 7-20% of Belady
- Seer performs 20-100% better than LRU

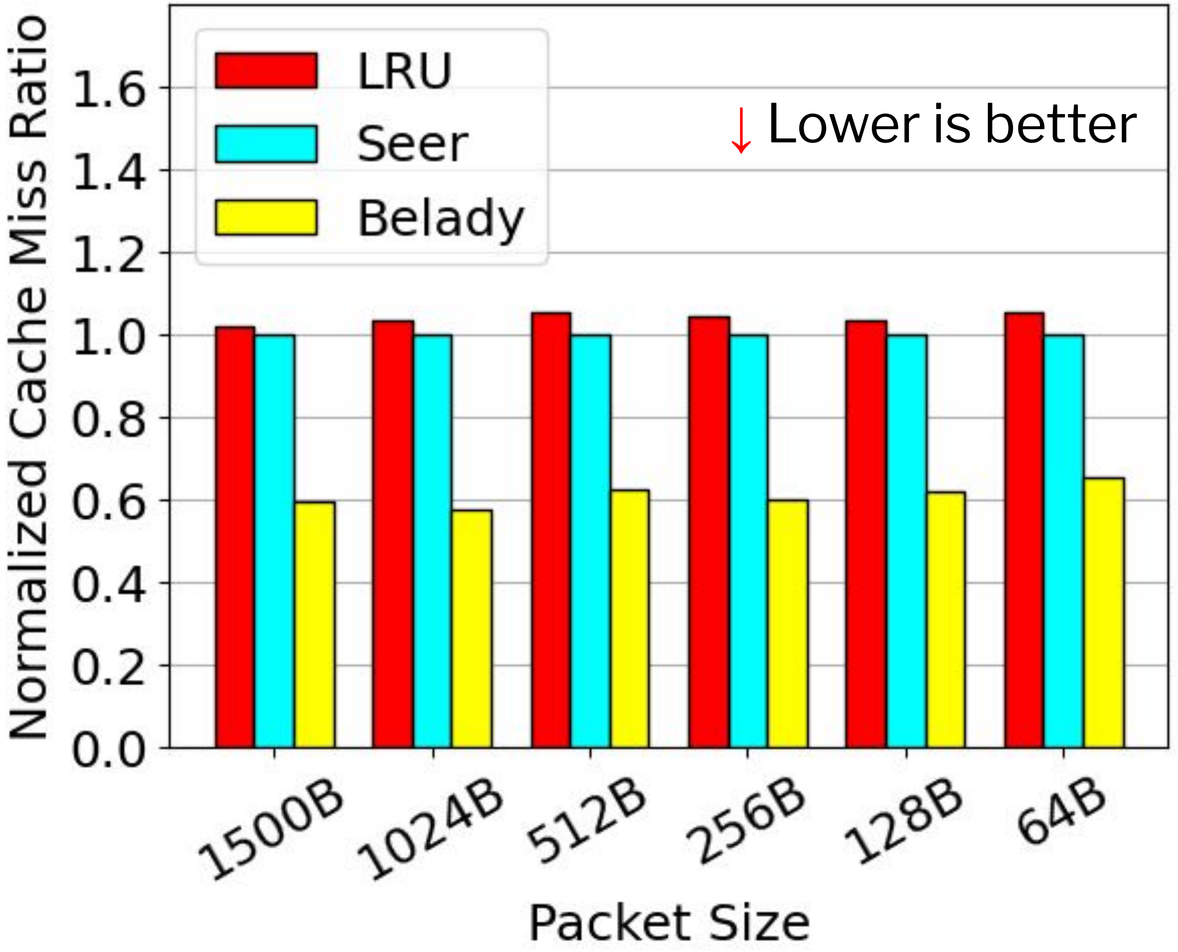


Performance for each packet size normalized w.r.t. corresponding Seer performance

PURDUE UNIVERSITY

Evaluation: Bad Case for Seer

- **Permutation** Traffic Pattern:
  - Permutation traffic over full bisection bandwidth fattree network results in least queueing at neighbor node
  - Provides least visibility into future state accesses
- Seer remains within 35-40% of Belady
- Seer performs 2-5% better than LRU



Performance for each packet size normalized w.r.t. corresponding Seer performance

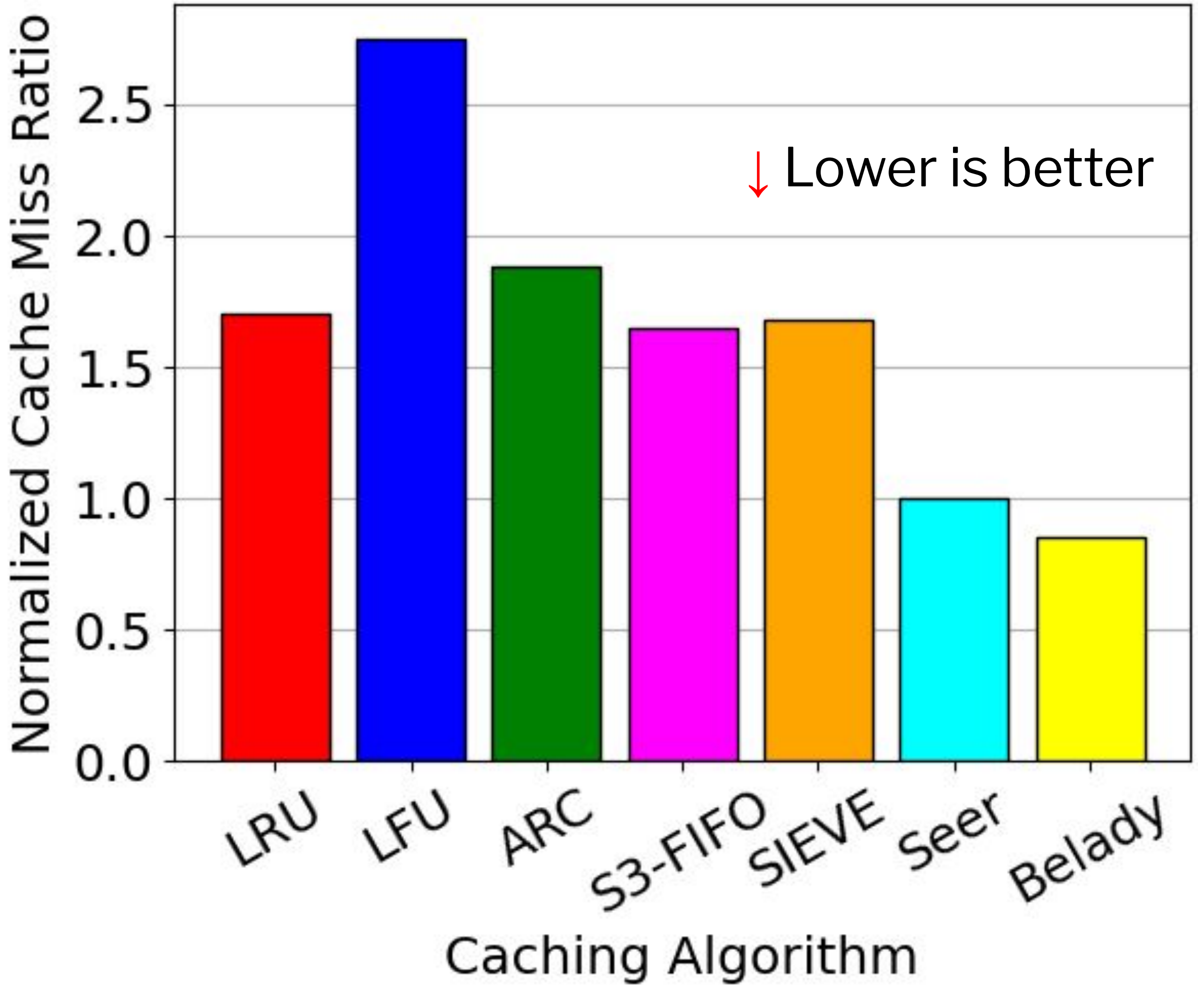
PURDUE UNIVERSITY

Evaluation: Realistic Workload

- **Websearch** workload
  - Representative of datacenter workload
  - Heavy-tailed flow size distribution

- 60-180% lower cache miss ratio for Seer compared to state-of-art

- **Flow completion time (FCT) show similar trend:**
  - Seer reduces FCT by 25-75% compared to LRU



Normalized w.r.t. Seer

Conclusion

- Seer enables future-aware online caching in a networked system

- Seer makes three key technical contributions:
  - Low-Overhead Protocol for Future State Access Notification
  - Design of Future-Aware Cache Manager
  - Fast Hardware Implementation

- Seer performs close to optimal offline caching in practice, with worst case performance bounded by state-of-the-art caching heuristic

Thank You

Any questions?

PURDUE UNIVERSITY

nsdi '24