Augmenting L4S with predictive out-of-band signaling.

Arghya Bhattacharya
Mentors: Edward Grinshpun and Chuck Payette
Department Head: TV Lakshman
Network Systems and Security Research Lab
Bell Labs Core Research

My Introduction

Low-latency, high-volume, variable-bitrate applications are coming into industry focus

- Examples:
  - AR/VR applications.
  - Autonomous-guided vehicles and drones.
  - Online multiplayer mixed-reality gaming and cloud gaming.
  - Remote surgery.
- Requirements:
  - Minimal buffering.
  - High-resolution adaptive-bitrate media.
  - Desired low jitter.
  - Minimal number of lost packets.

Mission Statement

- **L**ow queuing **L**atency, **L**ow **L**oss, and **S**calable throughput (L4S) is emerging to achieve low latency and low loss for high-volume variable-rate applications.
- Advantages:
  - ✔ Frequent control signals from the network without compromising network utilization.
  - ✔ Fast reaction time.
  - ✔ Very low overhead (2 ECN bits).
- Shortcomings of L4S:
  - ✕ Information that L4S provides is limited, only frequency and percentage of marking
  - ✕ Reactive nature of L4S congestion control that relies on marking feedback.
  - ✕ Suboptimal behavior (TBA)
    - ✕ Security considerations: service providers will have to trust the end devices to mark, do scheduling based on that, it can affect all traffic in network.
- Where do we come in?
  - Out-of-band predictive advice from the network may help!

Agenda

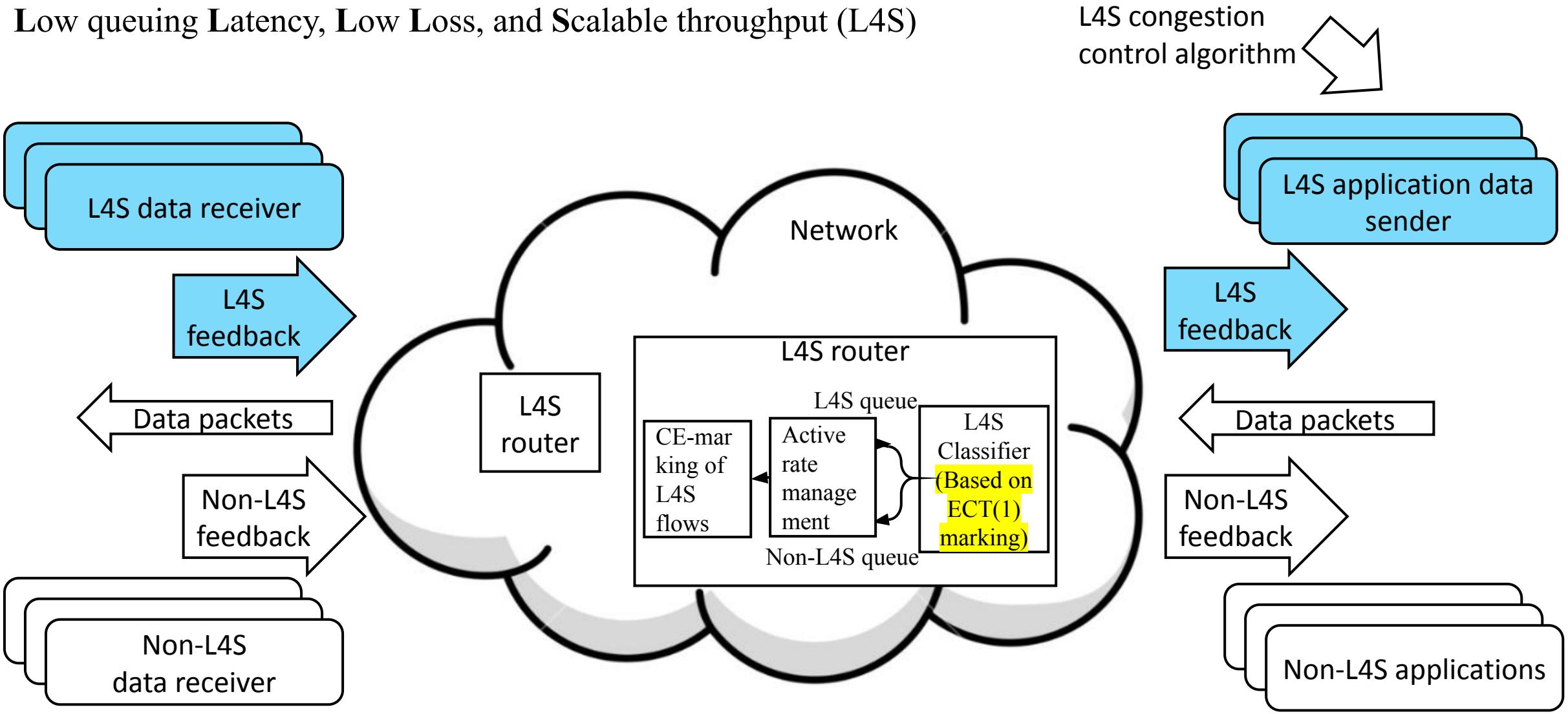
- Architecture of L4S.
- Brief overview of the congestion control algorithms.
- Experimental setup.
- Where does SCReAM need improvement?
- Future directions.

Agenda

- **Architecture of L4S.**
- Brief overview of the congestion control algorithms.
- Experimental setup.
- Where does SCReAM need improvement?
- Future directions.

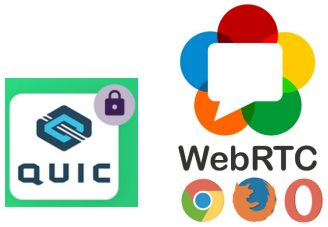
Emerging industry solution for low-latency, high-volume applications: L4S

- **L**ow queuing **L**atency, **L**ow **L**oss, and **S**calable throughput (L4S)



Transport protocol for low-latency, high-volume applications?

- Over UDP



- Peer-to-peer live media streaming
  - Real-time encoding in media server must be considered for L4S feedback.
- Example: cloud gaming, where there is a gaming server and several clients/users/players who are playing the game.
  - The server renders customized video for each player. The output is also a function of the inputs (e.g., keyboard, voice command) of the player.

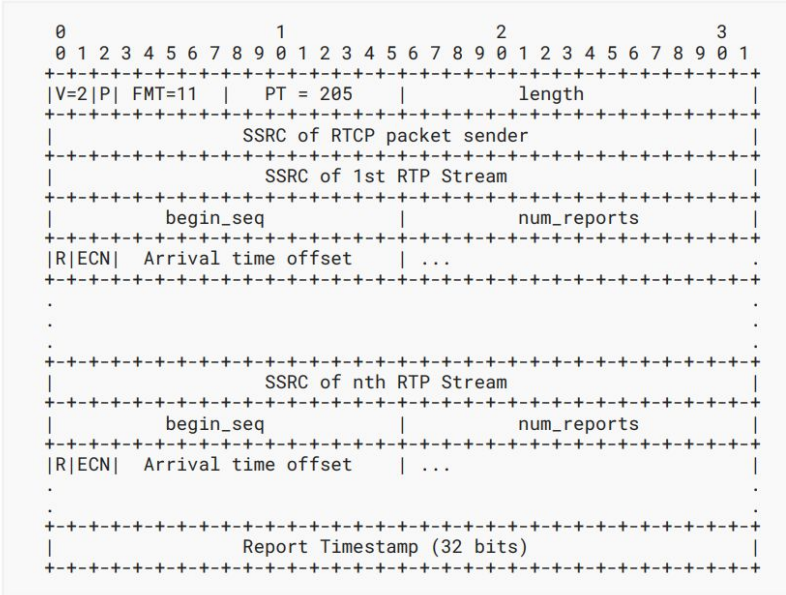
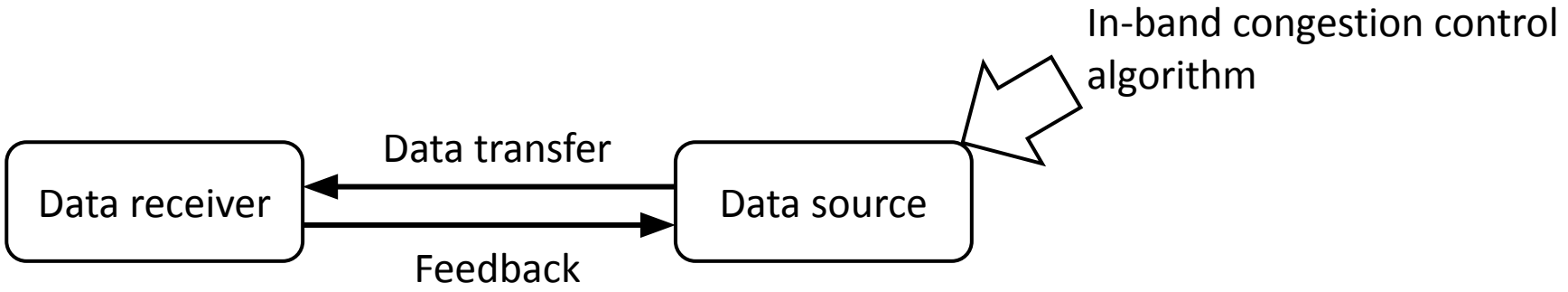


Figure 1: RTCP Congestion Control Feedback Packet Format



There is an intricate relationship between the feedback and the congestion control algorithm.

Agenda

- Architecture of L4S.
- **Brief overview of the congestion control algorithms.**
- Experimental setup.
- Where does SCReAM need improvement?
- Future directions.

Brief overview of CC algorithms

- **BBR**
- GCC
- BBRv2
- NADA
- Prague

- Bottleneck Bandwidth and Round-trip propagation time
- Actively estimates bottleneck bandwidth and RTT.
- Loss agnostic.
- Poor co-existence with the other algorithms.
- More BW allotted to flows with high RTTs leading to unfairness.

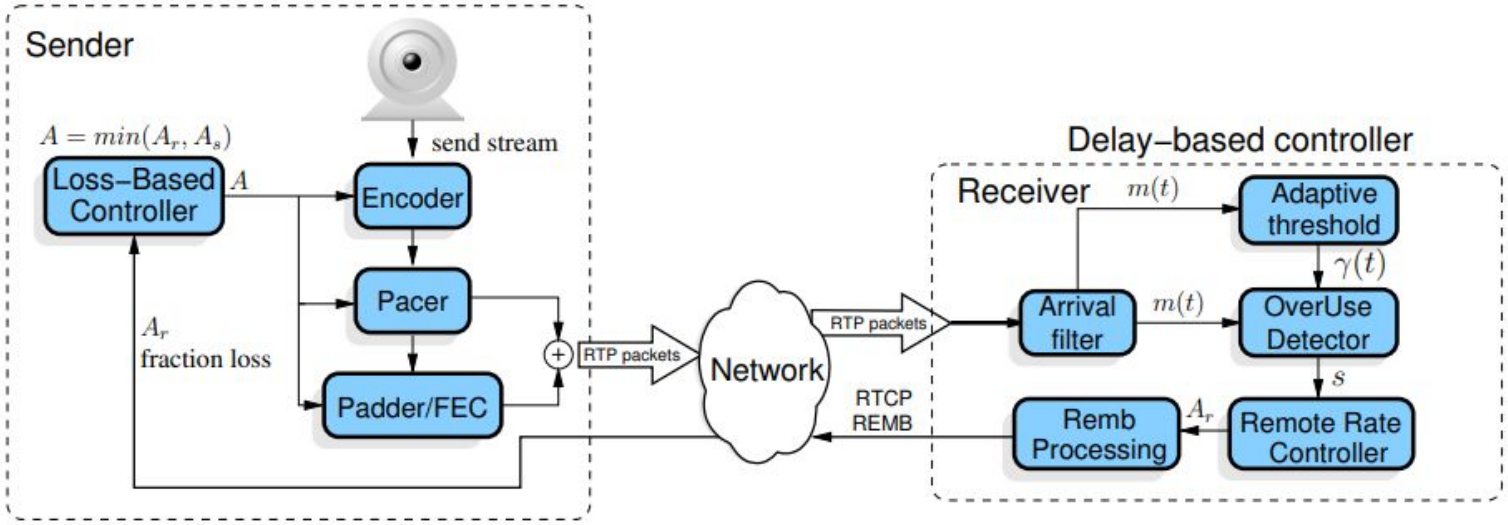
Brief overview of CC algorithms

- **GCC**
- BBRv2
- NADA
- Prague

```
6.4.2 RR: Receiver Report RTCP Packet

        0                   1                   2                   3
        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
header |V=2|P|    RC   |   PT=RR=201   |             length            |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                     SSRC of packet sender                     |
       +=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
report |                 SSRC_1 (SSRC of first source)                 |
block  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  1    | fraction lost |       cumulative number of packets lost       |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |           extended highest sequence number received           |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                      interarrival jitter                      |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                         last SR (LSR)                         |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                   delay since last SR (DLSR)                  |
       +=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
report |                 SSRC_2 (SSRC of second source)                |
block  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  2    :                               ...                             :
       +=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
       |                profile-specific extensions                    |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Feedback packets follow RFC3550 format.
- Recommendation is one or two feedbacks per frame (~33ms for 60fps media).
- Google's implementation uses almost one report a second. There is a limit of how much percentage of the overall throughout is used by the feedback channel.
- REMB (Receiver Estimated Maximum Bitrate) feedbacks sent by the receiver as proposed in draft-alvestrand-rmcat-remb-03. It imbibes the idea of a smart client, and relatively dumb server.

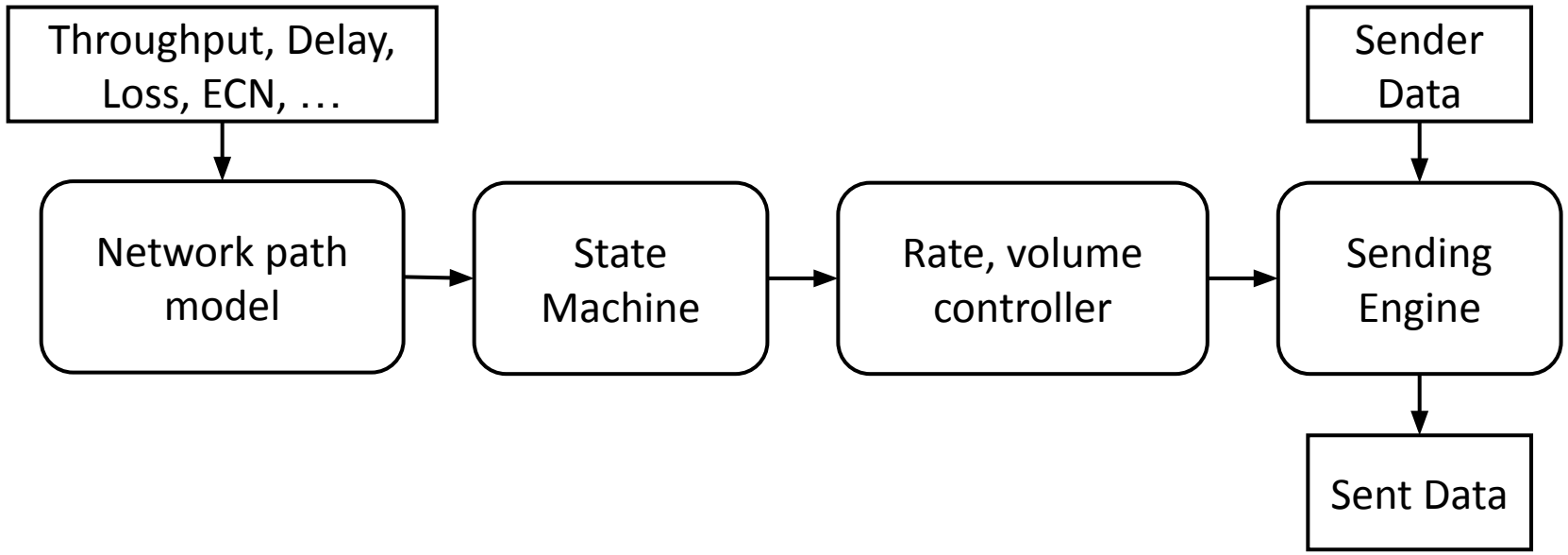


Brief overview of CC algorithms

- GCC
- **BBRv2**
- NADA
- Prague

Improved performance than BBR, GCC, DCTCP.

- Reduced queueing delay: RTTs lower than BBR.
- Reduced packet loss and explicit loss rate target.
- DCTCP-inspired ECN.
- Tested on YouTube, used for QUIC.



Brief overview of CC algorithms

- GCC
- BBRv2
- **NADA**
- Prague

- Network-assisted Dynamic Adaptation [RFC8698] imbibes the idea of a dumb client and smart server.
- Queueing delay, packet losses, and ECN marking ratios are forwarded by the data receiver in Receiver Reports (RR) [RFC6679].
- Implicit congestion signals and aggregated report on ECN-based congestion control.

Brief overview of CC algorithms

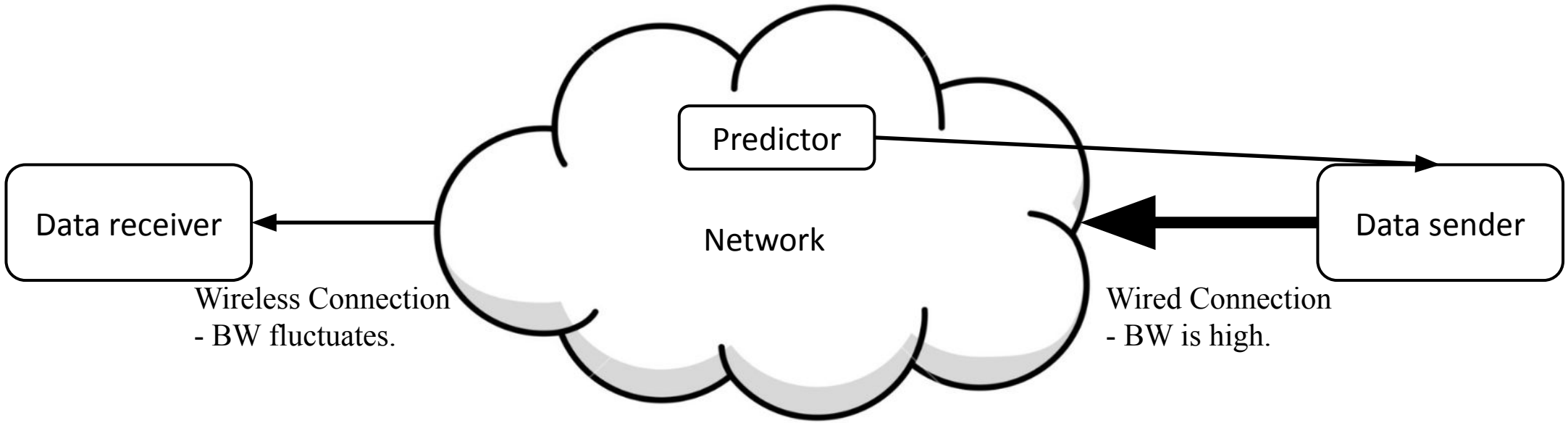
- GCC
- BBRv2
- NADA
- **Prague**

- SCReAM implements L4S Prague requirements in the feedback, RFC8888 format.
- Prague CC uses additive increase and multiplicative decrease (AIMD) strategy for congestion window.
  - A tiny continual sawtooth pattern of the order of a round-trip.
  - Tight control, and very low queueing delay and queue variation.
- This moves the industry towards smart server, dumb clien
- Implementation is work-in-progress to apply all the IETF and RFC standards.

Agenda

- Architecture of L4S.
- Brief overview of the congestion control algorithms.
- **Experimental setup.**
- Where does SCReAM need improvement?
- Future directions.

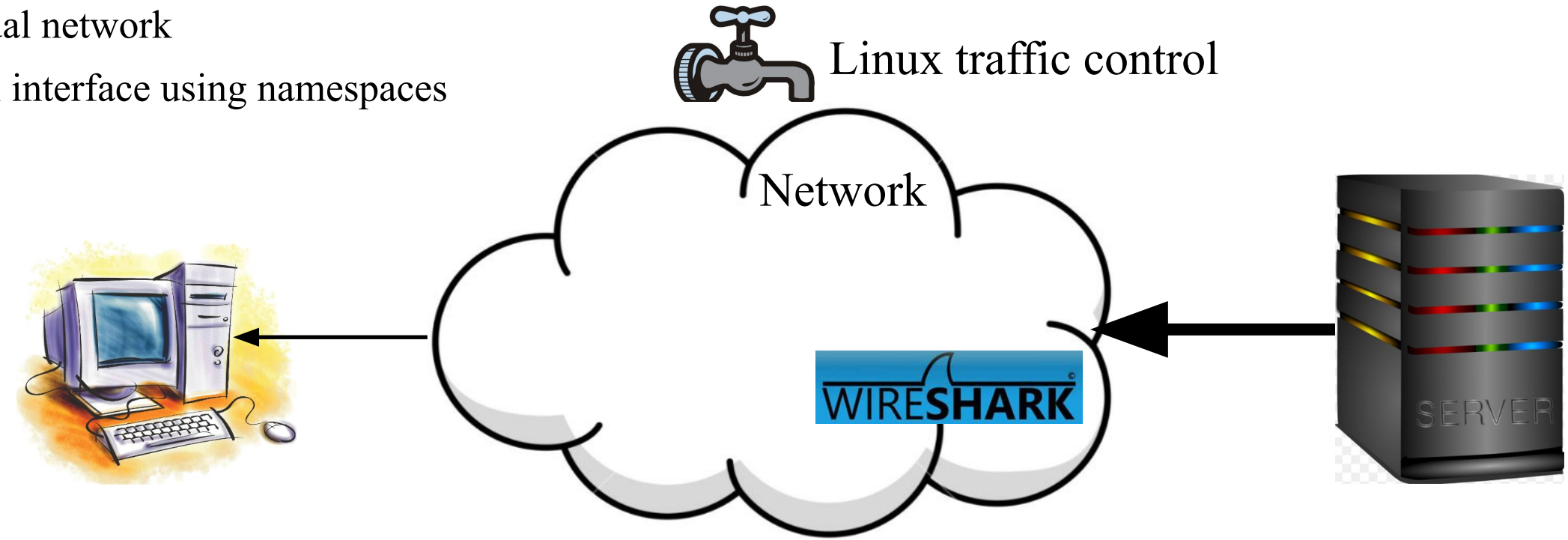
Goal of evaluation



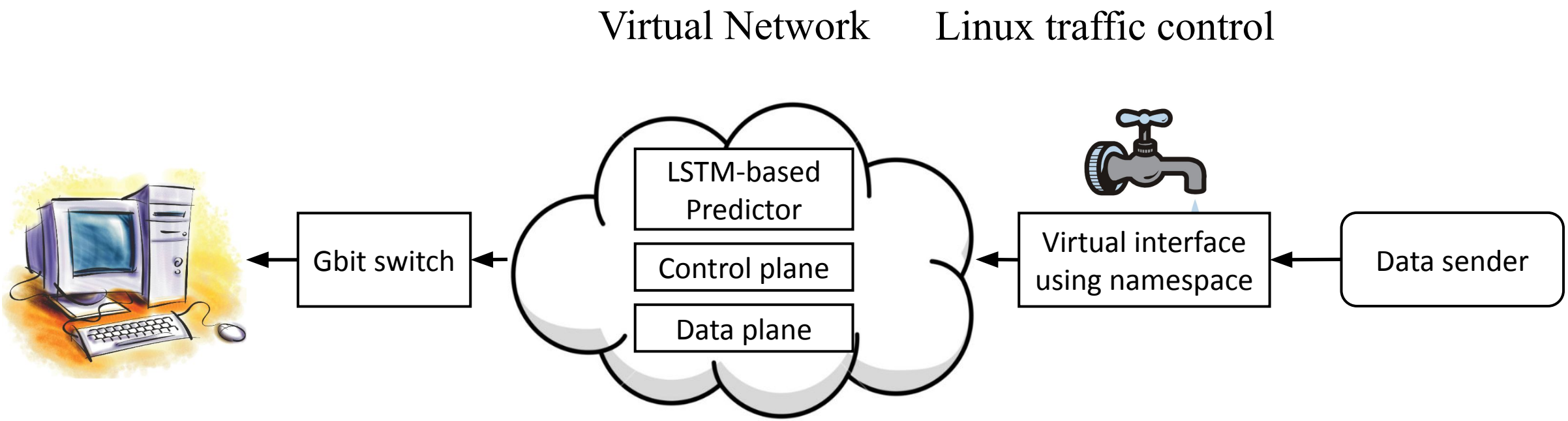
- Identify some of L4S shortcomings
  - Using SCReAM as testbed [https://github.com/EricssonResearch/scream]
- Can predictive out-of-band network feedback improve situations where L4S comes short?

Experimental setup

- Server machine and client machine [Two boxes connected via Gbit switch]
- Control plane and data plane network. [Exact network diagram]
- A virtual network
- Virtual interface using namespaces



Experimental setup



Can predictive out-of-band network feedback improve situations where L4S comes short?

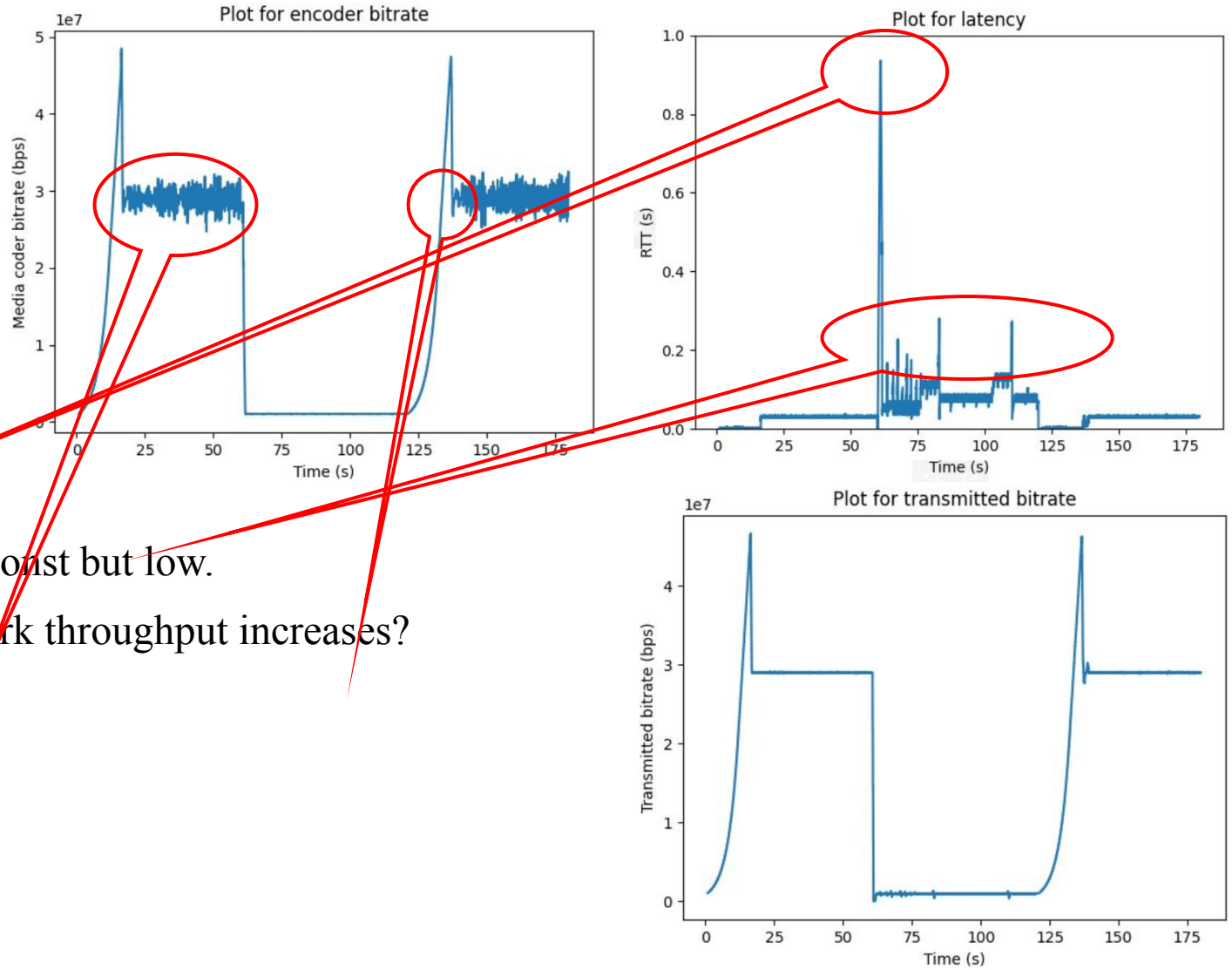
Agenda

- Architecture of L4S.
- Brief overview of the congestion control algorithms.
- Experimental setup.
- **Where does SCReAM need improvement?**
- Future directions.

Where does SCReAM lack?

Experiment:

- Network throughout fluctuates as follows:
  - 0-60s: 30Mbps, 60-120s: 1Mbps, 120-180s: 30Mbps.
- The feedbacks come every ~16-17ms.
- Case 1: Network does not do marking.

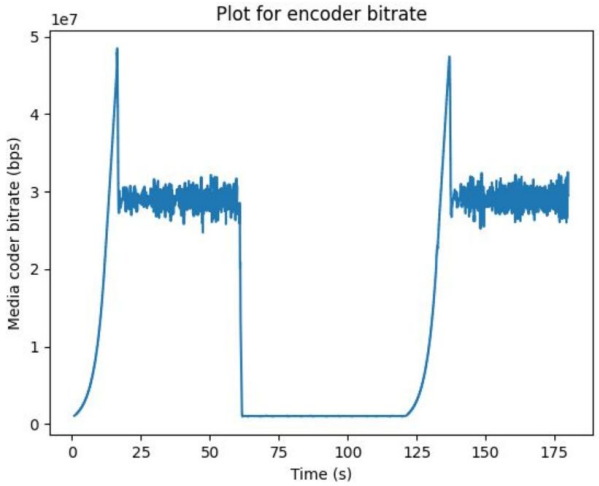


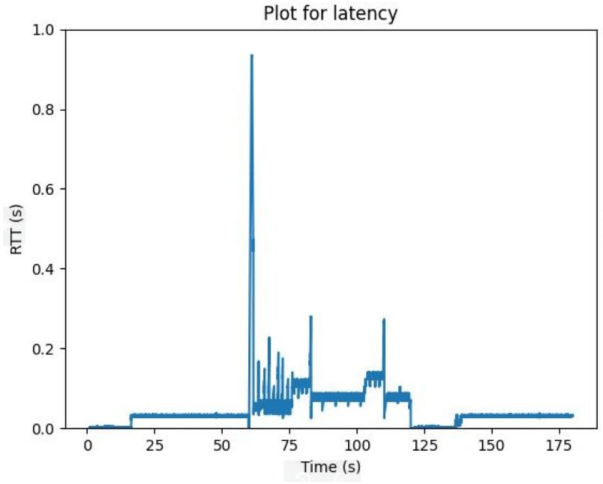
Shortcomings:

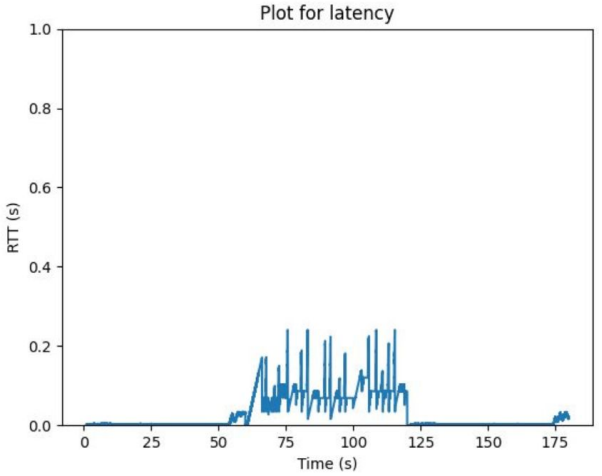
- Latency spike when net throughput drops.
- Latency is ~200ms when network throughput is const but low.
- How long to gain the encoder bitrate when network throughput increases?
- Why unnecessary oscillation beyond 30Mbps?

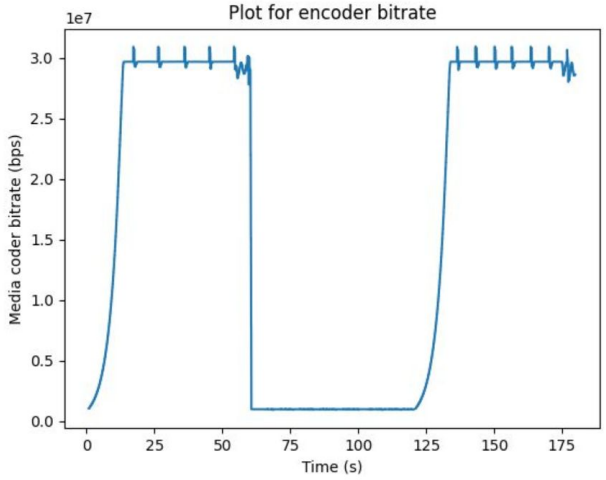
Congestion control using Predictive out-of-band feedback in addition to L4S feedback

- The sender knows the actual network throughput 2s prior from the predictor with 100% accuracy.
- Case 1: The encoder does not probe beyond the predicted throughput.



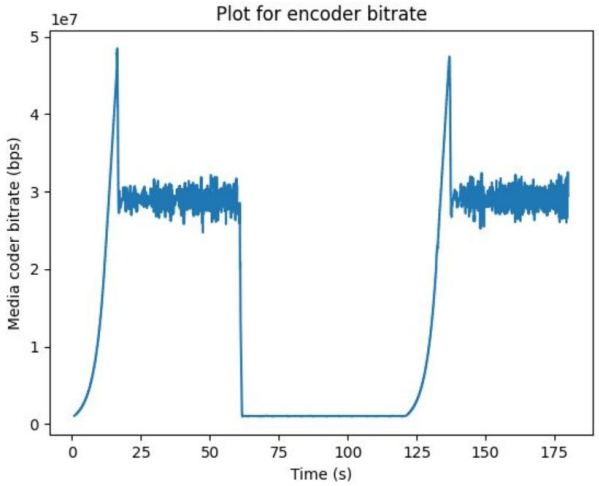


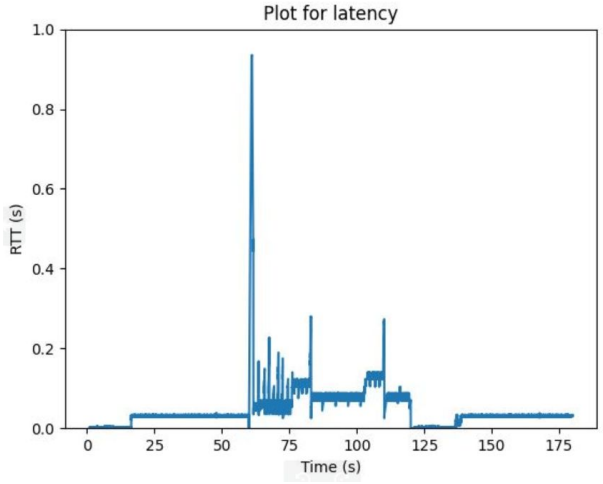


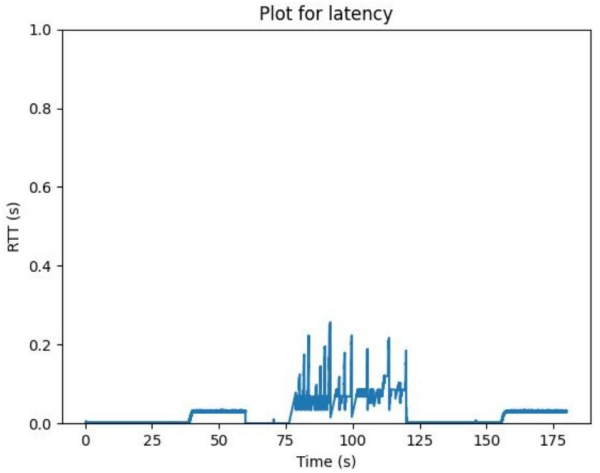


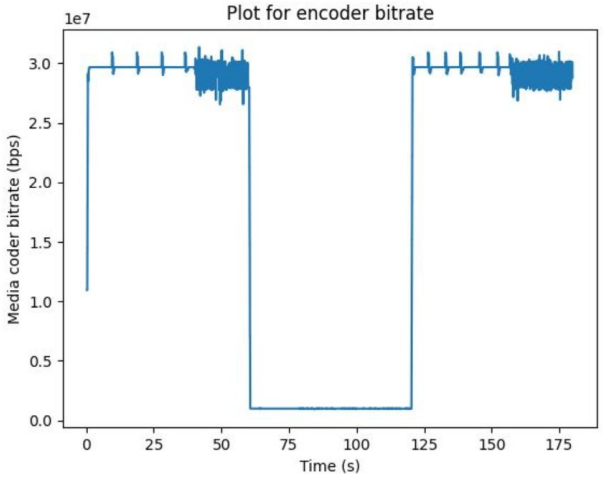
Congestion control using Predictive out-of-band feedback in addition to L4S feedback

- The sender knows the actual network throughput 2s prior from the predictor with 100% accuracy.
- Case 2: The encoder always blindly follows the predicted throughput.









Agenda

- Architecture of L4S.
- Brief overview of the congestion control algorithms.
- Experimental setup.
- Where does SCReAM need improvement?
- **Future directions.**

Future directions

- Similar problem persists in WebRTC applications: the server does not act on both implicit and explicit congestion notifications.
- We need to investigate BBRv2 under our empirical framework.
- It is not the end of the tunnel. We have not considered the security aspect of this algorithm. What happens if the application is greedy and marked unnecessarily to get the best service?

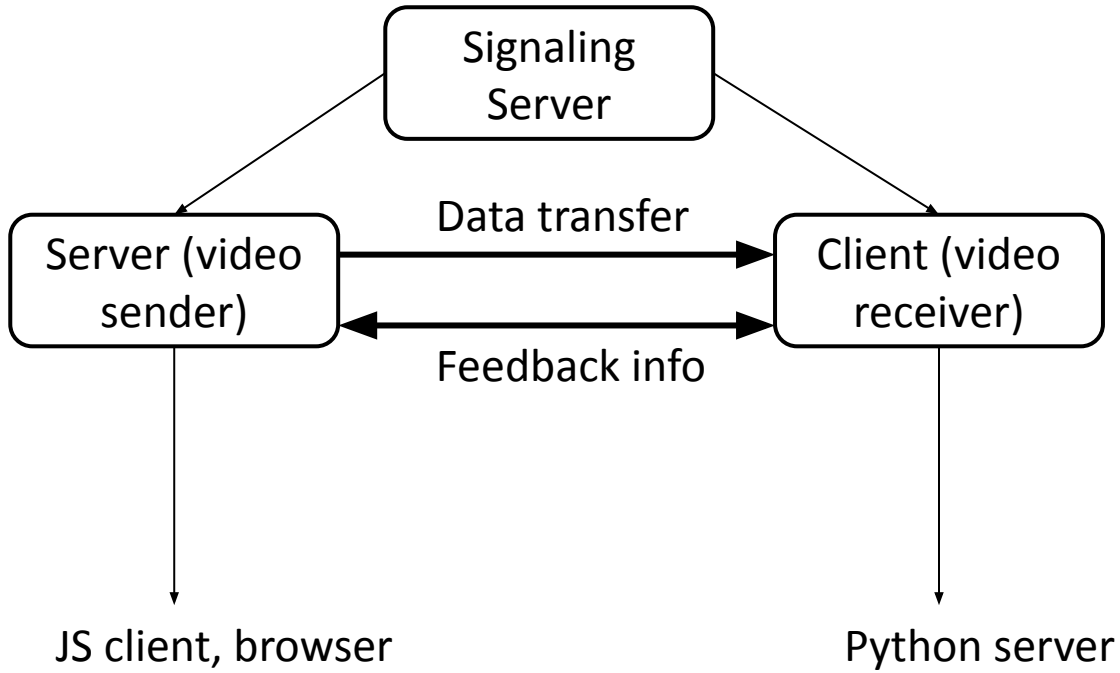
References

- RTP Media Congestion Avoidance Techniques (rmcat) (ietf.org)
- Real-Time Communication in WEB-browsers (rtcweb) (ietf.org)
- WebRTC beginner notes: Browser APIs and Protocols: WebRTC - High Performance Browser Networking (O'Reilly) (hpbn.co)
- L4S: L4S: Ultra-Low Queuing Delay for All | RITE (riteproject.eu)
- Video source: Big Buck Bunny » Download (blender.org)
- Google BBRv2: Evaluating BBRv2 on the Dropbox Edge Network – Dropbox
  - GitHub - google/bbr
- References of WebRTC implementations:
  - GitHub - pion/webrtc: Pure Go implementation of the WebRTC API
  - GitHub - aiortc/aiortc: WebRTC and ORTC implementation for Python using asyncio
  - src - Git at Google (googlesource.com)
  - external/webrtc - Git at Google (googlesource.com)
  - WebRTC samples
    - MediaStreamTrack Content Hints (webrtc.github.io)
    - MediaStreamTrack Content Hints (w3.org)
    - getUserMedia (webrtc.github.io)
- Description of **APIs**:
  - WebRTC 1.0: Real-Time Communication Between Browsers (w3c.github.io)
  - WebRTC API - Web APIs | MDN (mozilla.org)
  - MediaStreamTrack Content Hints (w3.org)
- Google group discussion:
  - discuss-webrtc - Google Groups
  - Ports: WebRTC port requirements? (google.com)
    - WebRTC ports: Understanding IP addresses and port ranges in WebRTC • BlogGeek.me
    - Use specific ports for webRTC - Stack Overflow
- Wireshark: How to Use Wireshark to Capture, Filter and Inspect Packets (howtogeek.com)

Thank you!

GitHub - aiortc/aiortc: WebRTC and ORTC implementation for Python using asyncio

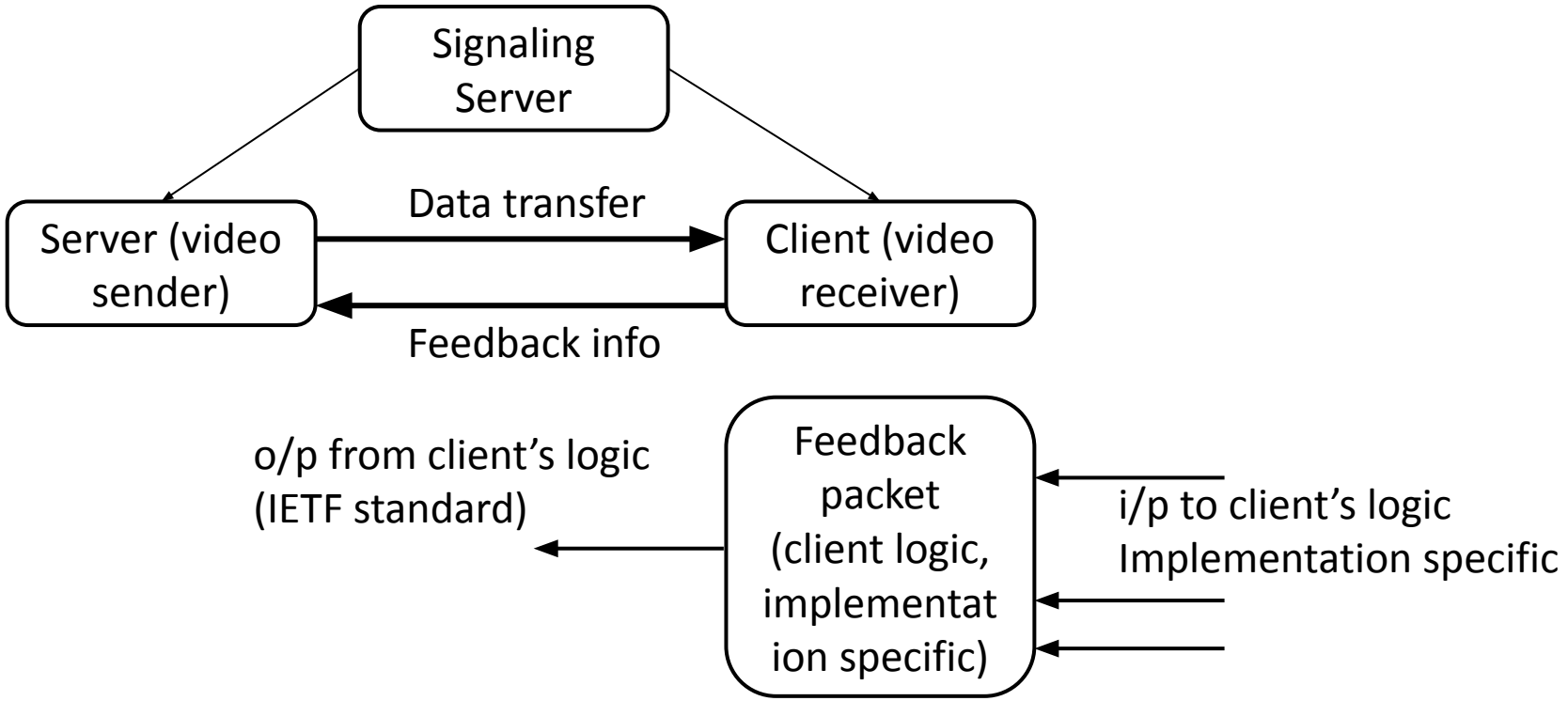
- Python based implementation of WebRTC.
- Video transported from the server to the client: one directional video traffic.
- Signaling server creation.
- Workflow:
  - SDP generation
  - DTLS handshake
  - SRTP keying, encryption and decryption for RTP and RTCP, NACK and PLI for packet loss.
  - SCTP based DataChannels.
- Part of PyPI: aiortc · PyPI
  - More trustworthy
- API Reference — aiortc documentation



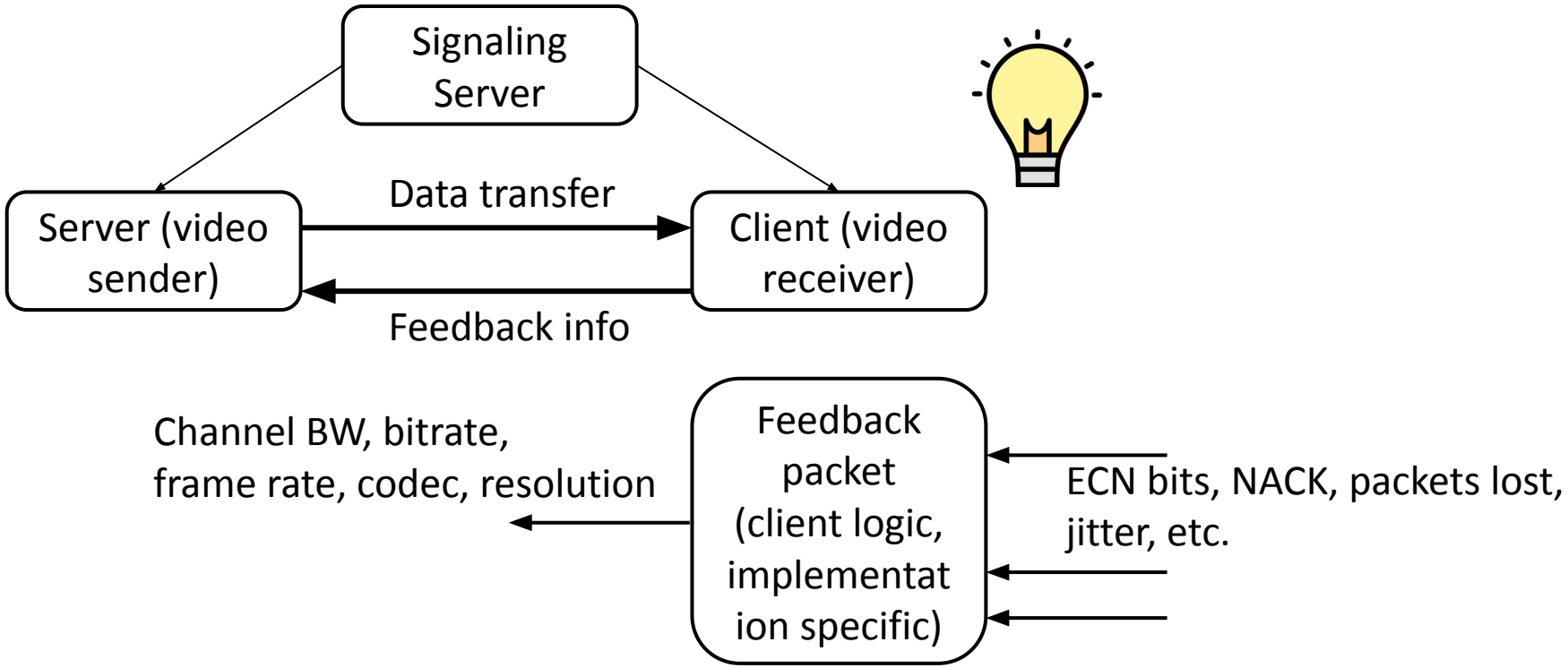
Google’s implementation – WebRTC – C/C++

- external/webrtc - Git at Google (googlesource.com)
  - Based on src - Git at Google (googlesource.com)
  - Basics of WebRTC: Real time communication with WebRTC | Google Codelabs
  - API reference: WebRTC 1.0: Real-Time Communication Between Browsers (w3c.github.io)
  - Experimental RTP header extensions (googlesource.com)
- Demo: GitHub - webrtc/samples: WebRTC Web demos and samples
  - Peerconnection demo: samples/src/content/peerconnection/constraints at gh-pages · webrtc/samples · GitHub
    - WebRTC-Experiment/Pre-recorded-Media-Streaming at master · Google-webRTC/WebRTC-Experiment · GitHub
  - Online demo: Constraints and statistics (webrtc.github.io)
  - GitHub - Google-webRTC/WebRTC-Experiment: WebRTC, WebRTC and WebRTC. Everything here is all about WebRTC!!
    - WebRTC-Experiment/broadcast at master · muaz-khan/WebRTC-Experiment · GitHub

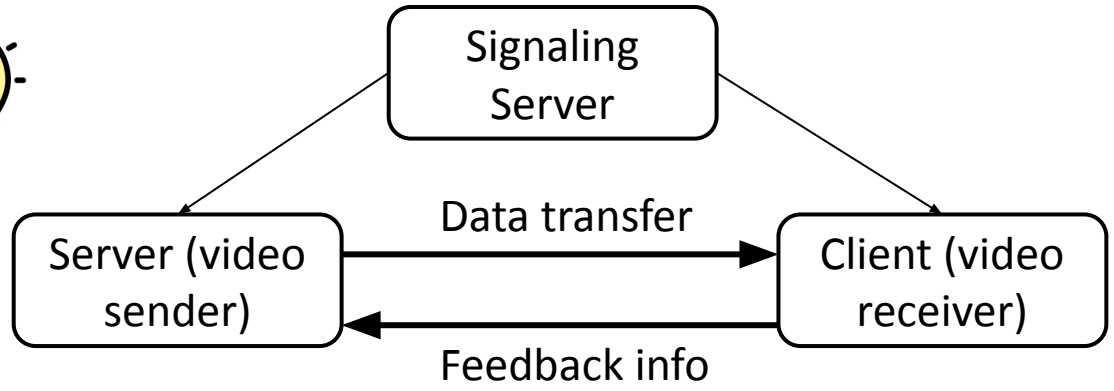
What is the feedback?

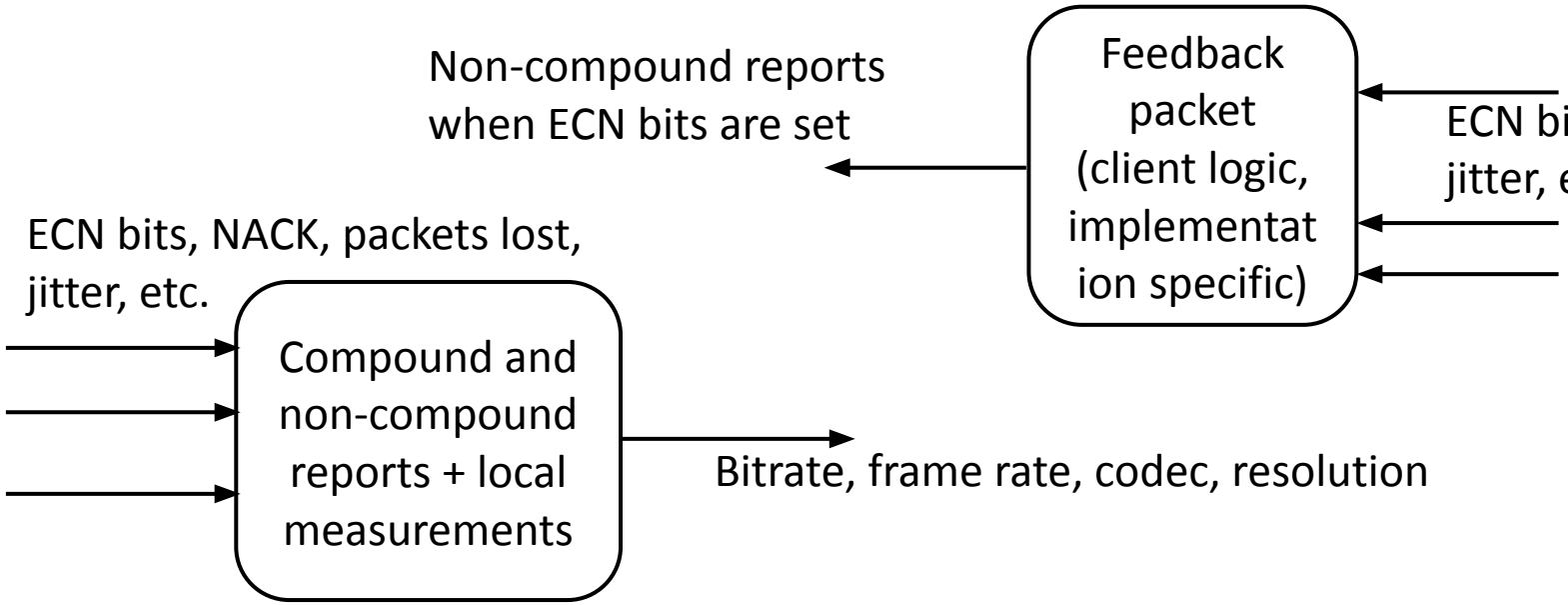


What is the feedback? Smart client



What is the feedback? Smart server, dumb client

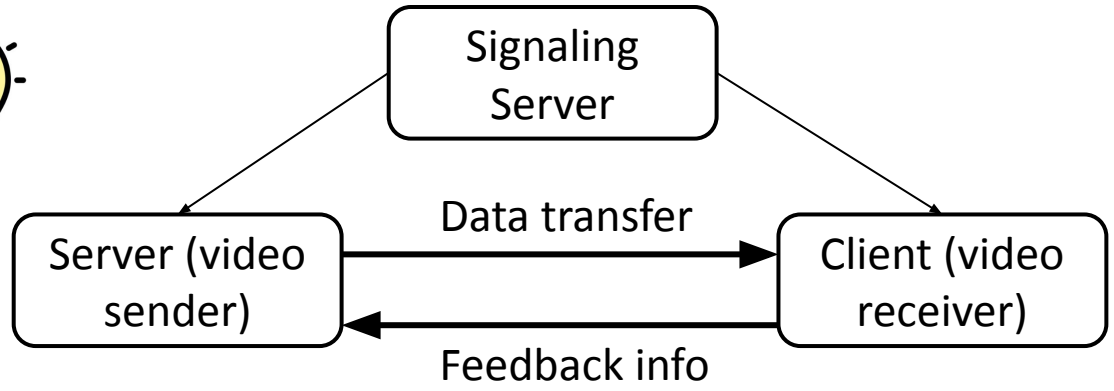




Client side:
1. L4S supported WebRTC
2. API to inspect ECN marks
3. Coroutine to shoot reports

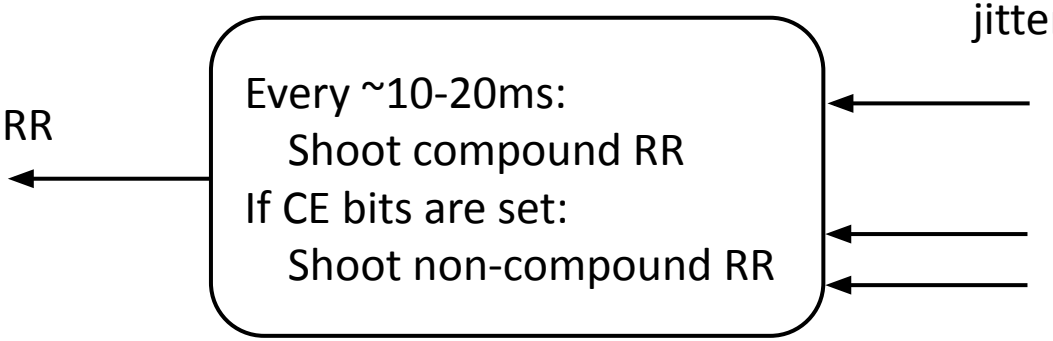
Server side:
1. APIs -- getStat() + ECN marks
2. Logic to determine the bitrate/frame rate/frame size
3. Dynamic transcoding

What is the feedback? Smart server, dumb client

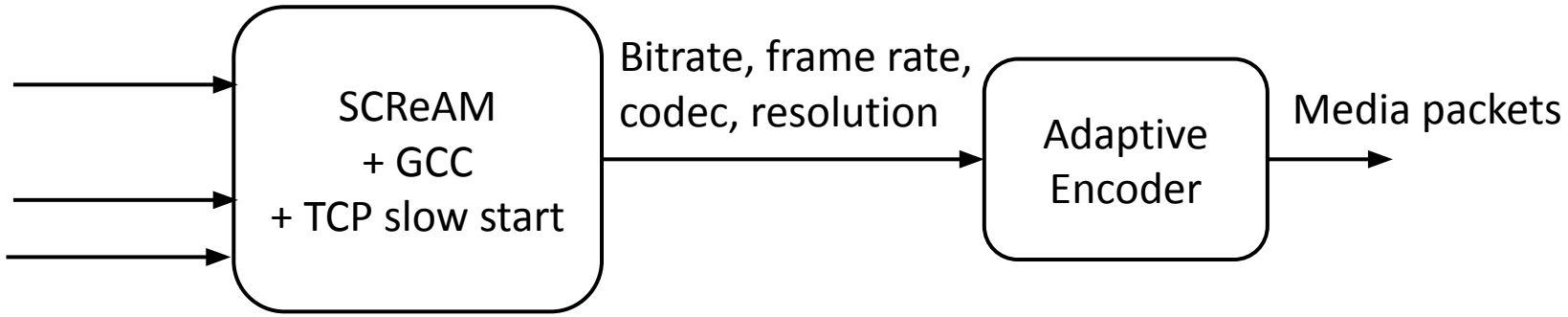


Client side:
1. L4S supported WebRTC
2. API to inspect ECN marks
3. Coroutine to shoot reports

ECN bits, NACK, packets lost, jitter, RTT, etc.



Compound and non-compound RR
+ local measurements



Server side:
1. APIs -- getStats() + ECN marks
2. Logic to determine the bitrate/frame rate/frame size
3. Dynamic transcoding