Reinforcement Learning for TCP Congestion Control

Viswanath Sivakumar

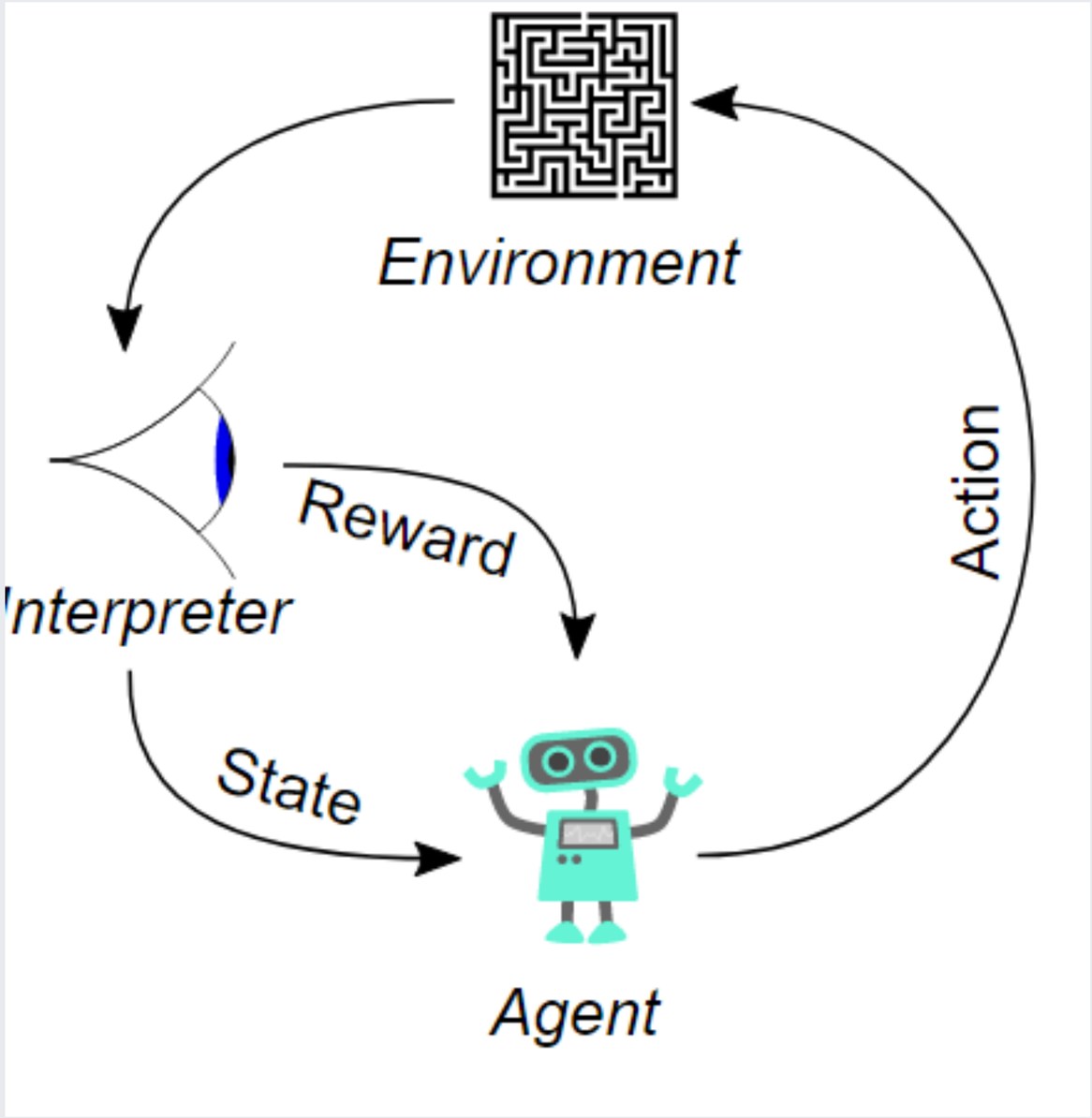
Facebook AI Research (FAIR)

Who am I?

Researcher at FAIR London

Machine Learning and Systems

- Reinforcement Learning
- Computer Vision, OCR



Who am I?

Researcher at FAIR London

Machine Learning and Systems

- Reinforcement Learning
- Computer Vision, OCR

Past: Traffic Infra @ FB:

- L7 Load Balancers, CDN, Edge termination, etc.
- Low-latency live video-streaming
- Several C++ networking libs:
  - https://github.com/facebook/proxygen
  - https://github.com/facebook/wangle
  - https://github.com/facebook/folly

Who am I?

Researcher at FAIR London

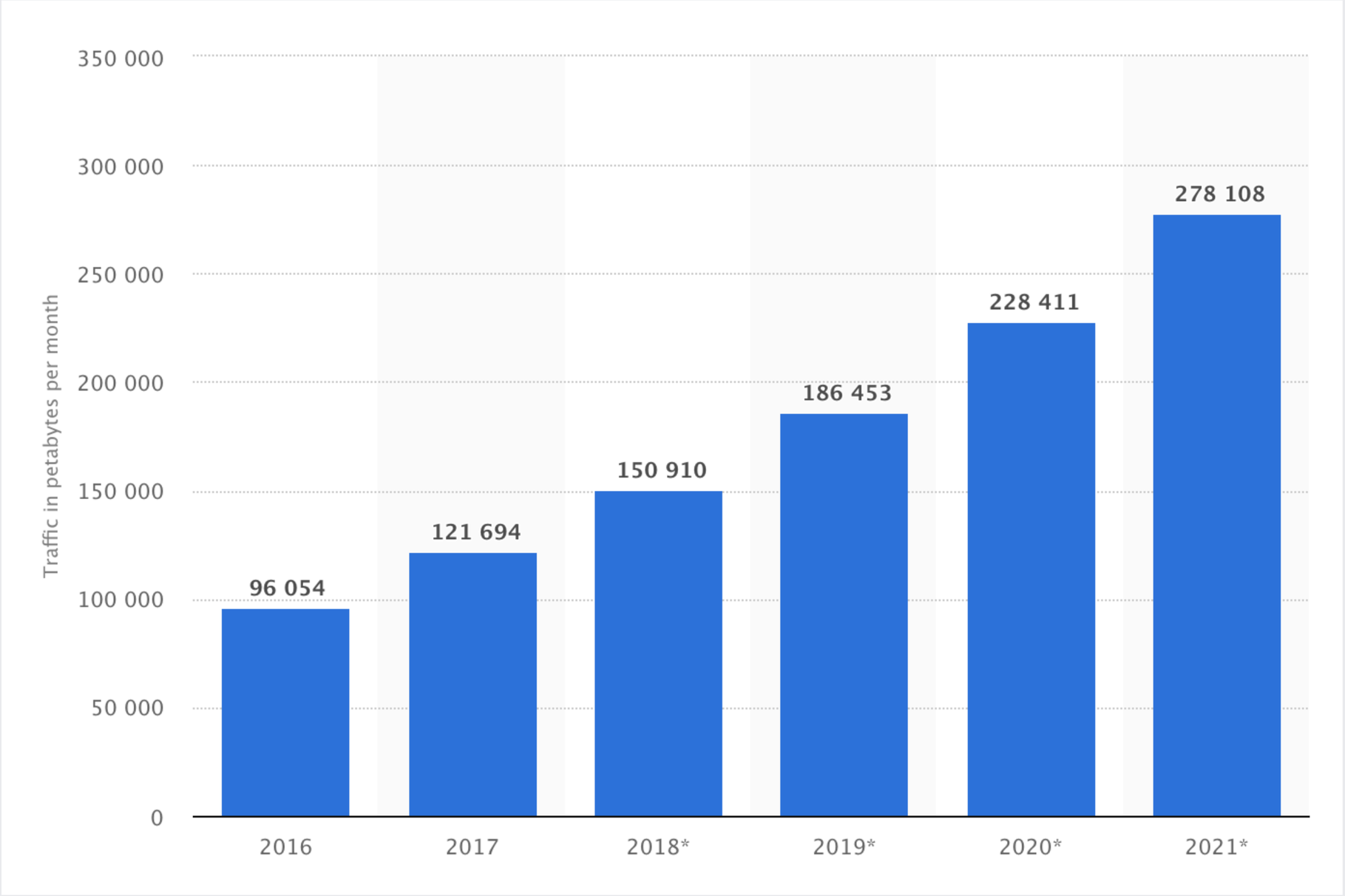
Machine Learning and Systems

- Reinforcement Learning
- Computer Vision, OCR

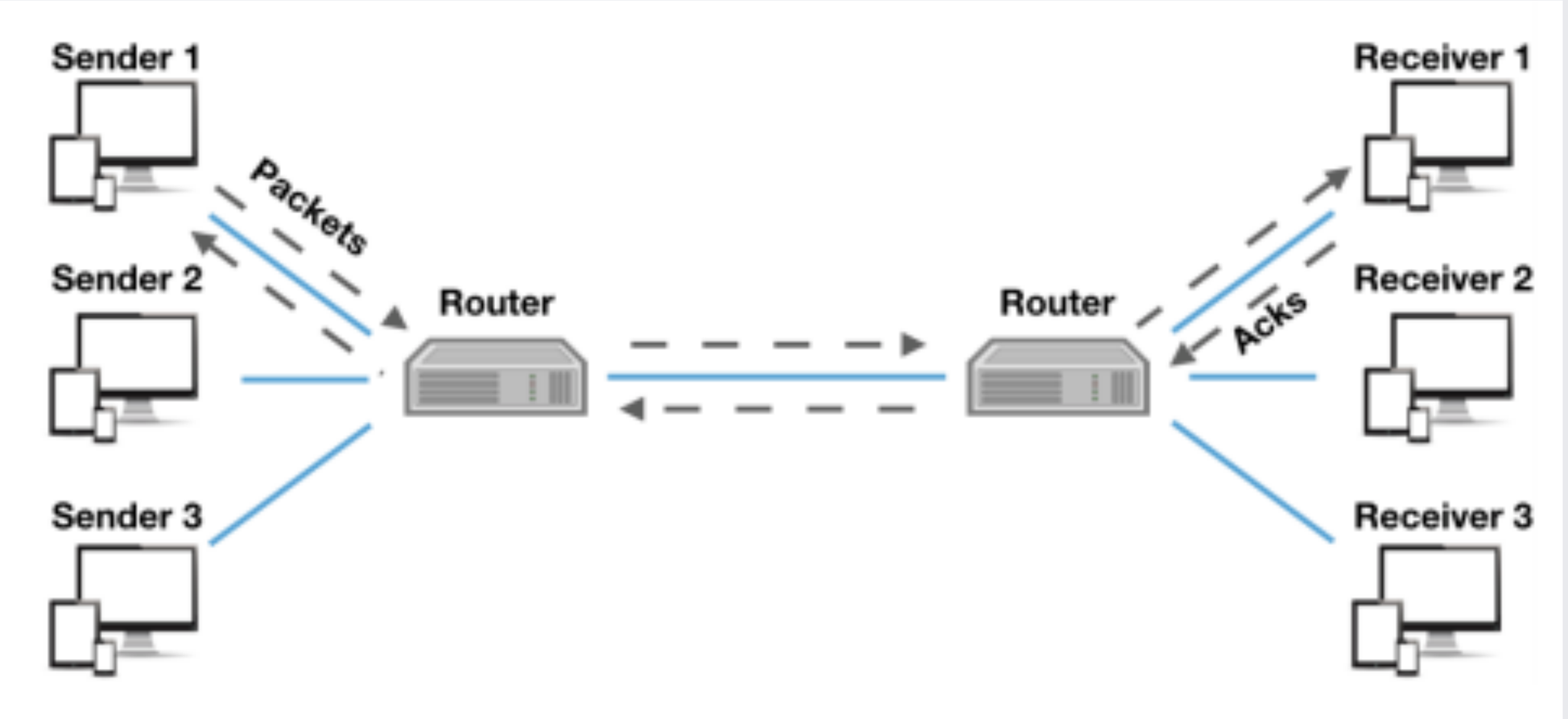
Past: Traffic Infra @ FB:

- L7 Load Balancers, CDN, Edge termination, etc.
- Low-latency live video-streaming
- Several C++ networking libs:
  - https://github.com/facebook/proxygen
  - https://github.com/facebook/wangle
  - https://github.com/facebook/folly

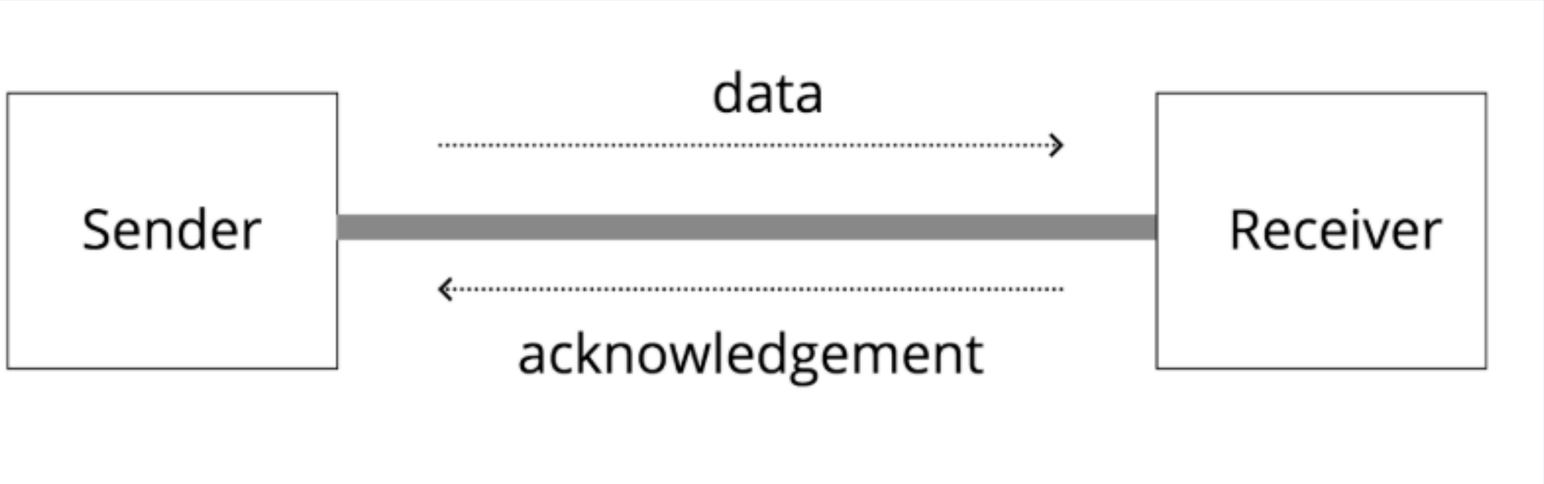
Global Internet Traffic (PB/month)



Network Congestion



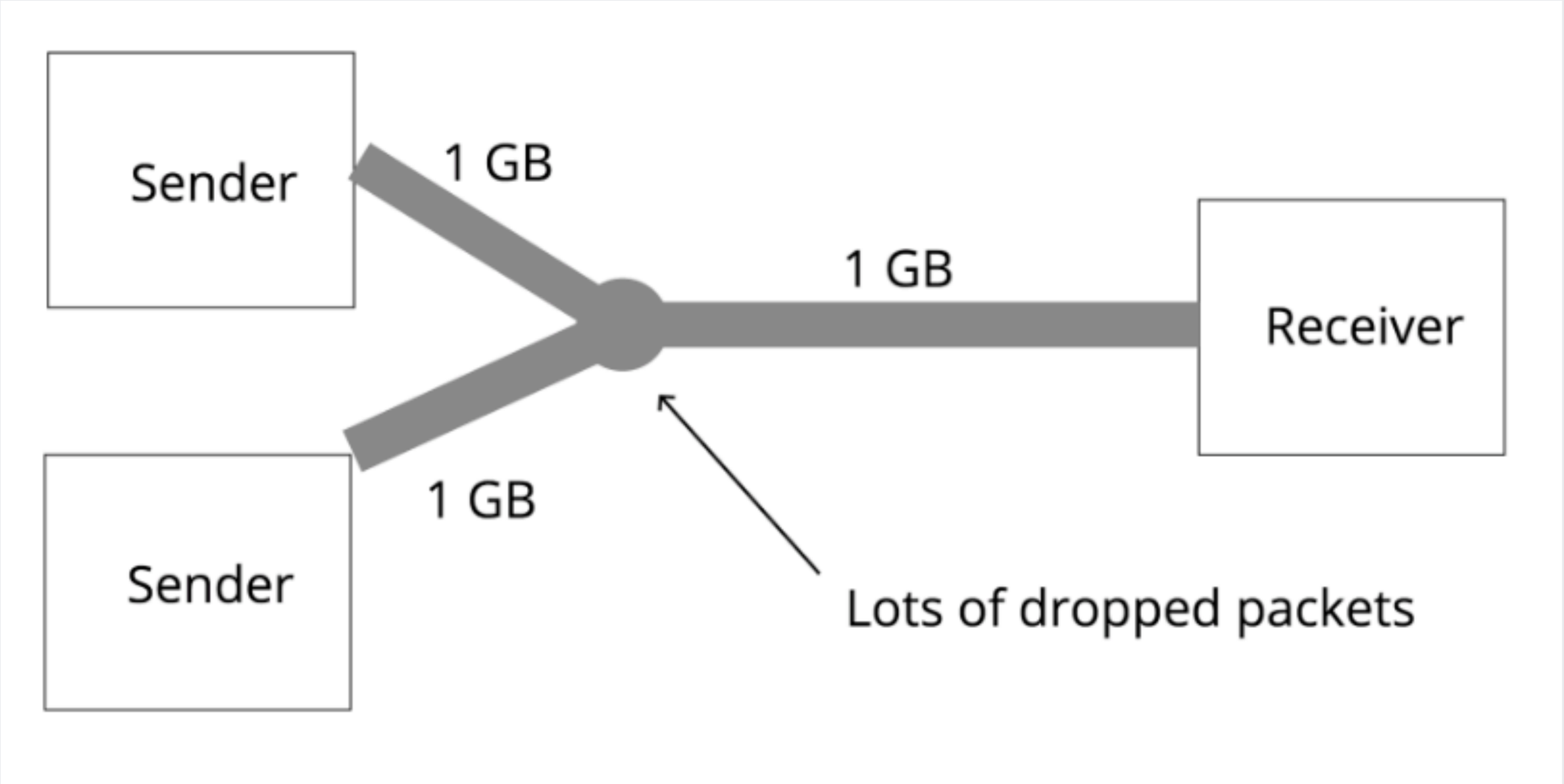
Computer Networks 101



ACK tells you:

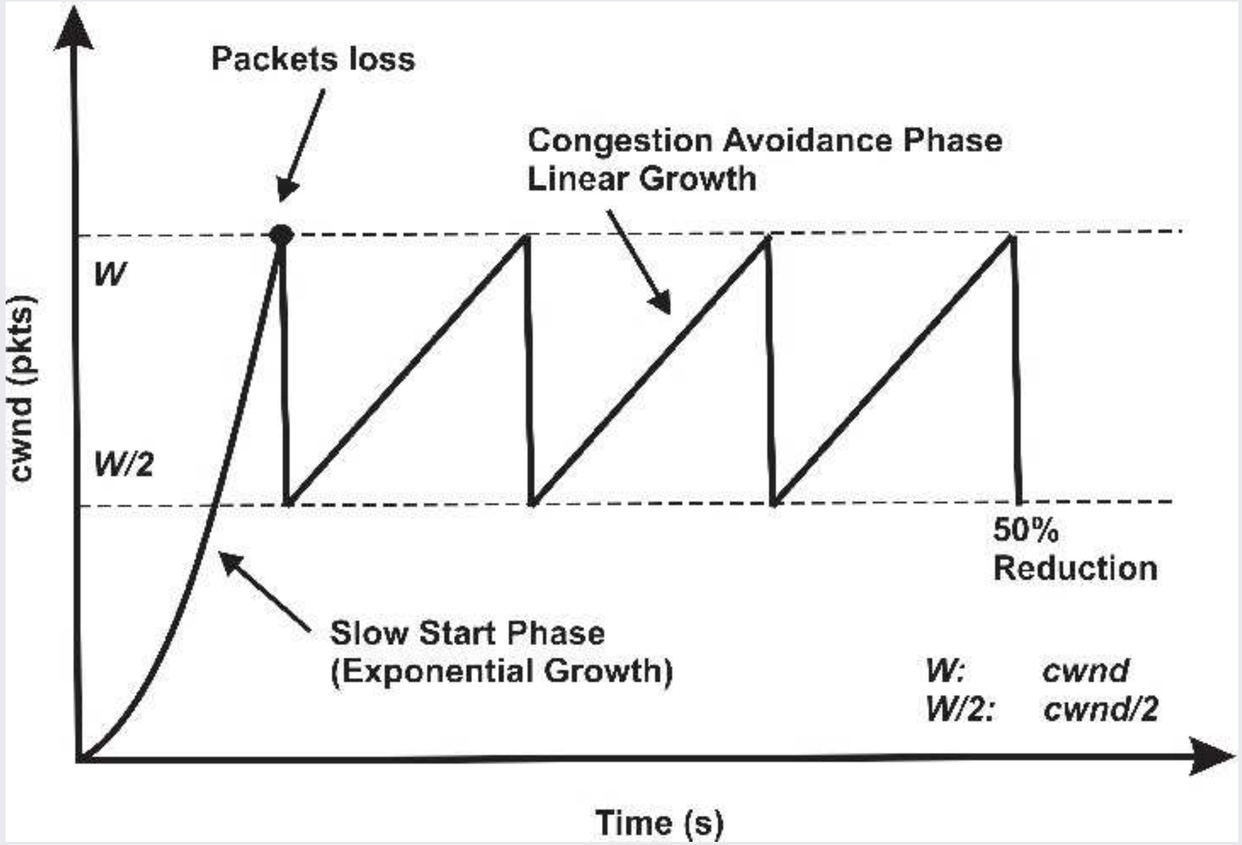
- The largest packet number received
- Measured round-trip time (RTT)
- ...

Computer Networks 101



Traditional Congestion Control

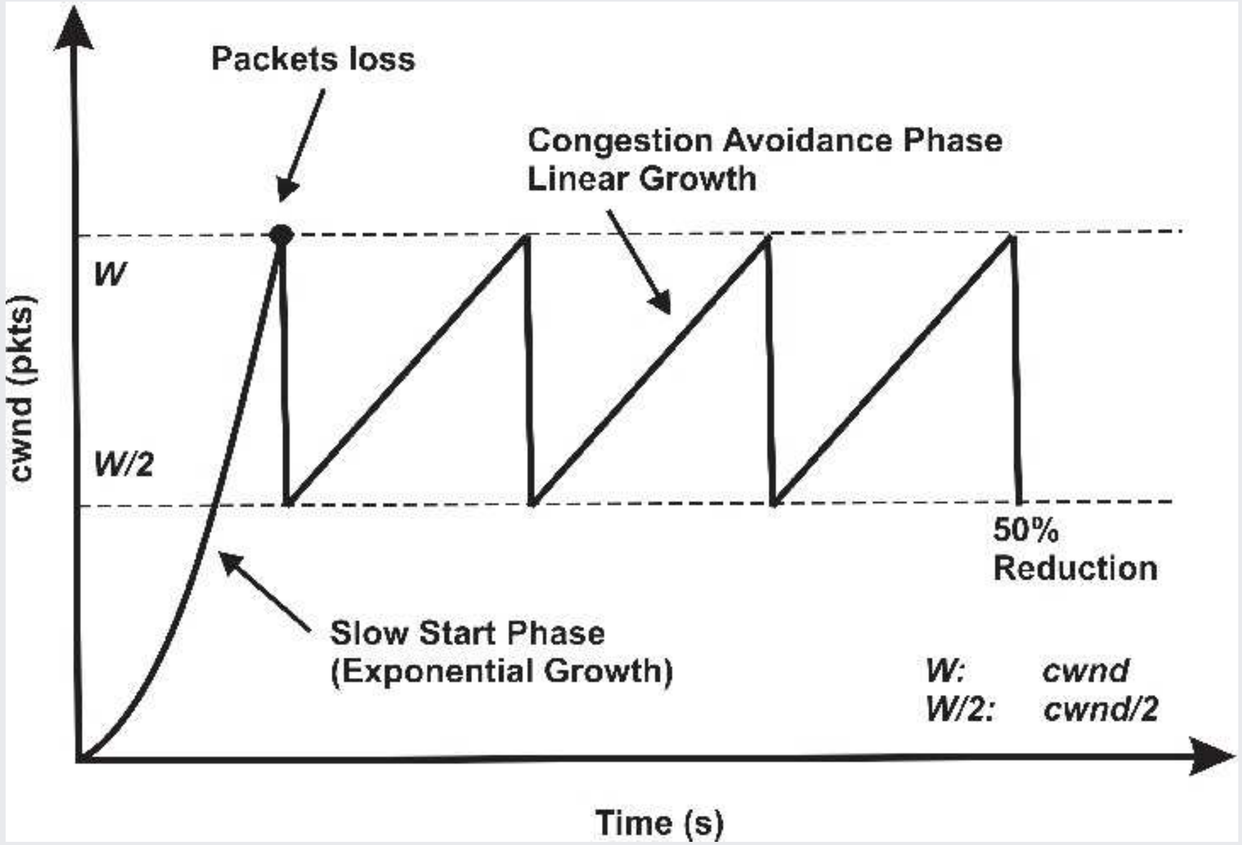
- Hand-engineered
- Decentralized
- Reactive, not predictive
- Often too conservative



Traditional Congestion Control

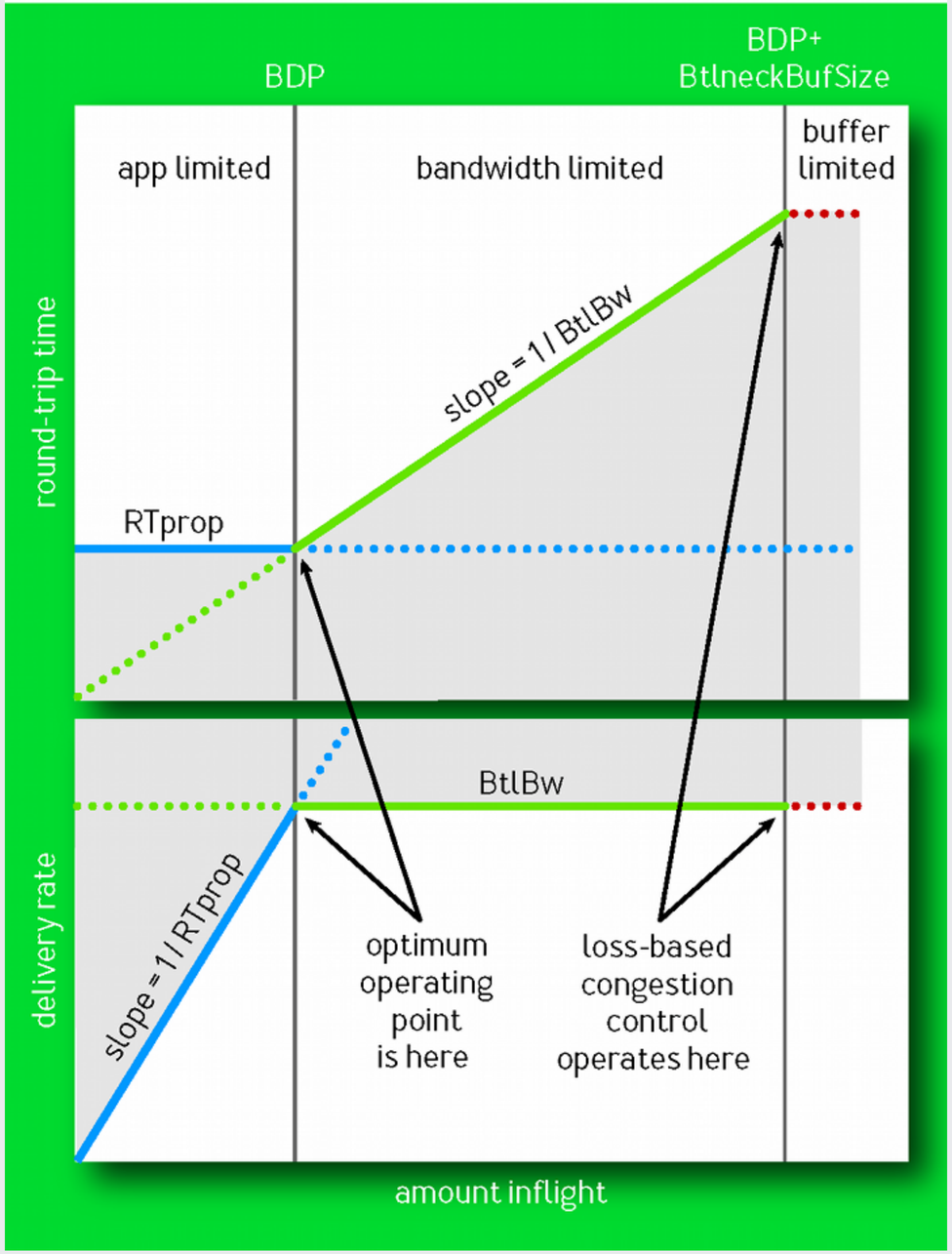
- Hand-engineered
- Decentralized
- Reactive, not predictive
- Often too conservative

30+ years of active research!



BBR

FIGURE 1: **DELIVERY RATE AND ROUND-TRIP TIME VS. INFLIGHT**



BBR: Congestion-Based Congestion Control, Measuring bottleneck bandwidth and round-trip propagation time, Cardwell et al.

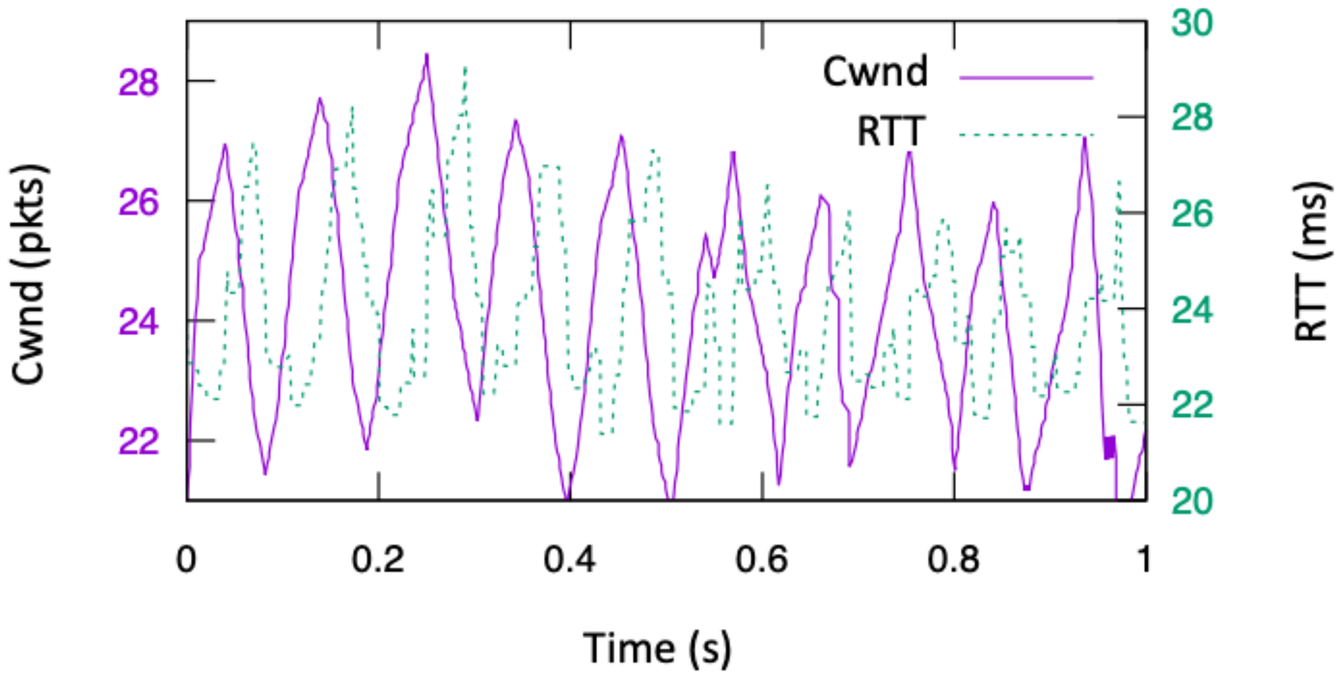
Copa

Calculate target rate $= r_t = \frac{1}{\delta d_q}$

Velocity for faster convergence

If current rate $<$ r_t: additively increase by $\frac{v}{\delta}$ pkts/RTT

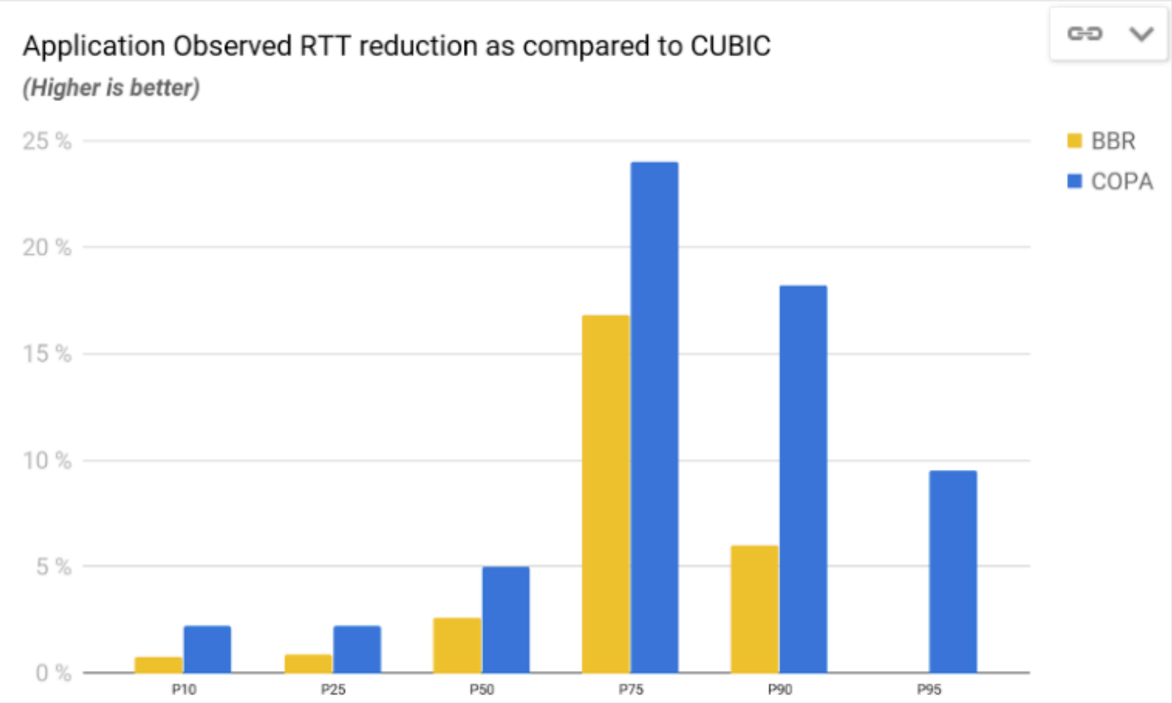
Else: additively decrease by $\frac{v}{\delta}$ pkts/RTT

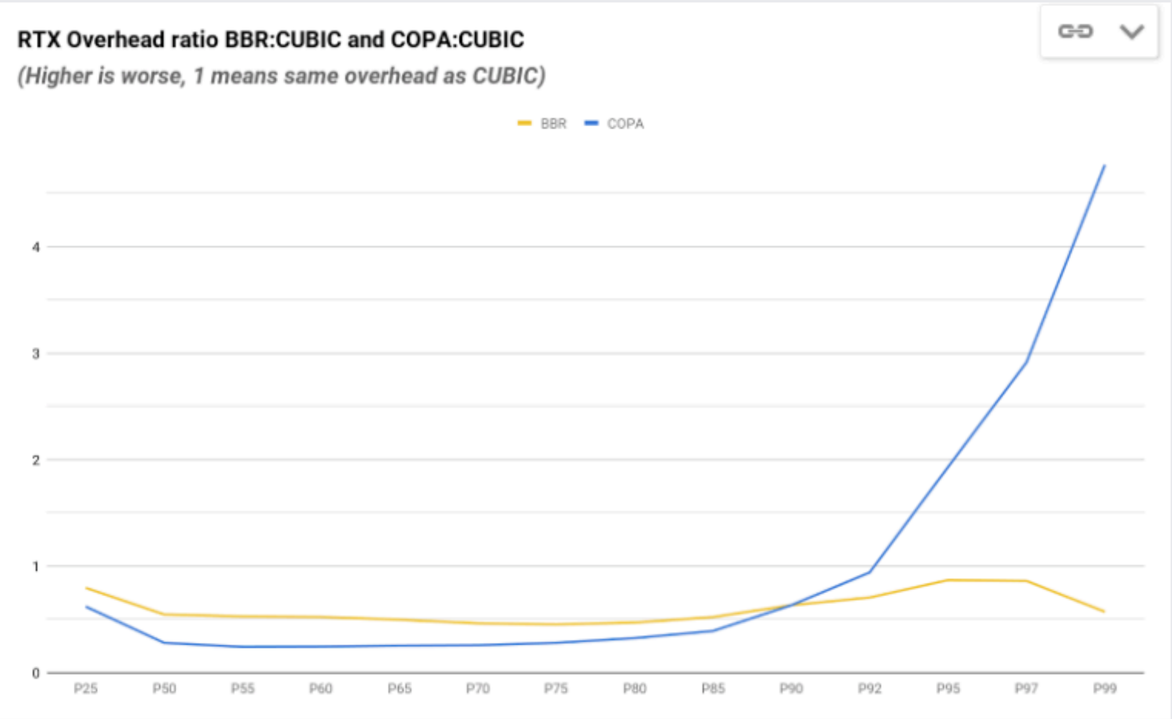


Copa: Practical Delay-Based Congestion Control for the Internet, Arun et al.

Challenges

- Still hand-engineered with certain assumptions
- Hard to encode all assumptions:
  - ACK batching known to break certain methods
  - Network policers affect delay-based algorithms
- Experimentation and deployment environments differ
- App requirements vary:
  - Live-video streaming: low delay
  - Video playback: high throughput
  - ...
- No single tunable method for varying scenarios

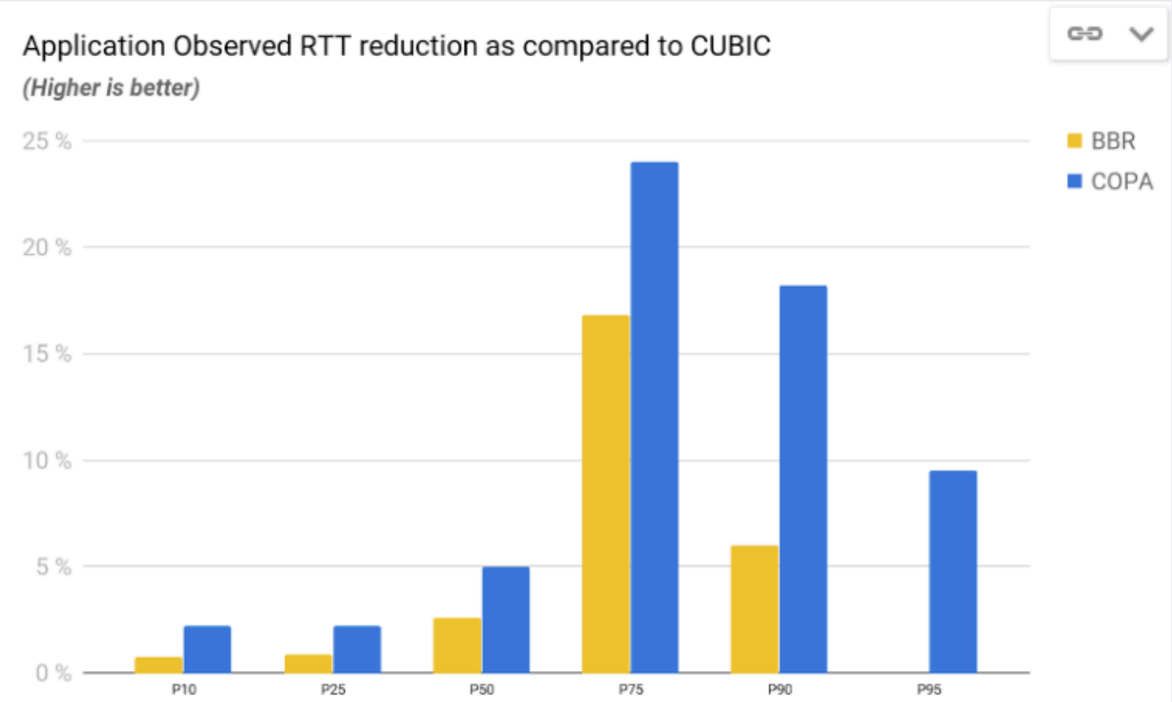


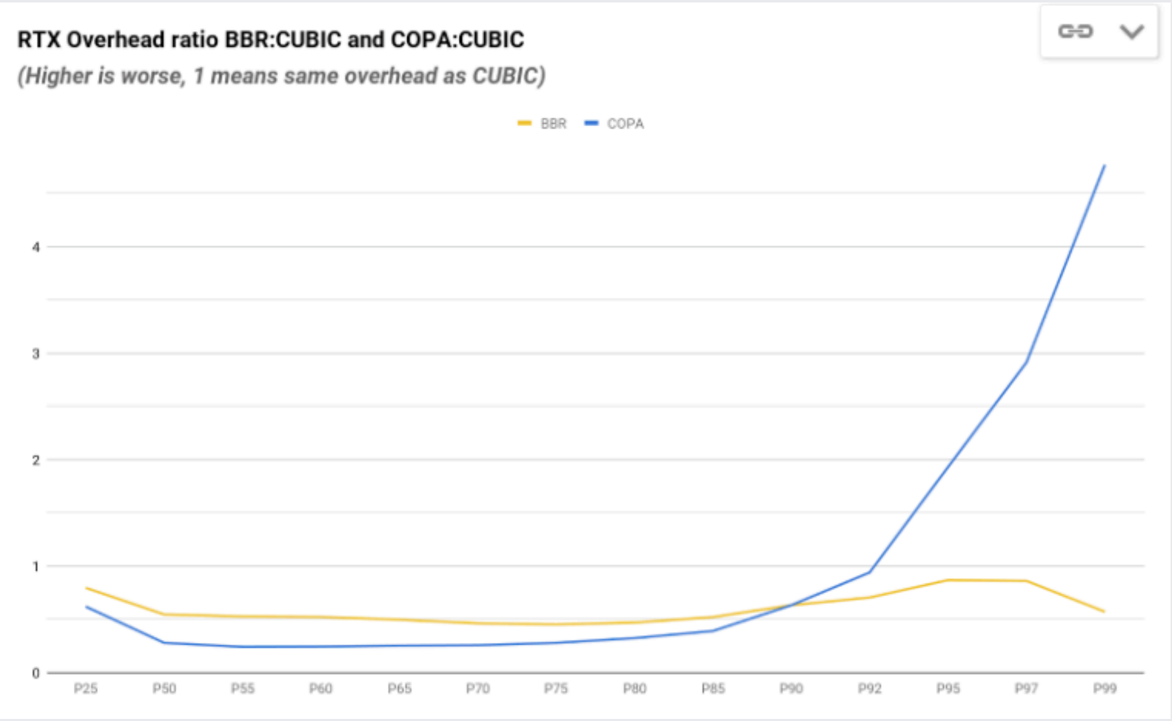


Challenges

- Still hand-engineered with certain assumptions
- Hard to encode all assumptions:
  - ACK batching known to break certain methods
  - Network policers affect delay-based algorithms
- Experimentation and deployment environments differ
- App requirements vary:
  - Live-video streaming: low delay
  - Video playback: high throughput
  - ...
- No single tunable method for varying scenarios

Can we learn from data?



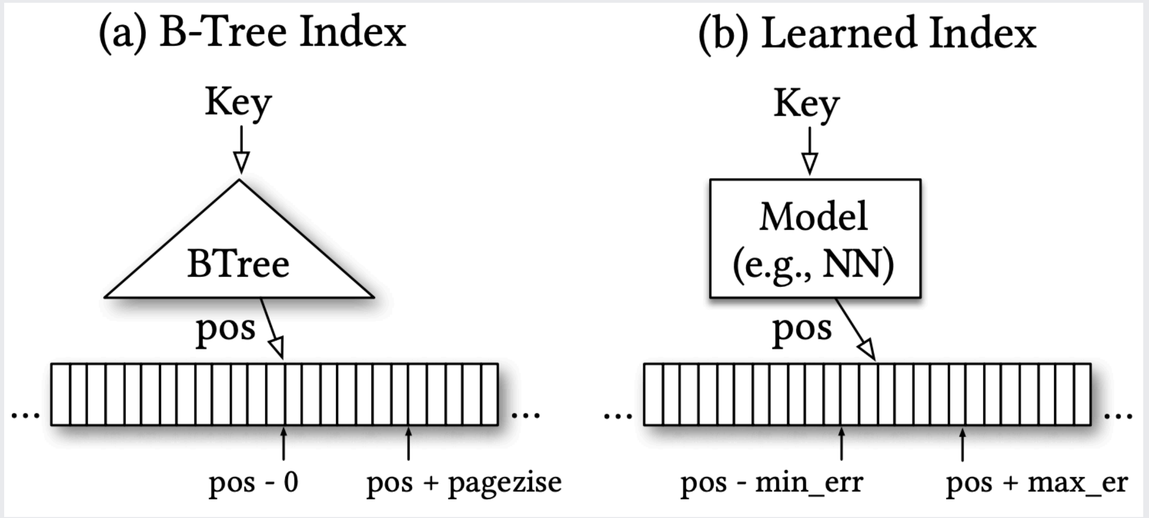


Machine Learning for Systems

Database Index Structures

The Case for Learned Index Structures, Kraska et al

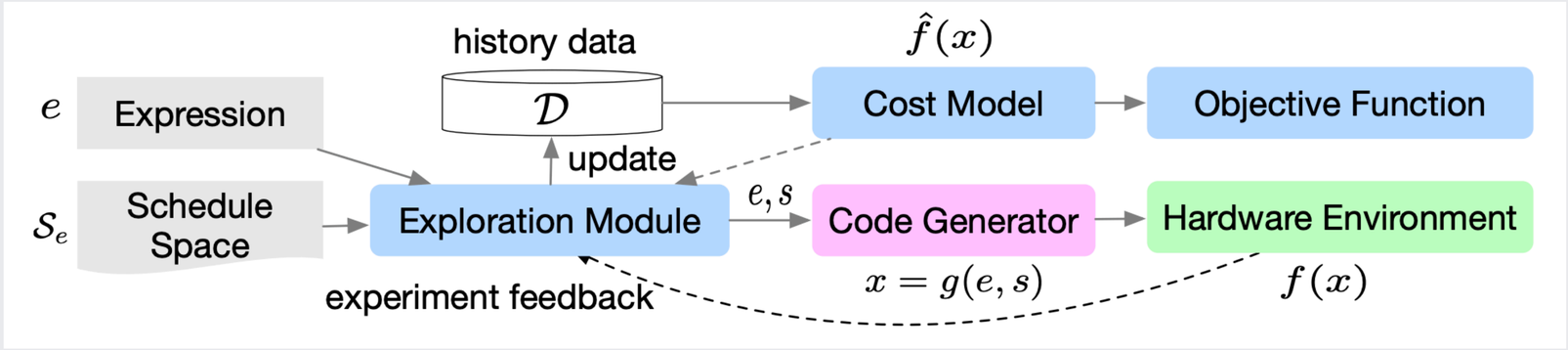
- Databases are really B-Trees
- B-Trees are really Decision Trees that overfit to the data
- O(log N) lookup times for all items
- Insight: Sorted Indexes are CDFs
- Could we learn usage patterns and optimize lookup times?



Code Generation

Learning to Optimize Tensor Programs, Chen et al

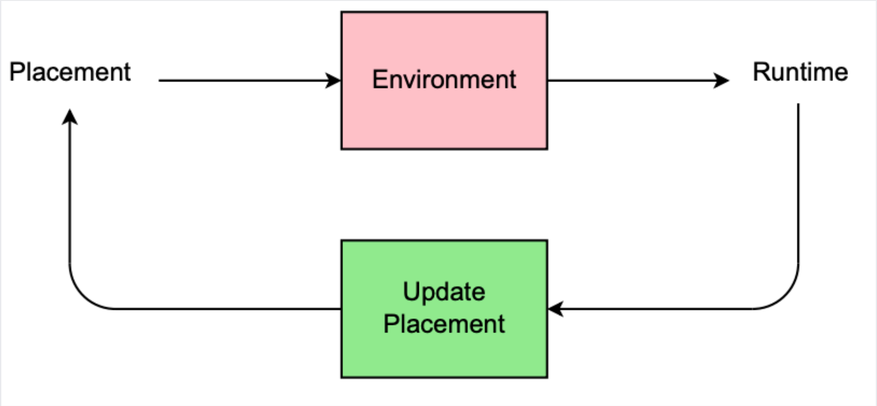
- Problem: Deep Learning runs on too many platforms, impossible to hand-optimize primitives
- Competitive performance on state-of-the-art hard-tuned libraries on various CPUs and GPUs
- Problem: $\arg\min_{s \in \mathcal{S}_e} f(g(e, s))$
  - Se: Schedule (transformation) space for a high-level expression e
  - x = g(e, s): Generated low-level code for expression e via schedule s
  - f(x): Runtime in hardware for x



Job Scheduling

Device Placement Optimization with Reinforcement Learning, Mirhoseini et al

- Optimal allocation of ops in a neural net to devices, such as during model parallelism
- Optimize for runtime
- State: P = {p1, ..., pm}, pi = device for op i
- Seq2Seq, Policy Gradients



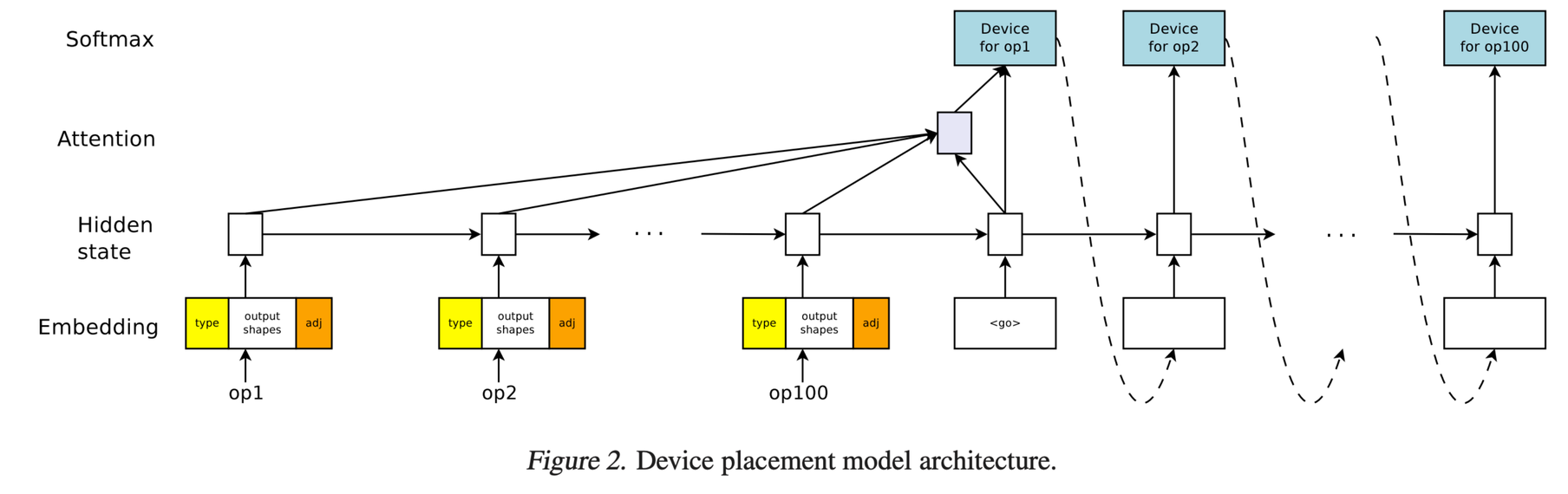
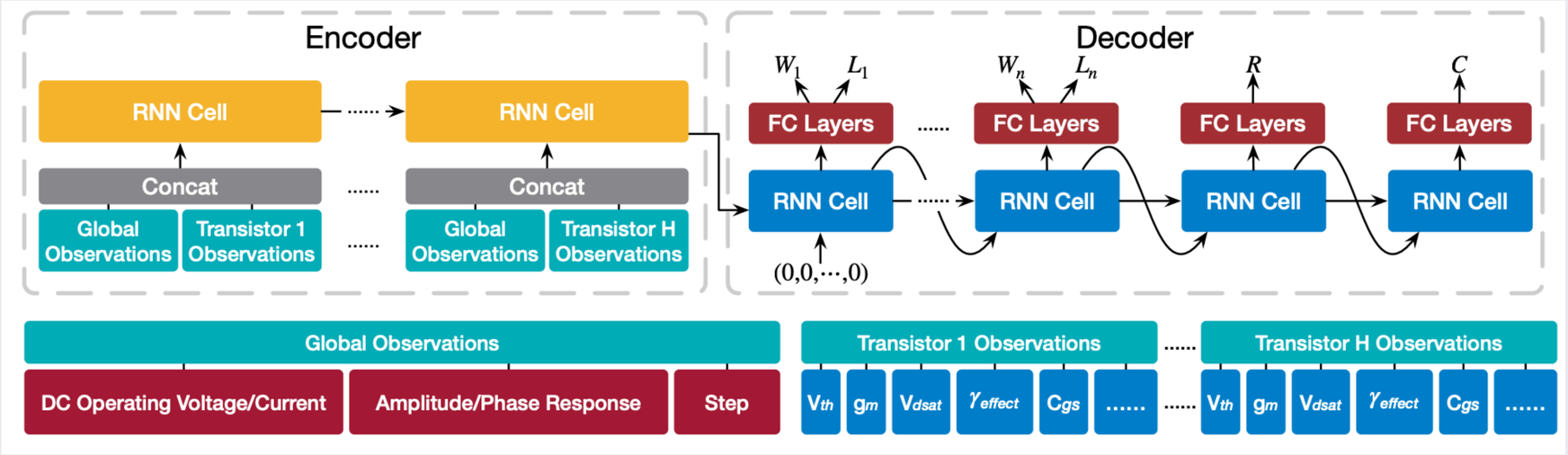


Figure 2. Device placement model architecture.

Circuit Design

Learning to Design Circuits, Wang et al

- Constrained optimization problem: Hard constraints with a goal to maximize or minimize something (such as power)
- Seq2Seq + RL
- State: Transistor info such as voltage, etc.
- Reward prioritizes satisfying hard constraint

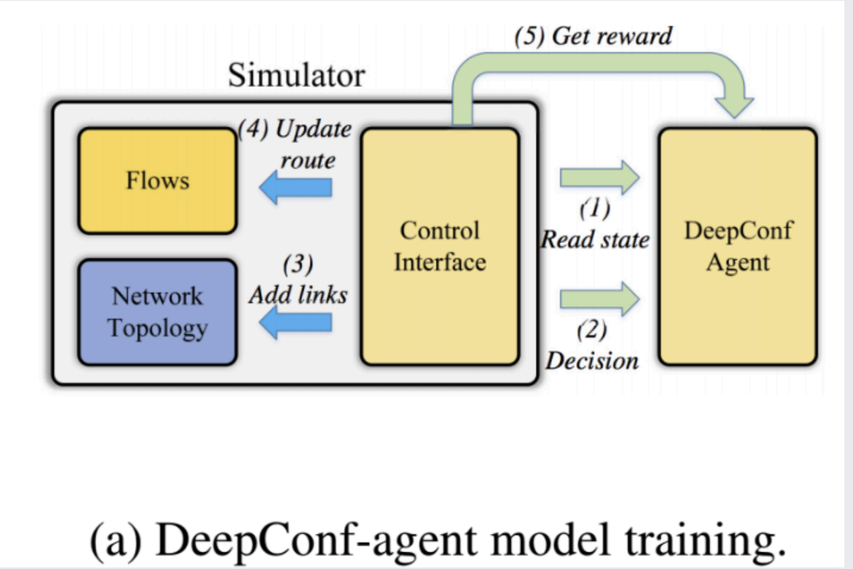


Network Topology

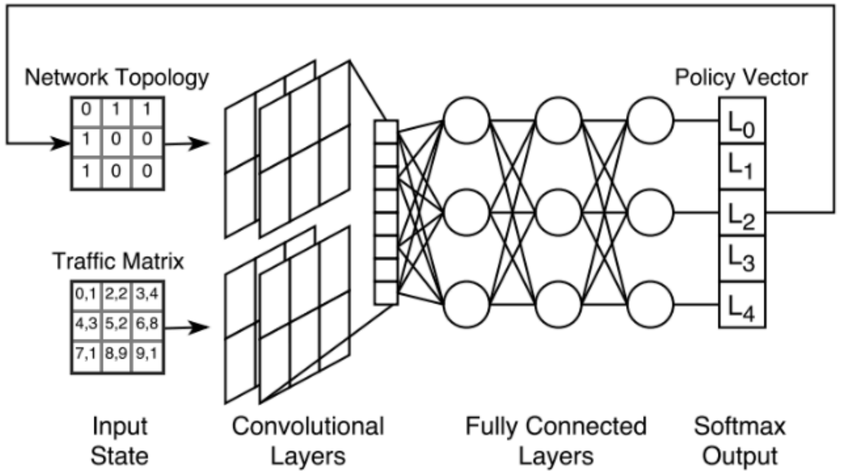
DeepConf: Automating Data Center Network Topologies Management with Machine Learning, Salman et al

- Datacenter network topologies have hardware constraints
- Require configuring params to maximize X (link utilization, etc) subject to these constraints
- Typically Integer Linear Programming problems (NP Complete)
- State: Current link allocation

  Reward: Total link utilization



(a) DeepConf-agent model training.



(b) The CNN model utilized by the DeepConf-agent.

mvfst-rl

An asynchronous RL platform for congestion control in QUIC transport protocol

mvfst-rl

I. QUIC Networking Stack (https://github.com/facebookincubator/mvfst):

- Handles Facebook production traffic
- UDP-based
- Application-layer (as opposed to Kernel) congestion control

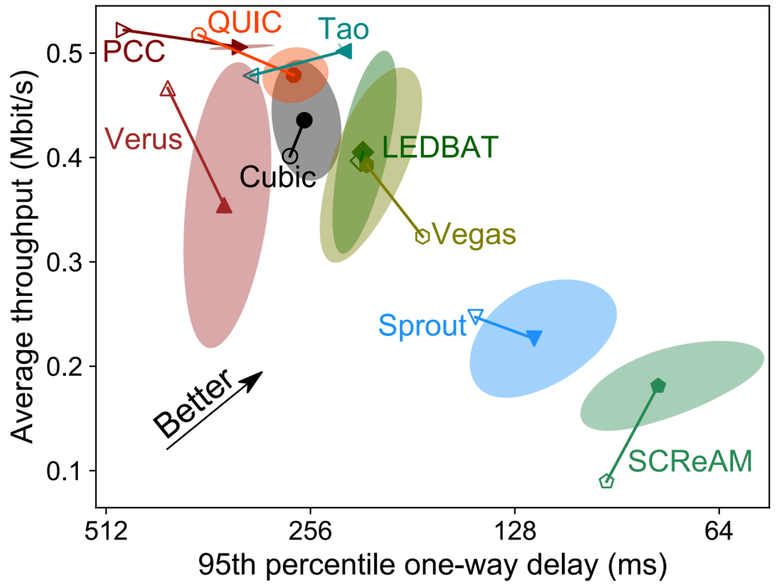
mvfst-rl

I. QUIC Networking Stack (https://github.com/facebookincubator/mvfst):

- Handles Facebook production traffic
- UDP-based
- Application-layer (as opposed to Kernel) congestion control

II. Pantheon Network Emulator (https://pantheon.stanford.edu/):

- Tunneled client-server setup
- Bayesian Optimization to calibrate network emulators



(a) Nepal to AWS India (wireless), 1 flow, 10 trials. Mean replication error: 19.1%. P188.

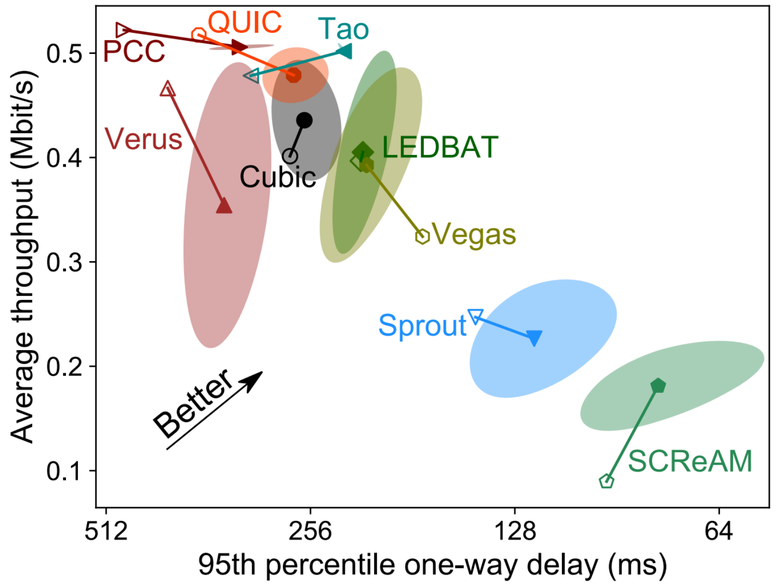
Pantheon: the training ground for Internet congestion-control research, Yan et al.

mvfst-rl

I. QUIC Networking Stack (https://github.com/facebookincubator/mvfst):
- Handles Facebook production traffic
- UDP-based
- Application-layer (as opposed to Kernel) congestion control

II. Pantheon Network Emulator (https://pantheon.stanford.edu/):
- Tunneled client-server setup
- Bayesian Optimization to calibrate network emulators

III. TorchBeast (https://github.com/facebookresearch/torchbeast):
- Asynchronous RL training in PyTorch
- Centralized learner on GPU
- Massively parallel actors / environments
- RPC-based interaction with actors / environments

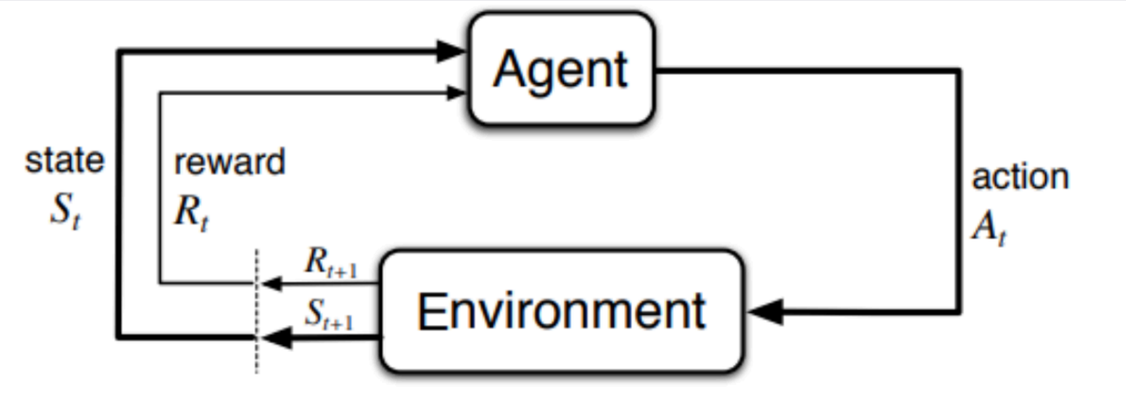


(a) Nepal to AWS India (wireless), 1 flow, 10 trials. Mean replication error: 19.1%. P188.

Pantheon: the training ground for Internet congestion-control research, Yan et al.

RL Basics

<S, A, R, S'>

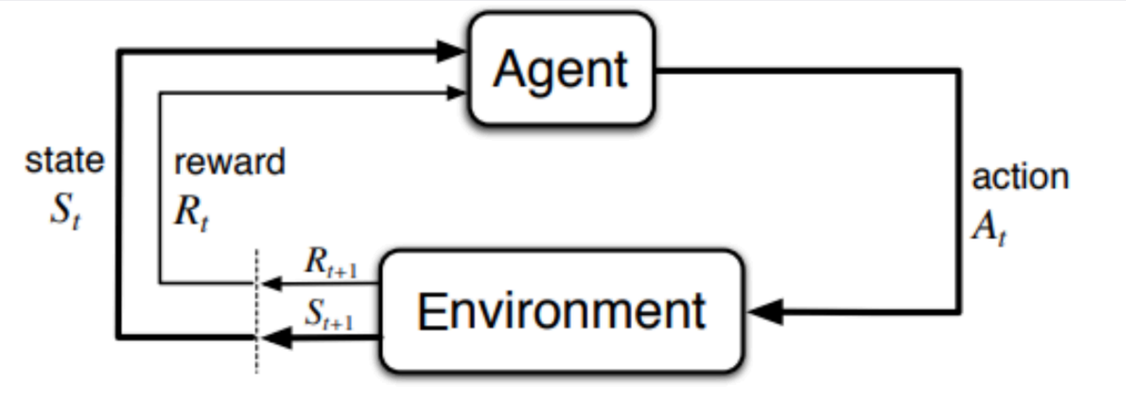


Markov Decision Process (MDP)

- Next state S' depends only on current state S and action A

RL Basics

<S, A, R, S'>



Policy Gradient Methods:

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$$

Markov Decision Process (MDP)

- Next state S' depends only on current state S and action A

Congestion Control as RL

States:

Network conditions:

- Round-trip time
- Bytes sent
- Packets dropped
- Network delay
- ...

Heterogeneous state space

```
struct TransportInfo {
  std::chrono::microseconds srtt;
  std::chrono::microseconds rttvar;
  std::chrono::microseconds lrtt;
  std::chrono::microseconds mrtt;
  uint64_t writableBytes;
  uint64_t congestionWindow;
  uint64_t pacingBurstSize;
  std::chrono::microseconds pacingInterval;
  uint32_t packetsRetransmitted;
  uint32_t timeoutBasedLoss;
  std::chrono::microseconds pto;
  uint64_t bytesSent;
  uint64_t bytesAcked;
  uint64_t bytesRecvd;
  uint64_t totalBytesRetransmitted;
  uint32_t ptoCount;
  uint32_t totalPTOCount;
  PacketNum largestPacketAckedByPeer;
  PacketNum largestPacketSent;
};
```

Congestion Control as RL

Action:

- Modify congestion window (cwnd)
- Max unacknowledged outstanding bytes

Congestion Control as RL

Action:

- Modify congestion window (cwnd)
- Max unacknowledged outstanding bytes

Action Space Choices:

1. Discrete: [1..2000]
   - Cons: Large action space
2. Fixed update: [cwnd / 2, cwnd - x, cwnd, cwnd + x, cwnd * 2]
   - Cons: Lack of finer control
3. Regressed multiplier: max_cwnd * x, x E (0.0, 1.0]
   - More suited for gradient-based methods

Congestion Control as RL

Action:

- Modify congestion window (cwnd)
- Max unacknowledged outstanding bytes

Action Space Choices:

1. Discrete: [1..2000]
   - Cons: Large action space
2. Fixed update: [cwnd / 2, cwnd - x, cwnd, cwnd + x, cwnd * 2]
   - Cons: Lack of finer control
3. Regressed multiplier: max_cwnd * x, x E (0.0, 1.0]
   - More suited for gradient-based methods

Congestion Control as RL

Reward:

- Maximize throughput
- Minimize delay

Typically,

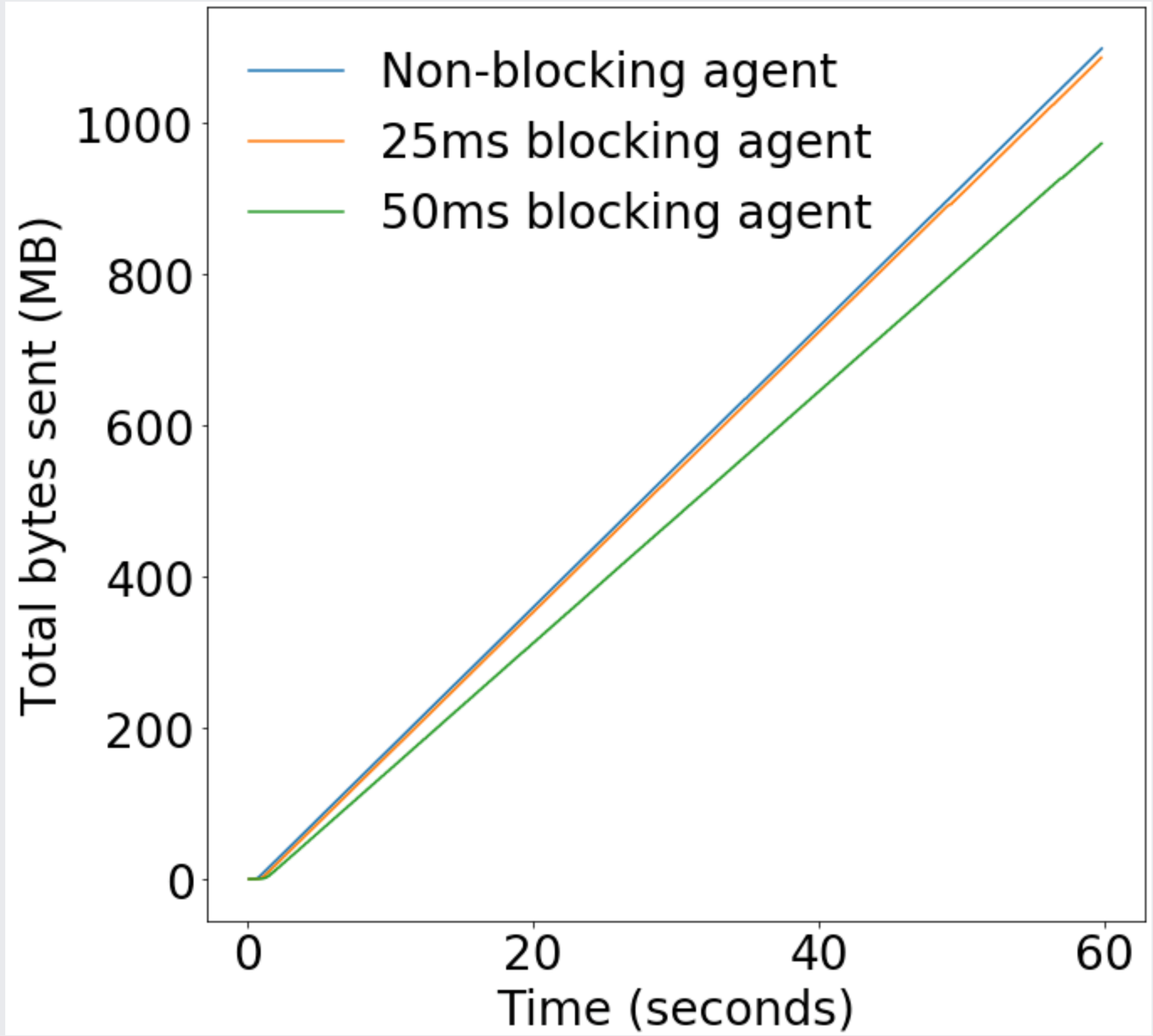
```
reward = throughput - alpha * delay
```

Traditional RL Environments

```
env = ...
state = env.reset()
for _ in range(1000):
  action = model(state)
  state, reward, done = env.step(action)
  if done:
    state = env.reset()
```

Traditional RL Environments

```
env = ...
state = env.reset()
for _ in range(1000):
  action = model(state)  # Blocks the environment
  state, reward, done = env.step(action)
  if done:
    state = env.reset()
```



25ms Policy Lookup => 1.1% fewer bytes sent

50ms Policy Lookup => 11.4% fewer bytes sent

Asynchronous Environment

Typical RL Env:

- <S, A, R, S'>
- <R, S'> corresponds to <S, A>

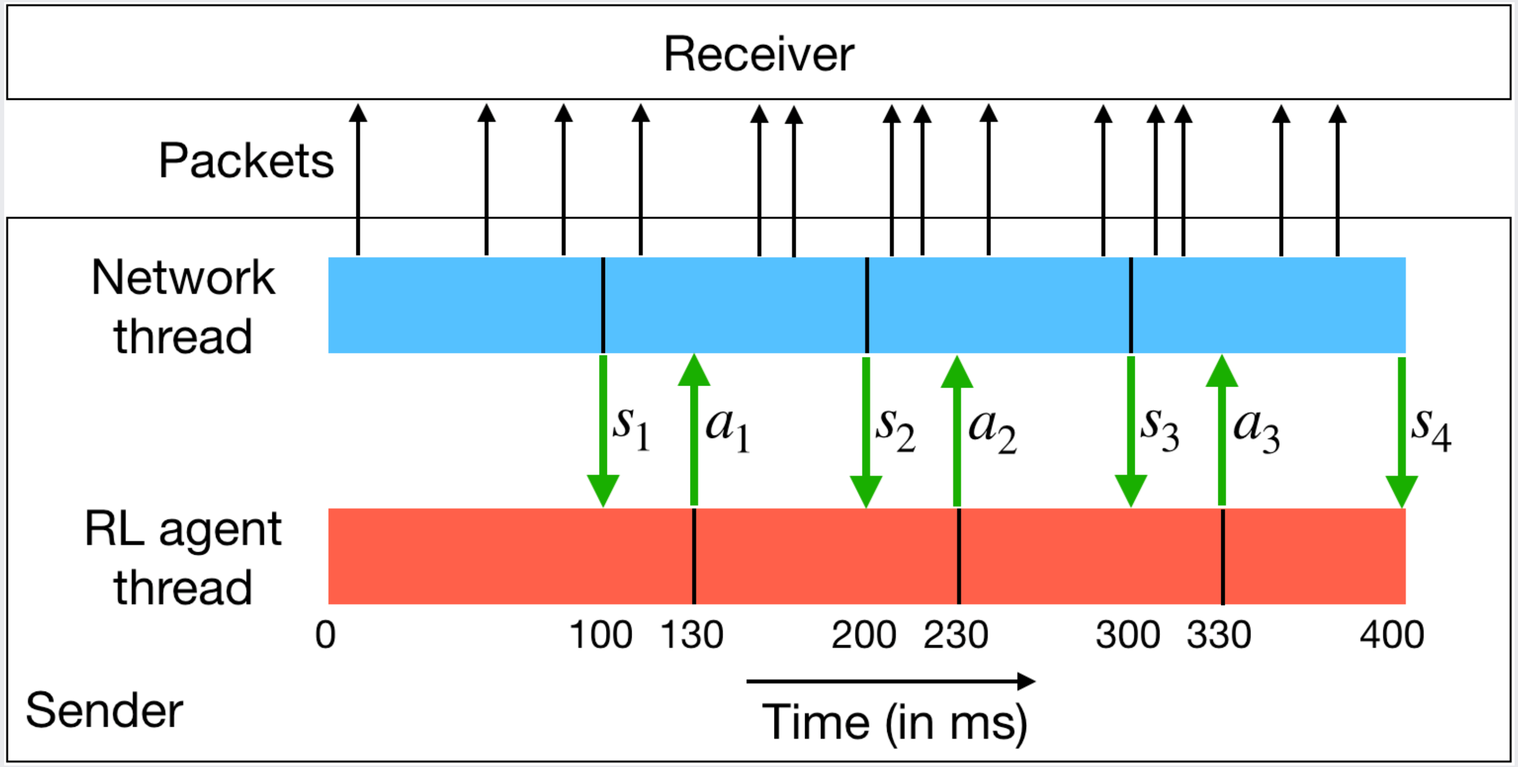
Asynchronous Environment

Typical RL Env:

- <S, A, R, S'>
- <R, S'> corresponds to <S, A>

Async Env:

- Sequence of S' and R'
- Non-blocking agent
- Another step/action A' taken before obtaining S' corresponding to A.



Asynchronous RL agent–sender interaction in mvfst-rl

Asynchronous Environment

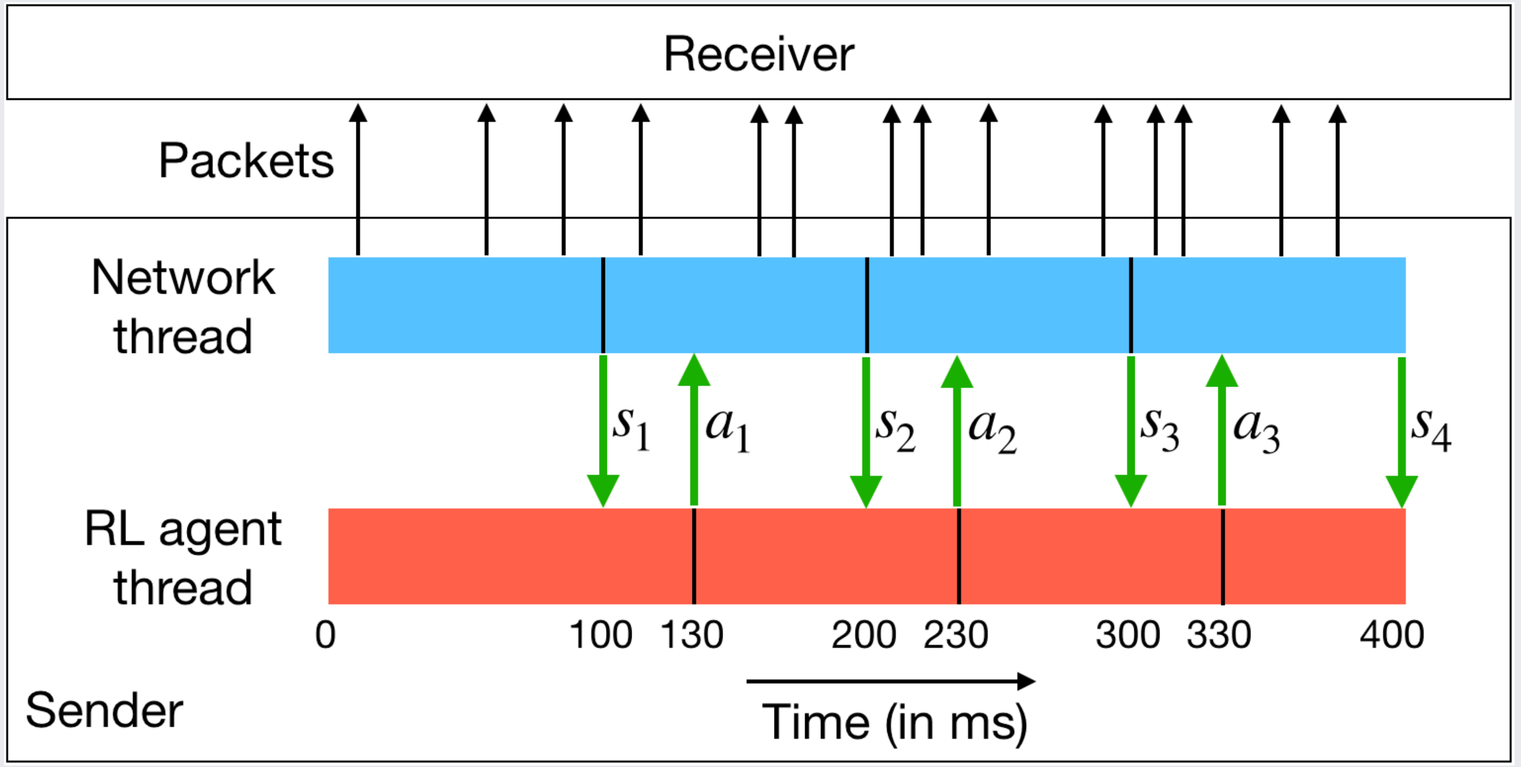
Typical RL Env:

- <S, A, R, S'>
- <R, S'> corresponds to <S, A>

Async Env:

- Sequence of S' and R'
- Non-blocking agent
- Another step/action A' taken before obtaining S' corresponding to A.

Not Markovian!



Asynchronous RL agent–sender interaction in mvfst-rl

MDP with Delayed Actions

Augmented State Space: $\hat{S} = S \times A^k$

k = Action history length

Environment State:

- 20 raw features
- Summary statistics: sum, mean, max, mean, std
  - 100 features
- Action history: one-hot actions + cwnd

State Vector: $100 + k \times (|A| + 1)$

Asynchronous RL Training

Policy Gradient Loss

$$maximize_\theta \, E_\pi[R(\tau)]$$

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$$

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$$

$$\nabla_\theta E_\pi[R(\tau)] = E_\pi[\nabla_\theta log\, \pi_\theta(\tau)\, R(\tau)]$$

Intuition: Nudge params towards higher probability for a trajectory with high reward.

Problem: High variance in $R(\tau)$.

Asynchronous RL Training

Policy Gradient Loss

$$maximize_\theta \, E_\pi[R(\tau)]$$

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$$

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$$

$$\nabla_\theta E_\pi[R(\tau)] = E_\pi[\nabla_\theta log\, \pi_\theta(\tau)\, R(\tau)]$$

Intuition: Nudge params towards higher probability for a trajectory with high reward.

Problem: High variance in $R(\tau)$.

Actor-Critic

Key: Estimate $R(\tau)$ with a model.

- This model is called the value function
- Trained using L2-loss

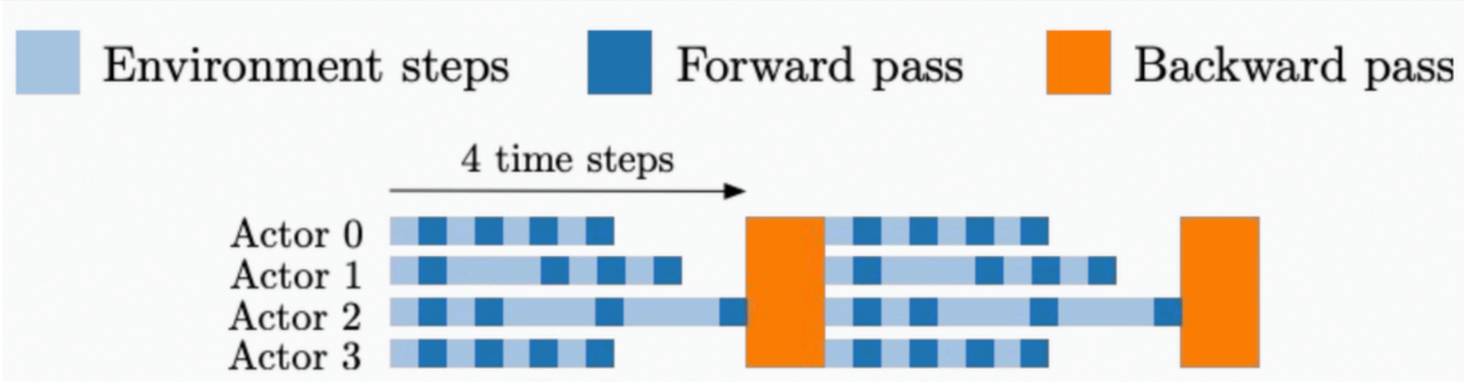
$$\nabla_\theta E_\pi[R(\tau)] = E_\pi[\nabla_\theta log\, \pi_\theta(\tau)\, V(s)]$$

Two models:

- Actor: Policy function $\pi_\theta(a \mid s)$ - next action
- Critic: Value function $V(s)$ - estimated reward

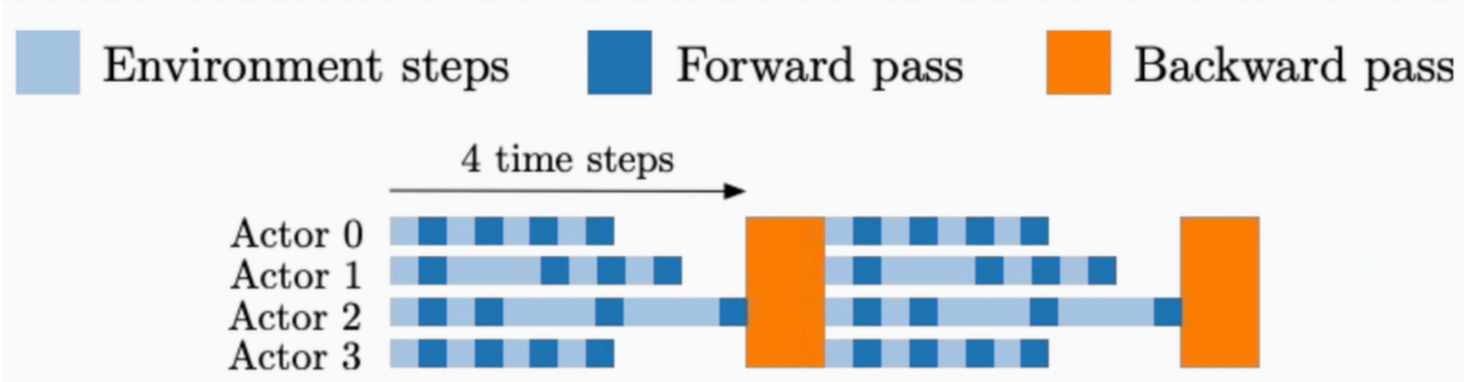
Asynchronous Off-Policy Training

Synchronous Actor-Critic

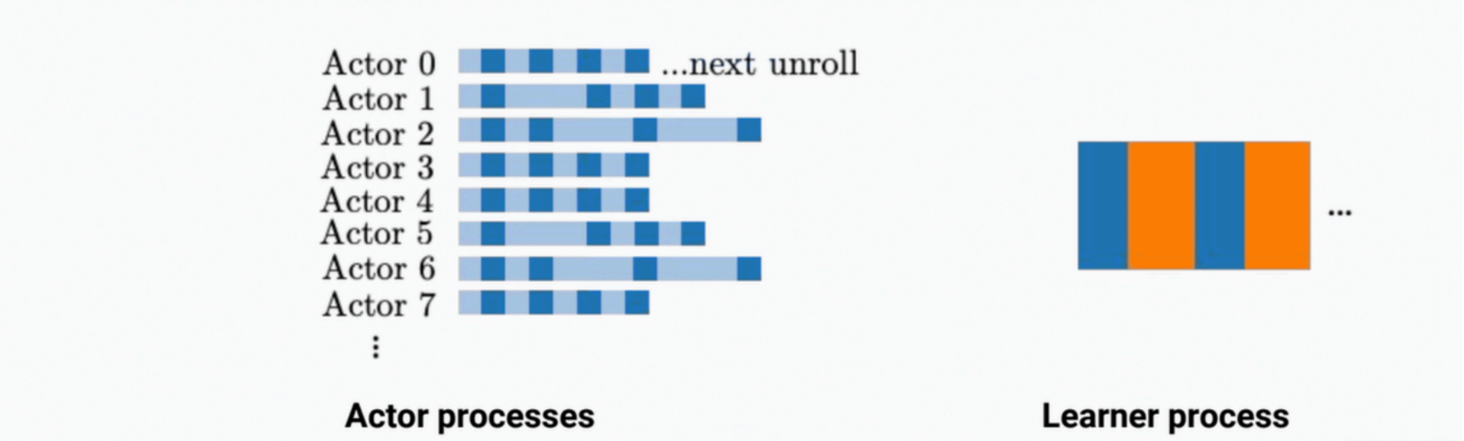


Asynchronous Off-Policy Training

Synchronous Actor-Critic



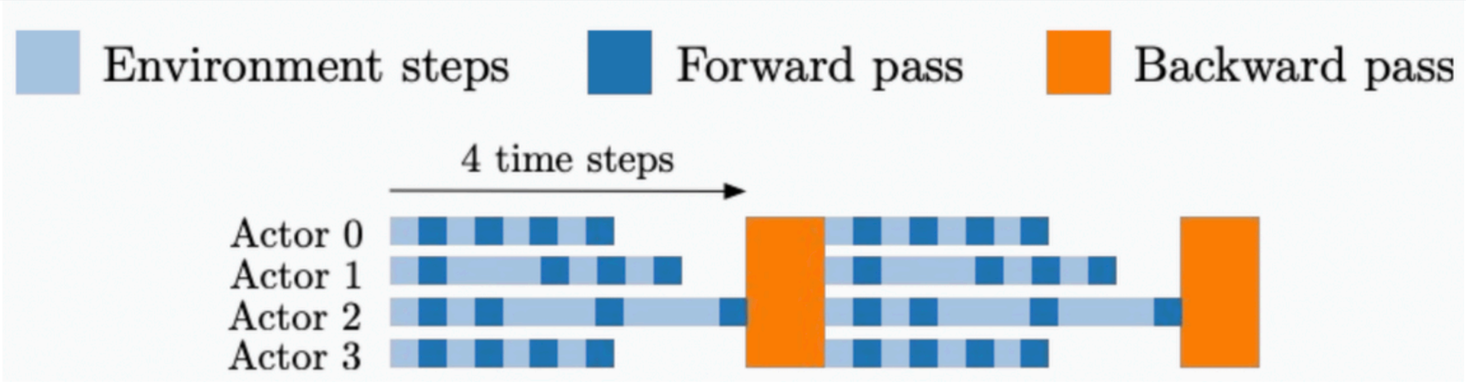
Asynchronous Actor-Critic



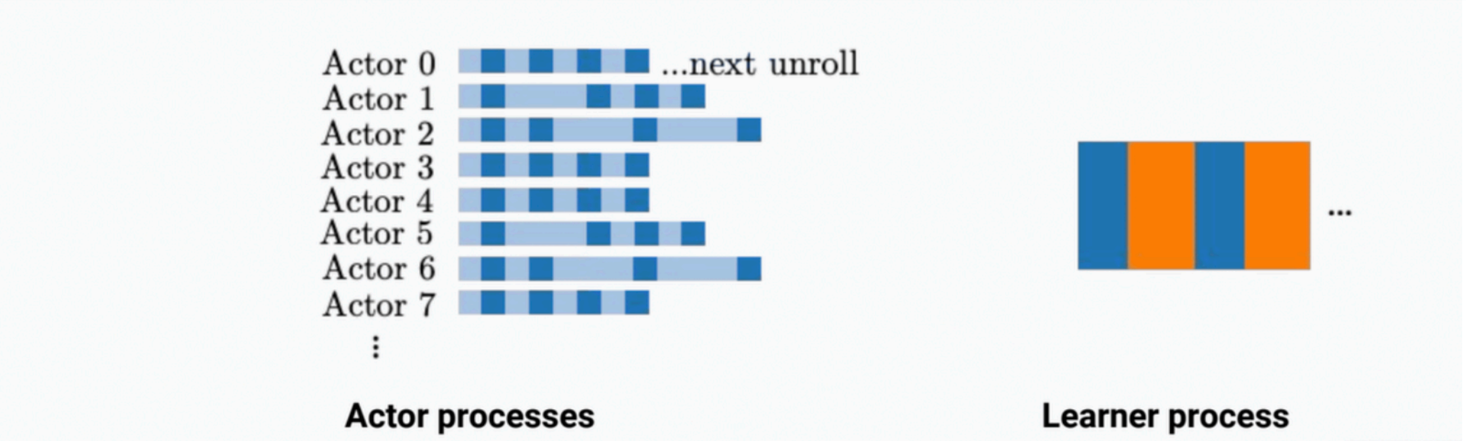
IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures, Espeholt et al.

Asynchronous Off-Policy Training

Synchronous Actor-Critic



Asynchronous Actor-Critic



Problem:

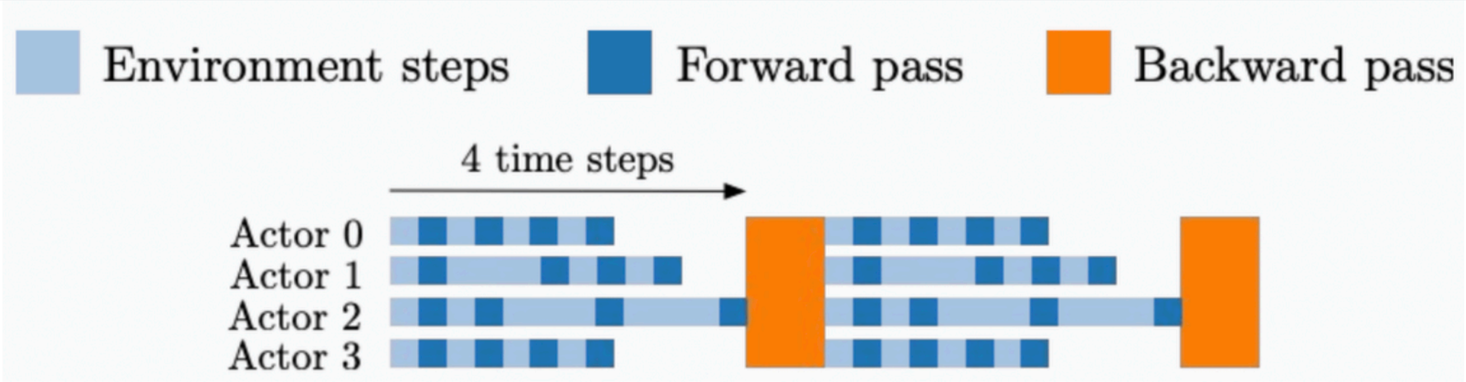
- Actor policy is older than learner policy
- Trajectories are sampled from older policy $\mu(a \mid s)$ instead of target policy $\pi(a \mid s)$.

$$\nabla_\theta E_\pi[R(\tau)] \neq E_\mu[\nabla_\theta log\, \pi_\theta(\tau)\, V(s)]$$

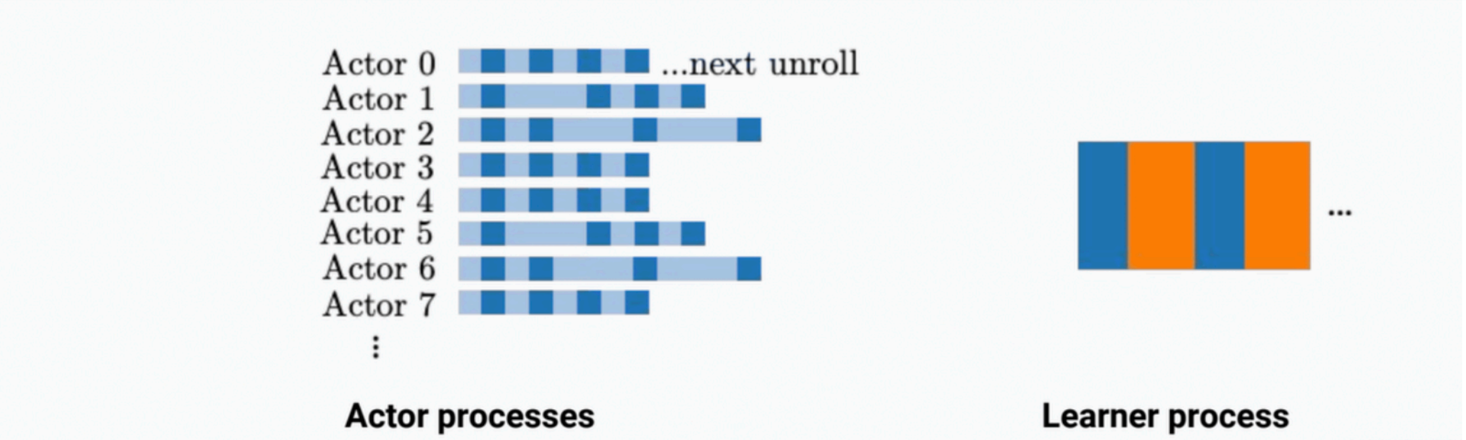
IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures, Espeholt et al.

Asynchronous Off-Policy Training

Synchronous Actor-Critic



Asynchronous Actor-Critic



Problem:

- Actor policy is older than learner policy
- Trajectories are sampled from older policy $\mu(a \mid s)$ instead of target policy $\pi(a \mid s)$.

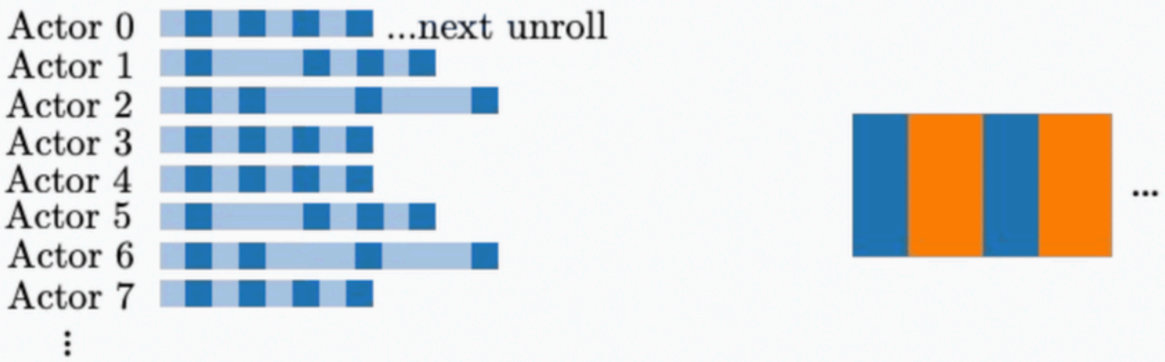
$$\nabla_\theta E_\pi[R(\tau)] \neq E_\mu[\nabla_\theta log\, \pi_\theta(\tau)\, V(s)]$$

Fixed by importance sampling

$$\nabla_\theta E_\pi[R(\tau)] = E_\mu[\frac{\pi(\tau)}{\mu(\tau)} \nabla_\theta log\, \pi_\theta(\tau)\, V(s)]$$

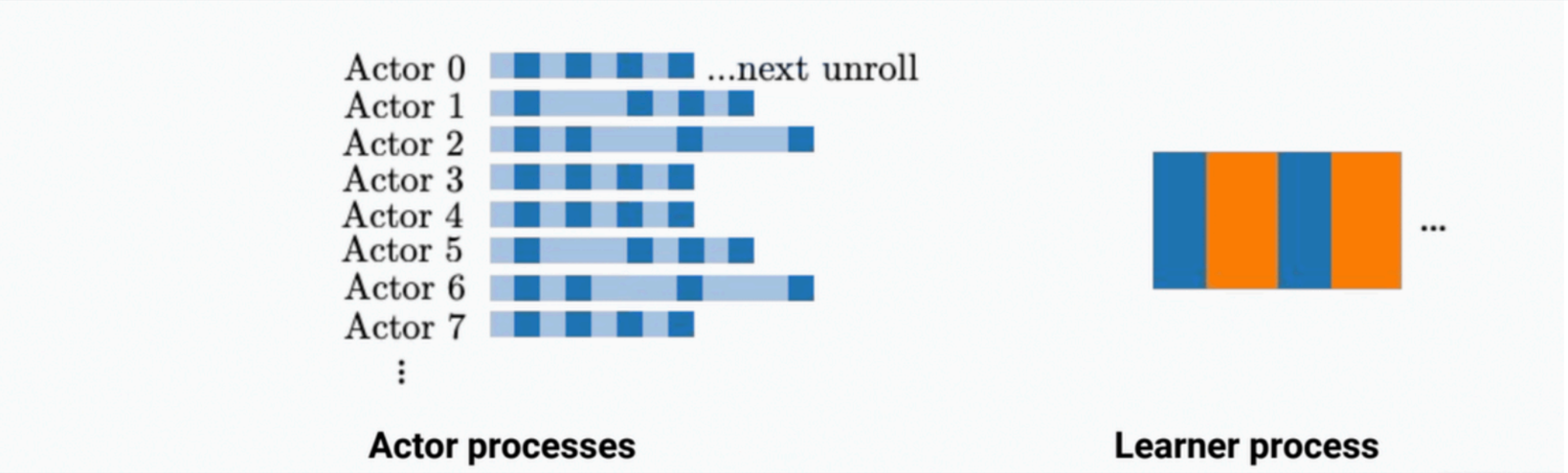
IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures, Espeholt et al.

mvfst-rl Training Architecture

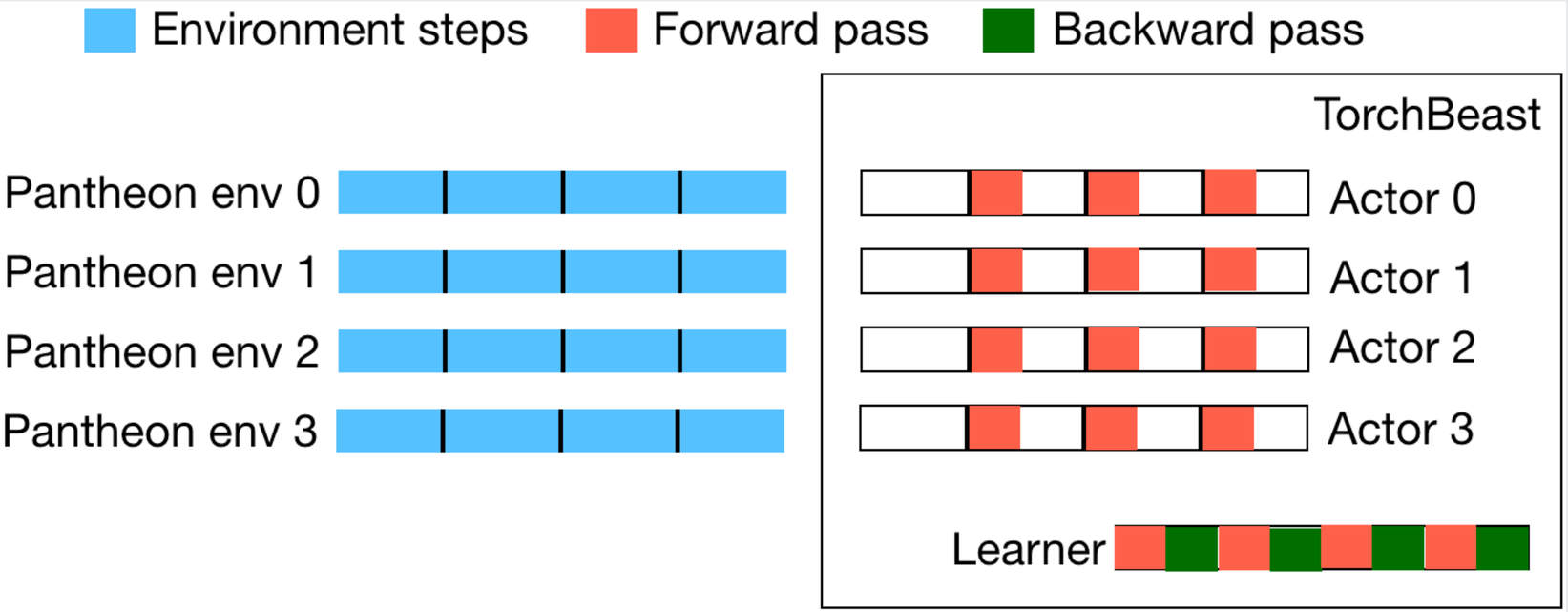


IMPALA:
- Async Actor-Learner
- Sync Environment-Actor

mvfst-rl Training Architecture

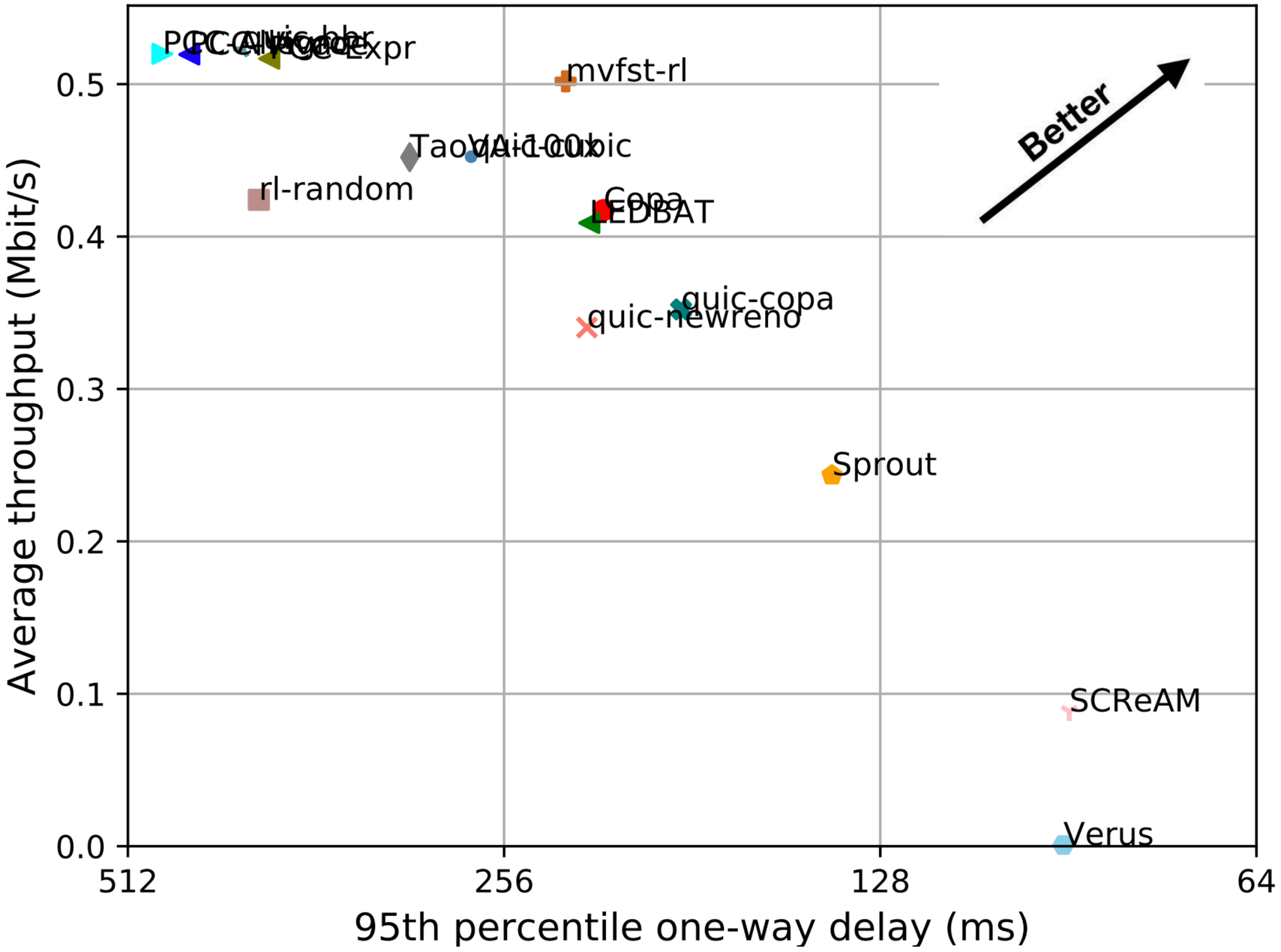


IMPALA:
- Async Actor-Learner
- Sync Environment-Actor

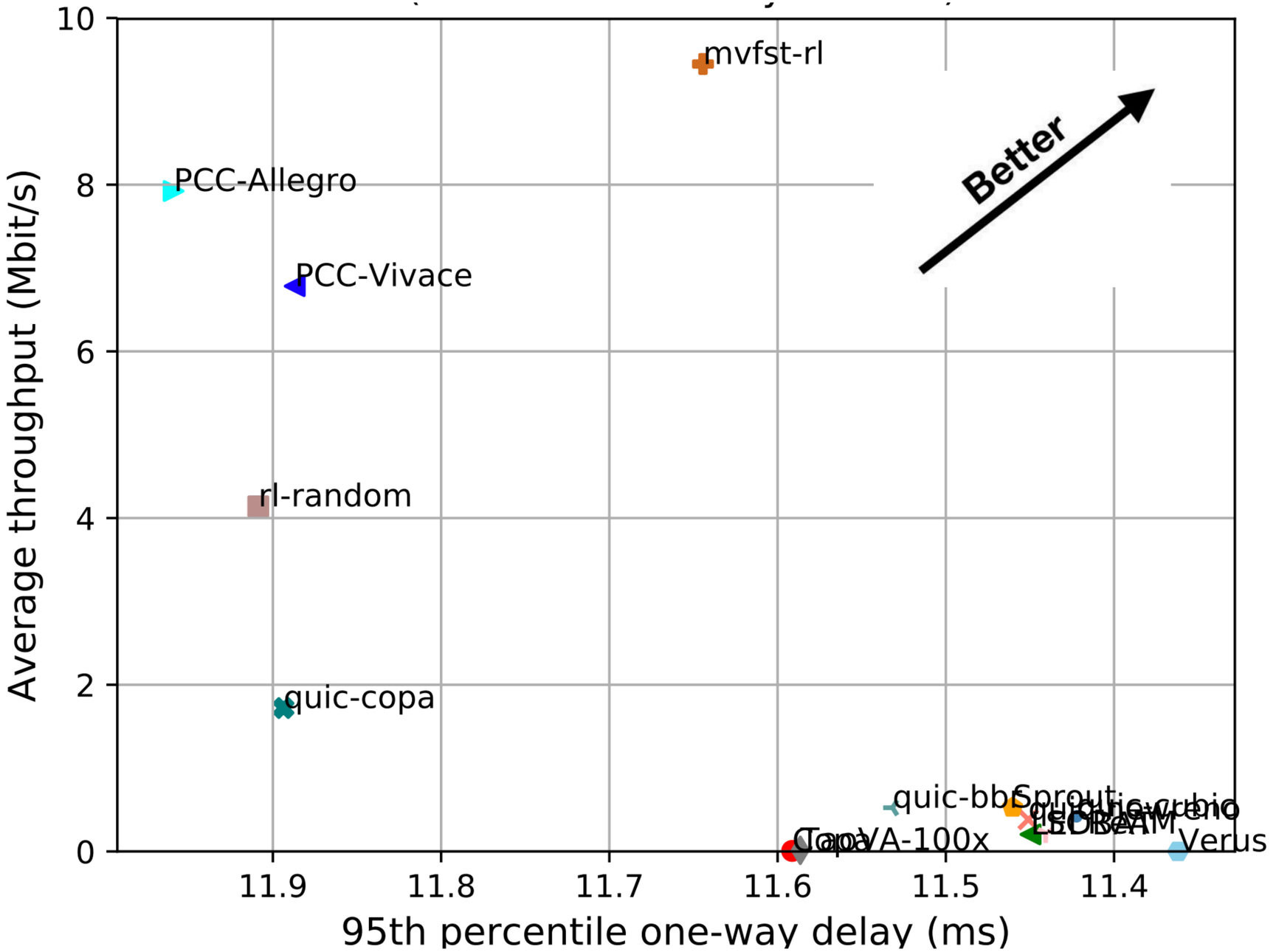


mvfst-rl:
- Async Actor-Learner
- Async Environment-Actor

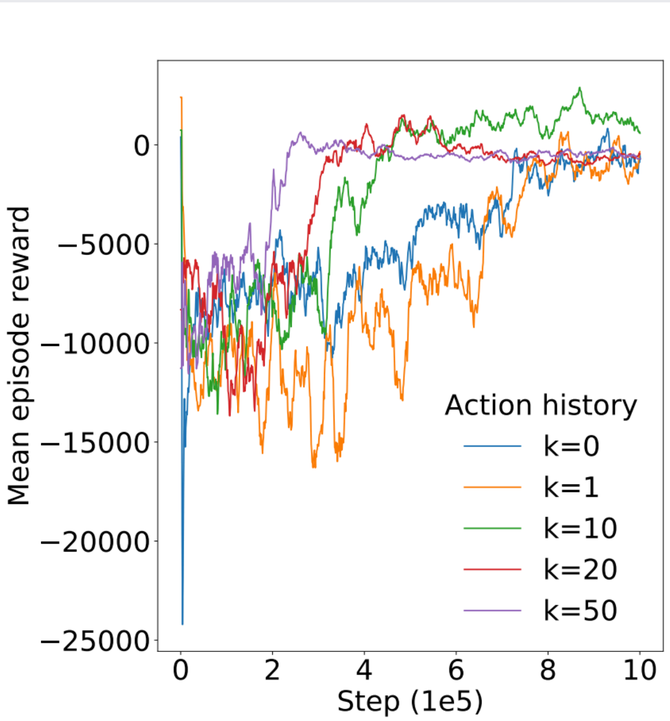
Results



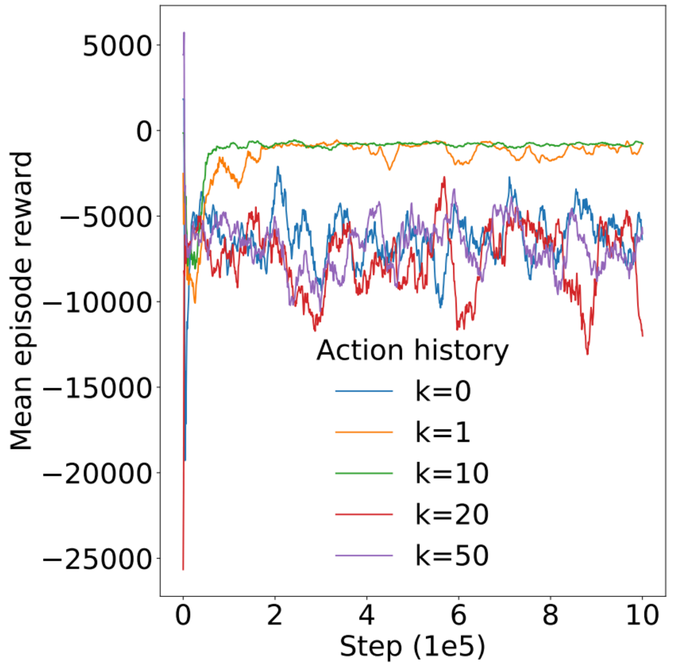
(a) Nepal to AWS India (calibrated)



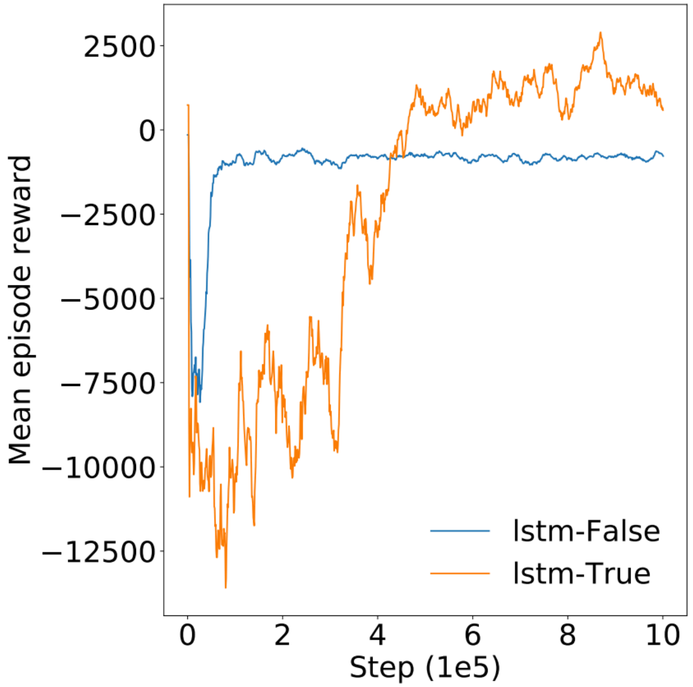
(b) Token-bucket policer (unseen environment)



(c) With LSTM



(d) Without LSTM



(e) Action history = 10

Challenges

- Generalization of single policy to multiple network scenarios
  - Reward scales vary significantly
  - 100 Mbps vs 0.5 Mbps networks
- Policy lookup relatively more expensive
- Online learning

github.com/facebookresearch/mvfst-rl

Workshop on ML for Systems at NeurIPS 2019, Vancouver

Feedback?