Relentless Congestion Control

Matt Mathis
Pittsburgh Supercomputing Center (PSC)

PFLDNet

21 May 2009

http://staff.psc.edu/mathis/relentless

http://staff.psc.edu/mathis/papers/PFLDNeT2009.pdf

PSC
PITTSBURGH SUPERCOMPUTING CENTER

Outline

- Goals
- Digression on scaling
- Relentless TCP
- Properties
- Implementation and Deployment

Note: this is only preliminary work

Goals

- Explore a different CC design point
  - Shift responsibility for capacity allocation and fairness from TCP to the network
    - Want to undo the TCP-friendly paradigm
  - TCP's role is to keep the network busy by maintaining queues of data for the it to deliver
    - Without being abusive or wasteful
    - Near “maximally” aggressive
- Compatibility with AIMD is not a goal

Where does standard TCP fail?

- Scale limits due to AIMD congestion control
  - Pointed out by Steve Low and others
- Consider the period of the AIMD sawtooth

  (for 100 ms RTT, 1500 Byte MTU, DBP queue)
  - At 10 Mb/s: 13s between losses
  - At 100 Mb/s: 130s
  - At 1 Gb/s: 1300s
- TCP can't possibly get sufficient feedback

Consider delay sensing CC

- Sample the network state on each RTT
  - Control is very accurate when it works
- But suffers due to short network queues
  - Not enough dynamic range in the signal
    - Reverts to loss behavior if the queues overflow
    - For lossless CC, maximum cross traffic and jitter from all sources has to be limited to Qtime/RTT
  - Control is fragile if there is too much noise
- Difficult to address some pathological cases

  E.g. Filling 10 Gb/s with one 5 Gb/s flow plus other mixed traffic

How to make loss driven CC scale?

- Need an upper bound on control interval
- Control interval must not depend on data rate
  - (Otherwise Moore's law will eventually dominate)
  - e.g. No more than k round trip times between losses
  - Which is 1 loss per k windows of data:

$$Rate_{Relentless} = \left(\frac{MSS}{RTT}\right)\left(\frac{1}{k}\right)\left(\frac{1}{p}\right)$$

Comparison with AIMD

- Standard model

$$Rate_{AIMD} = \left(\frac{MSS}{RTT}\right)\left(\frac{c}{\sqrt{p}}\right)$$

- Performance ratio

$$\frac{Rate_{Relentless}}{Rate_{AIMD}} = \left(\frac{1}{k\,c^2}\right)\left(\frac{c}{\sqrt{p}}\right)$$

- Observations:
  - This is just a constant times the average AIMD window size, in packets
  - It might be very large at large windows
  - Our only assumption was that there is a bound on the control interval that does not depend on data rate
  - This is not likely to be considered fair

Relentless TCP

- Pure implementation of VJ packet conservation
  - Packets are sent in response to packets arriving
  - (Ideally) the only window reduction would be loss
  - Additive Increase only when:
    - Lossless RTT and
    - Flight size == cwnd

- Cool new property
  - TCP portion of the primal-dual system has unity gain
    - e.g. If the queue is 20 packets too large, just drop 20 packets
    - If flow is using 1% too much capacity, drop 1% (for 1 RTT)
  - Claim: vastly simpler to approximate optimal AQM
    - Even for small flow populations
    - Do not have to estimate TCP's response to a single loss

One minor problem

- It controls hard against full drop tail queues
- Must have AQM to limit occupancy

An example “baseline” AQM

- Segregate flows
  - Assume some explicit scheduling policy
- In each (periodic) interval:
  - Monitor the minimum queue length
  - If it is above the set point,

    drop excess packets during the next interval
- The ideal periodic interval is 1 RTT
  - But it is not sensitive to huge miss-match
  - e.g. at 10RTT the average error is only 5 packets

Usage example: remote video upload

- Satellite link with TCP video upload + interactive
  - Assume 10 Mb/s, 600 ms RTT, 1500 B MTU
  - Pipe size is ~500 packets
  - Assume DSCP queuing or RR scheduling
- Relentless TCP + Baseline queue control (1s)
  - Expect one drop per 2 S (1 drop per 3 RTT)
    - About 1 per 1500 packets

Same example using standard TCP

- Optimal solution for 1 flow:
  - Need a loss when queue reaches 500 packets
  - Which is every 1000 RTT (due to delayed ACK)
  - Or once every ~700,000 packets (>10 minutes!)
  - 500 times lower loss rate than Relentless
  - More likely BER, etc prevents filling the link
  - THESE NUMBERS ARE ABSURD
- Work around by using multiple flows
  - Changes the peak queue size
  - Optimal AQM must estimate:
    - Flow population and/or
    - Effective RTT

  (Note that Relentless/Baseline does not need this)

Claimed properties of Relentless CC

- Model: rate is proportional to 1/p
  - One loss every 3 RTTs
  - Vastly higher equilibrium loss rates than standard TCP
    - By roughly the window size in packets
      (e.g. 500 in the previous example)
- Not at all AIMD-friendly
  - As expected

  (Workaround: use Lower Effort (LE) service [RFC3662])
- Can replace MD in any AIMD style CC
  - Especially good with delay sensing algorithms
  - I chose to start with unmodified Reno to simplify the dialog, not because I think it is optimal

Can not cause congestion collapse

- Implements the conservative principle
  - After loss cwnd reflects data actually delivered
- Has occasional lossless RTT
  - So the window fits into the network
- Incrementally opens the window
- No change to timeout, etc

Limitations of our Implementation

- Hammers on SACK and the recovery code
  - Was easy to notice things that don't look quite right
  - Newer kernels are significantly better
- Hammers on the network
  - Interesting non-repeatable oddities in traces
- Flight size vs cwnd
  - Current philosophy is to limit bursts:
    - Pulls cwnd down to flight size during recovery and other places
    - Running out of rwin or sender CPU causes cwnd reductions
    - Not ideal at large scale (1 Gb/s * 100 ms)
  - Workaround: set ssthresh to old_cwnd-loss
    - Tahoe style slowstart after loss with window reduction

Philosophical issues

- Limiting bursts to protect other flows comes from the TCP-Friendly mindset
- But we want to network to protect other flows
- It would be philosophically consistent to send line rate bursts and let the network deliver as much as it can
- But this requires significant changes in the shared (non dlkm) part of the stack
  - Work-around: set ssthresh to old cwnd-loss

Deployment Issues

- Not TCP-Friendly
- Use Less Effort marking for traffic isolation
  - Not deployed yet....
  - Side goal is to cause the deployment of LE service
- Several observations from ICCRG (yesterday) suggest the it may be “safe” even without LE

Further information

- See http://staff.psc.edu/mathis/relentless/
- Future results, etc
- Link to Linux implementation
  - Easily installed dlkm
  - All GPL is isolated to the download page

Backup Slides

Rethinking TCP-Friendly

- How can we move beyond “one-size-fits-all” CC?
- The TCP-friendly paradigm works pretty well
  - Although there are some well published problems
  - And it forbids Relentless TCP and other advances
- What might the Internet be like without it?
  - Can traffic management work at Internet scales?
  - How can we make the transition?
- The ID is intended to be a vision statement
  - Considering the good, bad and ugly
  - See draft-mathis-iccrg-unfriendly-00.txt

Goal: An alternate universe

- Routers control traffic (“allocate capacity”)
  - Segregate traffic
    - Send more losses to greedy flows
    - Shelter non-greedy flows
  - Think:
    - Fair Queuing (well not really...)
    - Approximate Fair Dropping (AFD)
    - RE-ECN
- TCP's goal is to keep the network busy
  - It is ok to be greedy (up to a point)
- Cool new property: Neither router behavior nor end-system behavior has to be standardized
  - ISPs can enforce their own “fairness” model
  - Allows TCPs to overcome adverse environments

Possible deployment scenario

1. Release Relentless TCP, LE marked
2. Other non-2581 protocols start using LE
3. LE definition is extended to include non-2581
4. ISPs implement or block LE service
   - If blocked, users complain
5. All ISPs eventually implement LE (and others?)
6. ISPs deploy traffic isolation for both services
   - Because they need better traffic controls
7. Requirement that non-AIMD use LE is relaxed

The existing Internet “fairness” paradigm

1) Routers send independent signals to all flows
2) All flows have similar response to signals
3) This response is defined by AIMD [RFC2581]

- Modeled by $$Rate = \frac{MSS}{RTT}\frac{0.7}{\sqrt{p}}$$ [Mathis97]
- Defining TCP-friendly Rate Control (TFRC), etc

But there are “fairness” problems

- Non-responsive (UDP?) flows
- Applications that open many connections
- Flows with extremely different RTTs
  - TCP matches window size (short term window fair)
- Insufficient Active Queue Management (AQM)
  - RFC 2309
- Short term fair is not at all long term fair
- Defense from DOS attacks
- Many many more
  - See the ID
  - Please contribute if you are aware of more