                Global Synchronization Protection for Packet Queues
                        draft-lauten-aqm-gsp-00.txt

Abstract

   Transmission capacity sharing TCP flows tend to synchronize among
   each other. This way rate variations of the individual flows, which
   are caused by the congestion control algorithms, do not even out. The
   effect is known as global synchronization. Large queuing buffer
   demand and large latency and jitter are the consequences. Global
   Synchronization Protection (GSP) is an extension of regular tail drop
   packet queuing schemes that prevents global synchronization. For
   large traffic aggregates the de-correlation between the individual
   flow variations reduces buffer demand and packet sojourn time by an
   order of magnitude and more. Even though quite simple, the solution
   has a theoretical rationale and is not heuristic, and it has been
   tested with a Linux implementation.

Status of this Memo

   This Internet-Draft will expire on August 13, 2014.

Copyright Notice

Table of Contents

1. Introduction

   The congestion window (CWND) of a particular TCP connection, in
   combination with the round trip time (RTT), limits the transmission
   rate of the flow, which enables adaptation of the sending rate to the
   actual network conditions, [1]. TCP uses a rather coarse congestion
   control feedback by halving the congestion window in response to
   packet loss. To fill a bottleneck link by 100% anyway, a packet
   buffer in front of the link is required. For a single TCP flow a
   buffer in the range of bottleneck capacity multiplied by the round
   trip time is required (bandwidth-delay product rule, BDP), [2]. For
   aggregated traffic of many flows the picture is not so clear.

Conservative estimations tend towards BDP of the whole aggregate, i.e. link capacity * RTT. At the other hand, rate reductions due to CWND halving are still only in the range of a particular flow rate. With the assumption of N sharing flows, this yields ideally a buffer size of only (link capacity/N)*RTT. Unfortunately this value cannot be reached in practice. It would require a uniform distribution of rate reductions by the different flows over time. In opposite, rate reductions of bottleneck sharing flows tend to synchronize among each other, which is called global synchronization. In worst case, with all flows synchronized, the buffer demand is back at BDP of the whole traffic, thus confirming the conservative estimation.

There are cases where global synchronization does not occur, in particular large number of flows (N>500), large spread of RTT between the different flows, and high frequency of flow renewals. In these cases the buffer size can be reduced to BDP/sqrt(N), which lies between the conservative and overly optimistic estimations above,[3]. Nevertheless there are still doubts, whether the absence of global synchronization is a general reliable design assumption for high capacity links, [4].

Most Active Queue Management (AQM) algorithms like RED [5] are aiming at better control of the queue size, which implies control over global synchronization. Global Synchronization Protection (GSP) goes the other way round. It suppresses the root cause of global synchronization and de-correlates the CWND variations of the competing flows, but it does not impact the behavior of a particular flow. This way it moves the buffer size demand down from conservative BDP of the whole link into the direction towards the ideal BDP of a single flow.

2. Conventions used in this document

In this document, the term "packet drop" is used for congestion notification, silently assuming that congestion marking for ECN could be equally applied.

In this document, the term "queue size" is preferably applied in number of bytes, however, the algorithm could be also applied to the number of packets, or even to the queuing delay (milliseconds).

3. Root cause of global synchronization

Global synchronization occurs in cases where a number of greedy TCP flows with comparably uniform RTT cross a tail drop queue in front of a shared transmission link. Tail drop means, a newly arriving packet is placed at the end of the queue if buffer space permits. Otherwise

it is dropped. The queue is drained from front of the queue at the
speed of the link as long as packets are available. Greedy TCP flows
means, the applications try to send as fast as possible and the flows
are not limited elsewhere in the network.

In congestion avoidance state, all senders gradually increase their
sending rate, in total exceeding the transmission capacity so that
the queue is filling up. At some point in time, a first packet is
dropped due to lack of buffer space. Ideally, the TCP flow, where the
dropped packet belongs to, reduces its sending rate, the queue
relaxes, and subsequently arriving packets can be placed in the
buffer. Senders are continuing to increase their sending rates until
the next drop, and so on.

Unfortunately, the rate reduction due to the dropped packet takes at
least one RTT to take effect at the queue entry. During that RTT
interval all senders continue to gradually increase their sending
rates, whereas the queue is still full. Further packets need to be
dropped. It can be shown analytically that for N flows with NewReno
and delayed ACK the number of drops is in the range of N/2.
(Experiments confirm this and show an even higher number with CUBIC.)
The outcome is that even though the rate reduction by one flow would
suffice, not one but half of the flows are triggered within one RTT
to reduce their sending rates - we have global synchronization.

4. Protecting queues of global synchronization

4.1. Basic algorithm

The basic algorithm is as follows: Set a threshold on queue size
below the actual buffer size. If a new packet arrives and the queue
size is above the threshold, then drop that packet. After that,
ignore any further threshold violation for a timeout interval of 1 -
3 RTT. After expiry of the timeout proceed as above.

Algorithm:

```
interval = e.g. 2 * RTT
threshold = e.g. 1/2 * buffer size
next_possible_drop = now
at any packet enqueuing do:
    if queue_size > threshold && now > next_possible_drop:
        drop this packet
        next_possible_drop = now + interval
    else
        enqueue this packet
end
```

The first dropped packet is triggering the rate reduction. During the
timeout the queue is growing further beyond the threshold until the
rate reduction takes effect at queue entry. Afterwards the queue size
should have dropped below the threshold, so that at expiry of the
timeout the threshold is typically not violated anymore. No explicit
action occurs at timeout expiry, which makes the parameter rather
insensitive to the actual traffic characteristics. Even if the
timeout interval is too short, the algorithm still reduces global
synchronization.

4.2. Interval adaptation at large flow numbers

The basic algorithm works well for moderate numbers of flows N, i.e.
in a range of 2 < N < 20. More precisely, at flow numbers N smaller
than the average CWND of one of the sharing flows. At larger numbers
the total rate increase during the timeout interval is larger than
the subsequent rate reduction by one of the flows. As consequence,
after timeout expiry the threshold is still violated, the queue is
growing further and further, and, eventually, reaches the buffer
limit and enters tail drop operation. The performance is still better
than plain tail drop and one could rely on the observation that at
large flow numbers global synchronization disappears, anyway.

Alternatively the initial timeout interval can be reduced, depending
on the actual traffic, in a way, where not just once, but twice, or
even more times per RTT the timeout expires. The adaptation criterion
is the proportion of time above and below threshold. In regular
operation according to the basic algorithm, the queue is most of the
time below the threshold. If, however, the queue is more frequently
above than below threshold, the interval should be reduced until
equilibrium is reached. In this condition the queue is oscillating

around the threshold, quite similar to other AQM schemes like e.g.
PED.

Algorithm:

```
at any packet enqueueing do:
    theta = cumulative time {above - below} threshold
    k = max(1, gamma * theta)
    interval = initial_interval/k
end
```

Gamma is a scaling parameter that controls the sensitivity of
adaptation.

4.3. Interval adaptation at small RTT

The RTT is not known exactly but there should be at least a rough
idea on the range of RTT for setting up the timeout interval. If this
estimation is much too large, a similar situation occurs like in the
large flow numbers case. The total rate increase during the timeout
interval (which turns out to be multiple RTTs) is larger than the
subsequent rate reduction by one flow. The adaptation rule is the
same as for large flow numbers.

4.4. Threshold adaptation

Tbc

4.5. Sanity checks and special cases

An additional rule can be introduced that prevents large packet
bursts from immediately triggering the drop: Restart the timeout not
only after a packet drop but also whenever a packet is arriving at an
empty queue.

5. Security Considerations

Global synchronization is a particular problem of many elastic flows
sharing a bottleneck. GSP is there to prevent this. But it does not
protect of unresponsive flows. If the congestion notification
according to section 4.1. randomly hits an unresponsive flow then the
expected rate reduction within the timeout interval might simply not
happen, which postpones the notification by one timeout interval. In
extreme cases, with a large amount of unresponsive traffic, GSP
behaves like plain tail drop.

6. IANA Considerations

   There are no actions for IANA.

7. Conclusions

   tbc

8. References

8.1. Normative References

8.2. Informative References

   [1]   Van Jacobson, Congestion avoidance and control, Proc. SIGCOMM
          '88, 1988

   [2]   M. Mathis, J. Semke, J. Mahdavi, and T. Ott, The Macroscopic
          Behavior of the TCP Congestion Avoidance Algorithm, Comput.
          Commun. Rev., 27.3, 1997, pp. 67-82.

   [3]   G. Appenzeller, I. Keslassy, and N. McKeown, Sizing router
          buffers, Proc. ACM SIGCOMM '04, 2004.

   [4]   G. Vu-Brugier, R. S. Stanojevic, D. J. Leith, R. N. Shorten, A
          critique of recently proposed buffer-sizing strategies, ACM
          SIGCOMM Computer Communication Review, 37.1, 2007

   [5]   S. Floyd, Van Jacobsen, Random Early Detection Gateways for
          Congestion Avoidance, IEEE/ACM Trans. Networking, 1.4, 1993

   [6]   K. Nichols, Van Jacobson, "Controlling Queue Delay", ACM Queue
          - Networks, 2012

   [7]   R. Pan, et al., PIE: A lightweight control scheme to address
          the bufferbloat problem, IETF, draft-pan-tsvwg-pie-00, 2012

9. Acknowledgments

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

Authors' Addresses

    Wolfram Lautenschlaeger
    Alcatel-Lucent
    Bell Labs
    Lorenzstrasse 10
    70435 Stuttgart
    Germany

    Email: Wolfram.Lautenschlaeger@alcatel-lucent.com