Equation-Based Congestion Control for Unicast Applications

Sally Floyd, Mark Handley
AT&T Center for Internet Research (ACIRI)

Jitendra Padhye
Umass Amherst

Jorg Widmer
International Computer Science Institute (ICSI)

Proceedings of ACM SIGCOMM, 2000

Outline

• Intro
• Foundations
• TFRC
• Experimental Evaluation
• Related Work
• Conclusions

Introduction

• TCP
  – Dominant on Internet
  – Needed for stability
  – AIMD
  – Window-based
• “Bulk-data” applications fine with TCP
  – But real-time find window fluctuations annoying
• Equation-based congestion control to the rescue!
  – Smooth the rate
  – (Note, class-based isolation beyond this paper)

But don’t we need TCP?

• Practical
  – Primary threat are from unresponsive flows
    – Choose UDP over TCP
  – Give others protocol so they have something!
• Theoretical
  – Internet does not require reduction by ½
    – Other rates have been 7/8 (DECbit)
  – Even “fairness” to TCP doesn’t require this
  – Needs some control to avoid high sending rate during congestion

Guiding Basics for Equation-Based Protocol

• Determine maximum acceptable sending rate
  – Function of loss event rate
  – Round-trip time
• If competing with TCP (like Internet) should use TCP response equation during steady state
• There has been related work (see later sections) but still far away from deployable protocol
• This work presents one such protocol
  – TFRC

TFRC Goals

• Want reliable and as quick as possible?
  – Use TCP
• Slowly changing rate?
  – Use TFRC (ms. vs. s.)
• Tackle tough issues in equation-based
  – Responsiveness to persistent congestion
  – Avoiding unnecessary oscillations
  – Avoiding unnecessary noise
  – Robustness over wide-range of time scales
  – Loss-event rate is a key component!
• Multicast
  – If all receivers change rates a lot, never can scale
Foundations of Equation-Based Congestion Control

- **TCP-Friendly Flow**
  - In steady-state, uses no more bandwidth than conformant TCP running under same conditions
- **One formulation:**
  \[ T = \frac{s}{R \sqrt{\frac{T}{T}} + t_{RTO}(3\sqrt{\frac{lp}{R}})p(1 + 32p^2)} \]
  - \( s \) – packet size
  - \( R \) – Round Trip Time
  - \( p \) – loss event rate
  - \( t_{RTO} \) – TCP timeout
  - (Results from analytic model of TCP)

Outline

- Intro
- Foundations
- **TFRC**
- Experimental Evaluation
- Related Work
- Conclusions

TFRC Basics

- Maintain steady sending rate, but still respond to congestion
- Refrain from aggressively seeking out bandwidth
  - Increase rate slowly
- Do not respond as rapidly
  - Slow response to one loss event
  - Halve rate when multiple loss events
- Receiver reports to sender once per RTT
  - If it has received packet
  - If no report for awhile, sender reduces rate

Protocol Overview

- Compute \( p \) (at receiver)
- Compute \( R \) (at sender)
- \( RTO \) and \( s \) are easy (like TCP and fixed)
- Computations could be split up many ways
  - Multicast would favor ‘fat’ receivers
- TFRC has receiver only compute \( p \) and send it to sender
- Next:
  - Sender functionality
  - Receiver functionality

Sender Functionality

- Computing RTT
  - Sender time-stamps data packets
  - Smooth with exponentially weighted avg
  - Echoed back by receiver
- Computing RTO
  - From TCP: \( RTO = RTT + 4 \times RTT_{var} \)
  - But only matters when loss rate very high
  - So, use: \( RTO = 4 \times R \)
- When receive \( p \), calculate new rate \( T \)
  - Adjust application rate, as appropriate

Receiver Functionality

- Compute loss event rate, \( p \)
  - Longer means subject to less ‘noise’
  - Shorter means respond to congestion
- After “much testing”:
  - Loss event rate instead of packet loss rate
  - Multiple packets may be one event
    - Should track smoothly when steady loss rate
    - Should respond strongly when multiple loss events
- Different methods:
  - Dynamic History Window, EWMA Loss Interval, Average Loss Interval
Computing Loss Event Rate

- Dynamic History Window
  - Window of packets
  - Even at 'steady state' as packets arrive and leave window, added 'noise' could change rate
- Exponentially Weighted Moving Average
  - Count packets between loss events
  - Hard to adjust weights correctly
- Average Loss Interval
  - Weighted average of packets between loss events over last \( n \) intervals
  - The winner! (Comparison not in paper here)

Loss Interval Computation

\[
\lambda (t; A) = \sum_{I=1}^{n} \frac{\beta (s_i - s_{i-1})}{\sum_{i=1}^{n} \beta (s_i - s_{i-1})}
\]

- \( \beta = 1 \) for \( 1 \leq i \leq n/2 \)
- \( \beta = 1 - (i - n/2) / (n/2 + 1) \)
- \( 1, 1, 1, 0.8, 0.6, 0.4, 0.2 \)
- Rate depends upon \( n \)
  - \( n = 8 \) works well during increase in congestion (Later section validates)
  - Have not investigated relative weights
- History discounting for sudden decreases in congestion
  - Interval \( s_i \) is a lot larger
  - Can speed up
- Loss event rate, \( \rho \), is inverse of loss interval

Instability from RTT Variance

- Inter-packet time varies with RTT
  - Fluctuations when RTT changes

Improving Stability

- Take square root of current RTT (\( M \) is sqrt of average)

\[
\lambda_{\text{outer}} = \frac{e^{\sqrt{R_0}}}{T^M}
\]
**Slowstart**
- TCP slowstart can no more than double congestion bottleneck
  - 2 packets for each ack
- Rate-based could more than double
  - Actual RTTs getting larger as congestion but measured RTTs too slow
- Have receiver send arrival rate
  - $T_{i+1} = \min(2T_i, 2T_{recv})$
  - Will limit it to double cong bwidth
- Loss occurs, terminate “slowstart”
  - Loss intervals? Set to ¼ of rate for all
  - Fill in normally as progress

**Loss Fraction vs. Loss Event Fraction**
- Obvious is packets lost/packets received
  - But different TCP’s respond to multiple losses in one window differently
    - Tahoe, Reno, Sack all halve window
    - New Reno reduces it twice
- Use loss event fraction to ignore multiple drops within one RTT
- Previous work shows two rates are within 10% for steady state queues
  - But DropTail queues are bursty

**Increasing the Transmission Rate**
- What if $T_{new}$ is a lot bigger than $T_{old}$?
  - May want to dampen the increase amount
- Typically, only increase 0.14 packets / RTT
  - History discounting provides 0.22 packets / RTT
- Theoretical limit on increase
  - $A$ is number of packets in interval, $w$ is weight
    $\delta_T = 1.2 \left( \sqrt{A + w1.2\sqrt{A}} - \sqrt{A} \right)$
  - So … no need to dampen more

**Response to Persistent Congestion**
- To be smooth, TFRC does not respond as fast as does TCP to congestion
  - TFRC requires 4-8 RTTs to reduce by ½
- Balanced by milder increase in sending rate
  - 0.14 packets per RTT rather than 1
- Does respond, so will avoid congestion collapse
- (Me, but about response to bursty traffic?)

**Response to Quiescent Senders**
- Assume sender sending at maximum rate
  - Like TCP
- But if sender stops, and later has data to send
  - the previous estimated rate, $T$, may be too high
- Solution:
  - if sender stops, receiver stops feedback
- Sender ½ rate every 2 RTTs
- (Me, what about just a reduced rate that is significantly less than $T$?
  - May happen for coarse level MM apps)
Outline

- Intro
- Foundations
- TFRC
- **Experimental Evaluation**
  - Simulation
  - Implementation
    - Internet
    - Dummynet
- Related Work
- Conclusions

Simulation Results (NS)

- TFRC co-exist with many kinds of TCP traffic
  - SACK, Reno, NewReno…
  - Lots of flows
- TFRC works well in isolation
  - Or few flows
- Many network conditions

TFRC vs. TCP, DropTail

- Mean TCP throughput (want 1.0)
- Fair (?)

TFRC vs. TCP, RED

- Even more fair
- Not fair for small windows
- (Me … bursty traffic with many flows?)

Fair Overall, but what about Variance?

- Variance increases with loss rate, flows

CoV of Flows (Std Dev / Mean)

- A fairness measure
- Average of 10 runs
- TFRC less fair for high loss rates (above typical)
- Same w/Tahoe and Reno, SACK does better
  - timer granularity is better with SACK
Individual Throughputs over Time

• 0.15 second interval (about multimedia sensitivity)
• Smoother rate from TFRC

Equivalence at Different Timescale

• Compare two flows
• Number between 0 and 1 (equation (4))
• Cases
  – Long duration flows in background
  – On-Off flows in background

Equivalence for Long Duration

• Single bottleneck
• 32 flows
• 15 Mbps link
• Monitor 1 flow
• 95% confidence interval
• Results hold over broad range of timescales

Outline

• Intro
• Foundations
• TFRC
• Experimental Evaluation
  – Simulation
    – Fairness and Smoothness (CoV) (done)
    – Long Duration (done)
  – On-Off flows
  – Implementation
• Related Work
• Conclusions

Performance with On-Off Flows

• 50 – 150 On/Off UDP flows
  – On 1 second, off 2 seconds (mean)
  – Send at 500 kbps rate
• Monitor TCP, Monitor TFRC

Equivalence with TCP with Background Traffic

• At high loss rates, less equivalent (40% more, less)
• (Me, room for improvement)
CoV with Background Traffic

• TFRC rate has less variance, especially at high loss rates

Effect on Queue Dynamics

• 40 flows, staggered start times
• TCP (top) has 4.9% loss and TFRC (bottom) has 3.5% loss
• 99% utilization for all
• Basically, look the same
• Extensive tests, w/RED and background look the same

Outline

• Intro
• Foundations
• TFRC
• Experimental Evaluation
  – Simulation (done)
  – Implementation
    – Internet
• Related Work
• Conclusions

Implementation Results

• TFRC on Internet
  – Microwave
  – T1
  – OC3
  – Cable modem
  – Dialup modem
• Generally fair
• (See tech report for details)

London to Berkeley

• 3 TCP flows, 1 TFRC flow
• TFRC slightly lower bandwidth but smoother
• Typical loss rates .1% to 5%

TCP Equivalence over Internet

• UCD
• Michigan
• UMASS (Lincoln)
• UMASS (Amherst)
• NCSA, Beeston
• Measurement Timescale (seconds)
• Equivalence Ratio
CoV over Internet

TFRC unfair to TCP when ...
- When flows have one packet per RTT
  - TFRC can get far more than its fair share
  - Due to ‘conservative’ clock (500ms) in FreeBSD?
- Some TCP variants are ‘buggy’
  - Linux vs. Solaris
  - (Me, a neat project)
- Real-world “Phase Effect” (?)

Testing the Loss Predictor
- How effective do $X$ intervals predict immediate future loss rate?
- But not just great prediction but reaction, too

Related Work
- TCP Emulation At Receiver (TEAR)
  - Compute window at receiver, convert to rate
- Rate Adaptation Protocol (RAP)
  - AIMD approach
  - No slow start, no timeout
- Other equation based
  - One ties with MPEG (application)
  - One TFRCP direct comparison

Issues for Multicast Congestion Control
- Still feedback every RTT
  - Must change to aggregate or hierarchical
  - Or lowest transmission rate
- Slowstart especially problematic as needs very timely feedback
- Synchronized clocks needed so receivers can determine RTT in scalable manner

Conclusions
- TFRC gives TCP-fair allocation of bandwidth over wide range of environments
- TFRC smoother than TCP
- Evaluated over wide range of network conditions
Future Work

- What is some retransmission?
  - How to divide up $T$
- What if some extra repair information?
  - How to divide up $T$?
- Duplex TFRC?
- ECN and TFRC?