Research, implementation, and improvement of MPTCP on mobile smart devices

Tongguang Zhang, Shuai Zhao, Bo Cheng, Bingfei Ren & Junliang Chen

INTERNATIONAL JOURNAL OF COMPUTERS AND APPLICATIONS, 2018
https://doi.org/10.1080/1206212X.2018.1455020

Research, implementation, and improvement of MPTCP on mobile smart devices

Tongguang Zhang, Shuai Zhao, Bo Cheng, Bingfei Ren and Junliang Chen

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China

ABSTRACT

As more and more mobile smart devices (MSDs) are equipped with multiple wireless network interfaces (e.g. WiFi, cellular, etc.), MultiPath TCP (MPTCP) can enable MSDs to send data over several interfaces or paths and can achieve better throughput and robust data transfers. MPTCP thus attracts an increasing interest from both academia and industry. This paper systematically studies MPTCP and clearly describes the relationship between each portions of MPTCP. Up to now, MPTCP has not been widely used in the mobile Internet, one of the reasons is that it is a challenge to research, implement and test MPTCP on the actual MSDs. To overcome the shortcoming of the existing researches on MPTCP being mainly limited to network simulators, a novel approach to port MPTCP to MSDs is proposed, which includes three main steps: first, run real mobile operating system (MOS) on VirtualBox; second, port MPTCP to MOS on VirtualBox and test; third, directly copy modified files in the second step to real MOS on MSD and test. By using this method, MPTCP V0.90 is successfully ported to a real smartphone running Android-7.1.1 for the first time. The validity of the method is proved by experiments. In addition, this paper improves the subpath establishment algorithm of MPTCP and applies the improved MPTCP to Mobile Ad Hoc Network (MANET) constructed by MSDs, the test result shows that our algorithm has better performance than the original MPTCP in achieving higher data throughput.

ARTICLE HISTORY
Received 26 February 2018
Accepted 28 February 2018

KEYWORDS
MPTCP; mobile smart device; VirtualBox; NS-3; Android; MANET

1. Introduction

With the rapid development of computer technology, semiconductor technology and communication technology, (1) great changes have taken place in the network environments, that is, from traditional Internet to mobile Internet, and then to Internet of Things (IoT); and (2) network terminal nodes also have great changes, namely, from PC to smart phones, and then to a variety of mobile smart devices (MSDs), a great variety of innovative network applications resulting from them have greatly affected people's lifestyle. Today's MSDs (e.g. tablets, smartphones, etc.) have multiple wireless network inter-faces (WiFi, 3G/4G/5G, Bluetooth, NB-IoT, etc.). This multi-homed feature with multiple network interfaces provides new opportunities for improving application performance and presents higher functional requirements for MSDs, such as fast recovery capability and load balancing function. Transmission Control Protocol (TCP) is, however, the dominant transport protocol, both on the traditional Internet and in mobile Internet. TCP was designed when hosts were equipped with a single network interface, all packets generated by the client/server must be sent from the same IP address while two hosts exchange data via a TCP connection. Despite the multi-homed feature of MSDs, which rarely use cellular and WiFi interfaces simultaneously due to most of their data traffic is controlled by TCP. Therefore, the use of mobile terminal multi-homed characteristics to achieve multi-path transmission is a research hotspot in the field of network protocols. Over the years, TCP has evolved and included various optimizations, such as mTCP, R-MTP, Parallel TCP, CMT-SCTP (Concurrent Multipath Transfer for SCTP) [1] and MPTCP (Multipath TCP) [2], among them, CMT-SCTP and MPTCP are the two most widely discussed and mature multi-path transport protocols. Judging from the current situation, MPTCP has better prospects for development. By enabling MPTCP connections to exchange data over different interfaces, it brings new possibilities to improve the user experience [3].

Up to now, MPTCP has not been widely used in the mobile Internet, one of the reasons, in plain language, is the lack of mass base. If there are many network protocol theory researchers and network protocol engineering and technical personnel who can research, implement and test MPTCP on the actual MSDs, rather than individual

CONTACT Tongguang Zhang jsjoscpu@163.com, ztguang@bupt.edu.cn

enterprises or individuals have the ability to apply MPTCP on the MSDs, it is bound to greatly speed up the popularity of MPTCP protocol in mobile Internet. There is not, however, an efficient method to apply MPTCP on MSDs. MPTCP theory researchers often use simulators to study MPTCP, e.g. NS3-MPTCP[1] module in NS-3. These research results are only theoretical, if they want to apply these results to the real network environment, there are a lot of work to be done. Therefore, we provide a method to solve the problem. The main contributions of this work can be summarized as follows.

(1) To clearly describe the relationship between each portions of MPTCP, we systematically study MPTCP and present the structural module map of MPTCP.
(2) For efficient proting MPTCP on actual MSDs, we firstly port Android-arm on VirtualBox (x86).
(3) Merge MPTCP into Android-x86 and present a test method of MPTCP.
(4) Successfully port and test MPTCP on actual MSD (Android-arm).
(5) Improve the subpath establishment algorithm of MPTCP and apply it to Mobile Ad Hoc Network (MANET) constructed by MSDs.

The rest of the paper is organized as follows: Section 2 reviews related works. Section 3 systematically studies MPTCP on MSDs. Section 4 presents an implementation method of MPTCP on MSDs and describes the experiments and result analysis of MPTCP. Section 5 improves multi-hop routing protocol and MPTCP subpath establishment algorithm. Section 6 concludes the paper and briefly provides directions for future work.

2. Related works

Over the years, multipath transmission protocols on different network protocol stack layers have emerged. [4] presents a complete taxonomy pertaining to multipath transmission (including link, network, transport, application, and cross layers), survey the state-of-the-art for each layer, investigate the problems that each layer aims to address, and make comprehensive assessment of the solutions. Many research works on transport layer have been done and some transport layer protocols evolved from TCP are presented, such as mTCP, R-MTP, Parallel TCP, SCTP [5], and MPTCP. MPTCP is a recent TCP extension that enables the transmission of the data belonging to one connection over different paths or interfaces [6]. MPTCP is an ongoing effort of the Internet Engineering Task Force's (IETF) MPTCP working group [7], that aims at allowing a TCP connection to use multiple paths to maximize resource usage and increase redundancy [8]. Nikravesh et al. [9] conduct an in-depth study of multipath for MPTCP settings, design and implement a flexible software architecture for mobile multipath called MPFlex, which decouples the high-level scheduling algorithm and the low-level OS protocol implementation, and also provides an ideal vantage point for flexibly realizing user-specified multipath policies and is friendly to middleboxes. Many researchers have analyzed the performance of MPTCP on the global Internet and/or cellular networks. Chen et al. [10] analyze the performance of MPTCP in WiFi/cellular networks using bulk transfer applications and conclude that MPTCP provides performance benefits compared to regular TCP. Deng et al. [11,12] have analyzed the performance of MPTCP in hybrid networks (cellular and WiFi). Their measurements show that MPTCP can indeed provide benefits by pooling network resources or enabling seamless handovers [3]. Detailed analysis of the performance of real smartphone applications with MPTCP. Deng et al. [12] compare the performance of single-path TCP over WiFi and LTE networks with MPTCP on multi-homed devices using active measurements and replaying HTTP traffic observed on mobile applications. Chen et al. [10] and Deng et al. [12] also perform measurements with the MPTCP implemen-tation in the Linux kernel. Raiciu et al. [13] discuss how MPTCP can be used to support mobile devices and provide early measurement results. Paasch et al. [14] propose, implement and evaluate path management strategies to efficiently support 3G and WiFi interfaces with MPTCP.

Multipath transmission provides new opportunities for improving mobile application performance. Therefore, industry has also been enthusiastically adopting MPTCP. In September 2013, some built-in apps in iOS7, e.g. voice recognition Siri application leverages MPTCP to send voice samples over both WiFi and cellular interfaces to cope with various failure scenarios [3,15]. In July 2015, Korean Telecom announced that they have enabled MPTCP Linux implementation [16] on Android smartphones (Samsung Galaxy S6 and Galaxy S6 Edge) to bond WiFi and LTE together. These smartphones reach download speeds of 800 Mbps and more [17,18]. In September 2015, OVH, a French ISP and hosting provider, announced their OverTheBox service [19] that uses MPTCP to enable SMEs (small and medium-sized enterprises) to bond several DSL over cable links together [17]. MPTCP Linux implementation v0.86 is ported to Android 4.4.4 with CyanogenMod 11 ROM on Samsung Galaxy S3 smartphones [9]. MPTCP Linux implementation v0.89.5 is ported to Android 4.4.4 [3]. Nevertheless, the papers do not show an efficient method to apply MPTCP protocol to MSDs. To fill this gap, we propose a method to port MPTCP on Android MSDs.

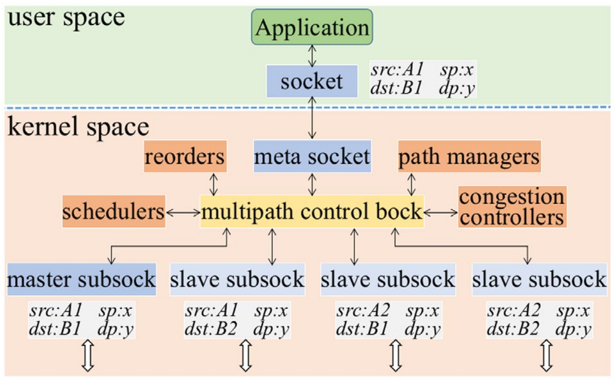

Figure 1. The relationship between each portions of MPTCP.

3. Research of MPTCP

As more and more MSDs are equipped with multiple wireless network interfaces, MSDs are expected to seamlessly use all available WiFi and cellular networks. Therefore, MPTCP attracts an increasing interest from both academia and industry and is used to enable MSDs to send data over several interfaces and to achieve better throughput and robust data transfers. IETF MPTCP working group aims at solving the problems of MPTCP architecture, congestion control, routing, security, API, etc. and ensuring that MPTCP can be backward compatible with traditional TCP. MPTCP can manage many paths, improve the transmission performance and robustness of end-to-end connections, and automatically transfer the traffic from the congested path to the better path. Due to Android is a popular mobile operating system in mobile Internet, in addition, Android Things is an Android-based operating system for MSDs in IoT, therefore, we research the MPTCP Linux implementation [16] and implement it on MSDs with Android installed.

After systematic analysis to MPTCP, the relationship between each portions of MPTCP is clear and the structural module map of MPTCP is designed. As can be seen in Figure 1, the MPTCP Linux implementation consists of several main parts: meta socket, master subsock, slave subsocks, multipath control block (mpcb), path managers, schedulers, reorders, and congestion controllers. If MPTCP is enabled on MSDs running Linux/Android, when an application creates a TCP connection, several data structures are created in the Linux/Android kernel space: a meta socket, mpcb, and subflows socket (master/slave subsocks).

(1) The *meta socket* is directly linked to the socket that is visible to the application in the Linux/Android user space. All data sent and received on a TCP connection passes through this socket.
(2) The *mpcb* [20] provides the ability to turn on or off subflows, and supervises the multiple subflows used in a given connection, runs the path managing algorithm (path managers) for starting or stopping subflows, runs the scheduling algorithm (schedulers) for feeding new application data to subflows, and runs the reordering algorithm (reorders) that allows providing an in-order stream of incoming bytes at the connection-level to the application.
(3) The *master subsock* is a standard sock structure for TCP communications. The *slave subsocks* are not directly visible to applications, and they are initiated, managed and closed by the mpcb. The master subsock and the slave subsocks together form a pool of subflows that the mpcb can use for scheduling outgoing data, and reordering incoming data at the subflow-level in the subflows.

The MPTCP Linux implementation includes three kinds of key algorithms that are not standardized by the IETF: path managers, schedulers, congestion controllers.

3.1. Path managers

An outstanding characteristic of MPTCP is able to use several paths (subflows) inside a single MPTCP connection. Path managers are pluggable kernel modules, are responsible to create and delete subflows and react to events such as the activation/deactivation of a network interface by creating/removing the appropriate subflows.

The establishment process (three-way handshake) of the first subflow (master subsock) is as following: first, the client sends SYN packet with a MPTCP-CAPABLE option to the server; second, the server sends SYN + ACK packet with a MPTCP-CAPABLE option to the client; last, the client sends ACK packet with a MPTCP-CAPABLE option to the server. After initiated a MPTCP connection and established the first subflow (master subsock), MPTCP allows the client/server to advertise its current list of additional IP-addresses to its peer. Firstly, When the server has additional IP-addresses, it will notify the client its additional IP-addresses by sending ACK packets with an ADD_ADDR option over the established subflow (master subsock) and then send another SYN packet with a MPTCP-JOIN option to the client. With the MPTCP-JOIN option, the new subflow (slave subsock) will be associated with the previous established MPTCP connection. Correspondingly the client notify the server its additional IP-address by sending ACK packet with an ADD_ADDR option over the established subflow (master subsock). Once received the packet, the server sends SYN packet with MPTCP-JOIN option to the client's newly notified IP-address, together with the exchanged hashed key for the MPTCP connection, and then successfully initiates a new subflow (slave subsock). The subflows (slave subsocks) associated with the MPTCP connection are not fixed. During the lifetime of the MPTCP connection, the set of subflows (slave subsocks) changes due to the client/server can add or remove a subflow (slave subsock) at any time.

Current path managers in Linux Kernel implementation includes: default, fullmesh, ndiffports, and binder.

3.1.1. Default

The path manager actually does not do anything. The client/server will not announce different IP-addresses nor initiate the creation of new subflows [16]. It, however, will accept the passive creation of new subflows.

3.1.2. Fullmesh

By using the path manager, the client can create subflows among all pairs of IP-addresses of the client and the server IP-addresses learned through the ADD_ADDR option, this creates CxS subflows if the client has C IP-addresses and the server has S IP-addresses. The path manager is well adapted for smartphones that interact with single-homed servers [21].

3.1.3. Ndiffports

The path manager will create subflows across the same pair of IP-addresses by modifying the source-port. Assumes that the client and the server are single-homed, the client creates subflows with different source ports to server. The path manager is well adapted for single-homed clients in datacenters.

3.1.4. Binder

The path manager use Loose Source Routing presented by the paper.

3.2. Schedulers

MPTCP schedulers are responsible to schedule packets over different active subflows. Current schedulers in Linux Kernel implementation includes: default, round-robin and redundant.

3.2.1. Default (lowest RTT first)

The scheduler first schedules data to the subflow with the lowest round-trip-time (RTT) until their congestion window is full. Then, it will send data on the subflow with the next higher RTT. In heterogeneous networks (cellular and WiFi), the scheduler can reduce the application delay and improve the user-experience. The default scheduler is the best known up to today.

3.2.2. Roundrobin

The scheduler sends data on all available subflows in a round-robin fashion. In heterogeneous networks, when transferring bulk data, the scheduler is not really round-robin because congestion window of some subflows will be filled. Its performance is bad and it is only used for academic/testing purposes.

3.2.3. Redundant

The scheduler tries to send data on all available subflows in a redundant way. It is useful to achieve the lowest possible latency by sacrificing the bandwidth.

3.3. Congestion controllers

Except for the congestion control algorithms, each MPTCP subflow behaves as a legacy New Reno TCP flow. After three-way handshake, each subflow maintains its own congestion window and re-transmission scheme, and begins with a slow-start phase before entering the congestion avoidance phase [10]. The design of congestion control in MPTCP must follow the following two principles:

(1) MPTCP and traditional TCP should have the same throughput at the connection-level, instead of each of the MPTCP subflows having the same throughput as the traditional TCP at the subflow-level.
(2) MPTCP should choose the better subflow when selecting subflows. The chosen congestion controller has definitely an impact on the performance of MPTCP.

Current available congestion controllers in Linux Kernel implementation includes: lia (alias Linked Increa-se Algorithm), olia (alias Opportunistic Linked Increase Algorithm), wVegas (alias Delay-based Congestion Control for MPTCP) and balia (alias Balanced Linked Adaptation Congestion Control Algorithm).

From the above, we can see that MPTCP includes various techniques to send data belonging to one MPTCP connection on multiple subflows. Detailed information about the operation of MPTCP may be found in [22–24]. So far MPTCP has not been widely used in the mobile Internet, one of the reasons is that there is not an efficient method to apply MPTCP to MSDs. MPTCP theory researchers often use simulators to study MPTCP. These research results are only theoretical, if they want to apply these results to the real network environment, there are a lot of work to be done. We fill this gap between theory and practice in next section by presenting a method to use MPTCP on MSDs with Android installed.

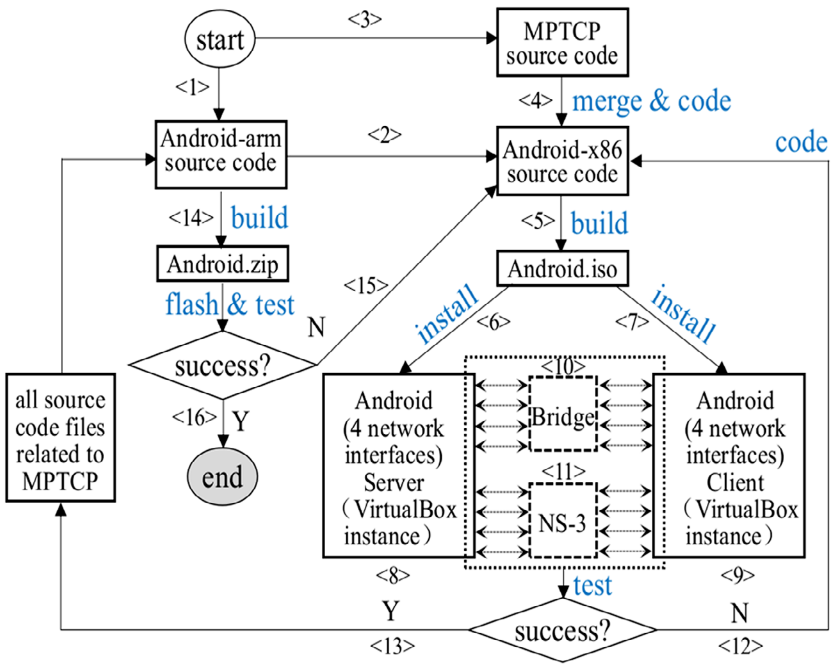

Figure 2. Block diagram of implementing MPTCP on MSDs.

4. Implementation of MPTCP on MSDs

4.1. Implementation method of MPTCP on MSDs

Due to Android has a large number of users in mobile Internet, in addition, Android Things is an Android-based operating system platform for smart devices in IoT, therefore, we research and implement MPTCP on MSDs with Android installed. Several backports of the MPTCP kernel on Android smartphones were released in the last years [3]. However, these ports were often relied on old versions of the MPTCP kernel. In this paper, we base on a newer version 0.90 of the MPTCP Linux kernel and port it to a smartphone running Android 7.1.1.

We propose a novel method to efficiently port MPTCP to actual MSDs. Figure 2 shows the block diagram of implementing MPTCP on MSDs. The novel method includes three main parts.

(1) Porting LineageOS to real smartphone

These instructions [25] can be consulted. The specific steps are shown below.

S1 (⟨1⟩): Get the source code of LineageOS-14.1-kiwi for Huawei honor 5x from github [26], and then get specific configuration and kernel (the device-specific code) of Huawei honor 5x.

S2 (⟨14⟩): Build LineageOS image from the source code, and get the LineageOS installer package, that is, lineage-14.1-kiwi.zip.

S3: Install lineage-14.1-kiwi.zip on the real smartphone.

S4: Test whether the smartphone can be used normally, if not normal, solve the logical problems existing in the source code, then repeat S2–S4 until the smartphone can be used properly.

(2) Porting LineageOS and MPTCP to VirtualBox

The key part of the implementation of the virtual world of FEP is to port LineageOS-arm to VirtualBox (x86). The process involves the following steps.

S1 (⟨2⟩): Copy Android source code files (/opt/Android-arm) to directory /opt/Android-x86, then enter /opt/Android-x86.

S2: Add start-up support for x86 system.

S3: Modify partial Linux kernel driver.

S4: Modify the build system of Android-x86.

S5 (⟨3⟩): Get the source code of MPTCP.

S6 (⟨4⟩): By analyzing the difference of folders and files between MPTCP Linux implementation and Android kernel, to merge the source code of MPTCP into the source code of Android-x86 kernel and modify the source code of MPTCP from the perspective of syntax.

S7 (⟨5⟩): Build Android-x86, if there are some syntax problems, repeat S2–S7, if there is no syntax problem, the ISO file, namely, cm_android_x86_64.iso will be generated, go to S8.

S8 (⟨6⟩⟨7⟩): Install cm_android_x86_64.iso in VirtualBox, if there are some problems during the installation process, repeat S2–S8, if there is no problem, successfully install LineageOS-x86 in VirtualBox, go to S9.

S9 (⟨8⟩⟨9⟩): Start LineageOS-x86 in VirtualBox, if there are some problems during the starting-up process, repeat S2–S9, if there is no problem, will successfully enter Android GUI (Graphical User Interface), go to S10.

Table 1. Comparisons between novel and traditional method.

Contrastive term	Traditional method	Novel method
Main test platform	Real smartphone	VirtualBox
The cycle of solving a problem	≥1.5 h	≤12 min
Test multipath transmission (MPTCP)	Very difficult	Very easy
Requirements for users	Basic users of Linux	Intermediate or advanced users of Linux

S10 (⟨10⟩⟨11⟩): By connecting two VirtualBox instances via Linux bridge or NS3 to test whether MPTCP can be used normally (specific testing process is provided in Section 4.2), if there are some problems, repeat S6–S10, if there is no problem, go to S11.

S11: Now, successfully port LineageOS-arm and MPTCP to X86 platform.

Note that it is difficult in the implementation of steps S2–S4.

(3) Merge MPTCP into Android on the real smartphone

Merge modified files of MPTCP into real Android Kernel, build and install it, then test function correctness of MPTCP on the real smartphone. The specific steps are shown below.

S1 (⟨13⟩): merge modified files of MPTCP into real Android Kernel.

S2 (⟨14⟩): Build LineageOS image from the source code, and get the installer package (lineage-14.1-kiwi.zip).

S3: Install lineage-14.1-kiwi.zip on the real smartphone.

S4: Test whether MPTCP can be used normally (specific testing process is provided in Section 4.2), if not normal, solve the logical problems existing in the source code, then repeat S2–S4.

Table 1 shows the comparisons between the novel and traditional method from the following four aspects:

(1) Main test platform

Our research team has been working on a project which requests that MSDs should support MPTCP. Before using the novel method, we tested whether MPTCP can be used normally directly on the real smartphone, however, we spent a lot of time and still failed. Therefore, we adopted the novel method and use the VirtualBox as the main test platform.

(2) The cycle of solving a problem

When using the traditional method, the cycle of solving a grammatical or logical error includes: (1) modify source code; (2) build Android; (3) flash Android to real smartphone; (4) test whether smartphone can start up successfully; (5) test whether MPTCP can be used normally on real smartphone. If there are some grammatical or logical errors, smartphone cannot start up successfully. The cycle takes at least 1.5 h. When using the novel method, the cycle of solving a grammatical or logical error includes: (1) modify source code; (2) build Android ISO; (3) install Android on VirtualBox; (4) test whether VirtualBox can start up successfully; (5) test whether MPTCP can be used normally on VirtualBox. The cycle takes no more than 12 min. Because there are more grammatical or logical errors when porting Android and merging MPTCP into Android, and each error corresponds to a cycle, thus we can save a lot of time using the novel method than using the traditional method.

(3) Test multipath transmission (MPTCP)

It is very difficult to test multipath transmission between two smartphones. However, it is very easy to test multipath transmission between two VirtualBox instances.

(4) Requirements for users.

The major weakness of the novel method is that it is hard to use for the basic users of Linux, especially for the users who have not used Linux.

4.2. Testing and result analysis

The test environment is running on an IBM Server with 32-cores 2.0 GHz Intel Xeon CPU, 64 GB memory, and 64-bit Fedora 24 installed. The experiment consists of two steps: (1) test MPTCP on VirtualBox; (2) test MPTCP on real smartphone.

4.2.1. Testing MPTCP on VirtualBox

In this section, we seek to answer the question: How much can a user benefit from using MPTCP over multiple network interfaces relative to using the either interface alone? As shown in Figure 2, two VirtualBox instances are connected via Linux bridge, Android-Server and Android-Client, respectively, have four network interfaces belonging to four different network segments. We conduct four experiments focusing on 1-path, 2-path, 3-path, and 4-path MPTCP scenarios, respectively. We use the Full Mesh path manager and the default RTT-based scheduler. As can be seen from Figure 3, Android-Client successfully download the file chrome.apk (64 MB) from Android-Server via 1, 2, 3, 4 MPTCP sub-paths, respectively. *Demos are available from* [27].

In the four experiments, the file chrome.apk is transmitted by consuming 282, 159, 110, 103 s, respectively. The

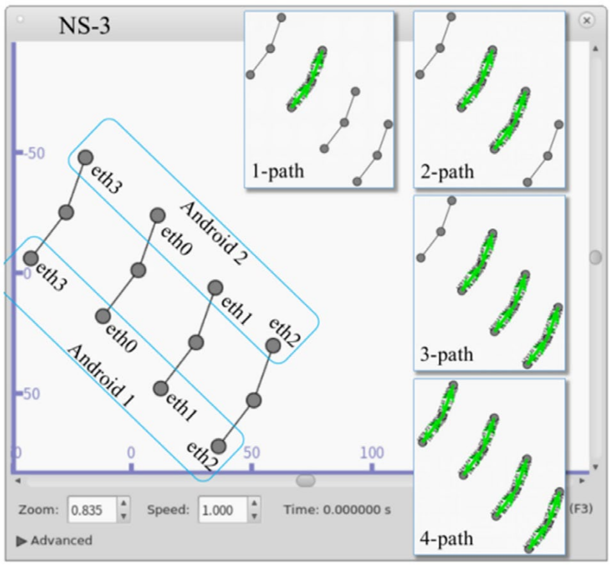

Figure 3. Testing MPTCP on VirtualBox.

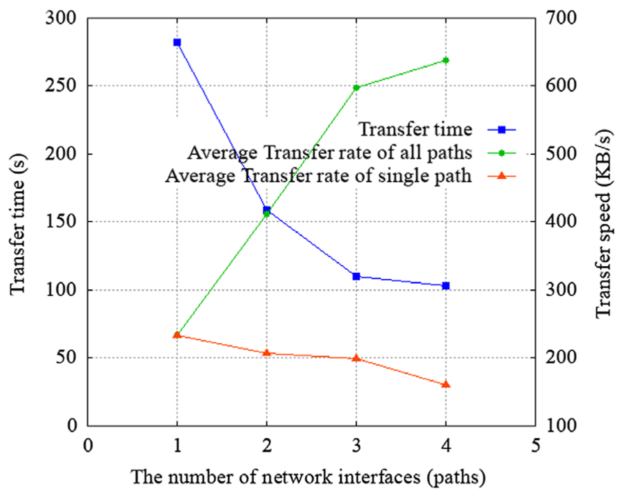

Figure 4. Data transmission performance.

average total transmission rates are 230, 410, 600, 640 KB, respectively. Average single path transmission rates are 230, 210, 200, 160 KB, respectively. Figure 4 shows the relation between the transmission time and the transmission rate. Theoretically, the transmission time and the network interface (path) number are linearly decreasing relationship, and the average total transmission rate and the network interface (path) number are linearly increasing relationship. In fact, however, it can be seen from Figure 4 that they are not strictly linear relationship, the root cause is the timing of using the MPTCP sub-paths. For example, in the fourth experiment (four-path), at the beginning, use the first sub-path (eth0↔eth0) to transfer data, the second sub-path (eth1↔eth1) is enabled to transfer data after 8 s, the third sub-path (eth2↔eth2) is enabled after another 8 s, the fourth sub-path (eth3↔eth3) is enabled after another 15 s. In addition, the average single path transmission rate theoretically has nothing to do with the network interface number. However, in fact, as shown in Figure 4, the average single path transmission rate decreases as the number of network interfaces increases, the main reason is the increase in system load (CPU, network, etc.).

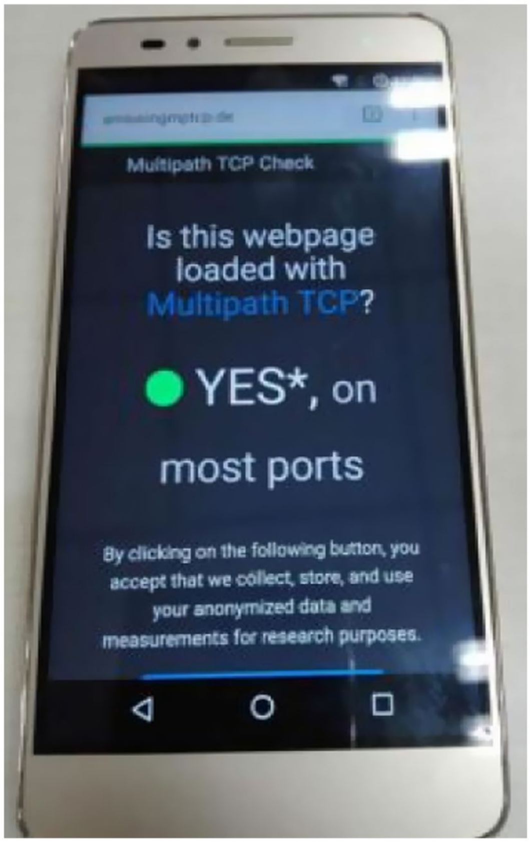
amiusingmptcp.de
Multipath TCP Check
Is this webpage loaded with Multipath TCP?
YES*, on most ports
By clicking on the following button, you accept that we collect, store, and use your anonymized data and measurements for research purposes.

Figure 5. Testing MPTCP on Smartphone.

4.2.2. Testing MPTCP on real smartphone

Based on the success of the previous experiments, we integrate the modified files of MPTCP into real Android, and verify whether MPTCP is enabled on smartphone by visiting the url http://amiusingmptcp.de/. Figure 5 shows MPTCP is ported to smartphone successfully.

5. Improvement of MPTCP

Natural disasters, e.g. earthquakes, floods, tsunamis, etc., often cause breakdown or interruption of communication infrastructures. To improve the efficiency of search-and-rescue, it is necessary to provide communications among people or MSDs. Smartphones are more powerful and can be used to construct MANETs. In addition, UAV cluster is a research hotspot and can be also used to carry out reconnaissance and surveillance. However, due to MSDs move in and out of network coverage areas and

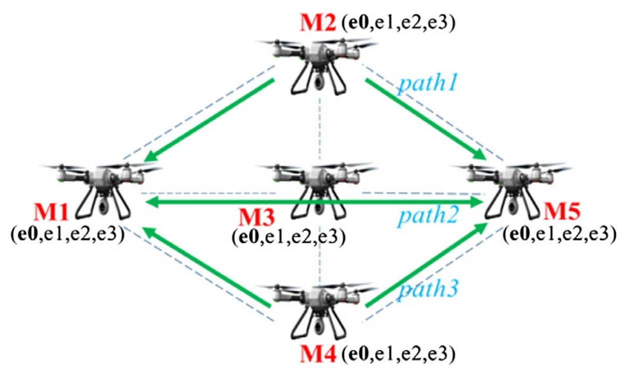

Figure 6. The application scenario of MPTCP in MANETs.

the topology is constantly changing, there are the issues exhibited by MANETs, that is, frequent disconnections and failed transmissions. Hence, we use MPTCP to solve the issues aiming to offer more efficient use of multiple subpaths and better network traffic load balancing. In this section, we improve multi-hop routing protocol (MHRP) that adds routing table entries according to the number of next hops and network interface number, and improve the establishment algorithm for MPTCP subpaths. To our best knowledge, it is the first time to simultaneously research and improve both the algorithm of adding routing table entries and the MPTCP subpaths creating algorithm.

Figure 6 shows a scenario of MPTCP in MANETs (MiM). We assume each mobile node (M1–M5) is equipped with four network interfaces (e0, e1, e2, e3). IP addresses allocated for network interfaces belong to different network segments. In expression 1, Y denotes the mobile nodes (M1–M5), X denotes the interfaces (e0, e1, e2, e3) in nodes (M1–M5).

$$112.26.X.Y/24,\ X \in [0,\ 3] Y \in [1,\ 5] \tag{1}$$

5.1. *Improving multi-hop routing protocol*

In this paper, OSPF MANET Designated Routers (MDR) is used as MHRP. In Figure 6, MDR can only generate routing information for e0, however, MPTCP needs to build subpaths using the interfaces (e0–e3). Therefore, it is need to improve MHRP to support for the establishment of MPTCP subpaths, the improved method is as follow: generate routing table entries for e0, meanwhile, only one routing table entry is generated for each destination address (e1–e3) respectively. Thus, in M1, there are six routing entries to M5 generated by the improved MDR, as represented in Equations (2) and (3), where E_0RE represents the routing entry for e0 in M1, E_XRE represents the routing entry for e1, e2, e3 in M1.

$$E_0\text{RE} = \text{ nexthop via } 112.26.0.Y \text{ eth0 to } 112.26.0.5,\ Y \in [2,\ 4] \tag{2}$$

$$E_X\text{RE} = \text{ nexthop via } 112.26.0.Y \text{ eth0 to } 112.26.X.5, X \in [1,\ 3] Y = X + 1 \tag{3}$$

The algorithm of improving multi-hop routing protocol is as follows.

Algorithm: Improving multi-hop routing protocol

```
VAR: num_nexthop //The number of next hops
VAR: no_if //Network interface number
    //corresponding to network
    //interface (e.g. e1,e2,e3)
VAR: rib    //route information base
1. BEGIN:
2. PROCEDURE: add_routing_entry
3.    //here, we assume that the destination ip is
4.    //112.26.0.5
5. IF p0 = 112.26.0.5
6. p1 = 112.26.1.5
7. p2 = 112.26.2.5
8. p3 = 112.26.3.5
9. END IF
10. num_nexthop = 0
11. nexthop = rib → nexthop
12. WHILE nexthop ! = NULL
13.     nexthop = nexthop → next
14.     num_nexthop++;
15. END WHILE
16. add_routing_entry (p1, rib, num_nexthop, 1);
17. add_routing_entry (p2, rib, num_nexthop, 2);
18. add_routing_entry (p3, rib, num_nexthop, 3);
19. return add_routing_entry (p0, rib, 0, 0);
20. END PROCEDURE
21.
22. PROCEDURE: choose_nexthop
23. IF (num_nexthop == 1
24.     || (num_nexthop == 2 && (no_if == 1 || no_if == 3))
25.     || (num_nexthop == 3 && no_if == 1))
26. use the first hop
27. END IF
28. IF ((num_nexthop == 2 && no_if == 2)
29.     || (num_nexthop == 3 && no_if == 2))
30. use the second hop
31. END IF
32. IF (num_nexthop == 3 && no_if == 3)
33.     use the third hop
34. END IF
35. END PROCEDURE
36. END
```

5.2. Improving subpaths establishment algorithm

Algorithm: Improving algorithm for creating subflow

```
VAR: local_addr[i] ← local ip addresses
VAR: remote_addr[j]
VAR: struct mptcp_loc4 loc
VAR: struct mptcp_rem4 rem
1.   BEGIN:
2.   PROCEDURE: create_subflow_worker
3.   remote_addr[j] ← peer ip addresses & ports
4.   FOR ip_src in local_addr[i]
5.   FOR ip_dst in remote_addr[j]
6.   IF ip_src can syn ip_dst successfully &&
7.       (ip_src/24 == ip_dst/24||ip_src/24 == 0x00001a70)
8.       rem.addr = ip_dst → addr
9.       rem.port = ip_dst → port
10.        loc.addr = ip_src → addr
11.        loc.port = ip_src → random_port
12.        mptcp_v4_subflows(loc, rem)
13.    END IF
14.    END FOR
15.    END FOR
16.    END PROCEDURE
17.    END
```

Table 2. The established subpaths.

Subpath	src-IP	dst-IP	Middle nodes	NoNH
Master-subpath	M5:e0	M1:e0	M2, M3, M4	3
Slave-subpath1	M5:e1	M1:e1	M2	1
Slave-subpath2	M5:e2	M1:e2	M3	1
Slave-subpath3	M5:e3	M1:e3	M4	1

The improving algorithm for creating subflow is shown as above. The key is to create subpaths using pairs of IP addresses that belong to the same network segment. Let us have a look at Figure 6 again, we assume M1 acts as server, M5 acts as client, the original MPTCP uses M5:(e0, e1, e2, e3) and M1:(e0, e1, e2, e3) to establish subpaths, such as the MPTCP subpaths of M5–M1, as represented in expression 4.

$$\text{src: } 112.26.X.5,\ \text{dst: } 112.26.Y.1,\ X, Y \in [0,\ 3] \quad (4)$$

$$\text{src: } 112.26.X.5,\ \text{dst: } 112.26.X.1,\ X \in [0,\ 3] \quad (5)$$

Why not be subpaths like expression 5? The root reason resides in the improvement to routing protocol in Section 5.1. For better network traffic load balancing, it is necessary to establish subpaths represented in expression 5. Hence, it is essential to improve the subpaths establishment algorithm.

Table 2 shows the subpaths established by the improved MPTCP. NoNH is the number of next hops. NoNH = 3 of master-subpath improves its connectivity reliability. The interfaces (M1–M5:e0) play three roles: (1) used by quagga to generate multi-hop routing entries; (2) used to establish master subpath; (3) forward packets as mid-nodes. The interfaces M1–M5:(e1, e2, e3) are used to establish slave subpaths.

5.3. Testing and evaluation

The test environment is the same to Section 4.2. We create five VirtualBox instances for Android, each of them has four network interfaces. Network topology is generated by NS-3 script [28]. We run command ***nc*** in M1 to make it as a file server, run command ***nc*** in M5 to download file *bash* from M1. The test results show that MPTCP is successfully running in MANETs. Note that the topology is simplified by letting NS-3 nodes remain quiescent without affecting the function test of MiM.

We also carry out 30 times experiments to compare data transfer performances of TCP-MDR (TCP with Multi-hop Routing), MPTCP-MDR (original MPTCP with Multi-hop Routing), Improved-MPTCP-MDR (improved MPTCP with Multi-hop Routing). As shown in Figure 7, Improved-MPTCP- MDR is better than TCP-MDR, however, MPTCP-MDR is worse than TCP-MDR. The reason is that original MPTCP cannot distinguish which subpaths belong to one pair of nodes and which subpaths belong to multiple pairs of nodes.

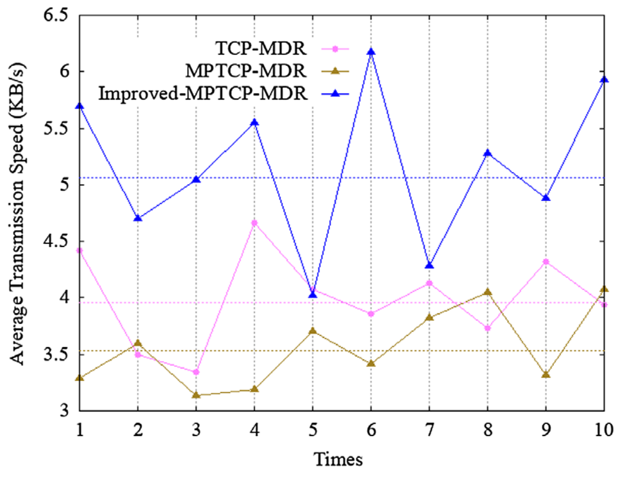

Figure 7. The evaluation of improved MPTCP.

All modified source code files and MPTCP demo (mp4 file) can be got from [28].

6. Conclusions

We have systematically studied MPTCP and presented the structural module map of MPTCP to clearly describe the relationship between each portions of MPTCP. The map and its explanation make it easy for researchers to understand MPTCP. In order to improve the efficiency of porting MPTCP, we have proposed a novel method to port MPTCP to mobile smart devices. According to the public document, we successfully port MPTCP V0.90 to a real smartphone with Android-7.1.1 installed for the first time. The validity of the method is proved by the experiments. In addition, we have improved MHRP that adds routing table entries according to the number of next hops and network interface number, and improved the establishment algorithm for MPTCP subpaths. To our best knowledge, we are the first to simultaneously research and improve both the algorithm of adding routing table entries and the MPTCP subpaths creating algorithm. In the fields of the research and application of network protocols in MANETs, IoT, mobile Internet, and wireless sensors networks, both researchers and industry can benefit from the method.

Note

1 https://github.com/mkheirkhah/mptcp

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was supported by National Natural Science Foundation of China [grant number 61501048], [grant number U1536111], [grant number 61372115], [grant number U1536112]; Beijing Natural Science Foundation [grant number 4182042]; The Fundamental Research Funds for the Central Universities [grant number 2017RC12].

Notes on contributors

Tongguang Zhang is currently working toward the PhD degree in computer science and technology in the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications (BUPT). His current research interests include mobile Internet technology, Internet of Things technology, communication software and distribute computing, embedded system and service computing.

Shuai Zhao received the PhD degree in computer science and technology from the Beijing University of Posts and Telecommunications (supervisor: Prof. Junliang Chen) in June 2014. He is lecturer of computer science in State Key Laboratory of Networking and Switching Technology at Beijing University of Posts and Telecommunications. His current research interests include Internet of Things technology and service computing.

Bo Cheng received the PhD degree in computer science from the University of Electronics Science and Technology of China, Chengdu, China, in 2006. He is a professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. His research interests network services and intelligence, Internet of Things technology, communication software and distribute computing, etc. He is a member of the IEEE.

Bingfei Ren is a PhD candidate in the State Key Laboratory of Networking and Switching Technology at Beijing University of Posts and Telecommunications (BUPT). He received the BS degree in software engineering in 2014 from BUPT in China. His main research interests include mobile computing, mobile security and wireless network.

Junliang Chen received the BS degree in electrical engineering from Shanghai Jiaotong University, China, in 1955 and the PhD degree in electrical engineering in May, 1961, from the Moscow Institute of Radio Engineering, formerly Soviet Russia. He is the chairman and a professor of the Research Institute of Networking and Switching Technology at the Beijing University of Posts and Telecommunications (BUPT). He has been working at BUPT since 1955. His research interests are in the area of communication networks and next generation service creation technology. He was elected as a member of the Chinese Academy of Science in 1991, and a member of the Chinese Academy of Engineering in 1994, for his contributions to fault diagnosis in stored program control exchange. He received the first, second, and third prizes of National Scientific and Technological Progress Award in 1988, 2004, and 1999, respectively.

References

[1] Becke B, Adhari H, Rathgeb EP, et al. Comparison of multipath TCP and CMT-SCTP based on intercontinental measurements. In Proceedings of IEEE Global Communications Conference; 2013; Atlanta, GA. p. 1360–1366.
[2] Peng Q, Walid A, Hwang J, et al. Multipath TCP: analysis, design, and implementation. IEEE/ACM Transactions on Networking. 2016;24:596–609.
[3] Coninck QD, Baerts M, Hesmans B, et al. Observing real smartphone applications over multipath TCP. IEEE Commun Mag. 2016;54(3):88–93.
[4] Li M, Lukyanenko A, Ou ZH, et al. Multipath transmission for the internet: a survey. In IEEE Communications Surveys & Tutorials. Vol. 4; Fourthquarter 2016. p. 2887–2925.
[5] Stream control transmission protocol. IETF RFC 4960; 2007.
[6] TCP Extensions for Multipath Operation with Multiple Addresses. IETF RFC 6824; 2013.
[7] MPTCP working group. [Online]. Available from: https://datatracker.ietf.org/wg/mptcp/documents/
[8] Shamszaman ZU, Ara SS, Chong I. Feasibility considerations of multipath TCP in dealing with big data application. In The International Conference on Information Networking 2013 (ICOIN); 2013; Bangkok. p. 708–713.
[9] Nikravesh A, Guo Y, Qian Z, et al. An in-depth understanding of MPTCP on mobile devices: measurement and system design. In ACM MobiCom; 2016. p. 189–201.
[10] Chen YC, Lim Y, Gibbens RJ, et al. A measurement-based study of multipath TCP performance over wireless networks. In Proceedings of 2013 ACM Conference on Internet Measurement Conference (IMC'13); 2013. p. 455–468.
[11] Lim YS, Lim Y, Nahum EM, et al. How green is multipath tcp for mobile devices?. In Proceedings of the 4th workshop on All things cellular: operations, applications, & challenges; 2014. ACM. p. 3–8.
[12] Deng S, Netravali R, Sivaraman A, et al. Wifi, LTE, or both?: measuring multi-homed wireless internet performance. In Proceedings of 2014 Conference on Internet Measurement Conference; 2014; Vancouver, BC, Canada: ACM. p. 181–194.
[13] Raiciu C, Niculescu D, Bagnulo M, et al. Opportunistic mobility with multipath TCP. In: ACM MobiArch. 2011;2011:7–12.
[14] Paasch C, Detal G, Duchene F, et al. Exploring mobile/wifi handover with multipath TCP. In ACM SIGCOMM workshop Cell Net; 2012; New York, NY: ACM. p. 31–36.

[15] iOS: MPTCP Support in iOS 7. [Online]. Available from: https://support.apple.com/en-us/HT201373
[16] Multipath TCP in the Linux Kernel. [Online]. Available from: http://www.multipath-tcp.org
[17] Coninck QD, Baerts M, Hesmans B, et al. A first analysis of MPTCP on smartphones. In: The 17th International Passive and Active Measurements Conference; 2016; Heraklion, Greece. p. 57–69.
[18] SungHoon S. KT's GiGA LTE. [Online]. Available from: https://www.ietf.org/proceedings/93/slides/slides-93-mptcp-3.pdf
[19] OverTheBox service. [Online]. Available from: https://www.ovhtelecom.fr/overthebox/
[20] Barré S, Paasch C, Bonaventure O. MultiPath TCP: from theory to practice. In: International Conference on Research in Networking; 2011. p. 444–457.
[21] Hesmans B, Bonaventure O. An enhanced socket API for Multipath TCP. In: Proceedings of 2016 Applied Networking Research Workshop; 2016; New York, NY: p. 1–6.
[22] An MPTCP Option for Network-Assisted MPTCP Deployments: Plain Transport Mode. Internet draft, draft-boucadair-mptcp-plain-mode-08; 2016. Work in progress.
[23] Extensions for Network-Assisted MPTCP Deployment Models. Internet draft, draft-boucadair-mptcp-plain-mode-10; 2017. Work in progress.
[24] Use Cases and Operational Experience with Multipath TCP. IETF RFC 8041; 2017.
[25] Build Lineageos ROM. [Online]. Available from: http://www.lineageosrom.com/2017/01/how-to-build-lineageos-rom-for-any.html
[26] LineageOS source code. https://github.com/LineageOS/
[27] Available from: https://github.com/ztguang/easyPort
[28] Available from: https://github.com/ztguang/MiM