Safe Congestion Control

draft-mathis-ccwg-safecc-00
Matt Mathis - IETF 118
Freelance

First principles approach to evaluating CCAs

- Score CCAs on behaviors known to cause problems
- Ultimate goal: complete and well defined robust measures of CC safety
    - Disallow behaviors that might harm other Internet users
        - Discourage behaviors that cause self harm or user surprises
    - Ideally any CCA that passes all tests would be unconditionally safe to deploy
- Current draft lists 13 criteria
- Each criteria is a stand alone test of CCA properties
    - They place bounds on the shapes of the control functions
    - Many failures can be discovered by inspecting designs
- No explicit comparisons between CCAs
    - But the scores might be compared

Document status

- SafeCC is a “working draft” intended for expert readers
  - No Background or tutorials
  - Very terse language without a lot of explanation
  - Some of the material in this presentation is not in the I-D yet
- I hope to migrate many parts into RFC5033bis
- Other parts are likely to land elsewhere (perhaps RFC5033bisbis)
  - Too much research is needed
  - They might unnecessarily stall RFC5033bis

Upper bound on **self induced loss**

- Goal is to protect all protocols in shared queues, not just other transports
  - DNS, SYN exchanges and all other single packet exchanges are particularly exposed
    - Often rely on simple RTO without prior RTT measurement
- Current draft says 2% (but does not describe test conditions)
  - Reno and CUBIC with SACK are out of conformance
    - Old Reno **without** SACK is probably OK
    - 25% or 33% loss on contrived networks **(Somebody test this please)**
    - Unacceptably high for widespread use
  - I would rather say 0.1%
    - But this is probably unrealistically low
- The current draft is not up to date with my thinking
  - We will need a published, well thought out justification for final text
  - Probably experimental results and a model in a separate paper

Steady state loss

- The relationship between loss probability and data rate must be monotonic
  - Otherwise it is likely that there are multiple stable rates for flows sharing the same bottleneck
  - and secondary symptoms such as bimodal data rates and late comer (dis)advantage
- Control period (1/frequency) should scale with RTT
  - i.e. upper and lower bounds on number of RTTs between CC adjustments
  - Control frequency should not be a function of data rate (at any fixed RTT)
    - Otherwise the CCA is unlikely to age well as the Internet continues to get faster
  - Note that Reno and CUBIC fail this criteria
- Similar principles apply to queueing delay and CE marks

Consider queueing delay

- All CCAs must limit queueing delay to some appropriate bound
    - Otherwise they might cause large standing queues, aka Bufferbloat
        - e.g. on any overbuffereed lossless bottleneck
- Note that ECN support in the CCA is insufficient
    - Bottlenecks that don't support AQM or ECN would still suffer
- Reno and CUBIC are non-compliant
    - And cause a lot of harm to other better behaved CCAs

The minRTT problem

- Robust minRTT estimators are a known to be problematic
    - Provably unsolvable [Jaffe1981NetTrans]
        - A flow can not detect when the minRTT has been inflated by a standing queue caused by other flows
    - Demonstrated failure for Vegas TCP
    - Some version of this problem applies to **ALL** current and future CCAs
    - BBR RTTprobing is designed to overcome this problem
        - but some small risk might remain
            - Grounds for wanting some form of AQM everywhere
    - FUTURE: we may need a uniform minRTT estimator, akin to the RTO estimator
        - Standardized for all CCAs and protocols
- Also must handle non-stationary minimum delay
        - Leo Satellites
        - Routing changes and path diversity

Freedom from starvation

- Large flows must not starve small and starting flows
  - The distinction between small and large must self scale
  - Must apply for all mixed traffic, with multiple CCAs
  - This will probably create a weak form of fairness implicit in balancing “large" vs “small”
  - Efficiency (filling arbitrary networks) is explicitly NOT required
    - Efficiency has been proven to conflict with freedom from starvation [Arun2022SigComm]
- More important than Fairness or Efficiency on many networks
- One approach is easy
  - Forbid CCAs from needlessly maintaining persistent full queues
  - This might eventually become grounds for disqualifying Reno and CUBIC
- Much more research is needed
  - I expect RFC5033bis to say something informal and somewhat vague
  - This is likely to require a separate paper

Concept of “under adverse conditions” (UAC)

- Linguistic shorthand
  - Generally statements of monotonicity over all network conditions
    - Simple concept
    - Complicated to say precisely
    - Brutal to repeat everywhere it is needed
  - Akin to epsilon-delta limit proofs in mathematics
- Imagine testing across the “entire” parameter space
  - Bandwidth, RTT, queue space, cross traffic, [random] loss, CE marks, etc
    - Many orders of magnitude in all dimensions
- **For all starting conditions and all small incremental changes**
  - **the stated property must hold**

Expand RFC5033's definition of congestion collapse

- RFC5033bis currently only requires RTO with exponential backoff
- I plan to contribute two new constraints
  - Overhead must not increase UAC
    - No duplicate data at the receiver
  - No regenerative congestion UAC
    - Congestion always delays future transmissions

Overhead must not increase UAC

- Underlying problem that caused the 1986-87 Internet collapses
    - Jacoboson88 provided a solution
- Observable at the receiver and precisely defined
    - Total_bytes_arriving / total_content_bytes must not increase UAC
    - Rerun the same workload under different conditions
        - Total bytes received should be constant, independent of network conditions
- Many failures can be discovered by thought experiments on designs
- Well understood in the transport area (and our documents)

No Regenerative Congestion UAC

- Never increase presented load under UAC
    - Retransmission and all future transmissions must be delayed
- Observable and precisely defined at the sender
    - Bytes sent vs time must shift to the right UAC
- Probably understood well enough in the transport area

Open questions and Next Steps

- Contributing text (and issues) to 5033bis
  - I am at a bit of a loss for generating mergeable PRs
- Justifying some bound on self induced loss
  - 2%, 1%, 0.5% ....
- Many unresolved questions about transactions and startup behavior
  - Slowstarts have to be partially exempt from steady state rules

Additional Material

Congestion collapse also applies to applications

- Application designers often think:
    - “TCP will protect the network from congestion collapse”
    - They do not consider congestion collapse to be their problem
- Applications (and libraries) often fail badly
    - Pervasive use of starting over on failures without saving partial data
        - Duplicate data at the receiver UAC
        - Probably not important for small objects
    - Anecdotal reports of failures caused by SW installs and other large objects
- Future Work: Application requirements to avoid congestion collapse
    - Out of charter for the time being

Apply Congestion Collapse tests to the entire stack

- Application bench tests
  - Run a fixed application workload
  - Vary network parameters across entire space
  - Flag conditions that cause increased overhead
- Can “easily” fix egregious failures
  - E.g. restart from partial data
- However none can be totally fixed
  - Signalling (e.g. SYN and SSL) must be repeated
  - Unread data in receiver’s resequencing queue must be repeated
- We can’t use MUST

Material vs Non-material

- RFC2119 language is too “absolute”
  - These have to be strongly suggested criteria
- Is a “violation” important?
  - The term “material” comes from US legal (court) language
- Current draft language for all criteria
  - SHOULD but MUST document and score exceptions
- Also need non-absolute language for “requirements”
  - Currently using “criteria”