Insights into Laminar TCP

draft-mathis-tcpm-laminar-tcp-01
https://developers.google.com/speed/protocols/tcp-laminar

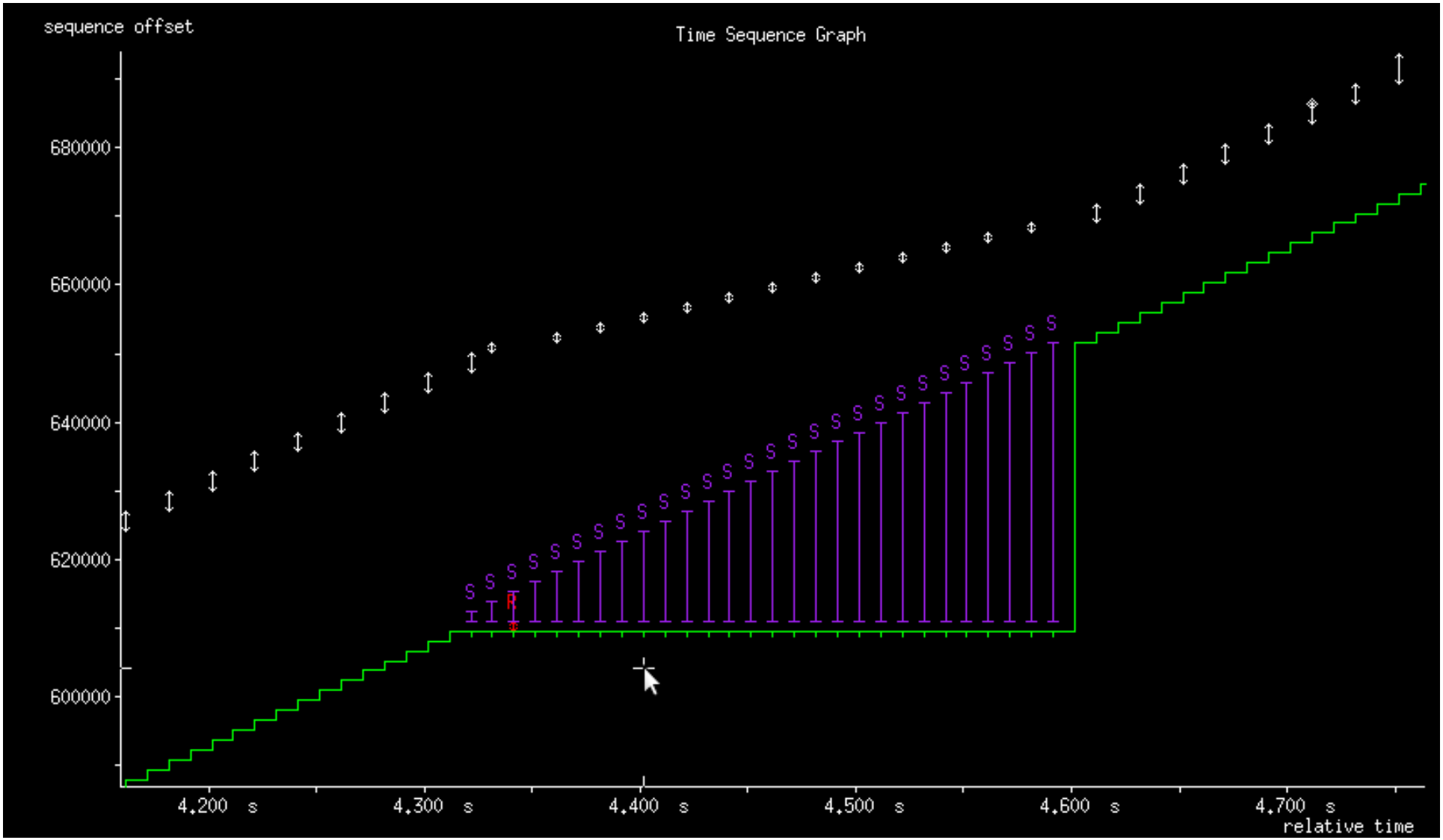
Matt Mathis
mattmathis@google.com

TCPM, IETF-84
July 30, 2012

Running code

- Patch against Linux 3.5
  - https://developers.google.com/speed/protocols/tcp-laminar
- Follows the ID fairly closely
  - Replaces nearly all congestion control code
  - Optimized for clarity of the new code
    - not minimal foot print
    - not minimal delta
  - Does not support new vs old comparison testing
  - Too intrusive to "go upstream" easily
- Suggestions would be very helpful
  - (But don't use tcpm for Linux specific discussions)

Laminar TCP - single loss



cwnd and ssthresh are overloaded

- cwnd carries both long term and short term state
  - Long term state sometimes gets saved in ssthresh
- ssthresh carries queue size estimate and (temp) cwnd
- Poorly defined interactions between:
  - Application stalls and congestion control
  - Application stalls and loss recovery
  - Reordering and congestion avoidance
  - Other unanticipated concurrent events
  - ...
- **Solution is to refactor & respecify Congestion Control**

Laminar: Two separate subsystems

- Pure congestion control
  - New state variable: CCwin
  - The quantity of data to be sent during each RTT
  - Carries state between successive RTTs
  - Not concerned with timing details, bursts etc
  - Can adapt decades of existing knowledge base
- Transmission scheduling
  - Controls exactly when to transmit
  - Packet conservation self clock (mostly)
    - Tries to follow CCwin
    - Does slow start, PRR, burst suppression
    - Future: Cwnd validation, pacing
  - Previously entwined in other algorithms
    - Not well understood as an independent subsystem

Transmission Scheduling

- Transmission scheduling
  - Packet conservation self clock (mostly)
  - Primary state is implicit, recomputed on every ACK
    - Currently no explicit long term state
  - Variables: pipe (3517), total_pipe and DeliveredData
- DeliveredData = delta(snd.una)+delta(sacked)
  - As defined in PRR
  - Computed on every ACK
  - Robust: SUM(DeliveredData) == net forward progress
- Default quantity of data to send is DeliveredData
  - This implements TCP's self clock

Total_pipe vs Pipe

- Total_pipe is pipe plus
  - DeliveredData for the current ACK, plus
  - Any pending TSO data (snd_bank)
- Thought experiment:
  - Constant circulating data (always send DeliverdData)
  - Assume pipe = 10
    - Which equals (snd.nxt - snd.una) outside of TCP
  - Which is always constant
  - During sender ACK processing
    - If the receiver delayed ACK, pipe = 8
    - If the receiver quick ACK'd, pipe = 9
    - But total_pipe is always 10

Total_Pipe

- Invariant across most protocol events
- Not changed by ACK processing
  - Except when segments are tagged "lost" by SACK
  - NB: technically an estimator if there are SACK holes
    - Due to lost/ooo ambiguity
- Changed by transmission scheduling
  - +1 per ACK to do slow start
  - -1 on some ACKs to do PRR
- Not altered by TSO or small application stalls
- Drops during application stalls
  - (Although this is really CWV and not Laminar)

New vs Old

- In Laminar, default is to send DeliveredData per ACK (or add it to snd_bank to permit TSO)
  - Adjusted +/- a little depending on (CCwin-total_pipe)
  - Note that:
    - CCwin is the long term estimate of the "fair" window
    - total_pipe the short term estimate of the actual winow
- In traditional TCP
  - Transmission is controlled directly by (cwnd-pipe)
    - TSO can choose to wait for pipe to drop further
  - Everything tweaks cwnd
    - cwnd mixes long term and short term estimates
    - In many states both pipe and cwnd are short term, while ssthresh is long term

Standards Impact: ~60 RFCs

- Most RFC references cwnd&ssthersh have minimal impact
  - Can be fixed generically, are experimental, or not TCP
- MIBs, etc, that instrument both cwnd and ssthresh
  - Require slightly more thought
- A handful describe algorithms using both cwnd and ssthresh
  - RFC 5681 - TCP Congestion Control
  - RFC 5682 - F-RTO
  - RFC 4015 - Eifel Response (aka undo)
  - RFC 6582 - NewReno
  - RFC 2861 - Cwnd Validation (and newcwv)
  - RFC 3571bis - SACK based loss recovery
  - PRRid - Proportional Rate Reduction (ID)
- These are already mentioned in the Laminar draft

Next steps - Running code

- Further refine the patch
  - Cosmetic changes to minimize the core Laminar patch
    - Extract some technically out-of-scope enhancements
  - One or more "Laminar Additions" patchs
    - Not strictly Laminar per se, but enabled by it
    - Enhancements extracted above from the core
    - Exact manifest TBD (See ICCRG later)
  - Possibly partition the core Laminar patch
      - To facilitate incremental testing
    - pipe vs total_pipe
    - Output (cwnd-pipe) vs snd_bank
    - ACK processing to compute sndcnt

Next steps - Further evolution?

- There were major opportunities for "feature creep"
- Current strategy is to narrow Laminar as much as possible
- Separate out all other enhancements
  - But we want to follow up on them
  - Feels like a large "ball of twine"
  - Each enhancement/refactor probably leads to another
  - .....

Planned new mailing list

- laminar@ietf.org

This list is for discussing Laminar TCP and how to proceed with it, through new or existing working groups in the IETF and/or IRTF. It is also intended for technical discussion of Laminar and refactoring of TCP algorithms in general.

Questions?

draft-mathis-tcpm-laminar-tcp-01

https://developers.google.com/speed/protocols/tcp-laminar

Backup Slides

(Mostly from prior IETF presentations)

Variables

- CCwin: (Target) Congestion Control window

- pipe: From 3517, data which has been sent but not ACKed or SACKed
- DeliveredData: Quantity of newly delivered data reported by this ACK (see PRR)
- total_pipe = pipe+DeliveredData+SndBank; This is all circulating data
- SndCnt: permission to send computed from the current ACK

Note that the above 4 are recomputed on every ACK

- SndBank: accumulated SndCnt to permit TSO etc

Default (Reno) Congestion Control

On startup:

```
CCwin = MAX_WIN
```

On ACK if not application limited:

```
CCwin +=  MSS*MSS/CCwin                    // in Bytes
```

On congestion:

```
if CCwin == MAX_WIN
    CCwin = total_pipe/2   // Fraction depends on delayed ACK and ABC
CCwin = CCwin/2
```

Except on first loss, CCwin does not depend on pipe!

Default transmission scheduling

```
sndcnt = DeliveredData            // Default is constant window
if total_pipe > CCwin:
    // Proportional Rate Reduction
    sndcnt = (PRR calculation)
if total_pipe < CCwin:
    // Implicit slowstart
    sndcnt = DeliveredData+MIN(DeliveredData, ABClimit)

SndBank += sndcnt
while (SndBank && TSO_ok())
    SndBank -= transmitData()
```

Algorithm updates

- Draft describes default Laminar versions of:
  - Congestion Avoidance (Reno)
  - Restart after idle
  - Congestion Window Validation
  - Pacing (generic)
  - RTO and F-RTO
  - Undo (generic)
  - Control Block Interdependence
  - Non-SACK TCP
- However there are many opportunities for improvement

Technical summary

- Today cwnd does both CC and transmission scheduling
  - Which are often in conflict
  - Every algorithm has to avoid compromising other uses
- Many pairs of functions interact poorly:
  - Congestion control and loss recovery
  - Application stalls and loss recovery
  - Pacing and CC
  - CC and restart after idle
  - etc
- Laminar separates CC and transmission scheduling
  - They become independent
  - Can evolve separately
  - No "cross subsystem" interactions

TCPM Issues

- Laminar removes ssthresh and cwnd
  - Updates or obsoletes approximately 60 RFC's
  - Interim plan: organize draft parallel to existing docs
- Most algorithm changes are straight forward
  - TCPM style standards (re)design
  - A few details have no precedent or otherwise call for significant redesign: Move to ICCRG?
- At what level (time?) does TCPM want to get involved?
  - Best if original authors redesign their own algorithms

Fluid model Congestion Control

On every ACK: // Including during recovery
    CCwin += MAX(DeliveredData, ABClimit)*MSS/CCwin
On retransmission:
    oldCC = CCwin
    if (CCwin == MAX_WIN):
        CCwin = initialCCestimate(total_pipe)
    CCwin = CCwin/2
    undoDelta = oldCC - CCwin
Undo:
    CCwin = MIN(CCwin+undoDelta, MAX_WIN)
    undoDelta = 0

Insensitive to reordering and spurious retransmissions!