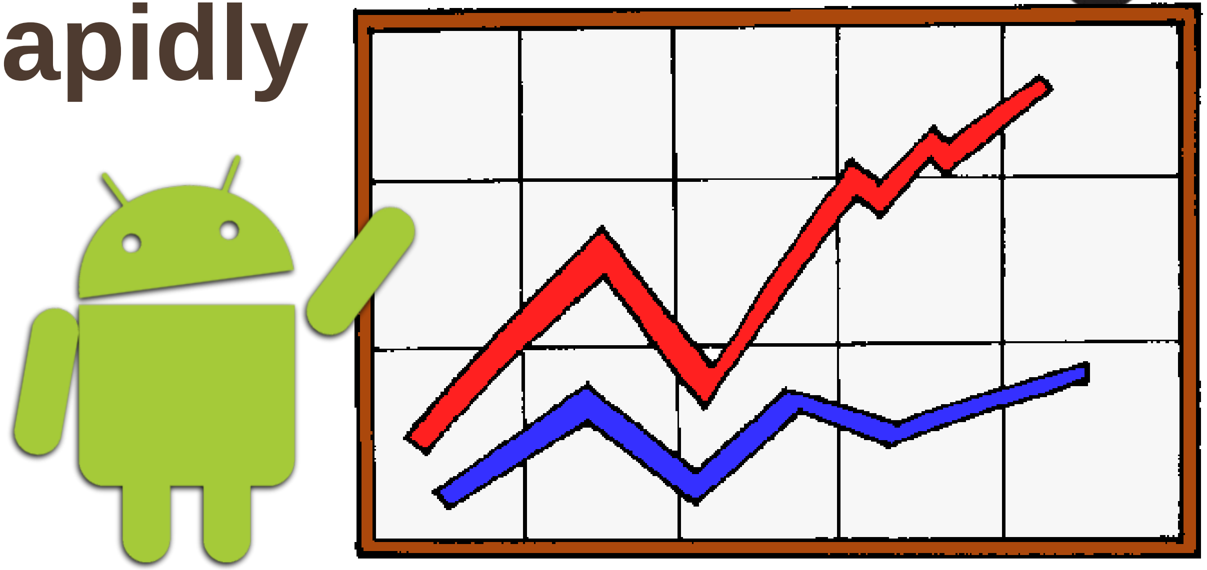
Dynamic Regulation of Mobile 3G/HSPA Uplink Buffer with Receiver-Side Flow Control

Yin Xu, Wai Kay Leong, Ben Leong
National University of Singapore

Ali Razeen
Duke University

Smartphones are everywhere

- US smartphone penetration exceeded 50% in Q2, 2012
- Mobile data traffic growing rapidly as well



Source: http://www.chetansharma.com/USmarketupdateQ22012.htm

Not just for surfing the web…

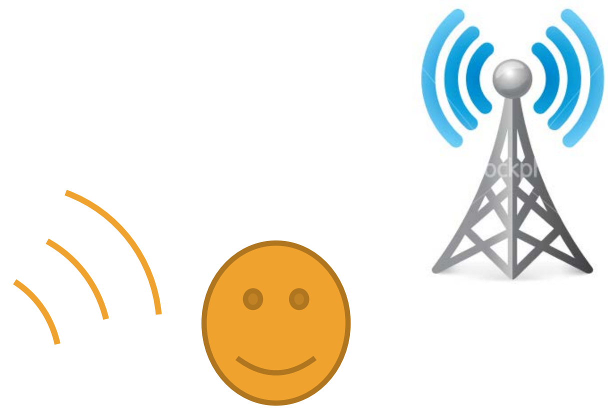
- **Users uploading significant amounts of data in the form of photos and videos**

e.g. AT&T observed 40% more data uploaded than downloaded during a football match

(7 Feb 2012)

- **Uploads often conducted in the background**

What are the users doing?



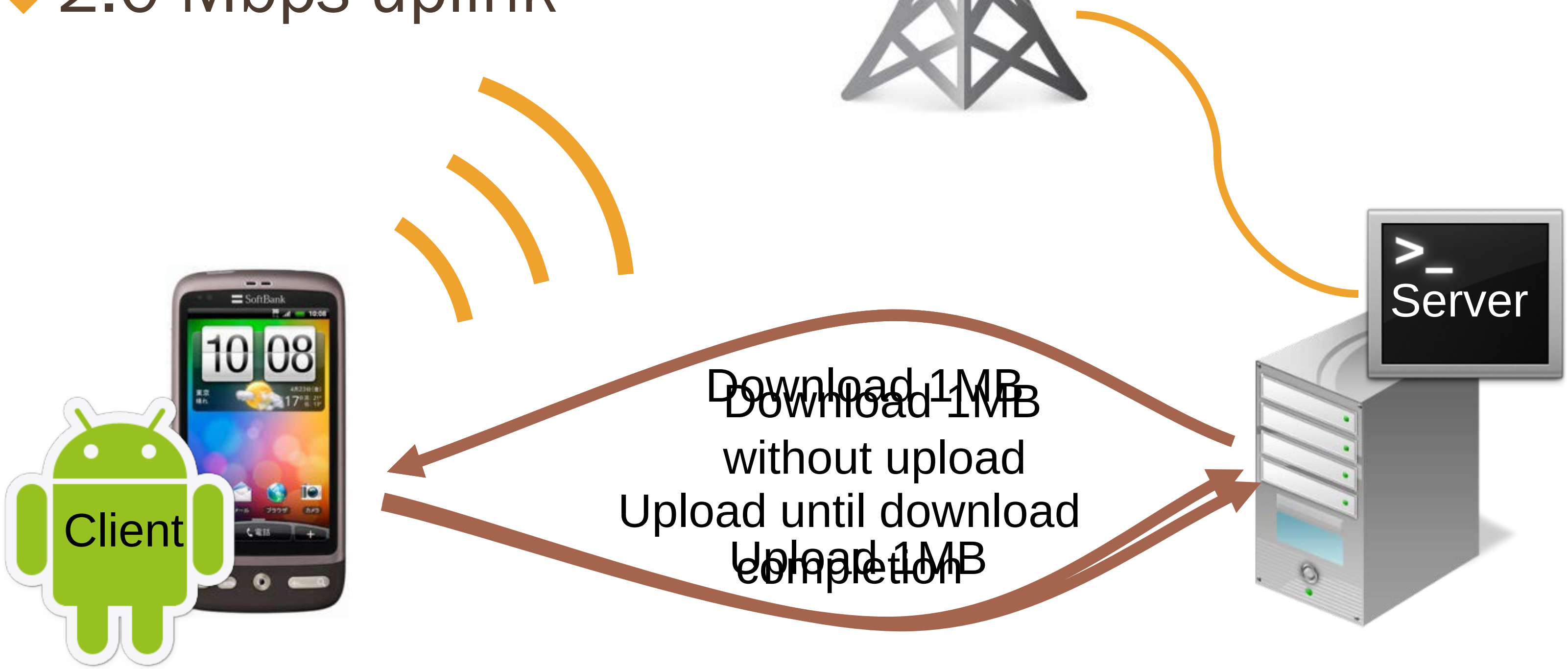
What happened?

The Problem

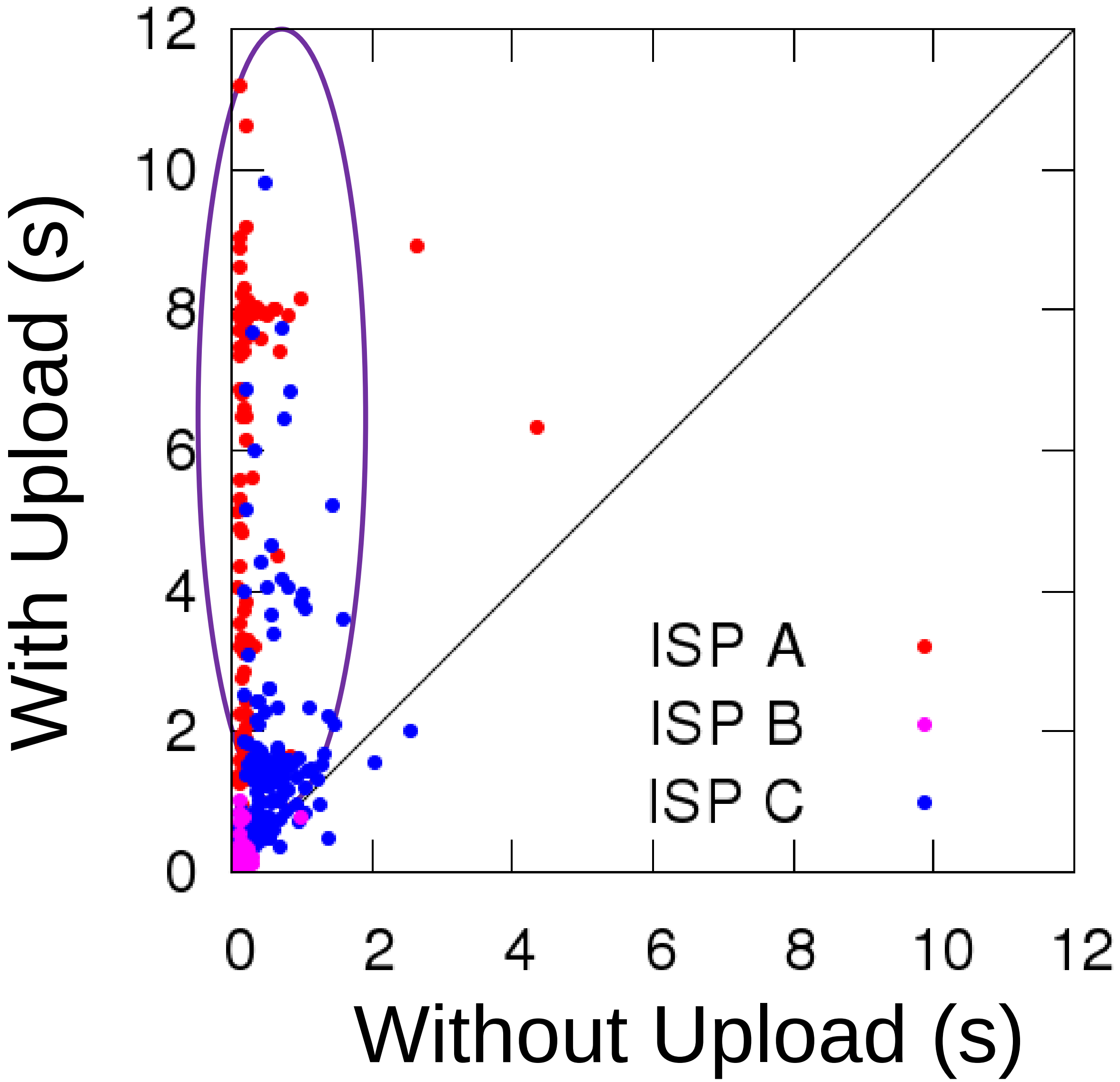
Background uploads can degrade downloads significantly!

Experiment Setup

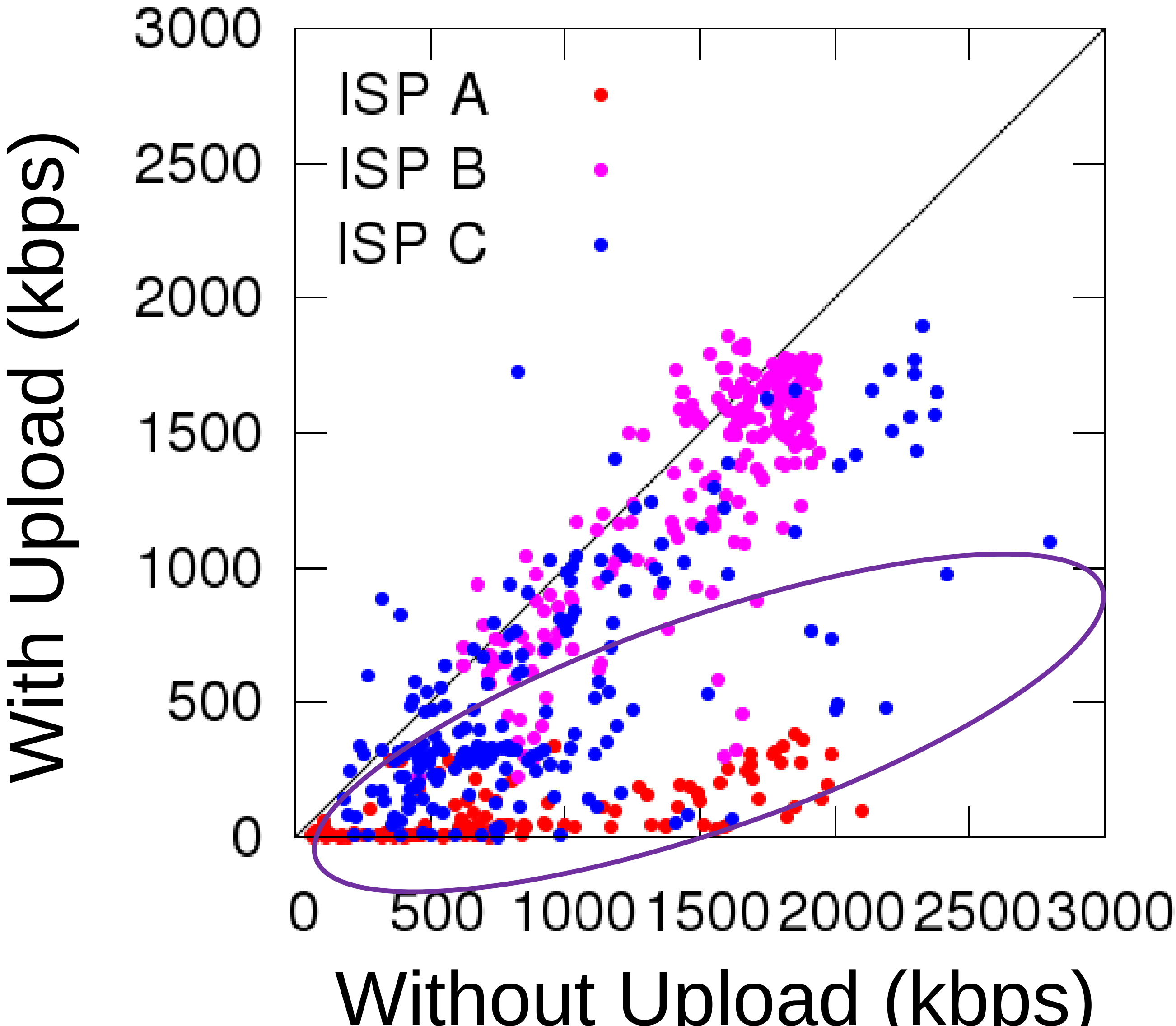
- Android Phones
- 3 Local Telcos
  - 7.2 Mbps downlink
  - 2.0 Mbps uplink



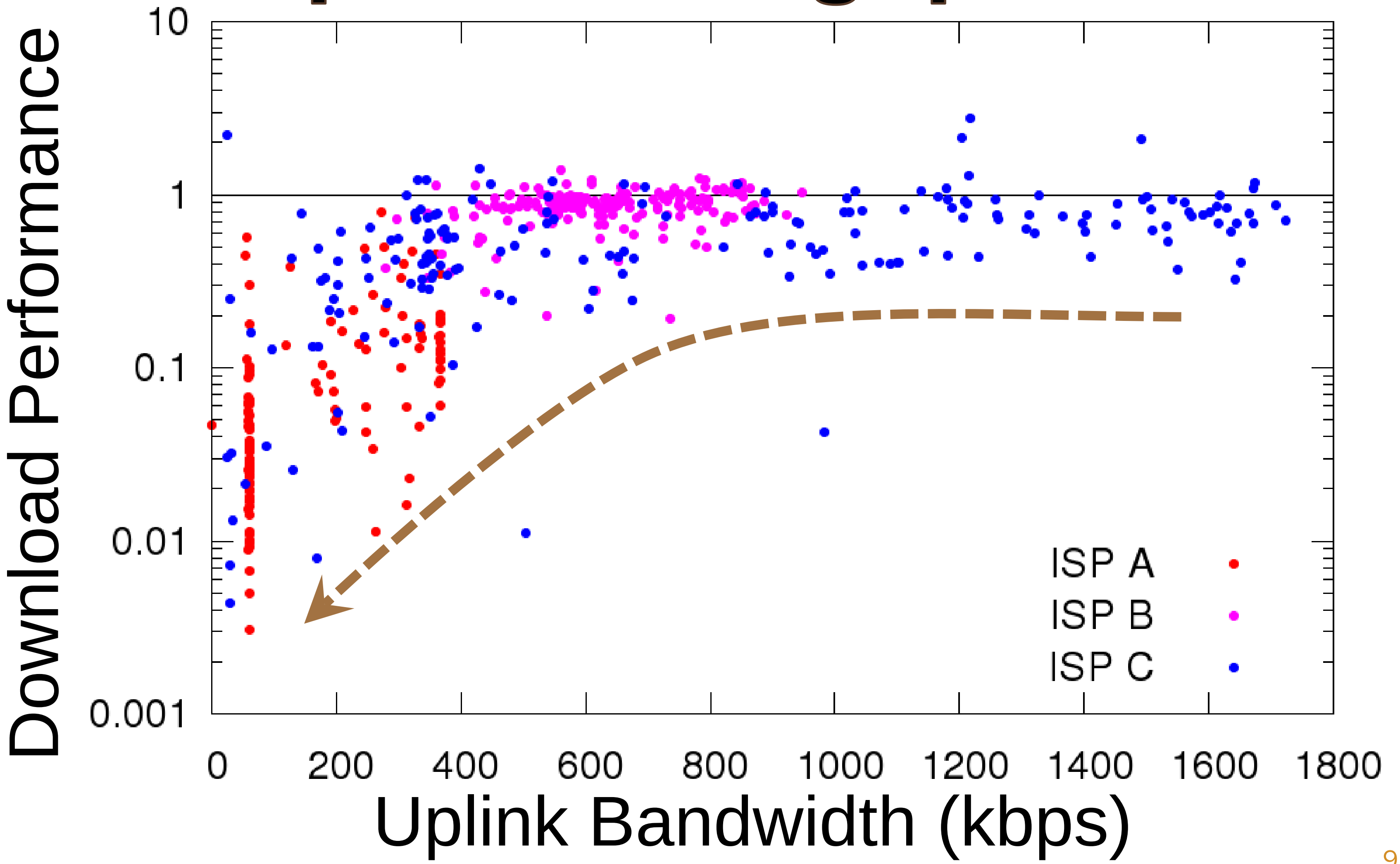
Download Roundtrip Time



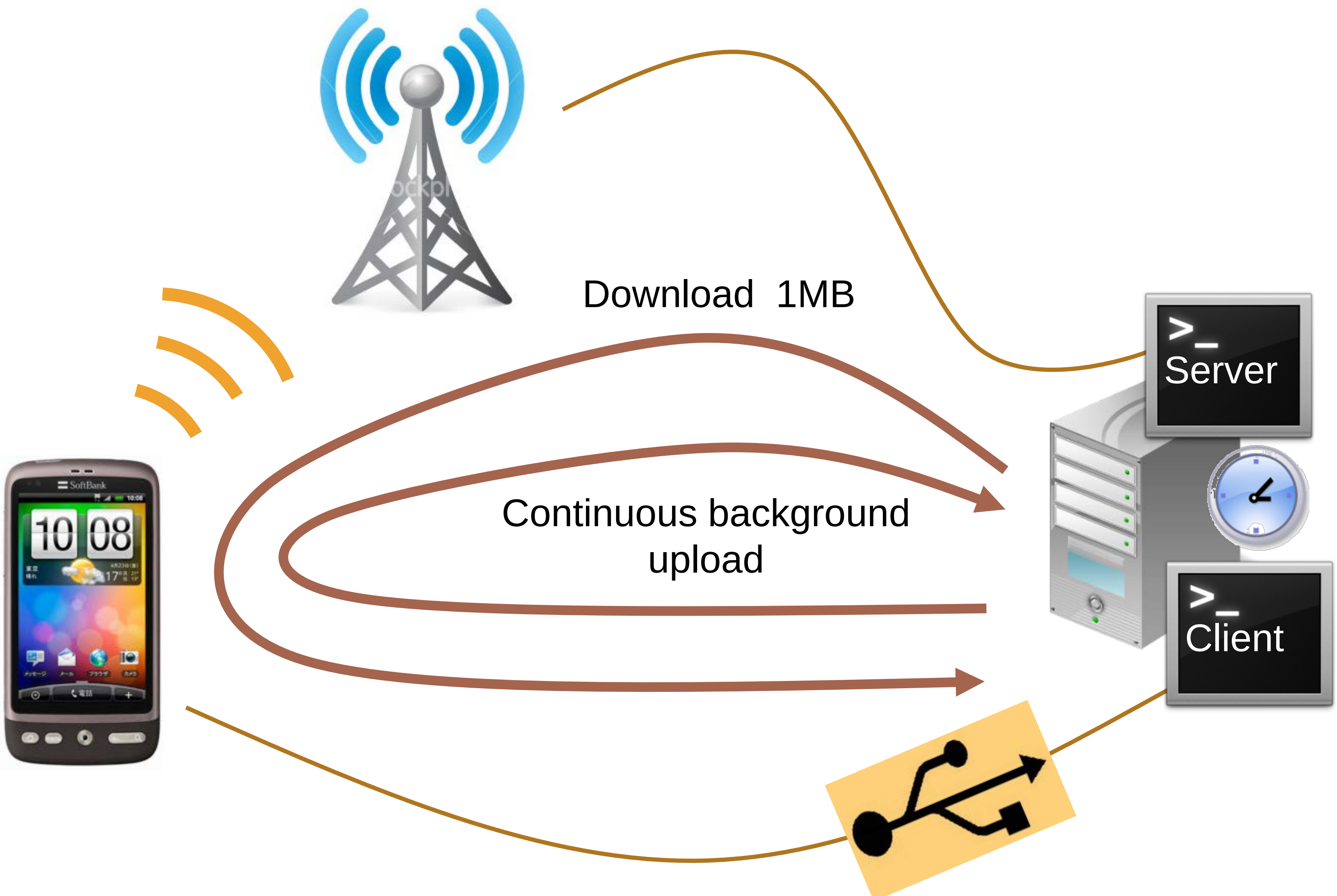
Download Throughput



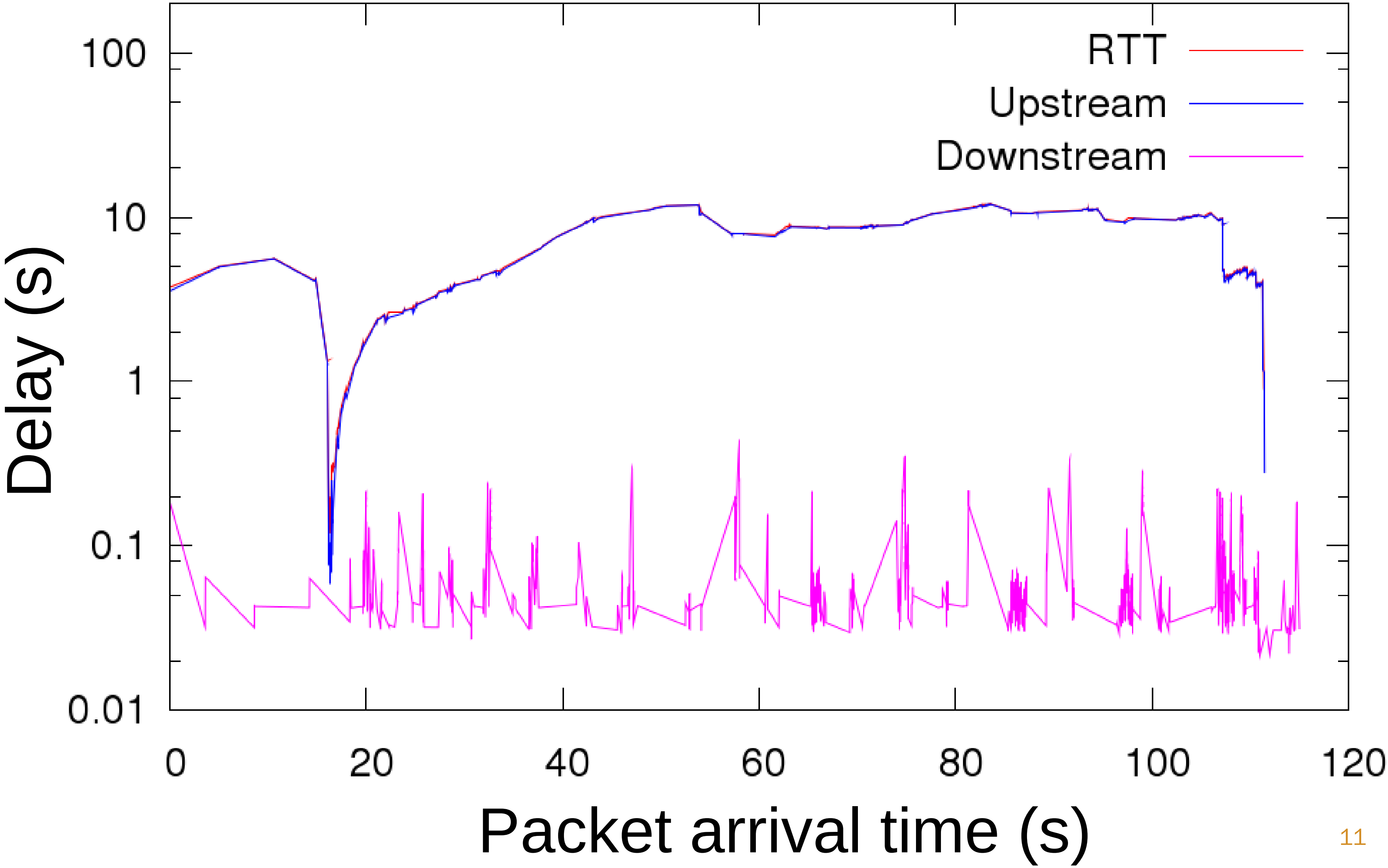
Problem is exacerbated by low uplink throughput



Experiment Setup (Loopback)



RTT dominated by uplink delay

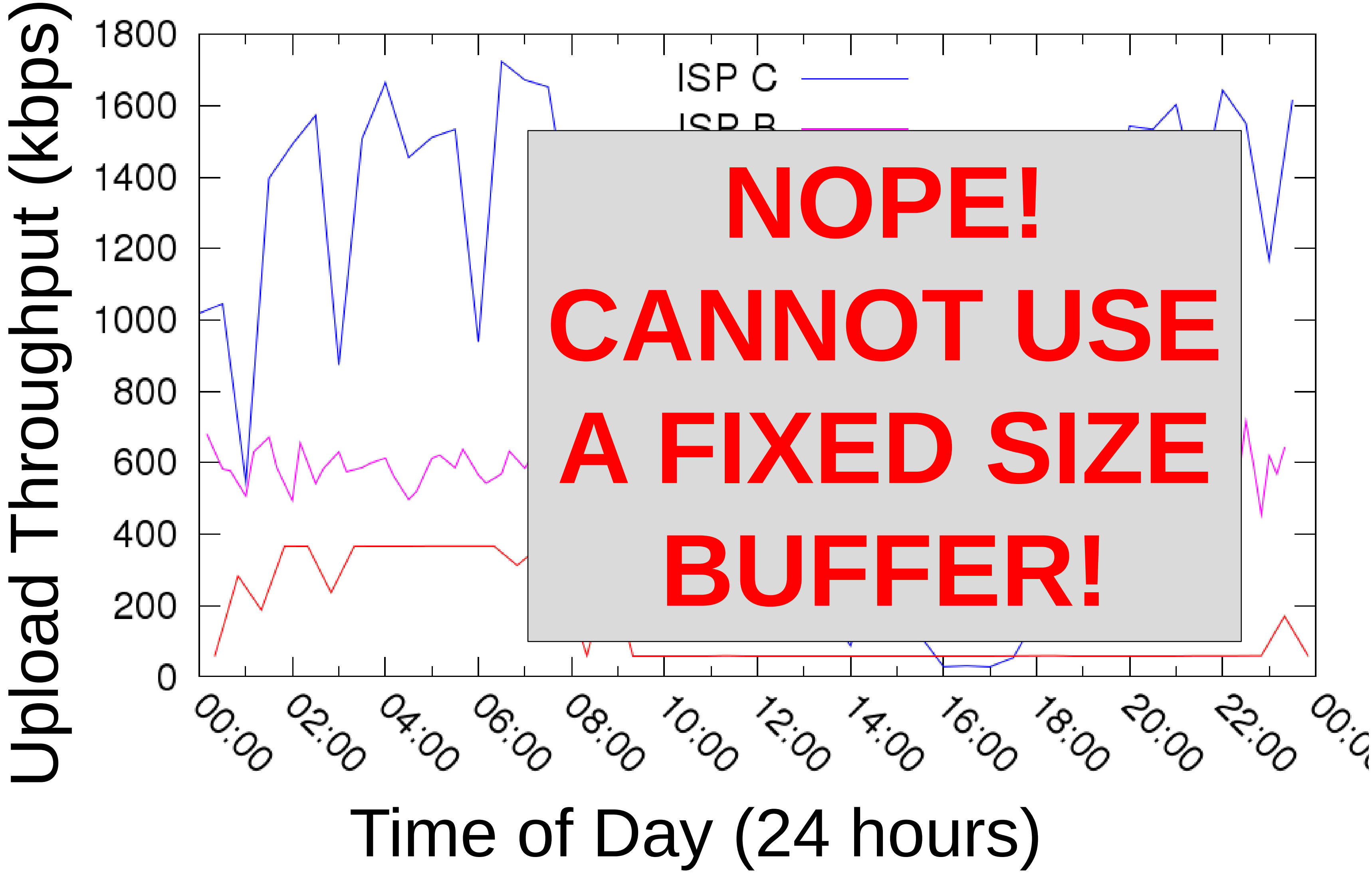


Understanding the Problem

- **NOT caused by ACK Compression**
- **Data Pendulum Problem** *[Heusse et al. 2011]*
  - Sized properly, buffers take turns to fill up
  - Sized improperly, low-speed link with large buffer becomes the sole bottleneck
- **Uplink is the bottleneck in a 3G/HSPA mobile network**

Can we just size the uplink buffer correctly?

Understanding the Problem



Previous Solutions

- **Optimizing how the ACKs are sent** *[Balakrishnan et al. 1999/2002]*
- **Using different queues for data and ACK packets** *[Podlesny et al. 2012]*
- **TCP Vegas** *[Brakmo et al. 1995]*

All *Sender-Side* Solutions

Why not Sender-Side Solutions?

- **Not General**
  - Only works for the devices already deployed with the solution
  - Devices may use network interfaces other than 3G/HSPA (e.g. Wi-Fi)
- **"Implement complexity at the server, not the client"**
  - It may take years to update client-side software *[Adya et al. 2011]*

Our Solution

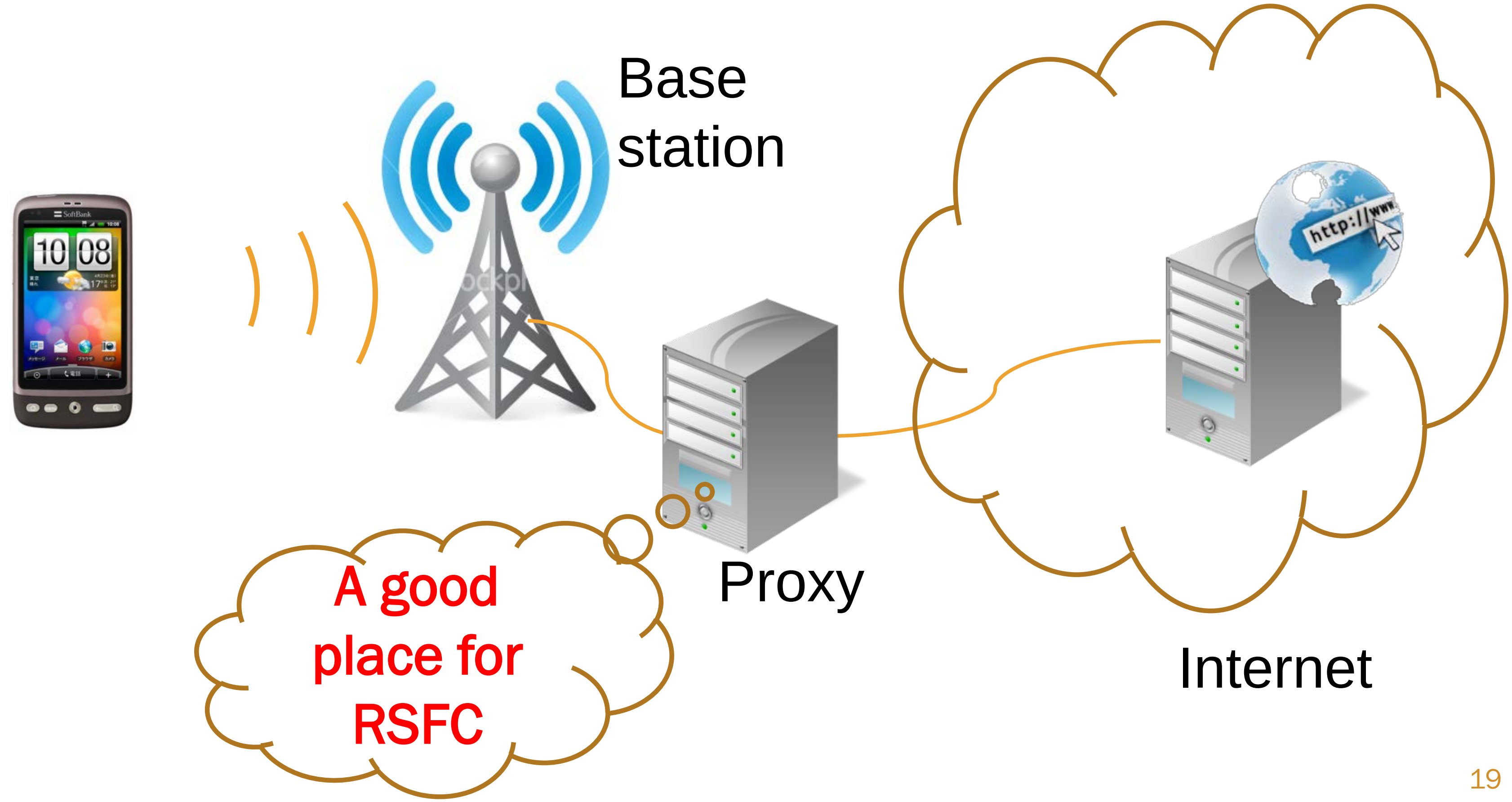
Receiver-Side Flow Control (RSFC)

Our Approach

- Can implement at ISP network proxies
- Works transparently for any device using the 3G/HSPA mobile network
- Changes immediately deployable

Practical Deployment

Easily deployed at ISP proxy



Receiver-Side Solutions

- **Freeze-TCP** *[Goff et al. 2000]*
- **Reducing delay for interactive applications while maintaining throughput for bulk transfers** *[Spring et al. 2000]*
- **Improving fairness** *[Kalampoukas et al. 2002, Andrew et al. 2008]*

Used for Other Purposes

Key Idea

Reduce # of packets in the uplink buffer by adjusting the TCP receiver window (`rwnd`)

What is the right `rwnd`?

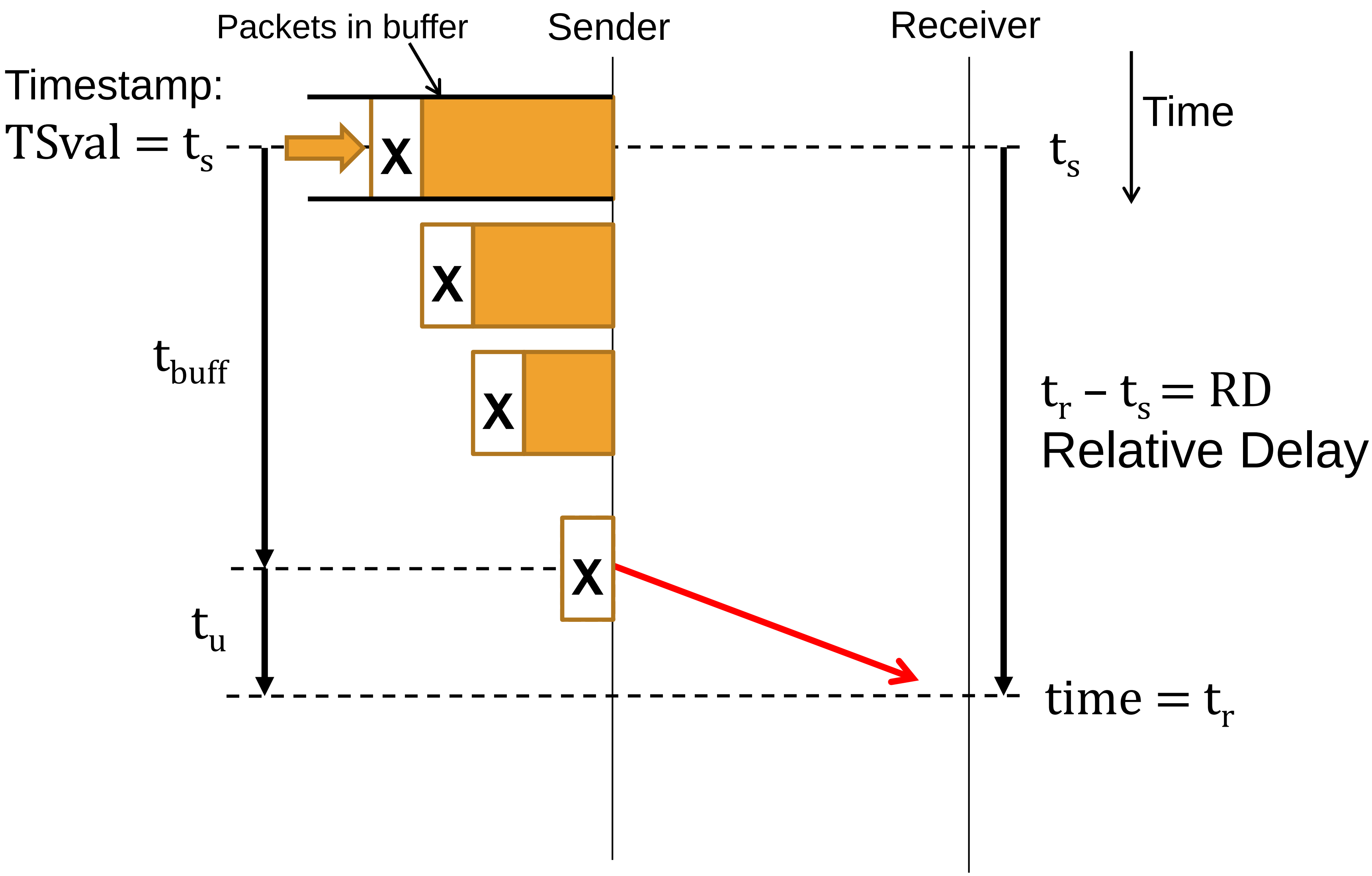
rwnd Value

- **Set to bandwidth-delay product (BDP)?**
- **Not so simple…**
  - How do we estimate BDP?
  - Network fluctuations

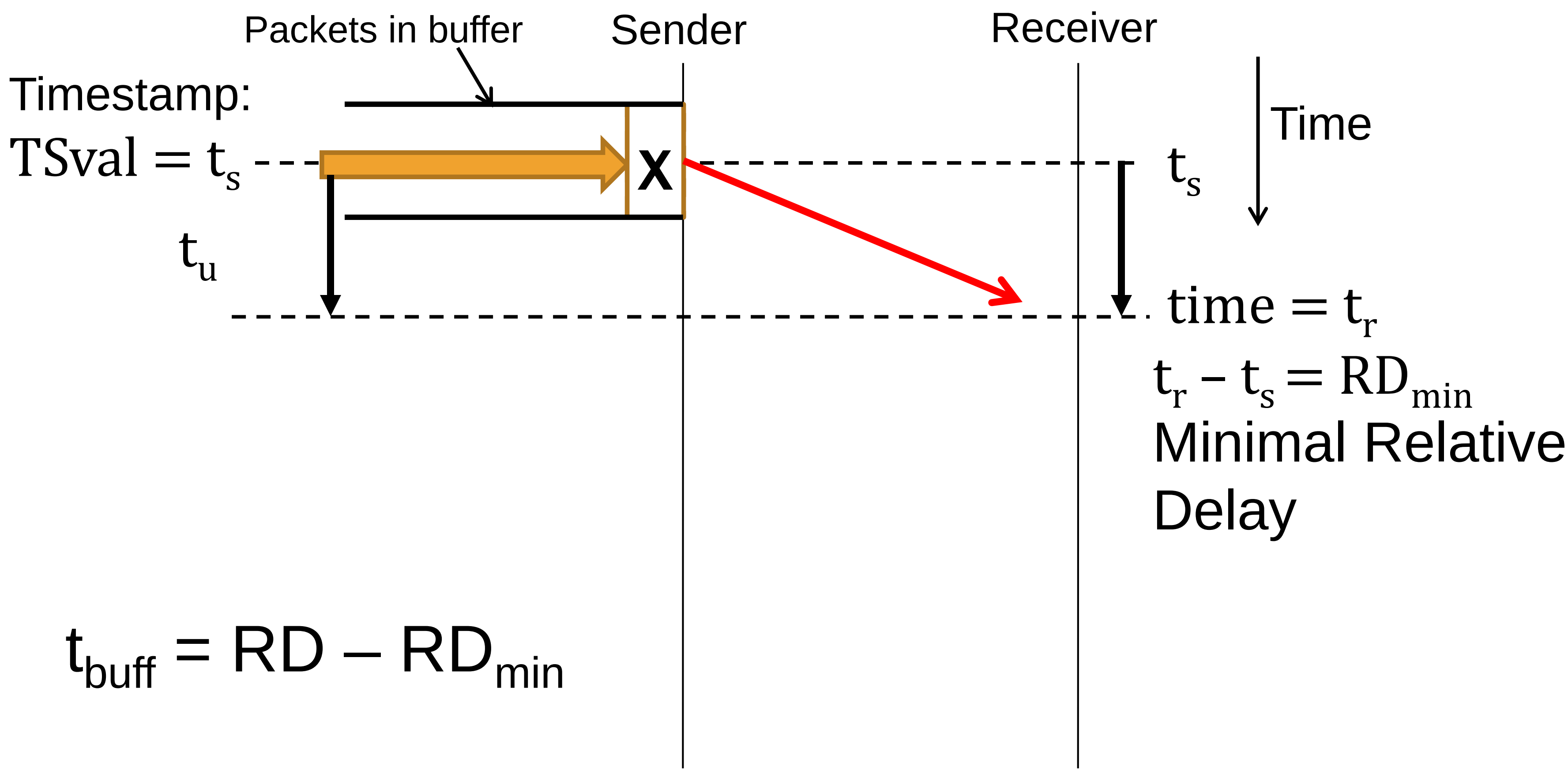
Approach: Negative Feedback

- Estimate time packet spends in buffer t_{buff} using TCP Timestamp
- Set a threshold T
  - $t_{buff} > T$, clamp rwnd
  - $t_{buff} < T$, increase rwnd

Estimating t_{buff}



Estimating t_{buff}



$$t_{buff} = RD - RD_{min}$$

No need to synchronize sender and receiver!

Estimating BDP

- Measure receive rate ρ at receiver
- Minimal RTT (RTT_{min})
- Ideal window is the bandwidth-delay product:
  - $rwnd = \rho \times RTT_{min}$

Summary

- $t_{buff} > T$, $rwnd = \rho \times RTT_{min}$ (fast state)
- $t_{buff} < T$, rwnd++ (slow state)
- In our implementation
  - T is set to RTT_{min}

Handling changes in the network

- Changes in bandwidth
- Decrease in the delay
- Increase in the delay
  - Slight increase:
    detect increased receive rate ρ
  - Large increase: **monitor** state

See details in paper!

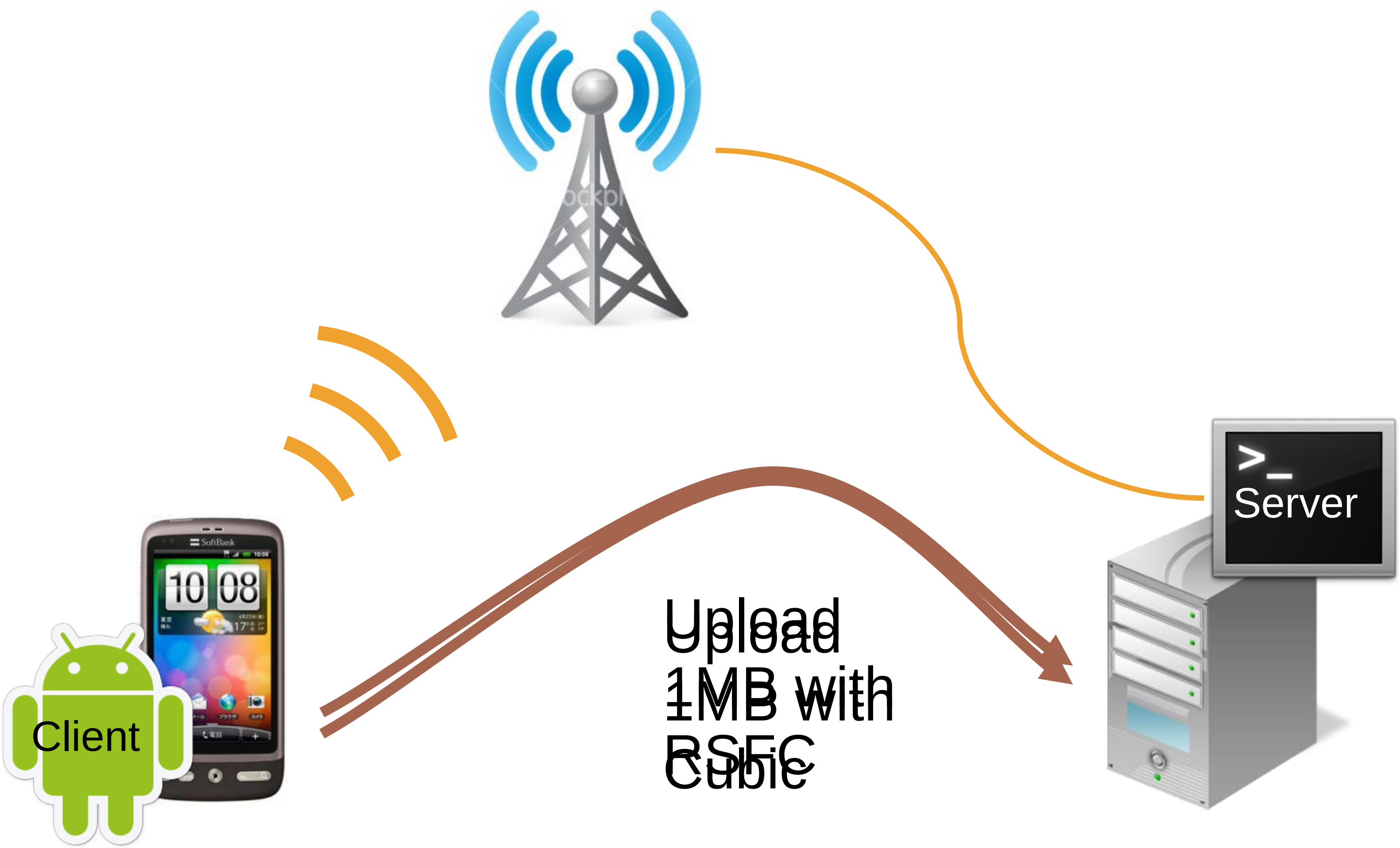
Evaluation

- Reduces RTT
- Improves download throughput
- Reduces webpage loading time
- Fair and efficient
- Adapts to changes in network conditions
- Compatible with sender-side algorithms

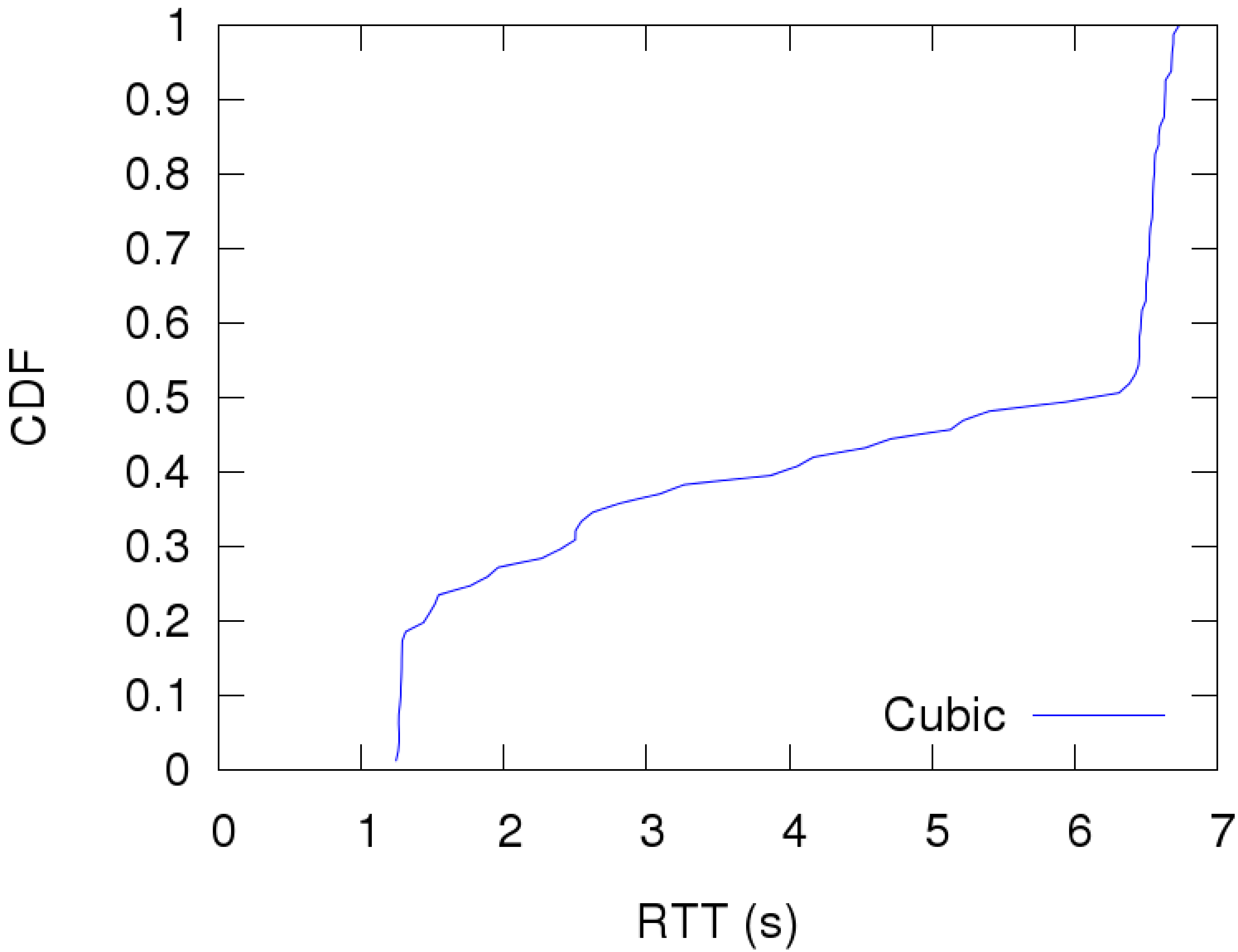
Evaluation

- Reduces RTT
- Improves download throughput
- Reduces webpage loading time
- Fair and efficient
- Adapts to changes in network conditions
- Compatible with sender-side algorithms

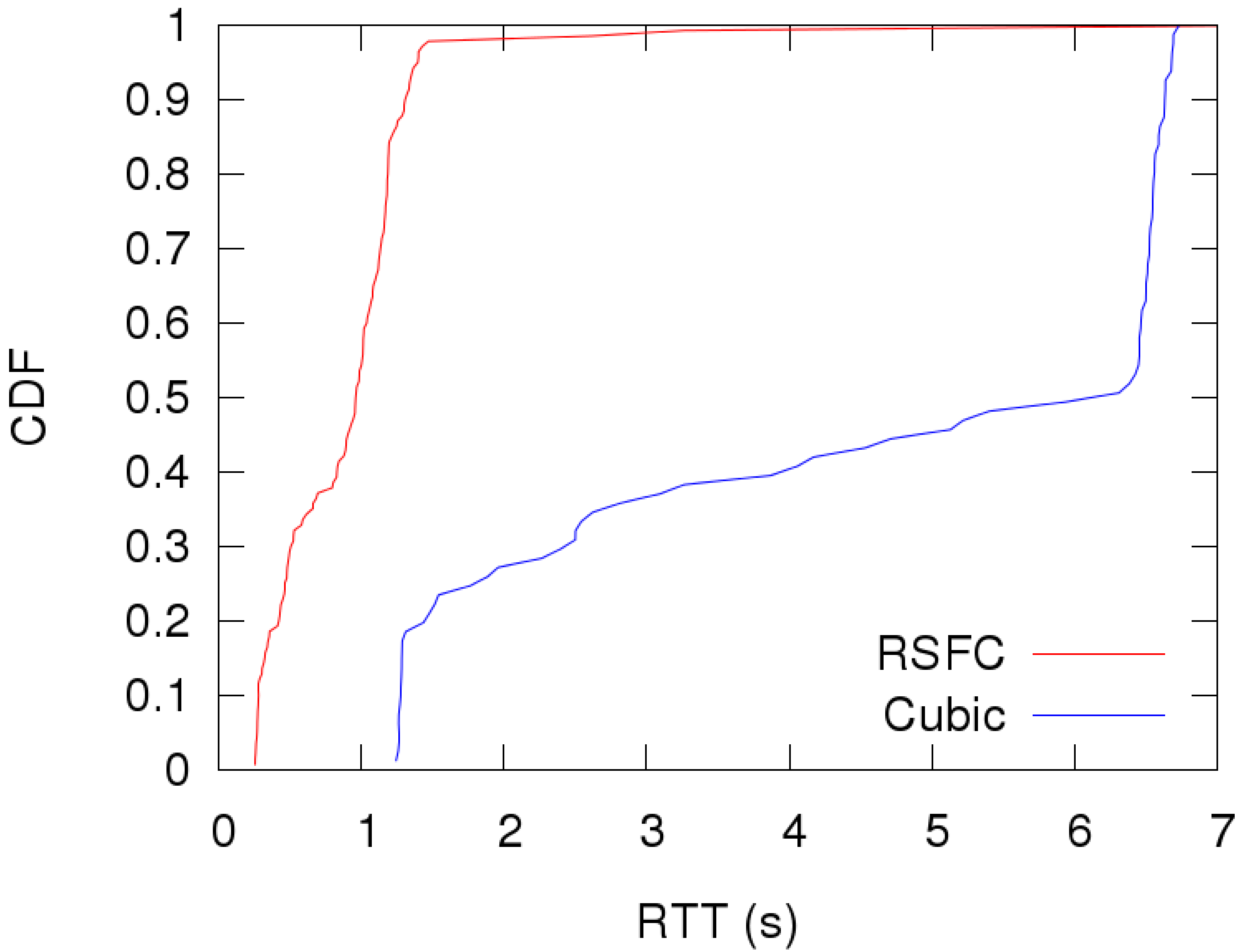
Reduce RTT



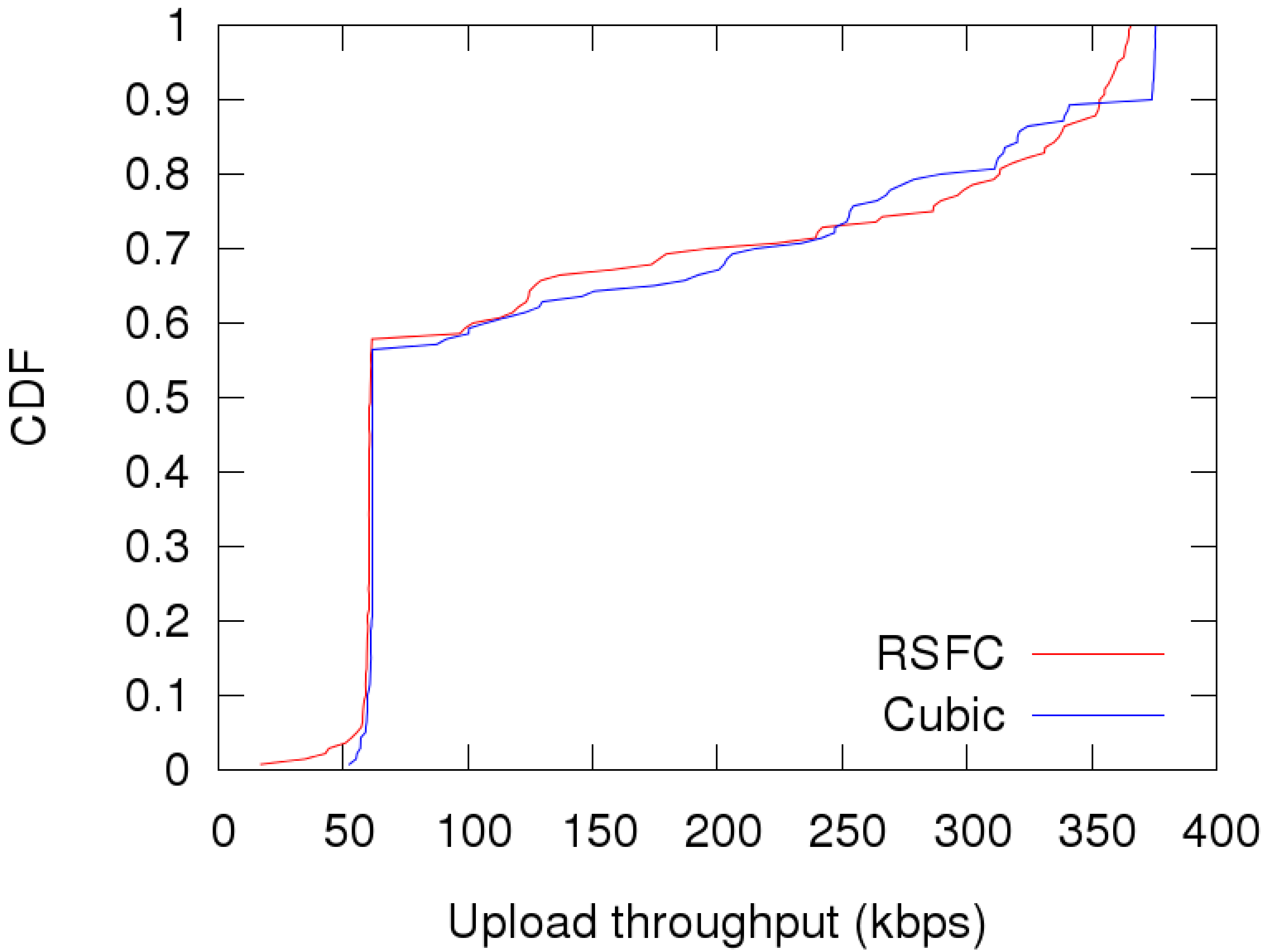
Reduction in RTT



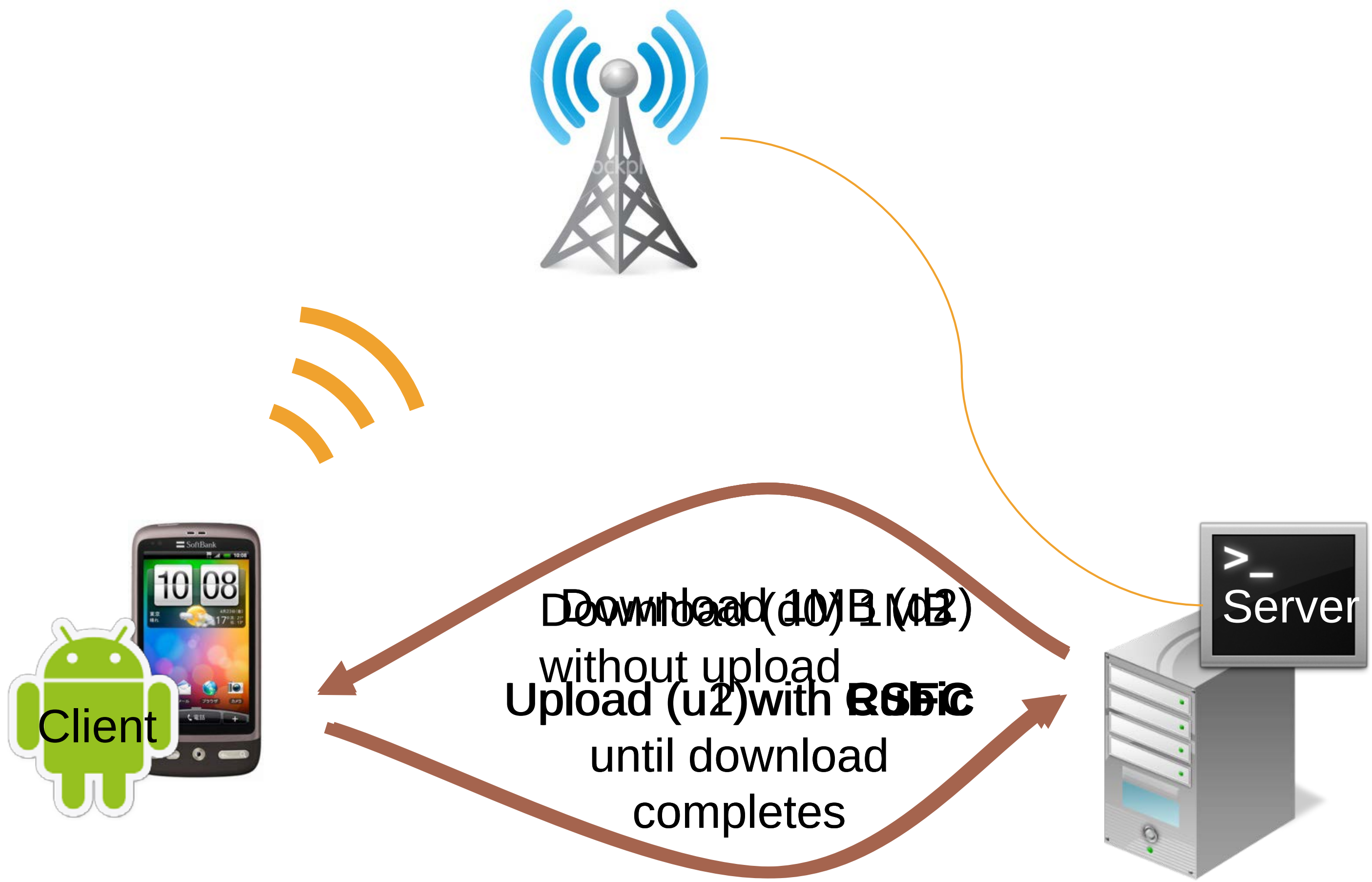
Reduction in RTT



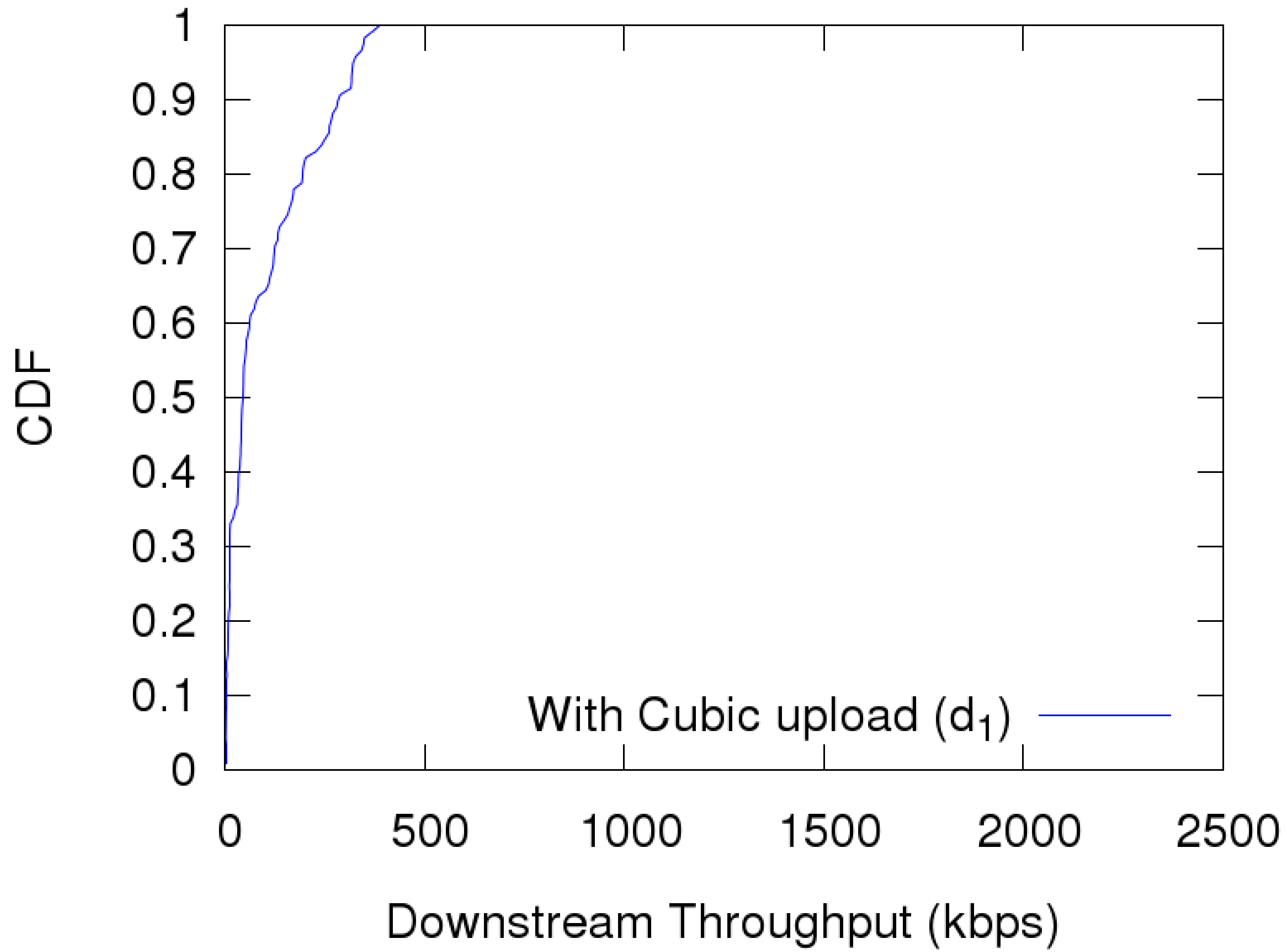
No Reduction in Throughput



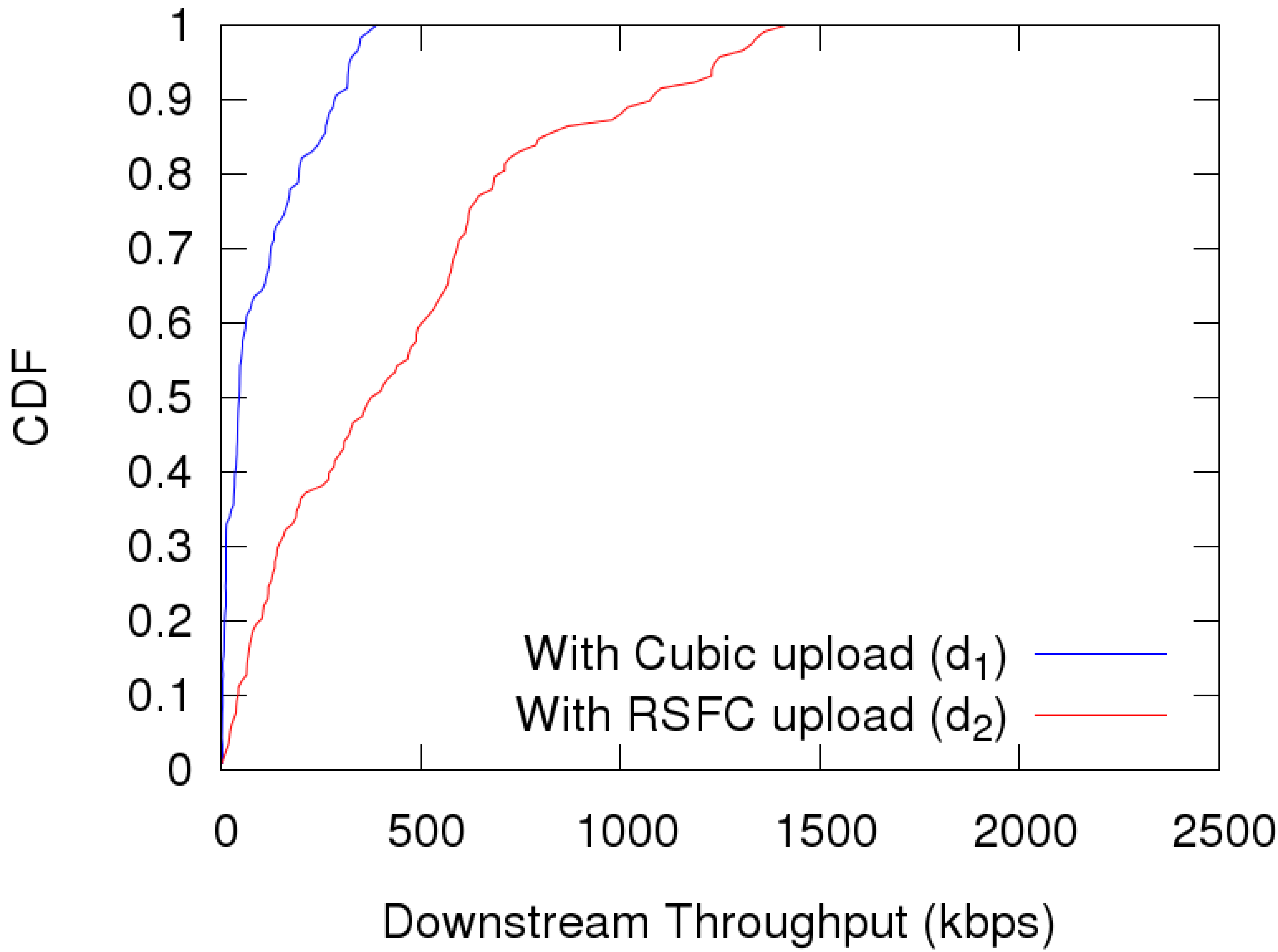
Improve Download Throughput



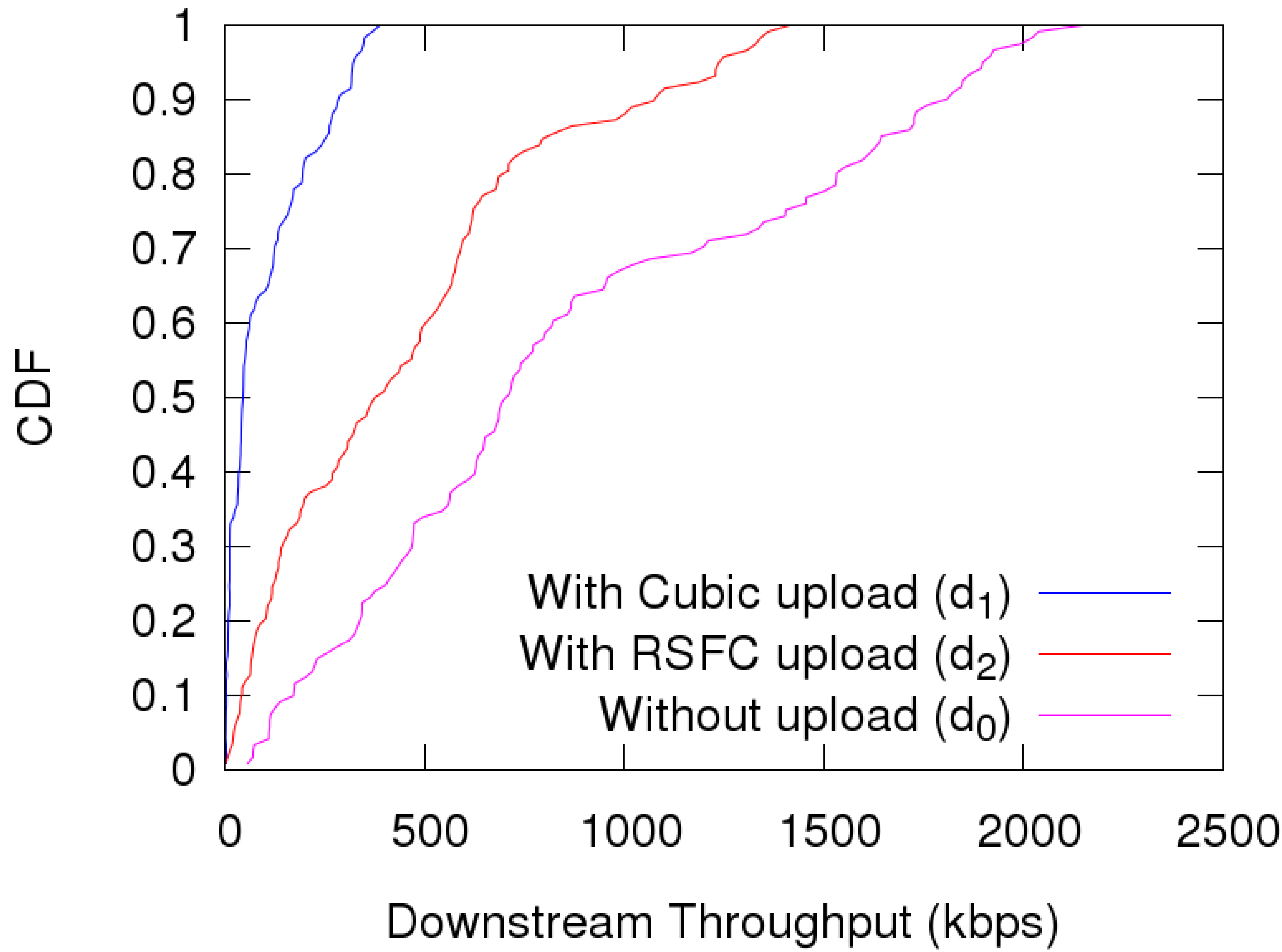
Better Downstream Performance



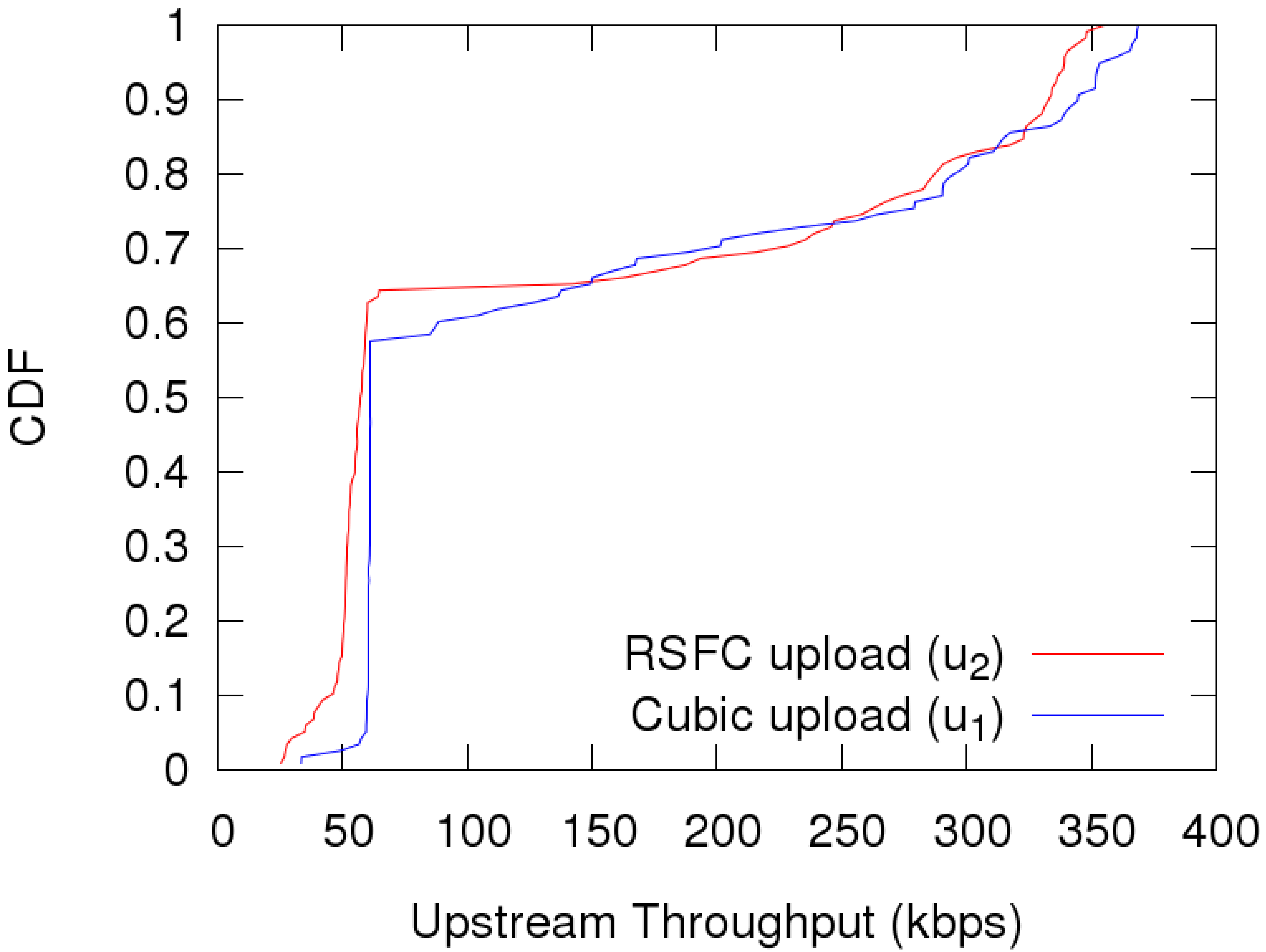
Better Downstream Performance



Better Downstream Performance

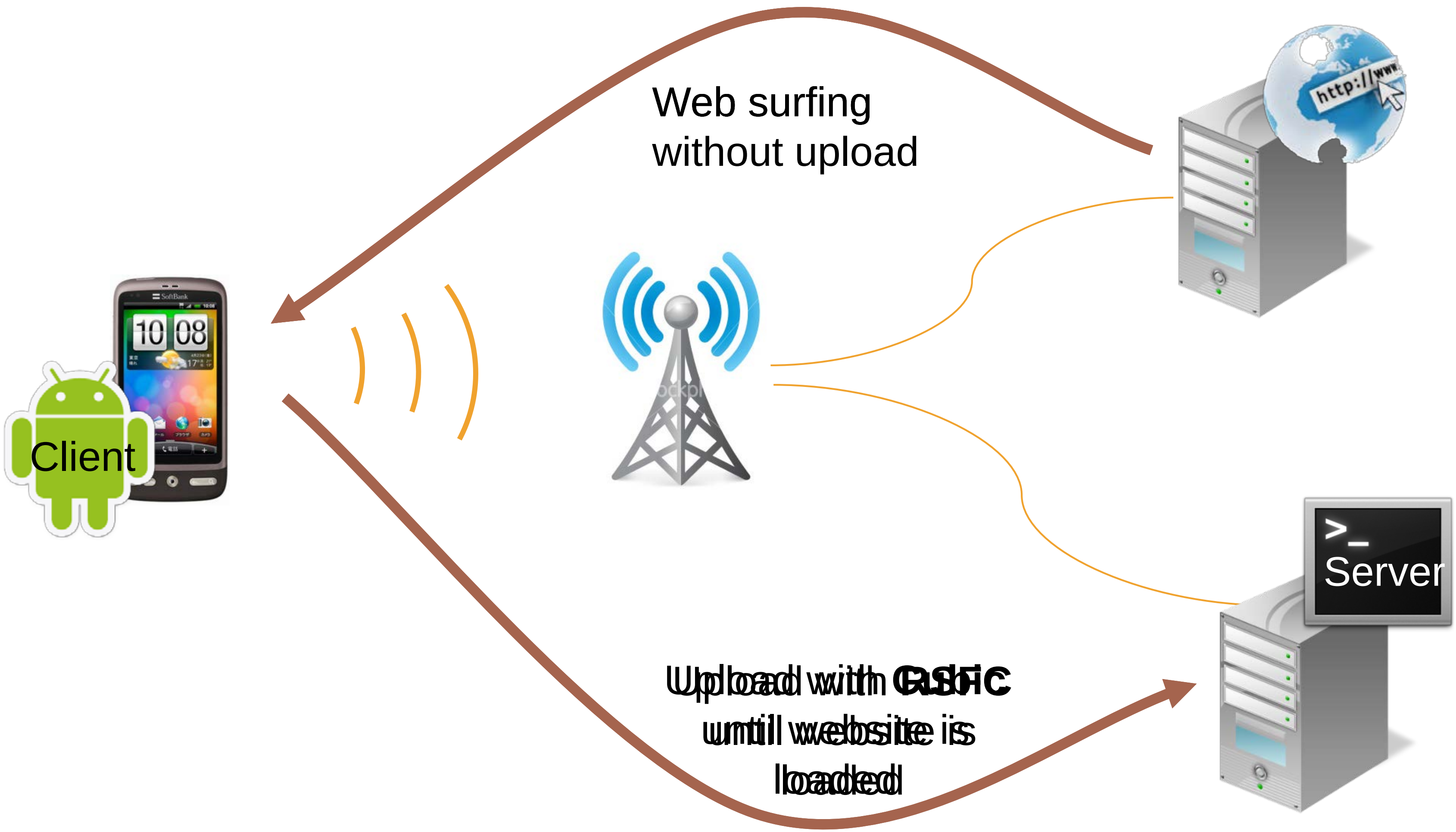


Little Impact on Upstream

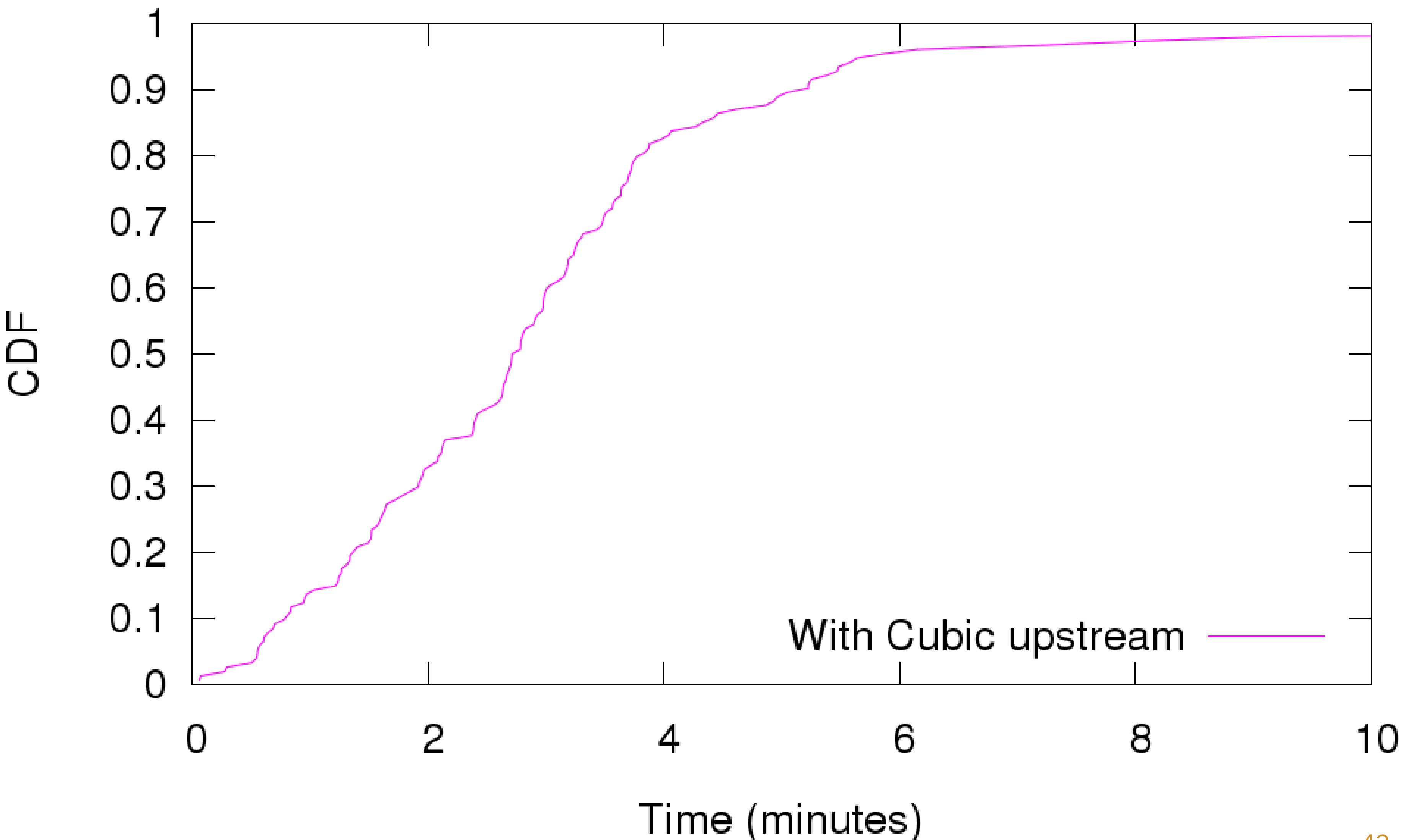


Improve Web Surfing

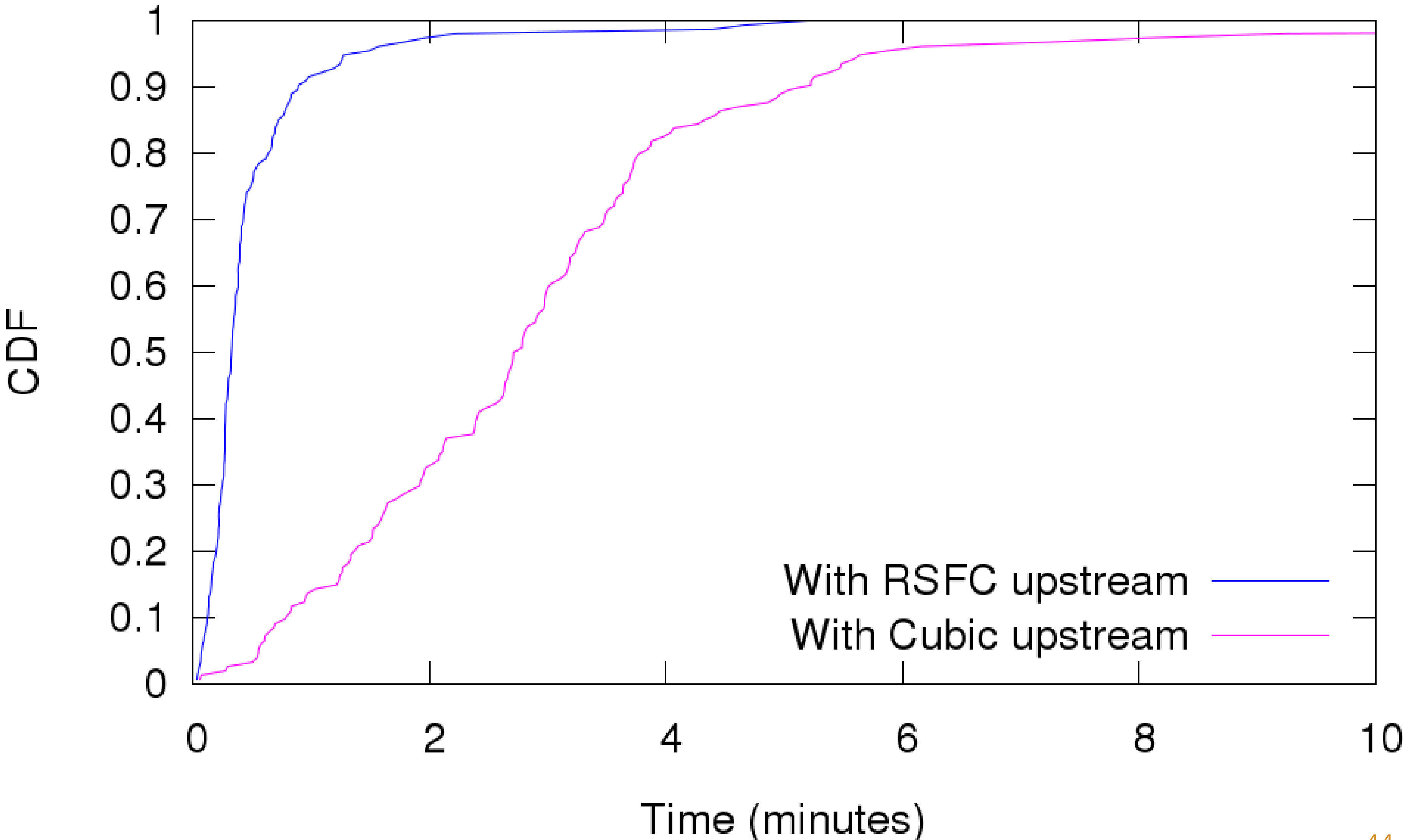
- Alexa top 100 sites



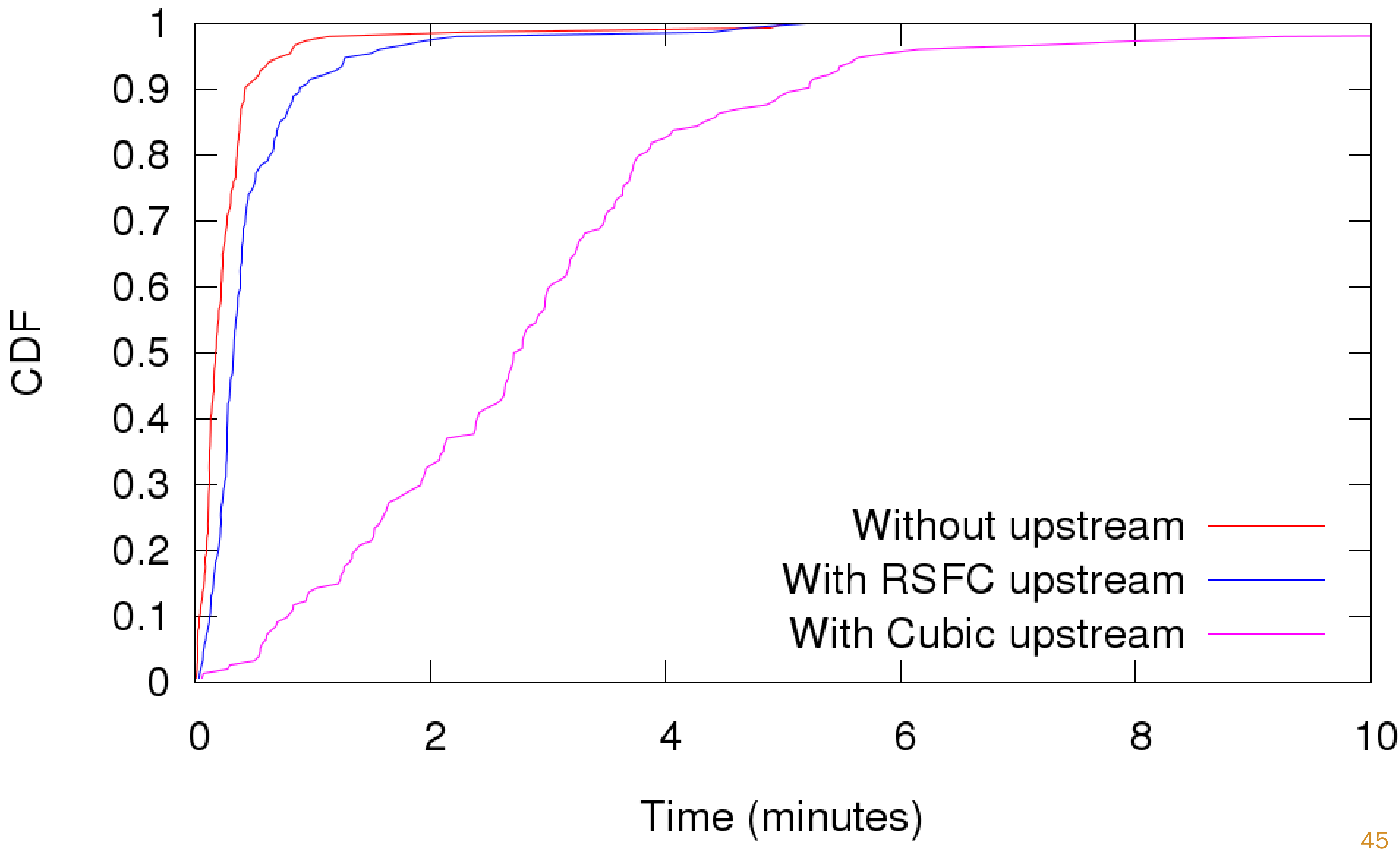
Webpages Load Faster



Webpages Load Faster



Webpages Load Faster



Conclusion

- Saturated uplink can cause serious performance degradation
- Receiver-Side Flow Control
  - ✓ Reduces queuing delay significantly
  - ✓ Improves downstream performance
  - ✓ Reduces loading time of webpages
  - ✓ Compatible with existing TCP variants
  - ✓ Easily deployed at ISP proxies

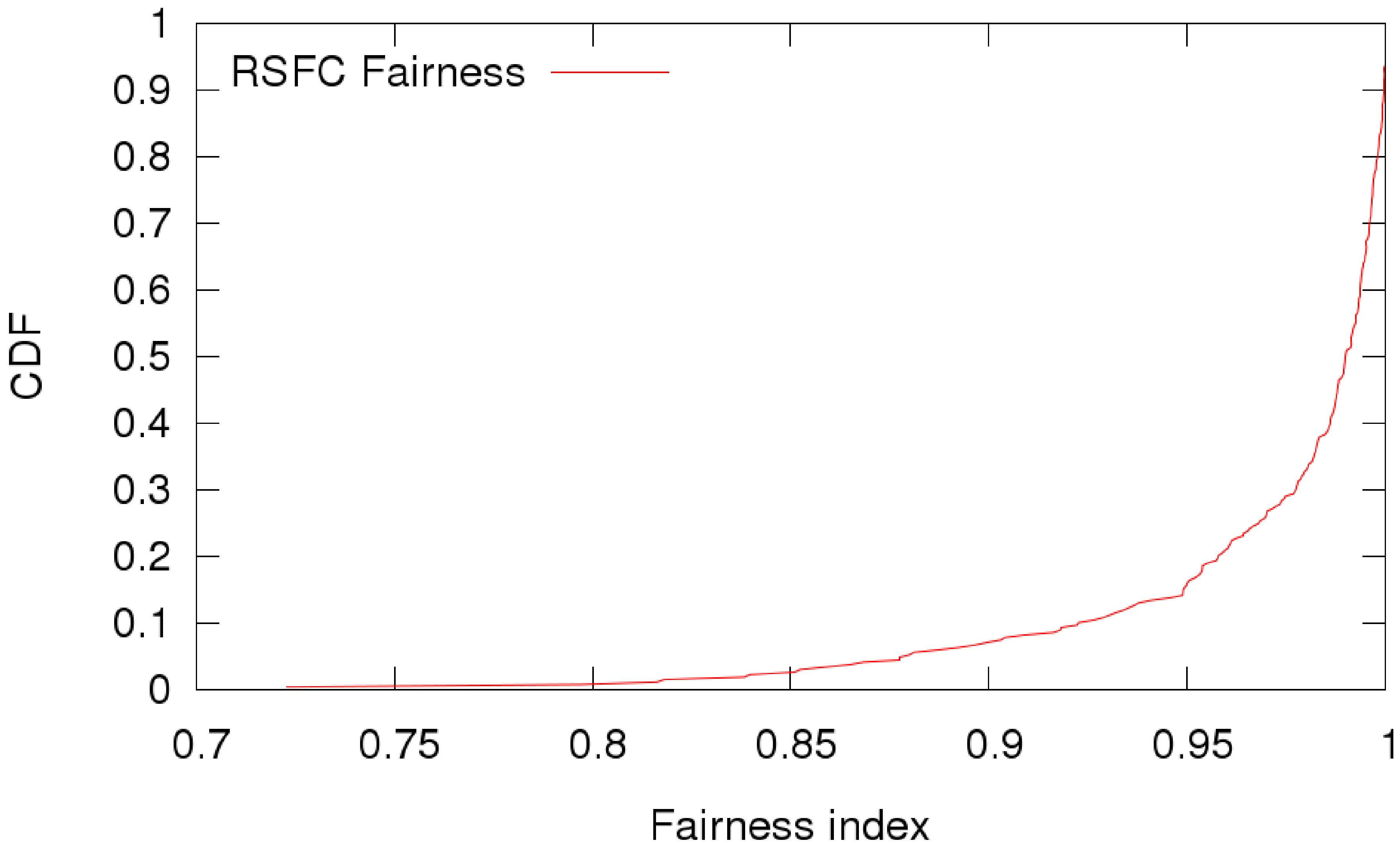
THANK YOU

Fairness of Competing RSFC Uploads

- Run two RSFC uploads concurrently
- Calculate Jain fairness index:

$$(R_1 + R_2)^2 / (2(R_1^2 + R_2^2))$$

Fairness of Competing RSFC Uploads

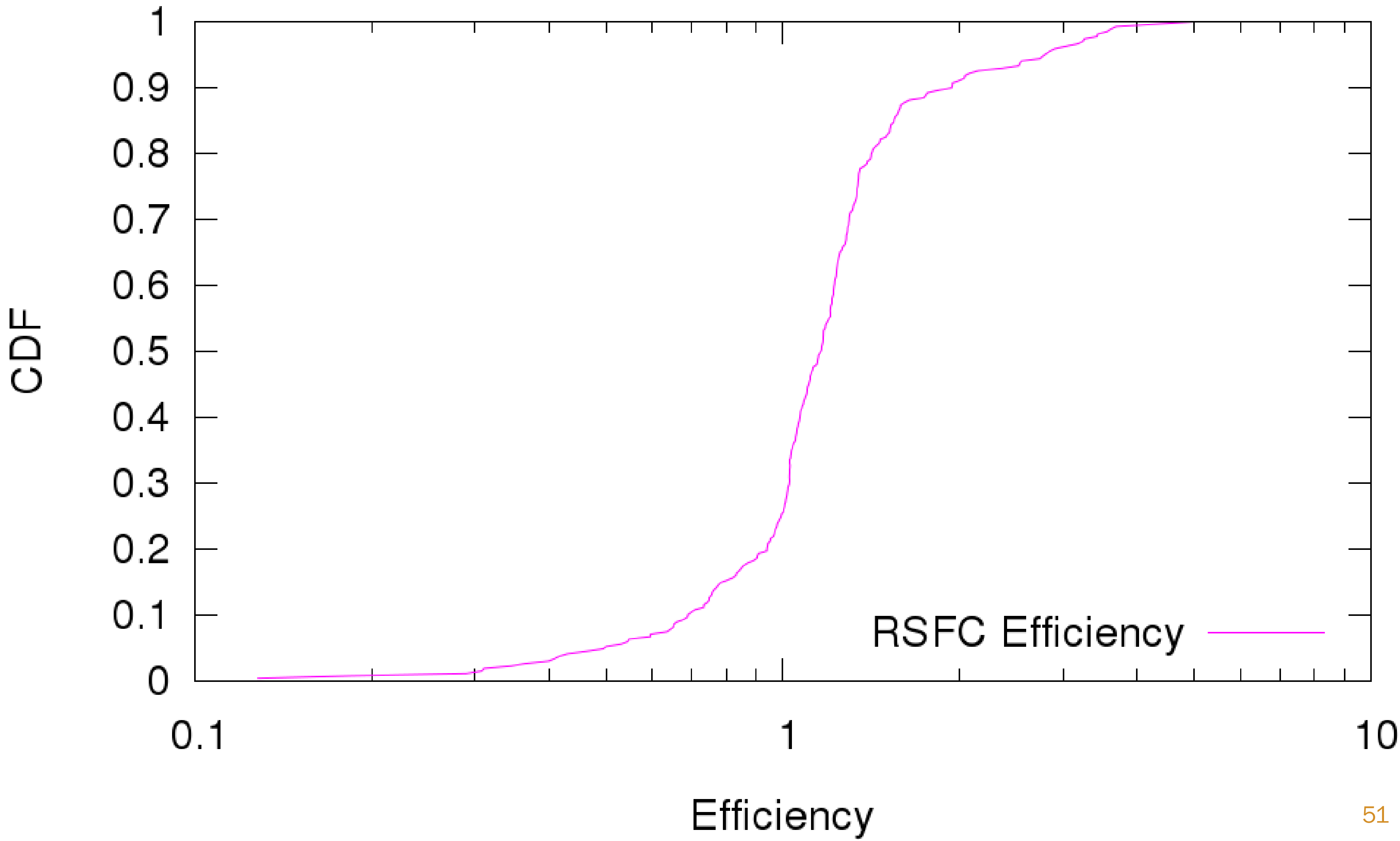


Efficiency of Competing RSFC Uploads

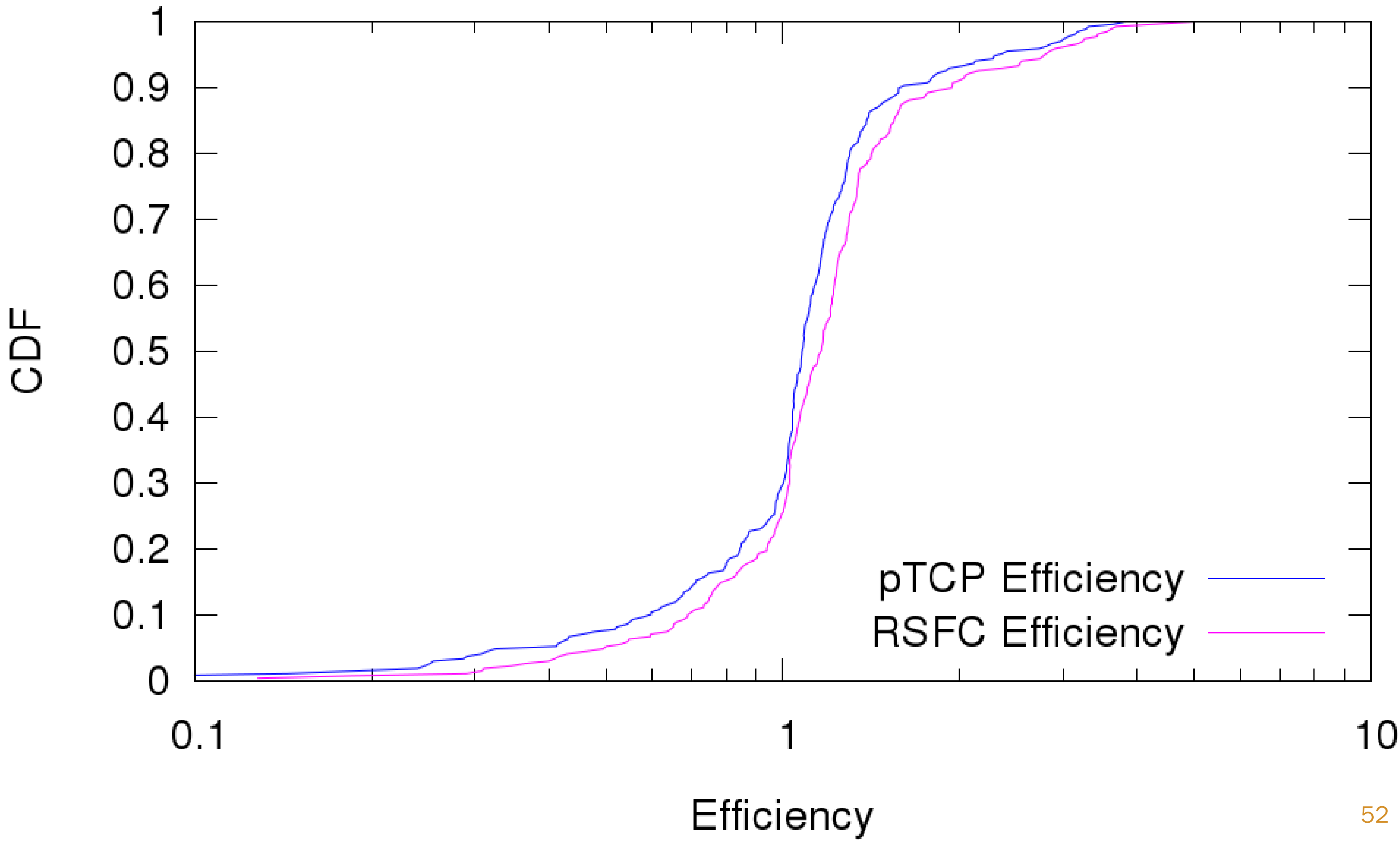
- Run two RSFC uploads concurrently
- Compare the aggregate throughput to a single TCP:

$$(R_1 + R_2) / R_{tcp}$$

Efficiency of Competing RSFC Uploads



Efficiency of Competing RSFC Uploads

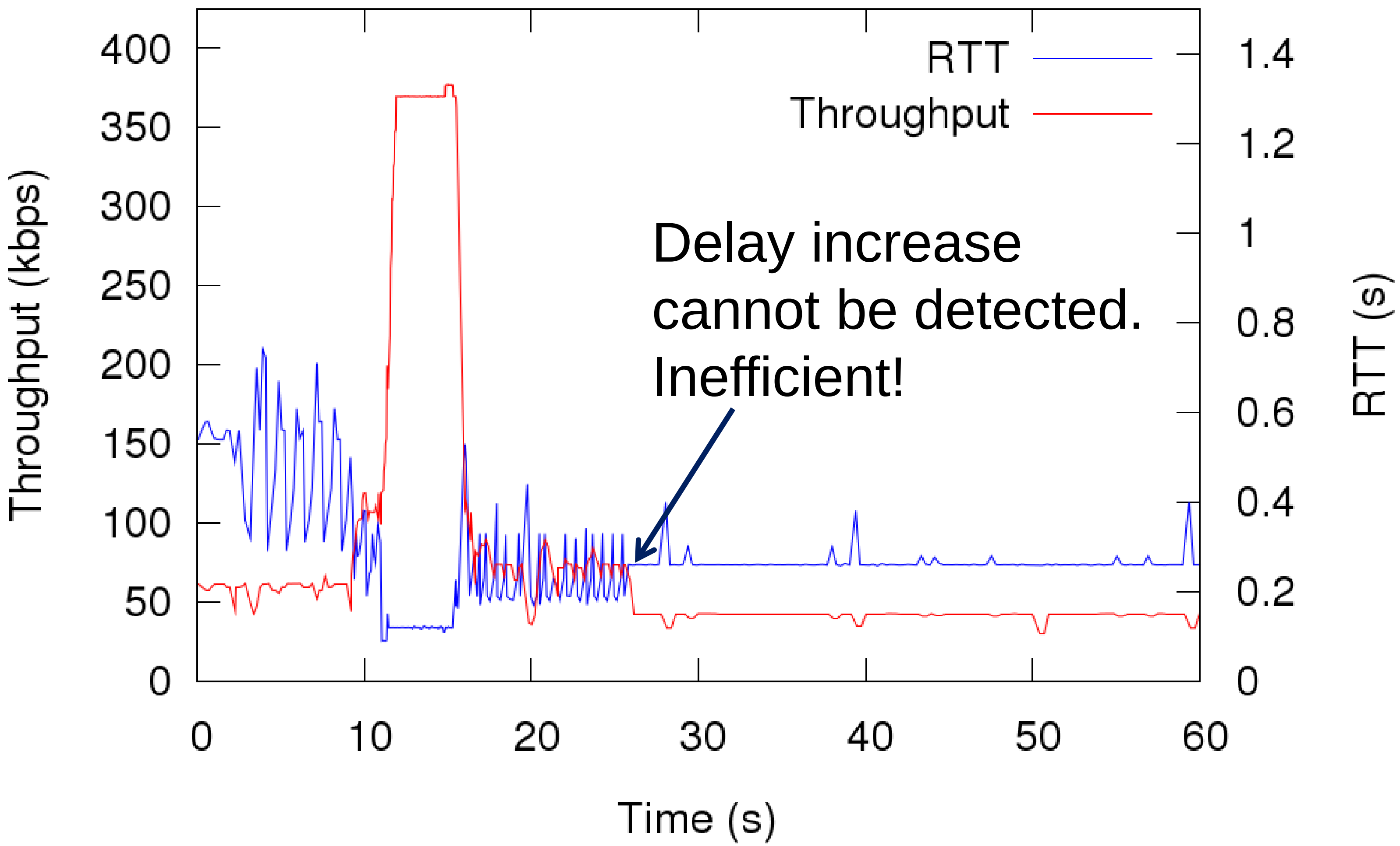


Adapting to Changing Network Conditions

- **Compare with one RSFC version without:**
  - Checking ρ
  - Monitor state

Adapting to Changing Network Conditions

Without the two methods



Adapting to Changing Network Conditions

With the two methods