TCPM Working Group
Internet Draft
Intended status: Standards Track
Expires: January 2012

J. Touch
USC/ISI

July 7, 2011

Automating the Initial Window in TCP
draft-touch-tcpm-automatic-iw-01.txt

Status of this Memo

Copyright Notice

Abstract

The Initial Window (IW) provides the starting point for TCP's feedback-based congestion control algorithm. Its value has increased over time to increase performance and to reflect increased capability of Internet devices. This document describes a mechanism to adjust the IW over long timescales, to make future changes more safely deployed and to potentially avoid reexamination of this value in the future.

Table of Contents

1. Introduction

TCP's congestion control algorithm uses an initial window value (IW), both as a starting point for new connections and after one RTO or more [RFC2581][RFC2861]. This value has evolved over time, originally one maximum segment size (MSS), and increased to the lesser of four MSS or 4,380 bytes [RFC3390][RFC5681]. For typical Internet connections with an maximum transmission units (MTUs) of 1500 bytes, this permits three segments of 1,460 bytes each.

The IW value was originally implied in the original TCP congestion control description, and documented as a standard in 1997 [RFC2001][Ja88]. The value was last updated in 1998 experimentally, and moved to the standards track in 2002 [RFC2414][RFC3390]. There have been recent proposals to update the IW based on further increases in host and router capabilities and network capacity, some

focusing on specific values (e.g., IW=10), and others prescribing a schedule for increases over time (e.g., IW=6 for 2011, increasing by 1-2 MSS per year).

This document proposes that TCP can objectively measure when an IW is too large, and that such feedback should be used over long timescales to adjust the IW automatically. The result should be safer to deploy and might avoid the need to repeatedly revisit IW size over time.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

In this document, the characters ">>" preceding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

3. Design Considerations

TCP's IW value has existed statically for over two decades, so any solution to adjusting the IW dynamically should have similarly stable, non-invasive effects on the performance and complexity of TCP. In order to be fair, the IW should be similar for most machines on the public Internet. Finally, a desirable goal is to develop a self-correcting algorithm, so that IW values that cause network problems can be avoided. To that end, we propose the following list of design goals:

- Little to no impact to TCP in the absence of loss, i.e., it should not increase the complexity of default packet processing in the normal case.

- Adapt to network feedback over long timescales, avoiding values that persistently cause network problems.

We expect that, without other context, a good IW algorithm will converge to a single value, but this is not required. An endpoint with additional context or information, or deployed in a constrained environment, can always use a different value. In specific, information from previous connections, or sets of connections with a similar path, can already be used as context for such decisions [RFC2140].

However, if a given IW value persistently causes packet loss during the initial burst of packets, it is clearly inappropriate and could be inducing unnecessary loss in other competing connections. This might happen for sites behind very slow boxes with small buffers, which may or may not be the first hop.

4. Proposed IW Algorithm

Below is a simple description of the proposed IW algorithm. It relies on the following parameters:

- MinIW = 3 MSS or 4,380 bytes (as per RFC3390]
- MaxIW = date.year - 2000
- MulDecr = 0.5
- AddIncr = 2 MSS
- Threshold = 0.05

We assume that the minimum IW (MinIW) should be as currently specified [RFC3390]. The maximum IW can either be set to a fixed value [Ch10], or set based on a schedule [Al10]. Regardless, we propose that the value adapt over time, so have specified it in terms of the current date. If that is not feasible or the time is not available, a fixed value can be used. We also propose to use an AIMD algorithm, with increase and decreases as noted.

Note that all of these parameters are up for discussion, though should be determined by the time this document is issued as an RFC. We do not anticipate that any of them are critical to the overall design, especially because both current proposals are degenerate

cases of the algorithm below for given parameters (notably MulDec = 1.0 and AddIncr = 0 MSS, thus disabling the automatic part of the algorithm).

The specific algorithm is as follows:

0. On boot:

```
IW = MaxIW; # assume this is in bytes, and an even number of MSS
```

1. Upon starting a new connection


```
CWND = IW;
conncount++;
IWnotchecked = 1; # true
```

2. If SYN-ACK includes ECN, treat as if the IW is too large

```
if (synackecn == 1) {
   losscount++;
   IWnotchecked = 0; # never check again
}
```

3. During a retransmission, check the seqno of the outgoing packet (in bytes)

```
if (IWnotchecked && ((ISN - seqno) < IW))) {
   losscount++;
   IWnotchecked = 0; # never do this entire "if" again
} else {
   IWnotchecked = 0; # you're beyond the IW so stop checking
}
```

4. Once a month or once every 1000 connections if no date is available:

```
if ((monthly == TRUE) || (conncount > 1000)) {
   if (losscount/conncount > threshold) {
      # the number of connections with errors is too high
      IW = IW * MulDecr;
```

```
      } else {
         IW = IW + AddIncr;
      }
   }
```

We recognize that this algorithm can yield a false positive when the sequence number wraps around. In that case, we might be able to use PAWS to avoid the issue, encourage the use of 64-bit sequence numbers internal to the implementation, or ignore the issue and just allow the false positives [RFC1323].

Standards language (as a shopping list):

MAY implement this as an alternative to RFC3390

If implemented:

MUST start IW at MaxIW - i.e., IW in the absence of other info

MUST limit MaxIW growth (Static or to year if poss)

MUST check once a month or 1,000 connections (the larger)

MUST decrease by at least 0.5x

MUST NOT increase by more than 2 MSS

SHOULD use IW that is integer multiple of 2 MSS (for ACK compression)

MUST decrease IW if > 95% connections have errors

MAY increase IW otherwise

But MUST limit increase to 2 MSS/year (is this needed?)

SHOULD be implemented to limit performance impact

SHOULD be implemented to avoid seqno wrap issues

(anything else?)

There are some TCP connections which might not be counted at all, such as those to/from loopback addresses, or those within the same subnet as that of a local interface (for which congestion control is sometimes disabled anyway). This may also include connections that terminate before the IW is full, i.e., as a separate check at the time of the connection closing.

The period over which the IW is updated is intended to be a long timescale, e.g., a month or so, or 1,000 connections, whichever is longer. An implementation might check the IW once a month, and simply not update the IW or clear the connection counts in months where the number of connections is too small.

5. Discussion

The following is intended as a list of notes to be discussed:

- Algorithm uses IW as an even multiple of MSS due to ACK compression
- Impact of SEQNO wraparound vs. use of PAWS
- Algorithm now assumes bytes, not segments

- Algorithm now counts losses only during the first IW after start; should the system ignore rechecking the burst after idle, i.e., do checks only once on the initial connection? To fix this, step #2 would use "IWstart" as the front of the IW, set this to ISN at connection start, and reset it to seqno during a slow-start restart. This isn't a lot of code, and takes effect only during restarts anyway - it's not in the fast path either.
- Impact of spurious retransmissions due to reordering (false positive)
- Granularity (per-machine -same as now, per-interface? per-subnet? vs. cost?)
- Need to keep ISN - needed for other uses (e.g., TCP-AO), and typically kept except in Linux.
- Need for persistent state if a reboot occurs within the 1-month

window of evaluation

- o Degenerate case due to failure is to act as if a fixed window, which is what we have now

- o Interaction with 2140

Basically 2140 sets CWND to something other than IW when it knows better; this doc is for IW which is used there only for 'new' places (or forgotten old ones).

- o Explain why RWIN is not involved

Receiver-limited space

Space for reordering

NOT congestion control

Although sender window isn't useful if larger than this

CWND is a path property; RWIN is an endpoint property

- o Reasons not to report-back:

- privacy concerns

- opportunity for spoofer poisoning the data (more on that in the doc)

- using a DNS query is a bad idea
  - requires every TCP stack support DNS queries
  - requires a resolution step in addition to the reporting
  - could cause the kernel to block on a timeout

- biased reporting
  if cellphones (e.g.) never do this, we won't know about a potentially large percent of endpoints

6. Observations

- o The IW may not converge to a global value; that's OK.

- o IW values can fluctuate; there should not be a significant impact to this if that's what's seen by this algorithm.

- o We do assume that losses during the IW are due to the IW being too large; persistent errors that drop packets for other reasons (e.g., OS bugs) can cause false positives, however this is consistent with TCP's general assumption that loss is caused by congestion that requires backoff. This algorithm treats the IW of new connections as a long-timescale backoff system.

7. Security Considerations

Obvious ones - poisoning the info (fake loss, fake success), what happens when one party disobeys, and whether anything is different

---

You can already do that within a connection too. Yes, you can pollute aggregate info by virtue of it being aggregate. There's a tradeoff of trust here - how much do you believe what's happening most of the time, and how do you react to it.

If most of the connections lie about receiving data, then you see a world where larger IW is working, and unless you detect data loss some other way, TCP worked exactly as it should.

IMO, the good news is that:

- the IW drops if you get lots of lies about dropped packets but then those endpoints could have just dropped the packets, and dropping the IW is the right response anyway

- the IW increases if you get lots of lies about non-drops, but then if you don't see anything else, you have no reason to claim that anything is amiss anyway

So yes, there's spoofing in the aggregate. The law of large numbers - connecting to lots of places - should help reduce that effect. But ultimately it's not all that clear that the reactions of this sort of poisoning aren't the right ones anyway.

In the end, it might be safer to require a high percent of connections react badly to IW (i.e., over 95%) - that means that

a) if you did see loss, someone bad is controlling nearly all of your connections anyway

b) if you don't see loss through TCP, and you didn't detect data drops by other means for that many connections, you really don't have a problem

I.e., increasing the threshold increases your ability to detect the false IW increase case, so it's safer...

8. IANA Considerations

This document has no IANA considerations. This section should be removed prior to publication.

9. Conclusions

<Add any conclusions>

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3390] Allman, M., Floyd, S., Partridge, C., "Increasing TCP's Initial Window", RFC 3390 (Standards Track), Oct. 2002.

[RFC5681] Allman, M., Paxson, V., Blanton, E., "TCP Congestion Control," RFC 5681 (Standards Track), Sep. 2009.

10.2. Informative References

[Al10] Allman, M., "Initial Congestion Window Specification",

(work in progress), draft-allman-tcpm-bump-initcwnd-00, Nov. 2010.

[Ch10] Chu, J., Dukkipati, N., Cheng, Y., Mathis, M., "Increasing TCP's Initial Window," (work in progress), draft-ietf-tcpm-initcwnd-01, Apr. 2011.

[Ja88] Jacobson, V., M. Karels, "Congestion Avoidance and Control", Proc. Sigcomm 1988.

[RFC1323] Jacobson, V., Braden, R., Borman, D., "TCP Extensions for High Performance", RFC 1323, May 1992.

[RFC2001] Stevens, W., "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC2001 (Standards Track), Jan. 1997.

[RFC2140] Touch, J., "TCP Control Block Interdependence", RFC 2140 / STD 7(Informational), Apr. 1997.

[RFC2414] Allman, M., Floyd, S., Partridge, C., "Increasing TCP's Initial Window", RFC 2414 (Experimental), Sept. 1998.

[RFC2581] Allman, M., Paxson, V., Stevens, W., "TCP Congestion Control," RFC2581 (Standards Track), Apr. 1999.

[RFC2861] Handley, M., Padhye, J., Floyd, S., "TCP Congestion Window Validation", RFC2861 (Experimental),

11. Acknowledgments

Mark Allman and Aki Nyrjinen contributed to the development of this algorithm. Members of the TCPM mailing list also participated in providing useful feedback.

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Joe Touch
USC/ISI
4676 Admiralty Way
Marina del Rey, CA 90292-6695 U.S.A.

Phone: +1 (310) 448-9151
Email: touch@isi.edu