PCC Vivace: Online-Learning Congestion Control

Mo Dong, Tong Men, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, Michael Schapira

NSDI 2018

Outline

- **Background**
- Design
- Implementation & Evaluation
- Review

Background – Recap PCC

- Treat congestion control as black-box optimization
- Sender:
  - **Try a rate**
  - **Measure performance**
  - **Compute utility**
  - **Adjust rate**

Background – Recap PCC

- **Strength:**
  - No hardcoded rules (unlike TCP)
  - Works across diverse networks
- **Weakness:**
  - Ad-hoc utility function
  - Fixed step-size rate control
  - Slow convergence
  - Not latency-aware

Motivation

- TCP:
  - Brittle
  - Heuristic-based
- PCC:
  - Slow convergence
  - Poor stability/reactivity tradeoff
  - High latency/bufferbloat
- **PCC and BBR still fail to achieve optimal tradeoffs**

Outline

- Background
- **Design**
- Implementation & Evaluation
- Review

Key Idea of PCC Vivace

- **Bring in online learning theory**
- Model congestion control as:
  - Repeated decision-making problem
- Use:
  - **Online convex optimization**
  - **Gradient ascent**
  - **No-regret learning**

Conceptual Mapping

- **Bring in online learning theory**

Networking	Learning Theory
Sending rate	Action
Throughput/loss/RTT	Feedback
Utility	Reward
Rate update	Learning step

Sender learns best rate via trial-and-error

What is No-Regret Learning?

- Definition:
  - Over time, performance ≈ best fixed decision in hindsight
- Why it matters:
  - **Works under uncertainty**
  - **Robust to dynamic networks**
  - **Enables multi-sender convergence**

PCC Vivace is the first congestion control scheme with formal no-regret guarantees

High-Level Architecture

- Same as PCC, but redesigned:
  - **Utility function**
    - Key contribution of PCC vivace
  - **Rate control algorithm**

Utility Function

$$u\left(x_i, \frac{d(RTT_i)}{dT}, L_i\right) = x_i^t - bx_i\frac{d(RTT_i)}{dT} - cx_i \times L_i, \quad (1)$$

- Reward: throughput x_i
- Penalize:
  - Latency gradient
  - packet loss L_i

Why RTT Gradient?

- Not absolute RTT!
- Example:
  - Queue already large → RTT always high
  - Reducing rate still increases RTT
- **RTT change (gradient)**
  - Enables correct decisions within a single interval

Properties of Utility Function

- With proper parameters:
  - Converges to fair equilibrium
  - Avoids bufferbloat
  - Tunable tradeoffs
  - Supports heterogeneous policies

Rate Control: Gradient Ascent

- Given current rate r, how should we change it?
- In original PCC:
  - Increase/decrease by a fixed percentage
  - Problem: step size is **blind**
- In Vivace:
  - Use **gradient of utility**
  - **Step size reflects *how much improvement is possible***

Rate Control: Gradient Ascent

- So it **estimates the gradient empirically**:
  - Try slightly higher rate: $r(1+\epsilon)$
  - Try slightly lower rate: $r(1-\epsilon)$
- Measure utilities:
  - $u_1 = u\big(r(1+\epsilon)\big)$
  - $u_2 = u\big(r(1-\epsilon)\big)$
- Then compute:
  - $\gamma = \frac{u_1 - u_2}{2\epsilon r}$
  - $r_{new} = r + \theta \cdot \gamma$

Why This is Better Than PCC?

- **PCC**
  - Fixed ε:
    - Too small → slow convergence
    - Too large → oscillation
- **PCC Vivace**
- Step size proportional to **slope of utility curve**
  - Big gradient → big step
  - Small gradient → fine tuning

But Naive Gradient Ascent Fails in Networks

- If you implement directly:
- **Problem 1: Huge jumps**
  - Gradient may be large
  - Rate jumps from 1 Mbps → 500 Mbps
- **Problem 2: Shrinking step size**
  - Classical theory: $\theta \to 0$
  - Leads to slow adaptation

PCC Vivace Fix

- **Confidence Amplifier**
  - If decisions are consistent, increase step size
- **Dynamic Change Boundary**
  - Gradient may be noisy → huge wrong updates
  - Cap rate change
- **Noisy Gradient Handling**
  - Real networks ≠ clean math.
    - Linear regression for RTT gradient
    - Low-pass filtering
    - Double-check abnormal results

Outline

- Background
- Design
- **Implementation & Evaluation**
- Review

Implementation

- PCC Vivace is implemented as a user-space based on UDT
- **UDP-based Data Transfer Protocol (UDT)**, is a high-performance data transfer protocol designed for transferring large volumetric datasets over high-speed wide area networks. Such settings are typically disadvantageous for the more common TCP protocol.
- Easy to modify, widely adopted by many learning-based CC research works

Evaluation

- Consistent High Performance
  - Random Loss
  - Satellite Links
  - Shallow Buffer
  - Reaction to Changes
- Convergence Properties
- TCP Friendliness
- Flexible Convergence Equilibrium
- Real-world Applications

Random Loss

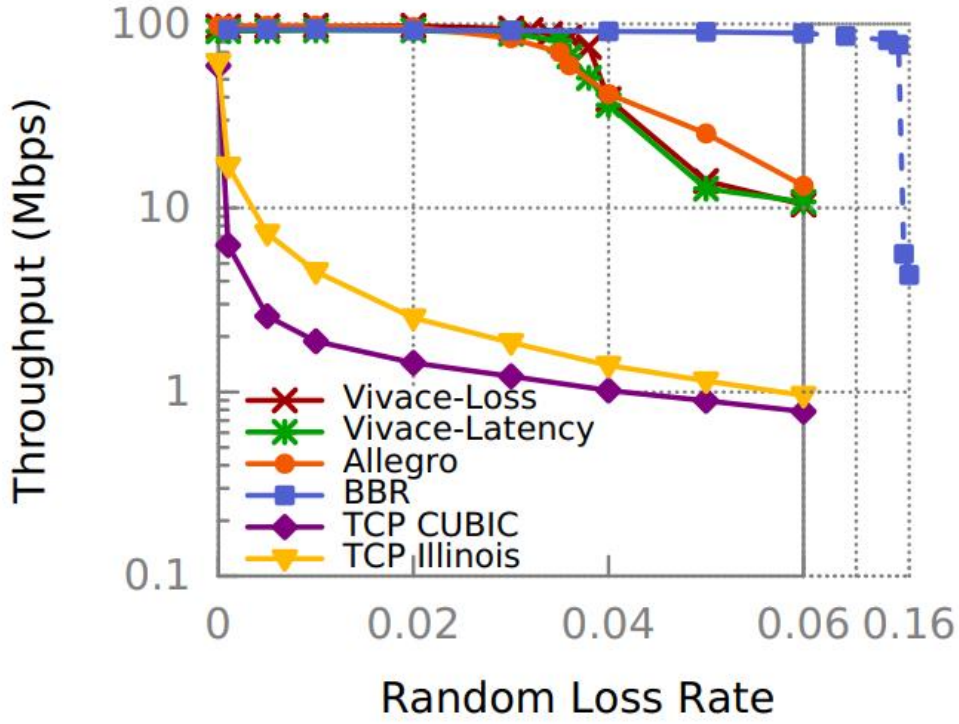


Figure 2: Random loss resilience

Satellite Links – Long RTT

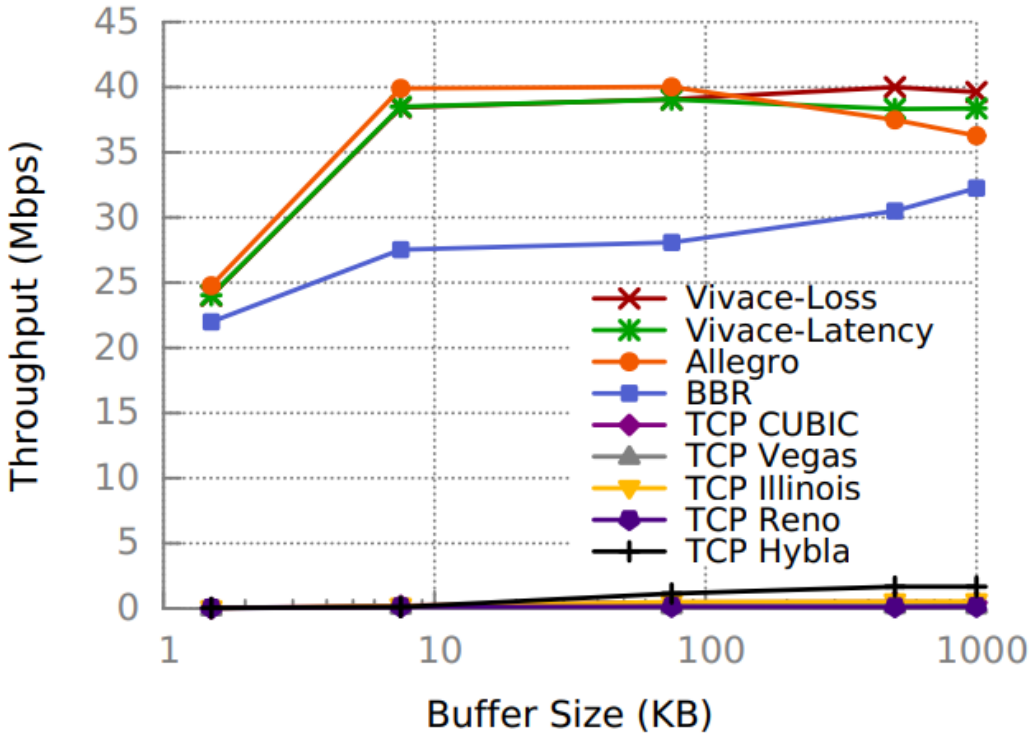
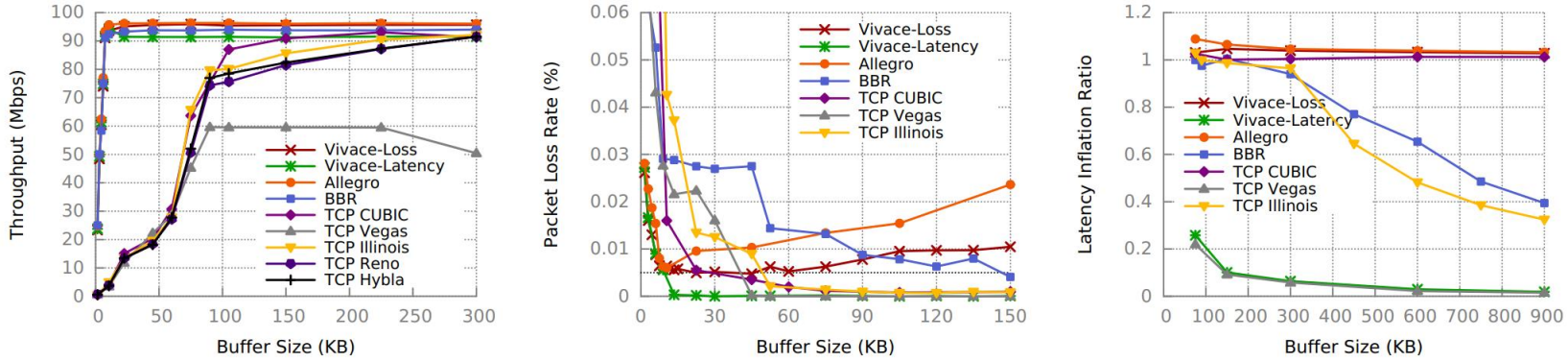


Figure 3: Long RTT tolerance

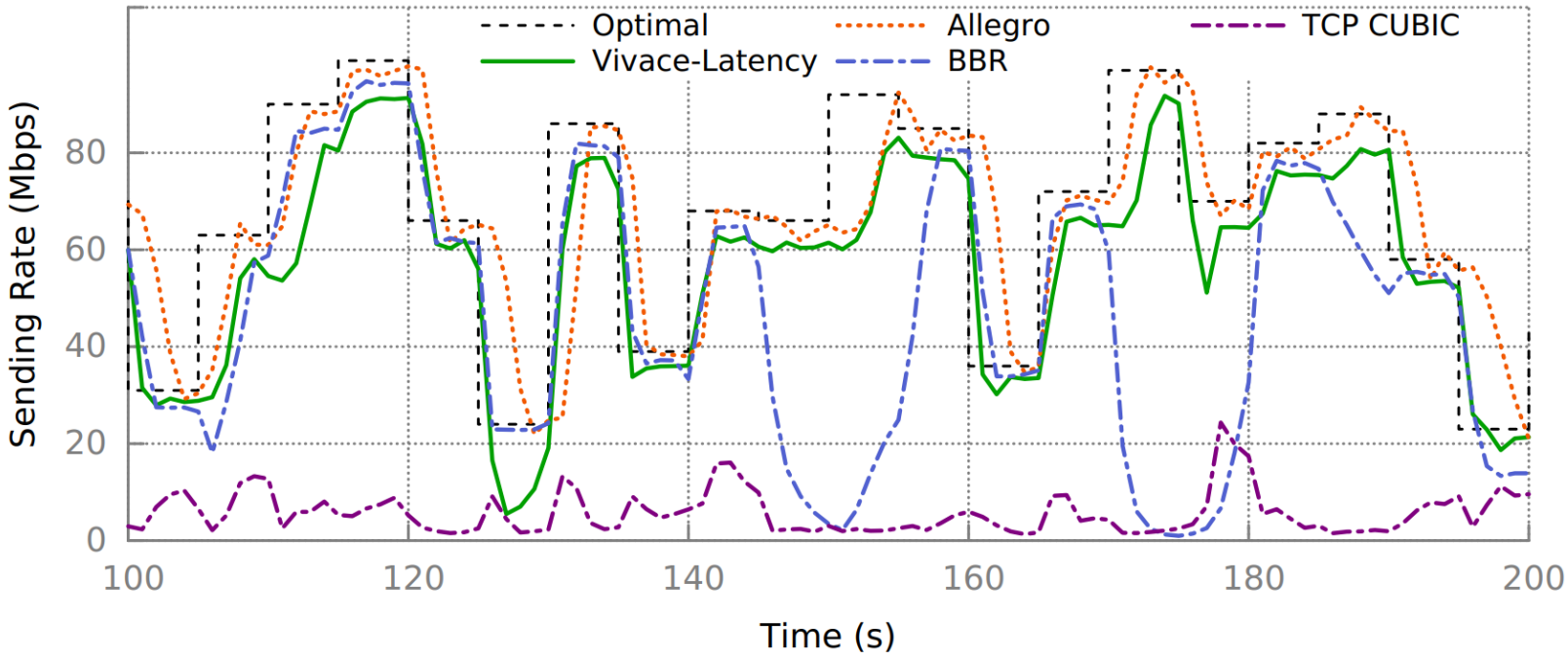
Shallow Buffer – DCN



(a) High capacity (b) Low packet losses (c) Negligible RTT overflow

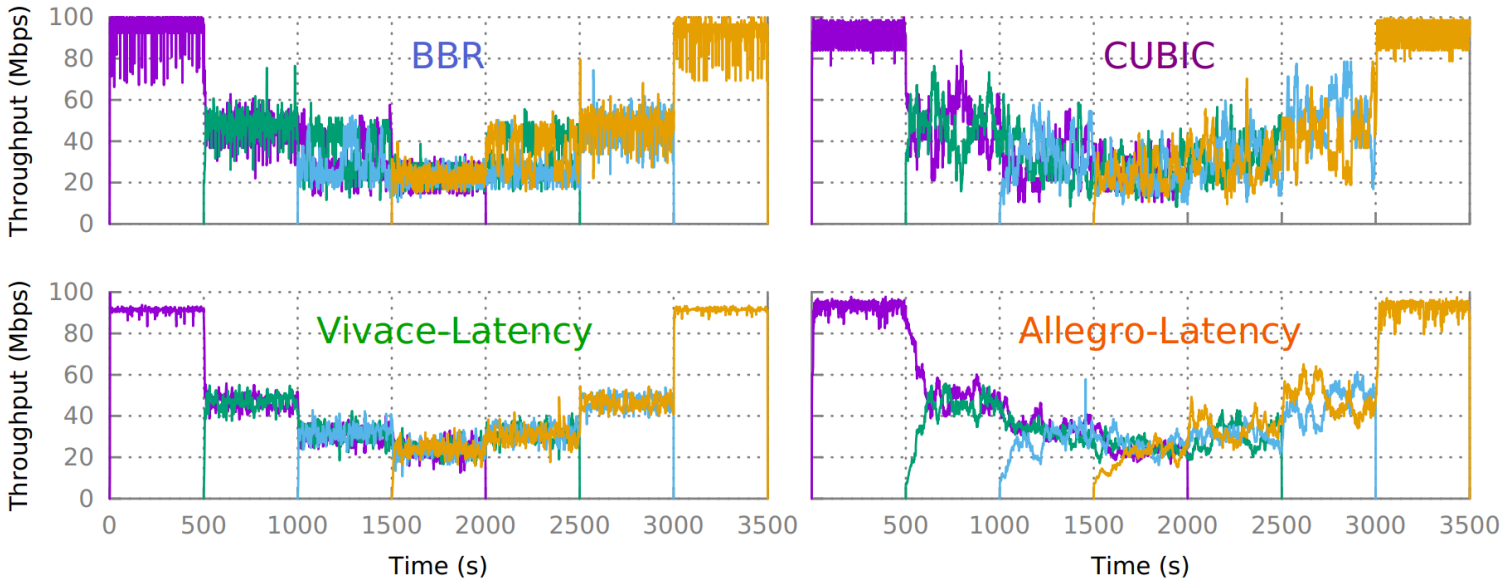
Figure 4: Vivace can achieve better performance with shallow buffer

Reaction to Changes

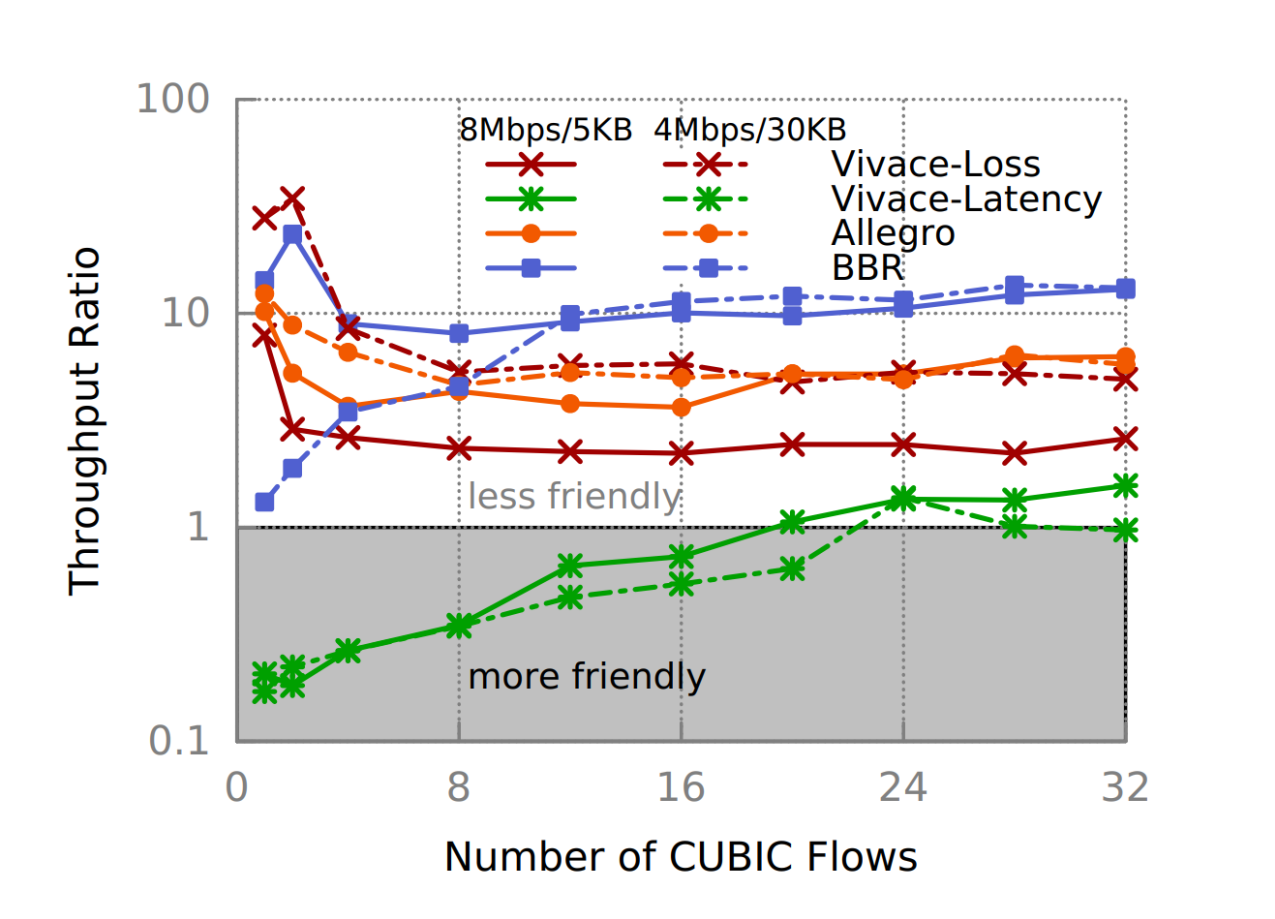


(c) More responsive latency-sensitivity

Convergence Speed and Stability



Improved Friendliness to TCP



Flexible Convergence Equilibrium

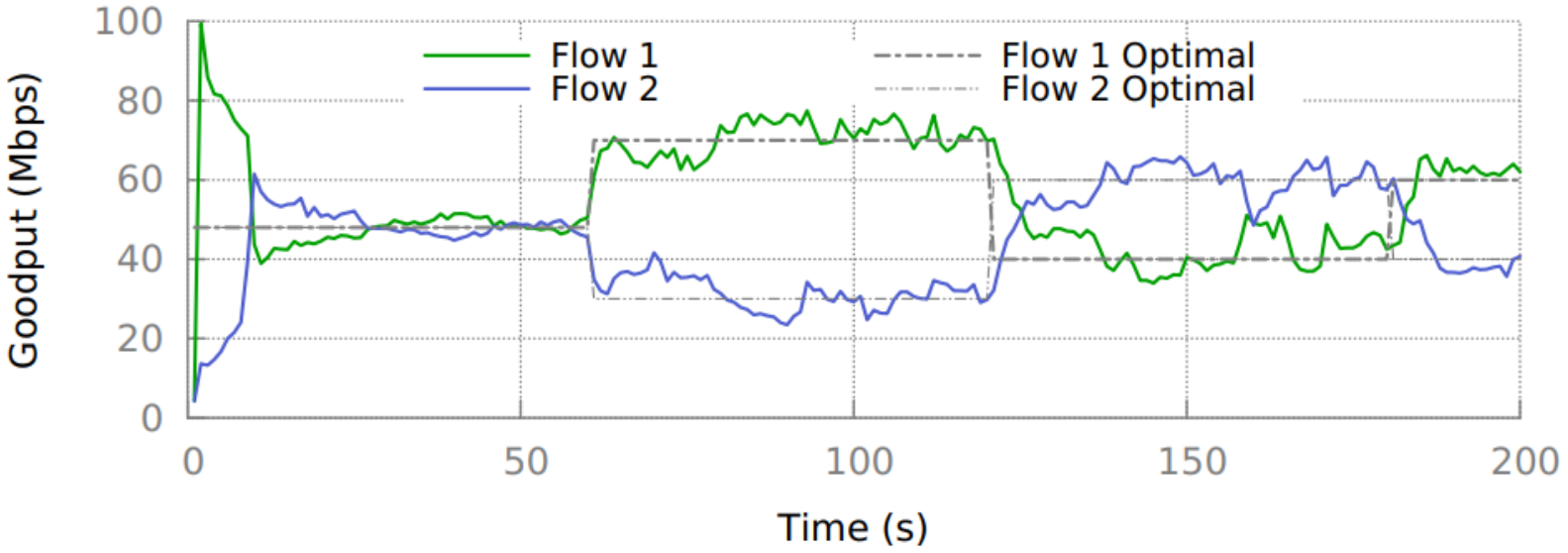
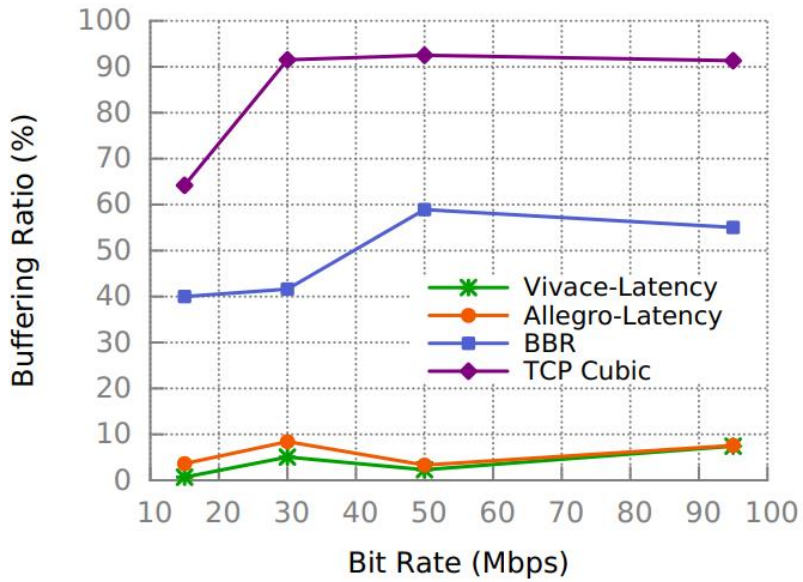
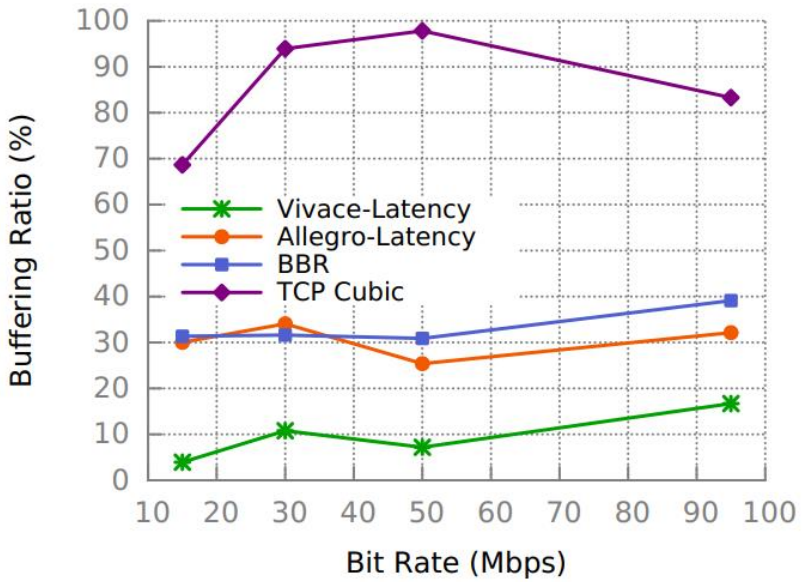


Figure 11: Flexible equilibrium by tuning utility knobs

Benefits in the Real World Video Streaming

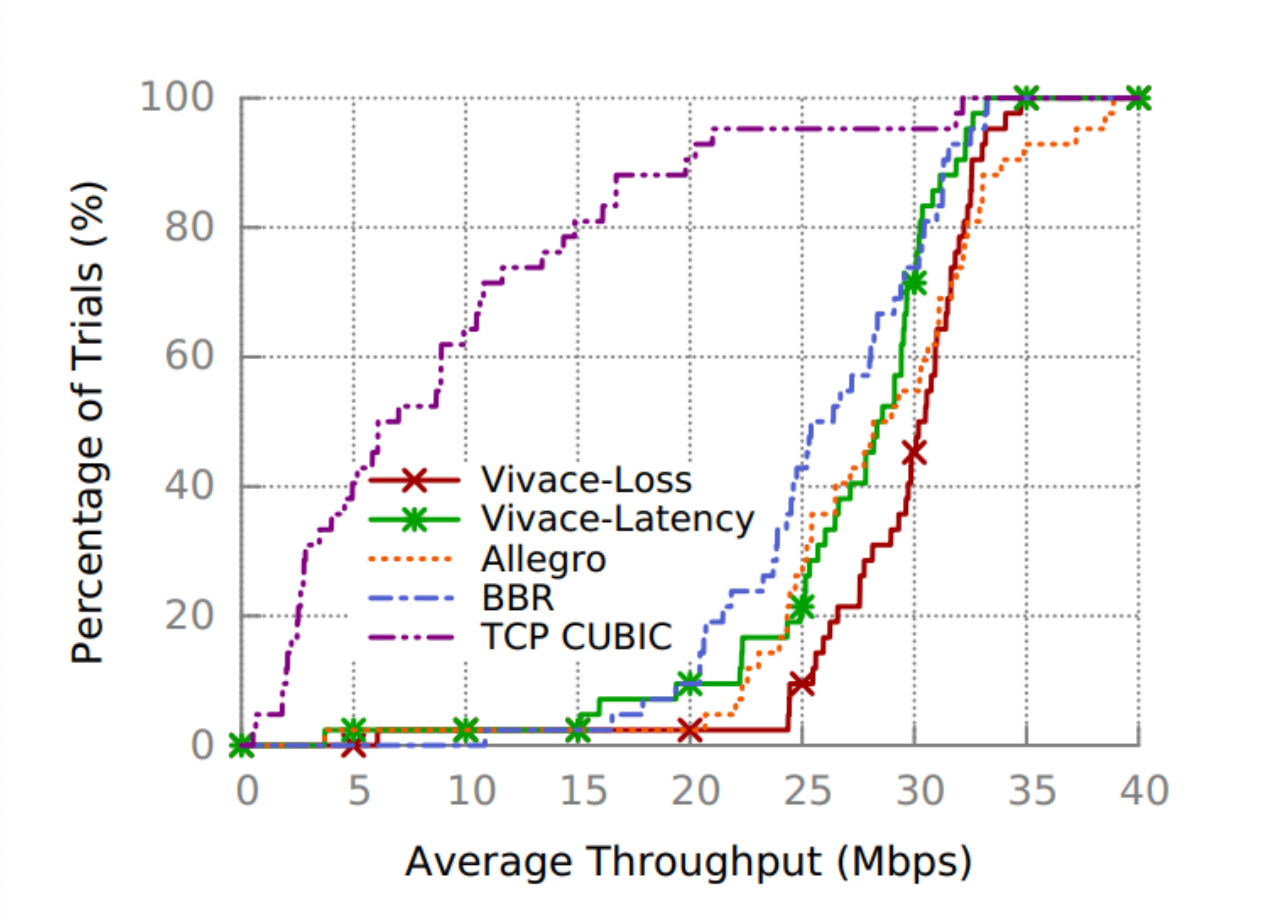


(a) Video streaming with varying latency



(b) Multiflow video streaming

Benefits in the Real World Wild Internet



Outline

- Background
- Design
- Implementation & Evaluation
- **Review**

Review

- PCC Vivace improves the original PCC
  - New utility function
    - Online learning theory
    - No regret learning
  - Rate control algorithm
    - Gradient descend

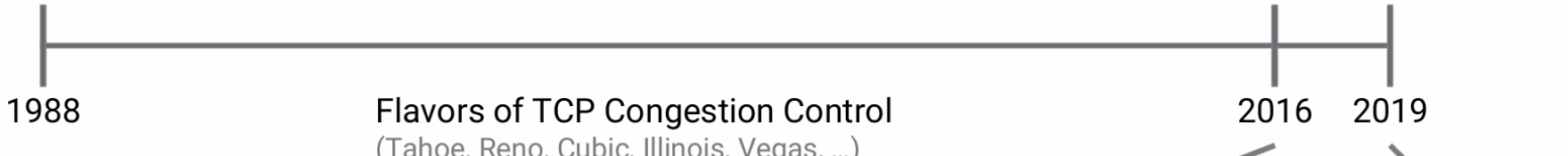
A Deep Reinforcement Learning Perspective on Internet Congestion Control

Nathan Jay, Noga H. Rotman, P. Brighten Godfrey,
Michael Schapira, Aviv Tamar

ICML 2019

Revisiting Congestion Control

Congestion Control Timeline



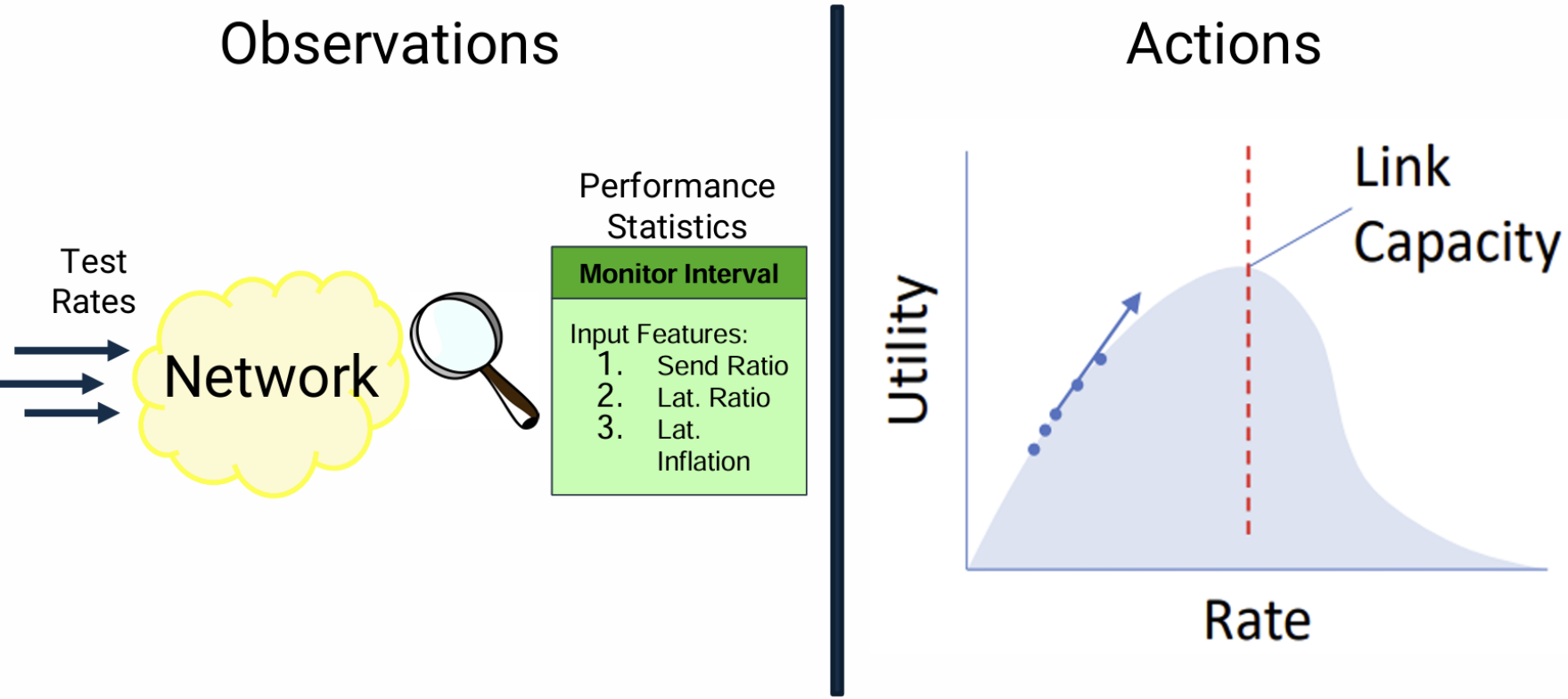
Flavors of TCP Congestion Control
(Tahoe, Reno, Cubic, Illinois, Vegas, …)

- Same network model
- Same action space
- Slightly different control algorithms

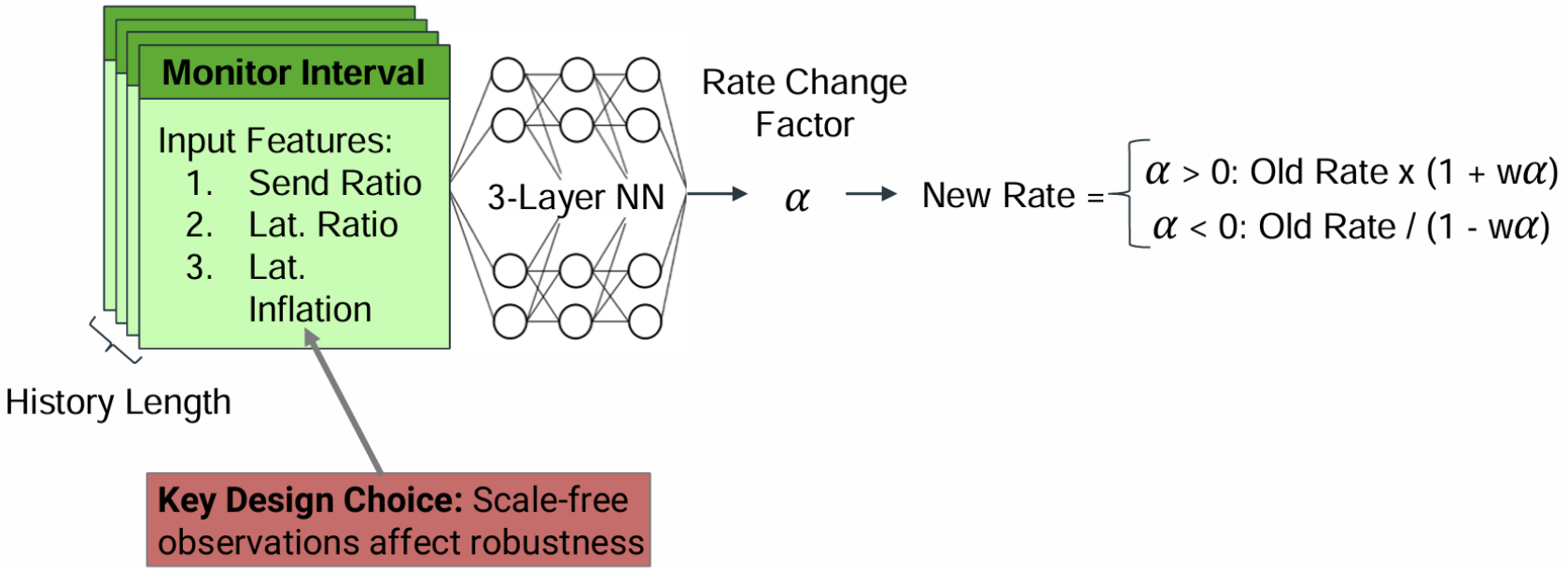
Introduction of **QUIC**, replaces significant amount of Google traffic.

- New models
- New action space (packet pacing added to Linux)
- Novel control algorithms and research (BBR, Copa, PCC)

Reward-based architecture: PCC



Agent Architecture



Training/Testing Environment

Training Environment:

- Simulated network
- Each episode chooses link parameters from a range:

Capacity	Latency	Loss	Queue
1 - 6mbps	50 - 500ms	0 - 5%	1 - ~3000pkt

- **Standard gym** at github.com/PCCProject/PCC-RL

Testing Environment:

- Real packets in Linux kernel network emulation
- Much wider testing range:

Capacity	Latency	Loss	Queue
1 - 128mbps	1 - 512ms	0 - 20%	1 - 10000pkt

State-of-the-art Results

Test Description:

- Emulated network, with real Linux kernel noise
- Time-varying link

Aurora is on the Pareto front of state-of-the-art algorithms