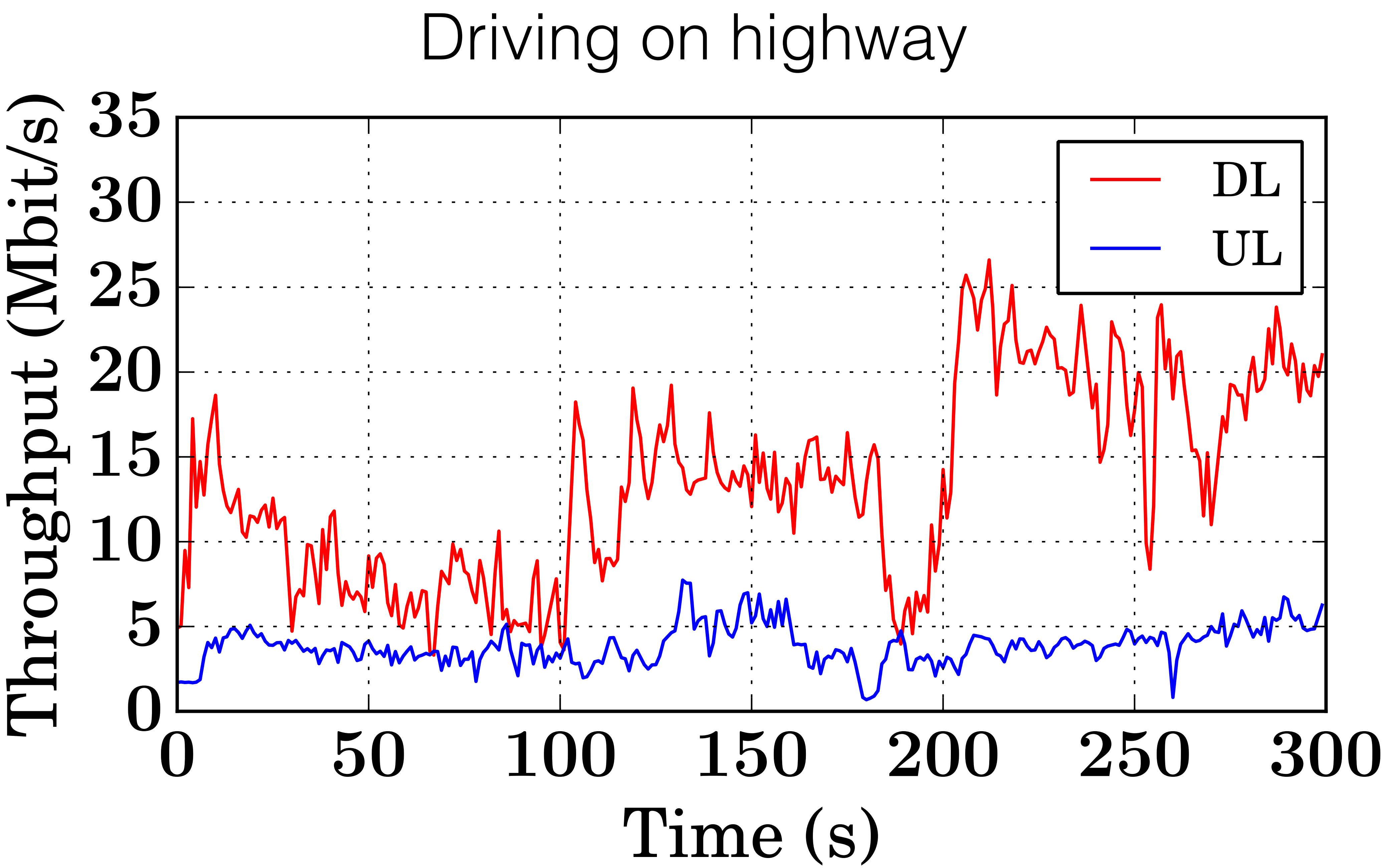
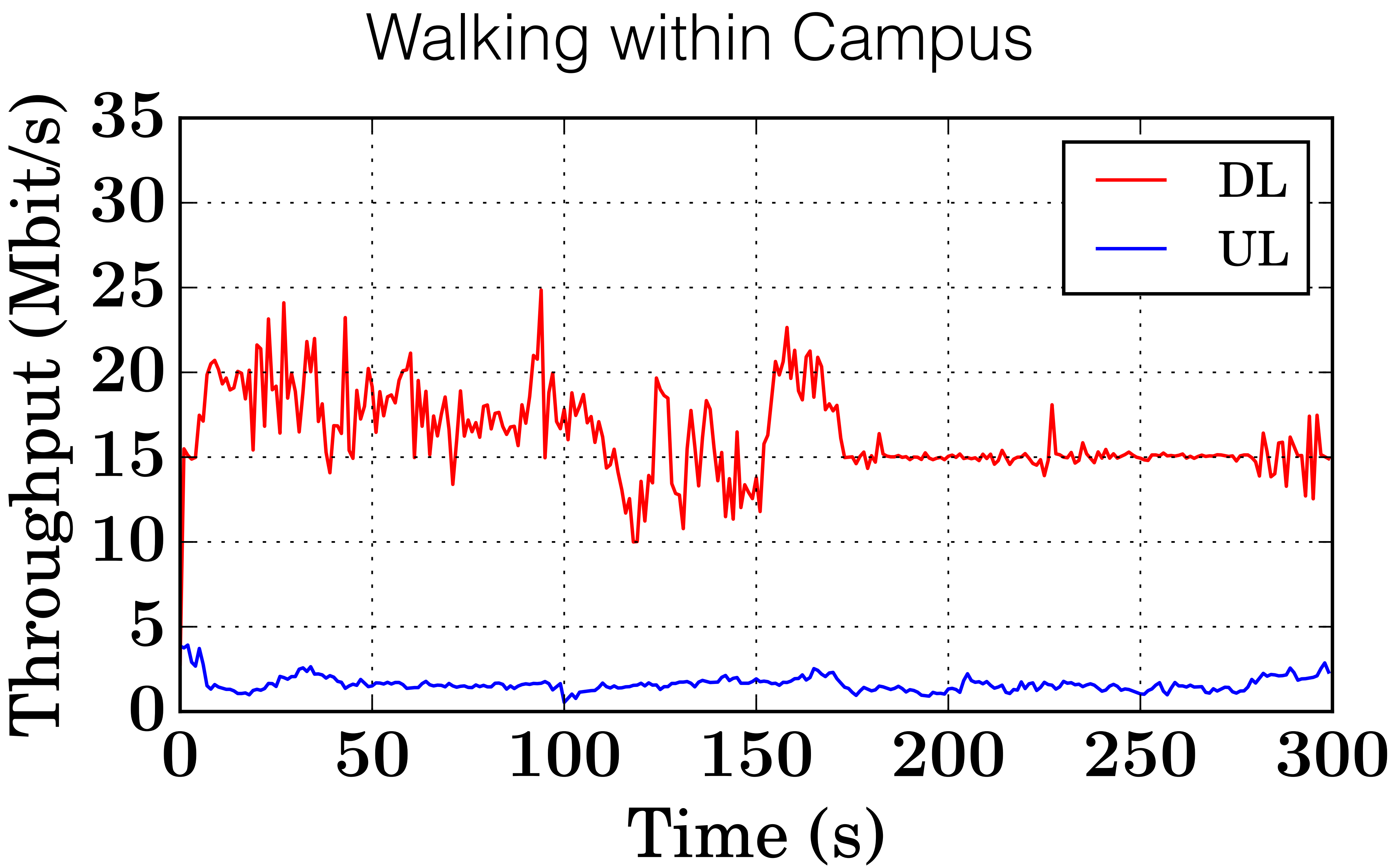
Adaptive Congestion Control for Unpredictable Cellular Networks

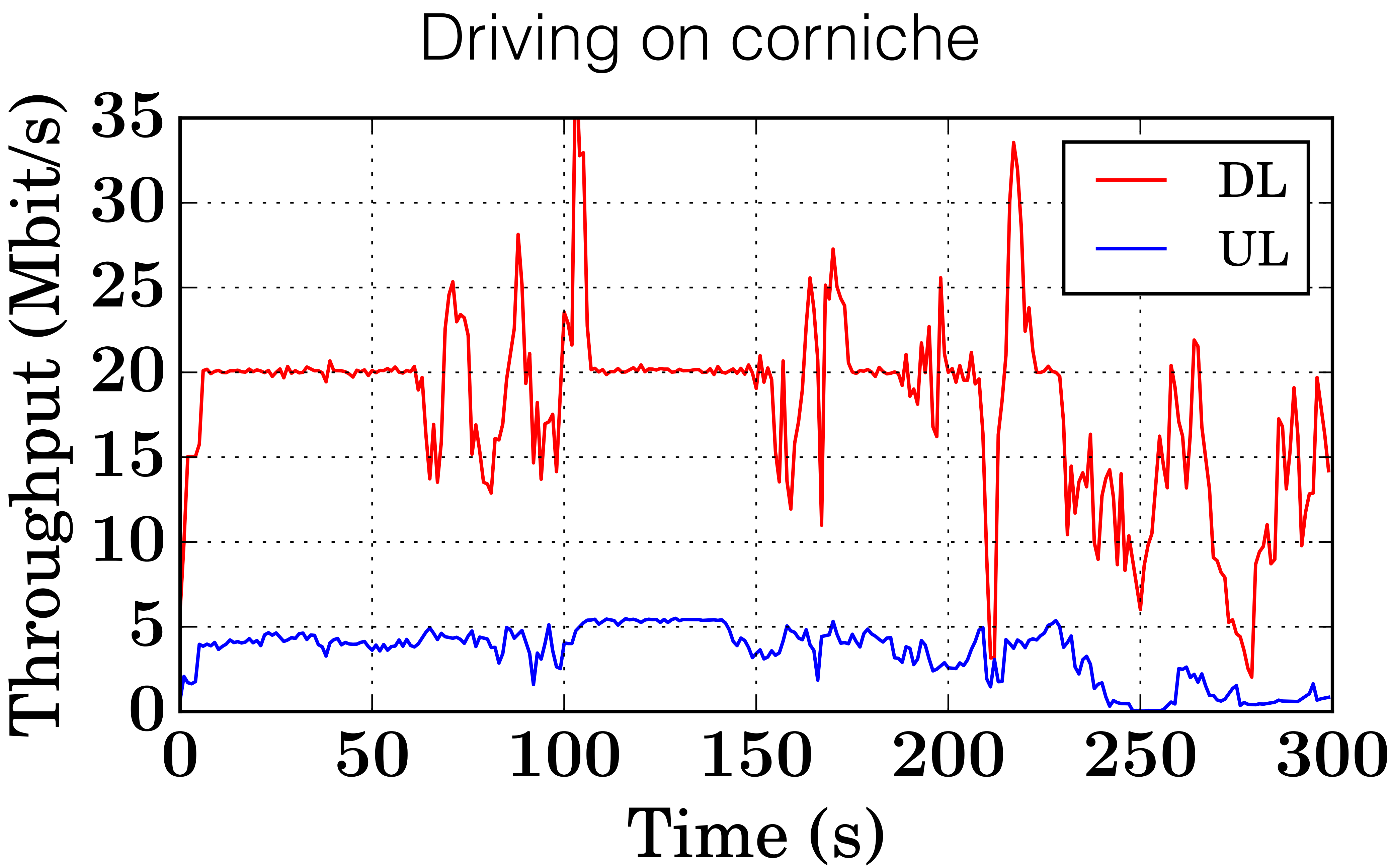
Highly variable cellular channel



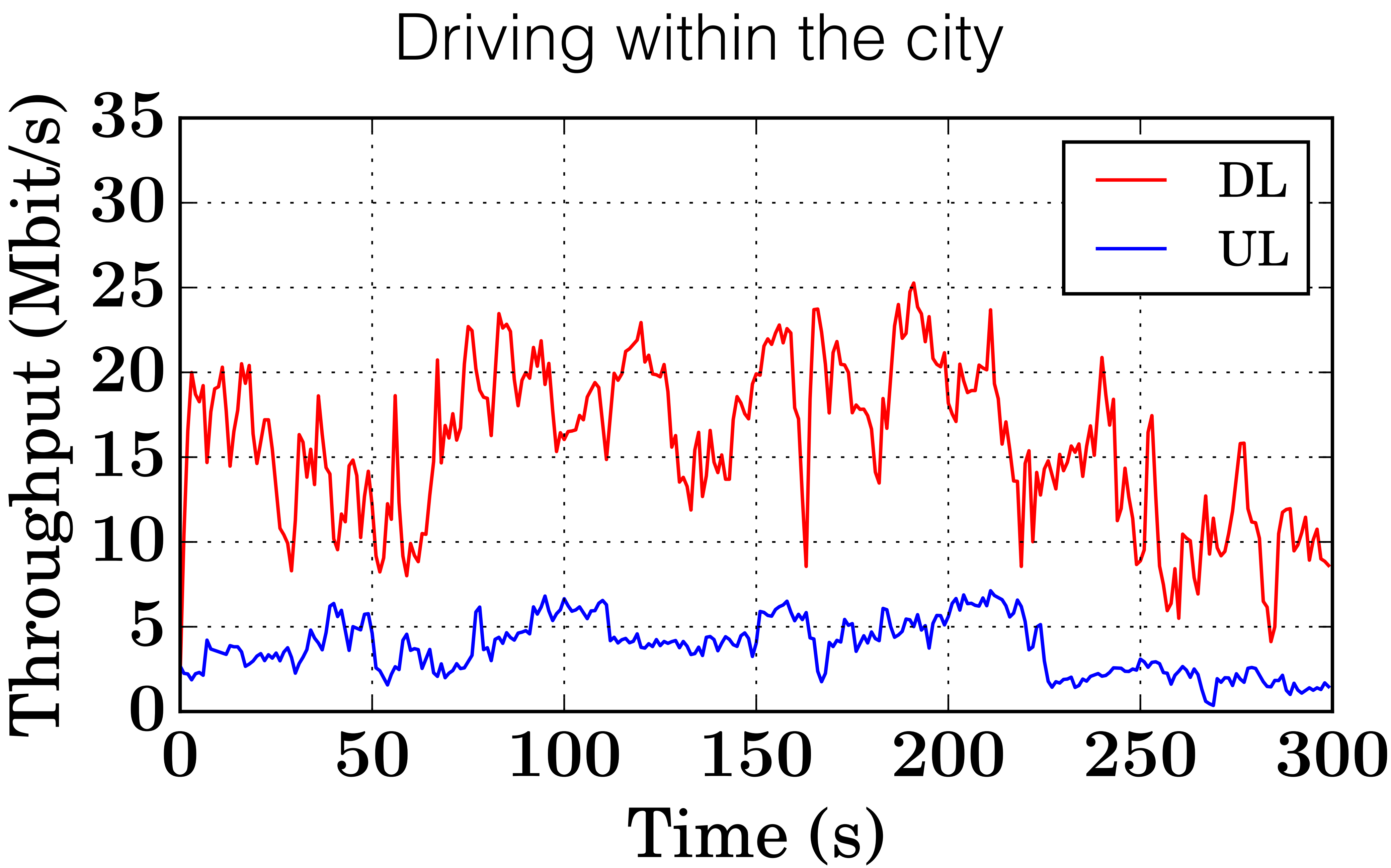
Highly variable cellular channel



Highly variable cellular channel

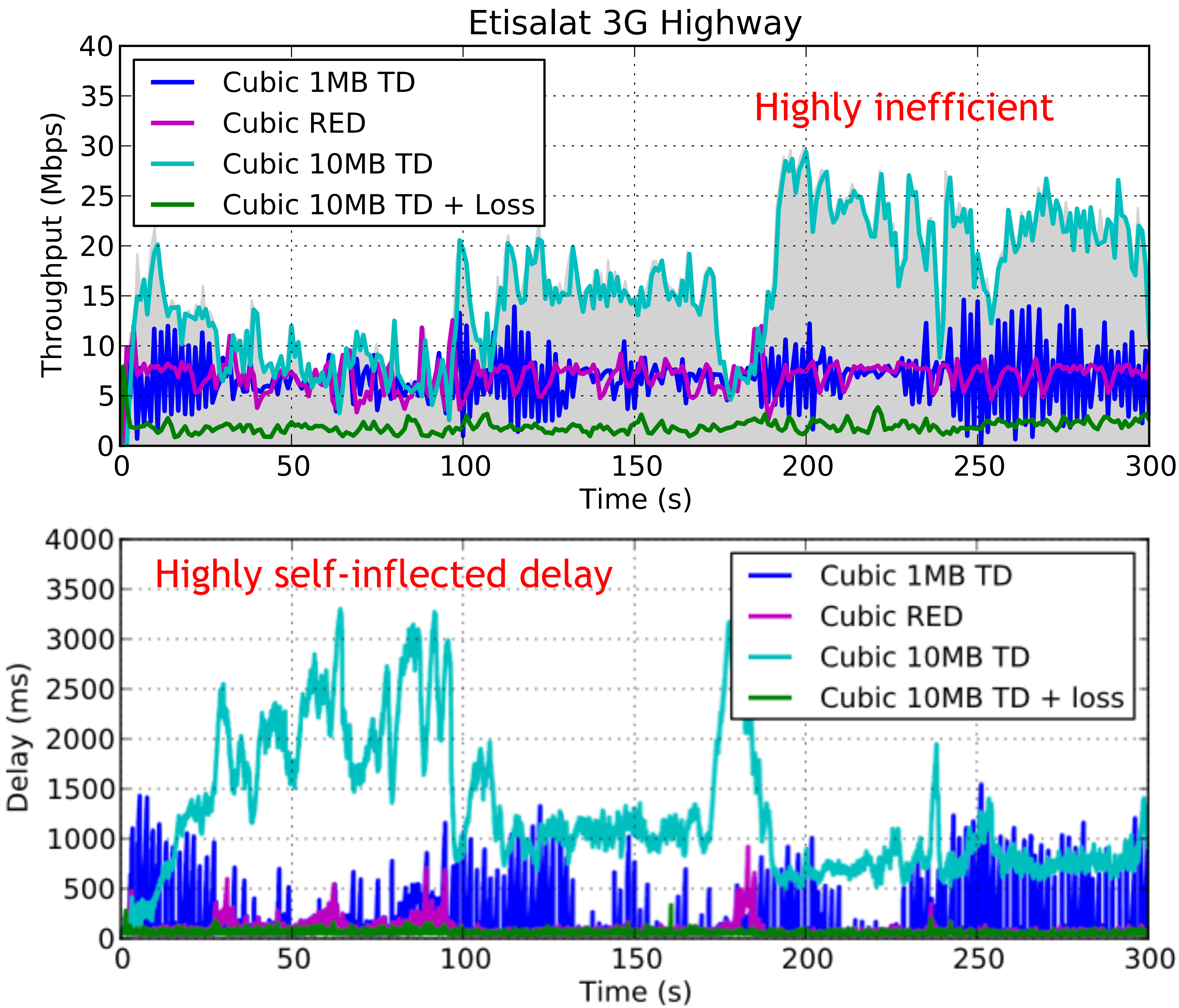


Highly variable cellular channel



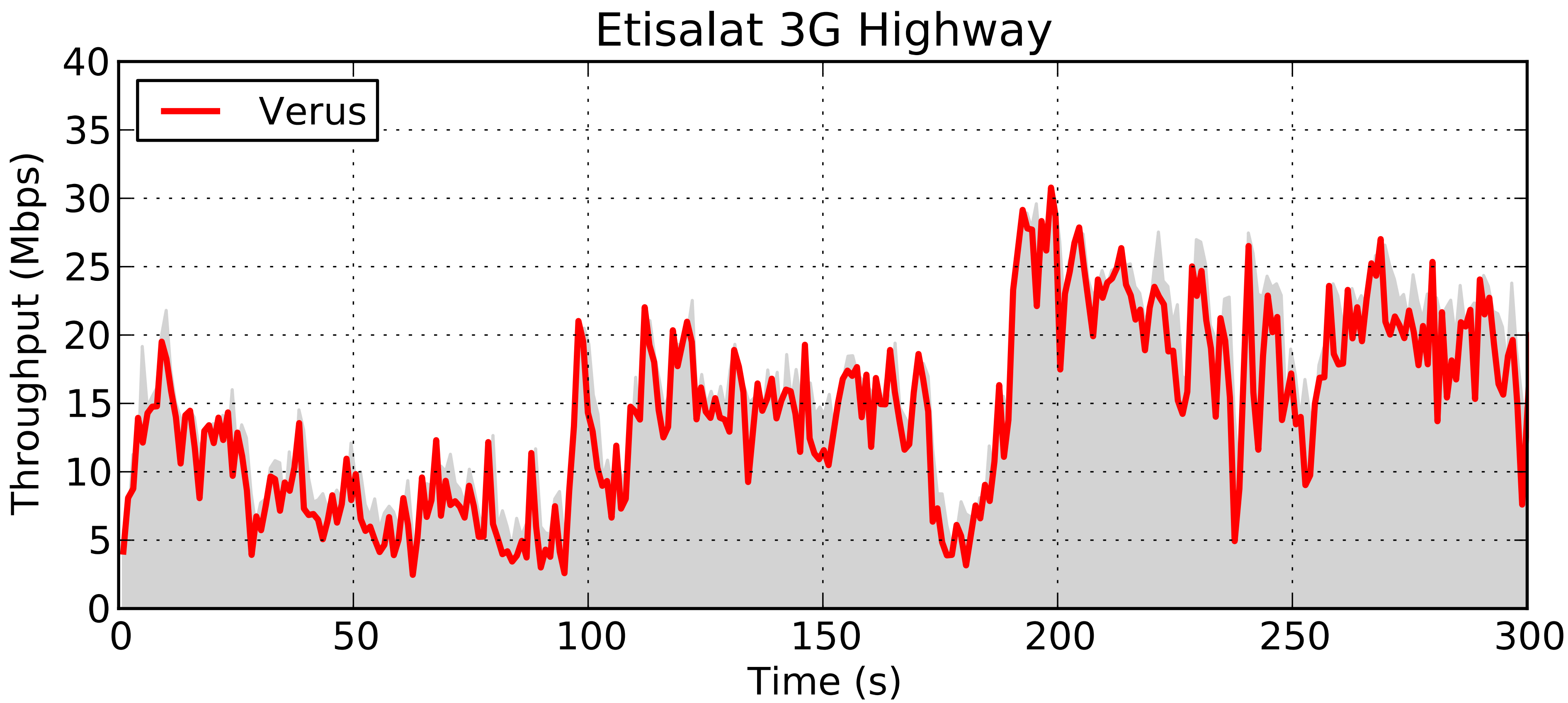
Why do we need a new congestion control protocol for cellular?

TCP cubic over Cellular

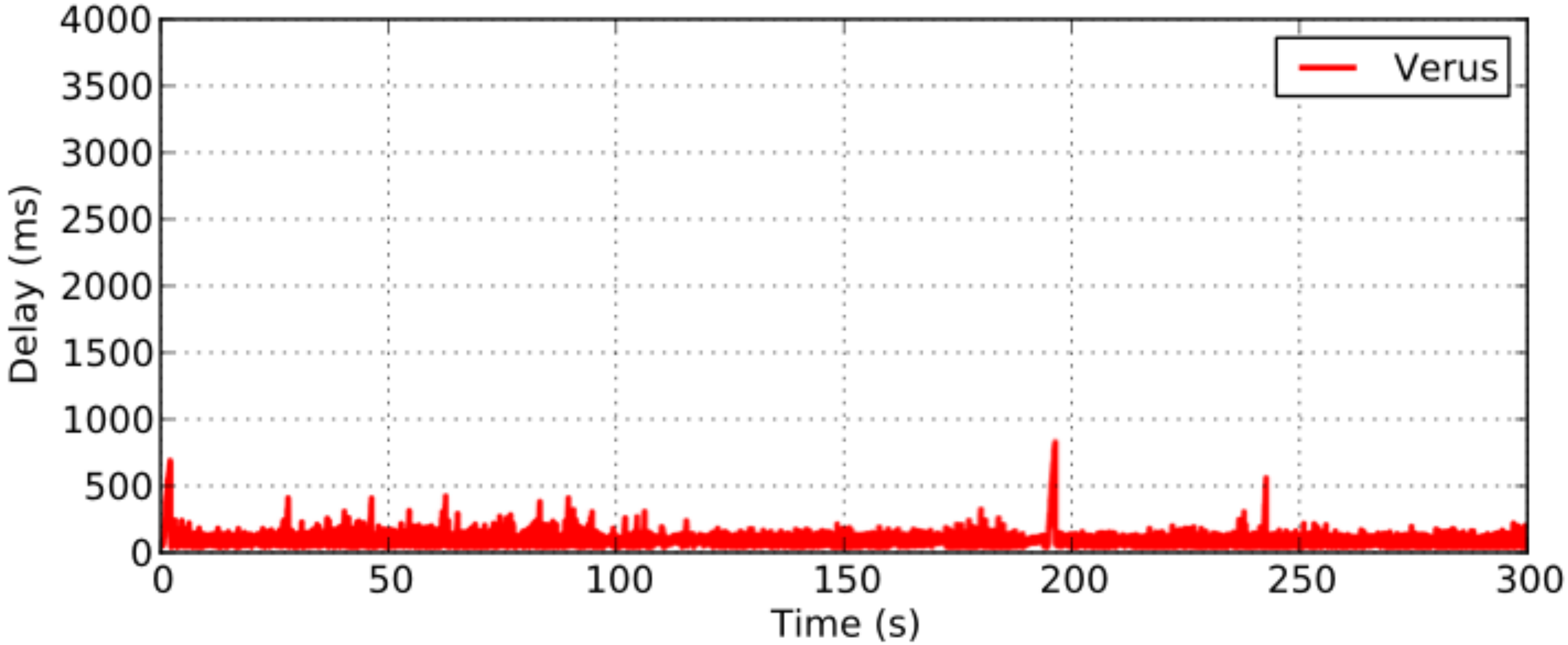


Can we do better?

Verus



Etisalat 3G Highway
Verus
Throughput (Mbps)
Time (s)
0
5
10
15
20
25
30
35
40
0
50
100
150
200
250
300



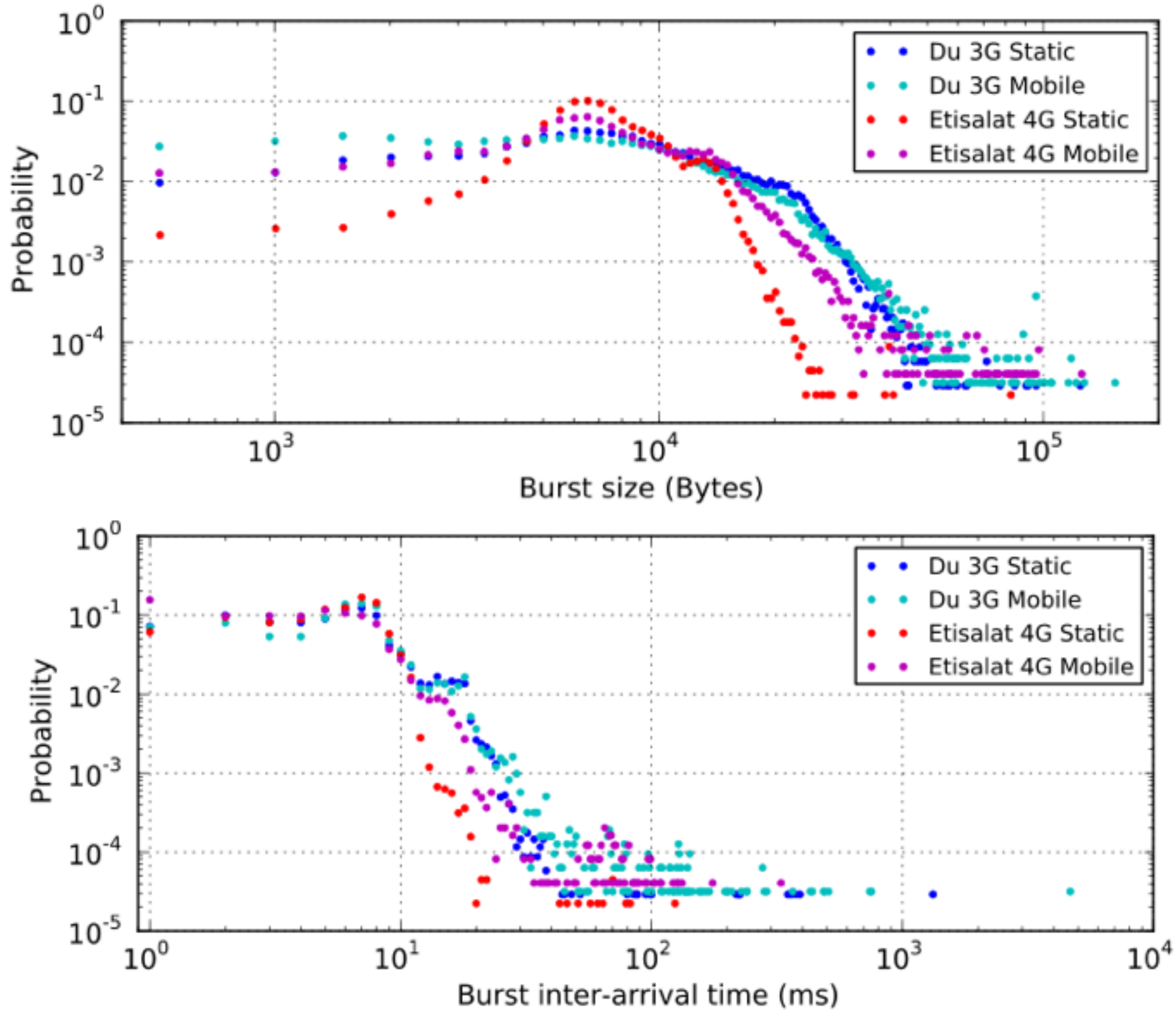
Verus
Delay (ms)
Time (s)
0
500
1000
1500
2000
2500
3000
3500
4000
0
50
100
150
200
250
300

Why is cellular different?

- Sprout* was a big motivation for our work
- Cellular networks experience:
  - Rapidly varying link rates
  - Occasional multi second outages
  - Deep packet queues
- Sprout assumes:
  - Stochastic modeling of the cellular channel
  - Self-interaction is the dominant factor in end-to-end delay rather than competing traffic

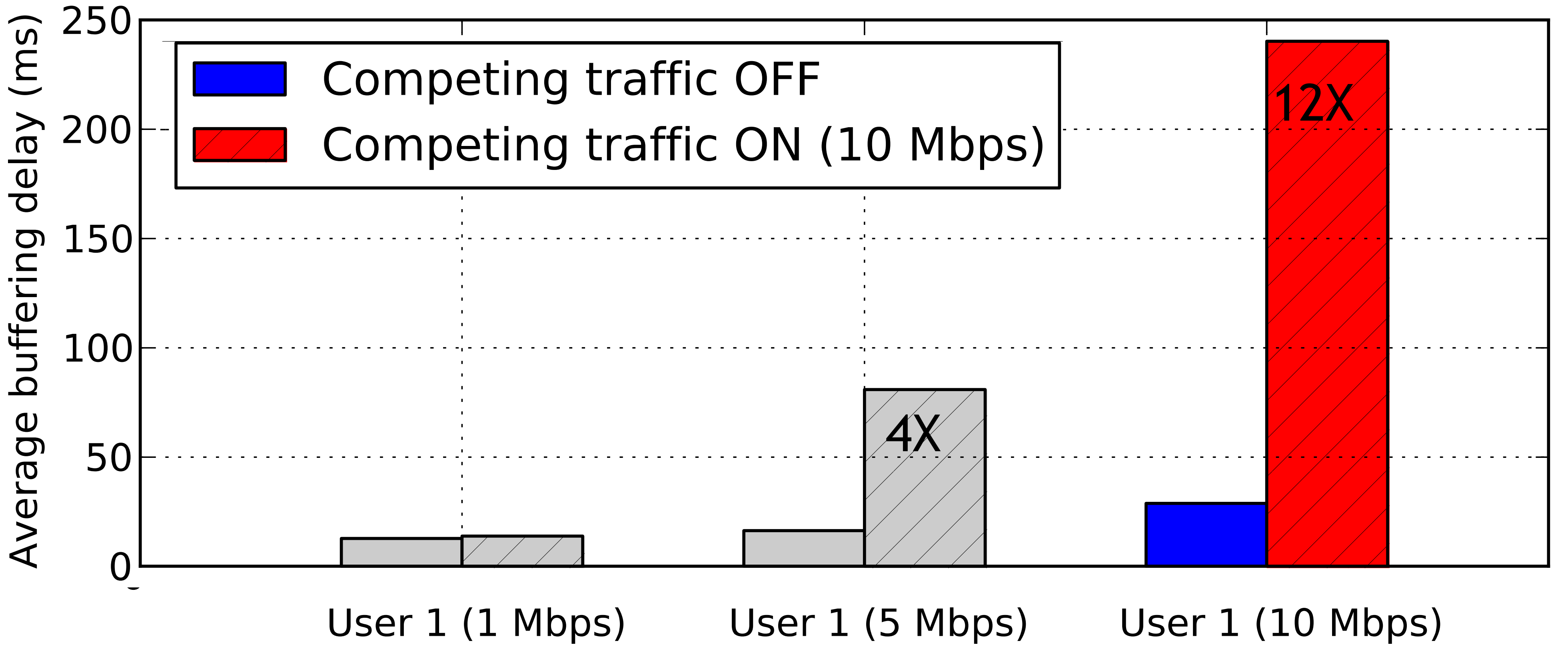
*Keith Winstein et. al., 'Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks', in NSDI 2013

Cellular channels are hard to predict



Competing traffic has a major impact

- Two users over real 3G network:
  - User 1 is constantly receiving a downlink stream
  - User 2 with ON/OFF 10 Mbps downlink stream



Verus

- Is an adaptive congestion control protocol for cellular networks

- Design goals:
  1. Track fast channel changes
  2. Balance throughput and delay
  3. Provide fairness between competing flows

Verus design

- Congestion control protocol requires some input:
  - Channel/network state modeling
  - Explicit network cooperation e.g. ECN
  - Loss detection
  - Delay feedback

- We use delay feedback
  - Only change end nodes
  - Proactively avoid congestion
  - Cheap signaling overhead

Verus design

- Don’t do channel prediction/modeling

- Build on TCP concepts:
  - Use slow start
  - Use Multiplicative Decrease (MD) on packet loss
  - Replace Additive Increase (AI) with a step based increase/decrease

- Learn the relationship between delay and sending window
  - Delay curve

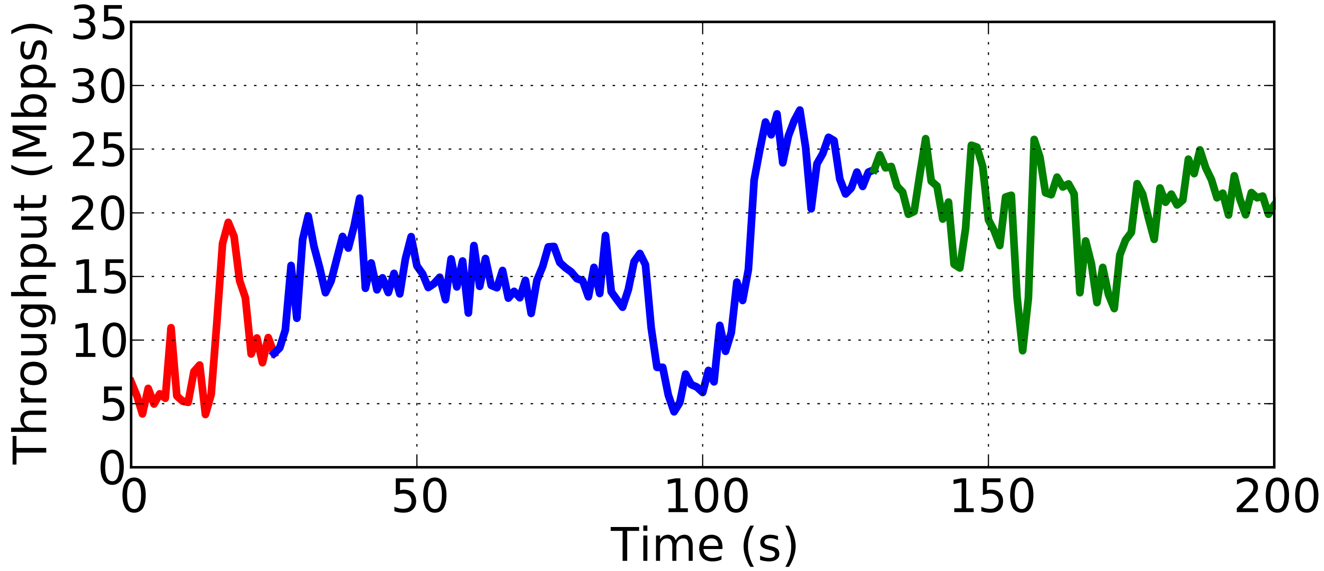
Verus key idea

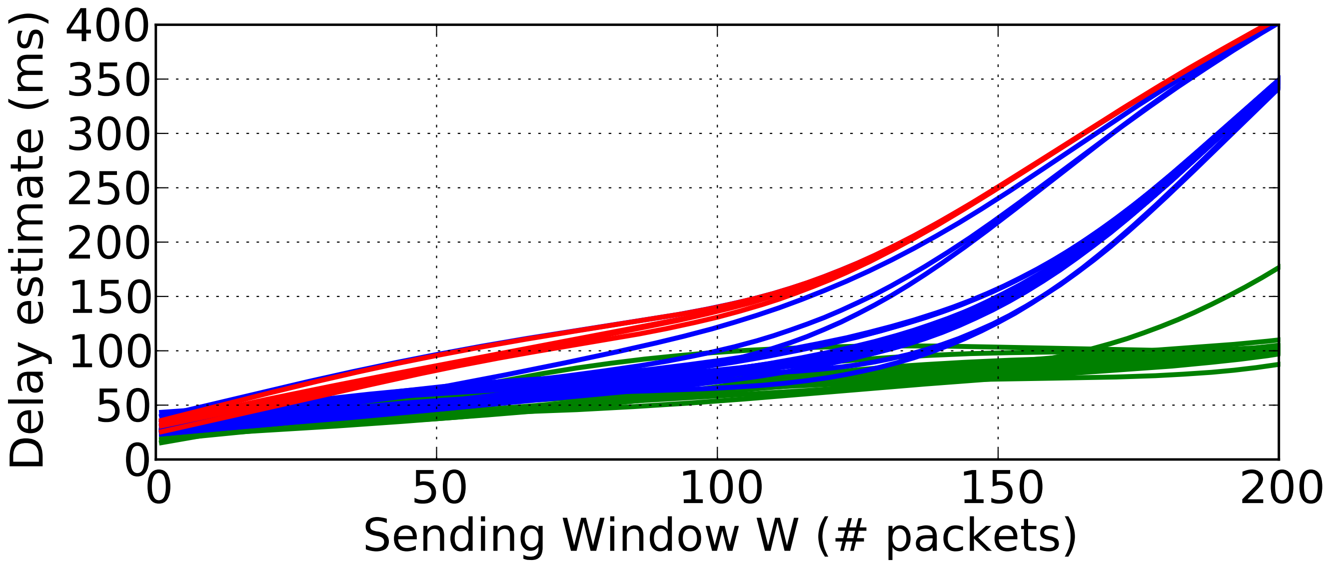
- Every 5 ms epoch we:
  - Estimate network delay
  - Infer the number of packets to be sent (W) to avoid congestion

$$W(t+1) = \boxed{f(\underbrace{d(t) \pm \delta(t)}_{\text{Estimated delay } (D_{est})})} \text{ Delay curve}$$

Delay curve

- A way to track network changes
- Reflects relationship between sending window and network delay
- Verus dynamically learns the network state
  - Through delay feedback (ACKs)





Verus estimated delay

- The estimated delay ($D_{est,i+1}$) at every epoch is:

$$D_{est,i+1} = \begin{cases} D_{est,i} - \delta_1 & \text{if } \Delta D_i > 0 \\ D_{est,i} + \delta_2 & \text{else} \end{cases}$$

Increased network delay:
reduce estimated delay

Decreased network delay:
increase estimated delay

- δ_1 and δ_2: decrease and increase steps (1 ms and 2 ms)
- ΔD: delay difference between $D_{max,i}$ and average $D_{max,i-1}$
- $D_{max,i}$: maximum delay during last epoch

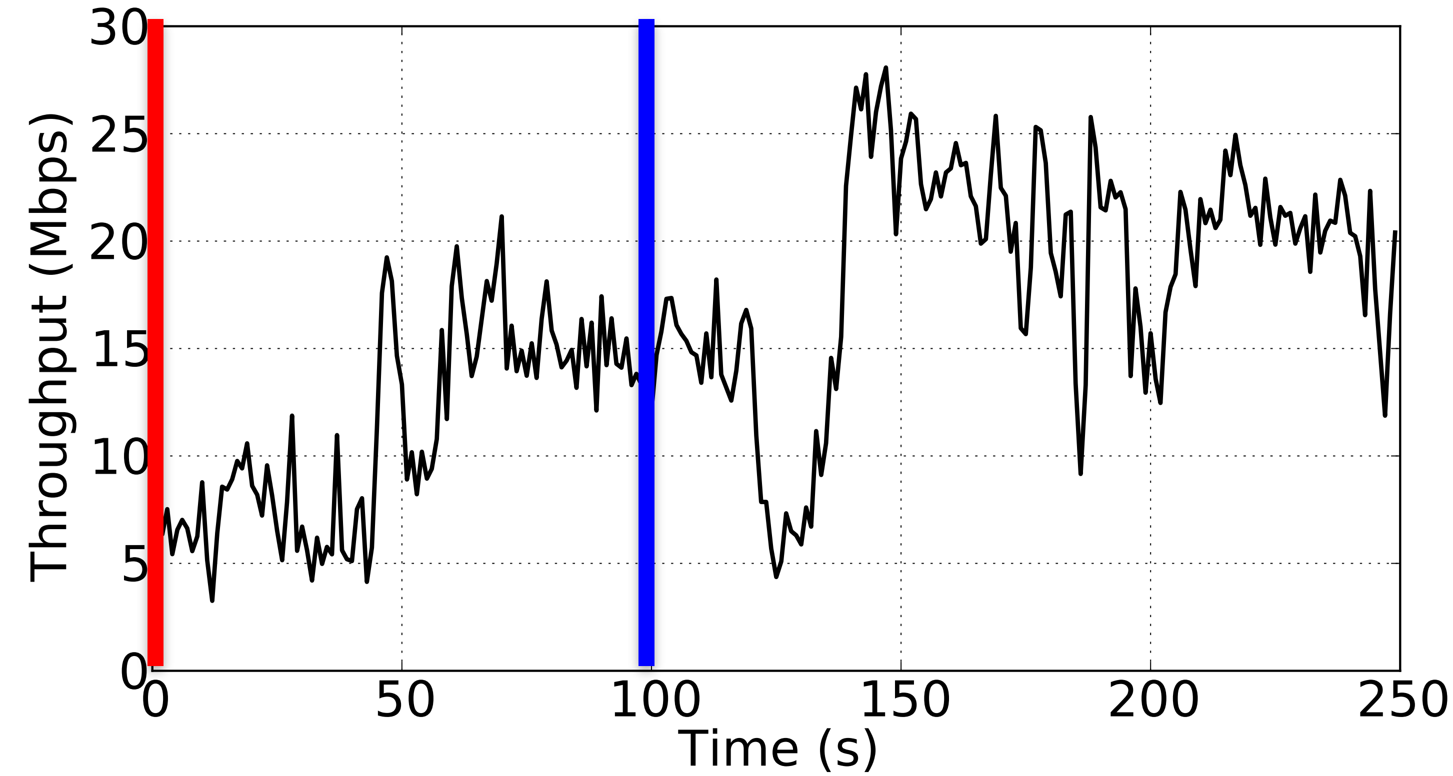
Goal 1: Tracking fast channel changes

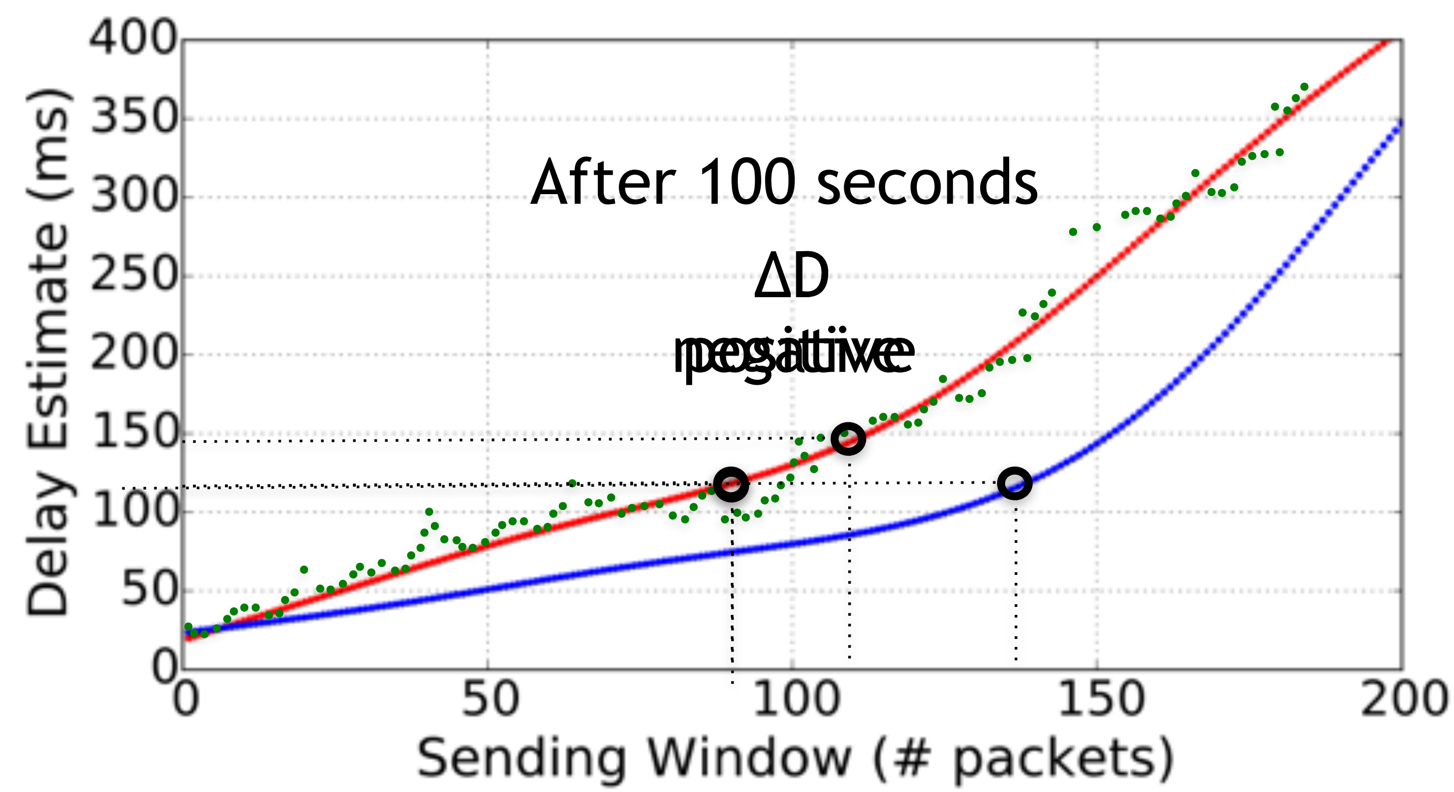
Slow start:
- Every ACK: add a point (W, delay)

Build delay curve:
- Cubic spline interpolation

Verus control loop:
- every epoch 5 ms

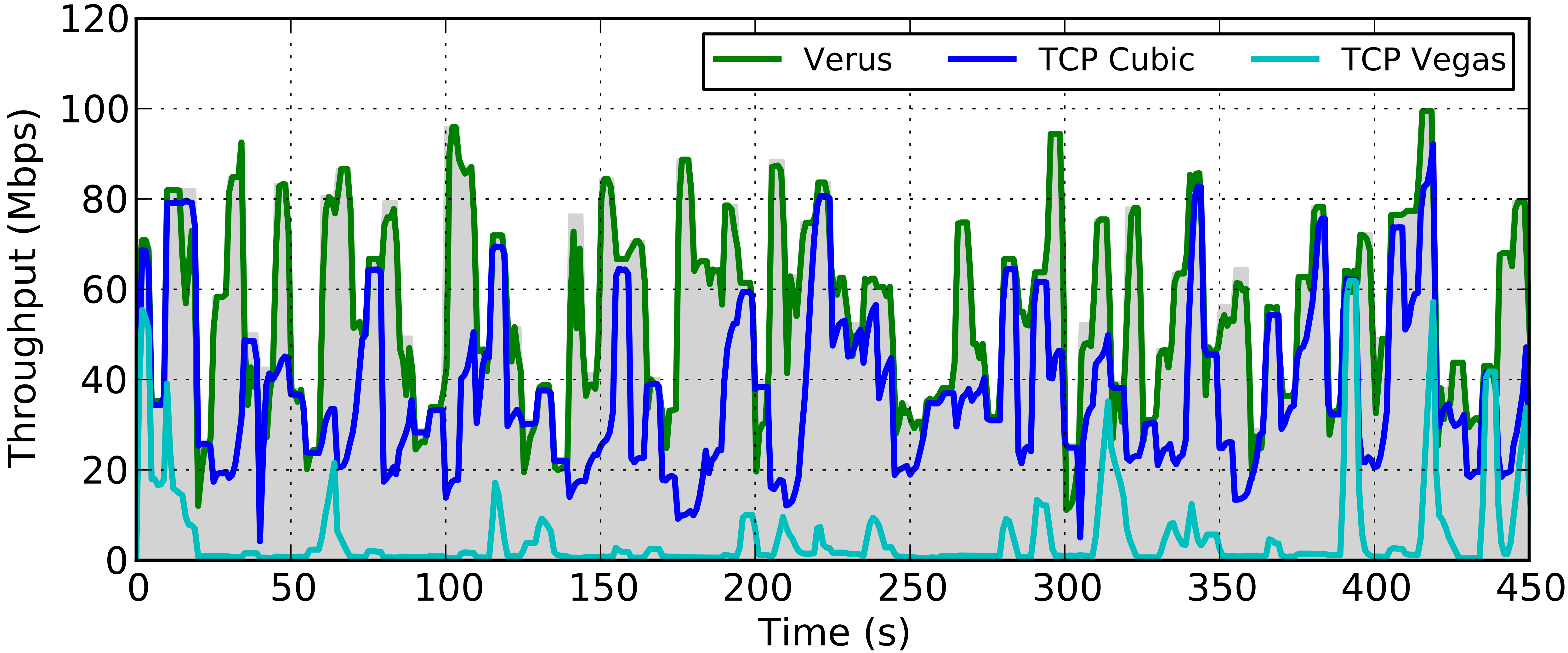
Rebuild delay curve:
- every 1 second





Goal 1: Tracking fast channel changes

<u>Every 5 sec:</u>
Link: 10-100 Mbps
Round trip time: 10-100 ms
Losses: 0-1 %



Goal 2: Balance throughput and delay

- Verus D_{est} is upper bounded by R

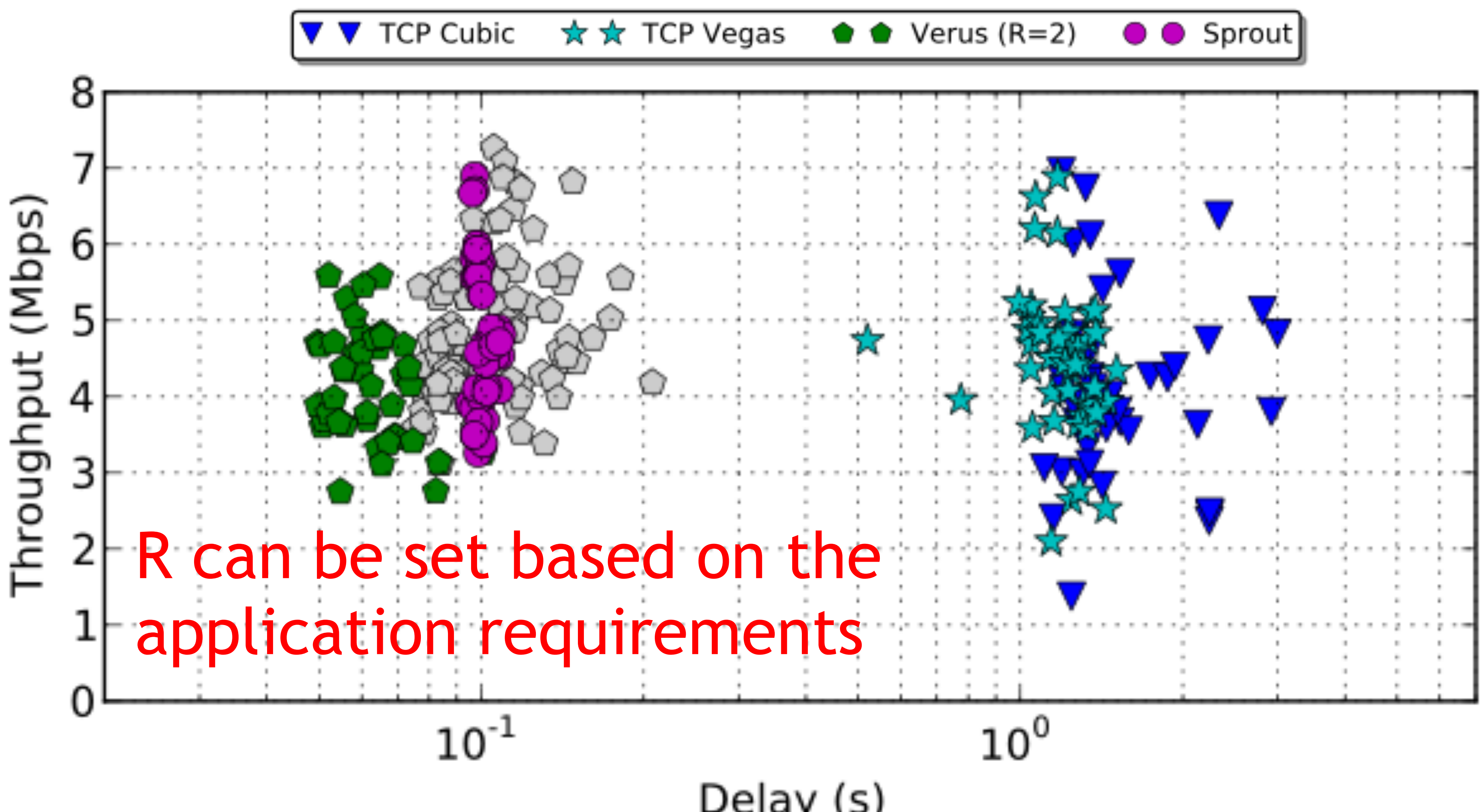
$$D_{est,i+1} = \begin{cases} D_{est,i} - \delta_2 & \text{if } \frac{D_{max,i}}{D_{min}} > R \\ D_{est,i} - \delta_1 & \text{elif } \Delta D_i > 0 \\ D_{est,i} + \delta_2 & \text{otherwise} \end{cases}$$

- R defines the ratio between max and min delay of the network

- Setting R specifies the trade-off between throughput and delay

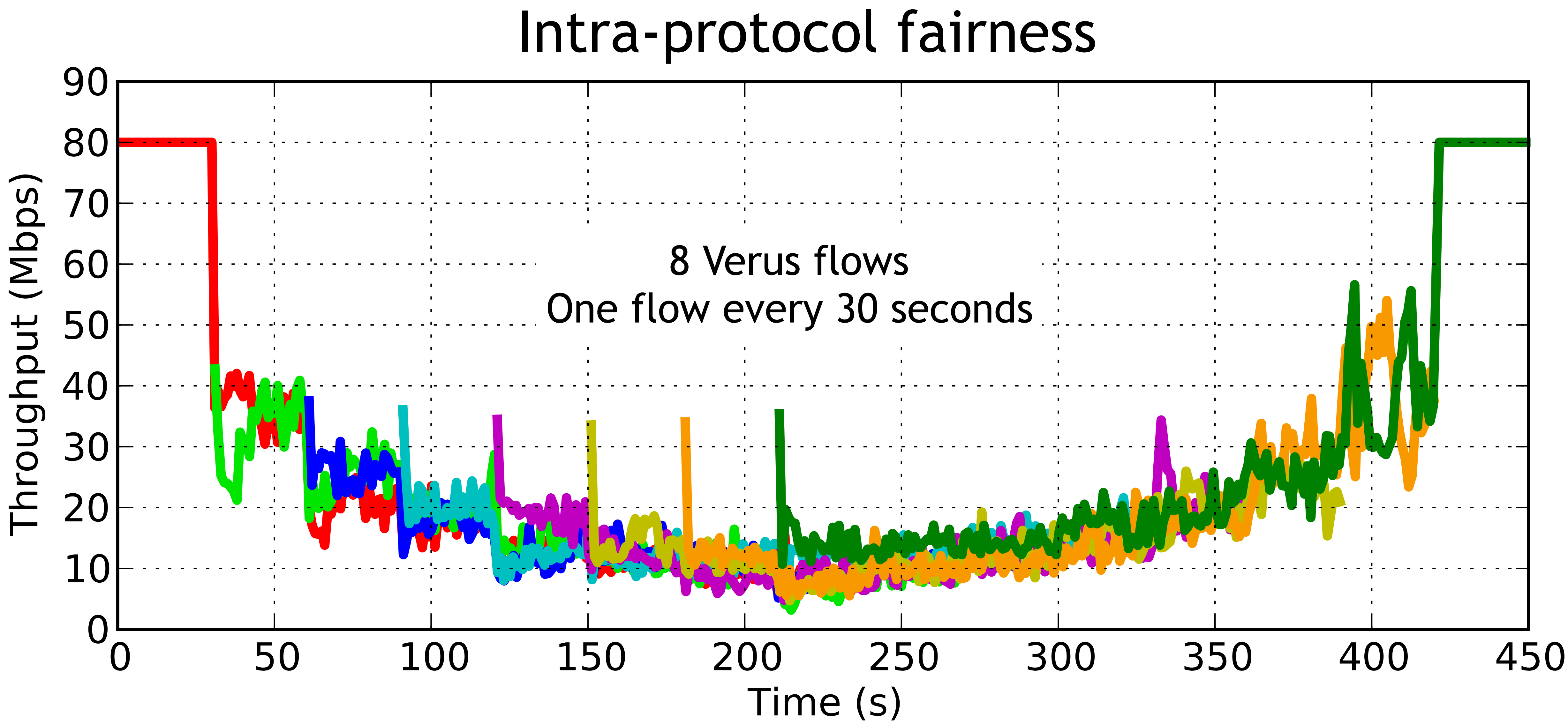
Goal 2: Balance throughput and delay

Experiments over real LTE network:
- Stationary scenario
- 3 phones each running 3 flows
- Repeated 5 times each

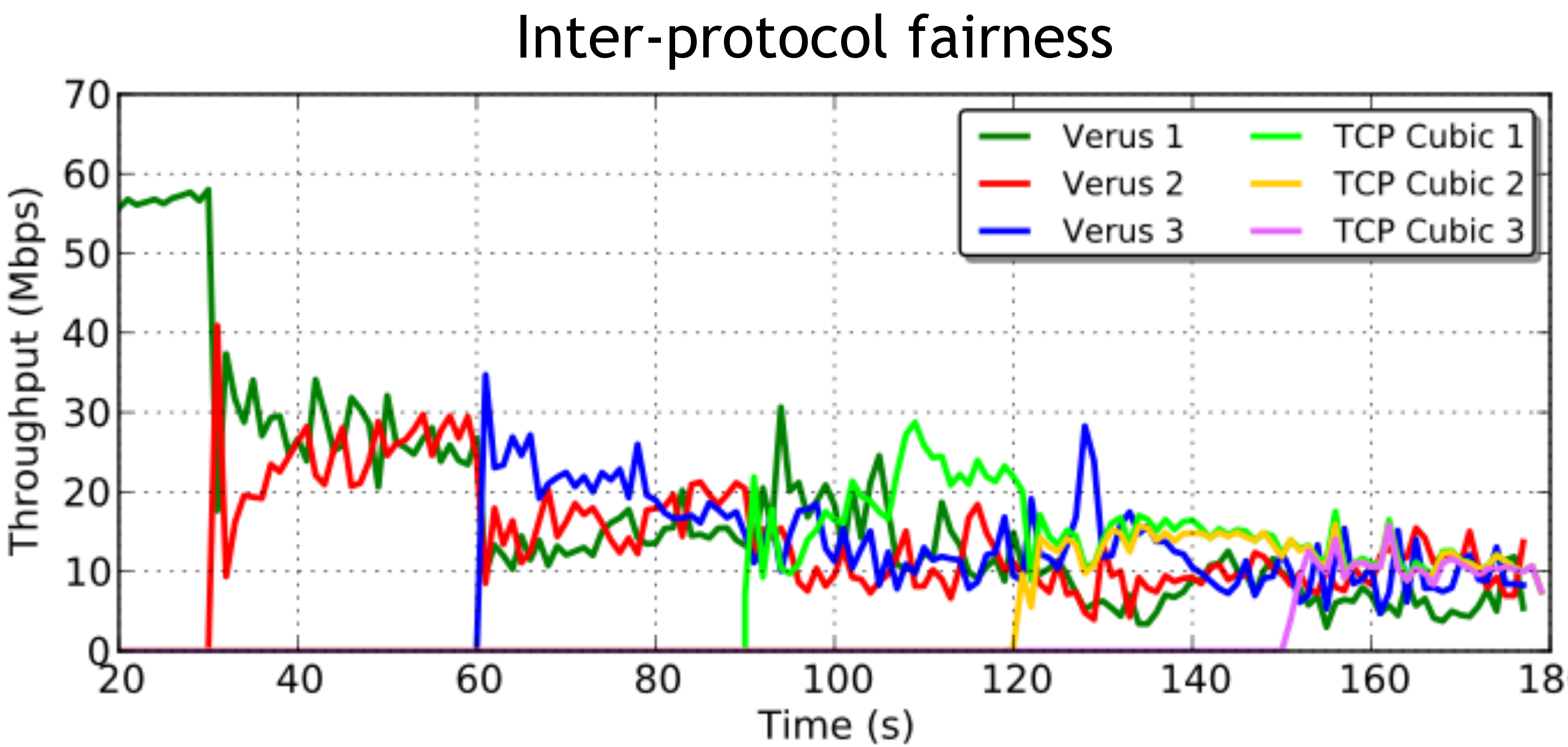


R can be set based on the application requirements

Goal 3: Provide fairness between competing flows



Goal 3: Provide fairness between competing flows



Verus source code is open source and available on github: http://yzaki.github.io/verus/