Networks Project Report

TCP Congestion Control
by CUBIC Algorithm

Prepared by :

- Anurag Goyal - 106119014
- Tharun A. - 106119134

https://github.com/anuragnitt/cubic-congestion-control

Abstract

Traditional AIMD (additive increase multiplicative decrease) based TCP congestion control algorithms, for eg. Tahoe and Reno, suffer from long response times in fast and long distance networks.

CUBIC is a TCP congestion control algorithm which overcomes this disadvantage and is used in the Linux and macOS operating systems.

We implemented a Linux kernel module which uses the CUBIC algorithm but also exposes certain control parameters of the algorithm to an interactive GUI which allows users to tweak their systems' TCP congestion control behaviour in realtime.

Users are able to control the rate at which the sender's congestion window grows and the multiplicative factor to decrease the congestion window on a loss event.

Further, we have implemented a modified version of the CUBIC algorithm known as Multipath CUBIC or mpCUBIC. A TCP connection might involve multiple subflows, i.e, a multipath TCP (MPTCP) connection. The congestion window should not increase too quickly because of the subflows increasing their respective congestion windows.

When CUBIC is the TCP congestion control algorithm, the throughputs of MPTCP flows can decrease in the traditional algorithm. mpCUBIC is a MPTCP congestion control scheme which overcomes this disadvantage.

Introduction

a. Importance of congestion control

Congestion occurs in a network when a link is handling more data than its capacity. This causes problems such as queueing delay and packet loss.

Communication protocols such as TCP restransmit lost/dropped packets which can further increase congestion.

Congestion control is the process to reduce network congestion. This is achieved by signalling the sender to reduce the rate of sending packets into the network by changing its congestion window size.

There are several congestion control algortihms such as Vegas, Veno, Westwood, Tahoe and Reno and all of these use an AIMD (additive increase multiplicative decrease) based approach.

b. Problem with AIMD based congestion control algorithms

A congestion control algorithm tries to achieve full utilization of a bottleneck link and a fair allocation bandwidth simultaneously.

Traditional AIMD based algorithms do not perform well in fast and long distance networks because of their large response times.

AIMD based growth function has a strict constant increase in the congestion window size when an ACK is received. It always tries to reach the maximum allowed congestion window size.

This results in TCP flows with shorter RTT (round trip time) growing faster than flows with longer RTT.

c. CUBIC – an alternative approach

CUBIC is a fast congestion control algorithm which supports fast transfer of large volumes of data.

On receving an ACK, CUBIC finds a target window size and the congestion window then grows towards that target.

The congestion window size as a function of t (time elapsed since the last congestion window reduction) and W_{max} (congestion window size when the last packet loss happened) is given by :

$$W(t) = C(t - K)^3 + W_{max}$$

CUBIC increases the window size aggressively when the window is far from the saturation point and slowly when it is close to the saturation point. This feature allows CUBIC to be very scalable when the bandwidth and delay product of the network is large and at the same time, be highly stable and also fair to standard TCP flows.

d. Outline of the paper

The paper has presented a mathematical analysis of the CUBIC algorithm along with the assumptions taken to implement it as a Linux kernel module.

The kernel module's special feature is that it exposes certain control parameters of the CUBIC congestion window function to user space through a GUI.

Users can then use the GUI to manipulate those parameters which would affect the behaviour of TCP congestion control happening in their operating system in realtime.

A simulation environment is used to create a bottleneck link and multiple TCP flows are started which share that bottleneck link. Captured data is visualized to analyze congestion window size, throughput and queueing delays.

Related Works

1. TCP fairness among modern TCP congestion control algorithms including TCP BBR

Authors have analyzed performance fairness between CUBIC TCP and TCP BBR. Performance is evaluated in conditions wherein connections of TCP BBR and CUBIC TCP are concurrently established. Demonstrated that the performance fairness between TCP BBR and CUBIC TCP is very low, especially with high latency. In the case of 64 ms RTT, CUBIC TCP performed 45 times better than TCP BBR. They applied CoDel for improving TCP fairness and evaluated the fairness. The evaluation showed that it did not work well in case of TCP BBR since it consumes much bandwidth. On the other hand, it is effective in case of TCP CUBIC as TCP BBR cannot obtain enough throughput.

2. D-TCP: Dynamic TCP congestion control algorithm for next generation mobile networks

This approach finds a Congestion Control Factor N which is used to adaptively change the congestion window dynamically instead of using the traditional AIMD approach. On a packet loss event, D-TCP doesn't decrease the congestion window to decrease multiplicatively. After dropping to a certain level the algorithm tries to bring the window to previous size adaptively. This helps to efficiently control the congestion window for better network utilization. Upon reaching the stable condition it tries to stay in that state for as long as possible.

3. An intelligent TCP congestion control method based on Deep-Q network

DQN is an intelligent TCP congestion control algorithm based on Deep Q network (DQN). An agent is constructed to interact with the network environment. The agent adjusts the size of the congestion window by observing the characteristics of the network state. The authors designed a weighted reward function to balance throughput and delay. DQN uses double layer neural networks. Experimental results showed that the throughput achieved by TCP-DQN can be much more than traditional congestion control algorithms when the latency is similar to those algorithms.

4. TCP CUBIC vs TCP BBR on the Highway

The paper presents a comparison of TCP CUBIC and TCP BBR while driving on a highway and using a 4G Long Term Evolution (LTE) network. Analysis shows the driving conditions caused the network ISP to change in between causing downtimes. CUBIC and BBR performed comparably for throughputs but CUBIC encounters more duplicate ACKs. Using large buffers allows to utilize small-scale variations in LTE capacity, so it is better to program the congestion control algorithms to keep the buffers appropriately filled. The data from this experiment can be useful in simulating LTE networks.

5. Multipath TCP congestion control with heterogeneous radio access technologies

The paper presents experimental analysis about the performance of multipath TCP (MPTCP) congestion control algorithms when operating over two radio access technologies, i.e, WiFi and LTE. Analysis shows that in a scenario where LTE uplink capacity is much less than WiFi capacity then CUBIC performs the best with increasing number of connections or when the data volume per connection decreases. If both LTE and WiFi have comparable per-node capacities then CUBIC and OLIA perform similarly if a single connection is used to transfer the data. When the flows are short-lived and the number of connections are relative high, CUBIC outperforms OLIA and LIA.

6. QTCP: Adaptive congestion control with Reinforcement Learning

QTCP is an effective RL based approach that derives decision policies and handles complex networks. QTCP uses the reinforcement reward signals to learn the congestion control rules. This approach reformulates the original function approximator with adjustable generalization granularity across states which makes it possible to abstract sufficient information from a wide range of enviroment signals. QTCP generalization acheives better throughput and lower queueing delays than TCP Reno. QTCP also has better RTT performance than TCP Reno.

Summary Table

S. No.	Title	Algorithm used	Problem solved	Performance improvement	Future work
1	TCP fairness among modern TCP congestion control algorithms including TCP BBR	Cubic and bottleneck bandwidth round-trip propagation time congestion control	Improving TCP fairness among TCP CUBIC and TCP BBR	Directly modifying TCP BBR parameters BtlBw and Rtprop improves the fairness.	Evaluation of fairness with queue management algorithms such as RED
2	D-TCP: Dynamic TCP congestion control algorithm for next generation mobile networks	Convergence based congestion control for maintaining TCP fairness	Suffering TCP throughput in lossy and high-BDP networks	Better throughput and fairness than legacy algorithms along with friendliness	Cross layer detection and dynamic cwnd to improve bandwidth estimation accuracy
3	An intelligent TCP congestion control method based on Deep-Q network	Deep-Q network for reward based learning of congestion control rules	Inability of legacy algorithms to adapt to changeable network environment	Significant increase in throughput compared to legacy algorithms with same latency	Optimization of TCP-DQN methods to better adapt to high-performance computing
4	TCP CUBIC vs TCP BBR on the Highway	TCP CUBIC and TCP BBR over LTE	Lack of experimental data for on-the-field self-inflicted delays	TCP BBR gives lower self-inflicted delays than TCP CUBIC	Analysis of the impact of handovers on TCP BBR performance.
5	Multipath TCP congestion control with heterogeneous radio access technologies	TCP CUBIC, LIA and OLIA over LTE and WiFi	Lack of efficient algorithm for multipath flows under heterogeneous channels	Classification of best performers among CUBIC, LIA and OLIA under varying uplink capacities	Comparative study of AIMD congestion control algorithms under same conditions
6	QTCP: Adaptive congestion control with Reinforcement Learning	Reinforcement learning to generate signals for leaning congestion control rules	TCP not being able to intelligently adjust behavious under changing network environments	Better throughput and delay performance than Reno and QTCP-Baseline	Use of faster RCNNs instead of RL and comparing their respective performance

Deviation and Contribution

The above mentioned works are either experimental studies or simulations of TCP congestion control algorithms. Those simulations don't provide a way to inculcate the proposed algorithms in the actual real-world network where each node's behaviour affects the entire network.

Our work overcomes this barrier by encapsulating TCP CUBIC in the form of a Linux loadable kernel module which serves as a plugin to be used as the congestion control scheme by the operating system.

Moreover, none of the above works allow manipulation of the mathematical parameters defined in the equations.

Our work exposes those variables from the kernel space to the user space by using virtual filesystem. They can be modified by using a GUI.

Further, we have extended the CUBIC algorithm to the domain of multipath TCP or MPTCP by controlling the congestion window growth and collapse of individual subflows. This modified algorithm is called multipath CUBIC or mpCUBIC.

mpCUBIC is also implemented in form of a loadable kernel module with its own GUI for manipulating the control parameters.

Finally, a simulation environment is used to analyze the throughput, delay performance and congestion window size trend of CUBIC and mpCUBIC.

Proposed Work

CUBIC increases the window size aggressively when the window is far from saturation point. This makes CUBIC very scalable, highly stable and fair to standard TCP flows. Window growth depends only on the real time between two consecutive congestion events. CUBIC uses both concave and convex profiles of a cubic function for window increase. Upon receiving an ACK in congestion avoidance, CUBIC computes the window growth rate during the next RTT period. It sets the predicted size as the candidate target value of congestion window. When two TCP CUBIC subflows go through one bottleneck link, the total congestion window size for an MPTCP connection is given by a cubic function whose cycle is a half of one subflow. Hence in mpCUBIC, the period between two packet losses will be twice the period in CUBIC to keep the total MPTCP congestion window size comparable with single path TCP CUBIC connection.

Detailed Explanation

CUBIC uses the following window growth function :

$$W(t) = C(t-K)^3 + W_{max}$$

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}}$$

C is a CUBIC parameter, t is the elapsed time from the last window reduction, K is the time period that the above function takes to increase W to W_{max} when there is no further loss event and β is the window decrease constant at a fast retransmit event.

Upon receiving an ACK, first it is checked whether the protocol is in TCP region or not. For this, we analyze the window size of TCP in terms of the elapsed time t.

We use an additive factor α and a multiplicative factor β. If TCP window size increases by α per RTT, the TCP window size is given by :

$$W_{tcp}(t) = W_{max}(1 - \beta) + 3\frac{\beta}{2 - \beta}\frac{t}{RTT}$$

If $cwnd$ is less than W_{tcp} then the protocol is in TCP region and $cwnd$ is set to W_{tcp} at each reception of ACK.

If the protocol is not in TCP region and $cwnd$ is less than W_{tcp}, then the protocol is in concave region and $cwnd$ is incremented by :

$$\frac{W(t + RTT)}{cwnd} - 1$$

If $cwnd$ is more than W_{tcp}, then the protocol is in convex region and the increment in $cwnd$ remains the same as above.

On the event of a packet loss, $cwnd$ is reduced by a factor of β. A value lower than 0.5 results in slow convergence so adaptive adjustment of β is an issue which is resolved by the interactive GUI.

So far we discussed the congestion control algorithm for a single path CUBIC flow. For simplicity we consider only two CUBIC subflows in an MPTCP connection, it is necessary to avoid increasing congestion window too fast which may arise because of the subflows increasing their respective congestion windows independently.

The total congestion window size of an MPTCP connection is also given by a cubic function whose cycle is a half of the cycle of a balanced CUBIC growth function, i.e, $\frac{K}{2}$.

Therefore, in order to make the total $cwnd$ of MPTCP comparable with single path CUBIC TCP connection, we establish :

1. The packet loss period is twice for MPTCP connection.
2. The maximum window size just before a packet loss is set to W_{max} in the total congestion window size for an MPTCP connection.

Congestion window size of one subflow in mpCUBIC is given by :

$$W(t) = \frac{1}{D}\left(C'\left(\frac{t}{2} - K\right)^3 + W_{max}\right)$$

W_{max} and t and K are the parameters used in CUBIC.

The total window size for one MPTCP connection in one cycle is given by :

$$W_{total}(t) = \begin{cases} \frac{1}{D}\left(C'\left(\frac{t}{2} - K\right)^3 + C'\left(\frac{t+K}{2} - K\right)^3 + 2W_{max}\right), 0 < t < K \\ \frac{1}{D}\left(C'\left(\frac{t}{2} - K\right)^3 + C'\left(\frac{t-K}{2} - K\right)^3 + 2W_{max}\right), 0 < t < 2K \end{cases}$$

1. At $t = 0$, $W_{total}(-0)(1 - \beta) = W_{total}(+0)$
   We obtain,
   $$\left(C'\left(-\frac{1}{2}K\right)^3 + 2W_{max}\right)(1 - \beta) = C'(-K)^3 + C'\left(-\frac{1}{2}K\right)^3 + 2W_{max}$$
   $$C' = \frac{16}{8 + \beta}C$$

2. At $t = 2K$, $W_{total}(2K - 0) = W_{max}$
   We obtain,
   $$\frac{1}{D}\left(C'\left(-\frac{1}{2}K\right)^3 + 2W_{max}\right) = W_{max}$$

   $$D = \frac{16}{8 + \beta}$$

Upon substituting the values of C' and D, we get the equation for the congestion window size of an mpCUBIC subflow :

$$W(t) = C\left(\frac{t}{2} - K\right)^3 + \frac{8+\beta}{16} W_{max}$$

Here, C, t, K, β and W_{max} are the CUBIC parameters.

Implementation as a Loadable Kernel Module

The congestion control algorithm being used by the system can be modified by the `sysctl` command setting :
`net.ipv4.tcp_congestion_control`

A congestion control loadable kernel module follows the given structure in order for the operating system to register it :

- We have to define the working of the following predefined functions :
  - `init()` - handles the first ACK segment
  - `pkts_acked()` - processes the new ACK segments
  - `cong_avoid()` - congestion avoidance processing
- Pointers of the above predefined functions are registered in a data structure defined as `struct tcp_congestion_opt`. In this structure, the name of the algorithm is defined which is used in the `sysctl` command.
- Each Ring-0 system call calls the `tcp_congestion_control` function when it is registered as a kernel module.
- `bictcp_cong_avoid()` is registered in the `tcp_congestion_opt` data structure as the congestion avoidance function that calculates the congestion window size.

Pseudocode of CUBIC Algorithm

Initialization:

$tcp\_friendliness \leftarrow 1,\ \beta \leftarrow 0.2$

$fast\_convergence \leftarrow 1,\ C \leftarrow 0.4$

$cubic\_reset()$

On each ACK:

begin

if $dMin$ **then** $dMin \leftarrow \min(dMin, RTT)$

else $dMin \leftarrow RTT$

if $cwnd \leq ssthresh$ **then** $cwnd \leftarrow cwnd + 1$

else

$cnt \leftarrow cubic\_update()$

if $cwnd_{cnt} > cnt$ **then**

$cwnd \leftarrow cwnd + 1, cwnd_{cnt} \leftarrow 0$

else $cwnd\_cnt \leftarrow cwnd\_cnt + 1$

end

Packet loss:

begin

$epoch\_start \leftarrow 0$

if $cwnd < W_{last\_max}$ **and** $fast\_convergence$ **then**

$W_{last\_max} \leftarrow cwnd * \left(1 - \frac{\beta}{2}\right)$

else $W_{last\_max} \leftarrow cwnd$

$ssthresh \leftarrow cwnd \leftarrow cwnd * (1 - \beta)$

end

Timeout:

begin

$cubic\_reset()$

end

$cubic\_update()$:

begin

$ack\_cnt \leftarrow ack\_cnt + 1$

if $epoch\_start \leq 0$ **then**

$epoch\_start \leftarrow tcp\_time\_stamp$

if $cwnd < W_{last\_max}$ **then**

$K \leftarrow \sqrt[3]{\frac{W_{last\_max} - cwnd}{C}}$

$origin\_point \leftarrow W_{last\_max}$

else

$K \leftarrow 0$

$origin_{point} \leftarrow cwnd$

$ack\_cnt \leftarrow 1$

$W_{tcp} \leftarrow cwnd$

$t \leftarrow tcp\_time\_stamp + dMin - epoch\_start$

$target \leftarrow originpoint + C(t - K)^3$

if $target > cwnd$ **then** $cnt \leftarrow \frac{cwnd}{target - cwnd}$

else $cnt \leftarrow 100 * cwnd$

if $tcp\_friendliness$ **then** $cubic\_tcp\_friendliness()$

end

$cubic\_tcp\_friendliness()$:

begin

$W_{tcp} \leftarrow W_{tcp} + \frac{3\beta}{2 - \beta} * \frac{ack\_cnt}{cwnd}$

$ack\_cnt \leftarrow 0$

if $W_{tcp} > cwnd$ **then**

$max\_cnt \leftarrow \frac{cwnd}{W_{tcp} - cwnd}$

if $cnt > max\_cnt$ **then** $cnt \leftarrow max\_cnt$

end

$cubic\_reset()$:

begin

$W_{last\_max} \leftarrow 0, epoch\_start \leftarrow 0, origin\_point \leftarrow 0$

$dMin \leftarrow 0, W_{tcp} \leftarrow 0, K \leftarrow 0, ack\_cnt \leftarrow 0$

end

Performance Evaluation

Simulation Environment

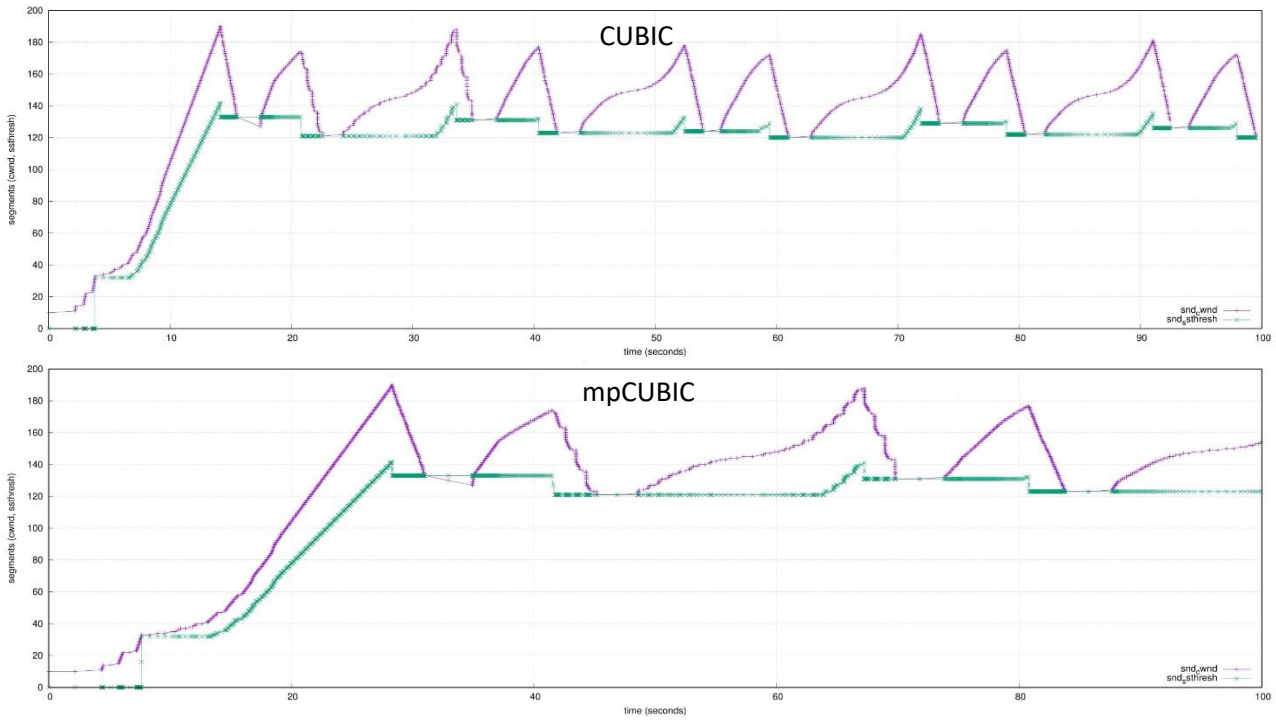
We used a MahiMahi simulation enviroment for simulating multiple TCP flows on Ubuntu 20.04 OS running version 5.11 of the Linux kernel. We setup a 12Mbps bottleneck uplink with a 117Kib drop-tail queue and an 80ms RTT.

The client is programmed to send a continuous stream of 1KiB packets to the server obeying the congestion control algorithm.

We monitored the following parameters :

a. Congestion window size of CUBIC, mpCUBIC flows.
b. Slow-start-threshold of CUBIC, mpCUBIC flows.
c. Throughput and queueing-delay of CUBIC, mpCUBIC flows.
d. Throughputs of multiple TCP flows sharing a bottleneck link.

a. Congestion Window Size

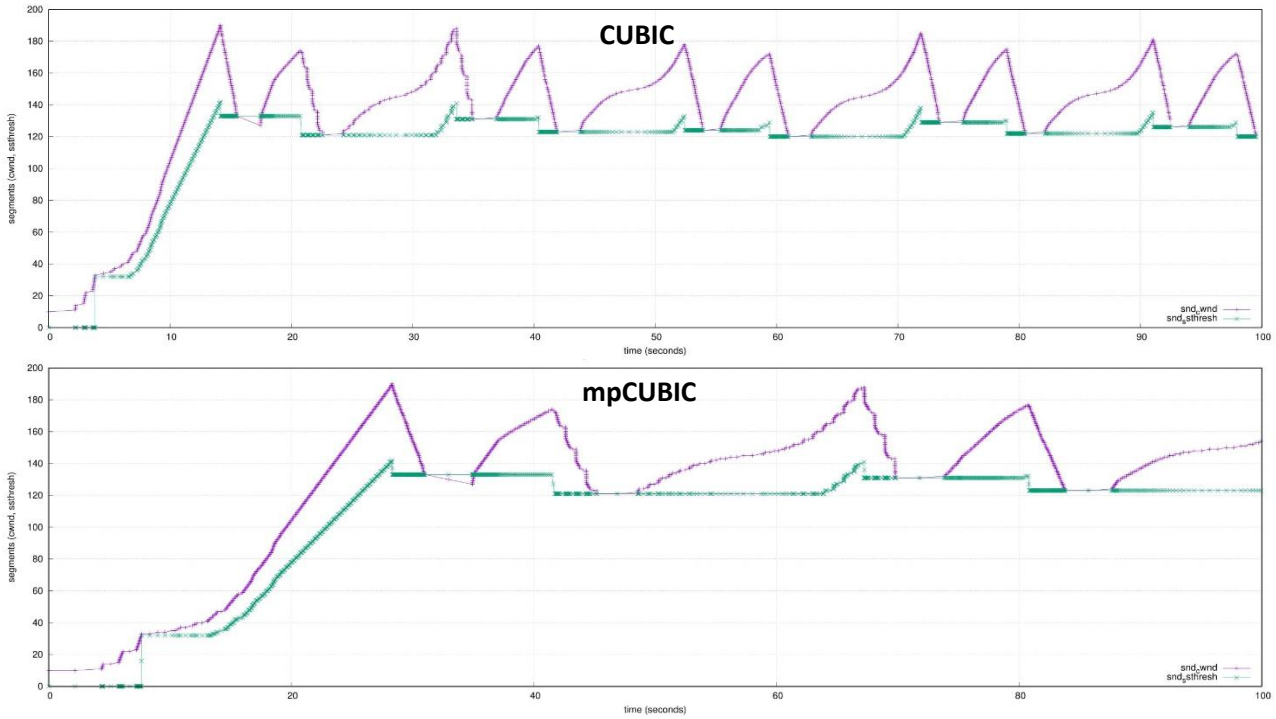


The first graph shows the total congestion window size of a MPTCP connection with two CUBIC TCP flows.
It can be seen that *cwnd* follows the cubic nature as there are both concave and convex parts in the graph as proposed theoretically.

The second graph shows the total congestion window size when mpCUBIC is used for the two subflows.
Here also *cwnd* follows cubic behaviour .The cycle of packet losses is roughly double of the cycle of CUBIC flows as shown by the theoretical equation :

$$W(t) = C\left(\frac{t}{2} - K\right)^3 + \frac{8+\beta}{16}W_{max}$$

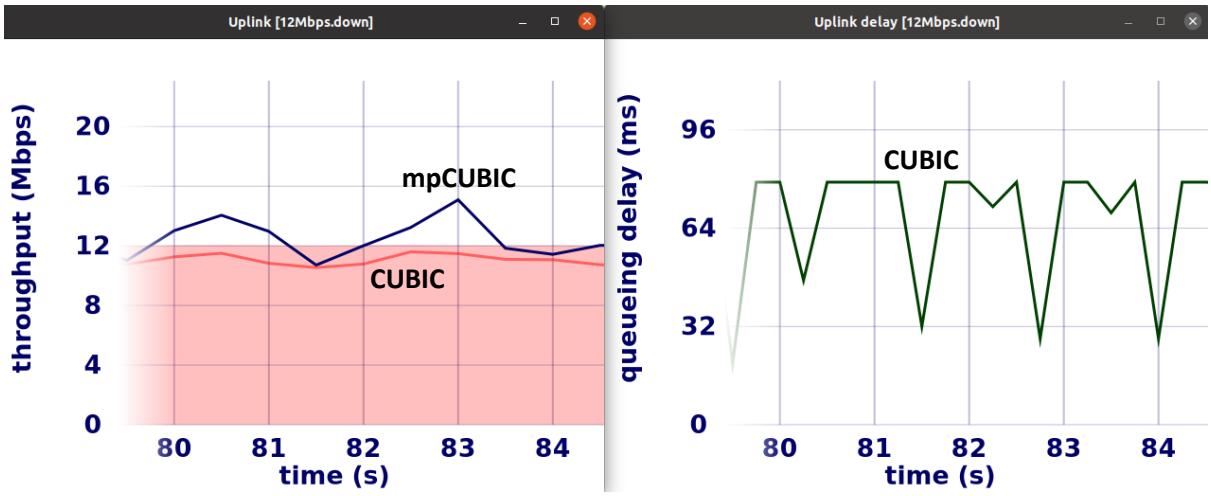
b. Slow-Start-Threshold



ssthresh grows linearly until the first packet loss occurs. It remains stable between consecutive loss events and grows and drops at those events.

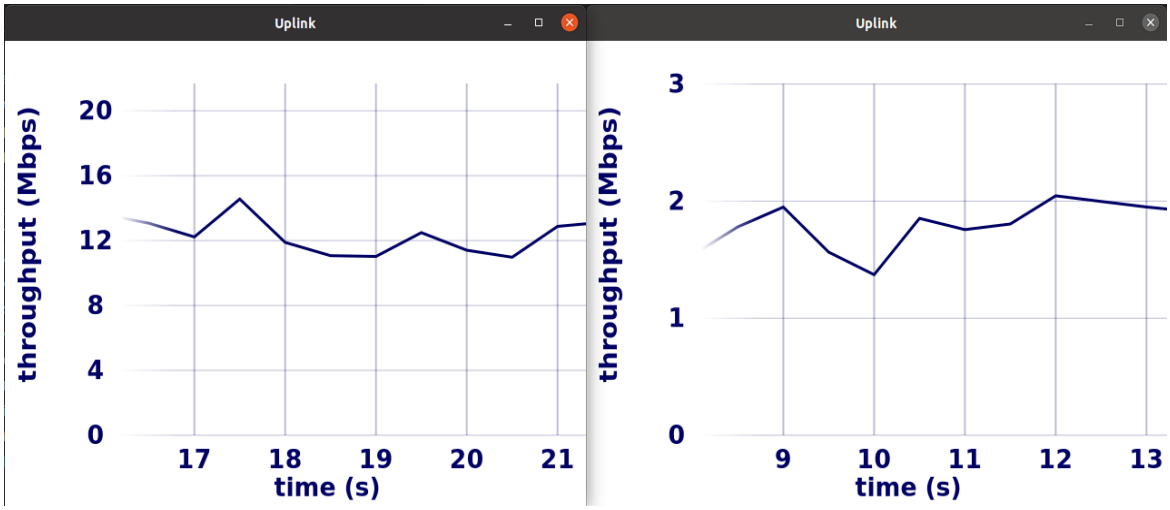
Data - https://github.com/anuragnitt/cubic-congestion-control/tree/master/plot

c. Throughput and Queueing-Delay



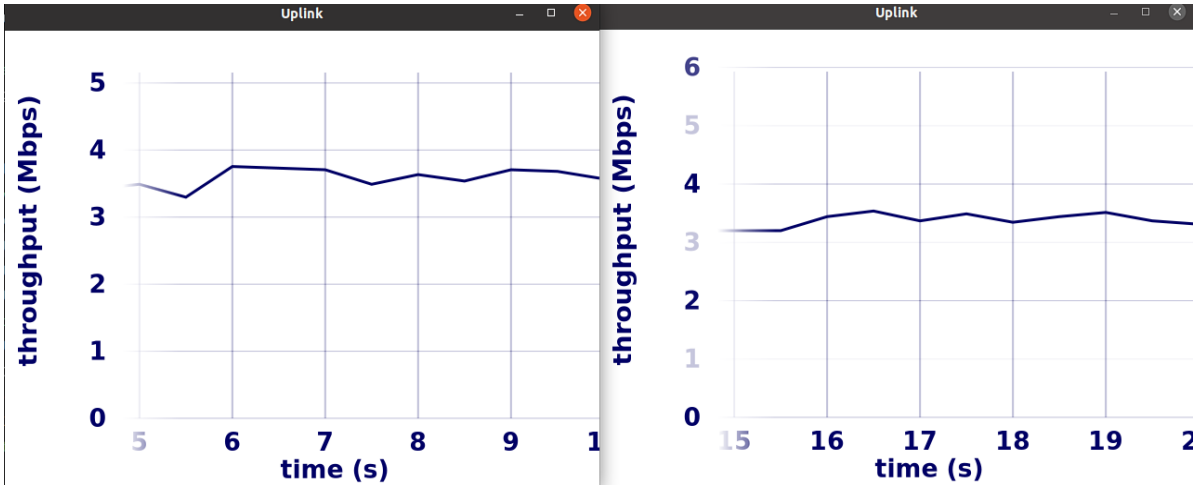
Single path CUBIC and mpCUBIC flows achieve same throughput when run on a 12Mbps uplink separately because their congestion windows grow in the same manner when there is no other TCP flow sharing the uplink.

d. Throughputs of Multiple TCP Flows Sharing Bottleneck Link



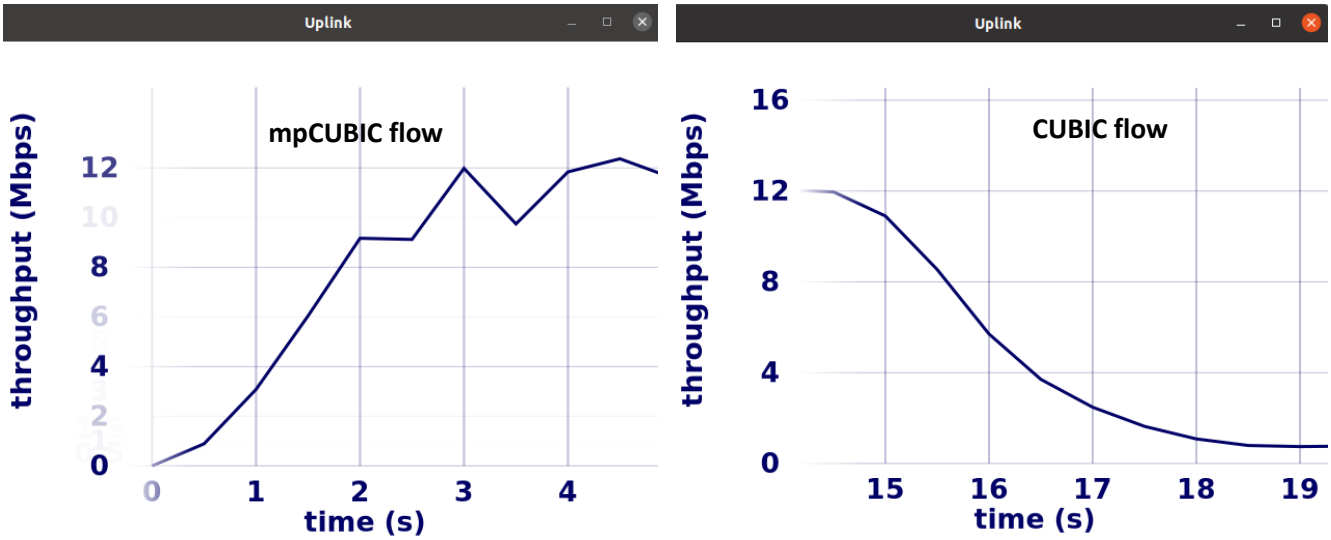
Two CUBIC flows sharing a 12Mbps bottlneck link with β = 1

β denotes the multiplicative decrease in the congestion window size on a packet loss.
A value of 1 means that there is no multiplicative decrease therefore the CUBIC flow which starts earlier ends up monopolizing the bottleneck link and the latter flow's throughput suffers.



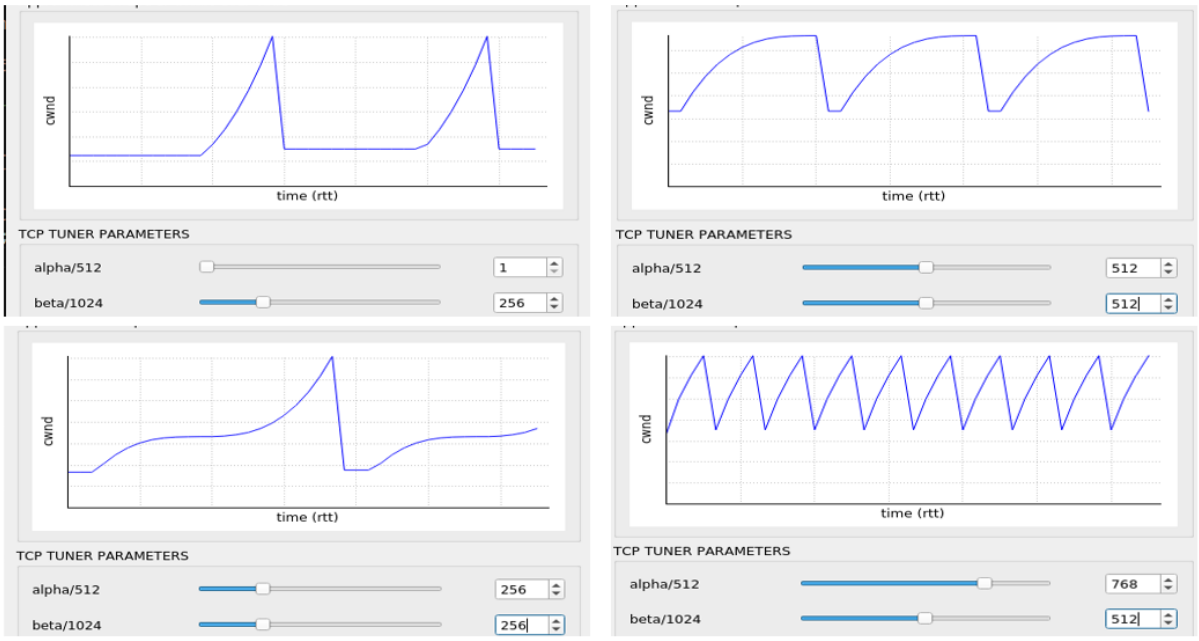
Two CUBIC flows sharing a 12Mbps bottlneck link with β = 0.25

A value of 0.25 for β means there is 75% reduction in congestion window size on a packet loss. Therefore, the older CUBIC flow is ready to reduce its congestion window size and ultimately both flows get a fair allocation of bandwidth.



Initially, there is an MPTCP connection having a 12Mbps bandwidth. First, a CUBIC subflow starts and achieves the maximum throughput because there is no other competitor subflow.

As soon as an mpCUBIC subflow start sharing the same MPTCP connection, it outperforms the already present CUBIC subflow in terms of throughput.



The GUI lets us manipulate the CUBIC control parameters α, β and also plots the approximate graph of $cwnd$ vs $time$.

Conclusions

We discussed about the CUBIC congestion control algorithm which is used for fast and long distance networks. It simplifies the older BIC-TCP congestion window control and maintains TCP-friendliness and RTT-fairness.

CUBIC uses a cubic window growth function in terms of elapsed time since the last packet loss event. The real-time nature of this algorithm keeps the window growth rate independent of RTT, which maintains TCP-friendliness under both short and long RTT paths.

We also discussed a modification of CUBIC for MPTCP connections, called as mpCUBIC to fit the multipath scenario. It provides slightly better performance than CUBIC under single path TCP connection.

When two mpCUBIC subflows are present, the packet loss cycle is doubled and the maximum total congestion window size for the MPTCP connection is equal to that of a single path CUBIC TCP connection.

We implemented both CUBIC and mpCUBIC as Linux kernel modules and tested their performance on a simulated bottleneck link.

We analyzed the congestion window size, slow-start-threshold, throughput and queueing-delay of CUBIC and mpCUBIC under single path TCP and multipath TCP environments.

Future Work

The special feature of this work was the exposing of CUBIC control parameter β from the kernel space to user space to be manipulated in real-time with the help of a GUI.

Until now this required human interaction with the GUI which is slower and inaccurate relative to a programmatic interaction.

The β value can be manipulated programmatically in real-time as signalled by the network environment.

We can train and use a machine learning model which would learn how to change β according to a reward-based learning approach.

This can result in dynamic and accurate congestion control which would work to improve the throughput and RTT performance in an unstable and changing network environment.

References

1. Kevin Miller and Luke Hsiao. TCPTuner: Congestion control your way
2. Toshihiko Kato, Shiho Haruyama and Ryo Yamamoto. mpCUBIC: A CUBIC-like congestion control algorithm for multipath TCP
3. Kanon Sasaki and Kouto Miyazawa. TCP fairness among modern TCP congestion control algorithms including TCP BBR
4. Madhan Raj Kanagarathinam and Sukhdeep Singh. D-TCP: Dynamic TCP congestion control algorithm for next generation mobile networks
5. Yinfeng Wang, Longxiang Wang and Xiaoshe Dong. An intelligent TCP congestion control method based on Deep-Q network
6. Feng Li and Xiaoxiao Jiang. TCP CUBIC vs TCP BBR on the Highway
7. Monika Prakash, Atef Abdrabou. Multipath TCP congestion control with heterogeneous radio access technologies
8. Kaushik Chowdhury and Waleed Meleis. QTCP: Adaptive congestion control with Reinforcement Learning