SKELETON OF Contents

<u>Abstract -</u>

MPTCP is an extension of TCP that allows the applications to use multiple interfaces over the same TCP connection, enabling them to utilize the aggregated bandwidth and provide resilience to network failure. These characteristics of MPTCP make it well-suited for streaming traffic. Dynamic Adaptive Streaming over HTTP (DASH), a streaming protocol dominating the Internet, ensures Quality of Experience (QoE) to content consumers by adapting the playback bitrate to match the available throughput. Efforts are being made to improve the QoE even further. When DASH is used with MPTCP as transport protocol, the aggregated throughput and increased reliability due to the use of multiple interfaces for a streaming session improves QoE. There have been several studies to analyze the performance of DASH on MPTCP under the presence of shared bottleneck link, and varying network resources like MPTCP buffer sizes and path latencies. However, the real Internet scenario consists of several types of traffic with varying loads competing for resources. In this work, we aim to perform an in-depth evaluation of the QoE parameters of DASH traffic with MPTCP under mixed traffic loads. Moreover, we leverage Linux network namespaces for performing the experiments, which provide us with a light weight and cost-effective alternative to physical testbeds and virtual machines. Besides, we use the state-of-the-art upstream implementation of MPTCP in the Linux kernel, which is relatively straightforward to configure as per the project requirements.

Multipath TCP (MPTCP) is an extension of TCP that facilitates the utilization of multiple interfaces on a single connection, thereby enhancing the aggregated bandwidth and resilience against network failures. Such kinds of advantages can be crucial for applications, especially streaming applications. Dynamic Adaptive Streaming over HTTP (DASH), a streaming protocol that dominates the internet, guarantees Quality of Experience (QoE) to content consumers by adapting the playback bitrate to match the available throughput.

<u>Introduction -</u>

Motivation

The present era of the Internet has billions of devices, large data centers, and mostly a large number of mobile devices. However, at its backbone, the network stack still uses the traditional TCP which has been existing for 4-5 decades. The traditional TCP can make use of only one interface at a single point in time. However, in the current generation, our mobile devices keep on moving from one network to another, which causes an interruption in the working of the application layer. Multipath TCP is used to overcome this issue, by allowing the application to use multiple interfaces or multiple paths over the same TCP connection. This helps the application to obtain an improved performance by using the aggregate bandwidth of multiple interfaces and also provides resilience to network failure. These characteristics of MPTCP can benefit all types of traffic, especially streaming traffic because it requires consistent throughput.

One of the very popular streaming protocols is HTTP Live Streaming (HLS) developed by Apple Inc [ref]. However, it is a proprietary protocol. Similarly, a few other proprietary protocols have been proposed over the decades. Efforts have been made by standardization organizations to propose a streaming protocol which can be used across all platforms. Moving Pictures Experts Group (MPEG) came up with a non-proprietary standard protocol for video streaming called Dynamic Adaptive Streaming over HTTP (DASH) [ref]. Similar to HLS, DASH is also an adaptive bitrate streaming protocol, which divides the streaming data (e.g videos) into smaller blocks and encodes them at different quality levels. This makes it possible for the streaming client application to switch between blocks of different quality levels, based on the dynamic network conditions. DASH has started slowly dominating the other streaming protocols over the Internet. Several streaming platforms like Netflix, Hulu and YouTube are relying on the MPEG-DASH protocol for streaming their content [ref]. With the increased usage of DASH in recent years and its importance for streaming data, several studies have been carried out to evaluate the Quality of Experience (QoE) of the DASH protocol conjointly with MPTCP in multi-homed devices.

In the Internet, several types of traffic can exist at the same time, like bursty traffic, mice traffic, interactive traffic or time-sensitive traffic. In such scenarios, the variance of these traffic can affect each other's performance. Like the elephant traffic (long lasting traffic) tends to occupy a large portion of the buffers at the router. It might lead to queue buildup, which causes the dropping of packets belonging to other types of traffic (like mice traffic). Similarly, on shared bottleneck links, different types of traffic compete for the bandwidth and as a result they end up affecting each other.

Several studies have been conducted to analyze the performance of DASH traffic using MPTCP in the presence of shared bottleneck link [?]. Over the years with the deepening research in MPTCP, several improvements have been proposed for the Path Management and Congestion Control algorithms to make them adaptable for different network conditions. Studies have been made to evaluate the performance of DASH against the QoE parameters under different MPTCP congestion control algorithms and schedulers [?]. However, the performance of MPTCP for DASH traffic has not been studied for varying traffic loads under BBR as congestion control for background traffic. Hence, we plan to perform an in-depth analysis of the QoE parameters of DASH traffic with MPTCP under varying traffic loads and conditions. We plan to use the following QoE metrics to evaluate the performance of DASH traffic with MPTCP: bitrate switches, average bitrate, and playback interruptions, network throughput, and startup delay. We leverage Linux network namespaces for performing the experiments, which provide us with a light weight and cost-effective alternative to physical testbeds and virtual machines. Besides, we use the state-of-the-art upstream implementation of MPTCP in the Linux kernel, which is relatively straightforward to configure as per the project requirements.

Contributions-

This project makes the following contributions - (i) Perform an in-depth experimental analysis of MPTCP connection establishment phase and its working. (ii) Emulate MPTCP in Linux network namespaces, with advanced routing rules, and MPTCP sockets. (iii) Evaluate MPEG-DASH performance using MPTCP under varying traffic loads and BBR as congestion control in background traffic. As of now, we have attained objectives (i) and (ii), and expect to complete the remaining objectives by the end of this course.

Organization -

This report is organized as follows. In Chapter 2, we have provided a background study of MPTCP, its components include Path manager ,Scheduler Congestion control and MPEG. Chapter 3 contains the literature review of the existing algorithms and studies to analyze the performance of DASH on MPTCP. Chapter 4 discusses the problem statement and major objectives of this work. A detailed description of the work done till now is given in Chapter 5, followed by the inferences. Finally, the conclusions and future work is given in Chapter 6.

Background Study -

1. Multipath TCP

MPTCP (Multipath TCP) is a transport layer protocol which is standardized by IETF. It is a Multipath transport protocol which makes use of the multiple interfaces (in multi-homed devices) available to transmit data simultaneously. This enables the user to use cumulative added up bandwidth of multiple interfaces, hence providing enhanced performance. It also provides resilience such that if one interface fails, other interfaces are still active and hence does not interrupt the application. One special feature of MPTCP which makes it better than its counterparts, is that it is backward compatible with TCP protocol.

As shown in Fig 1 multiple MPTCP subflows appear as a single connection to application on both the ends. However, each MPTCP subflow looks like a regular TCP subflow to the Network layer. Hence, on the wire, MPTCP traffic is observed as TCP traffic providing smooth operation with firewalls and MPTCP unaware middle boxes.

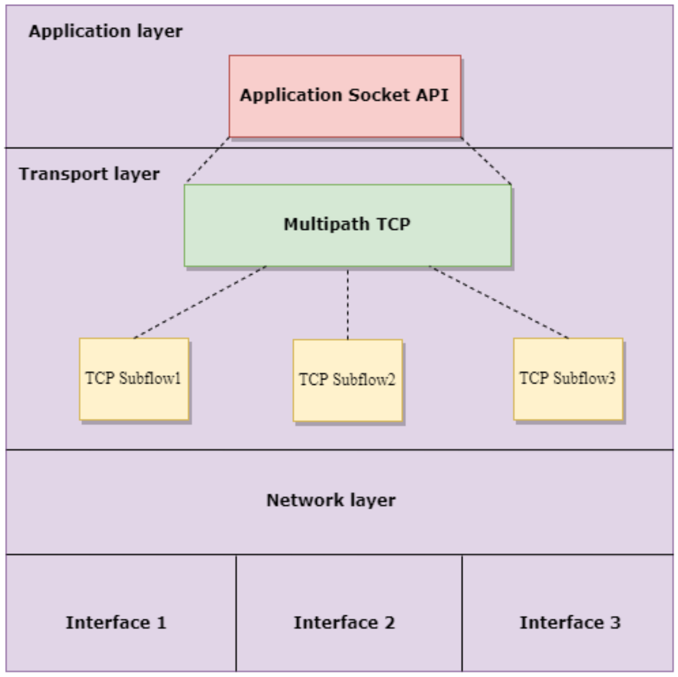


Fig 1 - MPTCP Architecture

Initially MPTCP was standardized in 6824 which is termed as MPTCPv0. Later, a modified version of MPTCP was proposed in RFC 8684 which obsoletes the RFC 6824. This version of MPTCP in RFC 8684 is also called MPTCPv1.

Currently the implementation of MPTCP exists in Linux Kernel and FreeBSD. FreeBSD implementation was started by Swinburn, however, is not under active development currently. Linux implementation of MPTCP has been going on for several years and is still an active project. The project was started by Sebastien Barre as his thesis in 2009. Since then several contributors have joined the project. Two variants of Linux implementation exist: out-of-tree and upstream implementation. Out-of-tree implementation is based on the RFC 6824 (which is also called MPTCPv0), and also supports MPTCPv1. It is not merged to the Linux kernel mainline because it requires several modifications to the original implementation of TCP in Linux. Upstream implementation is based on RFC 8684 (MPTCPv1). It is merged into the Linux kernel mainline from version 5.6 onwards. The project is still ongoing and requires several modifications until it can be made default in the kernel. In this work, we take a detailed look at the working of Multipath TCP in kernel version 6.0.9 which includes the upstream implementation, and hence we follow the RFC 8684 specifications.

1. Connection establishment

The connection establishment process in MPTCP is similar to that of TCP, except for the TCP Options (a 40-byte field in the TCP header) that are exchanged during this process. A new option called MP_CAPABLE is added to the header in case of MPTCP (as shown in Fig. 2).

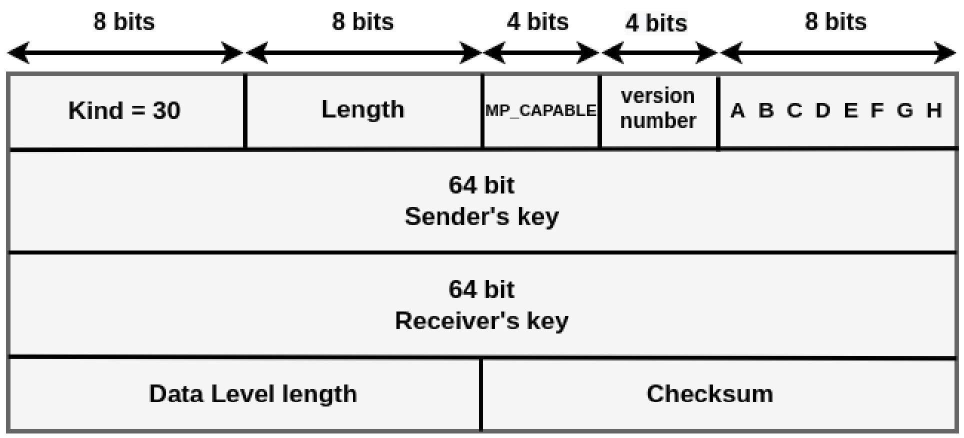


Fig. 2: MPTCP option with MP_CAPABLE subtype

When MPTCP is enabled, the MP_CAPABLE option is used in SYN, SYN/ACK, and ACK packets for setting up the initial MPTCP connection. The MP_CAPABLE option is mainly used for two things. First, it checks whether both endpoints support MPTCP or not. Second, it is used to exchange authentication keys and share a token to identify a connection.

There are various fields in the MP_CAPABLE option. The Kind field is of 8 bits, and a value of 30 has been assigned for all MPTCP options, the length field is of 8 bits, and the Subtype field is 4 bits. According to RFC 6824 and RFC 8684, IANA maintains a new sub-registry using the 4-bit subtype field. Fig. 3 shows the values assigned for the MPTCP subtype field.

Value	**Symbol**	**Name**
0x0	MP_CAPABLE	Multipath Capable
0x1	MP_JOIN	Join Connection
0x2	DSS	Data Sequence Signal (Data ACK and Data Sequence Mapping)
0x3	ADD_ADDR	Add address
0x4	REMOVE_ADDR	Remove Address
0x5	MP_PRIO	Change Sub-flow priority

0x6	MP_FAIL	Fallback
0x7	MP_FASTCLOSE	Fast close
0x8	MP_TCPRST	Sub-flow reset
0xf	MP_EXPERIMENTAL	Reserved for private use

Figure 3: MPTCP option subtype

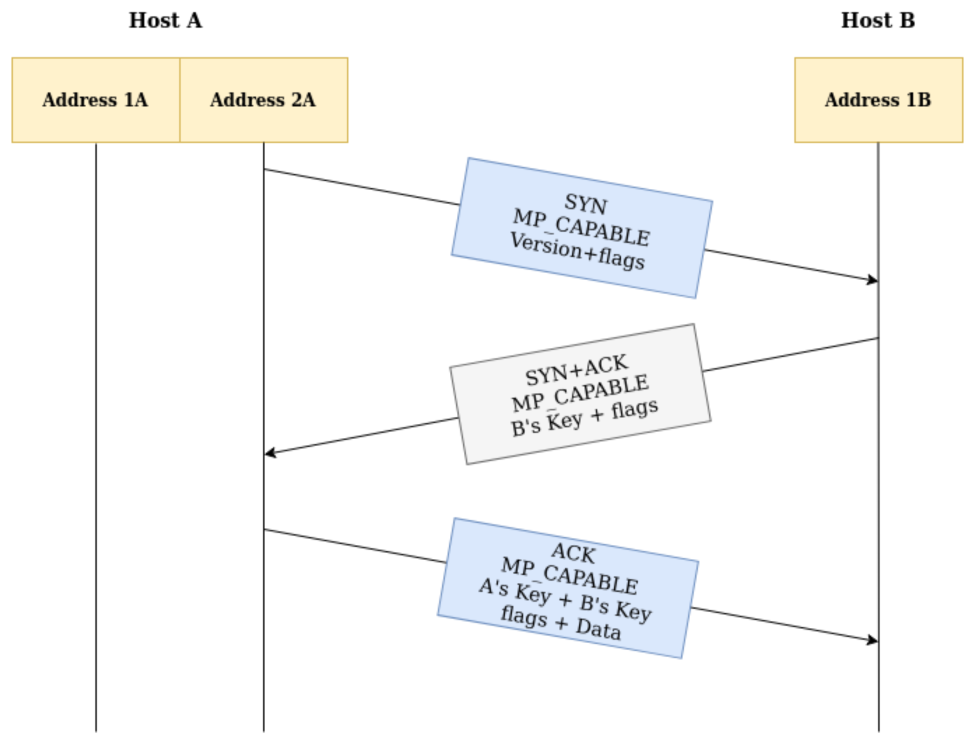


Fig. 4: MPTCP connection establishment

When the client wishes to connect, it sends a SYN packet to the server. According to RFC 8684, IANA maintains a new sub-registry using the 4-bit subtype field in the MPTCP option. The subtype field in the initial SYN packet consists of MP_CAPABLE. The MP_CAPABLE option is mainly used for two purposes. First, it checks whether both endpoints support MPTCP or not. Second, it is used to exchange authentication keys and cryptographic materials between peers. Hence, in the SYN packet, along with MP_CAPABLE, the version number and certain flags are also shared with the server.

On receiving the SYN packet, the server generates a random key (H2’s key) to be sent along with the SYN+ACK packet. The key generation process could be implementation specific. A 32-bit token is generated using this key. This token is subsequently used to identify the connection for all the future sub-flows.

On receiving the SYN+ACK packet, the client knows whether the server is also MPTCP capable or not, and it gets the server's key if it is MPTCP capable. Subsequently, the client makes use of the ACK packet to share its own generated random key (H1's key) with the server, which is later used for authentication purposes. The client can piggyback data, along with the ACK packet, if it has some data to be sent immediately.

2. Join Connection

When new sub-flows are set up between the same sender and receiver, the keys exchanged in the MP_CAPABLE option during the initial connection establishment play a considerable role in authenticating the connection. The additional sub-flows also begin with the same SYN, SYN/ACK, and ACK packets, but there is a difference in the TCP Options that are used while setting up additional sub-flows. Instead of using the MP_CAPABLE option, a separate TCP option called MP_JOIN is used. Like MP_CAPABLE, MP_JOIN is also created using the TCP option field, which is 40 Bytes long. Firstly, It is used to authenticate the connection established earlier using MP_CAPABLE. Then it is used to establish the sub-flow that the sender intends to make.

A new sub-flow is initiated between hosts A and B using different unused IP addresses of A and B. When the initial MPTCP connection begins with the MP_CAPABLE exchange, hosts know about their own addresses and can be knowledgeable about other hosts' addresses through signaling exchange. Any of the hosts(But usually the initial connection initiator host initiates the sub-flow) can create a sub-flow using the knowledge over a currently unused pair of IP addresses.

The token which is generated from the key is used as an identifier to identify which MPTCP connection it is joining. The HMAC(Hash-Based Message Authentication Code) is used for the authentication. The keys exchanged in the MP_CAPABLE handshake are used by HMAC and generate a random number that is exchanged in the MP_JOIN option.

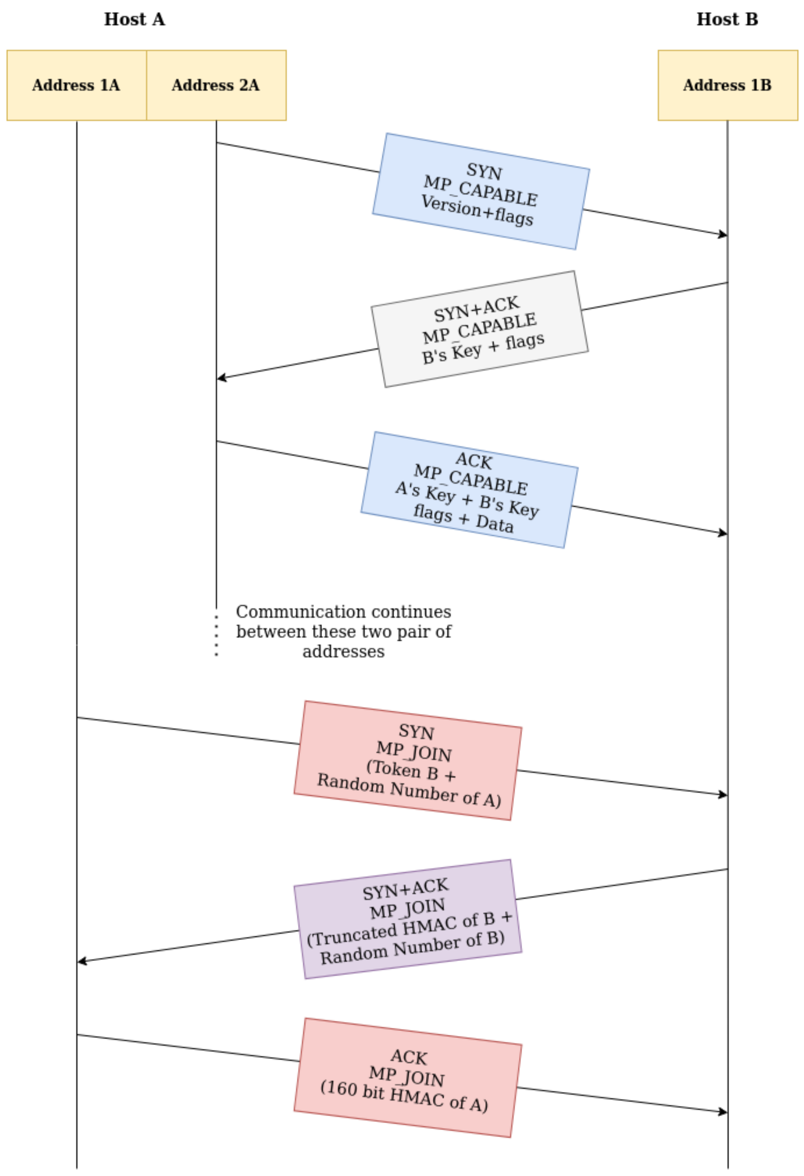


Fig. 5: Subflow creation using MP_JOIN

3. Data transfer

Data transfer in MPTCP takes through the option subtype DSS (Data Sequence Signal). It consists of two message types that can be sent via DSS. It includes Data Sequence Mapping and Data Acknowledgement. Each of them are described below -

1. Data Sequence Mapping - It sends the mapping between subflow number and connection level DSN (Data nSequence Number) such that, they can be reassembled on the other side.

2. Data ACK - It is used to send connection level acknowledgement.

DSS can contain either of the two or both the message at same time in an option, depending on the flags set. Application sends a signal stream of data to the MPTCP module, which then re-distributes the packets to different subflow. Similarly on the receiver side, packets being received from different subflows need to be collated again in order to obtain the entire message. This is where Data Sequence Mapping comes into the picture.

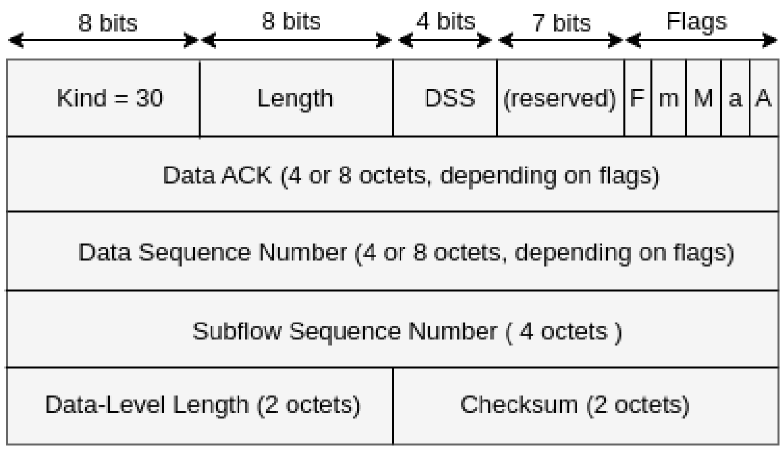

8 bits
8 bits
4 bits
7 bits
Flags
Kind = 30
Length
DSS
(reserved)
F
m
M
a
A
Data ACK (4 or 8 octets, depending on flags)
Data Sequence Number (4 or 8 octets, depending on flags)
Subflow Sequence Number (4 octets)
Data-Level Length (2 octets)
Checksum (2 octets)

Fig. 6: Data Sequence Signal (DSS) Option

All the reserved bits are set to 0.' A' flag set means ACK is present. Whereas 'a' means Data ACK is 8 octates. If 'a' is not set, it means Data ACK is 4 octates only. 'M' flag set means DSM (Data Sequence Mapping) is present, which includes DSN (Data Sequence Number), Subflow sequence number, Data level length and checksum. Similar to 'a' flag 'm' flag set means DSN is 8 octates. Hence flags like 'a' and 'm' are of importance if and only if their corresponding flags 'A' and 'M' are set, or flag 'a' and 'm' are ignored. Flag 'F' is used for data level fin, that is Data FIN.

Checksum is only present if during the initial connection establishment phase the checksum bit was negotiated. If the checksum bit was shared but the DSS option does not include the checksum, then the receiver should reset the connection by sending the MP_TCPRST option.

2. Different Components of MPTCP

MPTCP includes three most crucial components which are exclusively required for proper functions of MPTCP. They are Path Manager, Scheduler and Congestion Control Algorithms. They are discussed in detail below.

1. Path Manger

Path manager (PM) is an essential component of MPTCP which determines when the additional sub-flows are to be created or removed, and controls the advertisement of available addresses to the peers. It also determines the addresses used to create the subflows. Mptcpd (Multipath TCP Daemon) and ip mptcp are the tools that can configure several aspects of the MPTCP path manager. The upstream implementation of MPTCP supports two types of PM: in-kernel and userspace. A new sysctl variable called `pm\_type' has been introduced in the kernel version 5.19, which specifies the type of PM to be used. For example, `Pm_type = 0' (in-kernel path manager) and `Pm_type = 1' (userspace path manager). Fig. 7 shows the difference between the in-kernel and userspace PM.

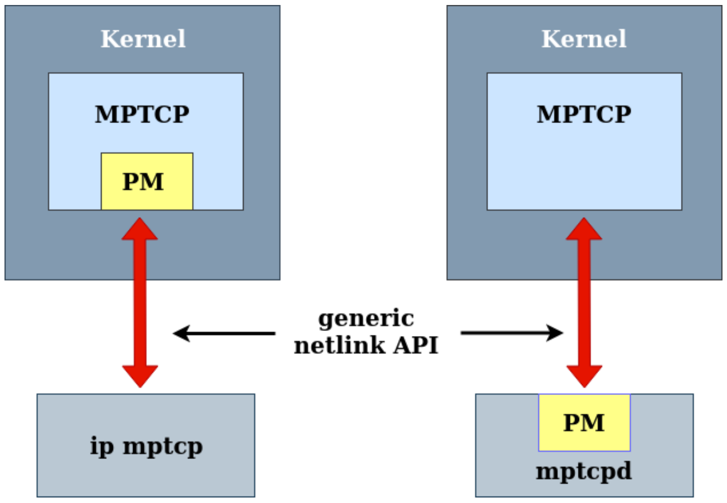


Figure 7: In-kernel vs Userspace PM

2. Scheduler

In the case of MPTCP, the endpoints are connected via multiple paths, so the packets need to be scheduled on multiple subflows. The MPTCP scheduler is the algorithm which selects the path to be used for transmitting a packet. The path selection (routing) at the network layer is based on the network condition without considering the endpoint and application requirements. However, the MPTCP scheduler can use the information about paths (gathered by end-hosts) to efficiently distribute the packets on various paths/subflows. Fig. 8 shows the working of a simple MPTCP scheduler. The application sends the data to the output queue. The scheduling algorithm decides the path to be used for transmitting the data.

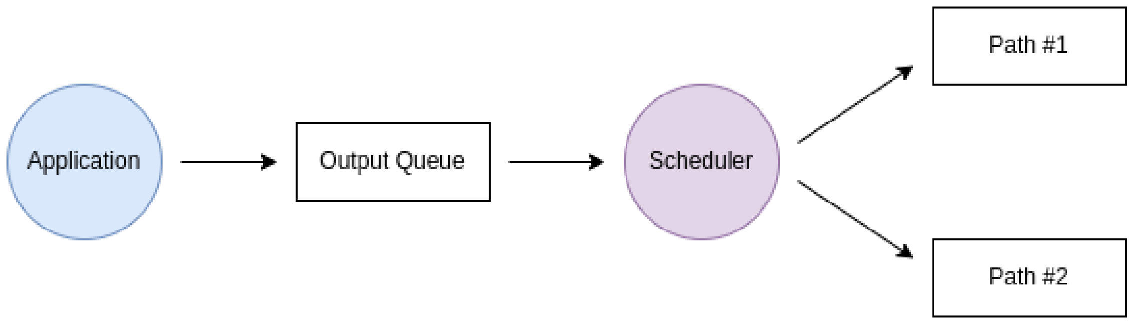


Fig. 8: Simple Scheduler

3. Congestion Control

Congestion Control is another important component of MPTCP which highly determines the performance of MPTCP. It only has to detect congestion and allocate resources, but also has to provide fairness among several subflows. Few of the important goals which are expected to be achieved by any MPTCP Congestion Control Algorithm are listed below.

1. It should provide at least the same or better throughput compared to a single TCP connection.
2. One subflow should not harm the other subflows. It basically means one subflow should not take up more resources than it is expected to take such that it hampers the performance of other flows.
3. A multipath congestion control algorithm should send as much traffic to a subflow, as much as its capacity at any particular point of time. This feature is also termed as resource pooling.

There are two most common categories of Congestion Control Algorithms based on whether they provide resource pooling or not. They include Uncoupled and Coupled Congestion Control Algorithms. Uncoupled Congestion control Algorithms considers each subflow independently and hence does not provide resource pooling. Few common examples for Uncoupled congestion control algorithms include Cubic, New Reno . Another drawback of this type of algorithm is that they do not ensure network fairness. Coupled Congestion Control Algorithms couples/collates the congestion subsystems of all the subflows at MPTCP connection level. They can provide the feature of resource pooling and network fairness. Examples of coupled congestion control algorithms are BBR, LIA, OLIA, BALIA, wVegas, etc.

BBR(Bottleneck Bandwidth and Round Trip Time) Congestion control -
BBR is designed to maximize network throughput while minimizing latency and packet loss, resulting in a better user experience for applications like web browsing, video streaming .BBR focuses on accurately estimating the available bandwidth and round-trip time (RTT) of the bottleneck link in a network path. BBR dynamically adjusts the sending rate of data based on the available bandwidth and the measured RTT. It aims to fill the bottleneck link with data without causing congestion or excessive queuing, thus utilizing the network's capacity efficiently.BBR continuously monitors and adapts to changes in network conditions, such as congestion or bandwidth fluctuations. It adjusts its sending rate to maintain optimal performance and avoid overloading the network.BBR has been shown to provide improved performance and reduced latency in real-world network scenarios.

3. MPEG-DASH -

MPEG-DASH is a standard streaming protocol which has been standardized by ISO. It has gained popularity in recent years as it is being used by several streaming and video content websites like YouTube and Netflix. DASH is quite similar to HLS which is currently the most popular streaming protocol. Both of them are based on HTTP protocol and use TCP as their transport layer protocol. They even provide adaptive bitrate streaming by segmenting and encoding the video content into multiple blocks of different quality levels. The encoding used in the HLS is H.264 or H.265. However, MPEG-DASH can be encoded in any format, which makes it a standard to be used across several platforms easily.

The DASH player at the client side, is responsible for identifying the network conditions in and requesting to switch between bitrates for an uninterrupted experience for the content consumers. MPEG-DASH segments the video content into smaller chunks mostly of interval between 2-10 second rather than fixed 6 second in case of HLS. Hence, it can be easily configured to carry out the segmentation at different interval levels. Mostly an index for different quality levels is maintained by DASH which varies from 2 MB to 20 MB.

Problem Description -

1. **Problem Statement**
   To perform experimental evaluation of Multipath TCP for MPEG-DASH traffic under varying traffic loads and BBR as congestion control for background traffic.

2. **Objectives**

1. To gain an in-depth understanding of the working of Multipath TCP and its components.
2. To gain an in-depth understanding of the working of MPEG-DASH (Dynamic Adaptive Streaming over HTTP).
3. To emulate Multipath TCP using Linux network namespaces and utilities.
4. To check the compliance of Linux Upstream implementation of MPTCP with RFC 8684.
5. To implement different scenarios with varying loads and test the performance of DASH on MPTCP.
6. To gain an in-depth understanding of the behavior of BBR congestion control in background traffic.

Experimental Setup

In this chapter, we provide a concise summary of the experimental configuration used to carry our work. Here we present the details of the experimental testbed, including the necessary components for streaming and the tools used for emulating different types of traffic. Furthermore, we outline the background traffic scenario that we considered for experimentation and provide a table describing its characteristics. Additionally, we provide an overview of the video dataset that we used for streaming, along with DASH configurations used for evaluating the streaming performance. 4.1 Tools and Testbed The majority of existing literature has utilized either virtual machines or physical testbeds to establish their network topology. In contrast, we have opted to employ Linux network namespaces for our experimental setup. This approach is a lightweight and highly scalable alternative to physical testbeds and virtual machines. Since it is a virtualization of the kernel stack itself, it has the capability to produce the same results as other methods. The experiments have been performed on a computer running Ubuntu 23.04 with Linux kernel version 6.0.9 having the upstream implementation of MPTCP.

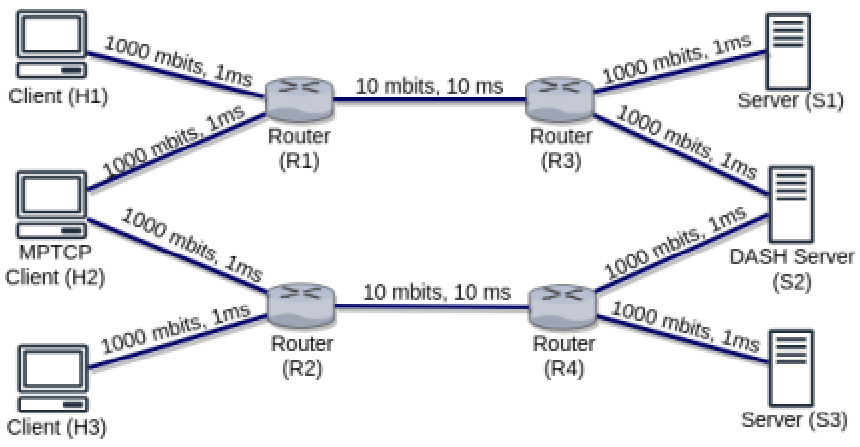


Fig: 4.1 Network Topology

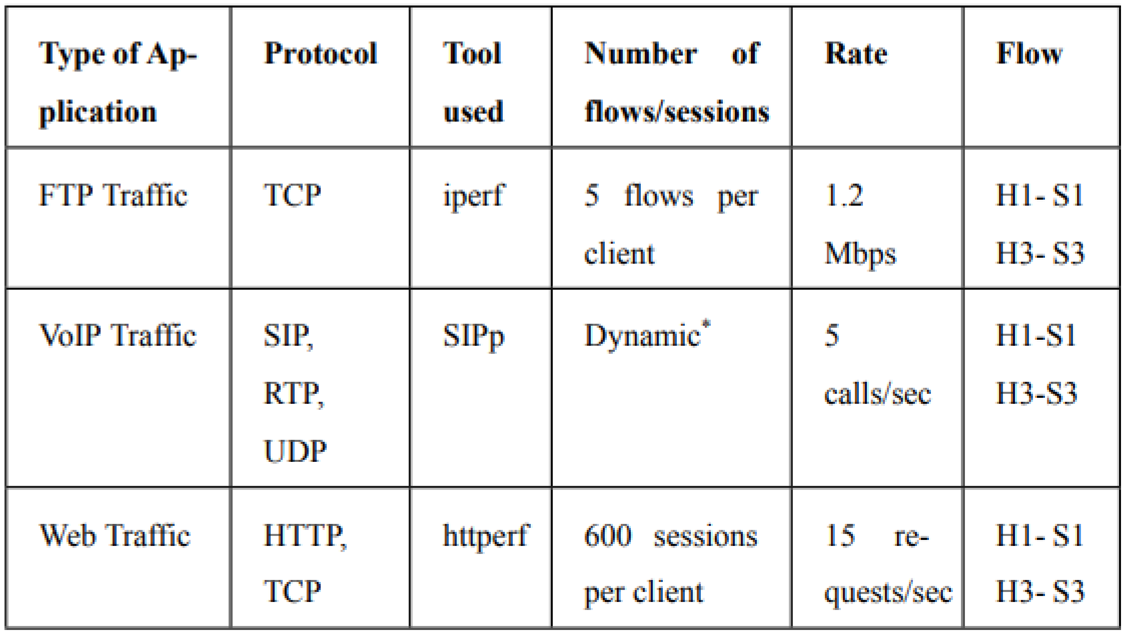
The topology consists of three servers, three clients connected via four routers as shown in Figure 4.1. In our topology client 'H2' and server 'S2' act as the MPTCP enabled endpoints each having two independent paths connecting them. We have adjusted the bandwidth and delay of each link using traffic control subsystem (tc) in Linux accordingly as shown in Figure 4.1. This ensures that congestion is created at routers 'R1' and 'R2' in the presence of the background traffic. We term 'H2-R1-R3-S2' as the primary link/subflow as it is used for establishing the initial connection between the two MPTCP endpoints, and the link 'H2-R2-R4-S2' as the secondary link/subflow. The server 'S2' is an HTTP server containing the encoded video segments for DASH streaming. We have used a Python3.11 HTTP server serving on a persistent connection. Server 'S1' and 'S3' are used for emulating the background traffic, each running different server programs, for serving FTP, VoIP and Web traffic. We have used a GPAC client as the DASH player running on client 'H2'. The other clients 'H1' and 'H3' are used for generating other types of background traffic. DASH, VOUT and HTTP logs generated by GPAC client were parsed by Python script to scrap the statistics and plot the results using GNUPlot.

Experimental Scenario

We have tried to create a testbed with different types of traffic in order to emulate a real Internet scenario. Such background traffic should be representative of the typical usage patterns observed on the Internet, such as web browsing, video streaming, file transfer, and other applications that generate unpredictable network traffic. To achieve this, we incorporated three dominant types of application traffic on the Internet - FTP, VoIP, and web traffic. However, defining the characteristics of each type of traffic, such as data rate and the number of flows, can be a challenging task for a testbed like ours. To address this issue, we referred to the ns-2 TCP evaluation suite (Wang, 2007) to obtain specifications for different types of traffic.

The different types of traffic and its specifications are shown in Table 4.1. FTP traffic was generated using iperf (Iperf) tool, which is a commonly used network performance testing utility in Linux. FTP makes use of TCP and acts as Elephant Traffic (Erman et al., 2007) in the network. In our experimentation, 5 FTP flows are created between host ‘H1’ and server ‘S1’ and 5 FTP flows between ‘H3’ and ‘S3’. Each of these flows have a sending rate of 1.2 Mbps, and continue sending data throughout the course of the experiment. VoIP traffic was generated using a utility called SIPp (SIPp) which is an open source tool for testing SIP protocol. However, along with SIP, we also generate RTP data over UDP to emulate the real audio traffic present over the Internet. SIPp client makes 5 calls/second to the SIPp server and transmits the necessary RTP data and finally tears down the call once done. VoIP traffic is continuously being generated throughout the experiment. Finally, we generate web traffic using httperf (Linux) which is again a Linux utility for testing HTTP protocol. It continuously sends dummy HTTP GET requests and gets back dummy data as response. In httperf a session is defined as collective time for sending certain requests. Here a session runs for 2 seconds. A total of 600 sessions are fixed and in each session the client sends 15 requests per second.

Table 4.1: Background Traffic Configuration

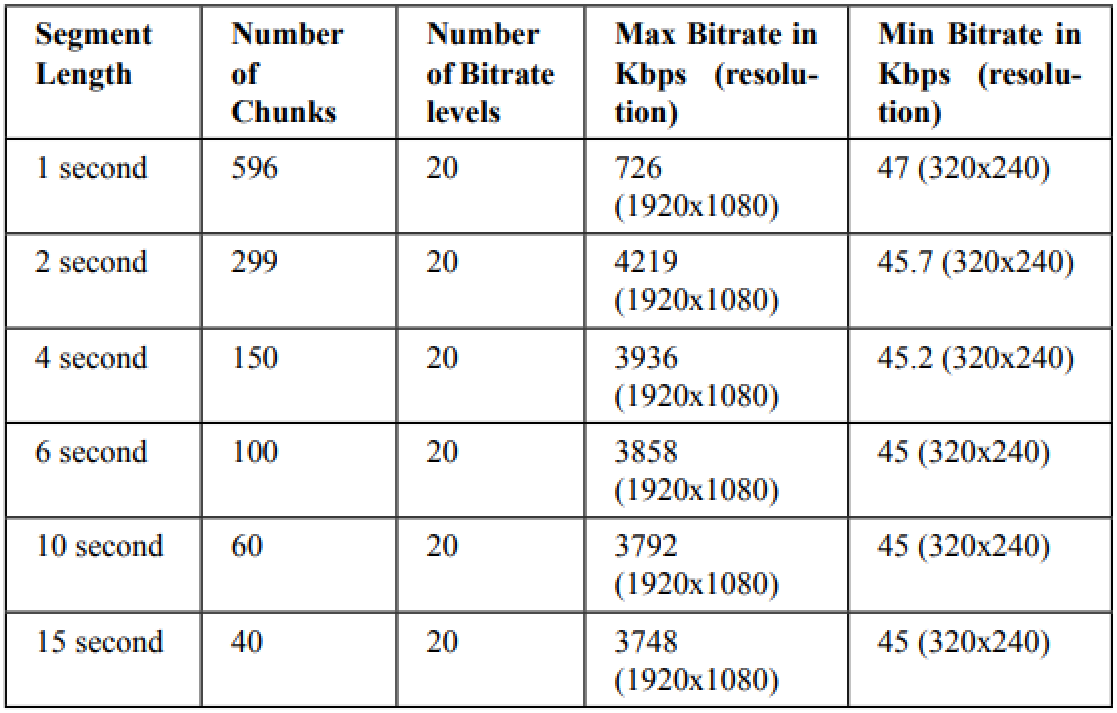


Type of Application	**Protocol**	**Tool used**	**Number of flows/sessions**	**Rate**	**Flow**
FTP Traffic	TCP	iperf	5 flows per client	1.2 Mbps	H1- S1 H3- S3
VoIP Traffic	SIP, RTP, UDP	SIPp	Dynamic*	5 calls/sec	H1-S1 H3-S3
Web Traffic	HTTP, TCP	httperf	600 sessions per client	15 requests/sec	H1- S1 H3- S3

* VoIP traffic does not have any fixed number of flows, it keeps generating calls at the specified rate.

This approach can help to simulate a realistic environment where the background traffic competes with the streaming traffic resulting in congestion and packet loss. This enables us to carry out a comprehensive evaluation of the protocol’s performance under conditions that more closely resemble real-world usage scenarios.

Table 4.2: Big Buck Bunny Dataset



Segment Length	Number of Chunks	Number of Bitrate levels	Max Bitrate in Kbps (resolution)	Min Bitrate in Kbps (resolution)
1 second	596	20	726 (1920x1080)	47 (320x240)
2 second	299	20	4219 (1920x1080)	45.7 (320x240)
4 second	150	20	3936 (1920x1080)	45.2 (320x240)
6 second	100	20	3858 (1920x1080)	45 (320x240)
10 second	60	20	3792 (1920x1080)	45 (320x240)
15 second	40	20	3748 (1920x1080)	45 (320x240)

DASH Dataset

To evaluate the performance of DASH streaming, we required streaming data that covered various resolutions, bitrate levels, and segment lengths. One approach was to create the dataset using a DASH encoding tool like ffmpeg (FFmpeg), but a better option was to use a standard pre-encoded dataset. Using a standard dataset allowed us to establish a standard benchmark for comparison with previous research. Therefore, we utilized the BigBuckBunny dataset (Lederer et al., 2012), which has been previously used in multiple studies, enabling us to compare our results with previous work and draw meaningful conclusions. The BigBuckBunny dataset contains six different segment lengths, ranging from one to fifteen seconds, each of which is encoded into twenty different bitrate levels. The details for each segment length in the dataset can be found in Table 4.2.

Streaming Configuration

Before we proceed to discuss our findings, it is crucial to highlight the streaming configurations that we employed in our experiments. The MPTCP endpoints were configured with a 22 fullmesh Path Manager and default scheduler. Additionally, we used the uncoupled congestion control algorithm supported by TCP upstream implementation, which maintains an independent congestion window for each flow. TCP BBR congestion control is used for background traffic i.e for h1, s1, h3 and s3 links we use TCP Cubic (Rhee et al., 2018) for MPTCP traffic , which is the default congestion control algorithm in Linux, was used in our experiments for h2 and s2 links. There are several other configurations of the GPAC player that may affect streaming performance. GPAC allows us to select the bitrate adaptation algorithm which adjusts the playback quality with changing network conditions. For our experimentation we made use of the rate adaptation algorithm (grate) provided by GPAC. The playback buffer threshold is fixed at 1 second and the maximum player buffer capacity is kept as 10 seconds. That is if the data in the buffer falls below threshold, the video playback freezes. However, if it is greater than threshold 1 second worth of data is taken up for playback and continues to request more segments. Once the buffer gets completely filled, it stops requesting more segments. Entire experimentation is carried over HTTP/1.1 persistent server.

Results and Analysis

In this chapter, we present the results obtained from the experiments, analyze the underlying factors that contribute to the observed behavior, and draw conclusions from them.

Evaluation Metrics

We used the following QoE metrics for comparing the performance of DASH streaming over TCP and MPTCP:

• Number of Bitrate Switches
• Average Bitrate
• Average Throughput
• Application Jitter
• Playback Buffer Level
• Application RTT
• Number of Interruptions.

Results

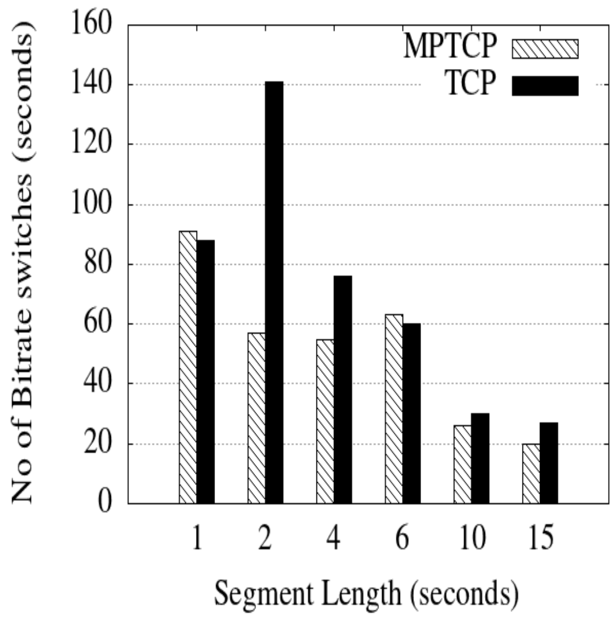


Fig 5.1: Number of Bitrate Switches

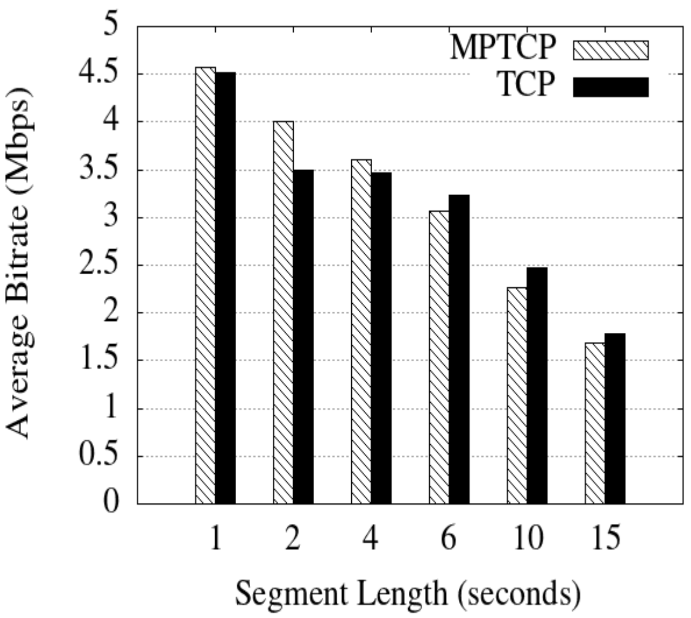


fig 5.2: Average Bitrate

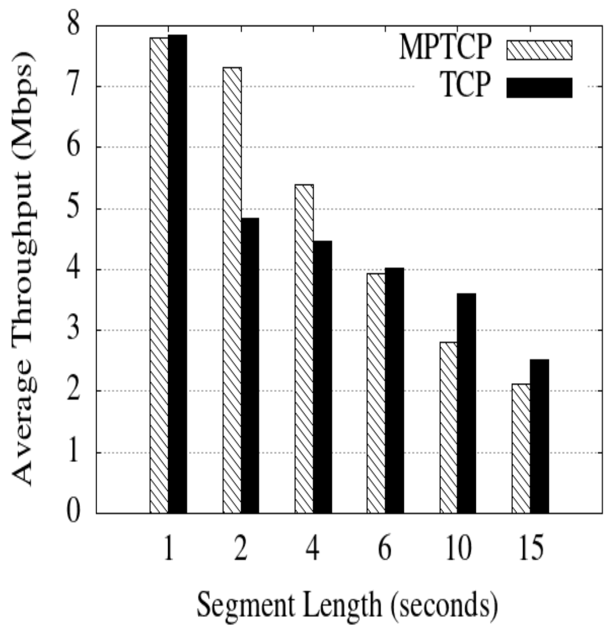


Fig 5.3: Average Throughput

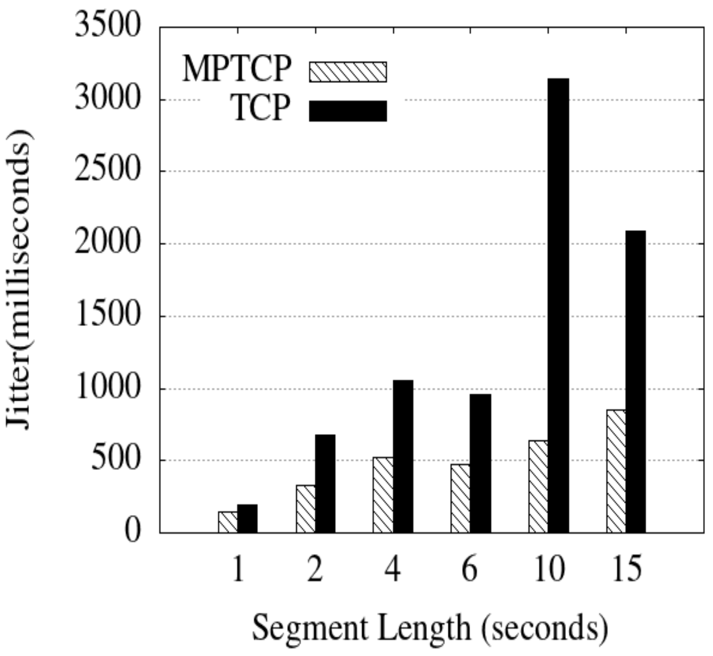


fig 5.4: Application Jitter

As it can be seen in Fig 5.1 the frequency of bitrate switches decreases with increasing segment length . Our results show that the highest frequency of switches occurred when the segment was one second for TCP and two seconds for MPTCP. However this frequency decreased significantly when the segment length increased to three seconds and continued to decrease with further increases in segment length. This is attributed to the use of small segment lengths, which generate a large number of chunks/segments , providing the adaptation algorithm with more opportunities to adjust the network requirements, resulting in a higher frequency of switches. Notably, the frequency of switches for MPTCP is consistently lower than that of TCP, with a significant difference observed for segment lengths between 1-2 segments, which gradually becomes equivalent to TCP with the increment in segment length.

MPTCP outperforms TCP in providing a higher average bitrate for smaller lengths. However as the segment length increases they show quite close performance. Since we are using the grate adaptation algorithm, which dynamically estimates the throughput and based on the current throughput requests for the next lower bitrate level segment from the server.hence the average bitrate is highly dependent on the throughput estimation and shows a similar curve.

MPTCP outperforms TCP in providing a higher throughput consistently for smaller segment lengths 1-6 seconds as the segment length increases and for higher segment length TCP gives better throughput as compared to MPTCP . It can also be observed that in the presence of background traffic, the throughput improvement using MPTCP is not as significant as expected compared to TCP.

Application jitter is significantly very low on using MPTCP compared to TCP as seen in Fig 5.4 This is mainly because of the secondary subflow. As the RTT fluctuates due to the presence of background traffic in the primary subflow, MPTCP utilizes the secondary link. This results in reduced jitter in case of MPTCP compared to TCP.

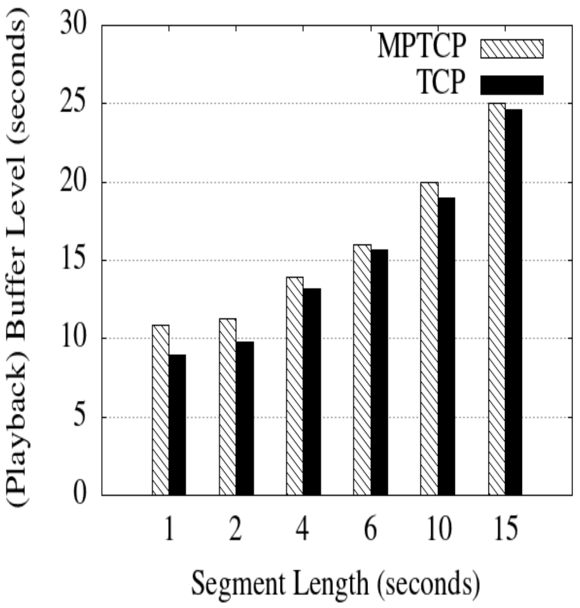


Fig 5.5 : (Playback) Buffer Level

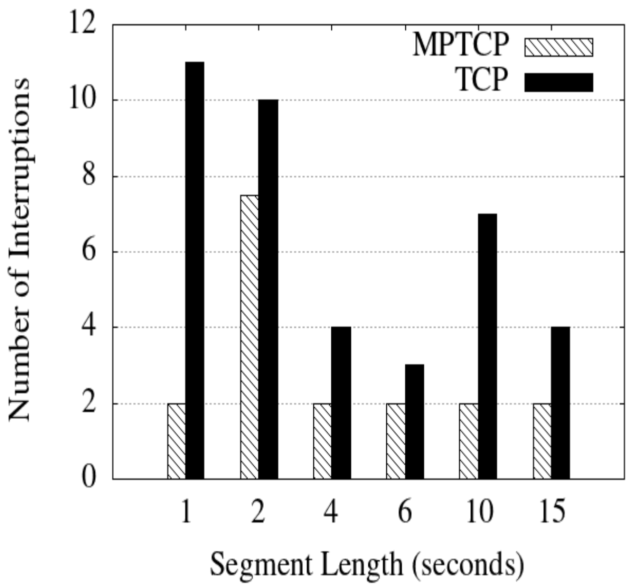


Fig 5.6: Number of Interruptions

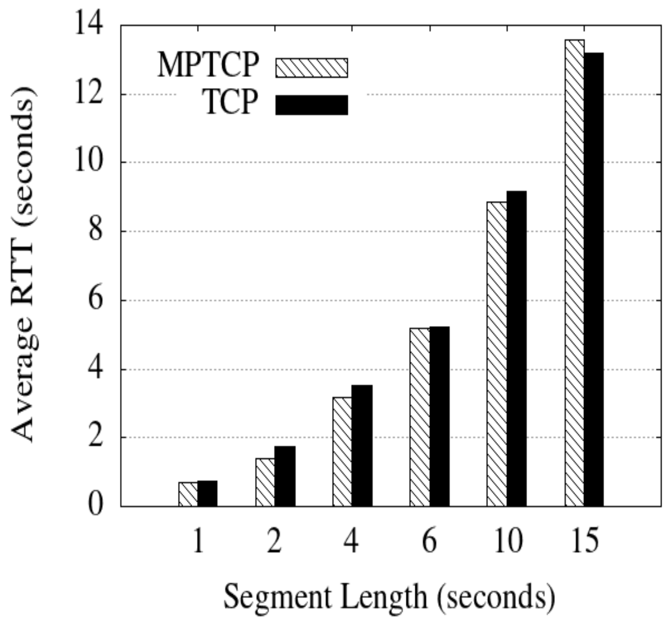


Fig 5.7 Average RTT

Fig 5.5 clearly shows that MPTCP maintains a higher buffer level for all the segment lengths and hence provides an improved playback and reduced jitter. As the segment length increases, the buffer level increases . This is mainly due to the playback buffer configurations. The playback buffer level goes on increasing as the segment lengths increase.

Number of Interruptions is very high in case of smaller segment lengths for TCP. As the segment length increases the number of interruptions is quite similar for different segment lengths in case of MPTCP. When the RTT exceeds 1 second, which is also the buffer threshold, there may be instances where the buffer level drops below the threshold before the arrival of the next segment. Such occurrences result in significant interruptions in playback.MPTCP is able to maintain the number of interruptions quite minimally and much better than TCP.

Average RTT is quite similar for both TCP and MPTCP. As the segment length increases the RTT goes on increasing.

Inferences

Based on the results obtained and discussed above , we get the following inferences:

- MPTCP performs better than TCP in many cases. MPTCP suffers from low bitrate switches as compared to TCP due to this its performance is high.
- Segment length is an important factor to consider when using MPTCP for DASH streaming and choosing the appropriate segment length can improve the overall streaming performance. When very small segment lengths are used in DASH, MPTCP performs better than TCP and for

higher segment lengths MPTCP and TCP gives quite similar throughput. Jitter is very low in case of MPTCP as compared to TCP.

- The performance improvement provided by MPTCP over TCP is not as promising for DASH streaming, in the presence of mixed traffic. Unlike other studies where MPTCP was found to perform twice as better as TCP in the best-case scenario.

Conclusions and Future Work:

In this study we evaluated how DASH streaming performs over MPTCP in diverse network conditions with varying types of network traffic. This likely includes scenarios with different levels of background traffic and congestion.Our evaluation is the first to use the linux MPTCP upstream implementation and thoroughly analyze its performance using different quality of experience parameters. We also have examined the effect of segment length on the overall QoE and our results indicate that lower segment length performs better in case of MPTCP as compared to TCP. However, for other segment lengths it is observed that MPTCP is highly affected by background traffic resulting in performance similar to TCP, and not providing expected benefits. This can be attributed to the fact that the MPTCP default scheduler is not well-suited for streaming applications such as DASH. Additionally, MPTCP faces issues such as Head-of-line-blocking, making it unsuitable for real time applications .This evaluation can provide valuable insights for the design of a scheduler specifically suited for video streaming.