

TCPM Working Group
Internet-Draft
Obsoletes: [2861](#) (if approved)
Intended status: Experimental
Expires: December 13, 2015

G. Fairhurst
A. Sathiaselan
R. Secchi
University of Aberdeen
June 11, 2015

Updating TCP to support Rate-Limited Traffic
draft-ietf-tcpm-newcwv-12

Abstract

This document provides a mechanism to address issues that arise when TCP is used to support traffic that exhibits periods where the sending rate is limited by the application rather than the congestion window. It provides an experimental update to TCP that allows a TCP sender to restart quickly following a rate-limited interval. This method is expected to benefit applications that send rate-limited traffic using TCP, while also providing an appropriate response if congestion is experienced.

It also evaluates the Experimental specification of TCP Congestion Window Validation, CWV, defined in [RFC 2861](#), and concludes that [RFC 2861](#) sought to address important issues, but failed to deliver a widely used solution. This document therefore recommends that the status of [RFC 2861](#) is moved from Experimental to Historic, and that it is replaced by the current specification.

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 <http://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 December 13, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
1.1.	Implementation of new CWV	5
1.2.	Standards Status of this Document	5
2.	Reviewing experience with TCP-CWV	5
3.	Terminology	7
4.1.	Initialisation	8
4.2.	Estimating the validated capacity supported by a path . .	8
4.3.	Preserving cwnd during a rate-limited period.	10
4.4.	TCP congestion control during the non-validated phase . .	11
4.4.1.	Response to congestion in the non-validated phase . .	12
4.4.2.	Sender burst control during the non-validated phase .	13
4.4.3.	Adjustment at the end of the Non-Validated Period (NVP)	14
4.5.	Examples of Implementation	15
4.5.1.	Implementing the pipeACK measurement	15
4.5.2.	Measurement of the NVP and pipeACK samples	16
4.5.3.	Implementing detection of the cwnd-limited condition	16
5.	Determining a safe period to preserve cwnd	17
6.	Security Considerations	18
7.	IANA Considerations	18
8.	Acknowledgments	18
9.	Author Notes	18
9.1.	Other related work	18
10.	Revision notes	20
11.	References	24
11.1.	Normative References	24
11.2.	Informative References	25
	Authors' Addresses	25

1. Introduction

TCP is used for traffic with a range of application behaviours. The TCP congestion window (cwnd) controls the number of unacknowledged packets/bytes that a TCP flow may have in the network at any time, a value known as the FlightSize [[RFC5681](#)]. FlightSize is a measure of the volume of data that is in flight. A bulk application will always have data available to transmit. The rate at which it sends is therefore limited by the maximum permitted by the receiver advertised window and the sender congestion window (cwnd). The FlightSize of a bulk flow increases with the cwnd, and tracks the volume of data acknowledged in the last Round Trip Time (RTT).

In contrast, a rate-limited application will experience periods when the sender is either idle or is unable to send at the maximum rate permitted by the cwnd. In this case, the volume of data sent (FlightSize) can change significantly from one RTT to another, and can be much less than the cwnd. Hence, it is possible that the FlightSize could significantly exceed the recently used capacity. The update in this document targets the operation of TCP in such rate-limited cases.

Standard TCP [[RFC5681](#)] states that a TCP sender SHOULD set cwnd to no more than the Restart Window (RW) before beginning transmission, if the TCP sender has not sent data in an interval exceeding the retransmission timeout, i.e., when an application becomes idle. [[RFC2861](#)] noted that this TCP behaviour was not always observed in current implementations. Experiments [[Bis08](#)] confirm this to still be the case.

Congestion Window Validation, CWV, introduced the terminology of "application limited periods". [RFC2861](#) describes any time that an application limits the sending rate, rather than being limited by the transport, as "rate-limited". This update improves support for applications that vary their transmission rate, either with (short) idle periods between transmission or by changing the rate at which the application sends. These applications are characterised by the TCP FlightSize often being less than cwnd. Many Internet applications exhibit this behaviour, including web browsing, http-based adaptive streaming, applications that support query/response type protocols, network file sharing, and live video transmission. Many such applications currently avoid using long-lived (persistent) TCP connections (e.g., [[RFC2616](#)] servers typically support persistent HTTP connections, but do not enable this by default). Such applications often instead either use a succession of short TCP transfers or use UDP.

Standard TCP does not impose additional restrictions on the growth of the congestion window when a TCP sender is unable to send at the maximum rate allowed by the cwnd. In this case, the rate-limited sender may grow a cwnd far beyond that corresponding to the current transmit rate, resulting in a value that does not reflect current information about the state of the network path the flow is using. Use of such an invalid cwnd may result in reduced application performance and/or could significantly contribute to network congestion.

[RFC2861] proposed a solution to these issues in an experimental method known as CWV. CWV was intended to help reduce cases where TCP accumulated an invalid (inappropriately large) cwnd. The use and drawbacks of using the CWV algorithm in [RFC 2861](#) with an application are discussed in [Section 2](#).

[Section 3](#) defines relevant terminology.

[Section 4](#) specifies an alternative to CWV that seeks to address the same issues, but does so in a way that is expected to mitigate the impact on an application that varies its sending rate. The updated method applies to the rate-limited conditions (including both application-limited and idle senders).

The goals of this update are:

- o To not change the behaviour of a TCP sender that performs bulk transfers that fully use the cwnd.
- o To provide a method that co-exists with Standard TCP and other flows that use this updated method.
- o To reduce transfer latency for applications that change their rate over short intervals of time.
- o To avoid a TCP sender growing a large "non-validated" cwnd, when it has not recently sent using this cwnd.
- o To remove the incentive for ad-hoc application or network stack methods (such as "padding") solely to maintain a large cwnd for future transmission.
- o To incentivise the use of long-lived connections, rather than a succession of short-lived flows, benefiting both the flows and other flows sharing the network path when actual congestion is encountered.

[Section 5](#) describes the rationale for selecting the safe period to preserve the cwnd.

[1.1.](#) Implementation of new CWV

The method specified in [Section 4](#) of this document is a sender-side only change to the the TCP congestion control behaviour of TCP.

The method creates a new protocol state, and requires a sender to determine when the cwnd is validated or non-validated to control the entry and exit from this state [Section 4.3](#). It defines how a TCP sender manages the growth of the cwnd using the set of rules defined in [Section 4](#).

Implementation of this specification requires an implementor to define a method to measure the available capacity using the pipeACK samples. The details of this measurement are implementation-specific. An example is provided in [Section 4.5.1](#), but other methods are permitted. A sender also needs to provide a method to determine when it becomes cwnd-limited. Implementation of this may require consideration of other TCP methods (see [Section 4.5.3](#)).

A sender is also recommended to provide a method that controls the maximum burst size, [Section 4.4.2](#). However, implementors are allowed flexibility in how this method is implemented and the choice of an appropriate method is expected to depend on the way in which the sender stack implements other TCP methods (such as TCP Segment Offload, TSO).

[1.2.](#) Standards Status of this Document

The document updates and obsoletes the methods described in [\[RFC2861\]](#). It recommends a set of mechanisms, including the use of pacing during a non-validated period. The updated mechanisms are intended to have a less aggressive congestion impact than would be exhibited by a standard TCP sender.

The specification in this draft is classified as "Experimental" pending experience with deployed implementations of the methods.

[2.](#) Reviewing experience with TCP-CWV

[\[RFC2861\]](#) described a simple modification to the TCP congestion control algorithm that decayed the cwnd after the transition to a "sufficiently-long" idle period. This used the slow-start threshold (ssthresh) to save information about the previous value of the congestion window. The approach relaxed the standard TCP behaviour [\[RFC5681\]](#) for an idle session, intended to improve application

performance. CWV also modified the behaviour when a sender transmitted at a rate less than allowed by cwnd.

[RFC2861] proposed two set of responses, one after an "application-limited" and one after an "idle period". Although this distinction was argued, in practice differentiating the two conditions was found problematic in actual networks (e.g., [Bis10]). While this offers predictable performance for long on-off periods ($\gg 1$ RTT), or slowly varying rate-based traffic, the performance could be unpredictable for variable-rate traffic and depended both upon whether an accurate RTT had been obtained and the pattern of application traffic relative to the measured RTT.

Many applications can and often do vary their transmission over a wide range of rates. Using [RFC2861] such applications often experienced varying performance, which made it hard for application developers to predict the TCP latency even when using a path with stable network characteristics. We argue that an attempt to classify application behaviour as application-limited or idle is problematic and also inappropriate. This document therefore explicitly avoids trying to differentiate these two cases, instead treating all rate-limited traffic uniformly.

[RFC2861] has been implemented in some mainstream operating systems as the default behaviour [Bis08]. Analysis (e.g., [Bis10] [Fai12]) has shown that a TCP sender using CWV is able to use available capacity on a shared path after an idle period. This can benefit variable-rate applications, especially over long delay paths, when compared to the slow-start restart specified by standard TCP. However, CWV would only benefit an application if the idle period were less than several Retransmission Time Out (RTO) intervals [RFC6298], since the behaviour would otherwise be the same as for standard TCP, which resets the cwnd to the TCP Restart Window after this period.

To enable better performance for variable-rate applications with TCP, some operating systems have chosen to support non-standard methods, or applications have resorted to "padding" streams by sending dummy data to maintain their sending rate when they have no data to transmit. Although transmitting redundant data across a network path provides good evidence that the path can sustain data at the offered rate, padding also consumes network capacity and reduces the opportunity for congestion-free statistical multiplexing. For variable-rate flows, the benefits of statistical multiplexing can be significant and it is therefore a goal to find a viable alternative to padding streams.

Experience with [[RFC2861](#)] suggests that although the CWV method benefited the network in a rate-limited scenario (reducing the probability of network congestion), the behaviour was too conservative for many common rate-limited applications. This mechanism did not therefore offer the desirable increase in application performance for rate-limited applications and it is unclear whether applications actually use this mechanism in the general Internet.

It is therefore concluded that CWV, as defined in [[RFC2861](#)], was often a poor solution for many rate-limited applications. It had the correct motivation, but had the wrong approach to solving this problem.

3. Terminology

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 [[RFC2119](#)].

The document assumes familiarity with the terminology of TCP congestion control [[RFC5681](#)].

The following additional terminology is used in this document:

cwnd-limited: A TCP flow that has sent the maximum number of segments permitted by the cwnd, where the application utilises the allowed sending rate (see [Section 4.5.3](#)).

pipeACK sample: A measure of the volume of data acknowledged by the network within an RTT.

pipeACK variable: A variable that measures the available capacity using the set of pipeACK samples.

pipeACK Sampling Period: The maximum period that a measured pipeACK sample may influence the pipeACK variable.

Non-validated phase: The phase where the cwnd reflects a previous measurement of the available path capacity.

Non-validated period, NVP: The maximum period for which cwnd is preserved in the non-validated phase.

Rate-limited: A TCP flow that does not consume more than one half of cwnd, and hence operates in the non-validated phase. This includes periods when an application is either idle or chooses to send at a rate less than the maximum permitted by the cwnd.

Validated phase: The phase where the cwnd reflects a current estimate of the available path capacity.

4. A New Congestion Window Validation method

This section proposes an update to the TCP congestion control behaviour during a rate-limited interval. This new method intentionally does not differentiate between times when the sender has become idle or chooses to send at a rate less than the maximum allowed by the cwnd.

The period where actual usage is less than allowed by cwnd, is named the non-validated phase. The update allows an application in the non-validated phase to resume transmission at a previous rate without incurring the delay of slow-start. However, if the TCP sender experiences congestion using the preserved cwnd, it is required to immediately reset the cwnd to an appropriate value specified by the method. If a sender does not take advantage of the preserved cwnd within the Non-validated period, NVP, the value of cwnd is reduced, ensuring the value better reflects the capacity that was recently actually used.

It is expected that this update will satisfy the requirements of many rate-limited applications and at the same time provide an appropriate method for use in the Internet. New-CWV reduces this incentive for an application to send "padding" data simply to keep transport congestion state.

The method is specified in following subsections and is expected to encourage applications and TCP stacks to use standards-based congestion control methods. It may also encourage the use of long-lived connections where this offers benefit (such as persistent http).

4.1. Initialisation

A sender starts a TCP connection in the validated phase and initialises the pipeACK variable to the "undefined" value. This value inhibits use of the value in cwnd calculations.

4.2. Estimating the validated capacity supported by a path

[RFC6675] defines a variable, FlightSize, that indicates the instantaneous amount of data that has been sent, but not cumulatively acknowledged. In this method a new variable "pipeACK" is introduced to measure the acknowledged size of the network pipe. This is used to determine if the sender has validated the cwnd. pipeACK differs

from FlightSize in that it is evaluated over a window of acknowledged data, rather than reflecting the amount of data outstanding.

A sender determines a pipeACK sample by measuring the volume of data that was acknowledged by the network over the period of a measured Round Trip Time (RTT). Using the variables defined in [\[RFC6675\]](#), a value could be measured by caching the value of HighACK and after one RTT measuring the difference between the cached HighACK value and the current HighACK value. A sender MAY count TCP DupACKs that acknowledge new data when collecting the pipeACK sample. Other equivalent methods may be used.

A sender is not required to continuously update the pipeACK variable after each received ACK, but SHOULD perform a pipeACK sample at least once per RTT when it has sent unacknowledged segments.

The pipeACK variable MAY consider multiple pipeACK samples over the pipeACK Sampling Period. The value of the pipeACK variable MUST NOT exceed the maximum (highest value) within the sampling period. This specification defines the pipeACK Sampling Period as $\text{Max}(3 \cdot \text{RTT}, 1 \text{ second})$. This period enables a sender to compensate for large fluctuations in the sending rate, where there may be pauses in transmission, and allows the pipeACK variable to reflect the largest recently measured pipeACK sample.

When no measurements are available (e.g., a sender that has just started transmission or immediately after loss recovery), the pipeACK variable is set to the "undefined value". This value is used to inhibit entering the non-validated phase until the first new measurement of a pipeACK sample. ([Section 4.5](#) provides examples of implementation.)

The pipeACK variable MUST NOT be updated during TCP Fast Recovery. That is, the sender stops collecting pipeACK samples during loss recovery. The method RECOMMENDS that the TCP SACK option [\[RFC2018\]](#) is enabled and the method defined in [\[RFC6675\]](#) is used to recover missing segments. This allows the sender to more accurately determine the number of missing bytes during the loss recovery phase, and using this method will result in a more appropriate cwnd following loss.

NOTE: The use of pipeACK rather than FlightSize can change the behaviour of a TCP when a sender does not always have data available to send. One example arises when there is a pause in transmission after sending a sequence of many packets, and the sender experiences loss at or near the end of its transmission sequence. In this case, the TCP flow may have used a significant amount of capacity just prior to the loss (which would be reflected in the volume of data

acknowledged, recorded in the pipeACK variable), but at the actual time of loss the number of unacknowledged packets in flight (at the end of the sequence) may be small, i.e., there is a small FlightSize. After loss recovery, the sender resets its congestion control state.

[Fai12] explored the benefits of different responses to congestion for application-limited streams. If the response is based only on the Loss FlightSize, the sender would assign a small cwnd and ssthresh, based only on the volume of data sent after the loss. When the sender next starts to transmit it can incur many RTTs of delay in slow start before it reacquires its previous rate. When the pipeACK value is also used to calculate the cwnd and ssthresh (as specified in this update in [Section 4.4.1](#)), the sender can use a value that also reflects the recently used capacity before the loss. This prevents a variable-rate application from being unduly penalised. When the sender resumes, it starts at one half its previous rate, similar to the behaviour of a bulk TCP flow [Hos15]. To ensure an appropriate reaction to on-going congestion, this method requires that the pipeACK variable is reset after it is used in this way.

4.3. Preserving cwnd during a rate-limited period.

The updated method creates a new TCP sender phase that captures whether the cwnd reflects a validated or non-validated value. The phases are defined as:

- o Validated phase: $\text{pipeACK} \geq (1/2) * \text{cwnd}$, or pipeACK is undefined (i.e., at the start or directly after loss recovery). This is the normal phase, where cwnd is expected to be an approximate indication of the capacity currently available along the network path, and the standard methods are used to increase cwnd (currently [[RFC5681](#)]).
- o Non-validated phase: $\text{pipeACK} < (1/2) * \text{cwnd}$. This is the phase where the cwnd has a value based on a previous measurement of the available capacity, and the usage of this capacity has not been validated in the pipeACK Sampling Period. That is, when it is not known whether the cwnd reflects the currently available capacity along the network path. The mechanisms to be used in this phase seek to determine a safe value for cwnd and an appropriate reaction to congestion.

Note: A threshold is needed to determine whether a sender is in the validated or non-validated phase. A standard TCP sender in slow-start is permitted to double its FlightSize from one RTT to the next. This motivated the choice of a threshold value of 1/2. This threshold ensures a sender does not further increase the cwnd as long as the FlightSize is less than $(1/2 * \text{cwnd})$. Furthermore, a sender

with a FlightSize less than $(1/2 * cwnd)$ may in the next RTT be permitted by the cwnd to send at a rate that more than doubles the FlightSize, and hence this case needs to be regarded as non-validated and a sender therefore needs to employ additional mechanisms while in this phase.

4.4. TCP congestion control during the non-validated phase

A TCP sender implementing this specification MUST enter the non-validated phase when the pipeACK is less than $(1/2) * cwnd$.

A TCP sender that enters the non-validated phase preserves the cwnd (i.e., the cwnd only increases after a sender fully uses the cwnd in this phase, otherwise the cwnd neither grows nor reduces). The phase is concluded when the sender transmits sufficient data so that $pipeACK > (1/2) * cwnd$ (i.e., the sender is no longer rate-limited), or when the sender receives an indication of congestion.

After a fixed period of time (the non-validated period, NVP), the sender adjusts the cwnd [Section 4.4.3](#)). The NVP SHOULD NOT exceed 5 minutes. [Section 5](#) discusses the rationale for choosing a safe value for this period.

The behaviour in the non-validated phase is specified as:

- o A sender determines whether to increase the cwnd based upon whether it is cwnd-limited (see [Section 4.5.3](#)):
 - * A sender that is cwnd-limited MAY use the standard TCP method to increase cwnd (i.e., a TCP sender that fully utilises the cwnd is permitted to increase cwnd each received ACK using standard methods).
 - * A sender that is not cwnd-limited MUST NOT increase the cwnd when ACK packets are received in this phase (i.e., needs to avoid growing the cwnd when it has not recently sent using the current size of cwnd).
- o If the sender receives an indication of congestion while in the non-validated phase (i.e., detects loss), the sender MUST exit the non-validated phase (reducing the cwnd as defined in [Section 4.4.1](#)).
- o If the Retransmission Time Out (RTO) expires while in the non-validated phase, the sender MUST exit the non-validated phase. It then resumes using the standard TCP RTO mechanism [[RFC5681](#)].

- o A sender with a pipeACK variable greater than $(1/2)*cwnd$ SHOULD enter the validated phase. (A rate-limited sender will not normally be impacted by whether it is in a validated or non-validated phase, since it will normally not increase FlightSize to use the entire cwnd. However, a change to the validated phase will release the sender from constraints on the growth of cwnd, and result in using the standard congestion response.)

The cwnd-limited behaviour may be triggered during a transient condition that occurs when a sender is in the non-validated phase and receives an ACK that acknowledges received data, the cwnd was fully utilised, and more data is awaiting transmission than may be sent with the current cwnd. The sender MAY then use the standard method to increase the cwnd. (Note, if the sender succeeds in sending these new segments, the updated cwnd and pipeACK variables will eventually result in a transition to the validated phase.)

4.4.1. Response to congestion in the non-validated phase

Reception of congestion feedback while in the non-validated phase is interpreted as an indication that it was inappropriate for the sender to use the preserved cwnd. The sender is therefore required to quickly reduce the rate to avoid further congestion. Since the cwnd does not have a validated value, a new cwnd value needs to be selected based on the utilised rate.

A sender that detects a packet-drop MUST record the current FlightSize in the variable LossFlightSize and MUST calculate a safe cwnd for loss recovery using the method below:

$$cwnd = (\text{Max}(\text{pipeACK}, \text{LossFlightSize}))/2.$$

The pipeACK value is not updated during loss recovery [Section 4.2](#). If there is a valid pipeACK value, the new cwnd is adjusted to reflect that a non-validated cwnd may be larger than the actual FlightSize, or recently used FlightSize (recorded in pipeACK). The updated cwnd therefore prevents overshoot by a sender significantly increasing its transmission rate during the recovery period.

At the end of the recovery phase, the TCP sender MUST reset the cwnd using the method below:

$$cwnd = (\text{Max}(\text{pipeACK}, \text{LossFlightSize}) - R)/2.$$

Where R is the volume of data that was successfully retransmitted during the recovery phase. This corresponds to segments retransmitted and considered lost by the pipe estimation algorithm at

the end of recovery. It does not include the additional cost of multiple retransmission of the same data. The loss of segments indicates that the path capacity was exceeded by at least R , and hence the calculated $cwnd$ is reduced by at least R before the window is halved.

The calculated $cwnd$ value MUST NOT be reduced below 1 TCP Maximum Segment Size (MSS).

After completing the loss recovery phase, the sender MUST re-initialise the `pipeACK` variable to the "undefined" value. This ensures that standard TCP methods are used immediately after completing loss recovery until a new `pipeACK` value can be determined.

The `ssthresh` is adjusted using the standard TCP method (Step 6 in [Section 3.2 of RFC 5681](#) assigns the `ssthresh` a value equal to $cwnd$ at the end of the loss recovery).

Note: The adjustment by reducing $cwnd$ by the volume of data not sent (R) follows the method proposed for Jump Start [[Liu07](#)]. The inclusion of the term R makes the adjustment more conservative than standard TCP. This is required, since a sender in the non-validated state may increase the rate more than a standard TCP would have done relative to what was sent in the last RTT (i.e., more than doubled the number of segments in flight relative to what it sent in the last RTT). The additional reduction after congestion is beneficial when the `LossFlightSize` has significantly overshot the available path capacity incurring significant loss (e.g., following a change of path characteristics or when additional traffic has taken a larger share of the network bottleneck during a period when the sender transmits less).

Note: The `pipeACK` value is only valid during a non-validated phase, and therefore this does not exceed $cwnd/2$. If `LossFlightSize` and R were small, then this can result in the final $cwnd$ after loss recovery being at most one quarter of the $cwnd$ on detection of congestion. This reduction is conservative, and `pipeACK` is then reset to undefined, hence $cwnd$ updates after a congestion event do not depend upon the `pipeACK` history before congestion was detected.

4.4.2. Sender burst control during the non-validated phase

TCP congestion control allows a sender to accumulate a $cwnd$ that would allow it to send a burst of segments with a total size up to the difference between the `FlightSize` and $cwnd$. Such bursts can impact other flows that share a network bottleneck and/or may induce congestion when buffering is limited.

Various methods have been proposed to control the sender burstiness [[Hug01](#)], [[A1105](#)]. For example, TCP can limit the number of new segments it sends per received ACK. This is effective when a flow of ACKs is received, but can not be used to control a sender that has not send appreciable data in the previous RTT [[A1105](#)].

This document recommends using a method to avoid line-rate bursts after an idle or rate-limited interval when there is less reliable information about the capacity of the network path: A TCP sender in the non-validated phase SHOULD control the maximum burst size, e.g., using a rate-based pacing algorithm in which a sender paces out the cwnd over its estimate of the RTT, or some other method, to prevent many segments being transmitted contiguously at line-rate. The most appropriate method(s) to implement pacing depend on the design of the TCP/IP stack, speed of interface and whether hardware support (such as TCP Segment Offload, TSO) is used. The present document does not recommend any specific method.

[4.4.3](#). Adjustment at the end of the Non-Validated Period (NVP)

An application that remains in the non-validated phase for a period greater than the NVP is required to adjust its congestion control state. If the sender exits the non-validated phase after this period, it MUST update the ssthresh:

$$\text{ssthresh} = \max(\text{ssthresh}, 3 * \text{cwnd} / 4).$$

(This adjustment of ssthresh ensures that the sender records that it has safely sustained the present rate. The change is beneficial to rate-limited flows that encounter occasional congestion, and could otherwise suffer an unwanted additional delay in recovering the sending rate.)

The sender MUST then update cwnd to be not greater than:

$$\text{cwnd} = \max((1/2) * \text{cwnd}, \text{IW}).$$

Where IW is the appropriate TCP initial window, used by the TCP sender (e.g., [[RFC5681](#)]).

Note: These cwnd and ssthresh adjustments cause the sender to enter slow-start (since $\text{ssthresh} > \text{cwnd}$). This adjustment ensures that the sender responds conservatively after remaining in the non-validated phase for more than the non-validated period. In this case, it reduces the cwnd by a factor of two from the preserved value. This adjustment is helpful when flows accumulate but do not use a large cwnd, and seeks to mitigate the impact when these flows later resume

transmission. This could for instance mitigate the impact if multiple high-rate application flows were to become idle over an extended period of time and then were simultaneously awakened by an external event.

4.5. Examples of Implementation

This section provides informative examples of implementation methods. Implementations may choose to use other methods that comply with the normative requirements.

4.5.1. Implementing the pipeACK measurement

A pipeACK sample may be measured once each RTT. This reduces the sender processing burden for calculating after each acknowledgement and also reduces storage requirements at the sender.

Since application behaviour can be bursty using CWV, it may be desirable to implement a maximum filter to accumulate the measured values so that the pipeACK variable records the largest pipeACK sample within the pipeACK Sampling Period. One simple way to implement this is to divide the pipeACK Sampling Period into several (e.g., 5) equal length measurement periods. The sender then records the start time for each measurement period and the highest measured pipeACK sample. At the end of the measurement period, any measurement(s) that are older than the pipeACK Sampling Period are discarded. The pipeACK variable is then assigned the largest of the set of the highest measured values.

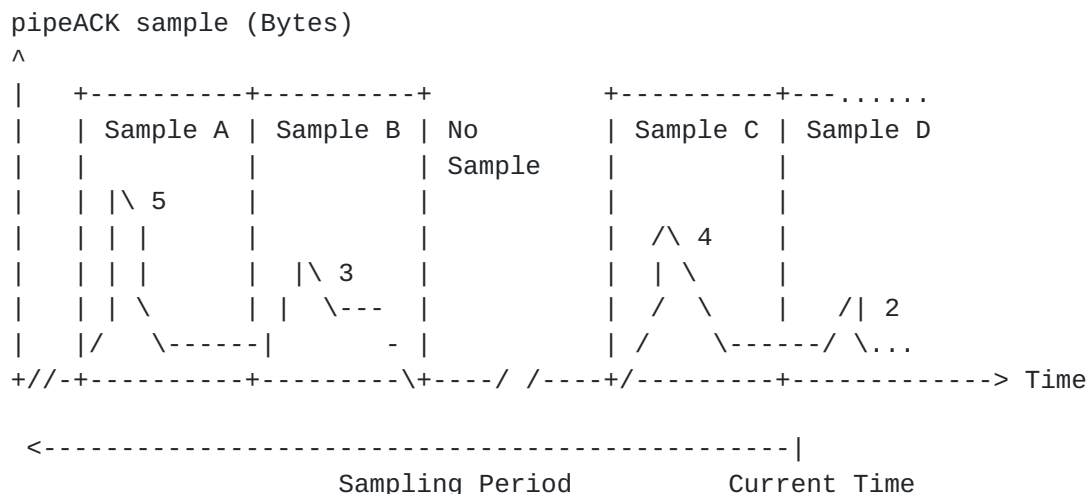


Figure 1: Example of measuring pipeACK samples

Figure 1 shows an example of how measurement samples may be collected. At the time represented by the figure new samples are being accumulated into sample D. Three previous samples also fall within the pipeACK Sampling Period: A, B, and C. There was also a period of inactivity between samples B and C during which no measurements were taken (because no new data segments were acknowledged). The current value of the pipeACK variable will be 5, the maximum across all samples. During this period, the pipeACK samples may be regarded as zero, and hence do not contribute to the calculated pipeACK value.

After one further measurement period, Sample A will be discarded, since it then is older than the pipeACK Sampling Period and the pipeACK variable will be recalculated, Its value will be the larger of Sample C or the final value accumulated in Sample D.

4.5.2. Measurement of the NVP and pipeACK samples

The mechanism requires a number of measurements of time. These measurements could be implemented using protocol timers, but do not necessarily require a new timer to be implemented. Avoiding the use of dedicated timers can save operating system resources, especially when there may be large numbers of TCP flows.

The NVP could be measured by recording a timestamp when the sender enters the non-validated phase. Each time a sender transmits a new segment, this timestamp can be used to determine if the NVP has expired. If the measured period exceeds the NVP, the sender can then take into account how many units of the NVP have passed and make one reduction (defined in [Section 4.4.3](#)) for each NVP.

Similarly, the time measurements for collecting pipeACK samples and determining the Sampling Period could be derived by using a timestamp to record when each sample was measured, and to use this to calculate how much time has passed when each new ACK is received.

4.5.3. Implementing detection of the cwnd-limited condition

A sender needs to implement a method that detects the cwnd-limited condition (see [Section 4.4](#)). This detects a condition where a sender in the non-validated phase receives an ACK, but the size of cwnd prevents sending more new data.

In simple terms, this condition is true only when the FlightSize of a TCP sender is equal to or larger than the current cwnd. However, an implementation also needs to consider constraints on the way in which the cwnd variable can be used, for instance implementations need to support other TCP methods such as the Nagle Algorithm and TCP Segment

Offload (TSO) that also use cwnd to control transmission. These other methods can result in a sender becoming cwnd-limited when the cwnd is nearly, rather than completely, equal to the FlightSize.

5. Determining a safe period to preserve cwnd

This section documents the rationale for selecting the maximum period that cwnd may be preserved, known as the NVP.

Limiting the period that cwnd may be preserved avoids undesirable side effects that would result if the cwnd were to be kept unnecessarily high for an arbitrary long period, which was a part of the problem that CWV originally attempted to address. The period a sender may safely preserve the cwnd, is a function of the period that a network path is expected to sustain the capacity reflected by cwnd. There is no ideal choice for this time.

A period of five minutes was chosen for this NVP. This is a compromise that was larger than the idle intervals of common applications, but not sufficiently larger than the period for which the capacity of an Internet path may commonly be regarded as stable. The capacity of wired networks is usually relatively stable for periods of several minutes and that load stability increases with the capacity. This suggests that cwnd may be preserved for at least a few minutes.

There are cases where the TCP throughput exhibits significant variability over a time less than five minutes. Examples could include wireless topologies, where TCP rate variations may fluctuate on the order of a few seconds as a consequence of medium access protocol instabilities. Mobility changes may also impact TCP performance over short time scales. Senders that observe such rapid changes in the path characteristic may also experience increased congestion with the new method, however such variation would likely also impact TCP's behaviour when supporting interactive and bulk applications.

Routing algorithms may change the the network path that is used by a transport. Although a change of path can in turn disrupt the RTT measurement and may result in a change of the capacity available to a TCP connection, we assume these path changes do not usually occur frequently (compared to a time frame of a few minutes).

The value of five minutes is therefore expected to be sufficient for most current applications. Simulation studies (e.g., [\[Bis11\]](#)) also suggest that for many practical applications, the performance using this value will not be significantly different to that observed using a non-standard method that does not reset the cwnd after idle.

Finally, other TCP sender mechanisms have used a 5 minute timer, and there could be simplifications in some implementations by reusing the same interval. TCP defines a default user timeout of 5 minutes [[RFC0793](#)] i.e., how long transmitted data may remain unacknowledged before a connection is forcefully closed.

6. Security Considerations

General security considerations concerning TCP congestion control are discussed in [[RFC5681](#)]. This document describes an algorithm that updates one aspect of the congestion control procedures, and so the considerations described in [RFC 5681](#) also apply to this algorithm.

7. IANA Considerations

There are no IANA considerations.

8. Acknowledgments

This document was produced by the TCP Maintenance and Minor Extensions (tcpm) working group.

The authors acknowledge the contributions of Dr I Biswas, Dr Ziaul Hossain in supporting the evaluation of CWV and for their help in developing the mechanisms proposed in this draft. We also acknowledge comments received from the Internet Congestion Control Research Group, in particular Yuchung Cheng, Mirja Kuehlewind, Joe Touch, and Mark Allman. This work was part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700).

9. Author Notes

RFC-Editor note: please remove this section prior to publication.

9.1. Other related work

RFC-Editor note: please remove this section prior to publication.

There are several issues to be discussed more widely:

- o There are potential interactions with the Experimental update in [RFC 6928](#) that raises the TCP initial Window to ten segments, do these cases need to be elaborated?

This relates to the Experimental specification for increasing the TCP IW defined in [RFC 6928](#).

The two methods have different functions and different response to loss/congestion.

[RFC 6928](#) proposes an experimental update to TCP that would increase the IW to ten segments. This would allow faster opening of the cwnd, and also a large (same size) restart window. This approach is based on the assumption that many forward paths can sustain bursts of up to ten segments without (appreciable) loss. Such a significant increase in cwnd must be matched with an equally large reduction of cwnd if loss/congestion is detected, and such a congestion indication is likely to require future use of IW=10 to be disabled for this path for some time. This guards against the unwanted behaviour of a series of short flows continuously flooding a network path without network congestion feedback.

In contrast, this document proposes an update with a rationale that relies on recent previous path history to select an appropriate cwnd after restart.

The behaviour differs in three ways:

- 1) For applications that send little initially, new-cwv may constrain more than [RFC 6928](#), but would not require the connection to reset any path information when a restart incurred loss. In contrast, new-cwv would allow the TCP connection to preserve the cached cwnd, any loss, would impact cwnd, but not impact other flows.
- 2) For applications that utilise more capacity than provided by a cwnd of 10 segments, this method would permit a larger restart window compared to a restart using the method in [RFC 6928](#). This is justified by the recent path history.
- 3) new-CWV is attended to also be used for rate-limited applications, where the application sends, but does not seek to fully utilise the cwnd. In this case, new-cwv constrains the cwnd to that justified by the recent path history. The performance trade-offs are hence different, and it would be possible to enable new-cwv when also using the method in [RFC 6928](#), and yield benefits.

o There is potential overlap with the Laminar proposal ([draft-mathis-tcpm-tcp-laminar](#))

The current draft was intended as a standards-track update to TCP, rather than a new transport variant. At least, it would be good to understand how the two interact and whether there is a possibility of a single method.

- o There is potential performance loss in loss of a short burst (off list with M Allman)

A sender can transmit several segments then become idle. If the first set of segments are all Acknowledged, the ssthresh collapses to a small value (no new data is sent by the idle sender). Loss of the later data results in congestion (e.g., maybe a RED drop or some other cause, rather than the maximum rate of this flow). When the sender performs loss recovery it may have an appreciable pipeACK and cwnd, but a very low FlightSize - the Standard algorithm therefore results in an unusually low cwnd $((1/2) * \text{FlightSize})$.

A constant rate flow would have maintained a FlightSize appropriate to pipeACK (cwnd, if it is a bulk flow).

This could be fixed by adding a new state variable? It could also be argued this is a corner case (e.g., loss of only the last segments would have resulted in RTO), the impact could be significant.

- o There is potential interaction with TCP Control Block Sharing(M Welzl)

An application that is non-validated can accumulate a cwnd that is larger than the actual capacity. Is this a fair value to use in TCB sharing?

We propose that TCB sharing should use the pipeACK in place of cwnd when a TCP sender is in the Non-validated phase. This value better reflects the capacity that the flow has utilised in the network path.

10. Revision notes

RFC-Editor note: please remove this section prior to publication.

Draft 03 was submitted to ICCRG to receive comments and feedback.

Draft 04 contained the first set of clarifications after feedback:

- o Changed name to application limited and used the term rate-limited in all places.
- o Added justification and many minor changes suggested on the list.
- o Added text to tie-in with more accurate ECN marking.
- o Added ref to Hug01

Draft 05 contained various updates:

- o New text to redefine how to measure the acknowledged pipe, differentiating this from the FlightSize, and hence avoiding previous issues with infrequent large bursts of data not being validated. A key point new feature is that pipeACK only triggers leaving the NVP after the size of the pipe has been acknowledged. This removed the need for hysteresis.
- o Reduction values were changed to 1/2, following analysis of suggestions from ICCRG. This also sets the "target" cwnd as twice the used rate for non-validated case.
- o Introduced a symbolic name (NVP) to denote the 5 minute period.

Draft 06 contained various updates:

- o Required reset of pipeACK after congestion.
- o Added comment on the effect of congestion after a short burst (M. Allman).
- o Correction of minor Typos.

WG draft 00 contained various updates:

- o Updated initialisation of pipeACK to maximum value.
- o Added note on intended status still to be determined.

WG draft 01 contained:

- o Added corrections from Richard Scheffenegger.
- o Raffaello Secchi added to the mechanism, based on implementation experience.

- o Removed that the requirement for the method to use TCP SACK option
- o Although it may be desirable to use SACK, this is not essential to the algorithm.
- o Added the notion of the sampling period to accommodate large rate variations and ensure that the method is stable. This algorithm to be validated through implementation.

WG draft 02 contained:

- o Clarified language around pipeACK variable and pipeACK sample - Feedback from Aris Angelogiannopoulos.

WG draft 03 contained:

- o Editorial corrections - Feedback from Anna Brunstrom.
- o An adjustment to the procedure at the start and end of Reoloss recovery to align the two equations.
- o Further clarification of the "undefined" value of the pipeACK variable.

WG draft 04 contained:

- o Editorial corrections.
- o Introduced the "cwnd-limited" term.
- o An adjustment to the procedure at the start of a cwnd-limited phase - the new text is intended to ensure that new-cwv is not unnecessarily more conservative than standard TCP when the flow is cwnd-limited. This resolves two issues: first it prevents pathologies in which pipeACK increases slowly and erratically. It also ensures that performance of bulk applications is not significantly impacted when using the method.
- o Clearly identifies that pacing (or equivalent) is requiring during the NVP to control burstiness. New section added.

WG draft 05 contained:

- o Clarification to first two bullets in [Section 4.4](#) describing cwnd-limited, to explain these are really alternates to the same case.
- o Section giving implementation examples was restructured to clarify there are two methods described.

- o Cross References to sections updated - thanks to comments from Martin Winbjork and Tim Wicinski.

WG draft 06 contained:

- o The section giving implementation examples was restructured to clarify there are two methods described.
- o Justification of design decisions.
- o Re-organised text to improve clarity of argument.

WG draft 07 contained:

- o Updated publication date.
- o Text on noting that cwnd shouldn't ever be made negative.
- o Updated text on ECN to clarify the process where R is a reduction based on ECN marks.

WG draft 08 contained:

- o Removed description of how to use Accurate ECN feedback. It is not clear that this document should specify a usage of a mechanism that has not been fully defined. Accurate ECN may lead to different congestion responses and these will need to be defined in the CC specifications for using Accurate ECN.

WG draft 09 contained:

- o Removed update to [RFC 5681](#) - the status of the present document is Experimental, and hence this document does not update [RFC 5681](#).

WG draft 10 contained edits following WGLC:

- o [Section 1.1](#) Implementation of new CWV: New section added to introduce the places where there are implementation flexibility.
- o [Section 4.4](#): Clarified that the MUST is to satisfy the goal to avoid a TCP sender growing a large "non-validated" cwnd, when it has not recently sent using the current size of cwnd, and fixed format of bullet 2 in 4.4.
- o [Section 4.5.2](#): rewritten section text.

WG draft 11 contained edits following IETF LC:

- o Updated text in [section 1.1](#).
- o Updated text in response to AD, Gen-ART, & Sec reviews.
- o LC call comments from Mirja Kuehlewind

WG draft 12 contained edits following IETF LC (Mirja Kuehlewind):

- o Additional text (based on text in annexe notes) to clarify use of pipeACK rather than FlightSize.
- o Corrected text on undefined pipeACK - to be consistent.
- o Added text on standard TCP method (reference to [RFC 5681](#)).
- o Separated text on implementation experience of "timers" into a new implementation subsection (4.5.2), to avoid this common implementation method being overlooked.

[11](#). References

[11.1](#). Normative References

- [RFC0793] Postel, J., "Transmission Control Protocol", September 1981.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", [RFC 2018](#), October 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC2861] Handley, M., Padhye, J., and S. Floyd, "TCP Congestion Window Validation", [RFC 2861](#), June 2000.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", September 2009.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", June 2011.
- [RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", [RFC 6675](#), August 2012.

11.2. Informative References

- [All05] Allman, M. and E. Blanton, "Notes on burst mitigation for transport protocols", March 2005.
- [Bis08] Biswas, I. and G. Fairhurst, "A Practical Evaluation of Congestion Window Validation Behaviour, 9th Annual Postgraduate Symposium in the Convergence of Telecommunications, Networking and Broadcasting (PGNet), Liverpool, UK", June 2008.
- [Bis10] Biswas, I., Sathiaselalan, A., Secchi, R., and G. Fairhurst, "Analysing TCP for Bursty Traffic, Int'l J. of Communications, Network and System Sciences, 7(3)", June 2010.
- [Bis11] Biswas, I., "PhD Thesis, Internet congestion control for variable rate TCP traffic, School of Engineering, University of Aberdeen", June 2011.
- [Fai12] Sathiaselalan, A., Secchi, R., Fairhurst, G., and I. Biswas, "Enhancing TCP Performance to support Variable-Rate Traffic, 2nd Capacity Sharing Workshop, ACM CoNEXT, Nice, France, 10th December 2012.", June 2008.
- [Hos15] Hossain, Z., "PhD Thesis, A Study of Mechanisms to Support Variable-rate Internet Applications over a Multi-service Satellite Platform, School of Engineering, University of Aberdeen", January 2015.
- [Hug01] Hughes, A., Touch, J., and J. Heidemann, "Issues in TCP Slow-Start Restart After Idle (Work-in-Progress)", December 2001.
- [Liu07] Liu, D., Allman, M., Jin, S., and L. Wang, "Congestion Control without a Startup Phase, 5th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet), Los Angeles, California, USA", February 2007.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", [RFC 2616](#), June 1999.

Authors' Addresses

Godred Fairhurst
University of Aberdeen
School of Engineering
Fraser Noble Building
Aberdeen, Scotland AB24 3UE
UK

Email: gorry@erg.abdn.ac.uk
URI: <http://www.erg.abdn.ac.uk>

Arjuna Sathiaselan
University of Aberdeen
School of Engineering
Fraser Noble Building
Aberdeen, Scotland AB24 3UE
UK

Email: arjuna@erg.abdn.ac.uk
URI: <http://www.erg.abdn.ac.uk>

Raffaello Secchi
University of Aberdeen
School of Engineering
Fraser Noble Building
Aberdeen, Scotland AB24 3UE
UK

Email: raffaello@erg.abdn.ac.uk
URI: <http://www.erg.abdn.ac.uk>

