User-Controlled Subflow Selection in MPTCP: A Case Study

Kshithija Liyanage[1]*, Tharindu Wijethilake[#2]*, Kasun Gunawardana[3]*, Kenneth Thilakarathne[4]*, Primal Wijesekera[5†], Chamath Keppitiyagama[6]*

***Abstract*— It is common to find multiple network interfaces connected to different Internet Service Providers (ISPs) in devices such as smartphones. Multipath TCP (MPTCP) enables TCP connections to use all these network interfaces in a single TCP connection in an application transparent manner. MPTCP schedules traffic of one TCP connection over subflows created over these network interfaces. It is evident that this requires some scheduling policy. There have been some attempts to allow applications to decide on the scheduling policy. However, this violates the application transparency of MPTCP, and applications do not have all the information required to decide on such a policy. In addition, this allows the applications to monopolize the network connection thus posing a security threat as well.**

We argue that only the owner of the device (the user) has the right to make that policy decision and only the user can make an informed decision on the scheduling policy. For example, the user has the information on the monetary cost of the connections through different interfaces.

In this paper we present a mechanism that allows the user to provide hints to the TCP scheduler to alter its scheduling policy. While this is not a mechanism to implement generic scheduling policies, it demonstrates how a user can guide the scheduling policies. As a proof of the concept, we demonstrate how MPTCP scheduler can be influenced to select a less stable and lossy path over a stable path based on a user preference.

Keywords*— *Multipath TCP, MPTCP Subflow, MPTCP Scheduler

I. Introduction

Consider a smartphone with two network interfaces; cellular and WiFi. This device is multihomed - i.e it is connected to the Internet through two different ISPs via the two network interfaces. It has the potential to reach a destination via the two ISPs utilizing the two network interfaces thus increasing the available bandwidth and the path redundancy.

Despite having multiple interfaces and network connections, communication of a particular application may be limited to a single network interface or connection due to the limitations imposed by underlying communication protocols and their implementations. For example, a TCP connection opened by an application can use only a single endpoint since a TCP connection is tied to a single IP address at one end.

There is a number of techniques to overcome these limitations and reap benefits of having multiple network interfaces. For example, assume that a device with two network interfaces. In such a device, it is possible for an application to be aware of both the interfaces and open two TCP connections to a destination through those interfaces. The application itself can schedule traffic over these two connections according to a policy. However, such a solution is neither scalable nor portable.

Link aggregation or channel bonding can be used to increase the bandwidth of a particular communication [1]. Channel bonding/aggregation techniques combine several network interfaces/connections in order to increase the bandwidth where the implementation may be through hardware, software or both. The traffic division of channel bonding will happen at the network layer or data link layer with respect to the OSI model, depending on the implementation i.e. if it is an Ethernet connection, the switch and the operating system of the host machine have to be configured to use channel bonding. However, in the above mentioned scenario channel bonding does not provide a solution. Since the interfaces are connected to two different ISPs it is impossible to bond the two interfaces at network or data link layers. Therefore, there have been a number of attempts in finding a solution at the transport layer and consequently MPTCP has been introduced[4].

Transport Layer Protocol (TCP) is confined to a single network interface in traditional TCP/IP implementation. However, MPTCP supports establishing TCP connections over multiple network interfaces. It is an extension that has been proposed by IETF to enable the Transport layer to utilize multiple network interfaces available in a device and improve the reliability of communication for its applications by enabling multiple redundant paths [4]. Moreover, the proposed extension also maintains the abstraction provided by the transport layer to the upper layers/applications through a mechanism that aggregates multiple TCP connections. Therefore, applications are not aware of multiple TCP connections. Thus applications need no modifications to use MPTCP.

Currently MPTCP is available for Linux operating systems which should be installed and configured deliberately. Apple iOS uses MPTCP for their voice assistant 'Siri'[17]. Apple iOS is claimed to be the first mobile operating system which implements MPTCP [18]. Further, Apple states that if an application needs to use its MPTCP, then the application needs to be written in support of using MPTCP [19] which implies that the particular application on Apple mobile platform is aware of MPTCP. i.e. Transport layer protocols are made visible at the application layer to a certain extent to have a better control over MPTCP in catering specific application requirements.

Correspondence: T. Wijethilak[#2] (E-mail: tnb@ucsc.cmb.ac.lk)
Received: 14-01-2-2021 Revised:16-03-2021 Accepted: 17-03-2021

This paper is an extended version of the paper "Priority Based Subflow Selection in MPTCP: A Case Study" presented at the ICTer 2020 conference.

K. Liyanage[1*], T. Wijethilake[#2*], K. Gunawardana[3*], K.Thilakarathne[4*], and C. Keppitiyagama[6*] are from University of Colombo School of Computing (UCSC). (kshithijal@gmail.com, tnb@ucsc.cmb.ac.lk, kgg@ucsc.cmb.ac.lk, kmt@ucsc.cmb.ac.lk, chamath@ucsc.cmb.ac.lk)

P. Wijesekera[5†] is from University of California, Berkeley & International Computer Science Institute, USA. (primal@cs.berkeley.edu)

DOI: http://doi.org/10.4038/icter.v14i2.7225

The emergence of MPTCP inclines the Transport layer's involvement in end-to-end routing decisions by allowing it to have one or more communication paths, subflows. However, MPTCP has to make these decisions based on the information available for traditional TCP. This makes a proclivity for MPTCP to select more stable communication paths, resulting from inheriting TCP design, despite having more viable alternative paths with a less opportunistic cost. We believe that enabling/assisting such decision-making ability at the Transport layer would help MPTCP attain its objectives.

We envisioned the benefits of users being able to take part in decision making with respect to the MPTCP path selection by providing their preference. We make a clear distinction between the user and applications. In this work user refers to the person who owns or controls the device and uses the applications. For example, in a circumstance where an ad-hoc network and a 4G mobile data connection are available for the user to communicate, the user would have chosen to connect through the ad-hoc network if the user is more concerned about the cost rather than the delay in communication. Only the user, not the individual applications on the device, has the information to make such a decision. Hence it is crucial to get user input in scheduling traffic over multiple network interfaces. However, current GNU/Linux MPTCP implementation does not have provisions for such information propagation from user to the MPTCP scheduler in the kernel.

Wijethilake et al. [8] and Neira-Ayuso et al. [7] discuss techniques of passing information from user space to kernel space by modifying the socket implementation which requires modification to the existing applications.

However, we believe that the user should be involved in the path selection decision where applications should not be affected by the changes made at the lower layers of the network stack. Therefore, an alternative way of passing a user provided hint from the user space to the kernel space, bypassing the application, was studied by Liyanage et al. [25]. In this study, we further explore that approach and, a novel MPTCP scheduling algorithm that takes user hints to decide and prioritize subflows.

II. Background

Mobile device usage has been continuously growing and, as of the year 2019, there were 4.68 billion [4] mobile users around the globe. Nowadays, most mobile devices are equipped with multiple communication interfaces, such as WiFi, Cellular, and Bluetooth. Generally, WiFi communication is cheaper than Cellular communication. Due to mobile devices' widespread usage, building an Ad-hoc network and establishing a communication path is possible. Using an Ad-hoc network can make communication more cost-effective, evading expensive communication via ISPs. However, this introduces several challenges, such as prioritizing network packets' flow through the Ad-hoc connection by using the MPTCP and giving the authority to choose when to prioritize the Ad-hoc connection to the user's mobile device. Thus, this study aims to combine the infrastructure and Ad-hoc network connections to prioritize the flow of network packets through the Ad-hoc network and obtain the user's hint in deciding when to prioritize the Ad-hoc connection.

A. Multipath TCP

MPTCP uses a TCP subflow per interface to establish an MPTCP connection between two hosts. The individual MPTCP subflows work as independent TCP connections by maintaining their own data structures to attain the reliability of TCP. MPTCP maintains additional data synchronization mechanisms to integrate data pertaining to a particular MPTCP connection that may be received out of order from multiple subflows [4].

In order to create a MPTCP communication, both the hosts must be configured with MPTCP. Otherwise it will use the traditional TCP for the communication. When establishing MPTCP connections over two hosts, it first initiates the connection via one of the available network interfaces. It is called as the first subflow in the context of MPTCP terminology. If both the hosts are compatible with MPTCP, the hosts can initiate another subflow via a second network interface available in the device. As an example, a mobile phone that has two network interfaces (i.e. WiFi interface and a Cellular interface) can initiate the first subflow using the WiFi interface when it establishes MPTCP connection with a server. This connection will examine whether both the devices are compatible with MPTCP. If the mobile phone and the server both are compatible, it will initiate the second subflow via the cellular interface. A client application running on the mobile phone will assume a single network connection. But underneath it has two different TCP connections via two network interfaces which probably operate over two different ISPs as shown in Figure 1.

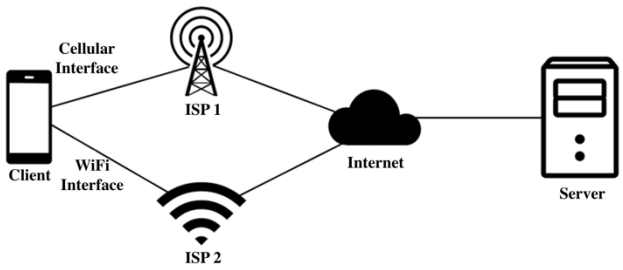

Fig. 1 Mobile phone connecting to a server using MPTCP via two ISPs

A vital attribute of MPTCP is its backward compatibility. If one of the hosts of MPTCP connection is incompatible with MPTCP, it will downgrade the connection to traditional TCP. Even though MPTCP achieves this level of flexibility, it does not change the existing headers of traditional TCP. Therefore, to communicate the control signals related to MPTCP, it uses the *'Options'* field of the TCP header segment. One of the challenges faced when using MPTCP in practice is that the MPTCP traffic could get dropped in the middle, such as a Firewall or Intrusion Prevention System, if those are not aware of MPTCP.

When initiating the connection between two hosts using the MPTCP, it used the same three-way handshake used as traditional TCP. It will send SYN, SYN/ACK and ACK messages to and from the hosts and establish the connection. However, to negotiate the MPTCP communication, it uses the set of MPTCP options. The MPTCP options will reside inside the "*Options*" field of the TCP header sections and exchanged in the initial three-way handshake. In the first

sub-flow, MPTCP will send the MP_CAPABLE option within the SYN message to the host which it intends to communicate with (Figure 2). When this message is received at the recipient and if the recipient is configured with MPTCP, it will reply to the message with SYN/ACK message including the MP_CAPABLE. If not it will reply to the SYN without including the MP_CAPABLE option. If the reply contains the MP_CAPABLE option, the initiator will recognize that the recipient is also compatible with MPTCP. When the reply contains the traditional TCP SYN/ACK without MP_CAPABLE option, it implies that the recipient is not configured with MPTCP. With the MP_CAPABLE option, MPTCP will negotiate a set of keys which will be used to authenticate the next subflows it will generate.

After establishing the first subflow, MPTCP will use other network interfaces to create the consecutive subflows. Again the TCP three-way handshake will happen, but with the MP_JOIN option included in the TCP "*Options*" field. MP_JOIN will use the keys negotiated in the first subflow using MP_CAPABLE to authenticate the newly created subflow with the destination host. If the host agrees to create the next subflow it will reply to the SYN message with the MP_JOIN options with the key materials to confirm the connection. This mechanism is used to join any number of subflows to the MPTCP connection.

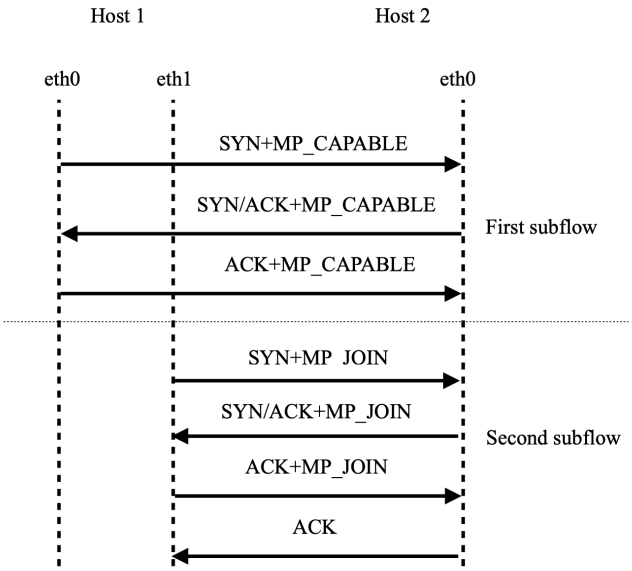

Fig. 2 MPTCP three-way handshake which uses MP_CAPABLE option in the first subflow and MP_JOIN option in the next subflow.

MPTCP uses two methods in scheduling packets at the initial stage of its connectivity 1. all the network packets are duplicated among the available subflows to improve redundancy. 2. segment and schedule the packets to the subflows in the round-robin method to improve overall performance and bandwidth [20]. However, further studies present more advanced schedulers and congestion controlling mechanisms for MPTCP [21]. Though the schedulers and congestion control mechanisms currently available in GNU/Linux systems, MPTCP schedules the packets to the subflows without any user or application intervention and is purely based on the information available through regular congestion control implementations at transport layer with respect to each subflow.

On the other hand, Frömmgen et al. and others, have shown the limitations of current schedulers of MPTCP and they suggested having an application aware MPTCP scheduler, which ultimately gives MPTCP more benefits than the throughput optimization [16]. The study further mentions that the applications might have different preferences or requirements, not only the throughput. For such cases, they have introduced a programming model which can be used to create and deploy, application and preference aware schedulers for MPTCP. However, in our study, we argued that, as the primary stakeholder of the system, the ability to make the decision about the path selection and scheduling has to be with the user of the particular device, which we consider as the user policy, rather than the application.

Most of the users who use devices are not technically capable of creating their own schedules or configuring the pluggable schedulers by using sysctl commands. Therefore when it comes to the end user, it has to be more user friendly for the end users of the devices to select or configure their user policy. For example, an user interface with a toggle to priorities the subflow which is connected to a particular service provider that may provide higher bandwidth or less charges.

Therefore, we believe that it is prudent to base the decision on prioritizing or selecting a communication path via a particular interface on a user policy and the information that the transport layer can discover/learn. As mentioned earlier, the user policy may be a composition of different factors such as risk, opportunistic monetary cost, throughput, and some other affinity towards a particular path. Further, these factors may be computed or taken directly as an input from the user because, in any information system, the user plays a vital role as a stakeholder. However, existing TCP/IP specification does not have provisions to capture such information or propagate such information to the Transport layer. Further, making such enhancement to the protocol without compromising existing systems and applications would be a challenging task.

In one of the early works, Wijesekera et al. suggested COMONet as a user transparent way of switching a conventional call between a costly mobile provider or a free community-driven ad-hoc network [3]. The switching between connections occurs at the application layer. However, MPTCP provides a less costly way to switch at the kernel level.

Use of heterogeneous paths (subflows) with MPTCP made researchers explore finding the optimal path based on its network performance. Thus, ample research has been conducted on finding the optimal path based on the path attributes [12][14]. Further, a number of researches have been carried out to discover application/context aware path selection [11] [13]. Despite these findings, we believe that the user's preference is also a vital factor in deciding the path, as the user possesses additional knowledge such as cost-effectiveness, reliability, etc. which is unavailable and not considered for current MPTCP path selection.

B. MPTCP Scheduler

Selection of a subflow and segmentation/integration of data into/from multiple subflows is done by MPTCP scheduler (Figure 3). The availability of the subflows for selection is provided by the path management component of the MPTCP. The MPTCP scheduler considers a number of factors such as, the TCP subflows' state (active or not), the congestion window of the subflow, the round trip time (RTT) estimation, etc. In making the subflow selection decision. A combination of these factors is considered as a subflow scheduling policy. Currently, there are four such scheduler policies in GNU/Linux MPTCP implementation, namely; MPTCP BLEST, MPTCP Round-Robin, MPTCP Redundant, and Default MPTCP Scheduler [4]. However, the scheduling policy in GNU/Linux can easily be changed to cater different requirements [15].

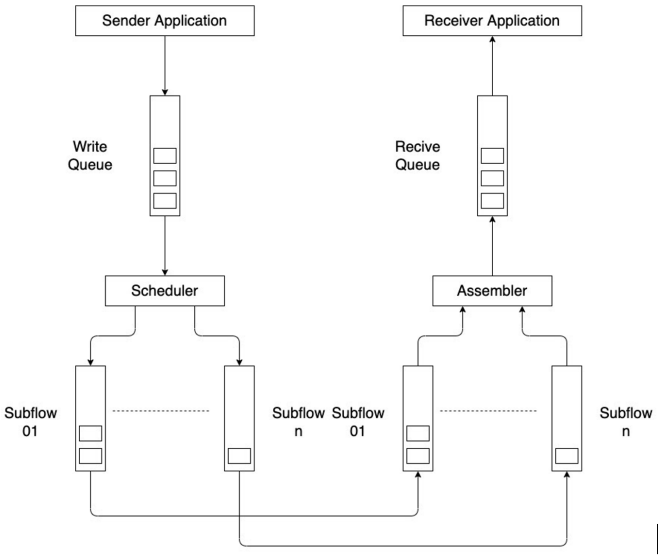

Fig. 3 MPTCP Scheduler

According to the study of Frömmgen et al., there may be different bandwidth requirements for the applications and some of the applications are very sensitive to the latency. In such cases the packets have to be scheduled based on the requirement of the application. Therefore they have highlighted the need of intelligent schedulers to MPTCP which can cater the need of these applications. With the implementation, they have provided a Python library for the user space application which is going to cover all the complexities of network sockets and scheduling. Therefore, the application developer can load their scheduling specification for the application by using the programming model provided. The implementation has several layers of functionalities which are used to optimize the compilation of the scheduler and interpretation. Finally they have a runtime environment in the GNU/Linux kernel to execute the application defined scheduler.

The real argument to make at this point is, what is the most suitable position to make the subflow selection decision. Is it at the MPTCP level, application level or at the user level. In the original MPTCP the subflow will be selected based on factors like TCP subflows' state and the congestion window of the subflow as we mentioned earlier. MPTCP has most of the information which is necessary to make the scheduling decision. In the proposed mechanism of Frömmgen et al, the decision making authority for scheduling has been given to the application. The third option is the user level. In this case the user can consider external factors when making the subflow decision, such as the cost for each connection on subflows, privacy issues, and the requirement of the user. Therefore when considering these three different levels, there are pros and cons to discuss.

When making the subflow selection and scheduling in the MPTCP level, the user and the application is totally blind about the path selection and the scheduling process. Users and the application does not even have any clue whether it is using MPTCP to communicate or whether it is using traditional TCP. All the hard work related to the scheduling is handled by the MPTCP. This is an advantage for the user as well as for the application. Such that the application nor the user does not need to worry about selecting the best path for scheduling the packets. But the problem is, does the user or application need to interfere with the scheduling process. In some cases, the user might have some requirements to allow or restrict sending packets via specific connections due to monetary cost or even privacy factors.

In this study, we incorporate a new policy to get the user's preference to assign different priorities for MPTCP subflows.

C. MPTCP recovery from packet losses

In TCP, there are three mechanisms available to recover from packet loss; Retransmission Timeouts (RTO), Fast Recovery (FR), and TCP Loss Probe (TLP) [5]. MPTCP also inherits these mechanisms. If the MPTCP handles the recovery, the lost packets are retransmitted through an available subflow according to the scheduler's policy at that particular time. If the TCP handled the recovery, packets would be retransmitted through the same subflow as before. GNU/Linux's implementation of MPTCP uses heuristics to decide whether the retransmission is FR based or RTO based. In FR, the segments use the same subflow as previous communication, wherein RTO, the scheduler, re-evaluates the packet transmission and would use a different subflow for recovery [5].

An Ad-hoc network is composed of nodes that are mobile and sparsely connected to create a communication network. Therefore, ad-hoc networks are intrinsically dynamic in their routing. As a result, they have significant delays in packet transmission which could result in frequent retransmissions [2]. Therefore, employing an Ad-hoc network as a subflow of MPTCP would make MPTCP select the more stable alternative subflows such as the flow over a stable fixed link [4]. GNU/Linux kernel supports both Ad-hoc networking and MPTCP. Therefore, employing an Ad-hoc network as an MPTCP subflow on the GNU/Linux environment has given us a viable experimentation set up to study how user preferences can be passed to the TCP layer to push MPTCP to select paths that it would otherwise abandon. User preference is passed to the kernel as a soft directive - a hint. This testbed, rather than a simulation, allows us to explore the issue in a realistic environment.

To this end, we show the viability of passing user hints to the kernel to focus on a subflow over the other alternatives. We contribute the following,

1. We developed a method to influence the path selection and scheduling of MPTCP with the preference/requirement of the end user.
2. We show that the proposed modifications are practical and with negligible overhead.
3. We show that the proposed work can pave the way to much interesting security research

III. Design

As mentioned before, the goal of this research is to prioritize the flow of network packets through the Ad-hoc connection and get the user preference for that process. Figure 4 illustrates a high level view of the experimental setup. One connection was created through ISP whereas the other connection established through an Ad-hoc network. Application layer made unaware of the MPTCP to make existing applications reusable despite the changes at lower layers of the network stack. User preference is set to Ad-hoc network in order to prioritize traffic through Ad-hoc network. A new scheduling mechanism is introduced to honor user preference in selecting MPTCP subflows. In order to achieve these goals, we have identified three main tasks. They are,

- Passing user hints to the kernel.
- Prioritizing Ad-hoc subflow.
- Implementing alternative loss recovery.

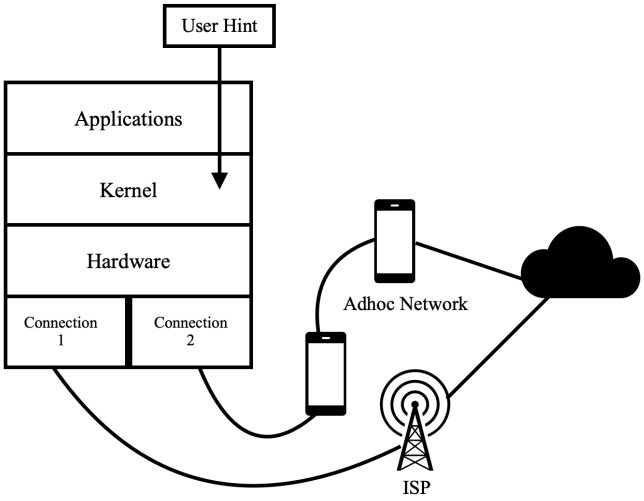

Fig. 4 Design

A. Passing user hints to the kernel

There are several mechanisms to pass user hints from user space to kernel space, such as using Netlink sockets [7] and using extra fields like sin_zero in socket [8]. However, it is prudent to pass the hint from user to the kernel without modifying user applications. GNU/Linux has an abstraction layer providing an interface to the kernel data structures via a pseudo file system called "\proc". Therefore, in order to convey the hint from user space to the kernel space without modifying an application, we use the proc file system [9]. We pass the Internet Protocol (IP) address of the subflow we prefer to prioritize using the proc filesystem, thereby the MPTCP scheduler can act accordingly.

Since, we use an Ad-hoc network as a possible path of communication, we keep the availability of an ad-hoc network in a new variable (is_adhoc_avail) which is stored in the MPTCP control buffer (mptcp_cb). In MPTCP architecture, MPTCP control buffer is visible to all subflows. We use MPTCP controller to set is_adhoc_avail variable upon detecting an ad-hoc network. The MPTCP scheduler is modified to refer to the variable to check the availability of ad-hoc networks when selecting subflows (see Fig. 5).

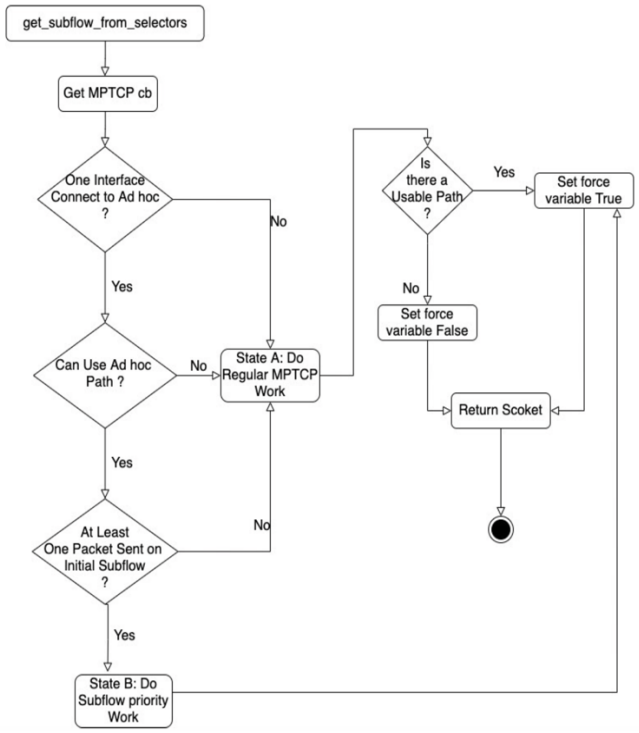

Fig. 5 Modified MPTCP Scheduler

B. Prioritizing Ad-hoc subflow

The default scheduler policy is to use the subflow with minimum latency and lowest number of scheduled packets. Therefore, given the nature of ad-hoc networks, there is very less probability of getting an ad-hoc network scheduled through the MPTCP default scheduler. Therefore, we propose a derived version of MPTCP scheduler which we have illustrated in Figure 3, henceforth referred to as the scheduler. The scheduler initially checks if there are any ad-hoc networks connected to one of the network interfaces by referring to the `is_adhoc_avail` variable set by MPTCP controller.

Even though the ad-hoc network is available, it may be in an error state. A subflow can become unusable for various reasons such as higher RTT, higher error rate, or complete loss of the channel. Such subflows are put into an error state using the `adhoc_priority` flag in the MPTCP control buffer to make sure that the scheduler does not use the particular subflow. We then keep track of the erroneous ad-hoc network subflow using a socket flag corresponding to the particular subflow [6]. The erroneous subflows related to ad-hoc networks are separately probed at regular intervals

using a retransmission packet and check if those can be reactivated. An acknowledgement received in response to a retransmission triggers reactivation reversing the value of `adhoc_priority` flag to make the subflow active. Thus, the subflow needs to be checked for usability by referring to the value of `adhoc_priority` in the MPTCP control buffer before scheduling. In this implementation, as it is shown in equation 1, we used the deactivation threshold, $\boldsymbol{\delta}$, to deactivate the prioritized sub-flow.

$$\boldsymbol{\delta} = 2 * min(SRTT) \quad (1)$$

SRTT is the Smoothed RTT and the min(SRTT) is the minimum of the SRTTs across all the subflows. Prioritized subflow is deactivated if its SRTT is larger than the deactivation threshold. Note that this threshold is an arbitrary value used for our experiments and it can be set by the user.

In design, TCP retransmission intervals get incremented exponentially over time thus making a particular path reactivation time to increase exponentially. Therefore, a connection which has regular interruptions, such as Ad-hoc networks, could not be effectively used with existing TCP design. Hence, we propose a redesigned TCP retransmission strategy incorporating user preference as a hint for MPTCP to reactivate a subflow thus making a sub-flow attached to an ad-hoc network considered for scheduling.

Considering the deficient reliability of ad-hoc networks, TCP connection initiation is not scheduled through the ad-hoc network. Therefore, subflow corresponding to the ad-hoc interface has only been scheduled after confirming that the initial communication has happened through one of the interfaces other than ad-hoc networks.

A subflow complying the conditions stated gets scheduled by the scheduler with packets to be sent to an MPTCP aware destination.

C. Implementing alternative loss recovery

Ad-hoc networks are volatile. Therefore, the standard loss recovery mechanism used by MPTCP has to be altered accordingly to the behaviour of Ad-hoc networks. In the standard MPTCP loss recovery, for each unsuccessful retransmission, the retransmission timeout doubles. But in our implementation, first it checks whether the retransmission is happening through an Ad-hoc socket. If it is an Ad-hoc socket, then the retransmission time doubles and keeps it constant for a defined amount of retransmission. The number is taken as part of the hint set by the user reflecting a weightage to use ad-hoc network. For our evaluations, we have taken the maximum weightage. After that it will switch back into doubling retransmission timeout as in the standard MPTCP. With this we are checking the availability or recovery of Ad-hoc connections more frequently compared to standard MPTCP. The main objective of this more persistent re-trying is to make sure that the priority is given to the ad-hoc network. We, however, believe that this will result in longer waiting but we hypothesize that there will be cases that longer waiting can be a worthy compromise over switching to unpreferable subflow.

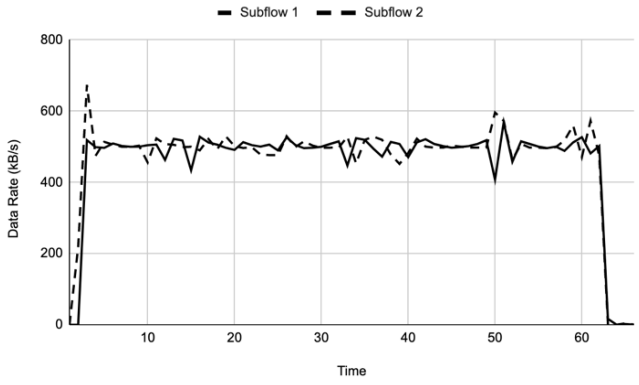

Fig. 6 Data rates on both the interfaces of Modified MPTCP with same latency

IV. Evaluation

As proposed in Section III, we modified the standard MPTCP in order to pass user hints to the TCP scheduler. To evaluate the correctness and the effectiveness of the modified protocol, we carried out several experiments in a virtual environment. Each host in the virtual environment was configured to have two WiFi network interfaces, such that one interface was configured with infrastructure mode, and the other was configured with Ad-hoc mode. Further, the communication links were restricted to have a maximum bandwidth of 1MB/s by using VMware [10] to emulate physical networks' behavior. The rest of this section is organized to present all the experiments.

A. Overhead of the new MPTCP kernel stack

The objective of our first experiment was to investigate whether the standard MPTCP is unduly affected by the changes introduced into the MPTCP kernel stack. Thereby, the behavior of data flow was examined using the data rate as a metric, and the modified MPTCP was compared against the standard MPTCP. To eliminate the bias of latency in one interface over the other, both the interfaces were configured with the same latency.

As we see in Fig. 6 and Fig. 7, both the standard and the modified MPTCP have reached stable data rates up to 500 kB/s. Achieving the same stable level of data rate shows that the presented modifications do not incur significant overheads.

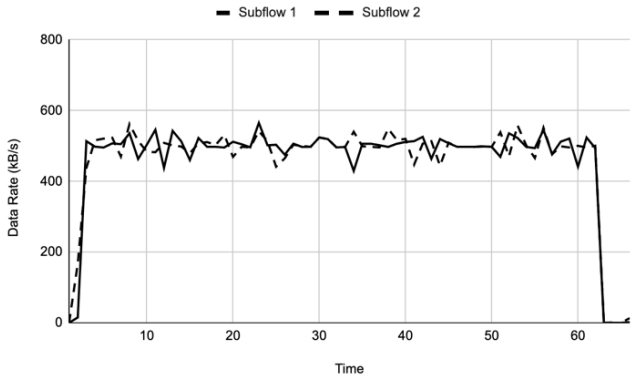

Fig. 7 Data rates on both the interfaces of Standard MPTCP with same latency

To explore the rate of data flow when using the modified MPTCP with the user controlled subflow selection, we first prioritized the network interface that was configured with Ad-hoc mode by providing a user hint. Then we initiated the connection using the prioritized interface and observed the data rate with the time. Thereafter, we explored the rate of data flow separately by initiating another connection using the second interface which was non-prioritized. Fig. 8 and Fig. 9 show the graphs for those two experiments respectively. With these two experiments, we discovered that the scheduler has scheduled the segments only to the prioritized interface and the data rate was dropped to a half of the full capacity. The sole reason for the bandwidth reduction is the user control over the subflow selection where the data was sent only through the prioritized Ad-hoc interface.

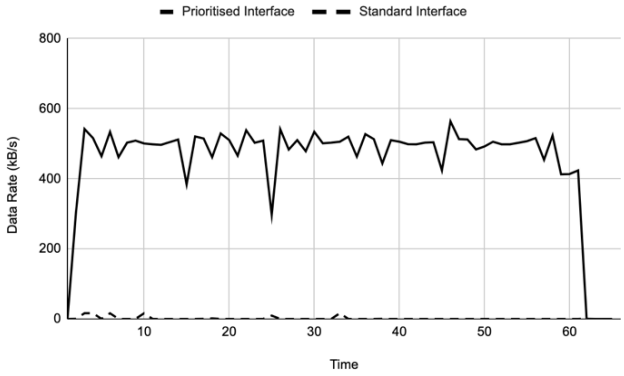

Fig. 8 Data rates when creating the initial connection through prioritized interface

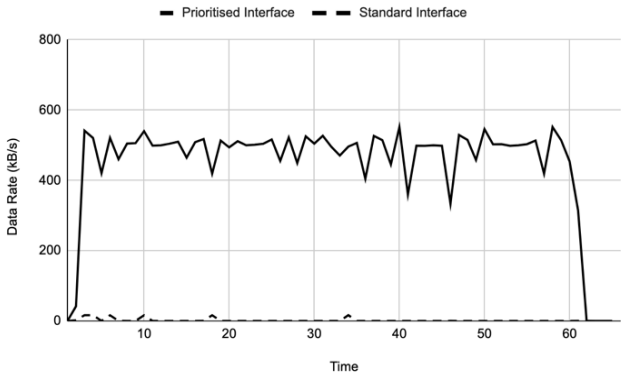

Fig. 9 Data rates when creating the initial connection through non prioritized interface

B. Path switching on modified MPTCP

Since the modified MPTCP permits prioritizing subflows, we have to test the path switching ability when the prioritized path fails. Typically, the Ad-hoc connection is unstable compared to the infrastructure connection in a host with an infrastructure based network connectivity and an Ad-hoc connectivity. Preferably, in this experimental setup, even though the ad-hoc connection is prioritized, the host has to switch to the infrastructure based connection when the ad-hoc connection fails. In order to test whether it switches back and forth from prioritized subflow to non-prioritized subflow, we have conducted an experiment.

To simulate breaking the prioritized subflow, we artificially block the prioritized subflow time-to-time. The latency of the connections was set to 40ms using tc command and the maximum bandwidth was set as 512kB/s. The server drops all packets from the Ad-hoc connection, which is the prioritized connections, for a five seconds period, with a gap of ten seconds. During those periods, we observed that the path is switched to the connection through the infrastructure based network and once the Ad-hoc connection is active, the connection switches back to the prioritized ad-hoc connectivity. Fig. 10 clearly shows that data rate via non-prioritized standard interface is minimal while the prioritized ad-hoc connection is having higher data rate and vice versa. It implies that when the prioritized connection is active the data flows through it, and when the prioritized connection fails data starts to flow through the non-prioritized infrastructure based connection.

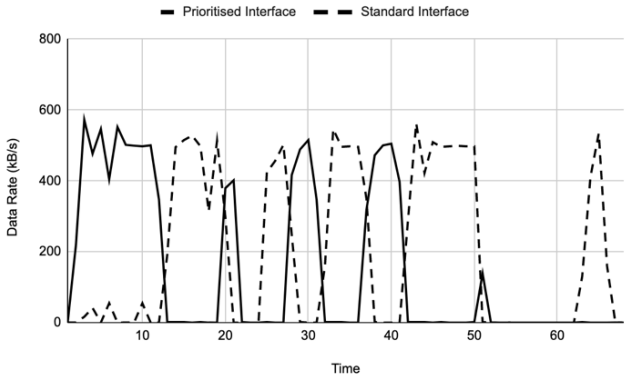

Fig. 10 Path switching between prioritized and non-prioritized interfaces.

C. Use of prioritized subflow with high latency

One of our main objectives of this study is to make MPTCP tolerate high latency in Ad-hoc networks based on user preference. In this experiment, the prioritized sub-flow, which is the Ad-hoc connection, has configured with high latency (< deactivation threshold) compared to the non prioritized subflow. Fig. 11 shows the prioritized sub-flow with high latency is still used to transfer the data.

For the evaluation 40ms latency was used in non prioritized subflow and 60ms latency was used in prioritized subflow.

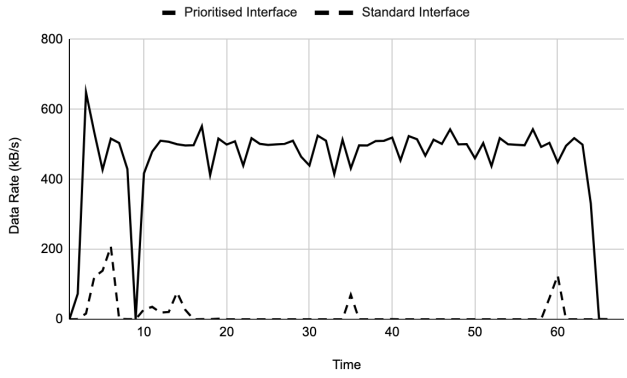

Fig. 11 With high latency (< deactivation threshold)

D. Switching the face of higher latency

In this experiment, 40ms was used as the latency of the non prioritized subflow and 200ms was used as the latency of the prioritized subflow. As shown in Fig. 12, when the latency difference is high, scheduler falls back into default behaviour and more data has been transferred through the interface with lower latency.

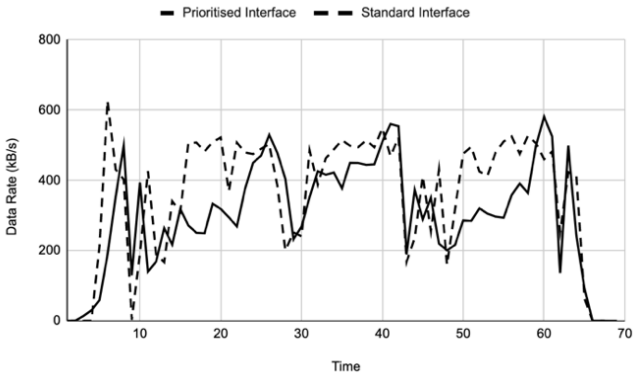

Fig. 12 Throughput between prioritized and non-prioritized interfaces when the latency difference is high

E. Prioritized subflow recovery speed evaluation

This experiment was conducted to observe the path recovery speed of modified MPTCP, compared with the standard MPTCP. It has used 40ms as the latency on all the subflows. In the standard MPTCP connection, one sub-flow was dropped and measured the time taken to recover the particular path. The same method was used in the Modified MPTCP and measured the time taken to recover the dropped subflow. It took around 90ms to recover the path in Modified MPTCP and around 140ms in Standard MPTCP. Therefore, the Modified MPTCP has shown a significant improvement than the standard MPTCP by recovering the path around 50ms earlier.

V. Discussion

The main objective of this study is to explore the viability of influencing the MPTCP scheduler in kernel on the selection of a subflow for MPTCP. We show that,

a) it is practically feasible to incorporate user preference when selecting subflows and scheduling paths,

b) the proposed modifications incur minimal overhead, making the proposed solution a practical alternative.

A. Conditional priority

The paper presents a use-case where a user can instruct the kernel to use the ad-hoc network over infrastructure mode. The hint creates a priority to a given subflow giving other alternatives a chance, i.e., the kernel will try to use the preferred subflow as much as possible while switching to another subflow as soon as the preferred subflow is no longer viable.

In a future setup, one could argue that users could potentially submit conditions to set up the priorities. In such a setup, whichever flow that fulfills the condition would be used to send the packets. We believe the proposed research will pave the way for future more complicated priority setup or directions.

B. Application-level decision making

Technically speaking, priority set up can be achieved at any layer of the TCP/IP stack. Lower in the stack has less flexibility for users but provides abstraction for users who want less configuration headache and higher up at the stack has more user control but could require additional user involvement which could be a distraction for some. However, it is important to discuss the pros and cons of each approach.

For application to participate in the subflow selection, the application should have access to a number of parameters, such as the state of the interface (active or not), the congestion window of the subflow, the RTT estimation, etc. which is not readily available at the application layer. Further, making such information available for the application layer would have widened the threat surface as well by making the application literally in control of the transport layer of the device. Recent events that surfaced applications rerouting traffic and moderating the content for surveillance purposes, it is better not to give applications such control [22].

C. User space decision making

With the design we proposed in the study, the MPTCP has the relevant and standard information to select the most suitable subflow for a particular situation to schedule the communication User hint and priorities (policy) feeded to the scheduler would act as additional information to make more effective decisions in catering the specific user needs i.e. the system will get qualitative and subjective insights such as opportunistic costs, privacy concerns, etc. in a quantitative form to make more accurate and effective decisions which otherwise are not available for the MPTCP scheduler / kernel.

The user space decision making or priority setting can be one level up than the proposed research. One could also design in a manner that users can specify priorities per application basis.

To demonstrate the effectiveness of the mechanism we used, we test our system between infrastructure mode and ad-hoc mode WiFi connections. If the vanilla MPTCP is allowed to take path decisions it prefers the infrastructure networks since it is less volatile than the Ad-hoc network. However, from the point of view of the user, the Ad-hoc network is less costly than the ISP based infrastructure network and the Ad-hoc network should be used whenever possible. Hence, we showed that we can influence the kernel to use the ad-hoc connection over the infrastructure mode.

D. Privacy and security concerns

MPTCP provides useful functionality from a congestion and flow control perspective. However, we believe that this functionality has exciting security and privacy implications that can benefit users safeguarding them from data-hungry malicious actors. In the era of tight cyber-surveillance and control of Internet Access, MPTCP can be a solution where

users can instruct the kernel for packet-level finer-grained data routing over preferred network connections.

Users with restrictions over Internet Access can instruct the kernel to send certain portions of network communication over the subflow with less scrutiny and control. Such fine-grained routing can be completely transparent to the user-level application, but more work needs to be done on session management at the kernel.The subflow management can also be a privacy win for users. Different subflows avoid ISP or state-level actors from full visibility into the communication at a given moment. That will reduce the possibility of network traffic reconstruction or data leakage from a given entity. Lack of full visibility will also let the user have more control over who can see which data in their network traffic.

One interesting future avenue would be to understand merging with Tor and MPTCP [23, 24]. Prior work has already looked into this but we believe it is interesting to understand how different decision making layers affect the consumer privacy and security expectations while merging with Tor.

These exciting avenues are possible only if the user can pass on their preferences and priorities to the kernel and transparently to the application. Thus, we believe this will open up exciting research, along with security and privacy. However, the current widespread TCP adoption is due to its simplicity and stability from the design of TCP to its implementations. While MPTCP provides much-needed functionality for security, cost reduction, etc., this could eventually challenge the core simplicity. This reason might further explain the conservative adoption of MPTCP in the wild. While current complicated user needs will call for more TCP functionality, we should be cautious not to harm the robust core of TCP.

VI. Conclusion

MPTCP has introduced dynamism to the traditional TCP by permitting to establish multiple paths through different network interfaces in a host. MPTCP Scheduler is responsible for segmentation/integration of data into/from multiple subflows and making the decision on selecting a subflow. However, we believe that the MPTCP Scheduler does not possess or cannot discover all the metrics required to make such a decision independently. In addition to that, though MPTCP increases the reliability and the throughput of the connection by providing redundant paths, the user has no control over path selection for the data flow.

Considering all these limitations, in this study, we proposed an approach to take user preferences into account when selecting subflows in MPTCP. We modified the MPTCP kernel stack and carried out several experiments to investigate the viability of this mechanism. The results of the experiments exemplified that our proposed mechanism can take user preferences into account when selecting MPTCP subflows. Also, we demonstrated that it is possible to pass user preferences as a hint to the kernel without changing the applications. A hint is not a hard rule, rather a soft directive to the MPTCP Scheduler to follow. In such a context, applications are unaware of the presence of MPTCP or the use of user-supplied hints. Further, experiments showed that our changes to the MPTCP Scheduler did not introduce extra overhead to the regular MPTCP operation. Finally, we used a test environment with a handicapped (in terms of latency and stability), but user-preferred, subflow to demonstrate that it is possible to cater to user preferences even in such an extreme environment.

References

[1] Z. Khan, H. Ahmadi, E. Hossain, M. Coupechoux, L. A. Dasilva, and J. J. Lehtomäki, “Carrier aggregation/channel bonding in next generation cellular networks: methods and challenges,” IEEE Network, vol. 28, no. 6, pp. 34–40, 2014.

[2] "RFC 793 - Transmission Control Protocol", Tools.ietf.org, 2020. [Online]. Available: https://tools.ietf.org/html/rfc793. [Accessed: 19- Jan- 2020].

[3] P. Wijesekera and C. Keppitiyagama, "COMONet: Community Mobile Network", arXiv preprint arXiv:2009.05966. 2020.

[4] “RFC 6824 - tcp extensions for multipath operation with multiple addresses.” https://tools.ietf.org/html/rfc6824. [Accessed on 03/20/2019].

[5] M. Handley, C. Raiciu, A. Ford, J. Iyengar, and S. Barre, “[5]"RFC 6182 - Architectural Guidelines for Multipath TCP Development", Tools.ietf.org, 2020. [Online]. Available: https://tools.ietf.org/html/rfc6182. [Accessed: 19- Feb- 2020].

[6] M. Lima, N. Fonseca, and J. de Rezende, “On the performance of tcp loss recovery mechanisms,” pp. 1812 – 1816 vol.3, 06 2003.

[7] P. Neira-Ayuso, R. Gasca and L. Lefevre, "Communicating between the kernel and user-space in Linux using Netlink sockets", Software: Practice and Experience, p. n/a-n/a, 2010. Available: 10.1002/spe.981 [Accessed 19 February 2020

[8] T. Wijethilake, K. Gunawardana, C. Keppitiyagama and K. de Zoyza, "An Alternative Approach to Authenticate Subflows of Multipath Transmission Control Protocol using an Application Level Key", in *13th International Research Conference*, General Sir John Kotelawala Defence University, 2020.

[9] T. Bowden, B. Bauer, J. Nerin and S. Feng, "The /proc filesystem", [online] Available: https://www.kernel.org/doc/Documentation/filesystems/proc.txt [Accessed: 19- Feb- 2020]

[10] "VMware Inc., "Using VMware Workstation Pro", VMware, Inc., 2019. [Online]. Available: https://docs.vmware.com/en/VMware-Workstation-Pro/. [Accessed: 19- Feb- 2020].

[11] R. Withnell and C. Edwards, “Towards a Context Aware Multipath-TCP,” 2015 IEEE 40th Conference on Local Computer Networks (LCN), 2015.

[12] S. H. Baidya and R. Prakash, “Improving the performance of multipath TCP over heterogeneous paths using slow path adaptation,” 2014 IEEE International Conference on Communications (ICC), 2014.

[13] A. Elgabli and V. Aggarwal, “SmartStreamer: Preference-Aware Multipath Video Streaming Over MPTCP,” IEEE Transactions on Vehicular Technology, vol. 68, no. 7, pp. 6975–6984, 2019.

[14] Y.-S. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, “ECF: An MPTCP Path Scheduler to Manage Heterogeneous Paths,” Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies, 2017.

[15] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, “Experimental evaluation of multipath TCP schedulers,” Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop - CSWS 14, 2014.

[16] A. Frömmgen, A. Rizk, T. Erbshäußer, M. Weller, B. Koldehofe, A. Buchmann, and R. Steinmetz, “A programming model for application-defined multipath TCP scheduling,” Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference, 2017.]

[17] O. Bonaventure, "Observing Siri : the three-way handshake", multipath-tcp.org, 2014. [Online]. Available: http://blog.multipath-tcp.org/blog/html/2014/02/24/observing_siri.html. [Accessed: 22- Feb-2020].

[18] "The first Multipath TCP enabled smartphones — MPTCP", Blog.multipath-tcp.org, 2021. [Online]. Available: http://blog.multipath-tcp.org/blog/html/2018/12/10/the_first_multipath_tcp_enabled_smartphones.html. [Accessed: 08- Mar- 2020].

[19] "Apple Developer Documentation", Developer.apple.com, 2020. [Online]. Available: https://developer.apple.com/documentation/foundation/urlsessionconfiguration/improving_network_reliability_using_multipath_tcp. [Accessed: 08- Mar- 2020].

[20] "An Evaluation of Multi-Path Transmission Control Protocol (M/TCP) with Robust Acknowledgement Schemes", IEICE TRANSACTIONS on Communications, vol. 87-, no. 9, pp. pp.2699-2707, 2004. Available: https://search.ieice.org/bin/summary.php?id=e87-b_9_2699. [Accessed 10 January 2021].

[21] S. Barré, C. Paasch and O. Bonaventure, "MultiPath TCP: From Theory to Practice", in NETWORKING 2011 - 10th International IFIP TC 6 Networking Conference, Valencia, 2011.

[22] D. Harwell and E. Nakashima, "Federal prosecutors accuse Zoom executive of working with Chinese government to surveil users and suppress video calls", *Washington post*, 2020. [Online]. Available: https://www.washingtonpost.com/technology/2020/12/18/zoom-helped-china-surveillance/. [Accessed: 27- Dec- 2020].

[23] W. De la Cadena, D. Kaiser, A. Panchenko and T. Engel, "Out-of-the-box Multipath TCP as a Tor Transport Protocol: Performance and Privacy Implications," *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)*, Cambridge, MA, USA, 2020, pp. 1-6, doi: 10.1109/NCA51143.2020.9306702.

[24] "annymous/oniontinc", *GitHub*, 2020. [Online]. Available: https://github.com/annymous/oniontinc. [Accessed: 12- Dec- 2020].

[25] K. Liyanage, T. Wijethilake, K. Gunawardana, K. Thilakarathne, P. Wijesekera and C. Keppitiyagama, "Priority Based Subflow Selection in MPTCP: A Case Study", in *2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer)*, Colombo, Sri Lanka, 2020.