Workgroup: WG Working Group
Internet-Draft: draft-mathis-tsvwg-safecc-02
Published: 10 March 2023
Intended Status: Experimental
Expires: 11 September 2023
Authors: M. Mathis
MLab

Safe Congestion Control

Abstract

We present criteria for evaluating Congestion Control Algorithms (CCAs) for behaviors that have the potential to cause harm to Internet applications or users.

Although our primary focus is the safety of transport layer congestion control, many of these criteria should be applied to all protocol layers: entire stacks, libraries and applications themselves.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-mathis-tsvwg-safecc/.

Discussion of this document takes place on the TSVWG Working Group mailing list (mailto:tsvwg@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/tsvwg/. Subscribe at https://www.ietf.org/mailman/listinfo/tsvwg/.

Source for this draft and an issue tracker can be found at https://github.com/mattmathis/safeCC/.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 11 September 2023.

Table of Contents

1. Preamble

This document is written in extra terse congestion control jargon. In the final version many single sentences in this draft will expand into full paragraphs.

Editorial comments to authors are enclosed in [square brackets] or tagged with @@@@.

Unformatted references appear below many sections.

[Remove this section before publication]

2. Introduction

We present criteria for evaluating Congestion Control Algorithms (CCA) for behaviors that have the potential to cause harm to Internet applications or users.

Ideally we would cast these criteria as requirements; however such an effort is doomed to fail because many of them have technical exceptions that are unavoidable in ways that are not important.

[Introduce non-material] For an example of this issue see Section 4.1

As an interim position: all implementations **SHOULD** comply with all criteria, and **MUST** document all exceptions and evaluate the risks associated with the exceptions. Under what circumstances and how severely they fail to comply, and what is the extent of the harm that non-compliance might cause?

To prove the criteria proposed in this note they should be used to evaluate current and legacy CCAs: we expect to find alignment between known CCA pathologies and failed criteria. Any discrepancies may suggest additional criteria or sharpen our understanding of how to decide if a failed criteria is material or not.

Indeed, Reno[rfc5681] and Cubic[Cubic] are known to fail several the criteria presented here, and as a consequence exhibit pathologies including bufferbloat[bufferbloat], [starvation] and poor scaling[Scaling].

3. Conventions and Definitions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Exhaustion a network overloaded to the extent that the average delivery rate is below one segment per flow per RTT.

Material failing a criteria in a manner that is likely to cause pathological behaviors under some conditions.

Non-Material
technically failing some criteria, but unimportant, insignificant or otherwise unlikely to cause pathological behaviors.

Under adverse conditions refers to any increase in any congestion signals (loss, delay, marks or reduced queue space or capacity, etc) from any initial state. For example introducing 1 Mb/s cross traffic to an otherwise ideal 10 Gb/s link is an adverse condition that should not trigger any of the misbehavior's described below.

4. Tentative list of criteria

These are generally in order of declining severity. Items at the top of the list have the potential to cause large scale internet disruptions if they are widely deployed. Items at the bottom of the list can cause unexpected or poor performance to the user.

4.1. Free from congestion collapse

Adverse conditions do not cause increasing overhead, specifically do not cause duplicate data at the receiver.

Test: for a fixed work load, the overhead must be constant, independent of the network congestions across the entire operating range of the application or network

If there is packet loss, the retransmits must exactly match the losses.

Example of an application that can cause congestion collapse: an automatic download engine that responds to transient network errors or persistent congestion by restarting downloads from the beginning. For example git-clone can not be restarted mid transfer. Failures caused by extreme overload or transient outages require removing and re-cloning the destination.

Non-material example of congestion collapse: A download engine that properly restarts from where it left off still needs to repeat the connection establishment, ssl negotiation and other signaling, thus increases its overhead by a few bytes. If the payload is not tiny, this is generally non-material. If the payload is tiny and the relative overhead might be large, and might be prone to congestion collapse.

4.2. Free from regenerative congestion

Adverse conditions must not cause additional presented load. Any congestion indication should cause transmissions to be later than they would have been without the congestion.

This criteria is well understood at the transport layer: all congestion signals must cause the sender to delay future transmissions, and at least slightly reduce their average sending rate.

This criteria is not well understood by application designers. Many applications open multiple transport connections and use aggressive retry strategies with insufficiently adaptive timers (see Section 4.13). This flawed strategy is generally an attempt to maintain constant performance without reacting adverse network conditions.

Some (past?) streaming video application are known to request additional video chunks on alternate connections without regard to the delivery status of chunks already in progress. Such a strategy often yields better performance when the application is a minority of the traffic, but can cause massive regenerative congestion and eventual collapse in a large scale deployment.

4.3. Bound steady state losses

Steady state bulk transport should not cause more than 2% loss [study needed] over any unchanging network.

Any transport with some form of selective acknowledgements can easily operate at a much higher loss rate. The real problem is the harm that transport might cause to all single packet transactions, including DNS queries and connection establishment for nearly all protocols and services. Single packet transactions generally can only use an RTO timer for recovery, often without any preceding RTT measurement, thus they typically take several orders of magnitude more time than any selective acknowledgment based recovery built into a transport protocol.

The question at hand is really how much harm should we permit transport to inflict on all other protocols?

For example, are we ok with happy eyeballs [happyEyeballs] getting the wrong answer 2% of the time, because the IPv6 connection establishment failed on the first message?

[BTW I consider 2% to be a bit excessive: 1% or 0.1% would be better, however that may be unrealistically low. Reno and Cubic can both easily cause much higher loss rates.]

[We need some studies to justify the appropriate value for the final document.]

4.4. Bound slowstart duration and loss

Slowstart into a droptail queue should not cause more than one RTT of loss nor cause more than 50% loss for that RTT. Provisional window or rate reductions should start promptly when losses or disorder is first detected, even before the loss recovery can decide if the missing segments are due to reordering or loss.

4.5. Bound losses on link changes

Step changes in link properties (RTT, bandwidth or queue size) or cross traffic should not cause losses that are larger than the change in maximum flight size supported by the link. Specifically, during loss recovery the transport is not permitted to send more data than the receiver reported as having been delivered. This is the strict Conservative property from Proportional Rate reduction.

4.6. No unnecessary slowstarts

All application stacks must use connection caching, Congestion Control state caching or some other mechanism such that application workloads are prevented from causing persistent or repeated overlapping slowstarts.

[RFC9040] TCP Control Block Interdependence

draft-kuhn-tsvwg-careful-resume-00 Careful convergence of congestion control from retained state with QUIC

4.7. Freedom from starvation

Flows below some resource threshold (data rate, window size, ConEx marks, etc) will successfully search upwards, as long as there is either idle capacity or other flows above the some threshold. To some extent the thresholds will depend on the path properties and other sources of noise.

@@@@ more work is needed here.

4.8. Bound standing queue

In the absence of losses or ECN, bulk flow should not cause steady state standing queues larger than k*minRTT*maxBW, for some predefined k, specific to the CCA. K must be smaller than 2 (maximum RTT would be 3*minRTT)

Note that this criteria implies that ECN based CCAs must also have some mechanism to limit data inflight, and that all CCAs must address the minimum RTT estimator problem described in Appendix A.

4.9. Bound control frequency

Control frequency scales with 1/rtt but is insensitive to data rate. This property is referred to as "scalable" in other sources.

[RFC9330] Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture

Robert Morris Scalable TCP Congestion Control

4.10. Maintain queue headroom

Individual flows do not persistently maintain full queues even if the queues are smaller than minRTT*maxBW. When there is queue full, Congestion Control should reduce its window enough to create some small headroom to prevent locking out new flows.

Ideally this criteria would also be applied to flow aggregates, however significant additional research would be needed. @@@@

4.11. Monotonic response

The CCA should have monotonic response to all congestion signals that it responds to (loss, marks, delay, etc) otherwise it will have multiple stable operating points for the same network conditions. It would be likely to exhibit stable pathologies such as latecomer (dis)advantage.

4.12. Balanced probe size

Balance the worst case queue backlog against the need to trigger mode shifting in links that use queue backlog as a trigger.

Self clock transport preserves ACK modulation from one RTT to the next. Many half duplex link layers implicitly use bursts preserved by transport self clock as part of optimizing their channel allocation algorithms. Batching or decimating ACKs on the return path can cause relatively large bursts of packets to traverse the entire forward path from the sender to the receiver, potentially causing jitter to other flows sharing the same queues.

Pacing can interact poorly with link layers that rely on queue backlogs to trigger transmissions or scheduling mode changes. These types of link scheduling algorithms are pervasive in wireless and other shared media where channel arbitration is relatively expensive. Indeed, the initial design of BBR's bandwidth probe phase

was inspired by the need to trigger mode changes in many wireless networks.

@@@@ More work needed here.

4.13. Self scaling

All protocol layers must be self scaling. If the network is too slow, the application must also slow down to avoid "stacking" requests.

Specifically all application timers that cancel or restart lower layers transactions must not start overlapping transactions and must use an RTO style retry algorithm based on observed transaction times, including exponential backoff on repeated failures. Alternatively an application might refuse to run after excessive failures.

This criteria must be applied recursively all the way up the protocol stack for all applications that might be unattended (e.g. cron jobs and IOT devices).

5. Security Considerations

This document provides evaluation criteria for Congestion Control and other implementations or algorithms that might be deployed on the internet. It has no direct security considerations of its own.

Over the long haul it is expected to increase the overall robustness of the Internet by helping to eliminate certain pathological behaviors that have the potential cause the Internet to be fragile under some conditions.

6. IANA Considerations

This document has no IANA actions.

7. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

Appendix A. Estimating the minimum RTT

It has been shown that all distributed algorithms to measure minimum RTTs in packet switched mesh networks are subject to failures caused by the inability to distinguish between true minimum path delays and delays that have been inflated by standing queues caused by other flows.

This failure mechanism was shown in a formal proof [noPower] and demonstrated in connection with Vegas TCP [vegas][VegasFailure].

BBR [BBR] uses a distributed algorithm designed to protect the network from one of the more easily observed failure cases, where multiple long running flows "stack" standing queues on queues created by prior flows. BBR attempts to explicitly synchronize minimum RTT measurements by having all flows reduce their sending rates for approximately 1 RTT every 10 seconds. The measurements are synchronized by the measurements themselves. When a flow observes a new minimum RTT sample, it set a 10 second timer to schedule its next measurement. If flows are indeed causing "stacked" queues, they are likely to get a new minimum RTT from some other flow's measurement, which will cause synchronized measurements on the next cycle.

It is not known if the minimum RTT algorithm used in BBR is sufficient to protect the Internet from all failure cases. We suspect that the BBR algorithm does not fully mitigate the problem as outlined in the proof [noPower].

However given the transactional nature of modern Internet workloads each flow has frequent idle, which helps other flows observe accurate minimum RTTs.

It is also not known if a naive minimum RTT algorithm, without any attempt a synchronized minimum RTT measurements, is sufficient for to protect the Internet from the problems described in Section 4.8.

L.S. Brakmo, S. O'Malley, and L.L. Peterson. "TCP Vegas: New techniques for congestion detection and avoidance", Computer Communication Review, Vol. 24, No. 4, pp. 24-35, Oct. 1994.

L.S. Brakmo and L.L. Peterson. "TCP Vegas: end to end congestion avoidance on a global internet", IEEE Journal on Selected Areas in Communications, Vol. 13, No. 8, pp. 1465-80, Oct. 1995.

J. Ahn, P. Danzig, Z. Liu, and L. Yan, "Evaluation of TCP Vegas: emulation and experiment", Computer Communication Review, Vol. 25, No. 4, pp. 185-95, Oct. 1995.

https://www.cs.princeton.edu/research/techreps/TR-616-00 [@@@@ Wrong paper?]

[noPower] J. Jaffe, "Flow Control Power is Nondecentralizable," in IEEE Transactions on Communications, vol. 29, no. 9, pp. 1301-1306, September 1981, doi: 10.1109/TCOM.1981.1095152.

Acknowledgments

TODO acknowledge.

Author's Address

Matt Mathis
Freelance, Measurement Lab

Email: mattmathis@measurementlab.net
URI: https://mattmathis.net/