Laminar TCP and Related Problems

draft-mathis-tcpm-laminar-tcp-00

Matt Mathis
mattmathis@google.com

ICCRG, IETF-83
Mar 27, 2011
(Main presentation will be in TCPM)

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
    - ...

Proposal: Refactor TCP

- New functional partition.
  - New state variables
  - Separate:
    - Congestion Control from
    - Transmission Scheduling
- Recast (most) existing standards into new variables
  - Rewrite to replace cwnd and ssthresh
  - Preserve well specified primary behaviors
  - Best for TCPM, with it's standards oriented perspective
- Opportunities to do a few things much better
  - Probably best for ICCRG

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

Variables

- CCwin: (Target) Congestion Control window
    - Replaces both ssthresh and cwnd

- 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 ID)
- 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()
```

TCPM Perspective

- Need Laminar versions of standard algorithms:
  - Congestion Avoidance (Reno)
  - Congestion Window Validation
  - RTO and F-RTO
  - Undo (generic)
  - Control Block Interdependence
  - Non-SACK TCP
  - ...
- The intent is to (mostly) preserve existing behavior
  - Ideally, packet by packet identical
  - Except in some known problem cases

Overview of Research Issues

- Both subsystems can be improved
  - Untangling the current spaghetti will foster evolution
  - Better CC algorithms
    - E.g. Even basic Reno can be improved
  - Better transmission scheduling
    - E.g. Hybrid pace and ACK clock
    - Pace after idle
    - ... many more ...
- Current complexity inhibits rogue CC
  - Simple hacks generally cause negative gain
  - How to prevent a "tragedy of the commons"

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 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

Congestion control risks

- Laminar will withstand aggressive CC algorithms
- What forces (might) regulate global congestion levels?

The congestion exposure (ConEx) WG is a huge step forward (Thurs 3rd PM meeting slot)

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