        Laminar TCP and the case for refactoring TCP congestion control
                 draft-mathis-tcpm-tcp-laminar-00.txt

Abstract

   The primary state variables used by all TCP congestion control
   algorithms, cwnd and ssthresh are heavily overloaded, carrying
   different semantics in different states. This leads to excess
   implementation complexity and poorly defined behaviors under some
   combinations of events, such as loss recovery during cwnd validation.
   We propose a new framework for TCP congestion control, and to recast
   current standard algorithms to use new state variables. This new
   framework will not generally change the behavior of any of the
   primary congestion control algorithms when invoked in isolation but
   will to permit new algorithms with better behaviors in many corner
   cases, such as when two distinct primary algorithms are invoked
   concurrently. It will also foster the creation of new algorithms to
   address some events that are poorly treated by today's standards.
   For the vast majority of traditional algorithms the transformation to
   the new state variables is completely straightforward. However, the
   resulting implementation will technically be in violation of all
   existing TCP standards, even if it is fully compliant with their
   principles and intent.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

   The primary state variables used by all TCP congestion control
   algorithms, cwnd and ssthresh, are heavily overloaded, carrying
   different semantics in different states. This leads to excess
   implementation complexity and poorly defined behaviors under some
   combinations of events, such as overlapping application stalls and
   loss recovery. Multiple algorithms sharing the same state variables
   lead to excess complexity and conflicting correctness constraints,
   making it unreasonably difficult to implement, test and evaluate new
   algorithms.

   We are proposing a new framework for TCP congestion control and it
   use new state variables that separate transmission scheduling, which
   determines precisely when data is sent, from congestion control,
   which determines the amount of data to be sent in each RTT. This
   separation greatly simplifies the interactions between the two
   subsystems and permits vast range of new algorithms that are not
   feasible with the current parameterization.

   This note describes the new framework, represented through its state
   variables, and presents a preliminary mapping between current
   standards and new algorithms based on the new state variables. At
   this point the new algorithms are not fully specified, and many have
   still unconstrained design choices. In most cases, our goal is to
   precisely mimic todays standard TCP, at least as far as well defined
   primary behaviors. In general, it is a non-goal to mimic behaviors
   in poorly defined corner cases, or other cases where standard
   behaviors are viewed as being problematic.

   It is called Laminar because one of its design goals is to eliminate
   unnecessary turbulence introduced by TCP itself.

1.1. Overview of the new algorithm

   The new framework separate transmission scheduling, which determines
   precisely when data is sent, from Congestion Control, which
   determines the total amount of data sent in any given RTT.

   The default algorithm for transmission scheduling is a strict
   implementation of Van Jacobsons' packet conservation principle
   [Jacobson88]. Data arriving at the receiver cause ACKs which in turn
   cause the sender to transmit an equivalent quantity of data back into
   the network. The primary state variable is implicit in the quantity
   of data and ACKs circulating in the network. This state observed
   through a new "total_pipe" estimator, which is a generalization of
   "pipe" as described in RFC 3517. [RFC3517]

A new state variable, CCwin, is the primary congestion control state variable. It is updated only by the congestion control algorithms, which are concerned with detecting and regulating the overall level of congestion along the path. CCwin is TCP's best estimate for an appropriate average window size. In general, it rises when the network seem to be underfilled and is reduced in the presence of congestion signals, such as loss, ECN marks or increased delay. Although CCwin resembles cwnd, it is actually quite different, for one thing the new parameterization does not use ssthresh at all.

Any time CCwin is larger than total_pipe, the default algorithm to grow total_pipe is for each ACK to trigger one segment of additional data. This is essentially an implicit slowstart, but it is gated by the difference between CCwin and total_pipe, rather than the difference between cwnd and ssthresh.

During Fast Retransmit, the congestion control algorithm, such as CUBIC, generally reduces CCwin in a single step. Proportional Rate Reduction [PRR] is used to gradually reduce total_pipe to agree with CCwin. PRR is based on Laminar principles, so its specification has many parallels to this document.

Connection startup is accomplished as follows: CCwin is set to MAX_WINDOW (akin to ssthresh), and IW segments are transmitted. The ACKs from these segments trigger additional data transmissions, and slowstart proceeds as it does today. The very first congestion event is a special case because there is not a prior value for CCwin. By default on the first congestion event only, CCwin would be set from total_pipe, and then standard congestion control is invoked.

The primary advantage of the Laminar framework is that by partitioning congestion control and transmission scheduling into separate subsystems, each is subject to far simpler simpler design constraints, making it far easier to develop many new algorithms that are not feasible with the current organization of the code.

1.2. Standards Impact

Since we are proposing to to refactor existing standards into new state variables, all of the current congestion control standards documents will potentially need to be revised. Note that there are roughly 60 RFC that mention cwnd or ssthresh, and all of them should be reviewed for material that may need to be updated.

This document does not propose to change the TCP friendly paradigm. By default all updated algorithms using these new state variables would have behaviors similar to the current TCP implementations. We do however anticipate some second order effects which we will address

in section XXX below. For example while testing PRR it was observed
that suppressing bursts by slightly delaying transmissions can
improve average performance, even though in a strict sense the new
algorithm is less aggressive than the old.

1.3. Meta Language

We use the following terms when describing algorithms and their
alternatives:

Standard - The current state of the art, including both formal
standards and widely deployed algorithms that have come into standard
use, even though they may not be formally specified. [Although PRR
does not yet technically meet these criteria, we include it here].

default - The simplest or most straightforward algorithm that fits
within the Laminar framework. For example implicit slowstart
whenever total_pipe is less than CCwin. This term does not make a
statment about the relative aggressiveness or any other properties of
the algorithm except that it is a reasonable choice and
straightforward to implement.

conformant - An algorithm that can produce the same packet trace as a
TCP implementation that strictly conforms to the current standards.

mimic - An algorithm constructed to be conformant to standards.

opportunity - An algorithm that can do something better than the
standard algorithm, typically better behavior in a corner cases that
is either not well specified or where the standard behavior is viewed
as being less than ideal.

more/less aggressive - Any algorithm that sends segments earlier/
later than another (typically conformant) algorithm under identical
sequences of events. Note that this is an evaluation of the packet
level behavior, and does not reflect any higher order effects.

Net more/less aggressive - Any algorithm that gets more/less average
data rate than another (typically conformant) algorithm. This is an
empirical statement based on measurement (or perhaps justified
speculation), and potentially indicates a problem with failing to be
"TCP friendly".

2. State variables and definitions

CCwin - The primary congestion control state variable.

DeliveredData - The total number of bytes that the current ACK
indicates have been delivered to the receiver. (See PRR for more
detail).

total_pipe - The total quantity of circulating data and ACKs. In
addition to RFC 3517 pipe, it includes DeliveredData for the current
ack, plus any data held for delayed transmission, for example to
permit a later TSO transmission.

sendcnt - The quantity of data to be sent in response to the current
event.

application stall - The application is failing to keep TCP in bulk
mode: either the sender is running out of data to send, or the
receiver is not reading it fast enough. When there is an application
stall, congestion control does not regulate data transmission and
some of the protocol events are triggered by application reads or
writes, as appropriate.

3. Updated Algorithms

A survey of standard, common and proposed algorithms, and how they
might be reimplemented under the Laminar framework.

3.1. Congestion avoidance

Under the Laminar framework the loss recovery mechanism does not, by
default, interfere with the primary congestion control algorithms.
The CCwin state variable is updated only by the algorithms that
decide how much data to send on successive round trips. For example
standard Reno AIMD congestion control [RFC5681] can be implemented by
raising CCwin by one segment every CCwin worth of ACKs (once per RTT)
and halving it on every loss or ECN signal (e.g. CCwin = CCwin/2).
During recovery the transmission scheduling part of the Laminar
framework makes the necessary adjustments to bring total_pipe to
agree with CCwin, without tampering with CCwin.

This separation between computing CCwin and transmission scheduling
will enable new classes of congestion control algorithms, such as
fluid models that adjust CCwin on every ACK, even during recovery.
This is safe because raising CCwin does not directly trigger any
transmissions, it just steers the transmission scheduling closer to
the end of recovery. Fluid models have a number of advantages, such
as simpler closed form mathematical representations, and are
intrinsically more tolerant to reordering since non-recovery
disordered states don't inhibit growing the window.

Investigating alternative algorithms and their impact is out of scope
for this document. It is important to note that while our goal here
is not to alter the TCP friendly paradigm, Laminar does not include
any implicit or explicit mechanism to prevent a Tragedy of the
Commons. However, see the comments in Section 6.

The initial slowstart does not use the CCwin, except that CCwin
starts at the largest possible value. It is the transmission
scheduling algorithms that are responsible for performing the
slowstart. On the first loss it is necessary to compute a reasonable
CCwin from total_pipe. Ideally, we might save total_pipe at the time
each segment is scheduled for transmission, and use the saved value
associated with the lost segment to prime CCwin. However, this
approach requires extra state attached to every segment in the
retransmit queue. A simpler approach is to have a mathematical model
the slowstart, and to prime CCwin from total_pipe at the time the
loss is detected, but scaled down by the effective slowstart
multiplier (e.g. 1.5 or 2). In either case, once CCwin is primed
from total_pipe, it is typically appropriate to invoke the reduction
on loss function, to reduce it again per the congestion control
algorithm.

Nearly all congestion control algorithms need to have some mechanism
to prevent CCwin from growing while it is not regulating
transmissions e.g. during application stalls.

3.2. Proportional Rate Reduction

Since PRR [I-D.ietf-tcpm-proportional-rate-reduction] was designed
with Laminar principles in mind, updating it is a straightforward
variable substitution. CCwin replaces ssthresh, and RecoverFS is
initialized from total_pipe at the beginning of recovery. Thus PRR
provides a gradual window reduction from the prior total_pipe down to
the new CCwin.

There is one important difference from the current standards: CCwin
is computed solely on the basis of the prior value of CCwin. Compare
this to RFC 5681 which specifies that the congestion control function
is computed on the basis of the FlightSize (e.g.
ssthresh=FlightSize/2) This change from prior standard completely
alters how application stalls interact with congestion control.

Consider what happens if there is an application stall for most of
the RTT just before a Fast Retransmit: Under Laminar it is likely
that CCwin will be set to a value that is larger than total_pipe, and
subject to available application data PRR will go directly to
slowstart mode, to raise total_pipe up to CCwin. Note that the final
CCwin value does not depend on the duration of the application stall.

WIth standard TCP, any application stall reducs the final value of
cwnd at the end of recovery. In some sense application stalls during
recovery are treated as though they are additional losses, and have a
detrimental effect on the connection data rate that lasts far longer
than the stall itself.

If there are no application stalls, the standard and Laminar variants
of the PRR algorithm should have identical behaviors. Although it is
tempting to characterize Laminar as being more aggressive than the
standards, it would be more apropos to characterize the standard as
being excessively timid under common combinations of overlapping
events that are not well represented by benchmarks or models.

3.3. Restart after idle, Congestion Window Validation and Pacing

Decoupling congestion control from transmission scheduling permits us
to develop new algorithms to raise total_pipe to CCwin after an
application stall or other events. Although it was stated earlier
that the default transmission scheduling algorithm for raising
total_pipe is an implicit slowstart, there is lots of opportunity for
better algorithms.

We imagine a new class of hybrid transmission scheduling algorithms
that use a combination of pacing and slowstart to reestablish TCP's
self clock. For example, whenever total_pipe is significantly below
CCwin, RTT and CCwin can be used to directly compute a pacing rate.
We suspect that pacing at the previous full rate will prove to be
somewhat brittle, yielding erratic results. It is more likely that a
hybrid strategy will work better, for example by pacing at some
fraction (1/2 or 1/4) of the prior rate until total_pipe reaches some
fraction of CCwin (e.g. CCwin/2) and then using conventional
slowstart to bring total_pipe the rest of the way up to CCwin

This is far less aggressive than standard TCP without cwnd validation
[RFC2861]or when the application stall was less than one RTO, since
standards permit TCP to send a full cwnd size burst in these
situations. It is potentially more aggressive than conventional
slowstart invoked by cwnd validation when the application stall is
longer than several RTOs. Both standard behaviors in these
situations have always been viewed as problematic, because interface
rate bursts are clearly too aggressive and a full slowstart is
clearly too conservative. Mimicking either is a non-goal, when there
is ample opportunity to find a better compromise.

Although strictly speaking any new transmission scheduling algorithms
are independent of the Laminar framework, they are expected to have
substantially better behavior in many common environments and as such
strongly motivate the effort required to refactor TCP implementations

and standards.

3.4. RTO and F-RTO

We are not proposing any changes to the RTO timer or the
F-RTO[RFC5682] algorithm used to detect spurious retransmissions.
Once it is determined that segments were lost, CCwin is updated to a
new value as determined by the congestion control function, and
Laminar implicit slowstart is used to clock out (re)transmissions.
Once all holes are filled, a hybrid paced transmissions can be used
to reestablish TCPs self clock at the new data rate. This can be the
same hybrid pacing algorithm as is used to recover the self clock
after application stalls.

Note that as long as there is non-contiguous data at the receiver the
retransmission algorithms require timely SACK information to make
proper decisions about which segments to send. Pacing during loss
recovery is not recommended without further investigation.

3.5. Undo

Since CCwin is not used to implement transmission scheduling, undo is
trivial. CCwin can just be set back to a prior value and the
transmission scheduling algorithm will transmit more (or less) data
as needed.

3.6. Control Block Interdependence

Under the Laminar framework, congestion control state can be easily
shared between connections[RFC2140]. An ensemble of connections can
each maintain their own total_pipe (partial_pipe?) which in aggregate
tracks a single common CCwin. A master transmission scheduler
allocates permission to send (sndcnt) to each of the constituent
connection on the basis of the difference between the CCwin and the
aggregate total_pipe, and a fairness or capacity allocation policy
that balances the flows. Note that ACKs on one connection in an
ensemble might be used to clock transmissions on another connection,
and that following a loss, the window reductions can be allocated to
flows other than the one experiencing the loss.

3.7. New Reno

The key to making Laminar function well without SACK is having good
estimators for DeliveredData and total_pipe. By definition every
duplicate ACK indicates that one segment has arrived at the receiver
and total_pipe has fallen by one. On any ACK that advances snd.una,
total pipe can be updated from snd.nxt-snd.una, and DeliveredData is
the change in snd.una, minus the estimated DeliveredData of the

preceding duplicate ACKs.

4. **Example Pseudocode**

The example pseudocode in this section incorporates (or subsumes) the
following algorithms:

On startup:

```
  CCwin = MAX_WINOW
  sndBank = IW
```

On every ACK:

```
  DeliveredData = delta(snd.una) + delta(SACKd)
  pipe = (RFC 3517 pipe algorithm)
  total_pipe = pipe+DeliveredData+sndBank
  sndcnt = DeliveredData     // Default outcome


  if new_recovery():
     if CCwin == MAX_WIN:
        CCwin = total_pipe/2 // First time only
     CCwin = CCwin/2         // Reno congestion control
     prr_delivered = 0       // Total bytes delivered during recov
     prr_out = 0             // Total bytes sent during recovery
     RecoverFS = total_pipe  //


  if !in_recovery() && !application_limited():
     CCwin += (MSS/CCwin)
  prr_delivered += DeliveredData  // noop if not in recovery
```

```
   if total_pipe > CCwin:
      // Proportional Rate Reduction
      sndcnt = CEIL(prr_delivered * CCwin / RecoverFS) - prr_out

   else if total_pipe < CCwin:
      if in_recovery():
         // PRR Slow Start Reduction Bound
         limit = MAX(prr_delivered - prr_out, DeliveredData) + SMSS
         sndcnt = MIN(CCwin - total_pipe, limit)
      else:
         // slow start with appropriate byte counting
         inc = MIN(DeliveredData, 2*MSS)
         sndcnt = DeliveredData + inc


   // cue the (re)transmission machinery
   sndBank += sndcnt
   limit = maxBank()
   if sndBank > limit:
      sndBank = limit
   tcp_output()

For any data transmission or retransmission:

tcp_output():
   while sndBank && tso_ok():
      len = sendsomething()
      sndBank -= len
      prr_out += len  // noop if not in recovery
```

5. Compatibility with existing implementations

On a segment by segment basis, the above algorithm is [believed to
be] fully conformant with or less aggressive than standards under all
conditions.

However this condition is not sufficient to guarantee that average
performance can't be substantially better (net more aggressive) than
standards. Consider an application that keeps TCP in bulk mode
nearly all of the time, but has occasional pauses that last some
fraction of one RTT. A fully conformant TCP would be permitted to
"catch up" by sending a partial window burst at full interface rate.
In some networks, such bursts might be very disruptive, causing
otherwise unnecessary packet losses and corresponding cwnd
reductions.

In Laminar, such a burst would be permitted, but the default
algorithm would be slowstart. A better algorithm would be to pace
the data at (some fraction of) the prior rate. Neither pacing nor
slowstart is likely to cause unnecessary losses, and as was observed
while testing PRR, being less aggressive at the segment level has the
potential to increase average performance[IMC11PRR]. In this
scenario Laminar with pacing has the potential to outperform both of
the behaviors described by standards.

6. Security Considerations

The Laminar framework does not change the risk profile for TCP (or
other transport protocols) themselves.

However, the complexity of current algorithms as embodied in today's
code present a substantial barrier to people wishing to cheat "TCP
friendliness". It is a fairly well known and easily rediscovered
result that custom tweaks to make TCP more aggressive in one
environment generally make it fragile and perform less well across
the extreme diversity of the Internet. This negative outcome is a
substantial intrinsic barrier to wide deployment of rogue congestion
control algorithms.

A direct consequence of the changes proposed in this note, decoupling
congestion control from other algorithms, is likely to reduce the
barrier to rogue algorithms. However this separation and the ability
to introduce new congestion control algorithms is a key part of the
motivation for this work.

It is also important to note that web browsers have already largely
defeated TCP's ability to regulate congestion by opening many
concurrent connections. When a Web page contains content served from
multiple domains (the norm these days) all modern browsers open
between 35 and 60 connections (see:
http://www.browserscope.org/?category=network). This is the Web
community's deliberate workaround for TCP's perceived poor
performance and inability fill certain kinds of consumer grade
networks. As a consequence the transport layer has already lost a
substantial portion of its ability to regulate congestion. It was
not anticipated that the tragedy of the commons in Internet
congestion would be driven by competition between applications and
not TCP implementations.

In the short term, we can continue to try to use standards and peer
pressure to moderate the rise in overall congestion levels, however
the only real solution is to develop mechanisms in the Internet
itself to apply some sort of backpressure to overly aggressive

applications and transport protocols. We need to redouble efforts by
the ConEx WG and others to develop mechanisms to inform policy with
information about congestion and it's causes. Otherwise we have a
looming tragedy of the commons, in which TCP has only a minor role.

Implementers that change Laminar from counting bytes to segments have
to be cautious about the effects of ACK splitting attacks[Savage99],
where the receiver acknowledges partial segments for the purpose of
confusing the sender's congestion accounting.

7. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an
RFC.

8. References

[Jacobson88]
          Jacobson, V., "Congestion Avoidance and Control",
          SIGCOMM 18(4), August 1988.

[RFC2140] Touch, J., "TCP Control Block Interdependence", RFC 2140,
          April 1997.

[RFC2861] Handley, M., Padhye, J., and S. Floyd, "TCP Congestion
          Window Validation", RFC 2861, June 2000.

[RFC3517] Blanton, E., Allman, M., Fall, K., and L. Wang, "A
          Conservative Selective Acknowledgment (SACK)-based Loss
          Recovery Algorithm for TCP", RFC 3517, April 2003.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion
          Control", RFC 5681, September 2009.

[RFC5682] Sarolahti, P., Kojo, M., Yamamoto, K., and M. Hata,
          "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting
          Spurious Retransmission Timeouts with TCP", RFC 5682,
          September 2009.

[I-D.ietf-tcpm-proportional-rate-reduction]
          Mathis, M., Dukkipati, N., and Y. Cheng, "Proportional
          Rate Reduction for TCP",
          draft-ietf-tcpm-proportional-rate-reduction-00 (work in
          progress), October 2011.

   [IMC11PRR]
             Mathis, M., Dukkipati, N., Cheng, Y., and M. Ghobadi,
             "Proportional Rate Reduction for TCP", Proceedings of the
             2011 ACM SIGCOMM conference on Internet measurement
             conference , 2011.

   [Savage99]
             Savage, S., Cardwell, N., Wetherall, D., and T. Anderson,
             "TCP congestion control with a misbehaving receiver",
             SIGCOMM Comput. Commun. Rev. 29(5), October 1999.

Author's Address

   Matt Mathis
   Google, Inc
   1600 Amphitheater Parkway
   Mountain View, California 93117
   USA

   Email: mattmathis@google.com