Zhuge: Achieving Consistent Low Latency for Wireless Real-Time Communications with the Shortest Control Loop

Presenter: Yunxiang Chi
09/27/2024

Problem Statement

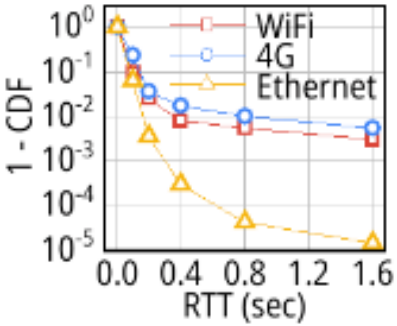
RTC applications' performance suffers from the tail latency

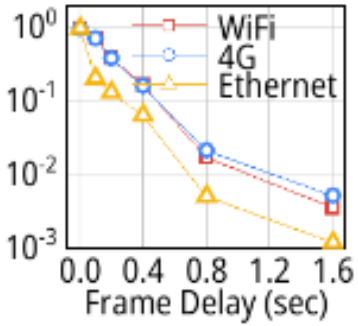
RTC applications requires consistent low latency:

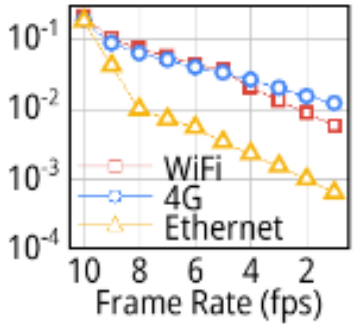
- Video conferencing: <150ms
- Cloud gaming: <96ms

Most of the time, wireless network provide median RTT ~100ms, but the 99th percentile latency is ~400ms

Wireless users encounter 2× more video lags of maintaining the optimal working point with different feedback than Ethernet users. Furthermore, the fps drops is 10× higher than that of wired networks

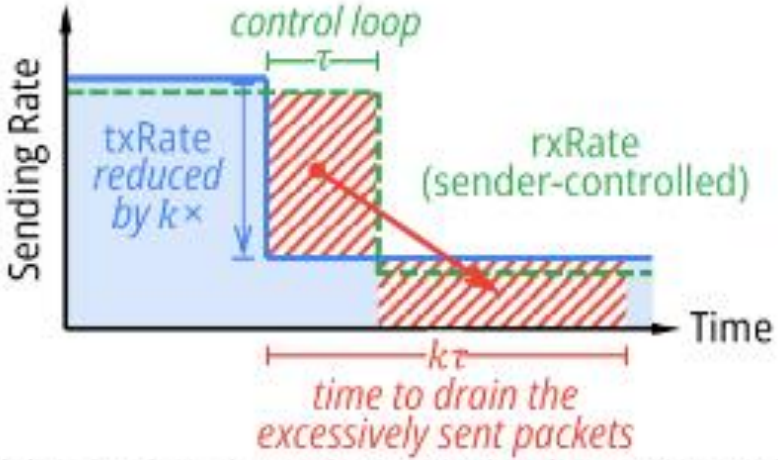




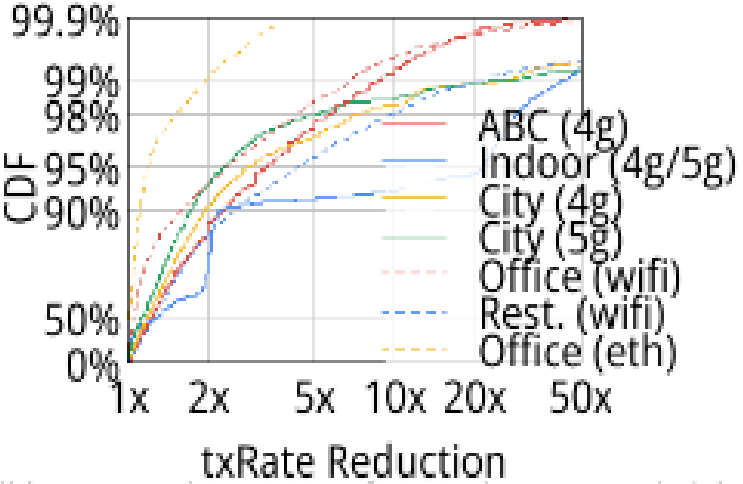


Problem Statement

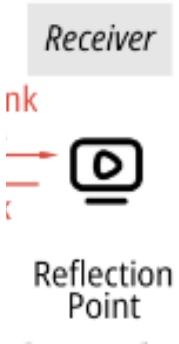
Senders react quite slowly toward transient congestion



(a) The bottleneck queue building up and draining after ABW reduction.



(b) Distribution of wireless available bandwidth reduction ratio.



Problem Statement

Senders react quite slowly toward transient congestion

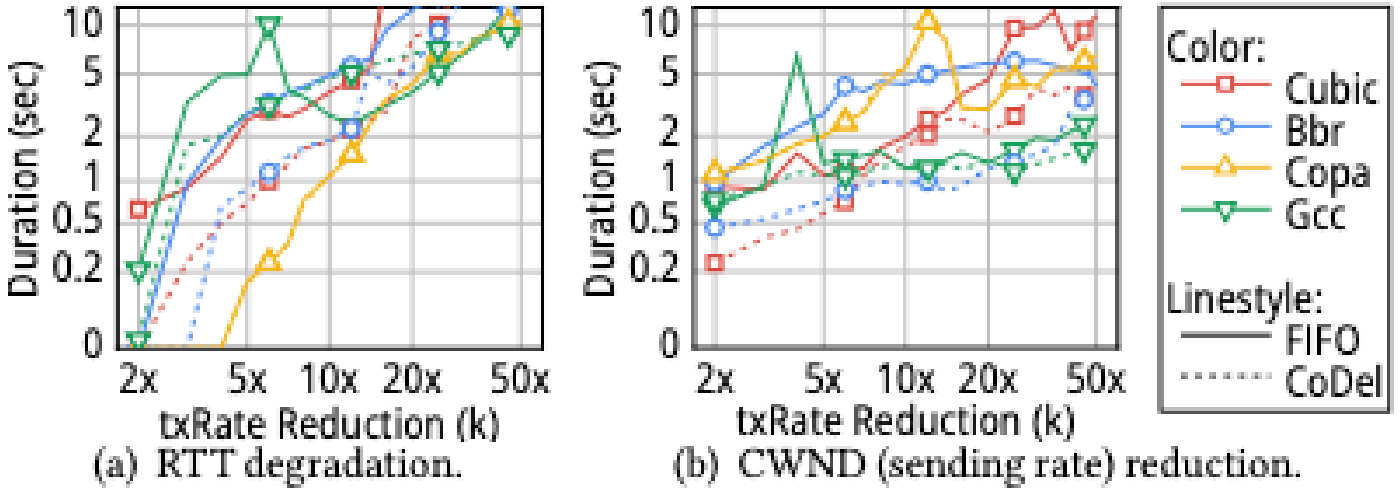
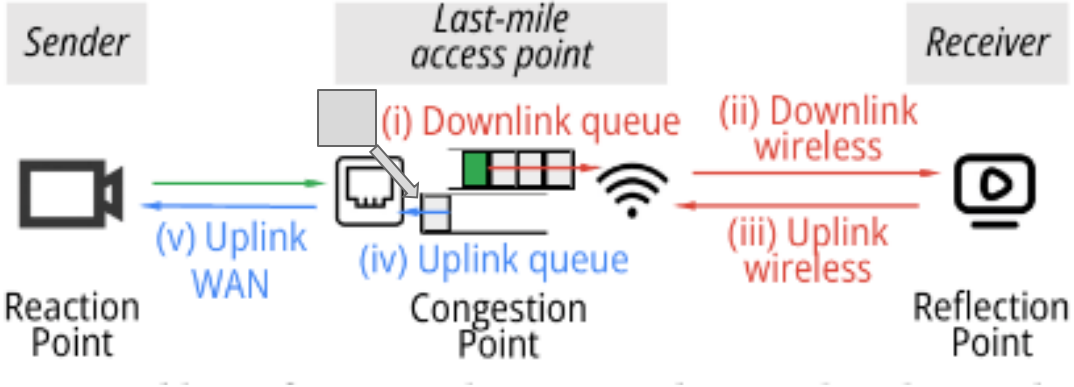


Figure 4: The convergence duration after wireless bandwidth drop for different CCAs and AQMs. *RTT degradation duration is the time when RTT > 200ms. CWND rate reduction duration is the time for CCA re-convergence.*

Insight

Rough Idea:

As the downlink queue starts to fill, AP can modify or delay packets in the uplink queue to allow congestion signals to reach the sender without the delay of the congested bottleneck in the full control path.



Challenges

- **Prediction:** Naively **(num of bytes queued)/(link capacity)**

  But link bandwidth fluctuates quickly: The estimation above -> inaccurate

- **Reporting:** Enabling routers to directly transmit newly defined messages back to senders

  Deployment barriers due to different entities managing APs and senders.

- **Compatibility:** Various existing protocols for signaling methods: BBR: weighted moving average of RTT, Copa: minimum RTT values, RTP: inter-packet timings

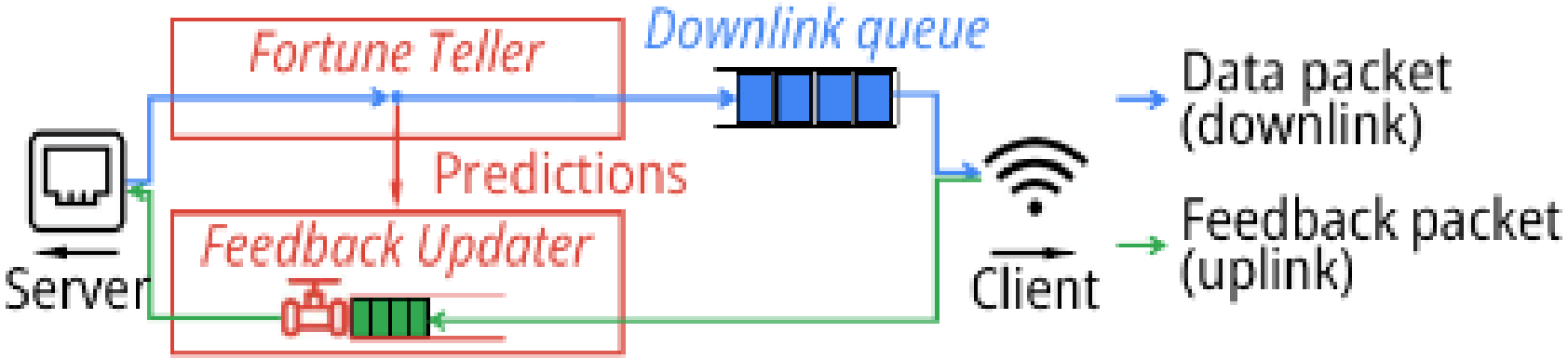
  Require AP to figure out a way to capture all of these factors without modifying sender & receiver

Zhuge’s Design

Zhuge

Key Idea:

Estimates the future latency of a packet upon its arrival at the wireless last mile to obtain network conditions as early as possible.



(Review) Challenges

- **Prediction:** Naively **(num of bytes queued)/(link capacity)**

  But link bandwidth fluctuates quickly: The estimation above -> inaccurate

- ~~**Reporting:** Enabling routers to directly transmit newly defined messages back to senders~~

  ~~Deployment barriers due to different entities managing APs and senders~~.

- **Compatibility:** Various existing protocols for signaling methods: BBR: weighted moving average of RTT, Copa: minimum RTT values, RTP: inter-packet timings

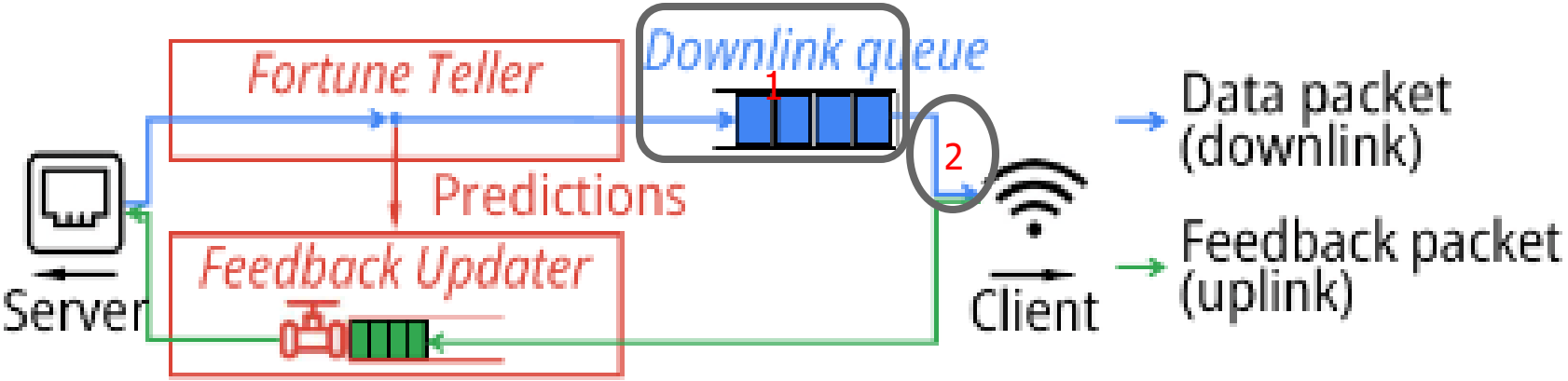
  Require AP to figure out a way to capture all of these factors without modifying sender & receiver

Fortune Teller

Fortune Teller

Strawman solution: (queue size)/(dequing rate)

transience-equilibrium nexus: A short sliding window will lead to drastic fluctuations of the predicted delays due to the bursts of arrivals and departures, and a long window will fail to quickly detect the change of network conditions.



Fortune Teller

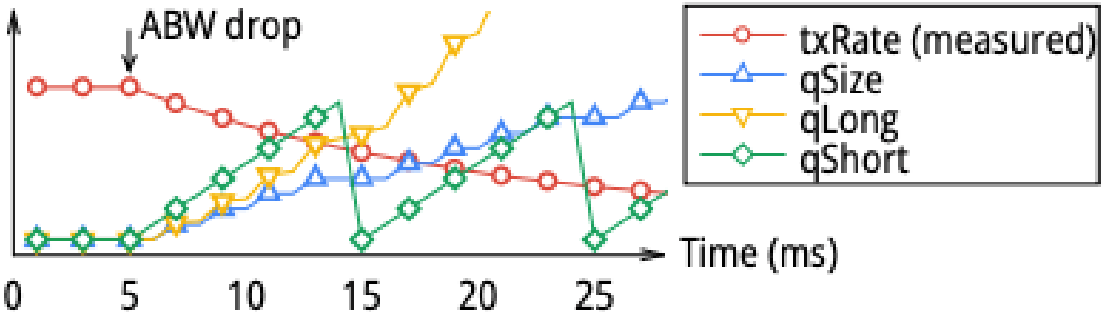
I. Queuing delay
   a. qLong: from the time when one packet arrives, to the time when that packet is at the front of the queue, which is used to cover the latency fluctuation induced by wireless contention and bursty RTC traffic.
   b. qShort: from the time one packet is at the front of the queue, to the time when that packet is finally dequeued
   c. qSize = max(sizeOfPacketsInQueue − maxBurstSize, 0)

totalDelay = qLong + qShort + tx

qLong = ***cur***(qSize) / ***avg***(txRate)

qShort = ***cur***(QFrontWaitTime)

tx = ***avg***(dequeueIntvl)



Fortune Teller

II. Transmission Delay: tx

- Observations:
  - 1. Only one data unit can be in transmission at a time in the wireless channel (aggregated MPDU, or AMPDU). Since multiple will interfere with each other.
  - 2. Recent Linux Mainline have exposed the lower layer queue in the wireless network stack is only used to aggregate multiple packets into a link layer frame

- Design: tx = avg(dequeueIntvl)

  the average interval between packet departures from the network layer queue, with a window similar to txRate

(Review) Challenges

- ~~**Prediction:** Naively **(num of bytes**~~

  ~~But link bandwidth fluctuates quickly~~

- ~~**Reporting:** Enabling routers to dire~~ ~~to senders~~

  ~~Deployment barriers due to different~~

		Protocol	CCA	Application
Out-of-band	(§5.2)	TCP QUIC [36]	PCC [24] BBR [18] Copa [12]	Meta Live [29] Windows 365 [42] Twitch [56] Tencent Start [5]
In-band	(§5.3)	RTP+RTCP [55]	GCC [19] NADA [65] Scream [37]	Google Stadia [23] Zoom [47] Microsoft Teams [54]

- **Compatibility:** Various existing protocols for signaling methods: BBR: weighted moving average of RTT, Copa: minimum RTT values, RTP: inter-packet timings

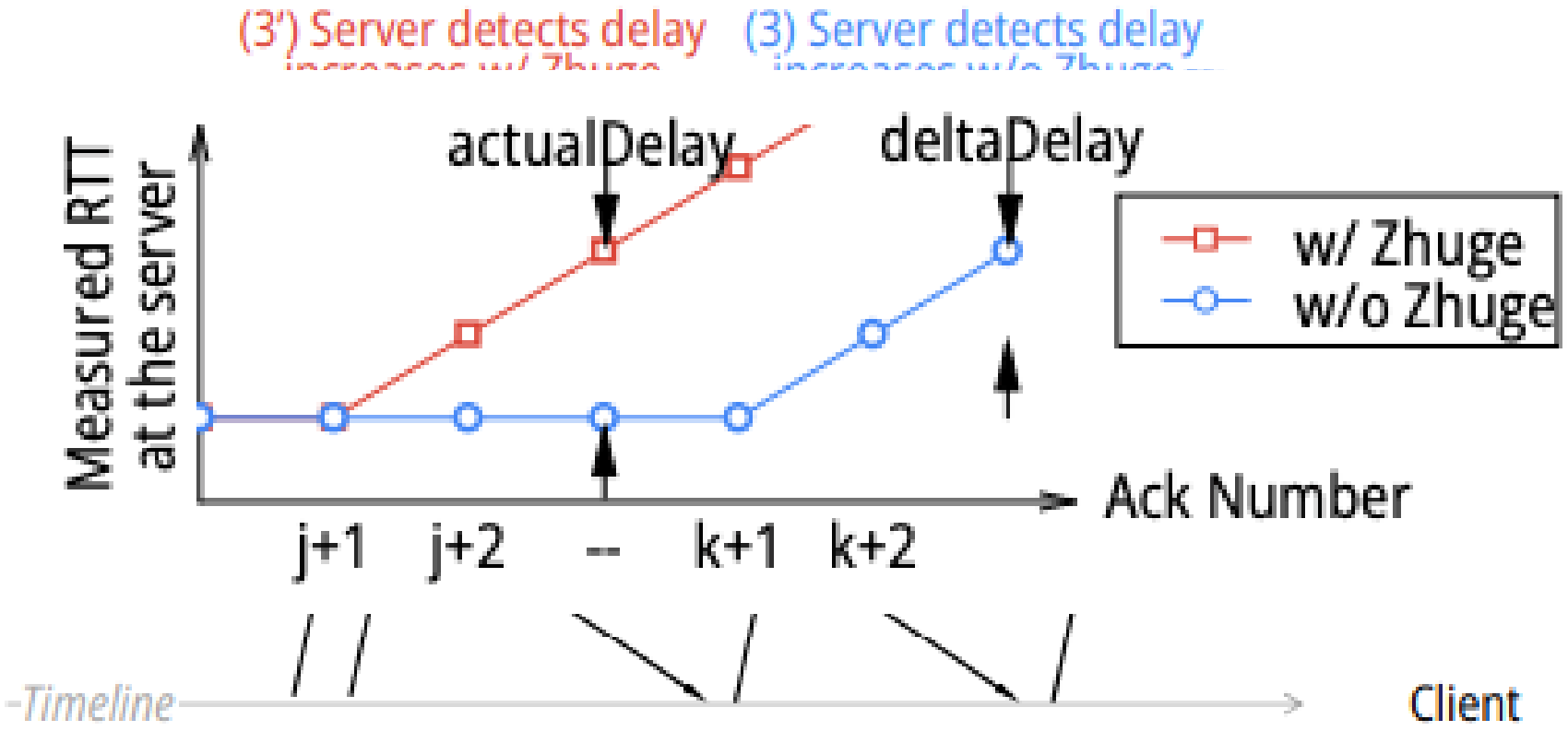
  Require AP to figure out a way to capture all of these factors without modifying sender & receiver

Feedback Updater

Feedback Updater

I. In-band feedback: the feedback information is explicitly written in the payload of a specific type of feedback packets. E.g. RTP&RTCP
II. Out-of-band feedback: sender calculates network conditions itself upon receiving the feedback packets i.e. without info related to rate control. E.g. TCP

Feedback Updater: Out-of-band



Feedback Updater: Out-of-band

- Delivering precise long-term latency in steady state
  - Record relative delay deltas
- Delivering precise short-term latency fluctuation
  - maintain a distribution of recent delay deltas of the downlink data packets, use the sampling value from the distribution when reporting the delay
- Preserving the order of feedback packets
  - Delay token
  - E.g. ACK1, 2 ,3 arrive at 0, 1, 2ms; sampled delay deltas: +5, +2, +4ms
  - Step 1: ACK 1 sent at 5ms, no token created
  - Step 2: ACK2 supposed to send at (1+2)ms, but sent at 5ms, create token of (5-2)ms
  - Step 3: ACK3 supposed to send at 5ms, apply token need to add (4-3)ms to the actual delay, sent at 6ms, doesn’t create token.

Feedback Updater: In-band

- Step1: Zhuge store the predicted delay together with its RTP transport-wide congestion control sequence number in the RTP header upon RTP pkt arrival.
- Step 2: Zhuge will construct a TWCC feedback packet based on stored delays and sequence numbers, and will only send the TWCC packets constructed by itself and drop all TWCC from the client. For other types of feedback packets, Zhuge simply forwards it.

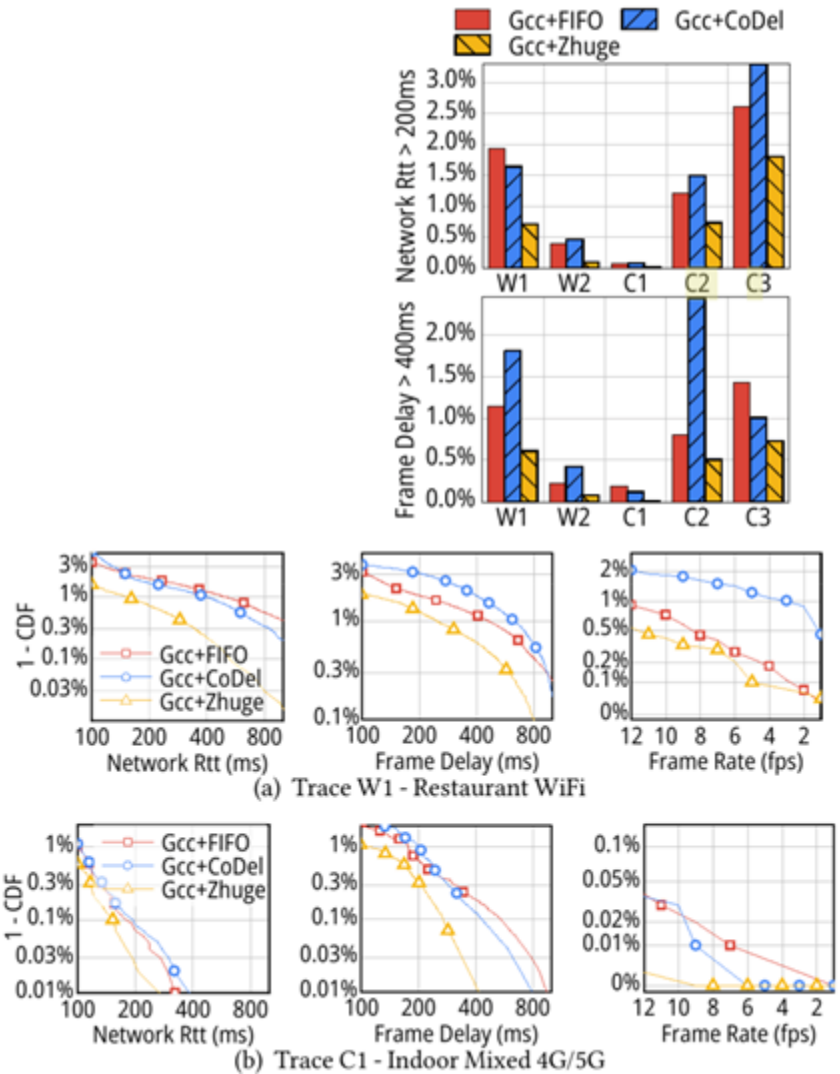
Evaluaion

Setup

- Implementation:
  - NS-3 simulator with simplified video encoder/decoder
  - Testbed using OpenWrt on Netgear WNDR 3800 router (802.11n)
- Video Settings: 1080p, 24fps, average bitrate 2Mbps
- Baselines:
  - RTP/RTCP: Gcc+FIFO, Gcc+CoDel, Gcc+Zhuge (+CoDel)
  - TCP: Copa, Copa+FastAck, ABC, Copa+ZhugeNetwork
- Traces:
  - WiFi: Restaurant (W1)
  - Office (W2)
  - Cellular: Indoor Mixed 4G/5G (C1)
  - City 4G (C2)
  - City 5G (C3)
- Metrics:
  - RTT (tail latency: >200ms)
  - Frame delay (delayed: >400ms)
  - Frame rate (low: <10fps)

Comparison with RTP

- reduce the ratio of long network RTT by 45% to 75% compared with the best baseline
- delayed frame ratio is reduced by 38% to 92% in different traces
- the P99 tail latency is reduced from 400ms to 170ms
- 400ms delayed frame ratio is reduced from 1% to 0.55% based on trace W1.
- Zhuge could also reduce the ratio of low frame rate by at least 50% in two traces



Comparison with TCP

- tail latency: Copa+Zhuge comprehensively outperforms Copa and Copa+FastAck
- For frame delay, Copa+Zhuge achieves the best performance over competitors where Copa+FastAck is slightly better.

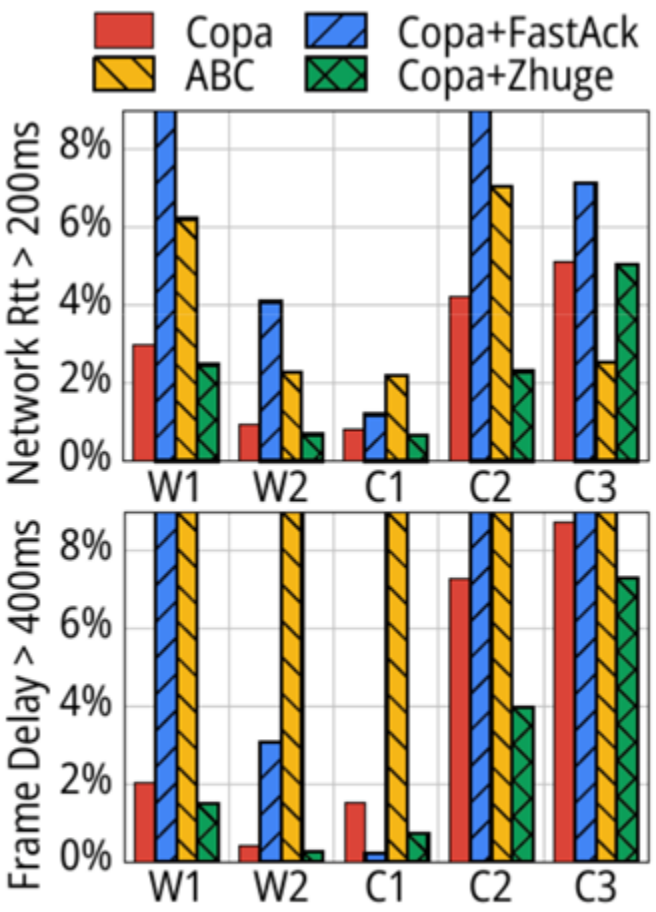


Figure 12: Results of trace-driven simulations over TCP.

Quickly adapt ABW drop

- RTP/RCTP: reduces the duration of network degradations and application performance by at least 50% in a wide range of settings
- TCP: reduce the duration of high network RTT by 14% to 64.3% when k <30. For

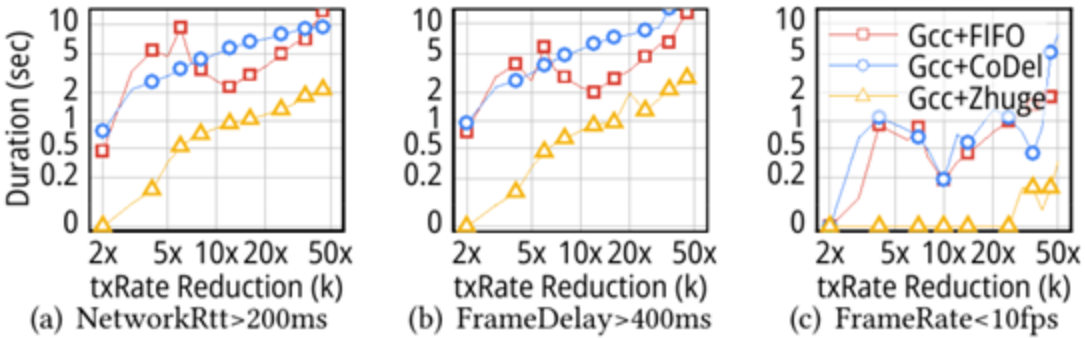


Figure 14: Performance comparison over RTP under ABW drop.

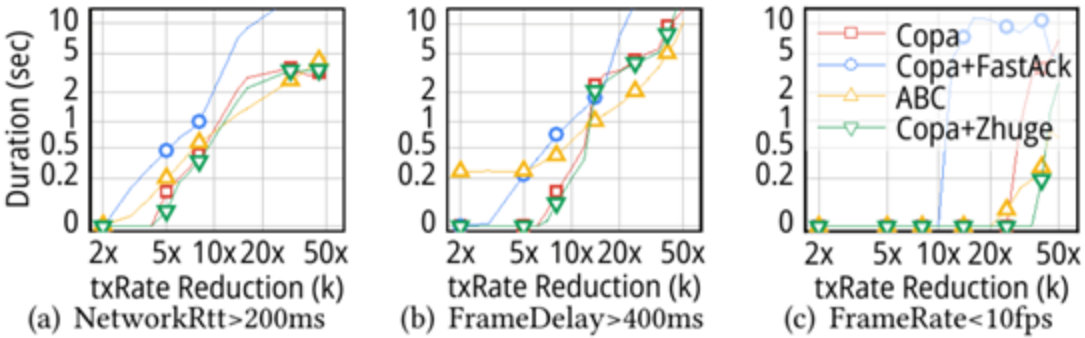


Figure 15: Performance comparison over TCP under ABW drop.

Flow competition & wireless interference

- Competition: reduce the duration of performance degradation by up to 40% in all cases
- Interference: reduce the frequency of degradation of both network condition and application performance by at least 50%

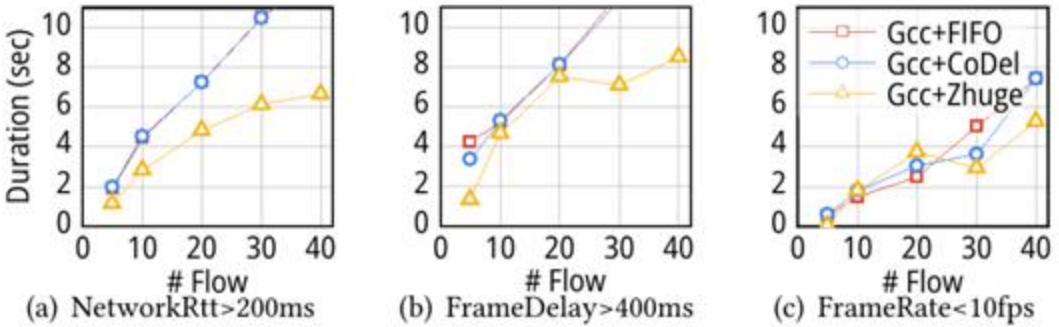


Figure 16: Performance comparison over RTP under competition.

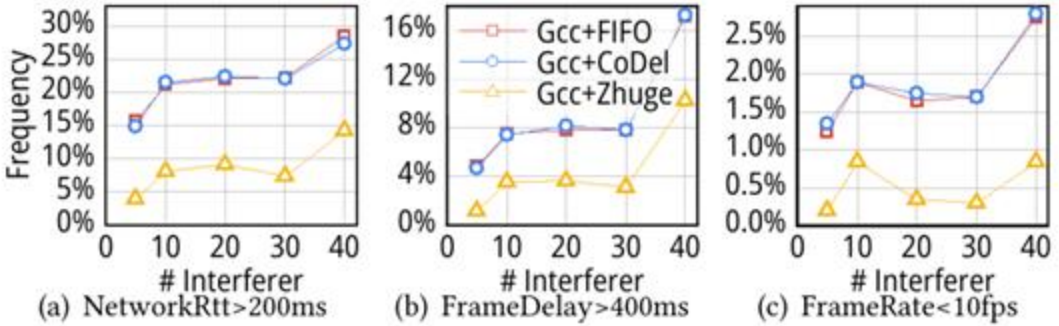


Figure 17: Performance comparison over RTP under interference.

Real-World Experiments

- both network RTT and frame delay of the RTC flow with Zhuge has been improved against baselines by 17% to 95% (network RTT) and 9% to 67% (frame delay) in all scenarios.

- can maintain similar average bitrate

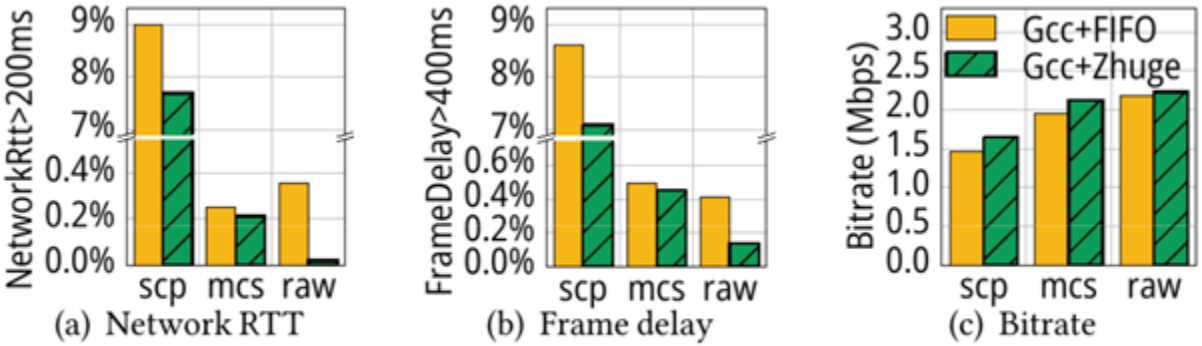


Figure 18: Testbed experiments of Zhuge with an RTC flow.

Estimation Accuracy

- In most cases, the prediction error is much less than the RTT in our experiment (50ms).
- when the estimated delay is low (1-64ms), the estimation is usually accurate. When the estimated delay is high (>64ms), the estimation could be inaccurate, but the real delays are still high enough (more than one RTT)

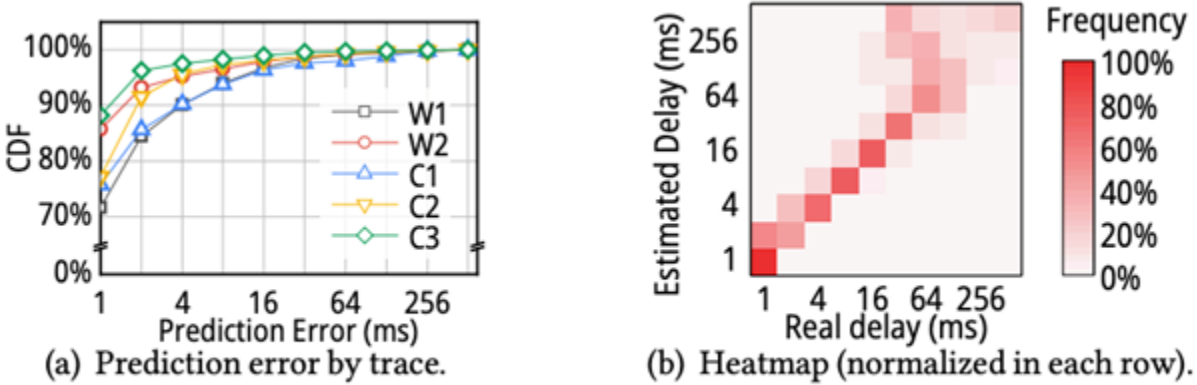


Figure 19: Prediction accuracy of Zhuge Fortune Teller.

Fairness and CPU utilization

- the bitrate fairness in the steady state is not affected by Zhuge with GCC over or Copa.
- For both GCC and Copa, the bitrate difference of the two flows are < 3%
- 10-year-ago APs could still support Zhuge to process 5 concurrent RTC flows

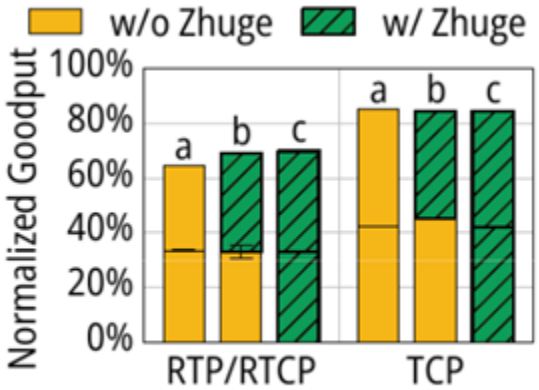


Figure 20: Fairness of Zhuge.

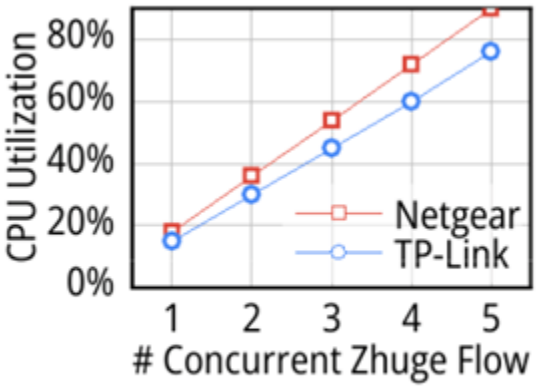
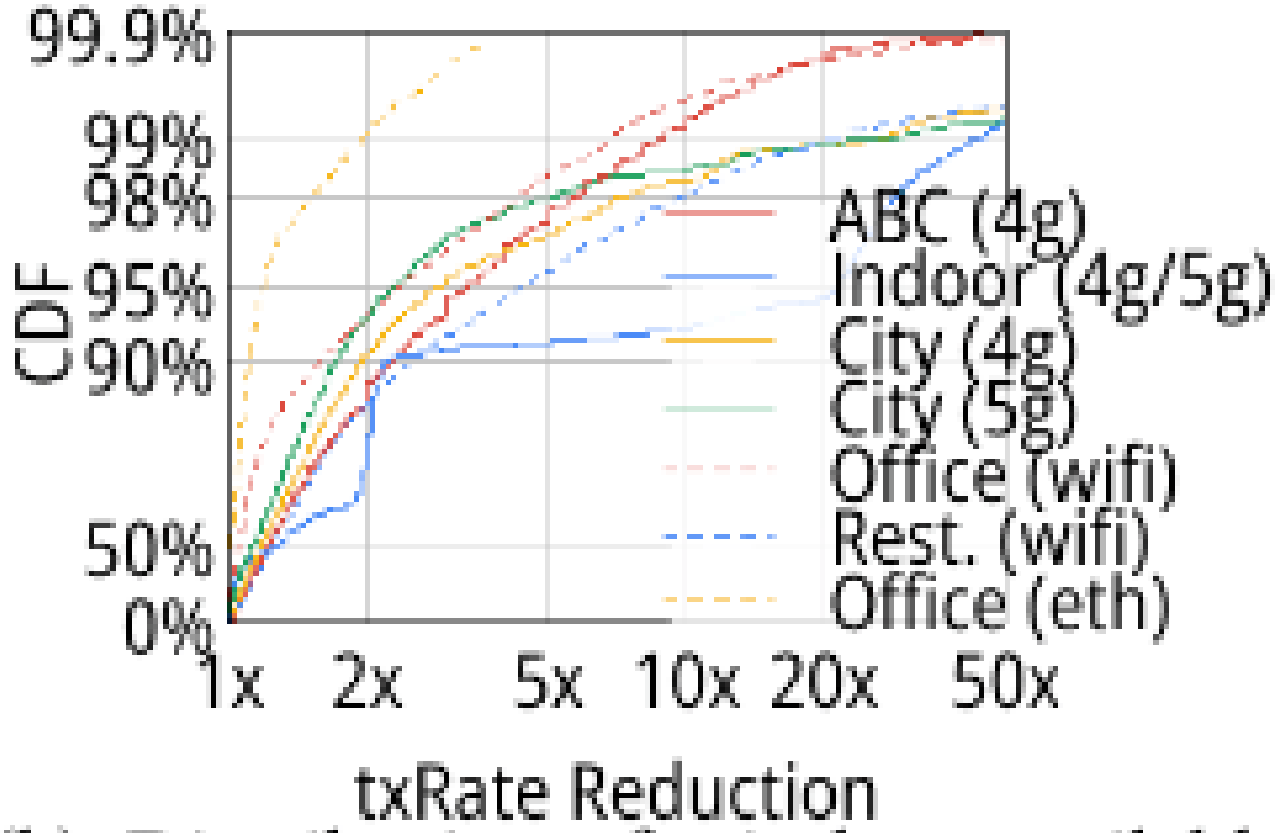


Figure 21: CPU Overhead.

Discussion



(b) Distribution of wireless available bandwidth reduction ratio.