Mitigating Egregious ACK Delays in Cellular Data Networks by Eliminating TCP ACK Clocking

Wai Kay Leong, Yin Xu, Ben Leong, Zixiao Wang

National University of Singapore
Asymmetry in Cellular Networks

- **Congestion in Uplink**
  - Concurrent Upload
    - RSFC [Xu et al. 2012]
  - Other Users

![Diagram showing uplink and downlink connections](image-url)
Egregious ACK Delays

- TCP congestion control is ACK-clocked
Egregious ACK Delays

- TCP congestion control is ACK-clocked
- Congested uplink can delay ACKs
  - Downlink becomes idle

[Diagram showing data transfer with acknowledgment delays]
Egregious ACK Delays

- TCP congestion control is ACK-clocked
- Congested uplink can delay ACKs
  - Downlink becomes idle

Data → Under Utilization → Idle Downlink

Data → Reduced Throughput

ACK
Solution: Eliminate ACK Clocking

**Idea:** If we know the bandwidth, we can send at maximum rate.

- Send Data at X Mb/s
- Bandwidth: X Mb/s
- Not dependent on ACK arrival

Data

ACK

No cwnd

Send Data at X Mb/s

Bandwidth: X Mb/s

Not dependent on ACK arrival
Challenge 1: Estimating Bandwidth

Idea

Bandwidth ≡ Receive Rate

• Use receiving rate as equivalent of available bandwidth

Condition

Done Passively

• To avoid modifications at the receiver

Solution

Use TCP Timestamps

• Enabled by default on Android and iPhones
Estimating Receive Rate

\[ \text{TSval} = t_{r0} \]

\[ \text{TSval} = t_{r1} \]

\[ t_{r1} - t_{r0} = \Delta t \]

\[ \frac{\Delta \text{ACK}}{\Delta t} = \rho \]

Receive Rate \( \rho \)
Challenges

1. Estimating Bandwidth

2. Timestamp Granularity too Coarse
   - Cannot estimate with high accuracy

3. Bandwidth variation
   - Have to keep updating estimation
Solution

Self-oscillating Feedback Loop

- Estimate Receive Rate $\rho$
- Send Rate $\sigma$

Send Faster ($\sigma > \rho$)
Send slower ($\sigma < \rho$)

No Congestion
Link Congested

How to detect congestion?
Detect Congestion

- **Idea:** Monitor Queuing Delay

  - **How?**
    - TCP Timestamps
    - **Relative Difference** between sender and receiver

  ![Buffer diagram](image)

  - $t_{buff} > 0$
  - $t_{buff} = 0$
Detecting Congestion

Relative Delay (RD)

Min Delay (RD_{min})

Queuing Delay (t_{buff})

Congestion detected when t_{buff} > T

RD = t_{r1} - t_{s1}

TSval = t_{r1}

RD - RD_{min} \approx t_{buff}
Summary of Algorithm

1. Initial Receive Rate Estimation
   - Send 10 packets
   - Estimate $\rho$ using replies

2. Buffer Management Mode
   a) Buffer Fill State
      - Send Faster ($\sigma > \rho$)
   b) Buffer Drain State
      - Send Slower ($\sigma < \rho$)

3. Monitor Mode
   - Probe network
   - Details in paper

TCP-RRE (Receiver-Rate Estimation)

$t\text{buff} > T$
$t\text{buff} < T$

Significant changes in network
Parameters?

• How much faster or slower to send?
• What threshold $T$ to use?
• When to switch to monitor state?

See details in paper
ns-2 Evaluation

Measured real networks to get simulation parameters
ns-2 Evaluation

1. Single Download with Slow Uplink
2. Single Download under Normal Conditions
3. Download with Concurrent Upload
4. Handling Network Fluctuation
5. TCP Friendliness
ns-2 Evaluation

1. Single Download with Slow Uplink
2. Single Download under Normal Conditions
3. Download with Concurrent Upload
4. Handling Network Fluctuation
5. TCP Friendliness
Download with Slow Uplink

Uplink Speed (kb/s) vs. Downlink Speed (Mb/s)

- Uplink Speed: 200 kb/s
- Downlink Speed: 1 Mb/s, 8 Mb/s

Data transfer and ACK mechanism depicted.
Uplink speed decreases
Download under Normal Conditions

![Graph showing average goodput vs. downlink bandwidth for TCP-RRE and TCP-CUBIC protocols.](ns-2)
Download under Normal Conditions

Packet Loss

1400+

TCP-CUBIC

0  2  4  6  8  10  12

Time (s)

0  200  400  600  800

Goodput (kB/s)

Pkt in buff

1200  1400  1600

ns-2
Download under Normal Conditions

- Goodput (kB/s)
- Pkts in buff

TCP-RRE
TCP-CUBIC

Time (s)
Download with Concurrent Upload

Download 1MB

Upload until download completion

Uplink Speed (kb/s)

Downlink Speed (Mb/s)

0.5  8

3000

250
Download with Concurrent Upload

Cumulative distribution

Throughput of TCP-RRE over Cubic

2.5X
Evaluation in Linux

- Several Places, different ISPs
- Multiple times
- CDF of all experiments at each place for each ISP
Evaluation in Linux

Cumulative distribution

TCP Goodput (Mb/s)
Evaluation in Linux

The figure shows a comparison of Cumulative distribution of TCP Goodput (Mb/s) for two upload methods:
- Cubic w upload
- RRE w upload

The chart indicates a +50% improvement in performance compared to the baseline.
Evaluation in Linux

Cumulative distribution

TCP Goodput (Mb/s)

Cubic w upload
RRE w upload
Cubic single
Evaluation in Linux

[Graph showing cumulative distribution of TCP Goodput (Mb/s) for different algorithms: Cubic w upload, RRE w upload, Cubic single, RRE single. The graph indicates an increase of +25%.]
Conclusion

Congested Uplink

Delayed ACKs

Poor Download Performance
Conclusion

• **TCP-RRE**
  – ACK Clocking
  – Rate Control with Feedback Loop

• **Use TCP Timestamp**
  – Estimate Receive Rate
  – Detect Congestion

• **Improves TCP**
  – Uplink is Slow
  – Uplink is Congested

• **Keep the Delay Low**

• **Fair to Other TCP Flows**
Thank You

QUESTIONS
Handling Network Fluctuations

CUBIC

Send rate  
Recv rate

Rate (Mb/s)

Pkt(s)

Buffer

0 5 10 15 20 25 30 35 40

100ms 3Mb/s 150ms 3Mb/s 200ms 3Mb/s 200ms 2Mb/s 100ms 2Mb/s 100ms 5Mb/s 100ms 2Mb/s
Handling Network Fluctuations

TCP-RRE

Rate (Mb/s)

Send rate
Recv rate

Buffer

PKts

TCP-RRE

0 5 10 15 20 25 30 35 40

0 100 200 300 400 500 600

100ms 3Mb/s
150ms 3Mb/s
200ms 3Mb/s
200ms 2Mb/s
100ms 2Mb/s
100ms 5Mb/s
100ms 2Mb/s
TCP Friendliness

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

\[
\frac{(R_1 + R_2)^2}{2(R_1^2 + R_2^2)}
\]
TCP Friendliness

Cumulative distribution

Jain's Fairness Index

- RRE vs Cubic
- Cubic vs Cubic
- Reno vs Reno
- Cubic vs RRE
- RRE vs RRE
- Vegas vs Vegas