Active Queue Management
and Packet Scheduling (aqm)
Internet Draft
Intended status:
Expires: September 2015 March 9, 2015

Wolfram Lautenschlaeger
Alcatel-Lucent
Bell Labs

Global Synchronization Protection for Packet Queues
draft-lauten-aqm-gsp-02.txt

Abstract

The congestion avoidance processes of several transmission capacity sharing TCP flows tend to be synchronized among each other, so that the rate variations of the individual flows do not compensate. In contrary, they accumulate into large variations of the whole aggregate. 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 background and is not heuristic. It has been tested with a Linux kernel implementation and shows equivalent performance as other relevant AQM schemes.

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

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 are aiming at better control of the queue size (RED [5]) or the queue delay (CoDel [6], PIE [7]), 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 try to 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 only one of the competing flows.

Experiments show that the stabilizing effect of GSP is equivalent to that of the other AQMs. It is a simple extension to plain tail drop queues. The basic algorithm is memoryless and does not need artificial randomization. Particularly for small numbers of flows it performs better than randomized AQMs.

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. Greedy TCP means, the flow is probing the available capacity on this particular link and is not limited elsewhere (up- or downstream). 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 head of the queue at the speed of the link as long as packets are available.

In congestion avoidance state, all senders gradually increase their sending rate, which is, after a while, exceeding the link capacity so that the queue in front of the link 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, its size goes down, and subsequently arriving packets again can be placed in the buffer. Senders continue to increase their sending rates until the next drop, and so on.

Unfortunately not one, but several packets get dropped in such incident for following reason: The rate reduction due to the first dropped packet needs 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 as much as half of the flows are triggered within one RTT to reduce their sending rates – we have global synchronization. A more detailed analytical and experimental investigation of the effect can be found in [8].

4. Protecting queues of global synchronization

4.1. Basic algorithm

The basic algorithm works 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 immediately 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:

```
initialization:
    interval = e.g. 2 * RTT
    threshold = e.g. 1/2 * buffer size
    timeout_expiry = now(), with now() returning the current time

at any packet arrival do:
    if queue_size > threshold && now() > timout_expiry:
        drop this packet
        timeout_expiry = now() + interval
    else
        enqueue this packet
```

The first dropped packet is triggering the rate reduction by one of the end points. 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 timeout expiry 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 1 < 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, periodically dropping during times above the threshold, quite similar like PED [9].

Algorithm:

```
initialization:
       tau = a preset parameter controlling the adaptation speed;
             e.g. 500 milliseconds or 5 * initial_interval
       initial_interval - the preset timeout interval as in the basic
           algorithm
       cumTime = 0; the cumulative time above/below threshold
       state = CLEAR; the recent overflow state of the queue

   at any packet arrival do:
       if the packet would overflow the queue (hard tail drop):
           state = OVERFLOW
       if state == OVERFLOW && queue is empty:
           state = DRAIN
       if state == DRAIN && queue_size > threshold:
           state = CLEAR

       update the cumulative time cumTime:
           account by twice the duration for queue episodes
               that are entirely ABOVE the threshold
           if status == CLEAR:
               account by negated duration for queue episodes
                   that are entirely BELOW the threshold

       clamp the cumulative time:
           cumTime = max(cumTime, 0)
           cumTime = min(cumTime, some sanity limit
                         of several minutes)
```

```
        calculate timeout interval (to be used at next drop decision):
            interval = initial_interval/(1+cumTime/tau)
```

The adaptation heuristics works as follows: The basic GSP algorithm executes single drops per threshold violation as long as the cumulative time (cumTime) is clamped at zero. As soon as the cumulative time gets positive, the adaptation algorithm implements an integral controller for the drop rate that the basic GSP algorithm executes during times of threshold violation. The transition between both operating conditions is smooth. The adaptation speed is controlled by the parameter tau.

Plain adaptation by cumulative times above / below threshold might latch up in circumstances where abrupt traffic changes cause massive buffer overflows. To avoid this, after a hard buffer overflow the accounting for times below threshold is suspended until the queue performed a full cycle of down to empty and back above threshold.

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, section 4.2.

4.4. 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.

```
at any packet arrival do:
    if queue is empty:
        timeout_expiry = now() + interval
```

5. Delay based operation

5.1. Queue delay vs. queue size

Recent new AQM proposals ([6], [7]) are focusing on queue delay rather than on queue size in bytes. One reason for this move is that ideally the steady state queue oscillation depends only on the RTT and on the number of sharing TCP flows - if measured in delay. The oscillation sets the minimum queue size for 100% link utilization. A larger queue creates only unnecessary delay (standing queue). If measured in bytes, however, the queue oscillation depends additionally on the link capacity. (This is where the bandwidth delay product rule comes from.)

Obviously it is preferable to stabilize the delay instead of the size. It eliminates the interface rate from the parameter list, which is particularly welcome in circumstances with unknown or variable drain rate. Such situations are typical for low priority queues in front of a priority scheduler and generally in wireless scenarios.

5.2. Delay based GSP

In section 4. we silently assumed queue size in bytes. However, the algorithm can be equally applied to the queue delay (packet sojourn time). In this case the threshold has to be in milliseconds, whereas the empty queue condition remains the same as before.

While the queue size in bytes or packets is typically maintained by ordinary queue implementations, obtaining the queue delay requires additional effort. Two solutions are available and both are applicable to GSP: Time stamping of packets like in CoDel [6] or estimating the drain rate for a translation of size into delay like in PIE [7].

Algorithm (time stamping):

```
at any arrival of a packet p do:
    p.time = now()

at any departure of a packet p do:
    queue_delay = now() - p.time
```

The basic algorithm of section 4.1. rephrased to delay based operation:

```
at any packet arrival do:
    if queue_delay > threshold && now() > timout_expiry:
        drop this packet
        timeout_expiry = now() + interval
    else
        enqueue this packet
```

Please note that packe_delay is a per queue variable, not per packet, i.e. the drop decision at enqueuing (tail drop) depends on the delay that another, most recently dequeued packet experienced. This approach is justified by the inherent inertance of the queue itself.

6. 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.

7. IANA Considerations

There are no actions for IANA.

8. Conclusions

tbc

9. References

9.1. Normative References

9.2. 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, P. Natarajan, C. Piglione, M.S. Prabhu, V. Subramanian, F. Baker, B. VerSteeg, PIE: A lightweight control scheme to address the bufferbloat problem, 14th High Performance Switching and Routing (HPSR), 2013 IEEE

[8] W. Lautenschlaeger, A deterministic TCP bandwidth sharing model, 2014, online http://arxiv.org/pdf/1404.4173v1

[9] A. Francini, Beyond RED: Periodic Early Detection for On-Chip Buffer Memories in Network Elements, Proc. IEEE High-Performance Switching and Routing Conference (HPSR 2011), Cartagena, Spain, July 4-6, 2011

10. 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