Laminar TCP and Related Problems

draft-mathis-tcpm-laminar-tcp-01

Matt Mathis
mattmathis@google.com

ICCRG, IETF-84

July 30, 2012

No NDA or GPL'd content

- This presentation is intended to be safe for all
- No non-standard algorithms
  - Except perhaps TCP Segmentation Offload (TSO)
- Covers status of current work
  - No code details

Current status

- There is a published clean patch against Linux 3.5
  - https://developers.google.com/speed/protocols/tcp-laminar
  - Optimized for (stand alone) clarity
  - No glaring bugs
- Planned additional work
  - Cosmetic cleanups (var names, etc)
  - Reduce the delta footprint
  - Split into multiple pieces
    - Laminar core
      - Probably subdivided into increments
    - Plus one or more addons

Why partition the patch?

- Laminar core is "just" a refactor
  - Does not change properties or performance
- Need to demo new algorithms based on Laminar
  - But not part of the core algorithm
- The tipping point will be when
  - core + (some) addons reach consensus

Possible Laminar Addons

- Restart after idle & cwnd validation (multiple versions?)
  - Laminar versions of existing algorithm(s)
  - At least one paced version
  - Existing Laminar core contains overly conservative cwv
    - To be moved to an addon patch
- Fluid model version of Reno (and cubic?)
  - See the prior ICCRG slides (below)
- Weighted Relentless
  - See May 2009 ICCRG
- Restate cubic hystart
  - Moved from CC to transmission scheduling

Important point: Laminar proper is performance neutral.
Additions are **required** to justify the effort.

Your input

- Play with the code
  - I am happy to accept suggestions & feedback
- Update your favorite CC module
  - I can't do the ones that I don't use
- If Laminar effects your (past or present) doc
  - Are there conflicts or other problems?
  - Does it make things easier/better?

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.

Laminar: Two separate subsystems

- Pure congestion control
  - New state variable: CCwin
  - Target quantity of data to be sent during each RTT
  - Carries state between successive RTTs
  - Not concerned with detailed timing, bursts etc
- Transmission scheduling
  - Primary state is implicit, recomputed on every ACK
  - Controls exactly when to (re)transmit data
  - Tries to follow CCwin
  - Little or no explicit long term state
  - Includes slowstart, burst suppression, (future) pacing
  - Variables: pipe (3517), total_pipe and DeliveredData

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()
```

Fluid model Congestion Control

(Reno done better, CCwin in fractional bytes)

On every ACK: // Including during recovery

```
CCwin += MAX(DeliveredData, ABClimit)*MSS/CCwin
```

On retransmission:

```
oCCwin = CCwin
if (CCwin == MAX_WIN):
        CCwin = initialCCestimate(total_pipe)
CCwin = CCwin/2
undoDelta = oCCwin - CCwin
```

Undo:

```
CCwin = MIN(CCwin+undoDelta, MAX_WIN)
undoDelta = 0
```

Fluid model properties

- Insensitive to reordering and packet boundaries
  - Total increment based on total forward progress in bytes
- Insensitive to spurious retransmissions
  - Undo and AI are both linear and order insensitive
- Closer agreement between the code and formal models
  - No "boundary condition" for data during recovery
  - CCwin rises during recovery while PRR reduces pipe

My bet: many things **we think we know** about congestion control are not totally right.

Transmission scheduling opportunities

- In existing implementations, TS is degenerate
  - Override long term CC state by futzing with cwnd
  - Sometimes put long term state in ssthresh
  - No "space" for new features
- Under Laminar hybrid self clock and paced is natural
  - Can pace following application stalls, etc
  - Compute rate from CCwin, total_pipe and RTT
- Huge "green field" of unexplored research opportunities
  - Many new problems seeking new solutions

Conclusion

- Laminar has the potential to change many things
- Entirely separate long and short time scales
- Entirely distinct algorithms for each
- Free both from code complexity and interactions
- Much opportunity for new research
- Much opportunity to re-evaluate old experiment