An ns-3 MPTCP Implementation

Kashif Nadeem and Tariq M. Jadoon(✉)

Electrical Engineering Department,
Syed Babar Ali School of Science and Engineering,
Lahore University of Management Sciences, Lahore, Pakistan
`kshfnadeem@gmail.com, jadoon@lums.edu.pk`

Abstract. Multipath TCP (MPTCP) achieves greater throughput by sending packets from a single byte stream across multiple interfaces and thus, potentially exploits multiple available network paths. This allows end hosts to aggregate bandwidth and network resources. Network simulators such as ns-3 [1] provide researchers with a convenient tool to evaluate protocols and architectures and their importance can not be overemphasized. There are currently 3 existing implementations of MPTCP in ns-3. We evaluate these implementations and find that they lack several key features and are therefore, inadequate for furthering research. We implement MPTCP in ns-3-dev (Developer's version) and introduce multiple path managers namely *default*, *ndiffports* and *fullmesh* creating an MPTCP patch for ns-3 [2]. The simulation results show improvements in throughput and Flow Completion Times (FCTs) in comparison with previous work. Our implementation [3] is compatible with the current version (ns-3.29).

Keywords: MPTCP · ns-3 · Computer networks · Simulator

1 Introduction

Applications such as Facebook, Google, etc., require low latency and place bounds on the response time of a query initiated by a user. Excessive delays in response time of queries impact revenue and user experience. Transmission Control Protocol (TCP) [4] fails to provide high throughput to large flows and complete latency sensitive flows within time bounds. As TCP only uses a single interface of an end host even if other interfaces available, it thus, under utilizes network resources. In recent years, a number of transport layer protocols have been proposed to improve throughput such as, DCTCP [5], D2TCP [6], PIAS [7], etc. However, these protocols also use a single interface and do not exploit multiple interfaces even if available.

Modern network devices such as computers, smart phones and tablets typically have more than one network interface and can thus, be multihomed. Smart phones have wi-fi and 3G/4G interfaces whilst, laptops have Ethernet and wi-fi interfaces. Previous studies have shown that simultaneously using multiple interfaces can achieve higher throughput and can complete large flows in a shorter time [8–11]. Multipath TCP uses the available interfaces of a network device to send application data to the destination through multiple network paths. MPTCP splits the application data stream amongst subflows whereby, each subflow follows a path based on a 5-tuple: source IP, destination IP, source port, destination port and layer-4 protocol. These sub-flows can be routed over different paths by using routing protocols such as Equal-Cost Multi-Path (ECMP) [12]. The subflows can follow the same network path or different paths depending upon the availability of paths between the two hosts. MPTCP aggregates the bandwidth of available links to the hosts by applying the concept of Resource Pooling i.e., the available resources or links appear as a single logical resource or link to the host [8]. Multipath TCP has been implemented in several operating systems such as Linux, Apple ios7, Mac and FreeBSD. Furthermore, MPTCP can be deployed in data centers [13] and better exploits data center topology, effectively utilizes available bandwidth and provides improved throughput.

The balance of the paper is organised as follows. In Sect. 2 we describe the architecture and design of MPTCP from RFCs. In Sect. 3, we discuss the available implementations of MPTCP in different ns-3 versions. Furthermore, we discuss compatibility issues and missing features of these implementations. In Sect. 4, we briefly discuss why there is a need to implement MPTCP in the latest ns-3 tree and provide an overview about changes in TCP classes and discuss our implementation procedure. In Sect. 5, we describe simulation experiments to compare our implementation with Coudron et al. [14]. We finish with conclusions and directions for future work.

2 MPTCP Details from RFCs

This section provides a brief overview of MPTCP architecture, design and compatibility issues.

2.1 MPTCP Architecture and Path Managers

RFC 6182 [15] provides an architectural overview of MPTCP and discusses compatibility challenges with the existing network stack and middle boxes. MPTCP explores possible paths by using all available interfaces (IPs) at source and destination hosts.

In Fig. 1, Host A and B have two network interfaces each with unique IP addresses. MPTCP can thus, exploit four possible paths: A1-B1, A1-B2, A2-B1 and A2-B2 and moreover, can establish subflows on each path. These paths are not necessarily disjoint therefore, subflows may traverse the same link or the same path. The number of available paths depends upon the underlying physical network between the end hosts. MPTCP conveys protocol specific information in the header through the TCP options field. Middle boxes such as Network Address Translators (NATs), Performance Enhancing Proxies (PEPs), Intrusion

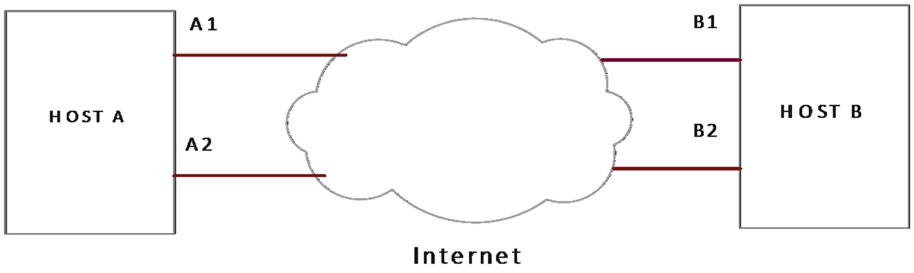

Fig. 1. Simple multipath scenario

Detection Systems (IDs) and Firewalls can potentially rewrite the TCP options or may drop a connection upon seeing unknown TCP options. RFC 3234 [16] mentions that a protocol designed without considering middle boxes can fail in the presence of middle boxes and suggests architectural design goals for new protocols and middle boxes. MPTCP design enforces the consideration of middle boxes so that subflows appear as legacy TCP flows to middle boxes. RFC 6897 [17] describes compatibility issues of MPTCP with existing applications and provides an application interface for MPTCP to work with legacy applications. This new design is based on the "Transport next-generation" (Tng) model [18]. The Tng model splits the transport layer into a "Semantic" layer which supports and implements functionality for the application layer and a "Flow+Endpoint" layer which manages the network-oriented part of the transport layer. The implementation of a network-oriented part in the transport layer enables the end hosts to interact with middle boxes as if they are working with legacy TCP. Figure 2 shows the Tng decomposed model of the internet and MPTCP protocol stack.

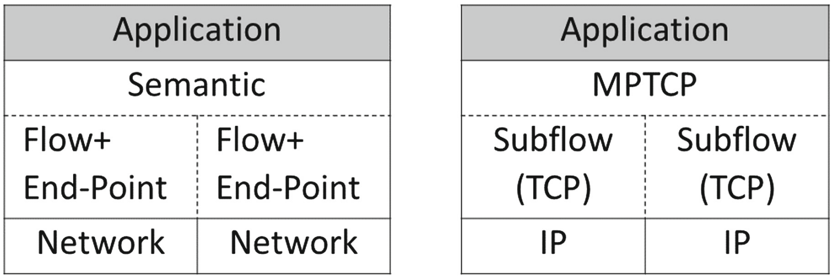

Fig. 2. Tng model (left) vs MPTCP model (right)

MPTCP implements several functions such as path management, packet scheduling, subflow interface and congestion control to create and manage subflows among multihomed hosts. Path Management is a core function of MPTCP which upon initial setup of the MPTCP connection creates subflows between the two end hosts. The Linux Kernel Implementation [19] provides four path managers. Users can select any one of them at compile time:

- **Default:** In default mode, the path management mechanism doesn't create new subflows. Hosts neither advertise IP addresses nor create new subflows however, passive creation of subflows is supported.
- **Fullmesh:** With this Path manager, multihomed hosts advertise addresses to peers and create a complete mesh of new subflows across all possible pairs of IP addresses. Considering the scenario depicted in Fig. 1; a fullmesh path manager will create four subflows between IP pairs: A1-B1, A1-B2, A2-B1 and A2-B2. Thus, the number of subflows is limited by number of the IP pairs.
- **ndiffPorts:** This path manager initiates subflows between the same IP pair using different source and destination ports. It can hence create any number of subflows between a pair of IP addresses such as A1-B1 shown in Fig. 1. The number of subflows created is controlled through a parameter.
- **Binder:** This path manager [20] uses Loose Source and Record Routing (LSRR) without modification of the end-user devices. Binder provides a list of available gateways to MPTCP subflows and ensures that subflows visit these gateways and explore all available paths in the network. The packets of subflows are distributed over the network using relays and proxies to explore available network paths.

2.2 MPTCP Design

RFC 6824 [21] provides a detailed description of MPTCP connection establishment, subflow initiation and MPTCP options used to carry information across the internet. Internet Assigned Numbers Authority (IANA) added a new TCP option for MPTCP with the symbolic name "Kind" with a 4-bit subtype field providing "MPTCP Option Subtypes". MPTCP option subtypes include `MP_CAPABLE`, `MP_JOIN`, `ADD_ADDR`, `MP_FAIL`, `DSS`, etc. Here we briefly discuss connection setup, subflow initiation and data sequence mapping.

- **MPTCP connection setup:** End hosts use the traditional 3-way TCP handshake mechanism SYN, SYN/ACK, ACK but each packet contains an `MP_CAPABLE` option as shown in Fig. 3. Moreover, the packets include a sender's key and a receiver's key that are used in future for the creation of new subflows. This MPTCP handshake ensures that the receiver and middle boxes are MPTCP capable. If any received packet doesn't contain the `MP_CAPABLE` option then it means that either the receiver or middle boxes are not MPTCP capable and the connection falls back to regular TCP.
- **Creating new subflows:** After the establishment of an MPTCP connection, hosts can create subflows when the first `DATA_ACK` is received through the Data Sequence Signaling (DSS) option. The connection initiation messages SYN, SYN/ACK and ACK include `MP_JOIN`. The sender sends a 32-bit token generated by SHA-1 from the receiver's key with the SYN packet to associate subflows with the MPTCP connection. Senders and receivers exchange

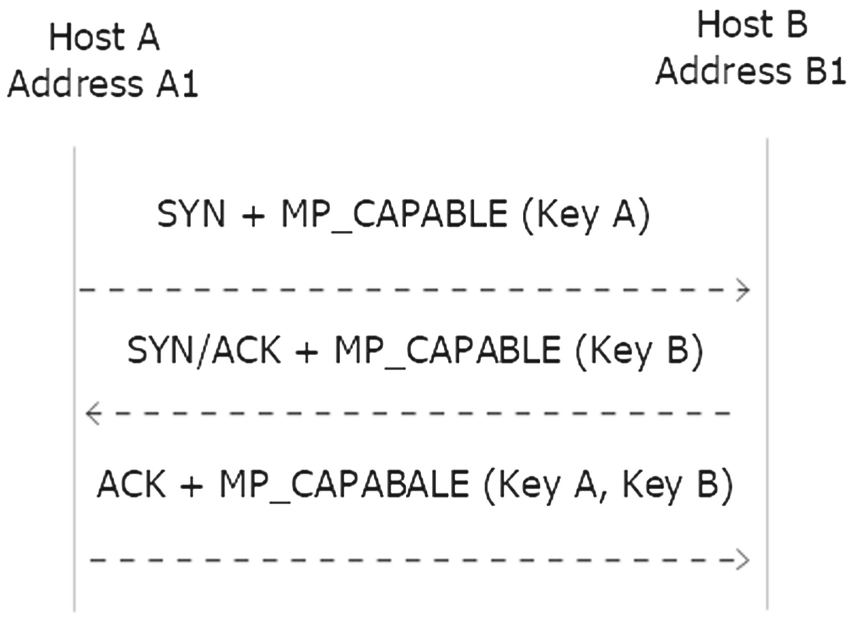
Host A
Address A1
Host B
Address B1
SYN + MP_CAPABLE (Key A)
SYN/ACK + MP_CAPABLE (Key B)
ACK + MP_CAPABALE (Key A, Key B)

Fig. 3. MPTCP 3-way handshake

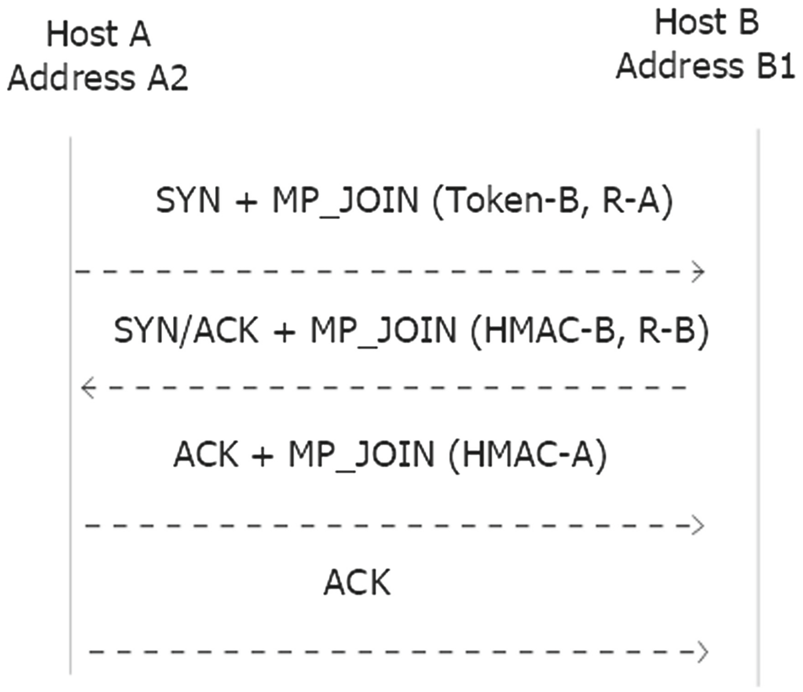

Fig. 4. Subflow initiation from multi addressed Host A

a nonce and Hashed Message Authentication Code (HMAC) for connection authentication. Any packet received without a `MP_JOIN` will result in falling back to traditional TCP. The subflow initiation process between host A with address A2 and host B with address B1 is shown in Fig. 4.

3 Existing MPTCP Implementations in ns-3 and Their Shortcomings

This section briefly describes the available ns-3 implementations of MPTCP and discusses their shortcomings.

3.1 ns-3 Implementations of MPTCP

At present, three implementations of MPTCP are available in ns-3. MPTCP was first implemented in ns-3 by Chihani et al. [22] in ns-3.6. The TCP stack was rewritten in ns-3.8 which makes this implementation incompatible with later ns-3 versions and is now obsolete. In this implementation, the sender and receiver do not exchange keys to associate new subflows with the MPTCP connection. They also didn't follow an authentication process for subflows through nonces and HMACs as described in [21].

The second implementation of MPTCP is in ns-3.19 by Kheirkhah et al. [23] and follows the Linux kernel implementation of MPTCP [24]. The authors create a MpTcpSocketBase class; which is a subclass of the TcpSocketBase class. Upon a successful MPTCP connection a MPTCP socket is created that provides an interface between the application and TCP flows. The MpTcpSocketBase object controls path management, scheduling, packet reordering and congestion control algorithms, etc. Another class MpTcpSubflow defines TCP subflows that communicate with the network layer. These two MPTCP implementations [22, 23] lack several features such as backward compatibility with the TCP stack and support a subset of MPTCP options as listed in Table 1 in [14]. The ns-3.19 implementation doesn't implement HMAC based authentication for subflows as shown in Fig. 4 and also lacks support for TCP timestamps, window scale options, congestion control algorithms such as Scalable, H-TCP, TCP Vegas, etc. Moreover, ns-3.19 lacks many new TCP features making it less appealing to the research community. In later ns-3 versions, TCP classes have changed substantially which makes this implementation incompatible with newer ns-3 versions.

The third implementation of MPTCP is in ns-3.23 by Coudron et al. [14]. This implementation covers several deficiencies of the previous implementations as shown in Table 1 in [14]. The MPTCP connection starts with a TCP socket and the client sends a SYN + MP_CAPABLE option with its key. The server receives SYN packet and then upgrades to a MPTCP socket if it is MPTCP capable and replies with a SYN/ACK + MP_CAPABLE option along with its server key. The client upon receiving SYN/ACK upgrades to a MPTCP socket and replies with an ACK completing the 3-way MPTCP handshake. This MPTCP socket provides an interface between the application and TCP subflows for packet scheduling, reordering and retransmission, etc. Round robin and fastest RTT schedulers are implemented and divide the application byte stream into segments for transmission on established subflows.

3.2 Problems with the ns-3.23 [14] Implementation

Coudron et al. [14] MPTCP implementation faces two kinds of problems:

Missing and Incomplete Features. Although, this implementation covers several deficiencies of [22,23] as shown in Table 1 in [14] yet it still lacks several basic MPTCP components shown in Table 1. This implementation doesn't implement *path management* discussed in Sect. 2.1 which is a core MPTCP function. Without path managers MPTCP behaves just like single-path TCP and doesn't create subflows even if hosts are multi-homed and multi-addressed. Coudron et al. [14] implementation creates a master subflow to transmit an application byte stream but subsequently does not create further subflows. Although there are three classes for MPTCP congestion control i.e., `MpTcpCongestionCoupled, MpTcpCCOlia and MpTcpCCUncoupled`, they are incomplete and not functional. Furthermore, this implementation also lacks the MP_FAIL option, infinite mapping, checksums and MP_PRIO.

Compatibility Issues. ns-3 is a continuously evolving project wherein each new version potentially introduces new models and classes as well as modifies existing models and classes. ns-3.25 refactored TCP removing, modifying and appending some TCP classes, functions and variables. Furthermore, ns-3.25 introduced new congestion control classes as well as Active Queue Management (AQM), policing and packet filtering. Similarly, ns-3.26 introduced new congestion *control classes* such as TCP Vegas, Veno, H-TCP and Illinois, etc. for legacy TCP. Several new queueing models such as Linux-like `pfifo_fast, FQ_CoDel, Byte Queue Limits, Adaptive RED` have been added in the Traffic Control Module. Moreover, ns-3.26 implements Fast retransmit and Fast recovery as described in RFC 5681 [25]. ns-3.27 incorporates SACK and LEDBAT in the TCP model amongst other models. ns-3.28 added IPV6 support for LTE, TCP

Table 1. A comparison of features in various MPTCP implementations

Features	Kheirkha et al.	Coudron et al.	ns-3-dev MPTCP
Path managers	Default, fullmesh, ndiffPorts	None	Default, ndiffPorts, fullmesh
Subflow creation	Yes	No	Yes
Congestion control	Yes	No	No
Compatibility[a]	No	No	Yes

[a]Compatibility with current ns-3 stack

pacing, FIFO and TBF queue, etc. These changes in TCP classes make Coudron et al. [14] incompatible with the current ns-3 stack. Thus, older ns-3 versions lack many networking components and features making them less attractive for present research work.

4 Our MPTCP Implementation in ns-3-dev

This section discusses our MPTCP implementation. We briefly discuss changes in TCP classes and describe the upstreaming process.

4.1 Why MPTCP into ns-3-dev?

A common problem with all three previous implementations is that they are incompatible with the current ns-3 stack. Because of this limitation, these implementations are unable to use the latest ns-3 models. Our ns-3-dev implementation is current with the ns-3 tree (ns-3-dev) and is compatible with the latest version ns-3.29 as well as potentially compatible with the future ns-3 versions.

4.2 Changes in TCP Stack of ns-3

As described earlier, the ns-3 TCP stack is rapidly evolving. The ns-3 TCP stack was *rewritten* in ns-3.8 and later *refactored* in ns-3.25 resulting in major changes and modifications to several functions and variables in TCP classes. There are many noticeable enhancements to TCP classes. Several functions have been modified and although they have the same name as in previous versions of ns-3 they contain different parameters.The TcpSocketBase class has had major changes over the evolution of ns-3. The class no longer handles congestion control and new congestion control classes that are subclasses of the *TcpCongestionOps* class have been introduced. The TcpSocketState class keeps track of the congestion state of a connection. Congestion control related variables such as m_highTxMark, m_nextTxSequence, etc., have been moved from the TcpSocketBase class into the TcpSocketState class. Furthermore, new variables for congestion control such as m_bytesInFlight, m_pacing, m_rcvTimestampValue, m_lastRtt, etc., have been introduced in the TcpSocketState class. Several variables have been added to the TcpSocketBase class such as m_highTxAck, m_bytesInFlight, m_dataRetrCount, m_dataRetries, m_sndScaleFactor, m_rcvScaleFactor, etc. Furthermore, some functions have been removed from TcpSocketBase such as FirstUnackedSeq(), GetRttEstimator(), SendEmptyPacket (TcpHeader& header), UpdateTxBuffer(), etc. Whereas, several new functions have been added such as UpdateRttHistory(), UpdateCwndInfl, LimitedTransmit (), FastRetransmit (), etc. Selective Acknowledgments (SACK) and Explicit Congestion Notification (ECN) have also been incorporated into the TCP stack in ns-3. The TcpTxBuffer class has been updated in accordance with RFCs and the Linux operating system to implement mechanisms for SACK and management of bytes in flight.

4.3 Upstreaming Process

We used Coudron et al. implementation [14] as base code and upstreamed it into the current ns-3-dev by making the necessary modifications to relevant classes. During the process, we have tried to minimize dependencies of MPTCP classes i.e., MpTcpSocketBase, MpTcpSubflow, TcpOptionMpTcp, etc. on TCP classes. We have also made appropriate changes to the TcpL4Protocol, TcpRxBuffer and TcpSocketBase classes to make existing MPTCP code compatible with ns-3-dev. While compiling MPTCP code we faced two types of errors: *compiler related errors* and *incompatibility* related errors with the TCP stack.

4.4 Path Management Implementation

Path management is a critical functionality of MPTCP and was missing in the previous ns-3.23 implementation. We implement a path management component to initiate subflows for three path managers `default, ndiffPorts and fullmesh` as described in Sect. 2.1. New classes MpTcpNdiffPorts and MpTcp-FullMesh were created for ndiffports and fullmesh path manager. Figure 5 describes the path management component of MPTCP. When a client receives the first DSS ACK it initiates path managers as described in RFC 6824 [21]. The user can configure a path manager prior to simulation. By selecting the ndiff-ports path manager one can control the number of subflows through a socket API MaxSubflows. Similarly, the fullmesh path manager creates a full-mesh of subflows amongst all available IP addresses at the sender and receiver.

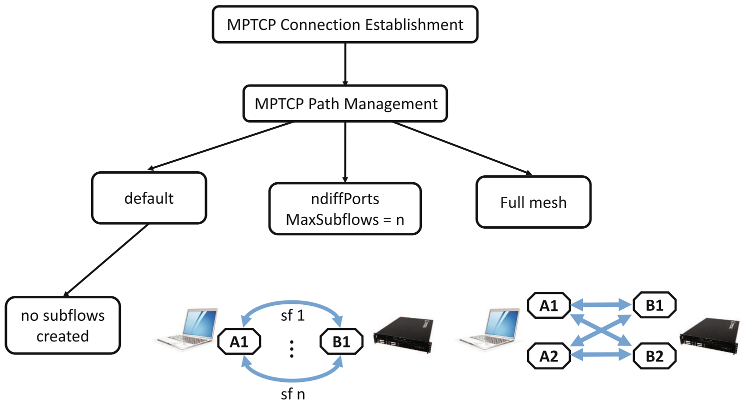

Fig. 5. Path managers functional diagram

5 Simulation

In order to evaluate the efficacy of our implementation, we compare simulation results with Coudron et al. [14]. We simulate a scenario where a multihomed client is connected to a Wide Area Network (WAN) with two network interfaces i.e. *Ethernet* and *wi-fi*. The client connects to routers with links having a bandwidth of 2 Mbps each while, all other links have a bandwidth of 2.4 Gbps as shown in Fig. 6. We enable per packet ECMP to route packets through the network.

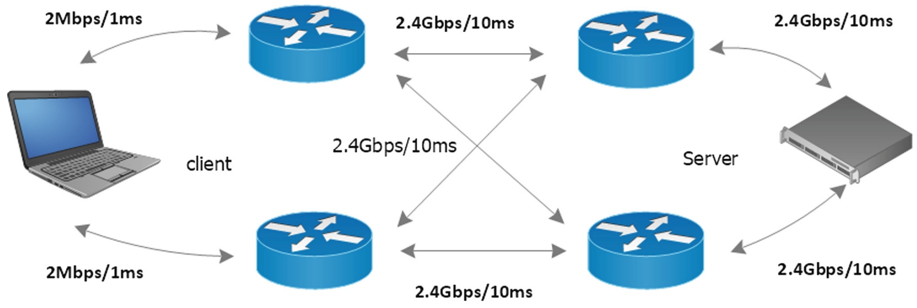

Fig. 6. Simulation topology

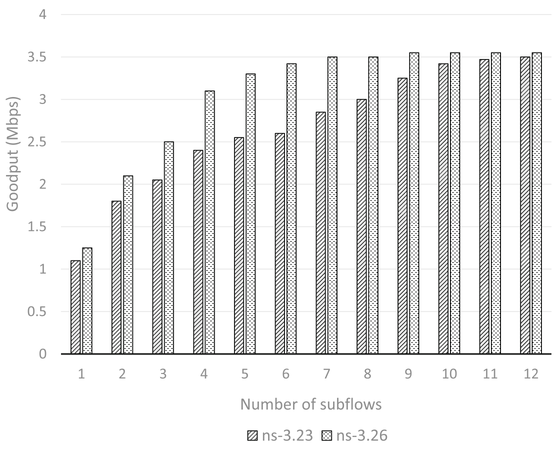

Fig. 7. Goodput for long flows

The client sends a large file to the server and we record the bytes received by the server for different number of MPTCP subflows. We then plot the *goodput* achieved by the MPTCP connection with different number of subflows as shown in Fig. 7. It is evident that with an increase in the number of MPTCP subflows the achieved goodput increases both with ns-3-dev and Coudron et al. [14] implementations. However, the ns-3-dev MPTCP achieves better goodput for less than 8 subflows and this can be attributed to TCP's fast retransmit and fast recovery as well as other enhancements made in ns-3. Notice that, the goodput saturates to approximately to 4 Mbps. Plain TCP performs poorly whereas, increasing the number of subflows allows better utilization of the available capacity. The results are in the same vein as [13]. Furthermore, we perform simulations for *short* and *long flows* for the same topology shown in Fig. 6. We create 4 subflows with ndiffports path manager for each MPTCP connection to complete these flows. We plot the flow completion times (FCTs) for the short flows in Fig. 8 and for the long flows in Fig. 9. The graphs show ns-3-dev MPTCP completes short and long flows earlier than Coudron et al.'s implementation [14]. From the simulation results, it is evident that our ns-3-dev MPTCP implementation performs better than Coudron et al. [14] as a consequence of new features and enhancements in the TCP stack of ns-3.

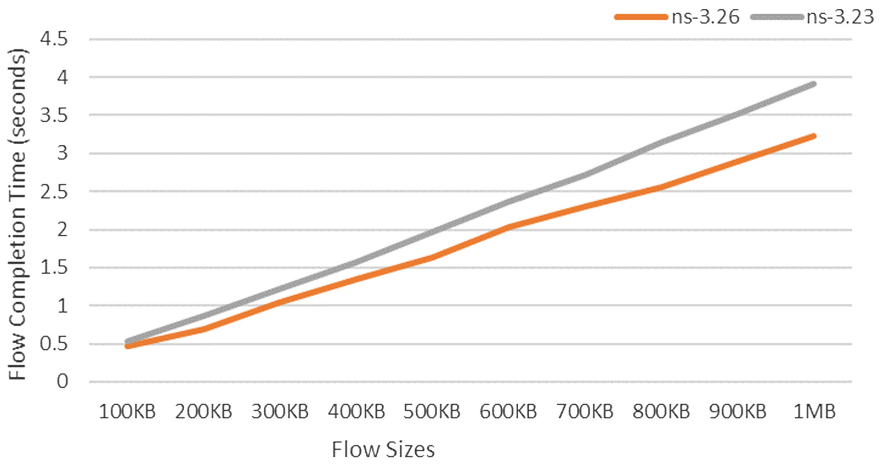

Fig. 8. Flow completion time for short flows

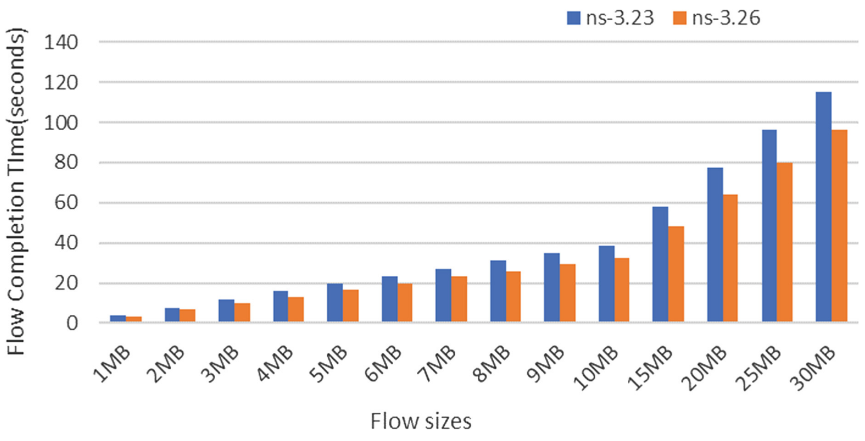

Fig. 9. Flow completion time for long flows

6 Conclusions and Future Work

Simulation tools such as ns-3 are of great help to researchers and network engineers because of the limited access to live network resources and/or testbeds. Presently, ns-3 provides an excellent simulation platform for network engineers and researchers. MPTCP has not been incorporated in the main ns-3 tree by ns-3 community thus far. Efforts have been made in the past to build MPTCP in ns-3 [22,23] and [14]. We build on these implementations, expanding their scope and developing an implementation compatible with the latest ns-3 TCP stack. We implement MPTCP in ns-3-dev and introduce path managers to initiate subflows. We create an MPTCP patch [3] for integration into the main ns-3 tree. Our implementation will help the research community to test new MPTCP proposals and improve upon its functionality and performance. We aim to incorporate our implementation in the ns-3 main tree and additionally develop support for MPTCP congestion control algorithms.

Acknowledgements. We would like to thank Matthieu Coudron for valuable discussions and assistance in navigating through the ns-3 development effort.

References

1. ns-3 website. https://www.nsnam.org/. Accessed 05 Nov 2018
2. ns-3 mptcp patch. https://codereview.appspot.com/369810043/. Accessed 31 Aug 2018
3. ns-3-dev mptcp. https://github.com/Kashif-Nadeem/ns-3-dev-git. Accessed 31 Aug 2018
4. Postel, J.: Transmission Control Protocol. Internet Requests for Comments (1981). https://www.rfc-editor.org/rfc/rfc793.txt
5. Alizadeh, M., et al.: Data center TCP (DCTCP). In: SIGCOMM (2010)
6. Vamanan, B., Hasan, J., Vijaykumar, T.: Deadline aware data center TCP (D2TCP). In: SIGCOMM (2012)
7. Bai, W., et al.: PIAS: practical information-agnostic flow scheduling for datacenter network. In: HotNets (2014)
8. Wischik, D., Handley, M., Bagnulo, M.: The resource pooling principle. ACM SIGCOMM CCR **38**(5), 47–52 (2008)
9. Ong, L., Yoakum, J.: An introduction to the stream control transmission protocol (SCTP). Internet Requests for Comments (2002). https://www.ietf.org/rfc/rfc3286.txt
10. Hasegawa, Y., Yamaguchi, I., Hama, T., Shimonishi, H., Murase., T.: Improved data distribution for multipath TCP communication. In: IEEE Globecom (2005)
11. Zhang, M., Lai, J., Krishnamurthy, A., Peterson, L., Wang, R.: A transport layer approach for improving end-to-end performance and robustness using redundant paths. In: USENIX (ATEC 2004) (2004)
12. Hopps, C.: Analysis of an Equal-Cost Multi-Path Algorithm. Internet Requests for Comments (2000). https://www.rfc-editor.org/rfc/rfc2992.txt
13. Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., Handley, M.: Improving datacenter performance and robustness with multipath TCP. In: ACM SIGCOMM (2011)

14. Coudron, M., Secci, S.: An implementation of multipath TCP in ns3. Comput. Netw. **116**, 1–11 (2017). https://doi.org/10.1016/j.comnet.2017.02.002
15. Ford, A., Raiciu, C., Handley, M., Barre, S., Iyengar, J.: Architectural guidelines for multipath TCP development. Internet Requests for Comments (2011). https://tools.ietf.org/html/rfc6182
16. Carpenter, B., Brim, S.: Middleboxes: taxonomy and issues. Internet Requests for Comments (2001). https://www.rfc-editor.org/rfc/rfc3234.txt
17. Scharf, M., Ford, A.: MPTCP application interface considerations. Internet Requests for Comments (2013). https://tools.ietf.org/html/rfc6897
18. Ford, B., Iyengar, J.: Breaking up the transport logjam. In: ACM HotNets (2008)
19. Multipath Tcp Linux Kernel Implementation. http://multipath-tcp.org/pmwiki.php/Users/ConfigureMPTCP. Accessed 31 Aug 2018
20. Boccassi, L., Fayed, M., Marina, M.: Binder: a system to aggregate multiple internet gateways in community networks. In: LCDNet 2013 (2013)
21. Ford, A., Raiciu, C., Handley, M., Bonaventure, O.: TCP extensions for multipath operation with multiple addresses. Internet Requests for Comments (2013). https://tools.ietf.org/html/rfc6824
22. Chihani, B., Collange, D.: Towards monolingual programming environments. In: WNS3 (2011)
23. Kheirkhah, M., Wakeman, I., Parisis, G.: Multipath-TCP in ns-3 (2015). https://arxiv.org/abs/1510.07721v1
24. Barré, S., Paasch, C., Bonaventure, O.: MultiPath TCP: from theory to practice. In: Domingo-Pascual, J., Manzoni, P., Palazzo, S., Pont, A., Scoglio, C. (eds.) NETWORKING 2011. LNCS, vol. 6640, pp. 444–457. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20757-0_35
25. Allman, M., Paxson, V., Blanton, E.: TCP congestion control. Internet Requests for Comments (2009). https://www.rfc-editor.org/rfc/rfc5681.txt