**TCP congestion control**

From Wikipedia, the free encyclopedia

[Jump to navigation](https://en.wikipedia.org/wiki/TCP_congestion_control#mw-head) [Jump to search](https://en.wikipedia.org/wiki/TCP_congestion_control#searchInput)

[Transmission Control Protocol](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) (TCP) uses a [network congestion-avoidance](https://en.wikipedia.org/wiki/Network_congestion-avoidance) algorithm that includes various aspects of an [additive increase/multiplicative decrease](https://en.wikipedia.org/wiki/Additive_increase/multiplicative_decrease) (AIMD) scheme, along with other schemes including **slow start** and **congestion window**, to achieve congestion avoidance. The **TCP congestion-avoidance algorithm** is the primary basis for [congestion control](https://en.wikipedia.org/wiki/Congestion_control) in the Internet.[[1]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-FOOTNOTEJacobsonKarels1988-1)[[2]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-RFC_2001-2)[[3]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-RFC_3390-3)[[4]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-4) Per the [end-to-end principle](https://en.wikipedia.org/wiki/End-to-end_principle), congestion control is largely a function of [internet hosts](https://en.wikipedia.org/wiki/Internet_host), not the network itself. There are several variations and versions of the algorithm implemented in [protocol stacks](https://en.wikipedia.org/wiki/Protocol_stack) of [operating systems](https://en.wikipedia.org/wiki/Operating_system) of computers that connect to the [Internet](https://en.wikipedia.org/wiki/Internet).



**Contents**

* [1 Operation](https://en.wikipedia.org/wiki/TCP_congestion_control#Operation)
* [2 Congestion window](https://en.wikipedia.org/wiki/TCP_congestion_control#Congestion_window)
* [3 Slow start](https://en.wikipedia.org/wiki/TCP_congestion_control#Slow_start)
* [4 Additive increase/multiplicative decrease](https://en.wikipedia.org/wiki/TCP_congestion_control#Additive_increase/multiplicative_decrease)
* [5 Fast retransmit](https://en.wikipedia.org/wiki/TCP_congestion_control#Fast_retransmit)
* [6 Algorithms](https://en.wikipedia.org/wiki/TCP_congestion_control#Algorithms)
  + [6.1 TCP Tahoe and Reno](https://en.wikipedia.org/wiki/TCP_congestion_control#TCP_Tahoe_and_Reno)
  + [6.2 TCP Vegas](https://en.wikipedia.org/wiki/TCP_congestion_control#TCP_Vegas)
  + [6.3 TCP New Reno](https://en.wikipedia.org/wiki/TCP_congestion_control#TCP_New_Reno)
  + [6.4 TCP Hybla](https://en.wikipedia.org/wiki/TCP_congestion_control#TCP_Hybla)
  + [6.5 TCP BIC](https://en.wikipedia.org/wiki/TCP_congestion_control#TCP_BIC)
  + [6.6 TCP CUBIC](https://en.wikipedia.org/wiki/TCP_congestion_control#TCP_CUBIC)
  + [6.7 Agile-SD TCP](https://en.wikipedia.org/wiki/TCP_congestion_control#Agile-SD_TCP)
  + [6.8 TCP Westwood+](https://en.wikipedia.org/wiki/TCP_congestion_control#TCP_Westwood+)
  + [6.9 Compound TCP](https://en.wikipedia.org/wiki/TCP_congestion_control#Compound_TCP)
  + [6.10 TCP Proportional Rate Reduction](https://en.wikipedia.org/wiki/TCP_congestion_control#TCP_Proportional_Rate_Reduction)
  + [6.11 TCP BBR](https://en.wikipedia.org/wiki/TCP_congestion_control#TCP_BBR)
  + [6.12 C2TCP](https://en.wikipedia.org/wiki/TCP_congestion_control#C2TCP)
  + [6.13 Elastic-TCP](https://en.wikipedia.org/wiki/TCP_congestion_control#Elastic-TCP)
  + [6.14 NATCP/NACubic](https://en.wikipedia.org/wiki/TCP_congestion_control#NATCP/NACubic)
  + [6.15 Other TCP congestion avoidance algorithms](https://en.wikipedia.org/wiki/TCP_congestion_control#Other_TCP_congestion_avoidance_algorithms)
* [7 Classification by network awareness](https://en.wikipedia.org/wiki/TCP_congestion_control#Classification_by_network_awareness)
  + [7.1 Black box](https://en.wikipedia.org/wiki/TCP_congestion_control#Black_box)
  + [7.2 Grey box](https://en.wikipedia.org/wiki/TCP_congestion_control#Grey_box)
  + [7.3 Green box](https://en.wikipedia.org/wiki/TCP_congestion_control#Green_box)
* [8 Usage](https://en.wikipedia.org/wiki/TCP_congestion_control#Usage)
* [9 See also](https://en.wikipedia.org/wiki/TCP_congestion_control#See_also)
* [10 Notes](https://en.wikipedia.org/wiki/TCP_congestion_control#Notes)
* [11 References](https://en.wikipedia.org/wiki/TCP_congestion_control#References)
  + [11.1 Sources](https://en.wikipedia.org/wiki/TCP_congestion_control#Sources)
* [12 External links](https://en.wikipedia.org/wiki/TCP_congestion_control#External_links)

**Operation**

To avoid [congestive collapse](https://en.wikipedia.org/wiki/Congestive_collapse), TCP uses a multi-faceted congestion-control strategy. For each connection, TCP maintains a [congestion window](https://en.wikipedia.org/wiki/TCP_congestion_control#Congestion_window), limiting the total number of unacknowledged packets that may be in transit end-to-end. This is somewhat analogous to TCP's [sliding window](https://en.wikipedia.org/wiki/Sliding_window) used for [flow control](https://en.wikipedia.org/wiki/Transmission_Control_Protocol#Flow_control). TCP uses a mechanism called [slow start](https://en.wikipedia.org/wiki/TCP_congestion_control#Slow_start)[[1]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-FOOTNOTEJacobsonKarels1988-1) to increase the congestion window after a connection is initialized or after a [timeout](https://en.wikipedia.org/wiki/Timeout_(computing)). It starts with a window, a small multiple of the [maximum segment size](https://en.wikipedia.org/wiki/Maximum_segment_size) (MSS) in size. Although the initial rate is low, the rate of increase is very rapid; for every packet acknowledged, the congestion window increases by 1 MSS so that the congestion window effectively doubles for every [round-trip time](https://en.wikipedia.org/wiki/Round-trip_delay_time) (RTT).

When the congestion window exceeds the slow-start threshold, *ssthresh*,[[a]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-5) the algorithm enters a new state, called [congestion avoidance](https://en.wikipedia.org/wiki/Congestion_avoidance). In congestion avoidance state, as long as non-duplicate ACKs are received[[b]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-6) the congestion window is additively increased by one MSS every round-trip time.

**Congestion window**

In TCP, the **congestion window** is one of the factors that determines the number of bytes that can be sent out at any time. The congestion window is maintained by the sender and is a means of stopping *a link* between the sender and the receiver from becoming overloaded with too much traffic. This should not to be confused with the sliding window maintained by the receiver which exists to prevent *the receiver* from becoming overloaded. The congestion window is calculated by estimating how much congestion there is on the link.

When a connection is set up, the congestion window, a value maintained independently at each host, is set to a small multiple of the MSS allowed on that connection. Further variance in the congestion window is dictated by an AIMD approach. This means that if all segments are received and the acknowledgments reach the sender on time, some constant is added to the window size. When the window reaches *ssthresh*, the congestion window increases linearly at the rate of 1/(congestion window) segment on each new acknowledgement received. The window keeps growing until a timeout occurs. On timeout:

1. Congestion window is reset to 1 MSS.
2. *ssthresh* is set to half the congestion window size before the timeout.
3. *slow start* is initiated.

A [system administrator](https://en.wikipedia.org/wiki/System_administrator) may adjust the maximum window size limit, or adjust the constant added during additive increase, as part of [TCP tuning](https://en.wikipedia.org/wiki/TCP_tuning).

The flow of data over a TCP connection is also controlled by the use of the [*receive window*](https://en.wikipedia.org/wiki/Transmission_Control_Protocol#Flow_control) advertised by the receiver. By comparing its own congestion window with the *receive window*, a sender can determine how much data it may send at any given time.

**Slow start**

"Slow Start" redirects here. For the Japanese manga, see [Slow Start (manga)](https://en.wikipedia.org/wiki/Slow_Start_(manga)).

Slow start is part of the [congestion control](https://en.wikipedia.org/wiki/Congestion_control) strategy used by TCP in conjunction with other [algorithms](https://en.wikipedia.org/wiki/Algorithm) to avoid sending more data than the network is capable of forwarding, that is, to avoid causing network congestion. The algorithm is specified by [RFC 5681](https://tools.ietf.org/html/rfc5681).

Although the strategy is referred to as slow start, its congestion window growth is quite aggressive, more aggressive than the congestion avoidance phase.[[1]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-FOOTNOTEJacobsonKarels1988-1) Before slow start was introduced in TCP, the initial pre-congestion avoidance phase was even faster.

Slow start begins initially with a congestion window size (CWND) of 1, 2, 4 or 10 MSS.[[5]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-7)[[3]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-RFC_3390-3):1 The value for the congestion window size will be increased by one with each [acknowledgement](https://en.wikipedia.org/wiki/Acknowledgement_(data_networks)) (ACK) received, effectively doubling the window size each round-trip time.[[c]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-8) The transmission rate will be increased by the slow-start algorithm until either a loss is detected, or the receiver's advertised window (rwnd) is the limiting factor, or *ssthresh* is reached. If a loss event occurs, TCP assumes that it is due to network congestion and takes steps to reduce the offered load on the network. These measurements depend on the exact TCP congestion avoidance algorithm used.

TCP Tahoe

When a loss occurs, [fast retransmit](https://en.wikipedia.org/wiki/TCP_congestion_control#Fast_retransmit) is sent, half of the current CWND is saved as *ssthresh* and slow start begins again from its initial CWND. Once the CWND reaches *ssthresh*, TCP changes to congestion avoidance algorithm where *each new ACK* increases the CWND by *MSS / CWND.* This results in a linear increase of the CWND.

TCP Reno

A fast retransmit is sent, half of the current CWND is saved as *ssthresh* and as new CWND, thus skipping slow start and going directly to the congestion avoidance algorithm. The overall algorithm here is called fast recovery.

Once *ssthresh* is reached, TCP changes from slow-start algorithm to the linear growth (congestion avoidance) algorithm. At this point, the window is increased by 1 segment for each [round-trip delay time](https://en.wikipedia.org/wiki/Round-trip_delay_time) (RTT).

Slow start assumes that unacknowledged segments are due to network congestion. While this is an acceptable assumption for many networks, segments may be lost for other reasons, such as poor [data link layer](https://en.wikipedia.org/wiki/Data_link_layer) transmission quality. Thus, slow start can perform poorly in situations with poor reception, such as [wireless networks](https://en.wikipedia.org/wiki/Wireless_LAN).

The slow start protocol also performs badly for short-lived connections. Older [web browsers](https://en.wikipedia.org/wiki/Web_browsers) would create many consecutive short-lived connections to the web server, and would open and close the connection for each file requested. This kept most connections in the slow start mode, which resulted in poor response time. To avoid this problem, modern browsers either open multiple connections simultaneously or [reuse one connection](https://en.wikipedia.org/wiki/HTTP_persistent_connections) for all files requested from a particular web server. Connections, however, cannot be reused for the multiple third-party servers used by web sites to implement [web advertising](https://en.wikipedia.org/wiki/Advertising_network), [sharing features](https://en.wikipedia.org/wiki/Like_button) of [social networking services](https://en.wikipedia.org/wiki/Social_networking_service),[[6]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-9) and [counter scripts of web analytics](https://en.wikipedia.org/wiki/Web_analytics#Page_tagging).

**Additive increase/multiplicative decrease**

The [additive increase/multiplicative decrease](https://en.wikipedia.org/wiki/Additive_increase/multiplicative_decrease) (AIMD) algorithm is a [closed-loop control algorithm](https://en.wikipedia.org/wiki/Control_system). AIMD combines linear growth of the congestion window with an exponential reduction when a congestion takes place. Multiple flows using AIMD congestion control will eventually converge to use equal amounts of a contended link.[[7]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-chui1989-10)

**Fast retransmit**

**Fast retransmit** is an enhancement to [TCP](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) that reduces the time a sender waits before retransmitting a lost segment. A TCP sender normally uses a simple timer to recognize lost segments. If an acknowledgement is not received for a particular segment within a specified time (a function of the estimated [round-trip delay time](https://en.wikipedia.org/wiki/Round-trip_delay_time)), the sender will assume the segment was lost in the network, and will retransmit the segment.

Duplicate acknowledgement is the basis for the fast retransmit mechanism. After receiving a packet an acknowledgement is sent for the last in order byte of data received. For an in order packet, this is effectively the last packet's sequence number plus its payload length. If the next packet in the sequence is lost but a third packet in the sequence is received, then the receiver can only acknowledge the last in-order byte of data, which is the same value as was acknowledged for the first packet. The second packet is lost and the third packet is not in order, so the last in order byte of data remains the same as before. Thus a *Duplicate acknowledgement* occurs. The sender continues to send packets, and a fourth and fifth packet are received by the receiver. Again, the second packet is missing from the sequence, so the last in order byte has not changed. Duplicate acknowledgements are sent to both these packets.

When a sender receives three duplicate acknowledgements, it can be reasonably confident that the segment carrying the data that followed the last in order byte specified in the acknowledgment was lost. A sender with fast retransmit will then retransmit this packet immediately without waiting for its timeout. On receipt of the re-transmitted segment, the receiver can acknowledge the last in order byte of data received. In the above example, this would acknowledge to the end of the payload of the fifth packet. There is no need to acknowledge intermediate packets.

**Algorithms**

The naming convention for congestion control algorithms (CCAs) may have originated in a 1996 paper by Kevin Fall and Sally Floyd.[[8]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-11)[[*failed verification*](https://en.wikipedia.org/wiki/Wikipedia:Verifiability)]

The following is one possible classification according to the following properties:

1. the type and amount of feedback received from the network
2. incremental deployability on the current Internet
3. the aspect of performance it aims to improve: high [bandwidth-delay product](https://en.wikipedia.org/wiki/Bandwidth-delay_product) networks (B); lossy links (L); fairness (F); advantage to short flows (S); variable-rate links (V); [speed of convergence](https://en.wikipedia.org/wiki/Speed_of_convergence) (C)
4. the fairness criterion it uses

Some well-known congestion avoidance mechanisms are classified by this scheme as follows:

| **Variant** | **Feedback** | **Required changes** | **Benefits** | **Fairness** |
| --- | --- | --- | --- | --- |
| (New) Reno | Loss | — | — | Delay |
| Vegas | Delay | Sender | Less loss | Proportional |
| High Speed | Loss | Sender | High bandwidth |  |
| BIC | Loss | Sender | High bandwidth |  |
| CUBIC | Loss | Sender | High bandwidth |  |
| C2TCP[[9]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-C2TCP-JSAC-12)[[10]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-C2TCP-13) | Loss/Delay | Sender | Ultra-low latency and high bandwidth |  |
| NATCP[[11]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-FOOTNOTEAbbaslooXuChaoShi2019-14) | Multi-bit signal | Sender | Near Optimal Performance |  |
| Elastic-TCP | Loss/Delay | Sender | High bandwidth/short & long-distance |  |
| Agile-TCP | Loss | Sender | High bandwidth/short-distance |  |
| H-TCP | Loss | Sender | High bandwidth |  |
| FAST | Delay | Sender | High bandwidth | Proportional |
| Compound TCP | Loss/Delay | Sender | High bandwidth | Proportional |
| Westwood | Loss/Delay | Sender | L |  |
| Jersey | Loss/Delay | Sender | L |  |
| BBR[[12]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-15) | Delay | Sender | BLVC, [Bufferbloat](https://en.wikipedia.org/wiki/Bufferbloat) |  |
| CLAMP | Multi-bit signal | Receiver, Router | V | Max-min |
| TFRC | Loss | Sender, Receiver | No Retransmission | Minimum delay |
| XCP | Multi-bit signal | Sender, Receiver, Router | BLFC | Max-min |
| VCP | 2-bit signal | Sender, Receiver, Router | BLF | Proportional |
| MaxNet | Multi-bit signal | Sender, Receiver, Router | BLFSC | Max-min |
| JetMax | Multi-bit signal | Sender, Receiver, Router | High bandwidth | Max-min |
| RED | Loss | Router | Reduced delay |  |
| ECN | Single-bit signal | Sender, Receiver, Router | Reduced loss |  |

**TCP Tahoe and Reno**

TCP Tahoe and Reno algorithms were retrospectively named after the versions or flavours of the [4.3BSD](https://en.wikipedia.org/wiki/4.3BSD) operating system in which each first appeared (which were themselves named after [Lake Tahoe](https://en.wikipedia.org/wiki/Lake_Tahoe) and the nearby city of [Reno, Nevada](https://en.wikipedia.org/wiki/Reno,_Nevada)). The Tahoe algorithm first appeared in 4.3BSD-Tahoe (which was made to support the [CCI Power 6/32 "Tahoe" minicomputer](https://en.wikipedia.org/wiki/Computer_Consoles_Inc.#Power_5_and_Power_6_computers)), and was later made available to non-AT&T licensees as part of the 4.3BSD Networking Release 1; this ensured its wide distribution and implementation. Improvements were made in 4.3BSD-Reno and subsequently released to the public as Networking Release 2 and later 4.4BSD-Lite.

While both consider retransmission timeout (RTO) and duplicate ACKs as packet loss events, the behavior of Tahoe and Reno differ primarily in how they react to duplicate ACKs:

* Tahoe: if three duplicate ACKs are received (i.e. four ACKs acknowledging the same packet, which are not piggybacked on data and do not change the receiver's advertised window), Tahoe performs a fast retransmit, sets the slow start threshold to half of the current congestion window, reduces the congestion window to 1 MSS, and resets to slow start state.[[13]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-FOOTNOTEKuroseRoss2008284-16)
* Reno: if three duplicate ACKs are received, Reno will perform a fast retransmit and skip the slow start phase by instead halving the congestion window (instead of setting it to 1 MSS like Tahoe), setting the slow start threshold equal to the new congestion window, and enter a phase called *fast recovery*.[[14]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-FOOTNOTEKuroseRoss2012277-17)

In both Tahoe and Reno, if an ACK times out (RTO timeout), slow start is used, and both algorithms reduce congestion window to 1 MSS.

**TCP Vegas**

Main article: [TCP Vegas](https://en.wikipedia.org/wiki/TCP_Vegas)

Until the mid-1990s, all of TCP's set timeouts and measured round-trip delays were based upon only the last transmitted packet in the transmit buffer. [University of Arizona](https://en.wikipedia.org/wiki/University_of_Arizona) researchers Larry Peterson and [Lawrence Brakmo](https://en.wikipedia.org/wiki/Lawrence_Brakmo) introduced TCP Vegas (named after [Las Vegas](https://en.wikipedia.org/wiki/Las_Vegas), the largest city in Nevada) in which timeouts were set and round-trip delays were measured for every packet in the transmit buffer. In addition, TCP Vegas uses additive increases in the congestion window. In a comparison study of various TCP congestion control algorithms, TCP Vegas appeared to be the smoothest followed by TCP CUBIC.[[15]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-18)

TCP Vegas was not widely deployed outside Peterson's laboratory but was selected as the default congestion control method for [DD-WRT](https://en.wikipedia.org/wiki/DD-WRT) firmware v24 SP2.[[16]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-19)

**TCP New Reno**

TCP New Reno, defined by [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [6582](https://tools.ietf.org/html/rfc6582) (which obsoletes previous definitions in [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [3782](https://tools.ietf.org/html/rfc3782) and [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [2582](https://tools.ietf.org/html/rfc2582)), improves retransmission during the fast-recovery phase of TCP Reno. During fast recovery, to keep the transmit window full, for every duplicate ACK that is returned, a new unsent packet from the end of the congestion window is sent. For every ACK that makes partial progress in the sequence space, the sender assumes that the ACK points to a new hole, and the next packet beyond the ACKed sequence number is sent.

Because the timeout is reset whenever there is progress in the transmit buffer, New Reno can fill large holes, or multiple holes, in the sequence space – much like [TCP SACK](https://en.wikipedia.org/wiki/TCP_SACK). Because New Reno can send new packets at the end of the congestion window during fast recovery, high throughput is maintained during the hole-filling process, even when there are multiple holes, of multiple packets each. When TCP enters fast recovery it records the highest outstanding unacknowledged packet sequence number. When this sequence number is acknowledged, TCP returns to the congestion avoidance state.

A problem occurs with New Reno when there are no packet losses but instead, packets are reordered by more than 3 packet sequence numbers. In this case, New Reno mistakenly enters fast recovery. When the reordered packet is delivered, ACK sequence-number progress occurs and from there until the end of fast recovery, all sequence-number progress produces a duplicate and needless retransmission that is immediately ACKed.[[*clarification needed*](https://en.wikipedia.org/wiki/Wikipedia:Please_clarify)]

New Reno performs as well as SACK at low packet error rates, and substantially outperforms Reno at high error rates.[[17]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-20)

**TCP Hybla**

TCP Hybla aims to eliminate penalties to TCP connections that incorporate a high-latency terrestrial or satellite radio links. Hybla improvements are based on analytical evaluation of the congestion window dynamics.[[18]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-21)

**TCP BIC**

Main article: [BIC TCP](https://en.wikipedia.org/wiki/BIC_TCP)

Binary Increase Congestion control (BIC) is a TCP implementation with an optimized CCA for high speed networks with high latency, known as [long fat networks](https://en.wikipedia.org/wiki/Long_fat_network).[[19]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-22) BIC is used by default in [Linux kernels](https://en.wikipedia.org/wiki/Linux_kernel) 2.6.8 through 2.6.18.[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]

**TCP CUBIC**

Main article: [CUBIC TCP](https://en.wikipedia.org/wiki/CUBIC_TCP)

CUBIC is a less aggressive and more systematic derivative of BIC, in which the window is a cubic function of time since the last congestion event, with the inflection point set to the window prior to the event. CUBIC is used by default in [Linux kernels](https://en.wikipedia.org/wiki/Linux_kernel) between versions 2.6.19 and 3.2.

**Agile-SD TCP**

Agile-SD is a Linux-based CCA which is designed for the real Linux kernel. It is a receiver-side algorithm that employs a loss-based approach using a novel mechanism, called *agility factor* (AF). to increase the bandwidth utilization over high-speed and short-distance networks (low-BDP networks) such as local area networks or fiber-optic network, especially when the applied buffer size is small.[[20]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-agilesd-23) It has been evaluated by comparing its performance to Compound-TCP (the default CCA in MS Windows) and CUBIC (the default of Linux) using NS-2 simulator. It improves the total performance up to 55% in term of average throughput.

**TCP Westwood+**

Main article: [TCP Westwood plus](https://en.wikipedia.org/wiki/TCP_Westwood_plus)

Westwood+ is a sender-only modification of TCP Reno that optimizes the performance of TCP congestion control over both wired and [wireless networks](https://en.wikipedia.org/wiki/Wireless_network). TCP Westwood+ is based on end-to-end [bandwidth](https://en.wikipedia.org/wiki/Bandwidth_(computing)) estimation to set the congestion window and slow-start threshold after a congestion episode, that is, after three duplicate acknowledgments or a timeout. The bandwidth is estimated by averaging the rate of returning acknowledgment packets. In contrast with TCP Reno, which blindly halves the congestion window after three duplicate ACKs, TCP Westwood+ adaptively sets a slow-start threshold and a congestion window which takes into account an estimate of bandwidth available at the time congestion is experienced. Compared to Reno and New Reno, Westwood+ significantly increases throughput over wireless links and improves fairness in wired networks.[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]

**Compound TCP**

Main article: [Compound TCP](https://en.wikipedia.org/wiki/Compound_TCP)

Compound TCP is a [Microsoft](https://en.wikipedia.org/wiki/Microsoft) implementation of TCP which maintains two different congestion windows simultaneously, with the goal of achieving good performance on LFNs while not impairing [fairness](https://en.wikipedia.org/wiki/Fairness_measure). It has been widely deployed in Windows versions since Microsoft [Windows Vista](https://en.wikipedia.org/wiki/Windows_Vista) and [Windows Server 2008](https://en.wikipedia.org/wiki/Windows_Server_2008) and has been ported to older Microsoft Windows versions as well as [Linux](https://en.wikipedia.org/wiki/Linux).

**TCP Proportional Rate Reduction**

TCP Proportional Rate Reduction (PRR)[[21]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-24) is an algorithm designed to improve the accuracy of data sent during recovery. The algorithm ensures that the window size after recovery is as close as possible to the slow start threshold. In tests performed by [Google](https://en.wikipedia.org/wiki/Google), PRR resulted in a 3–10% reduction in average latency and recovery timeouts reduced by 5%.[[22]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-25) PRR is available in [Linux kernels](https://en.wikipedia.org/wiki/Linux_kernel) since version 3.2.[[23]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-26)

**TCP BBR**

Bottleneck Bandwidth and Round-trip propagation time (BBR) is a CCA developed at Google in 2016.[[24]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-GOOGBBR-27) While most congestion control algorithms are loss-based, in that they rely on packet loss as a signal to lower rates of transmission, BBR, like Vegas, is model-based. The algorithm uses the maximum bandwidth and round-trip time at which the network delivered the most recent flight of outbound data packets to build an explicit model of the network. Each cumulative or selective acknowledgment of packet delivery produces a rate sample which records the amount of data delivered over the time interval between the transmission of a data packet and the acknowledgment of that packet.[[25]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-28) As network interface controllers evolve from megabit per second to gigabit per second performance, latency associated with [bufferbloat](https://en.wikipedia.org/wiki/Bufferbloat) instead of packet loss becomes a more reliable marker of the maximum throughput, making latency/model-based congestion control algorithms which provide higher throughput and lower latency, such as BBR, a more reliable alternative to more popular loss-based algorithms like CUBIC.

When implemented within [YouTube](https://en.wikipedia.org/wiki/YouTube), BBR yielded an average of 4% higher network throughput and up to 14% in some countries.[[26]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-29) BBR is also available for [QUIC](https://en.wikipedia.org/wiki/QUIC). It is available for Linux TCP since Linux 4.9.[[27]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-30)[[28]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-31)

BBR is efficient and fast, but its fairness to non-BBR streams is disputed. While Google's presentation shows BBR co-existing well with CUBIC,[[24]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-GOOGBBR-27) researchers like Geoff Huston and Hock, Bless and Zitterbart finds it unfair to other streams and not scalable.[[29]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-32) Hock et al also found "some severe inherent issues such as increased queuing delays, unfairness, and massive packet loss" in the BBR implementation of Linux 4.9.[[30]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-33)

Soheil Abbasloo et al. (authors of C2TCP) show that BBR doesn't perform well in dynamic environments such as cellular networks.[[9]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-C2TCP-JSAC-12)[[10]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-C2TCP-13) They have also shown that BBR has an unfairness issue. For instance, when a [CUBIC](https://en.wikipedia.org/wiki/CUBIC_TCP) flow (which is the default [TCP](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) implementation in Linux, Android, and MacOS) coexists with a BBR flow in the network, the BBR flow can dominate the CUBIC flow and get the whole link bandwidth from it (see figure 18 in [[9]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-C2TCP-JSAC-12)).

**C2TCP**

Cellular Controlled Delay TCP (C2TCP)[[9]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-C2TCP-JSAC-12)[[10]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-C2TCP-13) was motivated by the lack of a flexible end-to-end TCP approach that can satisfy various [QoS](https://en.wikipedia.org/wiki/Quality_of_service) requirements of different applications without requiring any changes in the network devices. C2TCP aims to satisfy ultra-low [latency](https://en.wikipedia.org/wiki/Network_delay) and high bandwidth requirements of applications such as [virtual reality](https://en.wikipedia.org/wiki/Virtual_reality), [video conferencing](https://en.wikipedia.org/wiki/Videotelephony), [online gaming](https://en.wikipedia.org/wiki/Online_game), [vehicular communication systems](https://en.wikipedia.org/wiki/Vehicular_communication_systems), etc. in a highly dynamic environment such as current [LTE](https://en.wikipedia.org/wiki/LTE_(telecommunication)) and future [5G](https://en.wikipedia.org/wiki/5G) [cellular networks](https://en.wikipedia.org/wiki/Cellular_network). C2TCP works as an [add-on](https://en.wikipedia.org/wiki/Plug-in_(computing)) on top of loss-based TCP (e.g. Reno, NewReno, [CUBIC](https://en.wikipedia.org/wiki/CUBIC_TCP), [BIC](https://en.wikipedia.org/wiki/BIC_TCP), ...) and makes the average delay of packets bounded to the desired delays set by the applications.

Researchers at [NYU](https://en.wikipedia.org/wiki/New_York_University)[[31]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-34) showed that C2TCP outperforms the delay/Jitter performance of various state-of-the-art TCP schemes. For instance, they showed that compared to BBR, CUBIC, and Westwood on average, C2TCP decreases the average delay of packets by about 250%, 900%, and 700% respectively on various cellular network environments.[[9]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-C2TCP-JSAC-12)

C2TCP is only required to be installed on the server-side.

**Elastic-TCP**

Elastic-TCP has been proposed in February 2019 by Mohamed A. Alrshah et al.[[32]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-elastictcp-35) to increase the bandwidth utilization over high-BDP networks to support recent applications such as cloud computing, big data transfer, IoT, etc. It is a Linux-based CCA which is designed for the Linux kernel. It is a receiver-side algorithm that employs a Loss-delay-based approach using a novel mechanism, called Window-correlated Weighting Function (WWF). It has a high level of elasticity to deal with different network characteristics without the need for human tuning. It has been evaluated by comparing its performance to Compound-TCP (the default CCA in MS Windows), CUBIC (the default of Linux) and TCP-BBR (the default of Linux 4.9 by Google) using NS-2 simulator and testbed. Elastic-TCP significantly improves the total performance in term of average throughput, loss ratio, and delay.[[32]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-elastictcp-35)

**NATCP/NACubic**

Recently, Soheil Abbasloo et. al. proposed NATCP (Network-Assisted TCP)[[11]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-FOOTNOTEAbbaslooXuChaoShi2019-14) a controversial TCP design targeting Mobile Edge networks such as [MEC](https://en.wikipedia.org/wiki/Mobile_edge_computing). The key idea of NATCP is that if the characteristics of the network were known beforehand, TCP would have been designed in a better way. Therefore, NATCP employs the available features and properties in the current MEC-based cellular architectures to push the performance of TCP close to the optimal performance. NATCP uses an out-of-band feedback from the network to the servers located nearby. The feedback from the network, which includes the capacity of cellular access link and the minimum RTT of the network, guides the servers to adjust their sending rates. As preliminary results show,[[11]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-FOOTNOTEAbbaslooXuChaoShi2019-14)[[33]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-36) NATCP outperforms the state-of-the-art TCP schemes by at least achieving 2x higher Power (defined as Throughput/Delay). NATCP replaces the traditional TCP scheme at the sender.[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]

To deal with backward compatibility issue, they proposed another version called NACubic. NACubic is a backward compatible design, requiring no change in TCP on the connected nodes. NACubic employs the received feedback and enforces a cap on the congestion window (CWND) and the pacing rate as required. [[11]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-FOOTNOTEAbbaslooXuChaoShi2019-14)

**Other TCP congestion avoidance algorithms**

* [FAST TCP](https://en.wikipedia.org/wiki/FAST_TCP)
* Generalized FAST TCP[[34]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-37)
* [H-TCP](https://en.wikipedia.org/wiki/H-TCP)
* [Data Center TCP](https://en.wikipedia.org/wiki/Data_Center_TCP)
* [High Speed TCP](https://en.wikipedia.org/wiki/High_Speed_TCP)
* HSTCP-LP[[35]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-ece.rice.edu-38)
* [TCP-Illinois](https://en.wikipedia.org/wiki/TCP-Illinois)
* TCP-LP[[35]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-ece.rice.edu-38)
* [TCP SACK](https://en.wikipedia.org/wiki/TCP_SACK)
* [Scalable TCP](https://en.wikipedia.org/wiki/Scalable_TCP)
* TCP Veno[[36]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-39)
* [Westwood](https://en.wikipedia.org/wiki/TCP_Westwood)
* XCP[[37]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-40)
* YeAH-TCP[[38]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-41)
* TCP-FIT[[39]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-42)
* Congestion Avoidance with Normalized Interval of Time (CANIT)[[40]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-43)
* Non-linear neural network congestion control based on genetic algorithm for TCP/IP networks[[41]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-44)

[TCP New Reno](https://en.wikipedia.org/wiki/TCP_congestion_control#TCP_New_Reno) was the most commonly implemented algorithm, SACK support is very common and is an extension to Reno/New Reno. Most others are competing proposals which still need evaluation. Starting with 2.6.8 the Linux kernel switched the default implementation from New Reno to [BIC](https://en.wikipedia.org/wiki/BIC_TCP). The default implementation was again changed to CUBIC in the 2.6.19 version. [FreeBSD](https://en.wikipedia.org/wiki/FreeBSD) uses New Reno as the default algorithm. However, it supports a number of other choices.[[42]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-45)

When the per-flow product of bandwidth and latency increases, regardless of the queuing scheme, TCP becomes inefficient and prone to instability. This becomes increasingly important as the Internet evolves to incorporate very high-bandwidth optical links.

TCP Interactive (iTCP)[[43]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-46) allows applications to subscribe to TCP events and respond accordingly enabling various functional extensions to TCP from outside TCP layer. Most TCP congestion schemes work internally. iTCP additionally enables advanced applications to directly participate in congestion control such as to control the source generation rate.

[Zeta-TCP](https://en.wikipedia.org/wiki/Zeta-TCP) detects the congestions from both the latency and loss rate measures, and applies different congestion window backoff strategies based on the likelihood of the congestions to maximize the [goodput](https://en.wikipedia.org/wiki/Goodput). It also has a couple of other improvements to accurately detect the packet losses, avoiding retransmission timeout retransmission; and accelerate/control the inbound (download) traffic.[[44]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-Zeta-TCP-47)

**Classification by network awareness**

Congestion control algorithms are classified in relation to network awareness, meaning the extent to which these algorithms are aware of the state of the network, and consist of three primary categories: black box, grey box, and green box.[[45]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-48)

Black box algorithms offer blind methods of congestion control. They operate only on the binary feedback received upon congestion and do not assume any knowledge concerning the state of the networks which they manage.

Grey box algorithms use time-instances in order to obtain measurements and estimations of bandwidth, flow contention, and other knowledge of network conditions.

Green box algorithms offer bimodal methods of congestion control which measures the fair-share of total bandwidth which should be allocated for each flow, at any point, during the system's execution.

**Black box**

* [Highspeed-TCP](https://en.wikipedia.org/wiki/Highspeed-TCP)[[46]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-49)
* [BIC TCP](https://en.wikipedia.org/wiki/BIC_TCP) (Binary Increase Congestion Control Protocol) uses a concave increase of the sources rate after each congestion event until the window is equal to that before the event, in order to maximise the time that the network is fully utilised. After that, it probes aggressively.
* [CUBIC TCP](https://en.wikipedia.org/wiki/CUBIC_TCP) – a less aggressive and more systematic derivative of BIC, in which the window is a cubic function of time since the last congestion event, with the inflection point set to the window prior to the event.
* [AIMD-FC](https://en.wikipedia.org/w/index.php?title=AIMD-FC&action=edit&redlink=1) (Additive Increase Multiplicative Decrease with Fast Convergence), an improvement of AIMD.[[47]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-50)
* [Binomial Mechanisms](https://en.wikipedia.org/w/index.php?title=Binomial_Mechanisms&action=edit&redlink=1)
* [SIMD Protocol](https://en.wikipedia.org/w/index.php?title=SIMD_Protocol&action=edit&redlink=1)
* [GAIMD](https://en.wikipedia.org/w/index.php?title=GAIMD&action=edit&redlink=1)

**Grey box**

* [TCP Vegas](https://en.wikipedia.org/wiki/TCP_Vegas) – estimates the queuing delay, and linearly increases or decreases the window so that a constant number of packets per flow are queued in the network. Vegas implements proportional fairness.
* [FAST TCP](https://en.wikipedia.org/wiki/FAST_TCP) – achieves the same equilibrium as Vegas, but uses [proportional control](https://en.wikipedia.org/wiki/Proportional_control) instead of linear increase, and intentionally scales the gain down as the bandwidth increases with the aim of ensuring stability.
* TCP BBR – estimates the queuing delay, but uses exponential increase. Intentionally slows down periodically for fairness and decreased delay.
* [TCP-Westwood](https://en.wikipedia.org/wiki/TCP-Westwood) (TCPW) – a loss causes the window to be reset to the sender's estimate of the bandwidth-delay product, which is the smallest measured RTT times the observed rate of receiving ACKs.[[48]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-51)
* C2TCP[[10]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-C2TCP-13)[[9]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-C2TCP-JSAC-12)
* [TFRC](https://en.wikipedia.org/wiki/TCP_Friendly_Rate_Control)[[49]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-52)
* [TCP-Real](https://en.wikipedia.org/w/index.php?title=TCP-Real&action=edit&redlink=1)
* [TCP-Jersey](https://en.wikipedia.org/w/index.php?title=TCP-Jersey&action=edit&redlink=1)

**Green box**

* [Bimodal Mechanism](https://en.wikipedia.org/w/index.php?title=Bimodal_Mechanism&action=edit&redlink=1) – [Bimodal Congestion Avoidance and Control](https://en.wikipedia.org/w/index.php?title=Bimodal_Congestion_Avoidance_and_Control&action=edit&redlink=1) mechanism.
* Signalling methods implemented by routers
  + [Random Early Detection](https://en.wikipedia.org/wiki/Random_Early_Detection) (RED) randomly drops packets in proportion to the router's queue size, triggering multiplicative decrease in some flows.
  + [Explicit Congestion Notification](https://en.wikipedia.org/wiki/Explicit_Congestion_Notification) (ECN)
* [Network-Assisted Congestion Control](https://en.wikipedia.org/w/index.php?title=Network-Assisted_Congestion_Control&action=edit&redlink=1)
  + NATCP[[11]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-FOOTNOTEAbbaslooXuChaoShi2019-14) - Network-Assisted TCP uses out-of-band explicit feedback indicating minimum RTT of the network and capacity of the cellular access link .
  + [VCP](https://en.wikipedia.org/w/index.php?title=VCP_(congestion_control)&action=edit&redlink=1) – The variable-structure congestion control protocol ([VCP](https://en.wikipedia.org/w/index.php?title=VCP_(congestion_control)&action=edit&redlink=1)) uses two ECN bits to explicitly feedback the network state of congestion. It includes an end host side algorithm as well.

The following algorithms require custom fields to be added to the TCP packet structure:

* [Explicit Control Protocol](https://en.wikipedia.org/w/index.php?title=Explicit_Control_Protocol&action=edit&redlink=1) (XCP) – XCP routers signal explicit increase and decreases in the senders' congestion windows.
* [MaxNet](https://en.wikipedia.org/w/index.php?title=MaxNet&action=edit&redlink=1) – MaxNet uses a single header field, which carries the maximum congestion level of any router on a flow's path. The rate is set as a function of this maximum congestion, resulting in [max-min fairness](https://en.wikipedia.org/wiki/Max-min_fairness).[[50]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-53)
* [JetMax](https://en.wikipedia.org/w/index.php?title=JetMax&action=edit&redlink=1) – [JetMax](https://en.wikipedia.org/w/index.php?title=JetMax&action=edit&redlink=1), like MaxNet, also responds only to the maximum congestion signal, but also carries other overhead fields

**Usage**

* BIC is used by default in Linux kernels 2.6.8 through 2.6.18. (August 2004 – September 2006)
* CUBIC is used by default in Linux kernels since version 2.6.19. (November 2006)
* PRR is incorporated in Linux kernels to improve loss recovery since version 3.2. (January 2012)
* BBR is incorporated in Linux kernels to enable model-based congestion control since version 4.9. (December 2016)

**See also**

* Transmission Control Protocol §§ [Congestion control](https://en.wikipedia.org/wiki/Transmission_Control_Protocol#Congestion_control)​ and [Development](https://en.wikipedia.org/wiki/Transmission_Control_Protocol#Development)
* [Network congestion § Mitigation](https://en.wikipedia.org/wiki/Network_congestion#Mitigation)
* [Low Extra Delay Background Transport](https://en.wikipedia.org/wiki/Low_Extra_Delay_Background_Transport) (LEDBAT)

**Notes**

 In some implementations (e.g., Linux), the initial *ssthresh* is large, and so the first slow start usually ends after a loss. However, *ssthresh* is updated at the end of each slow start, and will often affect subsequent slow starts triggered by timeouts.

  When a packet is lost, the likelihood of duplicate ACKs being received is very high. It is also possible in this case, though unlikely, that the stream just underwent extreme packet reordering, which would also prompt duplicate ACKs.

* 1.  Even if, actually, the receiver may delay its ACKs, typically sending one ACK for every two segments that it receives[[2]](https://en.wikipedia.org/wiki/TCP_congestion_control#cite_note-RFC_2001-2)

**References**

 [Jacobson & Karels 1988](https://en.wikipedia.org/wiki/TCP_congestion_control#CITEREFJacobsonKarels1988).

  *W. Stevens (January 1997).* [*TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*](https://tools.ietf.org/html/rfc2001)*.* [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.17487/RFC2001*](https://doi.org/10.17487%2FRFC2001)*.* [*RFC*](https://en.wikipedia.org/wiki/RFC_(identifier))[*2001*](https://tools.ietf.org/html/rfc2001)*.*

  *M. Allman; S. Floyd; C. Partridge (October 2002).* [*Increasing TCP's Initial Window*](https://tools.ietf.org/html/rfc3390)*.* [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.17487/RFC3390*](https://doi.org/10.17487%2FRFC3390)*.* [*RFC*](https://en.wikipedia.org/wiki/RFC_(identifier))[*3390*](https://tools.ietf.org/html/rfc3390)*.*

  [*"TCP Congestion Avoidance Explained via a Sequence Diagram"*](http://www.eventhelix.com/RealtimeMantra/Networking/TCP_Congestion_Avoidance.pdf) *(PDF). eventhelix.com.*

  *Corbet, Jonathan.* [*"Increasing the TCP initial congestion window"*](https://lwn.net/Articles/427104/)*. LWN. Retrieved 10 October 2012.*

  Nick O'Neill. "[What's Making Your Site Go Slow? Could Be The Like Button](http://allfacebook.com/whats-making-your-site-go-slow-could-be-the-like-button_b24121)". *AllFacebook*, 10 November 2010. Retrieved on 12 September 2012.

  *Chiu, Dah-Ming; Raj Jain (1989). "Analysis of increase and decrease algorithms for congestion avoidance in computer networks". Computer Networks and ISDN Systems.* ***17****: 1–14.* [*CiteSeerX*](https://en.wikipedia.org/wiki/CiteSeerX_(identifier))[*10.1.1.136.8108*](https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.136.8108)*.* [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1016/0169-7552(89)90019-6*](https://doi.org/10.1016%2F0169-7552%2889%2990019-6)*.*

  *Fall, Kevin; Sally Floyd (July 1996).* [*"Simulation-based Comparisons of Tahoe, Reno and SACK TCP"*](https://www.icir.org/floyd/papers/sacks.pdf) *(PDF). Computer Communications Review.* ***26*** *(3): 5–21.* [*CiteSeerX*](https://en.wikipedia.org/wiki/CiteSeerX_(identifier))[*10.1.1.586.2403*](https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.586.2403)*.* [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/235160.235162*](https://doi.org/10.1145%2F235160.235162)*.* [*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*7459148*](https://api.semanticscholar.org/CorpusID:7459148)*.*

  *Abbasloo, S.; Xu, Y.; Chao, H. J. (2019). "C2TCP: A Flexible Cellular TCP to Meet Stringent Delay Requirements". IEEE Journal on Selected Areas in Communications.* ***37*** *(4): 918–932.* [*arXiv*](https://en.wikipedia.org/wiki/ArXiv_(identifier))*:*[*1810.13241*](https://arxiv.org/abs/1810.13241)*.* [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1109/JSAC.2019.2898758*](https://doi.org/10.1109%2FJSAC.2019.2898758)*.* [*ISSN*](https://en.wikipedia.org/wiki/ISSN_(identifier))[*0733-8716*](https://www.worldcat.org/issn/0733-8716)*.* [*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*53107038*](https://api.semanticscholar.org/CorpusID:53107038)*.*

  *Abbasloo, S.; Li, T.; Xu, Y.; Chao, H. J. (May 2018). "Cellular Controlled Delay TCP (C2TCP)". 2018 IFIP Networking Conference and Workshops: 118–126.* [*arXiv*](https://en.wikipedia.org/wiki/ArXiv_(identifier))*:*[*1807.02689*](https://arxiv.org/abs/1807.02689)*.* [*Bibcode*](https://en.wikipedia.org/wiki/Bibcode_(identifier))*:*[*2018arXiv180702689A*](https://ui.adsabs.harvard.edu/abs/2018arXiv180702689A)*.* [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.23919/IFIPNetworking.2018.8696844*](https://doi.org/10.23919%2FIFIPNetworking.2018.8696844)*.* [*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-3-903176-08-9*](https://en.wikipedia.org/wiki/Special:BookSources/978-3-903176-08-9)*.* [*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*49650788*](https://api.semanticscholar.org/CorpusID:49650788)*.*

  [Abbasloo et al. 2019](https://en.wikipedia.org/wiki/TCP_congestion_control#CITEREFAbbaslooXuChaoShi2019).

  [*"BBR: Congestion-Based Congestion Control"*](https://queue.acm.org/detail.cfm?id=3022184)*. ACM Queue.* [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/3012426.3022184*](https://doi.org/10.1145%2F3012426.3022184) *(inactive 24 August 2020). Retrieved 6 December 2016.*

  [Kurose & Ross 2008](https://en.wikipedia.org/wiki/TCP_congestion_control#CITEREFKuroseRoss2008), p. 284.

  [Kurose & Ross 2012](https://en.wikipedia.org/wiki/TCP_congestion_control#CITEREFKuroseRoss2012), p. 277.

  [*"Performance Analysis of TCP Congestion Control Algorithms"*](http://www.wseas.us/journals/cc/cc-27.pdf) *(PDF). Retrieved 26 March 2012.*

  [*"DD-WRT changelog"*](http://www.dd-wrt.com/wiki/index.php/Changelog)*. Retrieved 2 January 2012.*

  *VasanthiN., V.; SinghM., Ajith; Kumar, Romen; Hemalatha, M. (2011). Das, Vinu V; Thankachan, Nessy (eds.). "Evaluation of Protocols and Algorithms for Improving the Performance of TCP over Wireless/Wired Network". International Conference on Computational Intelligence and Information Technology. Communications in Computer and Information Science. Springer.* ***250****: 693–697.* [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1007/978-3-642-25734-6\_120*](https://doi.org/10.1007%2F978-3-642-25734-6_120)*.* [*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-3-642-25733-9*](https://en.wikipedia.org/wiki/Special:BookSources/978-3-642-25733-9)*.*

  [*"Archived copy"*](https://web.archive.org/web/20071011095352/http:/hybla.deis.unibo.it/)*. Archived from* [*the original*](http://hybla.deis.unibo.it/) *on 11 October 2007. Retrieved 4 March 2007.*

  *V., Jacobson; R.T., Braden.* [*TCP extensions for long-delay paths*](https://tools.ietf.org/html/rfc1072)*.* [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.17487/RFC1072*](https://doi.org/10.17487%2FRFC1072)*.* [*RFC*](https://en.wikipedia.org/wiki/RFC_(identifier))[*1072*](https://tools.ietf.org/html/rfc1072)*.*

  *Alrshah, M.A.; Othman, M.; Ali, B.; Hanapi, Z.M. (September 2015). "Agile-SD: A Linux-based TCP congestion control algorithm for supporting high-speed and short-distance networks". Journal of Network and Computer Applications.* ***55****: 181–190.* [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1016/j.jnca.2015.05.011*](https://doi.org/10.1016%2Fj.jnca.2015.05.011)*.* [*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*2645016*](https://api.semanticscholar.org/CorpusID:2645016)*.*

  [*"Proportional Rate Reduction for TCP"*](http://tools.ietf.org/html/rfc6937)*. Retrieved 6 June 2014.*

  *Corbet, Jonathan.* [*"LPC: Making the net go faster"*](https://lwn.net/Articles/458610/)*. Retrieved 6 June 2014.*

  [*"Linux 3.2 - Linux Kernel Newbies"*](http://kernelnewbies.org/Linux_3.2#head-1c3e71416a9fdc2f59c1c251a97963f165302b6e)*. Retrieved 6 June 2014.*

  [*"BBR: Congestion-Based Congestion Control"*](https://research.google.com/pubs/pub45646.html)*. Retrieved 25 August 2017.*

  [*"Delivery Rate Estimation"*](https://tools.ietf.org/html/draft-cheng-iccrg-delivery-rate-estimation-00#section-2.2)*. Retrieved 25 August 2017.*

  [*"TCP BBR congestion control comes to GCP – your Internet just got faster"*](https://cloudplatform.googleblog.com/2017/07/TCP-BBR-congestion-control-comes-to-GCP-your-Internet-just-got-faster.html)*. Retrieved 25 August 2017.*

  [*"BBR congestion control [LWN.net]"*](https://lwn.net/Articles/701165/)*. lwn.net.*

  [*"BBR update"*](https://datatracker.ietf.org/meeting/100/materials/slides-100-iccrg-a-quick-bbr-update-bbr-in-shallow-buffers)*. datatracker.ietf.org.*

  [*"TCP and BBR"*](https://ripe76.ripe.net/presentations/10-2018-05-15-bbr.pdf) *(PDF). Retrieved 27 May 2018.*

  [*"Experimental Evaluation of BBR Congestion Control"*](https://doc.tm.uka.de/2017-kit-icnp-bbr-authors-copy.pdf) *(PDF). Retrieved 27 May 2018.*

  [*"Cellular Controlled Delay TCP (C2TCP)"*](https://wp.nyu.edu/c2tcp/)*. wp.nyu.edu. Retrieved 27 April 2019.*

  *Alrshah, M.A.; Al-Maqri, M.A.; Othman, M. (June 2019).* [*"Elastic-TCP: Flexible Congestion Control Algorithm to Adapt for High-BDP Networks"*](https://doi.org/10.1109/JSYST.2019.2896195)*. IEEE Systems Journal.* ***13*** *(2): 1336–1346.* [*arXiv*](https://en.wikipedia.org/wiki/ArXiv_(identifier))*:*[*1904.13105*](https://arxiv.org/abs/1904.13105)*.* [*Bibcode*](https://en.wikipedia.org/wiki/Bibcode_(identifier))*:*[*2019ISysJ..13.1336A*](https://ui.adsabs.harvard.edu/abs/2019ISysJ..13.1336A)*.* [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1109/JSYST.2019.2896195*](https://doi.org/10.1109%2FJSYST.2019.2896195)*.*

  *Abbasloo, Soheil (3 June 2019),* [*GitHub - Soheil-ab/natcp*](https://github.com/Soheil-ab/natcp)*, retrieved 5 August 2019*

  *Yuan, Cao; Tan, Liansheng; Andrew, Lachlan L. H.; Zhang, Wei; Zukerman, Moshe (5 September 2008). "A generalized FAST TCP scheme". Computer Communications.* ***31*** *(14): 3242–3249.* [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1016/j.comcom.2008.05.028*](https://doi.org/10.1016%2Fj.comcom.2008.05.028)*.* [*hdl*](https://en.wikipedia.org/wiki/Hdl_(identifier))*:*[*1959.3/44051*](https://hdl.handle.net/1959.3%2F44051)*.*

  [*"Rice Networks Group"*](http://www.ece.rice.edu/networks/TCP-LP/)*.*

  [*"TCP Veno: TCP Enhancement for Transmission over Wireless Access Networks"*](https://www.ie.cuhk.edu.hk/fileadmin/staff_upload/soung/Journal/J3.pdf) *(PDF). IEEE Journal on Selected Areas in Communication.*

  [*"XCP @ ISI"*](http://www.isi.edu/isi-xcp/)*.*

  [*"High speed TPC"*](http://www.csc.lsu.edu/~sjpark/cs7601/4-YeAH_TCP.pdf) *(PDF). www.csc.lsu.edu.*

  [*"Archived copy"*](https://web.archive.org/web/20110403142334/http:/media.cs.tsinghua.edu.cn/~multimedia/tcp-fit/)*. Archived from* [*the original*](http://media.cs.tsinghua.edu.cn/~multimedia/tcp-fit/) *on 3 April 2011. Retrieved 5 March 2011.*

  *Benaboud, H.; Berqia, A.; Mikou, N. (2002). "An analytical study of CANIT algorithm in TCP protocol". ACM Sigmetrics Performance Evaluation Review.* ***30*** *(3): 20.* [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/605521.605530*](https://doi.org/10.1145%2F605521.605530)*.* [*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*6637174*](https://api.semanticscholar.org/CorpusID:6637174)*.*

  *Rouhani, Modjtaba (2010). "Nonlinear Neural Network Congestion Control Based on Genetic Algorithm for TCP/IP Networks". 2010 2nd International Conference on Computational Intelligence, Communication Systems and Networks. pp. 1–6.* [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1109/CICSyN.2010.21*](https://doi.org/10.1109%2FCICSyN.2010.21)*.* [*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-1-4244-7837-8*](https://en.wikipedia.org/wiki/Special:BookSources/978-1-4244-7837-8)*.* [*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*15126416*](https://api.semanticscholar.org/CorpusID:15126416)*.*

  [*"Summary of Five New TCP Congestion Control Algorithms Project"*](http://forums.freebsd.org/showthread.php?t=22396)*.*

  [*"iTCP - Interactive Transport Protocol - Medianet Lab, Kent State University"*](http://www.medianet.kent.edu/itcp/main.html)*.*

  [*"Whitepaper: Zeta-TCP - Intelligent, Adaptive, Asymmetric TCP Acceleration"*](http://www.appexnetworks.com/Assets/PDF/ZetaTCP.pdf) *(PDF). Retrieved 6 December 2019.*

  *Lefteris Mamatas; Tobias Harks; Vassilis Tsaoussidis (January 2007).* [*"Approaches to Congestion Control in Packet Networks"*](https://web.archive.org/web/20140221123729/http:/utopia.duth.gr/~emamatas/jie2007.pdf) *(PDF). Journal of Internet Engineering.* ***1*** *(1). Archived from* [*the original*](http://utopia.duth.gr/~emamatas/jie2007.pdf) *(PDF) on 21 February 2014.*

  [*"HighSpeed TCP"*](http://www.icir.org/floyd/hstcp.html)*. www.icir.org.*

  [*"AIMD-FC Homepage"*](https://web.archive.org/web/20090113204941/http:/www.ccs.neu.edu/home/ladrian/abstract/aimdfc.html)*. neu.edu. Archived from* [*the original*](http://www.ccs.neu.edu/home/ladrian/abstract/aimdfc.html) *on 13 January 2009. Retrieved 13 March 2016.*

  [*"Welcome to Network Research Lab"*](http://www.cs.ucla.edu/NRL/hpi/tcpw/)*. www.cs.ucla.edu.*

  [*"Equation-Based Congestion Control for Unicast Applications"*](http://www.icir.org/tfrc/)*. www.icir.org.*

* 1.  [*"MaxNet -- Max-Min Fair, Stable Explicit Signalling Congestion Control"*](http://netlab.caltech.edu/maxnet/)*. netlab.caltech.edu.*

**Sources**

* [*Kurose, James*](https://en.wikipedia.org/wiki/Jim_Kurose)*; Ross, Keith (2008). Computer Networking: A Top-Down Approach (4th ed.). Addison Wesley.* [*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-13-607967-5*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-13-607967-5)*.*
* [*Kurose, James*](https://en.wikipedia.org/wiki/Jim_Kurose)*; Ross, Keith (2012). Computer Networking: A Top-Down Approach (6th ed.). Pearson.* [*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-13-285620-1*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-13-285620-1)*.*
* *Abbasloo, Soheil; Xu, Yang; Chao, H. Jonathon; Shi, Hang; Kozat, Ulas C.; Ye, Yinghua (2019).* [*"Toward Optimal Performance with Network Assisted {TCP} at Mobile Edge"*](https://www.usenix.org/conference/hotedge19/presentation/abbasloo)*. Renton, WA: USENIX Association.*
* *Afanasyev, A.; N. Tilley; P. Reiher; L. Kleinrock (2010).* [*"Host-to-host congestion control for TCP"*](http://lasr.cs.ucla.edu/afanasyev/data/files/Afanasyev/Host-to-host%20congestion%20control%20for%20TCP.pdf) *(PDF). IEEE Communications Surveys and Tutorials.* ***12*** *(3): 304–342.* [*CiteSeerX*](https://en.wikipedia.org/wiki/CiteSeerX_(identifier))[*10.1.1.228.3080*](https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.228.3080)*.* [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1109/SURV.2010.042710.00114*](https://doi.org/10.1109%2FSURV.2010.042710.00114)*.* [*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*8638824*](https://api.semanticscholar.org/CorpusID:8638824)*.*
* [*Jacobson, Van*](https://en.wikipedia.org/wiki/Van_Jacobson)*;* [*Karels, Michael J.*](https://en.wikipedia.org/wiki/Michael_J._Karels) *(November 1988).* [*"Congestion avoidance and control"*](http://ee.lbl.gov/papers/congavoid.pdf) *(PDF). ACM SIGCOMM Computer Communication Review.* ***18*** *(4): 314–329.* [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/52325.52356*](https://doi.org/10.1145%2F52325.52356)*.*

**External links**

* [*Approaches to Congestion Control in Packet Networks*](https://web.archive.org/web/20140221123729/http:/utopia.duth.gr/~emamatas/jie2007.pdf) *(PDF), archived from* [*the original*](http://utopia.duth.gr/~emamatas/jie2007.pdf) *(PDF) on 21 February 2014*
* [Papers in Congestion Control](http://www.shivkumar.org/research/cong-papers.html)
* [TCP Vegas Homepage](http://www.cs.arizona.edu/projects/protocols/)
* *Allman, Mark; Paxson, Vern; Stevens, W. Richard (April 1999).* [*"Fast Retransmit/Fast Recovery"*](https://tools.ietf.org/html/rfc2581#section-3.2)*.* [*TCP Congestion Control*](https://tools.ietf.org/html/rfc2581)*.* [*IETF*](https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force)*. sec. 3.2.* [*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.17487/RFC2581*](https://doi.org/10.17487%2FRFC2581)*.* [*RFC*](https://en.wikipedia.org/wiki/RFC_(identifier))[*2581*](https://tools.ietf.org/html/rfc2581)*. Retrieved 1 May 2010.*
* [TCP Congestion Handling and Congestion Avoidance Algorithms](http://www.tcpipguide.com/free/t_TCPCongestionHandlingandCongestionAvoidanceAlgorit-3.htm) – The TCP/IP Guide