RDMA : Congestion in Data Center

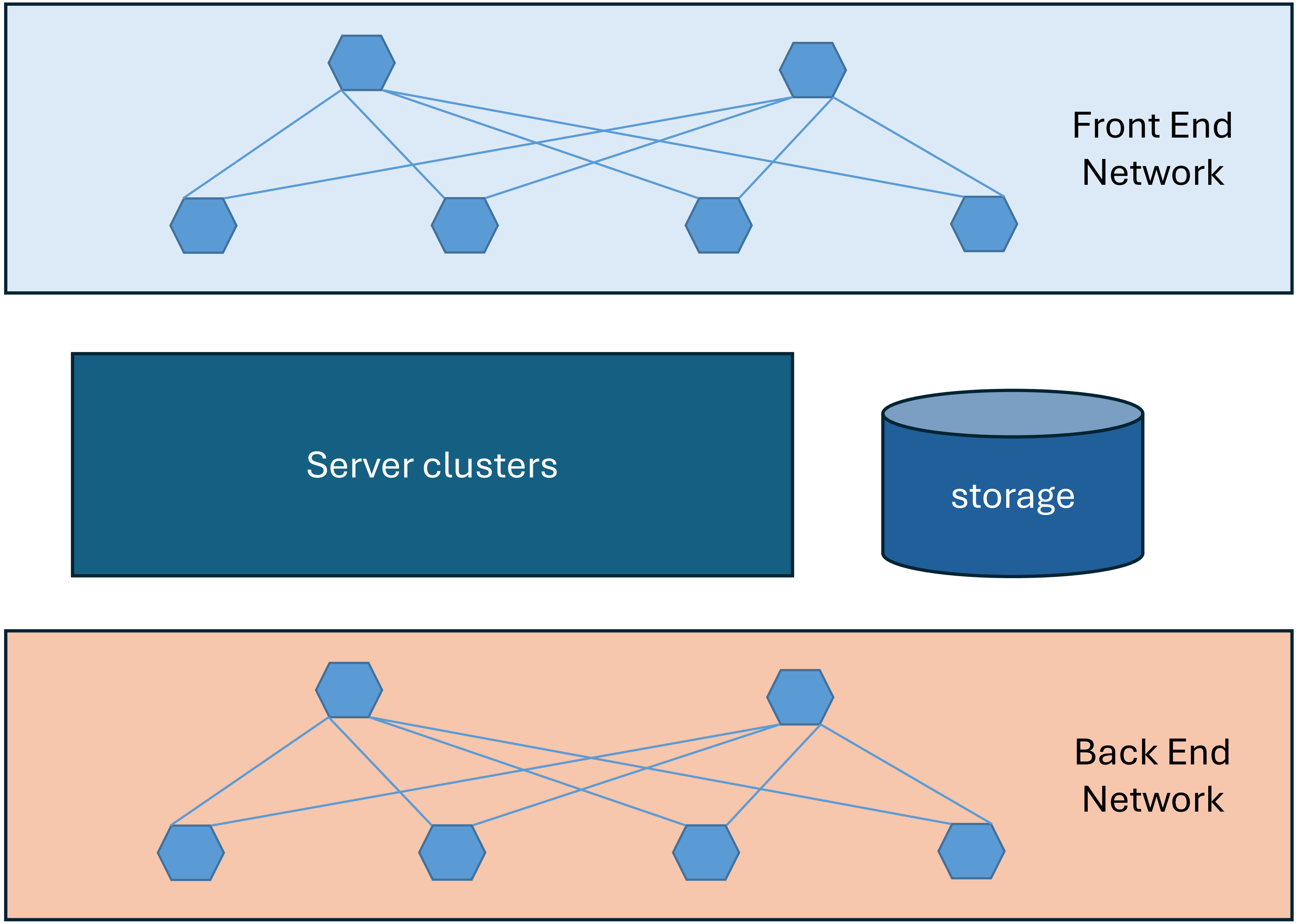
Vivek Kashyap

Ping Yu

Agenda

- Data Center Transformation
- Congestion Signals
- Survey of CC in DC
  - Some examples
  - Factors for coexistence
- Proposal
- CC Reinforcement Learning

Data Center Networks



Front End Network:

Connect to outside world and run various traffic

Back End Network:

Low latency networking

Storage – processed data

Data Center transformation

- New workload with new demands: High speed, low latency
  - AI/ML: systolic, bursty, high bandwidth, large messages, latency sensitive
  - HPC: high packet rate, small message, latency sensitive,
- Existing: Storage
  - Needs to be faster to meet the needs of AI/ML and HPC
- Protocol of choice: **RDMA**
  - Reduced CPU load
- Challenge: Low tail latencies and High Utilization
  - Network delays waste expensive compute time!
    - Lots of flows, hot-spots, incast
  - **Congestion Control**
- Technology of choice: **Ethernet**
  - Open, Cheap, Omnipresent, Default
  - **Lossy!**

RoCEv2: RDMA over Converged Ethernet

- RoCE implements IB semantics over Ethernet:
  - Go-Back-N semantics on loss : retransmissions, inefficient
  - PFC: lossless, HoL blocking, victim flows, deadlocks: Slow convergence
  - DCQCN: complex, fine tuning, unstable
  - ECMP: hash collisions

- Gap: Low utilization, high latencies
- Desire: High utilization, low latencies

- New investigations leveraging TCP research (and other). Some examples:
  - Intel Gaudi Extensions
  - Nvidia Extensions
  - Amazon SRD
  - Alibaba HPCC
  - Google Falcon
  - Tesla TTPoE
  - **Ultra Ethernet Consortium**

Congestion Control Signals

Network Signals

- Loss: TCP Cubic, Reno
  - AIMD response
- Notification: DCTCP, DCQCN
  - ECN – receiver sends feedback to sender. Rate control
- Delay: Swift, TCP Vegas, TIMELY
  - Lagging indicator of congestion
  - RTT and Rate control
  - Swift: pacing for incast

Switch signals

- ECN: AQM – at egress
  - Leading indicator of congestion
  - What of signal on ingress to queue? “Bolt”
- INT: HPCC
  - Get network information
    - Queue length, transmitted bytes
    - Link capacity, timestamp

May not always work together. Do we always isolate?

- Fairness: Rate = BW/N
- Completion time
- Scale automatically

Other factors: Multi-pathing/LB, OOO data placement, SACK

Three Examples

Swift

- Delay
  - Adapt rates to a target e2e delay
  - Accurately measure delays with timestamps
  - Use pacing of packets under extreme congestion
  - No Switch involvement

SMaRTT

- ECN + Delay + Trimming

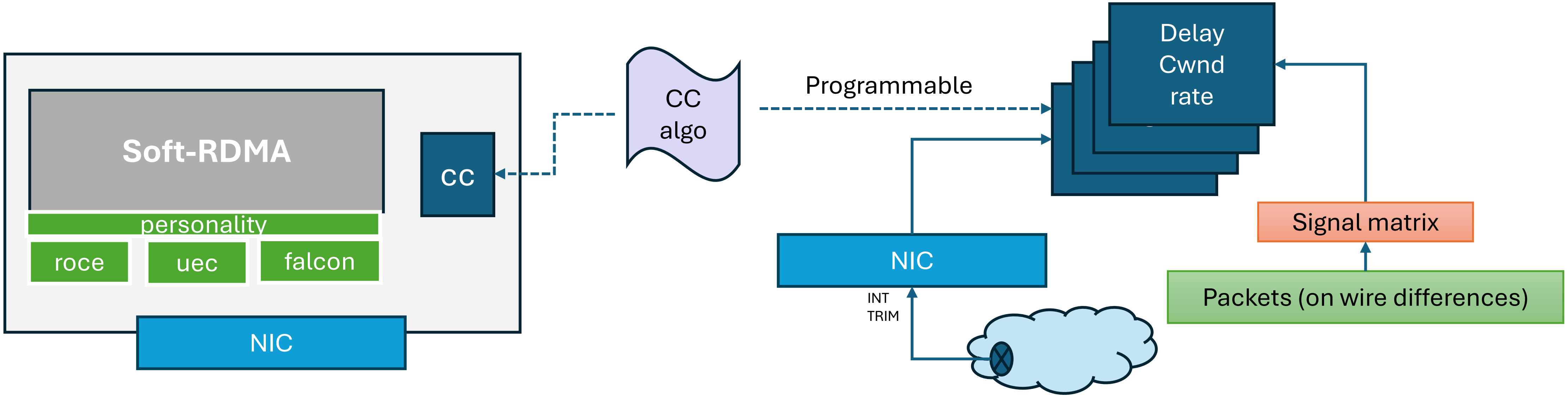
  !ecn, delay < target_delay
  proportional increase
  !ecn, delay > target_delay
  fair increase
  Ecn, delay >= target_delay
  multi decrease
  ECN, delay < target_delay
  fair decrease
- Switch involved: ECN and Trimming

HPCC

- In-band telemetry
  - Queue length, transmitted bytes
  - Link capacity, timestamp
- Quick convergence to high util b/w
- Switch Involved

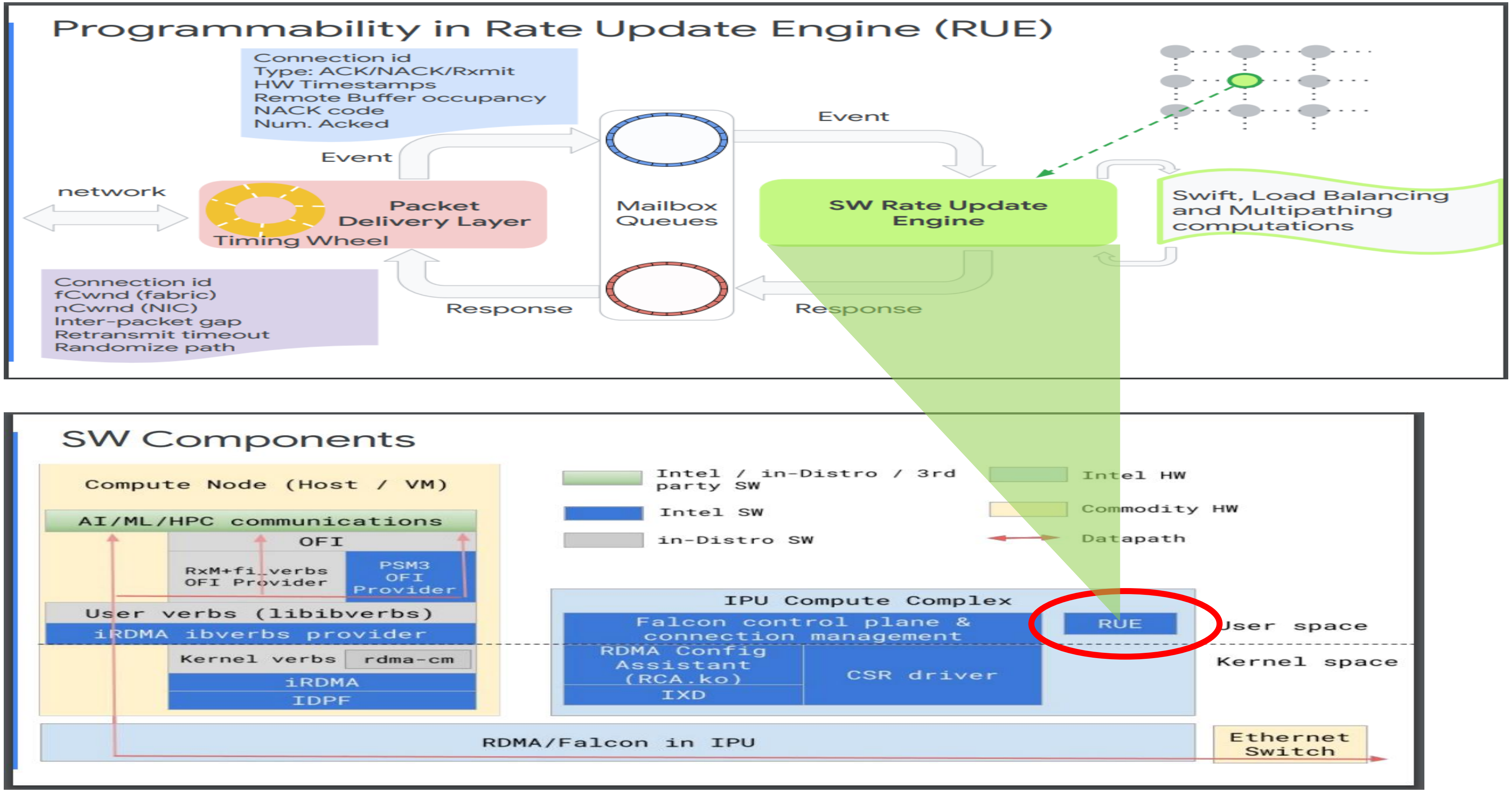
- Idealized simulation models do not capture all the factors
- Need practical insight into the workloads or their interaction
- Investigate co-existence
  - ECN indication causes decrease? Delay based fills the gaps ?

Real-world Plug and play CC algorithms



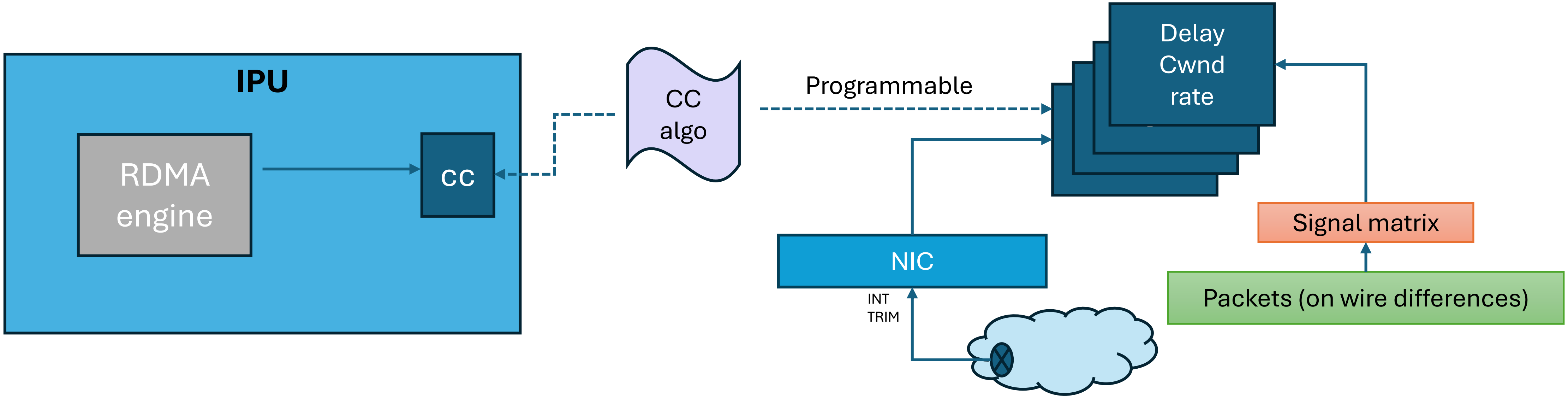
- **Proposal**: Soft-RDMA stack with CC plugin interface
  - Interaction on ‘real’ network
  - Functional verification
  - On foundation NICs
- Support a set of signals to the CC algorithm
  - Packet on wire may differ: RoCEv2, Falcon, UEC, SRD
- Default set of signals from the NIC
  - Delay, INT

Programmable CC: Falcon implementation



Source: netdev0x18: Falcon protocol, Nandita Dukkipati, Google

IPU: Plug and play CC algorithms



- Support a set of signals to the CC algorithm, programmable CC
  - Packet on wire may differ: RoCEv2, Falcon, UEC, *New algos*
- Default set of signals from the NIC
  - Delay, INT

RL for Congestion Control

AI for networking

Congestion Control with Deep Reinforcement Learning

- Fully-automated mechanism to train by interacting with real world network environment
- State
  - Observed from the environment
- Reward
  - Strategy to produce an action
  - Train a policy and maximize the reward
- Action
  - Congestion window
  - Sending rate

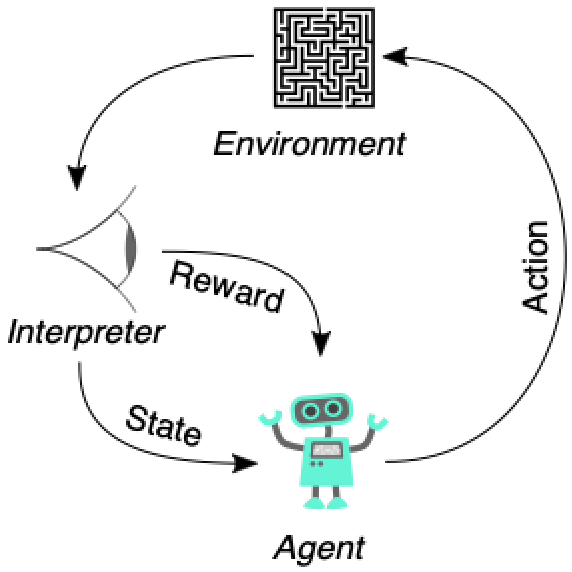
<u>Challenges</u>

Heuristics vs Machine Learning

- Clear rule
- Trial exploration

- User space vs Kernel space interface
- Computation overhead

Aurora: an example implementation of RL for congestion control

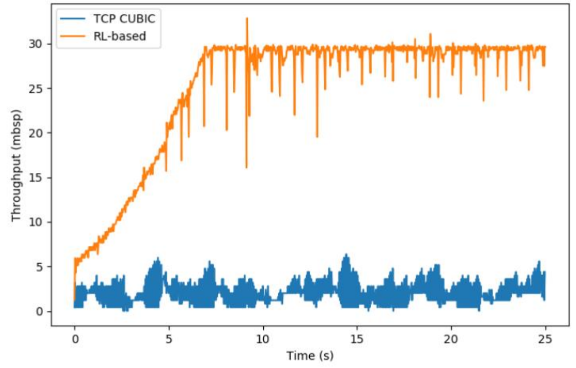


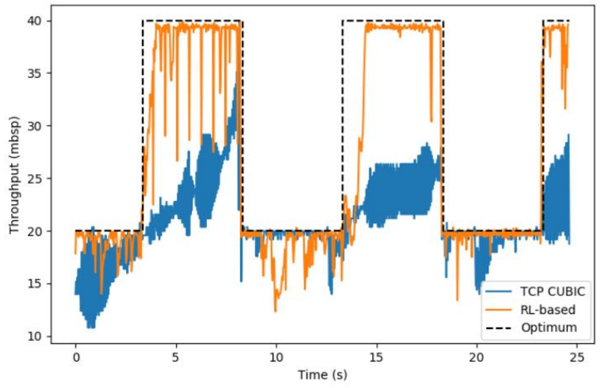
Proceedings of the 36th International Conference on Machine Learning

- Distinguishing non-congestion loss from congestion induced loss.
- Adapting to variable network conditions.

Action: changes to send rate
State: bounded histories of network statistics
(i) latency gradient (ii) latency ratio (iii) sending ratio
$S_t = (v_{t-(k+d)}, \ldots, v_{t-d})$
Reward: defined on scenario. Some for latency and some for throughput

Algorithm: PPO algorithm





RL appliance for real case

- Two level of control
  - First Control
    - TCP algorithm based on normal ack-based logic
  - Second Control
    - DRL agent calculates a new *cwnd* for the session
    - Choose from Heuristic and Machine learning. Utility is key.
    - A user space to kernel space notify mechanism is implemented
- Advantage
  - Per packet inference computation is huge, and RL based congestion window could be a high level direction for a batch of packets
  - Intel AMX based RL optimization

Software Architecture

- Marry Device memory TCP with RL CC
- Combine Heuristic with Machine Learning
- Interface between user space & kernel space
  - Get congestion data from Kernel
  - Set *cwnd* to kernel
- Optimized library to RL