International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 05 | May - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

# UNDERSTANDING THE TRADE-OFF IN TCP'S CUBIC WINDOWS CONGESTION: A COMPREHENSIVE ANALYSIS

AYUSH KACHHAP
MCA Department
*School of CS and IT*
Jain (Deemed-to-be-University)
Bengaluru, India.
ayushjustinkachhap@gmail.com

BHUVANA J
Assistant Professor
*School of CS and IT*
Jain (Deemed-to-be-University)
Bengaluru, India.
j.bhuvana@jainuniversity.ac.in

## Abstract

*This study investigates the difficulties of the CUBIC TCP algorithm, focusing on the trade-off between efficiency and fairness in window congestion control. The CUBIC approach, which was devised to optimize high-bandwidth and long-distance networks, is currently widely used as the default TCP congestion control algorithm in Linux and other systems.*

*Our work investigates the dynamics of CUBIC's window expansion, specifically the interaction between the cubic function and the most recent congestion event. We look at how this interplay impacts the distribution of window widths, throughput, round-trip time (RTT), loss rate, and queue size under different network configurations.*

***Keywords***: *TCP CUBIC, Window Congestion, Efficiency-Fairness Trade-Off, TCP's Cubic Function.*

## 1) Introduction

The CUBIC Windows Congestion algorithm is a TCP congestion control technique developed to improve TCP performance in high-speed networks. Ha, Rhee, and Xu proposed it in 2008. The approach is based on the idea of utilizing a cubic function to alter the congestion window (*cwnd*) depending on the round-trip time (RTT) and the amount of data in transit.

The CUBIC algorithm utilizes available bandwidth faster than TCP Reno while maintaining fairness to other TCP connections. The system achieves this by drastically boosting the *cwnd* at first, then progressively reducing its increase as it approaches the preceding challenge.

The key distinction between CUBIC and other TCP variations is how it increases its rate during congestion avoidance. When a CUBIC sender experiences packet loss while its *cwnd* is W, it reduces its *cwnd* by half. The *cwnd* is then significantly increased at first, but progressively slows as it approaches W. If the *cwnd* surpasses W without packet loss, CUBIC will steadily increase the *cwnd*.

The CUBIC algorithm is widely used and has been included into numerous operating systems and network devices. However, it has been chastised for inefficiencies such as delayed startup and congestion avoidance phases, which can result in inferior performance in certain network conditions.

The usual TCP congestion control solution employs the additive increase/multiplicative decrease (AIMD) strategy, which combines linear growth of the congestion window (*cwnd*) with exponential reduction during congestion.
This method attempts to avoid network congestion by dynamically adjusting the *cwnd* based on the level of congestion. When congestion is identified, the *cwnd* is reduced exponentially to prevent additional congestion. This procedure is repeated until the congestion is alleviated and the *cwnd* is increased again.

CUBIC TCP is a high-speed TCP variation that adjusts the *cwnd* based on RTT and data in transit. CUBIC TCP is expected to use available bandwidth faster than TCP Reno, while being fair to other TCP connections. The system achieves this by drastically boosting the *cwnd* at first, then gradually reducing its increase as it approaches the previous trouble point. Traditional TCP congestion control has a linear growth method, while CUBIC TCP employs a cubic function. This enables CUBIC TCP to take over available bandwidth faster and adapt to changing network conditions.

In essence, the CUBIC Windows Congestion algorithm is a TCP congestion control technique designed to enhance TCP performance in high-speed networks. It is based on the concept of employing a cubic function to change the *cwnd* in response to the RTT and amount of data in transit. The technique is extensively useful, however it has been criticized for being inefficient, prompting the development of alternative algorithms aimed at improving network performance and dependability.

## 2) Overview

TCP (transmission Control Protocol) Congestion control is a fundamental strategy for avoiding network congestion and ensuring reliable data delivery over the Internet. The additive increase/multiplicative decline (AIMD) strategy is the standard TCP congestion control technique. It combines linear expansion of the congestion window with exponential reduction when congestion arises.

Figure (2): The congestion window (*cwnd*) is a vital component of this approach, determining the maximum amount of data that can be sent before receiving an acknowledgement from the receiver. The *cwnd* is initially set to a modest multiple of the connection's maximum segment size (MSS) and then modified in accordance with the AIMD approach. When a connection is established, the sender initializes the *cwnd* to the MSS and then increments it by one.

The basic TCP congestion control approach is the additive increase/multiplicative decline (AIMD) technique. This technique combines linear expansion of the congestion window (*cwnd*) with exponential contraction as congestion increases.

The *cwnd* is initially set to a modest multiple of the connection's maximum segment size (MSS), which is then changed using the AIMD approach.

When a connection is established, the sender initializes the *cwnd* to the MSS, which is then increased by one MSS after each acknowledgment. This procedure continues until the *cwnd* exceeds a certain threshold, at which time the algorithm transitions to the congestion avoidance phase. During this phase, the *cwnd* increases linearly by one MSS for each acknowledgment received until it meets the receiver's window size.

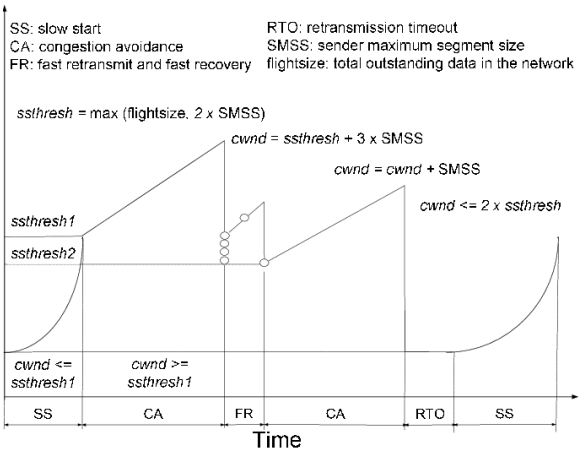


*Figure 2: TCP Congestion Window Growth*

### 2.1) Algorithm of CUBIC Congestion Control

TCP CUBIC improves high-bandwidth networks by controlling congestion window growth via a cubic function. The idea is to swiftly increase transmission rates without overloading the network. CUBIC's congestion window expansion is controlled by a cubic function, resulting in smoother and more aggressive growth, particularly on high-speed lines. The growth process consists of a concave phase in which the window quickly reaches the size before to the last congestion episode, followed by a convex phase in which CUBIC hunts for additional bandwidth.

When congestion is identified, the window size is recorded as *Wmax*, which is the greatest size before the last reduction. It then continues transmission with a lower window size, progressively increasing it based on the cubic function's concave and convex sections. It optimizes the cubic increase function by adjusting the scaling factor based on connection time, making it suitable for long-term connections and changing round-trip times.

**Algorithm**

HyStart, a new slow-start algorithm used in TCP CUBIC, improves start-up throughput in long-distance and high-bandwidth networks by avoiding extended burst losses by determining a safe escape point during slow-start. As discussed, *Wmax* is registered in any lose event that occurs after a window size decrease, i.e. decreasing *cwnd* by β (constant decrease factor), resulting in congestion avoidance pharse.

And again, window size starts to increase from zero using concave function (Section 4.3) of CUBIC function until it becomes *Wmax*.

The window growth function of CUBIC:

$$W(t) = C\,(t - K)^3 + Wmax \quad \text{---(1)}$$

where C is a CUBIC parameter,
t: The elapsed time from the last window size reduction, and
K: The time period that the above function takes to increase W to *Wmax* when there is no subsequent loss event.

When there is no further loss events occurrence, K is calculated by using equation:

$$K = \sqrt[3]{Wmax\beta/C} \quad \text{---(2)}$$

While function requires to increase W to *Wmax*, setting *cwnd* as (W (t + RRT)) resulting the *cwnd* growth shown in figure 2.1.

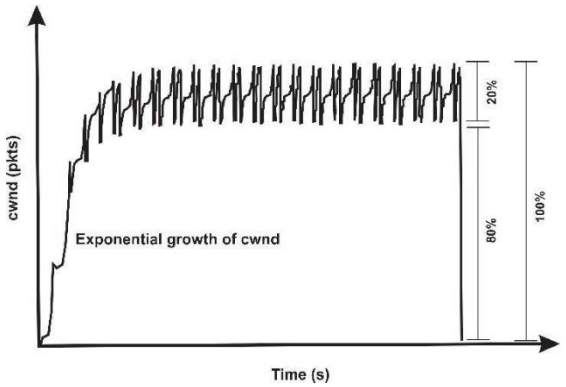


*Figure 2.1: TCP CUBIC Congestion Windows Growth.*

## 3) Comparison

### 3.1) TCP Tahoe

TCP Tahoe is an early version of TCP congestion management that incorporates the slow start, congestion avoidance, and fast retransmission algorithms. It was intended to prevent and recover from network congestion.

Mechanism used:

**Slow start**

Starts with a modest congestion window (*cwnd*). *cwnd* is doubled for each round-trip time (RTT) until it hits the slow start threshold (*SSTHRESH*). Allows *cwnd* to expand exponentially at the start

**Congestion Avoidance**

Begins when *cwnd* reaches the SS threshold. Increases *cwnd* by one MSS per RTT (additive increase). Allows *cwnd* to increase linearly.

**Fast retransmit**

Triggered by three duplicate ACKs. Retransmits a lost packet without waiting for a timeout. Reduces *ssthresh* to *cwnd*/2, resets *cwnd* to 1 MSS.

When a packet loss is detected via a timeout, TCP Tahoe enters slow start again, setting *cwnd* to 1 MSS and *ssthresh* to *cwnd*/2. This allows it to quickly recover from heavy congestion.

Figure (3.1), explains: TCP CUBIC, on the other hand, is optimized for high-speed networks and employs a cubic function to change the congestion window based on round-trip time and data in transit. CUBIC is believed to be more proactive in expanding window size following a congestion occurrence than Tahoe. It is also less aggressive in increasing window size as it nears the ideal place. CUBIC is more scalable and efficient for high-speed networks with high bandwidth delay product (BDP) and packet loss rates. Unlike TCP Tahoe, which was an early version that used basic congestion control techniques, CUBIC is a more complex algorithm that supports efficient and equitable resource allocation.

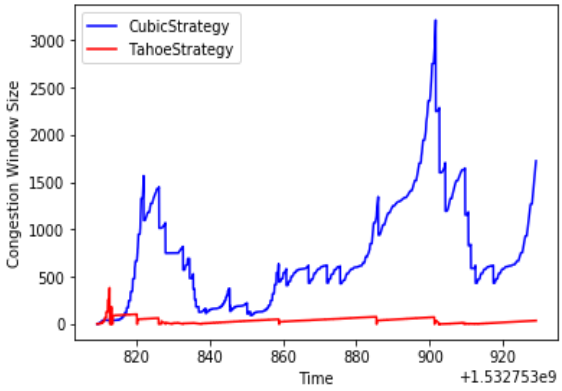


*Figure 3.1: Congestion Window Growth of TCP's CUBIC and Tahoe.*

### 3.2) New TCP Reno

TCP New Reno is an enhanced version of the TCP Reno congestion control algorithm that overcomes the performance difficulties discovered in TCP Reno. It incorporates improvements to circumvent TCP Reno limits and improve network performance.

TCP New Reno has increased its swift recovery capabilities. Unlike TCP Reno, TCP New Reno remains in the rapid recovery phase until all missing packets are acknowledged. This enhancement enables TCP New Reno to recover more quickly from packet losses while maintaining efficient data transfer rates.

Figure (3.2) TCP New Reno keeps the key characteristics of TCP Reno while introducing modifications to improve its performance in modern network situations. It outperforms TCP Reno in cases with repeated packet drops, ensuring faster throughput. A study suggested U-New Reno, a transmission control protocol that uses an algorithm to find the optimal congestion window size for underwater wireless sensor networks. This indicates that continuous research is being conducted to tailor TCP New Reno to specific network conditions. TCP New Reno builds on the foundation of TCP Reno, improving its speedy recovery mechanism. This allows it to better tolerate packet losses while maintaining improved performance in specific network conditions. The algorithm is still undergoing continual research and improvement.

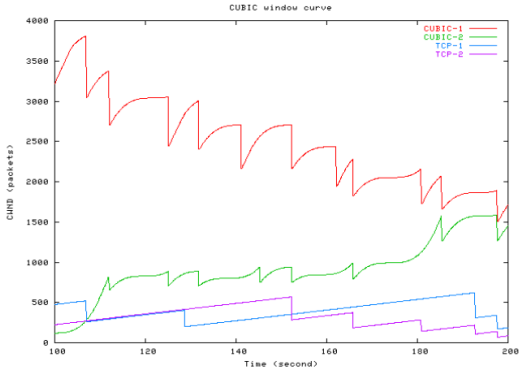


*Figure 3.2: CUBIC window curve*

### 3.3) BBR (Bottleneck Bandwidth RTT)

TCP BBR (Bottleneck Bandwidth and Round-Trip Time) is a congestion control method created by Google that tries to maximize delivery rate while minimizing latency. Unlike classic loss-based congestion control algorithms, BBR estimates bottleneck connection bandwidth and round-trip time (RTT) to determine sending rate.

The main principles of BBR are:
To measure bottleneck bandwidth, track the maximum delivery rate across a 10-second sliding frame. Using the minimal RTT to estimate the round-trip propagation time. Pacing the transmitting pace to approach the expected bandwidth. Keeping the amount of data in flight close to the expected bandwidth-delay product.

The main principles of BBR are:
To measure bottleneck bandwidth, track the maximum delivery rate across a 10-second sliding frame. Using the minimal RTT to estimate the round-trip propagation time.
Pacing the transmitting pace to approach the expected bandwidth
Keeping the amount of data in flight close to the

expected bandwidth-delay product. BBR operates in four major states: STARTUP: Exponential growth to fast fill the pipe; stops when the bandwidth estimate plateaus. DRAIN: Draining the queue generated during STARTUP.
PROBE_BW: Cycling the pacing increase to explore and evenly share bandwidth. PROBE_RTT: Sending slowly to probe the minimal RTT.

Figure (3.3): TCP CUBIC uses a cubic function to adjust the congestion window (*cwnd*) based on the time since the last congestion event. It detects congestion by packet loss, resulting in a drop in the *cwnd*. It is also famous for its fairness and scalability across a wide range of network circumstances.

However, BBR employs a more proactive approach, determining the transmission rate using the available bottleneck connection bandwidth and RTT. BBR, on the other hand, employs a model-based approach to controlling the sending rate while keeping the amount of data in flight near to the bandwidth-delay ratio. And has been shown to have significantly higher throughput and lower latency than CUBIC, especially in high-speed networks with packet loss.

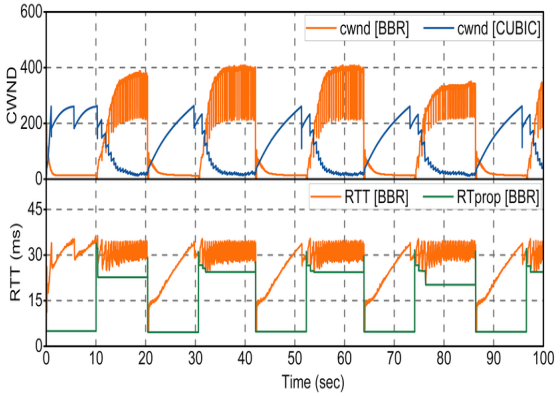


*Figure 3.3: Congestion windows and RRT in BBR and CUBIC*

### 3.4) TCP Vegas

TCP Vegas, another variation of the TCP congestion control algorithm that intend to improve network performance by more accurately analysing, detecting and responding to network congestion. TCP Vegas adjusts the congestion window more accurately than traditional TCP algorithms like Reno or CUBIC by evaluating the connection's round-trip time (RTT) and real throughput.

In TCP Vegas, the sender continuously examines packet RTTs and adjusts the congestion window accordingly. If the RTT becomes too large, indicating network congestion, the sender reduces the congestion window, which slows data transmission. TCP Vegas enhances network resource sharing stability and fairness when compared to traditional TCP methods.

It is not commonly utilized in production networks, but its concepts have impacted the development of alternative congestion control methods. TCP Vegas's concept of using end-to-end measurements to detect and respond to network congestion has been borrowed by other algorithms such as TCP New Reno and TCP Bic, highlighting its importance in computer network technology.

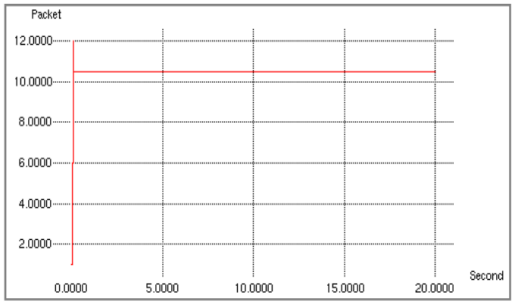


*Figure (3.4) Congestion Window Growth of TCP Vegas*

CUBIC uses packet loss as the main indicator of congestion. It is more aggressive in increasing the *cwnd* to utilize available bandwidth. Itis generally fair towards other TCP flows and can fully utilize available bandwidth.

Vegas on the other hand, uses increasing RTT values to detect congestion before packet loss occurs. It is less aggressive and aims to maintain the actual throughput close to the expected throughput. It may not co-exist well with other congestion control algorithms because it is the fastest to detect and throttle congestion. It may not be able to fully utilize the available bandwidth when competing with AIMD mechanisms.

To recap, CUBIC is a more aggressive, loss-based algorithm that seeks to maximize throughput, whereas Vegas is a delay-based algorithm that aims to avoid congestion while retaining low latency. The decision

between the two is made depending on the specific network conditions and requirements.

## 4) Discussion

### 4.1) Trade-Off in Cubic Congestion Control: Fairness

CUBIC is designed to be more aggressive in raising the congestion window size, taking advantage of high-bandwidth networks and optimizing throughput. However, this aggression might raise fairness concerns, particularly when competing with flows using different congestion control techniques. CUBIC's rapid window growth may affect the fairness of bandwidth allocation for various flows. It's rapid congestion window growth can overestimate available bandwidth, leading to network congestion concerns.

It is more complicated than standard TCP algorithms, making implementation more difficult and requiring additional network design considerations. It is intended to manage network dynamics effectively, but its performance can be influenced by specific circumstances, resulting in different efficiency across scenarios.

Adoption of CUBIC may be problematic due to compatibility issues with existing network equipment and devices, especially when switching from other TCP congestion control approaches.

Sangtae Ha, Injong Rhee, and Lisong Xu's research article "CUBIC: A New TCP-Friendly High-Speed TCP Variant" and Rodolfo I. Ledesma Goyzueta, Yu Chen's research article "A Deterministic Loss Model Based Analysis of CUBIC" discusses aggressiveness and fairness to other flows. According to the study, CUBIC is found to be more aggressive in expanding the congestion window in order to capitalize on high-speed networks.

The technique employs a cubic function to drive window expansion, resulting in faster convergence to optimal transmission rate than previous algorithms such as Reno. However, this hostility has a negative impact on fairness. According to the study, CUBIC's window growth is purely determined by the time since the last congestion event, as opposed to the round-trip time (RTT) used in previous algorithms.

This RTT independence enables CUBIC to be more equitable across flows of various RTTs. However, CUBIC is more aggressive than other algorithms, which can result in lesser bandwidth in some cases.

The study concludes that the trade-off between aggression and fairness is a design choice in CUBIC. The method prioritizes optimal performance in high-bandwidth networks over complete fairness in all scenarios.

To summarize, the key trade-off in CUBIC is that it compromises some fairness, particularly for less aggressive congestion control algorithms, in order to get higher throughput and faster convergence to optimal transmission rates in high-speed networks. This design decision is thoroughly described in the study paper mentioned above.

### 4.2) Understanding Cubic Window Size Functions

CUBIC registers *Wmax* as the window size where the loss event occurred and performs a multiplicative decrease of the congestion window by a factor of β, where β is a window decrease constant and the regular fast recovery and retransmission of TCP. After entering congestion avoidance mode from fast recovery, it starts to increase the window by applying the cubic function's concave profile. The cubic function is set to plateau at *Wmax*, therefore concave growth will continue until the window size reaches *Wmax*. Following that, the cubic function is transformed into a convex profile, and convex window growth begins. This type of window modification like concave and then convex, enhances protocol and network stability while maintaining high network use.

### 4.3) Region in CUBIC

In the context of TCP CUBIC congestion control, the concept of regions is critical in deciding algorithm behavior based on congestion window size and conventional TCP window size at any given time. In the TCP-friendly zone, denoted by *cwnd* < WTCP(t), where *cwnd* is the current congestion window size and WTCP(t) is the standard TCP window size at time t, CUBIC operates in a friendly manner to standard TCP.

When CUBIC is in the TCP-friendly region, the congestion window size (*cwnd*) is set to the standard TCP window size (WTCP(t)) with each acknowledgment (ACK).

If CUBIC is not in the TCP-friendly region and *cwnd* > WTCP(t), the protocol falls into the concave region. In this region, *cwnd* rises according to Eq. (1) until *Wmax* is attained. When *cwnd* exceeds *Wmax*, the protocol is in the convex region. The window growth function is the same for concave and convex sections.

The initial growth rate is sluggish near the *Wmax* neighborhood.

**4.4) Fast Convergence mechanism**

CUBIC adds a heuristic to the protocol to increase convergence rate. This heuristic allows CUBIC flows to share bandwidth with incoming flows. Increased bandwidth allows for more incoming flows. When fast convergence is enabled, the method compares the prior and current *Wmax* after a packet loss event. If the current *Wmax* is lower than the prior one, it means the link has fewer available resources.

Algorithm:

if (*Wmax* < *W_last_max*) {
*W_last_max* = *Wmax*;
*Wmax*=*Wmax**(2-β)/2;
} else
*W_last_max* = *Wmax*;

Here,
*Wmax* (integer) is the current maximum window size,
*W_last_max* (integer) is the last maximum value registered, and
β is the multiplicative decrease factor.

## 5) Experimental analysis

From section 2.1, 4.3, 4.4: As K, the average period, and the window size rises from $W_i$ to $W_{i+1}$ until it reaches its maximum (*Wmax* = ω). Furthermore, CUBIC's extreme overestimation of bandwidth may lead to packet loss. We know that, CUBIC is not dependent on RRT, although it does extend the time required to grow the window size.

Using the Equation (2) and (1) the growth function for dynamic RRT:

$$W(t) = C\left(tRTT - \sqrt[3]{\frac{\omega\beta}{C}}\right)^3 + \omega$$

And the expected Average Window Size, taking the RRT as the time unit until the packet lose event occurs:

$$E[W_{CUBIC}] = \sqrt[4]{C\left(\frac{RTT}{P}\right)^3\left(\frac{4-\beta}{\beta}\right)}$$

The relationship between parameters K and C varies with packet loss rates p. In this network context, CUBIC performs better as a congestion control protocol, hence a delay of RTT=100 ms was used. For example, for $p=10^{-8}$and C=0.04, the anticipated time to reach *Wmax* is K=99 s, excluding additional packet loss events. Similarly, with $p=10^{-8}$and C=0.4, the default CUBIC value, K=55 s is achieved. For a constant value of C, aggressive window size growth rates result in a decreased predicted time K.

## 6) Conclusion

The primary trade-off in TCP CUBIC congestion control is striking a balance between aggressiveness in expanding the congestion window size and fairness to other flows. CUBIC is expected to be more aggressive in increasing the congestion window size in order to exploit high-bandwidth networks and maximize throughput.
However, this aggression might raise fairness concerns, particularly when competing with flows using different congestion control techniques. CUBIC's rapid window growth could affect the fairness of bandwidth distribution among distinct flows.
CUBIC's rapid congestion window growth can overestimate available bandwidth, leading to network congestion concerns.

Overall, the experimental research demonstrates the dynamic nature of CUBIC congestion control, its independence from RTT, the effect of parameter modifications on performance, and the significance of aggressive growth rates in enhancing network efficiency and congestion management.

## 7) Reference

[1] CUBIC: A New TCP-Friendly High-Speed TCP Variant

Sangtae Ha, Injong Rhee Dept of Computer Science North Carolina State University Raleigh, NC 27695 sha2, rhee@ncsu.edu

Lisong Xu, Dept of Comp. Sci. and Eng. University of Nebraska Lincoln, Nebraska 68588 xu@cse.unl.edu

https://www.cs.princeton.edu/courses/archive/fall16/cos561/papers/Cubic08.pdf

[2] A Deterministic Loss Model Based Analysis of CUBIC

Rodolfo I. Ledesma Goyzueta, Yu Chen

Department of Electrical and Computer Engineering, Binghamton University, Binghamton, NY 13902

Email: rledesm1@binghamton.edu,

ychen@binghamton.edu

https://bingweb.binghamton.edu/~ychen/CUBIC_ICNC2013_CameraReady.pdf

[3] A Survey on Performance of Congestion Control Mechanisms for Standard TCP Versions

Ghassan A. Abed, Mahamod Ismail and Kasmiran Jumari

Faculty of Engineering and Built Environment, Universiti Kebangsaan Malaysia 43600 UKM, Bangi, Selangor, Malaysia.

https://www.researchgate.net/publication/256868797_A_Survey_on_Performance_of_Congestion_Control_Mechanisms_for_Standard_TCP_Versions

[4] EXPERIMENTAL EVALUATION OF TCP CONGESTION CONTORL MECHANISMS IN SHORT AND LONG DISTANCE NETWORKS

Mudassar Ahmad, Md Asri Ngadi, Mohd Murtadha Mohamad

Department of Computer Science, Faculty of Computing,

Universiti Teknologi Malaysia (UTM), 81310 Johor, Malaysia.

E-mail : mudassar.utm@gmail.com,

dr.asri@utm.my@utm.my,

murtadha@utm.my

https://www.researchgate.net/publication/280976972_Experimental_evaluation_of_TCP_congestion_contorl_mechanisms_in_short_and_long_distance_networks

[5] CUBIC TCP

https://en.wikipedia.org/wiki/CUBIC_TCP

[6] CUBIC vs. Tahoe

https://squidarth.com/cubic-vs-tahoe

[7] Newer TCP Implementations

http://intronetworks.cs.luc.edu/1/html/newtcps.html

[8] What are the pros and cons of using TCP Reno vs TCP Cubic in high-speed networks?

https://www.linkedin.com/advice/0/what-pros-cons-using-tcp-reno-vs-cubic

[9] Bennouri Hajar, Berqia Amine, Patrick N'Guessan Koffi. "U-New Reno transmission control protocol to improve TCP performance in Underwater Wireless Sensors Networks." General Computer Science. 09/01/2022.

https://www.sciencedirect.com/science/article/pii/S1319157821002081

[10] Performance Analysis of BBR Congestion Control Protocol Based on NS3

Hao Zhang, Haiting Zhu, Yu Xia, Lu Zhang, Yuan Zhang,Yingying Deng

School of Internet of Things, Nanjing University of Posts and Telecommunications, Nanjing, China

https://ieeexplore.ieee.org/document/8916538

[11] TCP Congestion Control Algorithms Comparison

https://www.speedguide.net/articles/tcp-congestion-control-algorithms-comparison-7423

[12] TCP BBR congestion control comes to GCP – your Internet just got faster

July 21, 2017

Neal Cardwell, Yuchung Cheng

Senior Staff Software Engineer

https://cloud.google.com/blog/products/networking/tcp-bbr-congestion-control-comes-to-gcp-your-internet-just-got-faster

[13] Article: BBR-CWS: Improving the Inter-Protocol Fairness of BBR

Yeong-Jun Song, Geon-Hwan Kim and You-Ze Cho

School of Electronics Engineering, Kyungpook National University, Daegu 41556, Korea; syj5385@knu.ac.kr (Y.-J.S.); kgh76@ee.knu.ac.kr (G.-H.K.)

* Correspondence: yzcho@ee.knu.ac.kr

Received: 24 April 2020; Accepted: 14 May 2020; Published: 22 May 2020

[14] Net100's TCP Vegas

https://tnlandforms.us/ornlwww/netperf/vegas.html