EFFICIENT ARCHITECTURE WITH A GENERAL OPTIMIZED REDUNDANT ERROR BASED MPTCP SCHEDULER

Neha R. Thakur and Ashwini S. Kunte
Department of Electronics and Telecommunication, University of Mumbai, India

Abstract

The path scheduler of Multipath TCP (MPTCP) is responsible to distribute packets in an optimized way on available multiple sub-flows. Due to heterogeneous scenarios of MPTCP sub-flows, MPTCP should have a good packet scheduler that can handle versatile network sub-flow conditions. This article looks over different MPTCP schedulers and finds key problems like Out-of-order (OFO) packets issue, Receiver side blocking, sub-flows, and bandwidth disunion. These problems appeal to the design of good MPTCP architecture providing an efficient packet scheduler. This article proposes an MPTCP architecture with a General and optimized MPTCP scheduler which overcomes the problems of some known schedulers. The proposed Redundant Error-based MPTCP Scheduler (REMS) meets the three goals, first, it achieves ranking of available paths based on congestion window and outstanding packets on it, second goal is smart sequencing schedule with redundant error calculation. REMS experimentation shows, it improves throughput and quality of experience of application with bandwidth aggregation and decreasing application delay considerably. This article also shows the comparison of the proposed scheduler with previously known schedulers.

Keywords:

Redundant Error, Packet Holding, Multipath Packet Scheduler, Start Sequence, End Sequence

1. INTRODUCTION

MPTCP is the concept derived from TCP. Today’s smart devices are technologically evolved. These devices can use several internet interfaces simultaneously by using Internet Protocol (IP) addresses, this capability is called multi-homing. To explore the flavor of this facility and utilize available resources in an optimized way, Internet Engineering Task Force (IETF) [1] has given the standard MPTCP, its open-source code is available in multipath “www.multipath-tcp.org”. MPTCP is the hidden upper layer of TCP. In MPTCP enabled host, MPTCP is hidden from the application layer and network layer. MPTCP is made up of one or many independent TCP connections, working all together with different interfaces concurrently to provide aggregated capacity, maximize throughput, and enhance the resilience of the network. Multi-homed devices can be better utilized by using MPTCP for speedy and seamless communication. MPTCP uses a connection that is logically made up of many TCP connections. These TCP connections are established with a three-way handshake method. Each TCP connection in MPTCP is called a sub-flow of MPTCP. Each sub-flow in MPTCP uses a different IP address at the end host. Over

MPTCP connection, the sequence number for data is called a connection sequence number or data sequence number and at sub-flow level sequence number for data is called sub-flow sequence number. MPTCP connection is established between two MPTCP enabled end hosts. Initially, only one TCP flow is started in MPTCP connection and as per requirement, more TCP flows are joined in MPTCP as multiple sub-flows. Middleboxes will treat individual MPTCP sub-flow as TCP connection. MPTCP is transparent to the Network layer and Application layer.

1.1 WHY MULTIPATH TCP?

MPTCP over TCP has two main advantages, capacity aggregation, and redundant connection. Wi-Fi and LTE are two different interfaces with their capacity. TCP cannot use them simultaneously when both of them available but MPTCP can use both of them simultaneously and aggregate their capacity. Thus, MPTCP may give theoretically maximized throughput and application performance improved. MPTCP uses many TCP connections running concurrently, so if any TCP connection fails, other redundant TCP connections are there to serve the application. IOS uses MPTCP in the SIRI application for redundancy. Linux code is available for MPTCP.

MPTCP’s architecture is made up of three major building blocks, path manager, congestion control, and packet scheduler. Path manager decides to add sub-flows to MPTCP architecture. Congestion control decides the total number of packets that should be present on each sub-flow per RTT. Path scheduler is accountable for efficient distribution of packets on available sub-flow. Improper path scheduler can lead to degradation of the Quality of service of the application.

1.2 PATH MANAGER

MPTCP connection between two systems Host A and Host B is established by establishing a single path TCP connection first. single connection adds further sub-flows with ADD ADDR command option of MPTCP. Path manager takes path management decisions. MPTCP is provided with three different path managers [2].

- **Default Path Manager**: This type of path manager does not broadcast IP addresses also it does not add new sub-flows. It accepts the new creation of sub-flows.
- **Full-Mesh Path Manager**: All available IPs of the client can make the connection with the available IPs of a server.
- **Path Manager**: For a specific sub-flow, the same IP address pair is used but with different TCP ports always. This path manager.

MPTCP uses three methods to control path usage

- **Primary Mode**: It is the default mode and it utilizes all available interfaces.
- **Backup Mode**: Only active sub-flows are chosen to transmit data and other passive sub-flows are redundant to act as backup.
- **Single Path Mode**: MPTCP acts like single-path TCP by using a single active sub-flow at a time to transmit data.

TCP has only one path, so path manager is not required but in MPTCP multiple sub-flows are available and there is a need for path manager to manage them as shown in Fig.1. Each sub-flow in MPTCP is associated with Source IP and Destination IP pair. MPTCP's Default path manager will add new sub-flow on the request of a new interface.

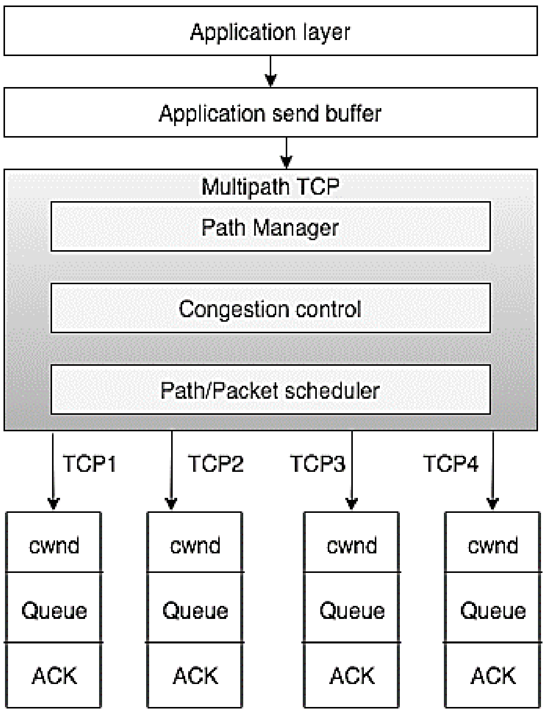


Fig.1. Default MPTCP with default scheduler

1.3 CONGESTION CONTROL IN MPTCP

The capacity of MPTCP sub-flows is different so we required good congestion control technique. If congestion control is independent for each sub-flow, it will create the head of line blocking at the receiver end because of the link within a group of sub-flows of MPTCP. MPTCP uses coupled congestion control. It is robust to failure and it uses resource pooling in terms of available congestion window by a congested path. By using coupled congestion control MPTCP may achieve targets like

- MPTCP should not dominate other single-path TCP connections in the network.
- MPTCP should work at least as that of single-path TCP.
- MPTCP should give seamless delivery of traffic by the congested path by accounting upper two targets.

Few congestion control algorithms for TCP and MPTCP are LIA, OLIA, BALIA, and D-LIA [3]. All these algorithms follow the Additive Increase and Multiplicative Decrease (AIMD) method. AIMD has three phases slow start, congestion avoidance, and fast retransmission phase.

In the slow start phase congestion window of sub-flow doubles till the slow-start threshold, once the first packet drops congestion avoidance phase starts. In the congestion avoidance phase, the congestion window of sub-flow increases by one for each RTT. Unacknowledged packets on sub-flow are more than the capacity of congestion window of sub-flow then the fast retransmission phase starts. During the phase, new packets are not transmitted on any of the sub-flows.

1.4 PATH SCHEDULER

In MPTCP multiple sub-flows are used, which are heterogeneous concerning path delays, congestion windows, and bandwidths so packets send on many sub-flows will not come in proper sequence at the receiver end, higher-order packets will be transmitted first, and lower order packets may be transmitted with delays. This may cause many out of orders packets and head-of-line blocking at the receiver end. If the receiver does not have enough space to hold OFO packets then due to request time out, many and spurious retransmissions may occur, and the loss ratio will increase.

This drawback of the head of line blocking can be avoided by using an efficient MPTCP scheduler in MPTCP architecture. MPTCP sends data by using bandwidth aggregation of many TCP paths so a scheduler is necessary for MPTCP. MPTCP scheduler is triggered when data arrives at the sender side from the application. Many authors have given their algorithms and ideas on the MPTCP scheduler. They have used various key performance factors in different combinations for implementing the MPTCP scheduler. They achieved goals like bandwidth aggregation, receiver buffer optimization, loss recovery, etc. but no one achieved these goals altogether.

In this article, authors propose an MPTCP scheduler that is capable of doing bandwidth aggregation, head of line monitoring, and energy conservation by considering the feedback of the MPTCP receiver.

Further, in this paper, the contents are organized as follows. Related work gives an overview of existing techniques in this area. The methodology section explains the proposed MPTCP approach in detail step by step. The following sections are methodology, results, and discussion of the proposed approach. Finally, the conclusion section gives remarks about the experimentation undertaken and the future work.

2. RELATED WORK

In MPTCP architecture many packet schedulers are studied. MPTCP protocol is implemented in the Linux kernel. Various MPTCP scheduler algorithms for throughput optimization [4] and reduced download time [5] have been proposed by researchers. Numerous performance characteristics can influence throughput and application download time (QOS) using MPTCP. They were viewed differently by various scholars. Few authors used all available sub-flows, but some of them used only one or two effective sub-flows out of many sub-flows. Bandwidth aggregation is accomplished by using multiple sub-flows. Some authors used sender buffer [6] as a sub-flow selection parameter, while others used receiver buffer for sub-flow quality estimation [7].

The majority of them estimate sub-flow efficiency using sub-flow characteristics such as Round-Trip Time (RTT), Congestion Window (cwnd), and queue size. Few send all application layer packets in sequence from the sender side, others send them in mixed order so that all packets arrive at the receiver end in the correct order. Few authors designed schedulers that took into account the status of the router's buffer as well as the buffers of the end devices [8]. Some schedulers use a proactive strategy [9], while others use a reactive strategy [10]. Some schedulers prioritize fresh packets over retransmitted packets during packet transmission from the sender end. Some schedulers can also retransmit packets on a new route instead of the old path. Few schedulers assign a priority to applications [11]. Programmable Multipath [12] is a system that allows us to select one scheduler from a large number of schedulers based on our needs and network conditions. It is a medium through which we can create new Scheduler and test them in the ProgMP system. Many authors have contributed to MPTCP schedulers, but only a few schedulers are theoretically compared in Table.1.

Few MPTCP schedulers are studied with their functionality, advantages, and disadvantages.

- **MinRTT MPTCP Scheduler**: This scheduler chooses a fast path as a priority. This scheduler sends traffic on a fast path until it is used fully, then sends traffic on the next fast path till it gets exhausted. In heterogeneous scenarios due to the large difference in round trip time of sub-flows, many OFO packets get accumulated at the MPTCP receiver, and they cause head of line blocking and receiver buffer overflow. In MinRTT scheduler, this situation is handled using penalization of the slow path and fast retransmission of packets on a fast path (PR) and because of this most of the time fast path is utilized and the slow path is used for very less amount of time. The Key performance parameter used in the scheduler is RTT. Its work is shown in Fig.2.

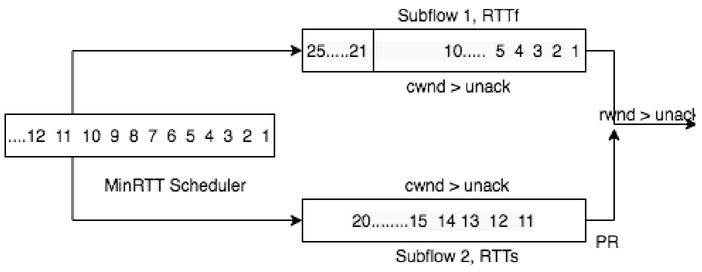


Fig.2. MPTCP MinRTT scheduler

- **Round Robin [20]**: This scheduler does not prefer any sub-flow. It sends packets irrespective of their preferences in a Round Robin fashion. Due to heterogeneity slow sub-flows get congested and the scheduler losses its performance. Its methodology is shown in Fig.3.
- **Blind Round Robin**: It is MPTCP scheduler which considers three performance metrics like receiver congestion window congestion window and inflight packets or unacknowledged packets on the sub-flows. It collects data at sender side then it selects best sub-flow by checking its congestion window, if congestion window is not exhausted by unacknowledged packets, then packets are sent on the same sub-flow otherwise packets are sent on the other alternative sub-flow. For all sub-flows this procedure is repeated while allocating packets on the sub-flows. If receiver window is exhausted then all sub-flows are blocked for a while. Receiver window blocking is tested with respect to summation of unacknowledged packets of all the sub-flows in MPTCP. Its work is shown in Fig.4.

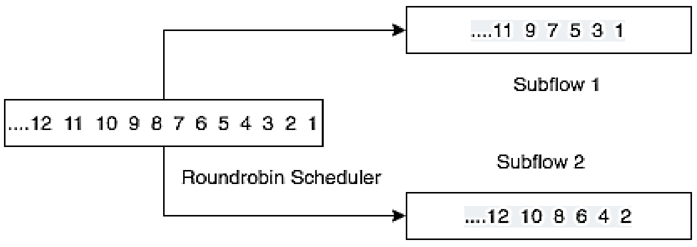


Fig.3. Round Robin MPTCP scheduler

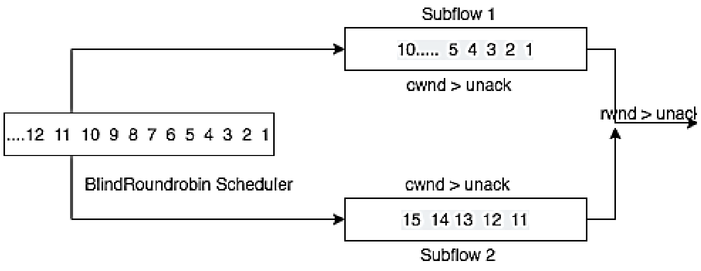


Fig.4. Blind Round Robin MPTCP Scheduler

- **Redundant [21]**: Redundant scheduler in MPTCP gives perfect redundancy in the network for transmitting data packets. The redundant scheduler broadcasts the packets on every available sub-flow of MPTCP. The receiver sends combined acknowledgment of received packets on any one of the sub-flows. Bandwidth is wasted in this type of scheduler. Bandwidth aggregation is 100% in this scheduler. Many other schedulers use this scheduler as fast retransmission policy which is shown in Fig.5.

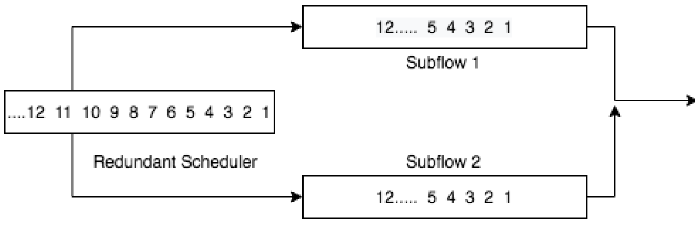


Fig.5. Redundant MPTCP Scheduler

- **Delay Aware Packet Scheduler in MPTCP (DAPS) [14]**: Delay aware packet scheduler does bandwidth aggregation by using fast and slow both sub-flows simultaneously. DAPS scheduler uses longest sub-flow's One Way Delay ($OWD = RTTs/2$). It sends $n = (RTTs/RTT_f)$ packets on fast sub-flow and remaining packets on slow sub-flow. It is assumed that packets to be sent are more than the congestion window of fast sub-flow, then only slow sub-flow is utilized otherwise fast sub-flow is enough to send all data. Schedule 'S' contains which sequence number chunk to be sent on which sub-flow in RTTs/2 duration of time. DAPS do not consider MPTCP send Window and path loss. DAPS takes care of OFO packets at the receiver, it sends OFO packets at the sender to receive 'in order packets' at the receiver. DAPS do not exhaust fast sub-flow fully but it puts only n packets on fast sub-flow and remaining packets on slow sub-flow. Its work is shown in Fig.6.

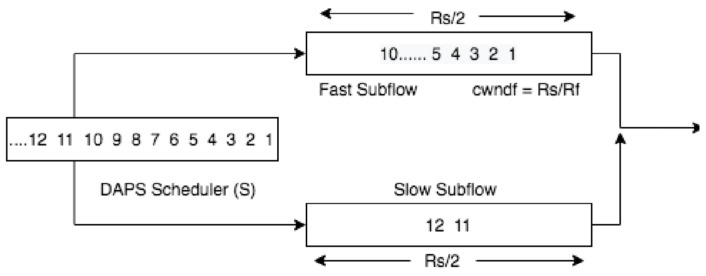


Fig.6. Delay Aware MPTCP Scheduler

- **Blocking Estimation Based MPTCP scheduler (Blest) [22]**: MinRTT does PR when it comes across head of line blocking and because of it MinRTT uses mainly fast sub-flow and neglects slow sub-flow. It hardly uses slow sub-flow. Blest avoids PR by managing OFO packets at the receiver. In a homogeneous scenario, Blest avoids HOL blocking by restricting OFO packets. Blest initiates algorithm with MinRTT but as its congestion window fills, it goes to slow sub-flow and calculates the number of segments 'X' that may be transferred on slow sub-flow. Before transferring data 'X' on slow sub-flow, it checks whether to wait for fast sub-flow or to transfer the current data on the slow sub-flow. This decision is taken based on the congestion window of fast sub-flow and RTTs/ RTTf factor for RTTs duration, that is during longest sub-flow OWD. Path probing is done by using the congestion window of fast sub-flow cwnd. Its work is shown in Fig.7.

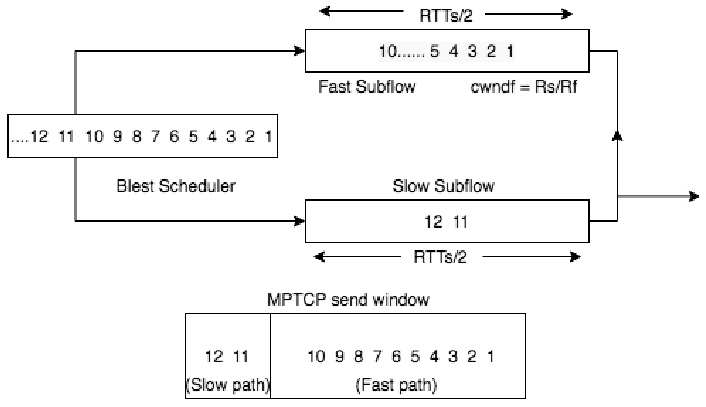


Fig.7. Blest MPTCP Scheduler

- **Loss Aware MPTCP Scheduler (LAMPS) [23]**: LAMPS switches from MinRTT to Redundant and Redundant to MinRTT MPTCP scheduler. This switching decision is taken based on path loss factor. Considering sub-flow's loss factor, transfer time '*T*' of all sub-flows is calculated. The Least transfer time sub-flow is selected for transmission of data and the algorithm followed is MinRTT. As path loss of fast sub-flow increases beyond the desired threshold value then the redundant scheduler starts working in the MPTCP connection. Prior scheduled packets are also recovered at the current time and sent on all sub-flows simultaneously. The path probing method used is also good. It is described in Fig.8.

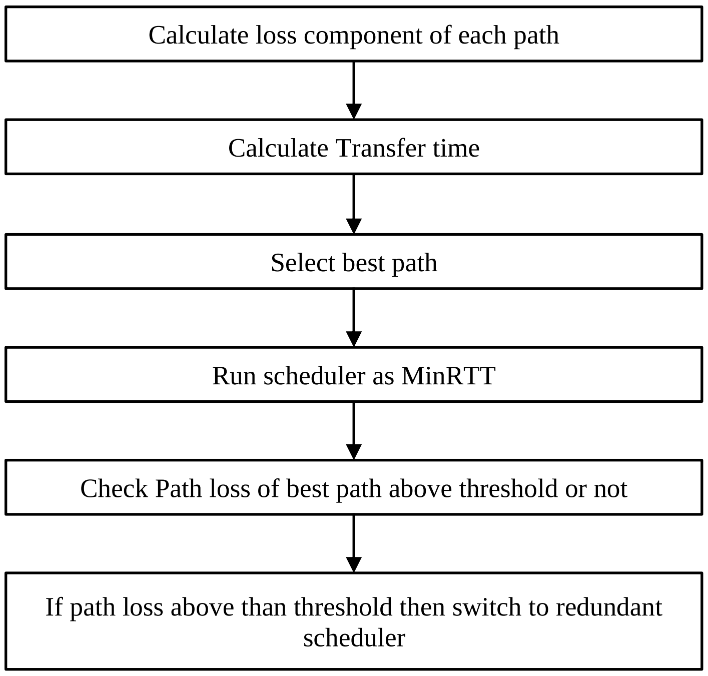

Calculate loss component of each path
Calculate Transfer time
Select best path
Run scheduler as MinRTT
Check Path loss of best path above threshold or not
If path loss above than threshold then switch to redundant scheduler

Fig.8. LAMPS MPTCP Scheduler

3. BACKGROUND AND MOTIVATION

Comparison of various know schedulers is done with deep experimentation and found some observations from the analytical study. MPTCP is set up using the Linux kernel implementation. Two computers with 8 GB RAM, a Raspberry

Pi, and an ADSL router are used in the experiment. MPTCP is available on both machines, with Linux kernels and MPTCP versions of 4.19.105 and MPTCP-v95, respectively. A homogeneous scenario is generated with (Wi-Fi, Wi-Fi) interfaces, Heterogeneous scenario is generated with (Ethernet, Wi-Fi) and (LTE-Wi-Fi) interfaces. Experiments are run for two sets, 20 seconds and 200 seconds for each MPTCP scheduler to download different data sizes traffic generated by the 'iperf' traffic generator [24].

3.1 STUDY AND OBSERVATIONS OF EXPERIMENTATION

Observations are listed based on key performance factors.

3.1.1 Throughput:

For known scheduler of MPTCP like MinRTT, Round Robin, Redundant, and Blest throughput is recorded in kilobytes for different data sizes like 64KB, 128 KB, 256 KB, 1 MB, and 256 MB. It is observed that when data load is less all the schedulers give almost similar performance.

Table.1. Theoretical Comparison of MPTCP Schedulers

MPTCP Schedulers	Method	Performance Parameters	Advantages	Disadvantages
Round Robin	Sub paths are used alternately	CWND	Capacity aggregation.	It is very poor for heterogeneous transmission.
MinRTT	Use of fastest sub path	RTT, CWND	Easy to implement.	Heterogeneity restricts even small file to download.
REMP (Redundant) [13]	It uses all paths to transmit same data.	-	Speedy but with move overheads	More redundant packets so more overhead of buffer.
CP [14]	Sends data in proactive manner	Buffer capacity	Probing of fast path.	Mobility is not considered
DAPS [15]	Tried to utilize all sub paths	RTT	No receiver block	It doesn’t work in all network scenarios.
ECF [16]	Uses shortest path to send the data	RTT, cwnd, MSS, amount of data to be sent	Fast completion time	May underutilize some of the sub-flows.
OTIAS [17]	OWD is used, min RTT is considered without considering cwnd.	RTT	Less parameters.	Many out of order packets.
Blest [18]	Gives priority to fast sub path	RTT, MSS	Receiver blocking is avoided	Slow sub path is underutilized
DEMS [19]	All sub flows complete the task at same time.	Capacity of paths	Good for specific setups.	Chunk size of data and exact bounds matters.

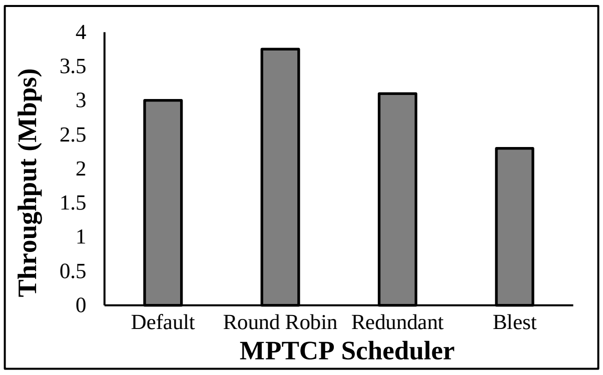


Fig.9. Throughput of MPTCP schedulers for data download of size 64KB

In all the observations considering different data sizes, Round Robin scheduler gives outstanding performance for 64 KB data size downloads. It is shown in Fig.9. As the data size of download increases, Round Robin is not a stable MPTCP scheduler whereas in this scenario Blest gives better performance.

In a heterogeneous scenario of Ethernet and Wi-Fi interface, throughput is recorded for two sets of experiments like 20sec and 200sec data transfers and it has been observed that Blest, MinRTT and Redundant scheduler perform well but Round Robin scheduler degrades its performance drastically. As shown in Fig.10 the Round Robin scheduler could not withstand with a heterogeneous scenario in comparison with other MPTCP schedulers. Among all scheduler as heterogeneity increases Blest performs well in terms of throughput.

Another heterogeneous scenario is run for 200 seconds using Wi-Fi and LTE networks as shown in Fig.11. In this experiment, Blest and MinRTT scheduler works well but Redundant and Round Robin schedulers give poor performance.

In another set of an experiment to measure throughput homogeneous scenario is used. It is shown in Fig.12. A homogeneous scenario is created by using Wi-Fi and Wi-Fi interfaces. It is found that the Round Robin scheduler performs very well in all schedulers and gives maximum throughput.

3.1.2 Download Time:

Four different schedulers of MPTCP MinRTT, Round Robin, Redundant, and Blest are run for downloading different data sizes like 64KB, 128 KB, 1 MB and 256 KB of data. It has been observed that for a small size of data that is 64 KB download time, all schedulers behave similarly as shown in Fig.13. As data size for download increases Round Robin and Blest degrade their performances. For moderate data sizes like 128 KB and 256 KB Blest gives the least download time than other schedulers as shown in Fig.14.

3.1.3 Path Utilization and Bandwidth Aggregation:

In homogeneous and heterogeneous scenarios, Blest and MinRTT schedulers utilize only fast path with the least RTT for maximum time. These two schedulers use slow path for a very small fraction of time, whereas the Round Robin scheduler tried to use slow and fast paths equally. The Redundant scheduler uses all the sub-flows exactly for an equal amount of time as shown in Fig.15.

3.2 MOTIVATION AND FINDINGS BASED ON OBSERVATIONS

All schedulers provide seamless redundancy in the network by using multiple available sub-flows. In homogeneous scenarios, Round Robin’s performance is best among Blest, Round Robin, Redundant, and MinRTT schedulers but as heterogeneity increases, Round Robin scheduler starts degrading its performance.

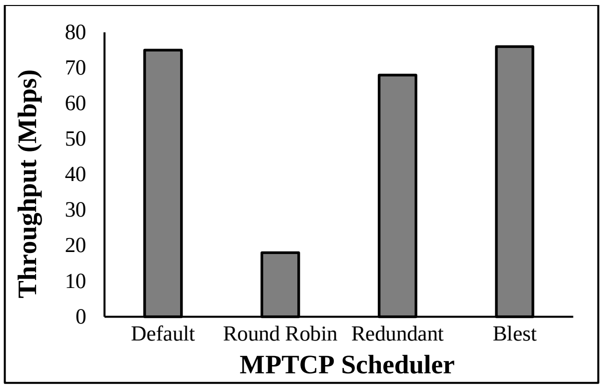


Fig.10. MPTCP scheduler Throughput in heterogeneous scenario

For the download of small data sizes up to 64 KB all the four schedulers perform almost in a similar way. As data size of download increases Round Robin scheduler becomes unstable and Blest scheduler gives a moderate performance. For 256 MB data size Blest also degrades in performance.

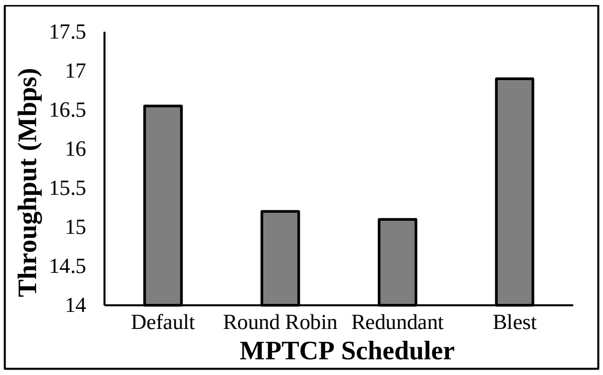


Fig.11. Throughput of MPTCP schedulers in heterogeneous scenario

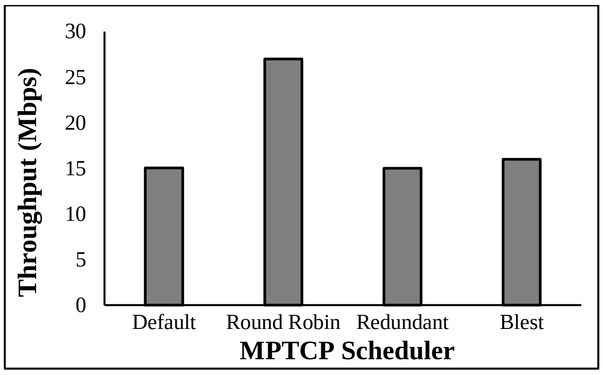


Fig.12. Throughput of MPTCP schedulers in homogeneous scenario

The Round Robin scheduler and Redundant scheduler give good bandwidth aggregation. Redundant scheduler gives 100% bandwidth aggregation. Blest and MinRTT schedulers used fast path for maximum time and slow path for very little time so that bandwidth aggregation is almost nil.

None of the above schedulers used path loss as a key performance parameter. Blest and MinRTT schedulers use only RTT as the performance measure parameter. There is the requirement of an MPTCP path scheduler, which can withstand homogeneous as well as heterogeneous network scenarios.

Round Robin scheduler is the best option in homogeneous scenarios. As heterogeneity increases in the MPTCP network, the scheduler should handle the OFO packets at the receiver also receiver buffer should be used in an optimized way.

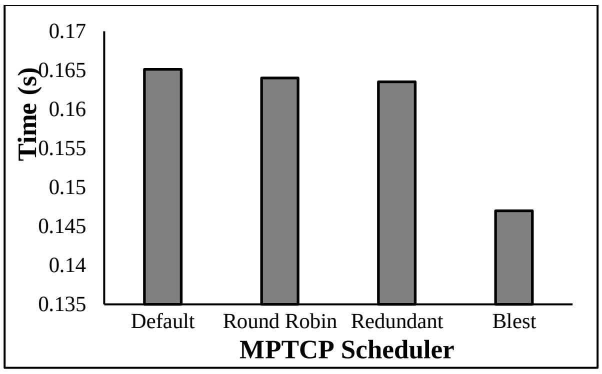


Fig.13. Download time in seconds of MPTCP schedulers for data download of size 64KB

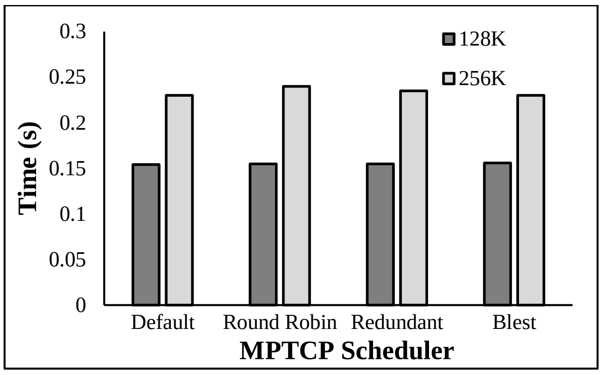


Fig.14. Download time in seconds of MPTCP schedulers for data download of size 128KB and 256KB

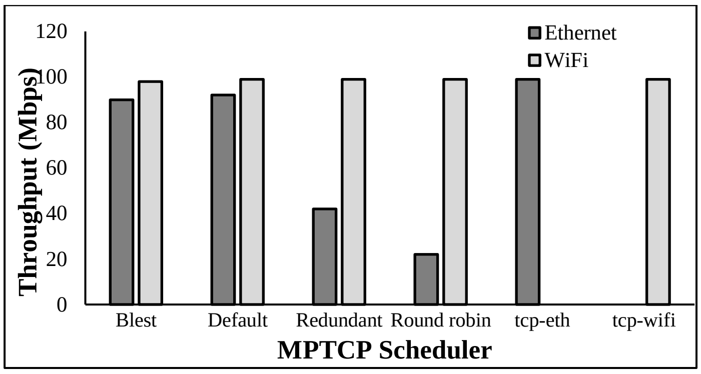


Fig.15. Path utilization and bandwidth aggregation of different MPTCP schedulers

At the sender side, receiver buffer can be mapped with MPTCP send window and OFO packets at the receiver can be reduced. When we leave the fast path due to congestion, the fast path should be traced continuously with a good probing method. So there is the need for efficient MPTCP architecture with a good scheduler which can utilize all available sub-flows efficiently in coordination with the MPTCP congestion control method and it should use receiver buffer in an optimized way by giving receiver feedback to the sender buffer to avoid receiver buffer overflow. MPTCP scheduler should use some sequencing logic to avoid OFO packets at the receiver side so that many retransmissions are avoided and application download time will decrease and throughput will increase.

4. METHODOLOGY

4.1 OBJECTIVES OF EFFICIENT MPTCP ARCHITECTURE

- The effective scheduler should have very small schedules so that recovery of schedules would be easier during a network failure.
- The effective scheduler should have spontaneous schedules.
- The scheduler should have receiver feedback incorporated.
- The scheduler should be more from redundant to Round Robin.
- Path selection and recommendation should be autonomous and functional in the scheduler.
- Application delay = Wait time(queue) + RTT + ACK time, since RTT and ACK time is constant, the scheduler must work on Wait time at receiver buffer.

4.2 WORKING OF AN EFFICIENT MPTCP ARCHITECTURE

As shown in Fig.16, the application buffer gives data to TCP send buffer. TCP send buffer transmits the data to MPTCP send buffer, which consists of Send queue, inflight queue, and retransmission queue. This data is given to scheduler. Scheduler takes multivariate input to schedule data on different sub-flows in different RTT cycles.

Path manager unit gives information of number of added sub-flows with fixed congestion window to the scheduler. Congestion window unit gives the information of current capacity of available sub-flows to the scheduler. Receiver buffer sends acknowledgment for specific RTT to the path scheduler and asks next data on the sub-flows. Each sub-flow consists of its congestion window, queue, and an acknowledgment part. These three things will decide the current capacity of the sub-flows and this information is provided to the congestion control unit from sub-flow.

4.3 WORKING OF MPTCP PATH SCHEDULER

An efficient scheduler will work in three functional phases: redundant phase, overambitious phase and Round Robin phase.

Suppose there are two paths with heterogeneity in RTTs and cwnds, cwnd1 and cwnd2 as shown in Fig.17. There is a common queue of the send buffer. Initially, both the paths will start fetching the common queue data from the start sequence, so the scheduler will work in a redundant phase. In this phase, a redundant error is negative. After some time, a fast path will receive the acknowledgment of initial sent data and sends new data on the same fast path whereas transmission of previous data on a slow path is still going on. When the slow path finishes old data transfer, it will come to know about other fast sub-flow which is ahead in data transfer, so it leaves the gap of redundant error and transfers next sequence of packets from common queue. In this way, scheduler enters into an overambitious mode with positive redundant error. Later when redundant error reduces to zero, the scheduler works in Round Robin mode. It is desirable to work the scheduler in Round Robin mode and reduce

the error to zero. It is observed from previous experimentation that Roun robin among all schedulers gives good throughput and good bandwidth aggregation in a homogeneous scenario. The proposed path scheduler has two essential modules, the path Sorting module and the redundant error module as explained in the next subsections.

4.3.1 Path Sorting Module:

It takes inputs such as congestion window and outstanding packets of each sub-flow and it calculates the current capacity of each sub-flow. Workflow of this module is shown in Fig.18 and the algorithm of the module is given in Algorithm 1. Every sub-flow maintains its queue. The total number of RTTs required to transmit the data on a particular sub-flow is calculated based on its queue, cwnd, and current capacity. The arrival time for data transfer on each sub-flow is calculated by using Total RTTs. Finally, all available sub-flows are sorted in ascending order of arrival times. If two sub-flows will have the same arrival times then their congestion windows are compared and the maximum congestion window sub-flow is preferred. 'Eq.(1)-Eq.(3)' are used to get minimum arrival time sub-flow.

Algorithm 1: Path Sorting Module

Capacity of the $path_j$

$$C_j = cwnd_j - unackedpackets_j \tag{1}$$

Total number of RTTs required:

$$T_i = (packets\_buffer_j - c_j)/cwnd_j \tag{2}$$

Arrival time of $path_i^j$:

$$(AT_i) = (T_i) * sRTT \tag{3}$$

4.3.2 Redundant Error Module:

In Fig.19, there are two sub-flows, sub-flow1 and sub-flow2. Sub-flow1 has cwnd1=2 and Sub- flow2 has cwnd2=3 and for every respective RTT their cwnd increase by 1. Both sub-flows fetch data from the common queue. First sub-flow1 and sub-flow2 transmit packets 1, 2 and 1, 2 and 3 simultaneously from a common queue so they are in redundant mode. Sub- flow1 is fast and it gets acknowledgment of packets 1,2 in its fast RTT, sub-flow2 is still transmitting packets 1, 2 and 3.

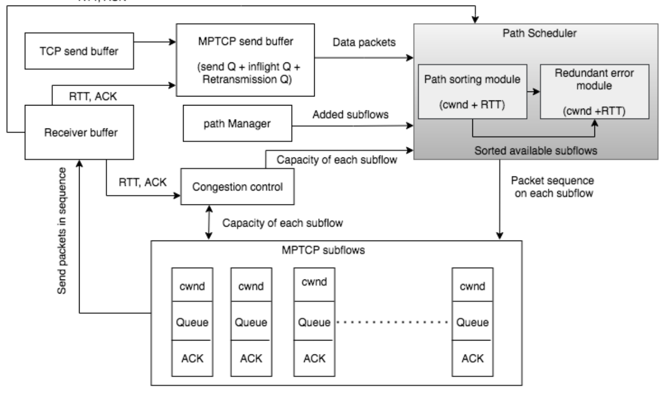


Fig.16. Efficient MPTCP with redundant error scheduler

Sub-flow1 takes new packets 3, 4, 5 for transfer. In next turn sub-flow2 completes its data transfer of packets 1, 2 and 3 and it will come to know that another fast sub-flow is available who can transmit 5 more packets than it, so, now sub-flow will transmit next 4 packets 6, 7, 8, 9 and sub-flow2 will leave the gap of redundant error = 5 in a common queue and sends next 3 packets from 11. Redundant error calculation table is also shown in Table.2.

The module in Fig.20 initializes the start and end sequence number of all available sub-flows also it will initialize redundant error. Then after iterating all available sorted sub-flows, redundant error and their start and end sequence numbers are updated. Thus, planned packets are sent on each sub-flow out of order so that they will be received at the receiver in sequence, and application Wait time reduces and throughput increases. Packets are scheduled on all available sub-flows so bandwidth aggregation is also achieved. The workflow is illustrated in Algorithm 2. The Eq.(4)-Eq.(8) are used to calculate sequence numbers on sub-flows and redundant errors.

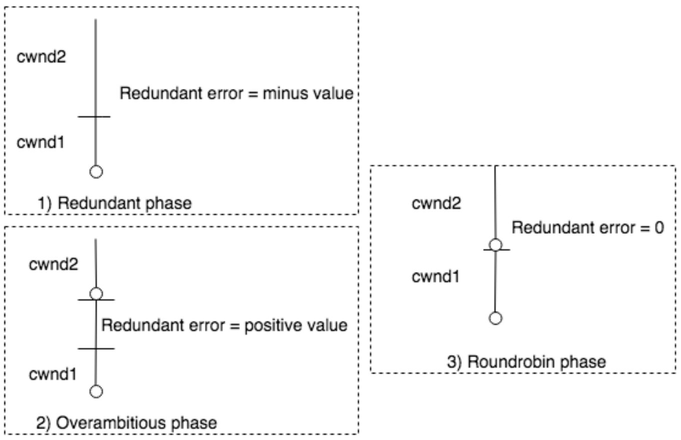
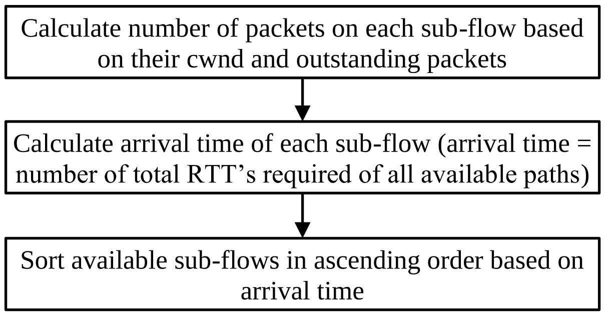


Fig.17. Three phases of Efficient MPTCP scheduler

Start sequence of fast sub-flow is calculated as:

$$seq_{11} = Data\_sequence \tag{4}$$



Calculate number of packets on each sub-flow based on their cwnd and outstanding packets

Calculate arrival time of each sub-flow (arrival time = number of total RTT's required of all available paths)

Sort available sub-flows in ascending order based on arrival time

Fig.18. Path sorting module

End sequence of fast sub flow is calculated as:

$$seq_i^2 = seq_i^1 + cwnd_1 + inflight_1 \tag{5}$$

Start sequence of other sub flows is calculated as:

$$seq_i^1 = \max(dataseq,\ seq_{i-1}^1 + abs(red\_error) + 1) \tag{6}$$

End sequence of other sub flows is calculated as:

$$seq_i^2 = seq_i^1 + cwnd_i - inflight_i \tag{7}$$

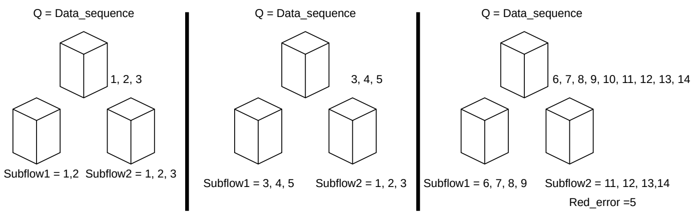


Fig.19. Redundant error calculation

Redundant error for sub flow is calculated as:

$$red\_error_i = seq_i^1 - seq_{i-1}^2 \tag{8}$$

Theoretical calculation of application delay: Theoretical comparison of different known schedulers with a proposed scheduler concerning download time is shown in Table 3. Initialization is done as follows.

- RTT Path1(fast) = 5ms, W1 = 5
- RTT Path2(slow) = 20ms, W2 = 20
- Receiver window = 60

Table.2. Redundant error calculation

Time	seq11	seq21	seq12	seq22	Ack1	Ack2	Ack	error
0-5	1	5	1	5	5	0	5	-4
10	6	11	22	27	11	5	11	-10
15	12	17	22	27	17	5	17	5
20	18	23	28	33	23	27	27	-1
25	28	33	28	39	33	27	33	-5
30	34	39	51	56	39	39	39	-11
35	40	45	51	56	45	39	45	6
40	46	51	57	62	51	56	56	0

Table.3. Download time of MPTCP schedulers calculated theoretically

MPTCP Scheduler	Download time (ms)
Default	60
Blest	50
Round Robin	60
Redundant	60
REMPS	40

5. EXPERIMENT AND EVALUATION

5.1 EXPERIMENT SETUP

Linux kernel implementation is used for MPTCP setup. This setup enables us to test out extreme scenarios in a safe environment. The Experiment is implemented by using two computers with 8 GB RAM, raspberry pi, and an ADSL router. Both the computers are MPTCP enabled with Linux kernel and MPTCP version of 4.19.105, MPTCP-v95 respectively. Fig.21 shows how we set up a basic topology. A Router is used between MPTCP source host and MPTCP destination host. Each network link is allotted with proper bandwidth and latency with the help of the ’tc qdisc’ command as shown in Table.4. The default interface can also be set by using the ’iproute’ command. With proper setting each time two interface combination is made in experiment setup like1) Wi-F+Ethernet 2) Wi-Fi + LTE and 3) Wi-Fi + Wi-Fi.

Algorithm 2: Redundant error module

```
Step 1: Initialize data_seq = next_chunk_seq from send buffer,
        redundant_error = -1 and all sequence no. = 0.
Step 2: Send redundant sequence no. packets on all available
        sub-flows.
Step 3: Get sorted sub-flows from path sorting module basis
        $cwnd \geq sent\_packets$ on every sub-flow.
Step 4: Calculate start and end sequence of packets on fast sub-
        flow.
Step 5: Calculate start and end sequence of packets on rest of the
        sub-flows
Step 6: calculate redundant_error
Step 7: Send data on all sub-flows
Step 8: Redundant Error Module
```

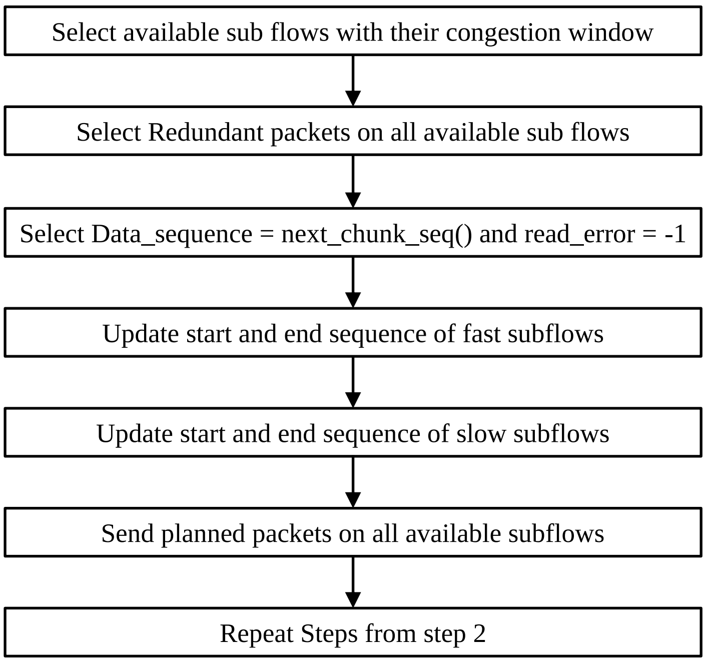

Select available sub flows with their congestion window
Select Redundant packets on all available sub flows
Select Data_sequence = next_chunk_seq() and read_error = -1
Update start and end sequence of fast subflows
Update start and end sequence of slow subflows
Send planned packets on all available subflows
Repeat Steps from step 2

Fig.20. Redundant error module

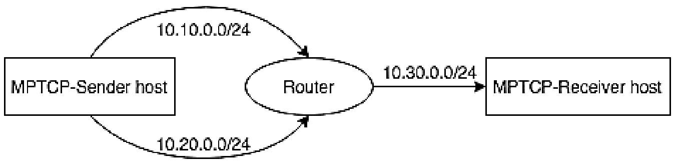


Fig.21. MPTCP setup for Bandwidth aggregation

Table.4. Interfaces used in Experiments

Interface	Bandwidth	RTT
Ethernet	80Mbps	10ms
Wi-Fi	8 Mbps	50ms
LTE	16 Mbps	70ms

The proposed schedulers MPTCP architecture is compared with default MPTCP architecture using known schedulers. Experiments are run in two sets, first for 20 seconds and second for 200 seconds. For each set of experiments iperf3 traffic generator is used with no loss links. Experiments are run for 20 seconds and 200seconds for the proposed scheduler and known schedulers individually with different data sizes 64KB, 100KB, 500KB, 1000KB. Using Python script, System socket command readings are recorded and dataset for different experiments are collected. Collected data is analyzed using ’R tool’ concerning various key performance parameters.

The proposed scheduler’s MPTCP architecture is compared with known scheduler’s MPTCP architectures like

- **Default Scheduler**: It uses the least RTT interface for maximum time. For a very small amount of time, it uses a slow interface. To handle OFO packets and retransmissions it uses Penalization and retransmission algorithm.
- **Round Robin Scheduler**: It uses an alternate interface for data transmission irrespective of knowledge of RTTs of interfaces.
- **Redundant Scheduler**: It broadcast data on all available interfaces in a redundant manner. Combined data acknowledgment of data is sent on anyone sub-flow.
- **Blest Scheduler**: It sends data on fastest sub-flow in priority. If fast sub-flow is not available then estimated transfer time of data is calculated on fast and slow sub-flow and whichever is less that path will be chosen for data transfer.

5.2 EVALUATION

The evaluation of proposed scheduler’s MPTCP architecture is made by comparing proposed scheduler with known schedulers. Comparison of proposed schedulers with known schedulers is made with respect to various key performance parameters like percentage path utilization, RTT behavior, average throughput, average application download time, and unacknowledged packets.

5.2.1 Percentage Path Utilization:

As shown in Table 5, for 20 seconds experiment, it has been observed that in both MinRTT and Blest schedulers, Wi-Fi interface is less used, only 7% in comparison to the Ethernet interface, which is 93%. Ethernet interface has low and consistent RTT whereas Wi-Fi has a large and fluctuating RTT. In a similar type of experiment as per the basic behavior of Round Robin and Redundant schedulers, they utilize both the interface almost equally. Due to Redundant error module designed in proposed scheduler it tries to use both slow and fast sub-flow equally, Wi-Fi 48% and Ethernet 52%. It is proved that proposed algorithm does best bandwidth aggregation in heterogeneous scenario of (Wi-Fi, Ethernet) with proper sequencing of data on available sub-flows using redundant error calculation.

Table.5. Path Utilization of different MPTCP schedulers compared with proposed scheduler for 20sec duration

MPTCP Schedulers	Wi-Fi	Ethernet

Blest	7%	93%
Default	7%	93%
Redundant	46%	54%
Round Robin	61%	39%
TCP LTE	0%	100%
TCP Wi-Fi	100%	0%
Proposed	48%	52%

In 200s experiment, it has been observed that, as per the behavior of MinRTT scheduler, it does not use interface (Wi-Fi) with large RTT, it prefers to use LTE interface for scheduling maximum data. BLEST scheduler also, uses only a minimum RTT interface (LTE). It uses an LTE interface and Wi-Fi interface in a proportion of 88% and 13% respectively as shown in Table.6. Round Robin and Redundant schedulers try to use both the interfaces. Proposed scheduler also uses both the interfaces almost equally. In all the experiments, it has been observed that Redundant scheduler (ReMP) and Proposed scheduler utilize both the interfaces fully. It is observed that in heterogeneous and homogeneous scenarios, proposed scheduler uses both the interfaces in equal contribution. It is possible only because of redundant error calculation and proper sequencing of data packets on available sub-flows of MPTCP architecture.

Table 6. Path Utilization of different MPTCP schedulers compared with proposed scheduler for 200sec duration

Schedulers	LTE	Wi-Fi
Blest	88%	13%
Default	90%	10%
Redundant	50%	50%
Round Robin	38%	62%
TCP LTE	100%	0%
TCP Wi-Fi	0%	100%
Proposed	47%	53%

In Literature [25] MPTCP is a protocol that is implemented in the Linux kernel. Various MPTCP scheduler algorithms for throughput optimization [8] and reduced download time [9] have been proposed by a number of writers. They warn that using a path with a long RTT will always cause the receive buffer size to overflow. Other studies, such as [9] and [14], add to the evidence of this problem. Furthermore, [1], [9], and [14] all provide experimental proof of how MPTCP output degrades when the receive buffer size is small. In proposed scheduler, Receiver buffer overflow is restricted by using all available sub-flows with proper data scheduling on it.

5.2.2 Average Throughput and Application Data Download Time:

Average Throughput achieved is best for the Proposed scheduler among all schedulers. Blest and MinRTT scheduler perform well in comparison to Round Robin and Redundant scheduler in heterogeneous scenarios, their throughput has been observed better than that of Round Robin and Redundant scheduler but their throughput is less than Proposed scheduler. The reason is MinRTT and Blest scheduler use fastest sub-flow for maximum time, and they neglect slow sub flow. Proposed scheduler can get the benefit of bandwidth aggregation of available sub-flows in MPTCP architecture by using proper scheduling of data on available sub-flows using redundant error module. There are very less re-transmission in case of proposed scheduler because of receiver feedback used at sender side to plan the data on different sub-flows. Since receiver buffer is utilized properly and less number of OFOs are at the receiver side, throughput obtained by Proposed scheduler is more in comparison with other MPTCP schedulers. Sometimes Round Robin and Redundant schedulers performance are even below single-path TCP as shown in Table.7.

Table.7. Average Throughput of different MPTCP schedulers compared with Proposed MPTCP scheduler

MPTCP Schedulers	Throughput in Kbps
Default	19.442
Round Robin	16.0985
Redundant	18.876125
Blest	19.37375
Proposed	29.54234

Experiments are also run to download data of size 64KB, 100KB, 500KB, and 1MB, and average download time in seconds of different schedulers of MPTCP, Proposed scheduler of MPTCP and vanilla TCP is observed and compared as shown in Fig.25 and Table 8. The download time of Proposed scheduler is least among all the schedulers. Overall MPTCP Schedulers with Proposed scheduler of MPTCP do perform bandwidth aggregation and reduce download time in comparison to TCP but Proposed MPTCP scheduler downloads application with variable data sizes in the least time because of proper bandwidth aggregation of all available sub-flows in homogeneous and heterogeneous scenarios.

5.2.3 Unacknowledged Packets:

In these experiments, in a heterogeneous environment, unacked packets are being observed more for the Redundant scheduler and unacked packets are very less for Proposed MPTCP scheduler. It is shown in Fig 26 and ’Table 9’. Proposed scheduler uses receiver feedback at the sender side to select proper sub-flow and to calculate redundant error, based on that data is scheduled in proper way on all available sub-flows and because of that OFOs or unacked packets are very less in case of Proposed scheduler of MPTCP.

Table.8. Average application download time of different MPTCP schedulers compared with Proposed MPTCP scheduler

Data	Default	Round Robin	Redundant	Blest	Proposed
64Kb	49.762	49.262	49.085	44.158	32.234
100Kb	52.102	52.468	51.657	51.305	40.561
500Kb	68.276	69.293	72.572	67.6465	51.234
1MB	131.8	139.54	132.67	142.7106	101.22

Table.9. Unacknowledged packets of different MPTCP schedulers compared with Proposed MPTCP scheduler

Schedulers	Homo-geneous	Homo-geneous	Hetero-geneous	Hetero-geneous
Default	27	47	54	65
Round Robin	27	69	53	105
Redundant	46	70	67	120
Blest	19	34	41	54
Proposed	3	5	7	12

6. CONCLUSION

MPTCP architecture should use all available TCP sub-flows in multi-homed devices concurrently and efficiently. Previous MPTCP architectures use different schedulers but while using available sub-flows simultaneously, they generate many OFO packets at the receiver. MPTCP scheduler used in this literature makes MPTCP architecture efficient by using proper packet sequencing, receiver feedback and many key performance factors like RTT, cwnd, OFO packets on each sub-flow. This architecture uses redundant error as key measuring factor. This architecture uses path sorting model and Redundant error model to increase throughput of application and to decrease average download time of the application. This MPTCP architecture with redundant error-based scheduler, improves quality of experience of the application.

REFERENCES

[1] A. Ford, C. Raiciu, M. Handley and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", Available at https://datatracker.ietf.org/doc/html/rfc6824, Accessed at 2013.

[2] A. Ford, C. Raiciu, M. Handley, S. Barre and J. Iyengar, "Architectural Guidelines for Multipath TCP Development", Available at https://www.rfc-editor.org/rfc/rfc6182, Accessed at 2011.

[3] T. Lubna, I. Mahmud and Y.Z. Cho, "D-LIA: Dynamic Congestion Control Algorithm for MPTCP", *ICT Express*, Vol. 6, No. 4, pp. 263-268, 2020.

[4] X. Corbillon, R. Aparicio-Pardo, N. Kuhn, G. Texier and Simon, "Cross-Layer Scheduler for Video Streaming over MPTCP", *Proceedings of International Conference on Multimedia Systems*, pp. 1-12, 2016.

[5] B. Han, F. Qian, L. Ji and V. Gopalakrishnan, "MP- DASH: Adaptive Video Streaming over Preference-Aware Multipath", *Proceedings of International Conference on Emerging Technologies*, pp. 1-14, 2018.

[6] H. Kim, J.B.H. Oh and A. Lee, "Improvement of MPTCP Performance in Heterogeneous Network using Packet Scheduling Mechanism", *Proceedings of International on Conference on Communications*, pp. 842-847, 2012.

[7] F. Yang and P. Amer, "Work in Progress: Using One-Way Communication Delay for In-Order Arrival MPTCP Scheduling", *Proceedings of International on Conference on Communications and Networking*, pp. 122-125, 2014.

[8] T. Shreedhar, N. Mohan, K. Sanjit, J. Kaul, and Kangasharju, "QAware: A Cross-Layer Approach to MPTCP Scheduling", *Proceedings of International on Conference on Networking and Workshops*, pp. 1-9, 2018.

[9] W. Yang, P. Dong, W. Tang, X. Lou, H. Zhou, K. Gao and G. Wang, "A MPTCP Scheduler for Web Transfer", *Computers, Materials and Continua*, Vol. 57, No. 2, pp. 205-222, 2018.

[10] F. Yang, P. Amer and N. Ekiz, "A Scheduler for Multi-Path TCP", *Proceedings of International Conference on Computer Communication and Networks*, pp. 1-7, 2013.

[11] W. Lu, D. Yu, M. Huang and B. Guo, "PO-MPTCP: Priorities-Oriented Data Scheduler for Multimedia Multi-Pathing Services", *International Journal of Digital Multimedia Broadcasting*, Vol. 2018, pp. 1-9, 2018.

[12] A. Frommgen, A. Rizk, T. Erbshauber, M. Weller, B. Koldehofe, A. Buchmann and R. Steinmetz, "A Programming Model for Application-Defined Multipath TCP Scheduling", *Proceedings of ACM/IFIP/USENIX Conference on Middleware*, pp. 134-146, 2017.

[13] D. Wischik, C. Raiciu, A. Greenhalgh and M. Handley, "Design, Implementation and Evaluation of Congestion Control for Multipath TCP", *Proceedings of Conference on Networked Systems Design and Implementation*, pp. 1-8, 2011.

[14] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani and R. Boreli, "DAPS: Intelligent Delay-Aware Packet Scheduling for Multipath Transport", *Proceedings of IEEE International Conference on Communications*, pp. 1222-1227, 2014.

[15] F. Yang, Q. Wang and P.D. Amer, "Out-of-Order Transmission for In-Order Arrival Scheduling for multipath TCP", *Proceedings of International Conference on Advanced Information Networking and Applications Workshops*, pp. 749-752, 2014.

[16] Y.S. Lim, E.M. Nahum, D. Towsley and R.J. Gibbens, "ECF: An MPTCP Path Scheduler to Manage Heterogeneous Paths", *Proceedings of International Conference on emerging Networking Experiments and Technologies*, pp. 147-159, 2017.

[17] B.H. Oh and J. Lee, "Constraint-Based Proactive Scheduling for MPTCP in Wireless Networks", *Computer Networks*, Vol. 91, pp. 548–563, 2015.

[18] P. Hurtig, K.J. Grinnemo, A. Brunstrom, S. Ferlin, O. Alay and N. Kuhn, "Low-Latency Scheduling in MPTCP", *IEEE/ACM Transactions on Networking*, Vol. 27, No. 1, pp. 302-315, 2019.

[19] Y. Guo, A. Ethan, Z. M. Nikravesh, F. Mao, S. Qian, and K. Sen, "DEMS: Decoupled Multipath Scheduler for Accelerating Multipath Transport", *Proceedings of International Conference on Mobile Computing and Networking*, pp. 477-479, 2017.

[20] C. Paasch, S. Ferlin, O. Alay and O. Bonaventure, "Experimental Evaluation of Multipath TCP Schedulers", *Proceedings of ACM SIGCOMM Workshop on Capacity Sharing*, pp. 27-32, 2014.

[21] A. Frommgen, A. Buchmann, T. Zimmermann and K. Wehrle, "ReMP TCP: Low latency Multipath TCP", *Proceedings of IEEE International Conference on Communications*, pp. 1-7, 2016.

[22] S. Ferlin, O. Alay, O. Mehani and R. Boreli, "BLEST: Blocking Estimation-Based MPTCP Scheduler for Heterogeneous Networks", *Proceedings of IFIP Networking Conference and Workshops*, pp. 431-439, 2016.

[23] E. Dong, M. Xu, X. Fu and Y. Cao, "LAMPS: A Loss Aware Scheduler for Multipath TCP over Highly Lossy Networks", *Proceedings of IEEE Conference on Local Computer Networks*, pp. 1-9, 2020.

[24] V.H. Tran, Q.D. Coninck, B. Hesmans, R. Sadre and O. Bonaventure, "Observing Real Multipath TCP Traffic", *Computer Communications*, Vol. 94, pp. 114-122, 2016.

[25] D. Yao, X. Su, B. Liu and J. Zeng, "A Mobile Handover Mechanism based on Fuzzy Logic and MPTCP Protocol under SDN Architecture", *Proceedings of IEEE International Conference on Communications and Information Technologies*, pp. 141-146, 2018.