Research Article

MPTCP Tunnel: An Architecture for Aggregating Bandwidth of Heterogeneous Access Networks

Xiaolan Liu ^{1,2}, Danfeng Shan ^{1,2}, Ran Shu ^{1,2} and Tong Zhang ^{1,2}

¹Tsinghua National Laboratory for Information Science and Technology, Beijing, China

²Department of Computer Science and Technology, Tsinghua University, Beijing, China

Correspondence should be addressed to Xiaolan Liu; lxljun@163.com

Received 8 August 2017; Revised 30 December 2017; Accepted 22 January 2018; Published 26 March 2018

Academic Editor: Ernestina Cianca

Copyright © 2018 Xiaolan Liu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Fixed and cellular networks are two typical access networks provided by operators. Fixed access network is widely employed; nevertheless, its bandwidth is sometimes not sufficient enough to meet user bandwidth requirements. Meanwhile, cellular access network owns unique advantages of wider coverage, faster increasing link speed, more flexible deployment, and so forth. Therefore, it is attractive for operators to mitigate the bandwidth shortage by bundling these two. Actually, there have been existing schemes proposed to aggregate the bandwidth of two access networks, whereas they all have their own problems, like packet reordering or extra latency overhead. To address this problem, we design new architecture, *MPTCP Tunnel*, to aggregate the bandwidth of multiple heterogeneous access networks from the perspective of operators. MPTCP Tunnel uses MPTCP, which solves the reordering problem essentially, to bundle multiple access networks. Besides, MPTCP Tunnel sets up only one MPTCP connection at play which adapts itself to multiple traffic types and TCP flows. Furthermore, MPTCP Tunnel forwards intact IP packets through access networks, maintaining the end-to-end TCP semantics. Experimental results manifest that MPTCP Tunnel can efficiently aggregate the bandwidth of multiple access networks and is more adaptable to the increasing heterogeneity of access networks than existing mechanisms.

1. Introduction

Nowadays, operators present two approaches to the access of Internet services: one is fixed access network (cable, xDSL, FTTH, etc.), and the other is cellular access network (2G, 3G, LTE, etc.). Because of reliability and stability, fixed access networks transmit more than 90% of total Internet traffic in most countries [1, 2]. Popular as it is, fixed access network is in a dilemma in bandwidth provision sometimes. On the one hand, there is a surge in bandwidth demand from enterprises to individual users [3]. On the other hand, the bandwidth of fixed access networks is limited by physical media. Being a territorial network, fixed access network faces difficulties in deploying a new one or updating existing ones in a short period. Though FTTH (fiber to the home) and optical network are prior choices essentially, there still exist limitations in their deployment, especially in old downtowns or remote rural areas. Meanwhile, there are numerous advantages owned by cellular access network, such as larger

coverage, faster increasing link speed, and more flexible deployment.

From the perspective of operators, it is equally important to ensure the best quality of service (QoS) as well as the quality of experience (QoE) and to maintain the least operating costs. The benefits of cellular network can be leveraged to solve the bandwidth shortage of fixed access network with negligible construction costs. As shown in Figure 1, most CPE (customer premise equipment) provided by operators can access both fixed and cellular networks simultaneously. Combined with the deployment of CPE and a bundling gateway (BGW), it is feasible and economical for operators to solve the bandwidth shortage of fixed access network by aggregating the bandwidth of fixed and cellular access networks.

Hybrid access (HYA) network architecture [4], one kind of multiple access network aggregation mechanisms, is an encapsulation approach. It bundles multiple network paths with the IP Tunnel technology. Any packet from any TCP flow is scheduled to different link by a traffic scheduler and

FIGURE 1: A network scenario in which a user can access Internet service through fixed and cellular access networks simultaneously. CPE is a multipath-capable device that can access fixed and cellular networks at the same time. The dashed circle denotes the coverage of the eNB.

transmitted in IP Tunnel after being appended to a new IP header. Different links have disparate latencies. Packets of the same TCP flow may arrive at another side of access networks out of order, which incurs the reordering problem that affects network performance greatly. However, it is difficult for traffic scheduler to schedule traffic according to link states without the feedback from the transport layer. Furthermore, a special designed mechanism is needed at BGW to tackle the reordering problem in HYA. Another kind of mechanism is a Plain Transport Mode (PTM) of a network-assisted-MPTCP deployment model [5]. This approach bundles multiple access networks with MPTCP and mitigates the reordering problem essentially. Nevertheless, PTM establishes one MPTCP connection for each TCP flow. The establishment of a new MPTCP connection incurs extra latency [6] and CPU costs and, thus, decreases network performance greatly. To summarize, these two kinds of mechanisms introduce new problems, respectively, when solving the bandwidth shortage of fixed access network. In this context, a new approach that can address reordering problem while not decreasing the network performance is required.

For this purpose, we propose new architecture named *MPTCP Tunnel* to tackle the bandwidth shortage problem of fixed access network. The entire design of MPTCP Tunnel consists of the following gist:

- (i) Bundling multiple access networks with MPTCP. Our architecture uses MPTCP to aggregate the bandwidth of heterogeneous access networks. With MPTCP properties, it can address the out-of-order problem resulting from heterogeneity of access networks naturally and adapt to the dynamic variation of network states.
- (ii) Only one MPTCP connection is established between two proxies. Such a practice avoids the extra latency induced by the establishment of multiple MPTCP connections. Packet from any TCP flow is dealt with in the same way when forwarded through the MPTCP connection. That is, there is no need to differentiate TCP flows, so MPTCP Tunnel can adapt to multiple traffic types and TCP flows.
- (iii) The original IP packet is received as a whole from the link layer and transmitted in the MPTCP connection

through the access network. After that, it will be forwarded to link layer directly. The preserving of the packet header maintains the end-to-end TCP semantics.

We evaluate MPTCP Tunnel with comprehensive experiments in Linux and compare it with HYA under varying network states. Our results demonstrate that MPTCP Tunnel can bundle fixed and cellular access networks efficiently and is more adaptable to dynamic variation of network states than HYA. The throughput decrease of MPTCP Tunnel is only 25.5% and 12.5% to that of HYA when latency increases to 100 ms and packet loss rate reaches to 5%, respectively.

The rest of the paper is organized as follows: in Section 2, we outline the related work of this paper. The design requirements of MPTCP Tunnel are discussed in Section 3. Section 4 elaborates the detailed design of the architecture, and the implementation is presented in Section 5. We evaluate the MPTCP Tunnel performance in Section 6. Finally, this work is discussed in Section 7 and concluded in Section 8.

2. Related Work

There are numerous researches about the aggregation of multiple heterogeneous access networks. Generally, they can be divided into operator-based and mobile-host-based mechanisms. In the first type of mechanism, end hosts cannot support multiple links. Devices that can support MPTCP are deployed in network by operators. The device located in home network is a CPE (provided by operator), and the one located in operators' network can be an existing network device (e.g., broadband network gateway) or an operator specially deployed device (e.g., BGW) that bundles multiple access links. This kind of research can be further divided into two classes: encapsulated approach and nonencapsulated approach.

HYA [4] is a network-layer encapsulated approach. HYA uses IP Tunnel established between CPE and BGW to bundle multiple access networks, as shown in Figure 2(a). An IP packet is appended to a new IP header and transmitted in IP Tunnel through access networks. The traffic is distributed by a packet scheduler either in a per-flow style or in a per-packet style. Relatively speaking, the latter is more flexible and can well utilize the available bandwidth of each access network. However, different access links have different network latencies. The packets of the same TCP flow may arrive at another side of the access network out of order, which impairs network performance greatly [7, 8]. It is difficult for a traffic scheduler to distribute packets to the appropriate links without the feedback from the transport layer. Though it is suggested that BGW should perform packet reordering in [4], it is challenging for a network-layer device to perform transport-layer function.

A Plain Transport Mode (PTM) of multipath TCP (MPTCP) [9–11] deployment model is proposed in [5]. It is aimed at promoting MPTCP deployment, but it can also be used to aggregate multiple access network bandwidth. It is a nonencapsulated approach. PTM, as shown in Figure 2(b), establishes MPTCP connections between two

FIGURE 2: Schematic illustrations of HYA, PTM, and MPTCP Tunnel. CPE and Proxy_C are MPTCP-supported devices located in the local network, while BGW, concentrator, and Proxy_I are those located in the Internet side. HYA uses IP Tunnel to bundle access networks, while PTM and MPTCP Tunnel use MPTCP connection to do it.

MPTCP-capable devices (CPE and MPTCP concentrator). Each TCP flow is assigned to one MPTCP connection. In order to share end hosts' information, an MPTCP option is designed to piggyback information of end hosts. And a binding entry must be maintained at CPE and concentrator to record the mapping of addresses. However, the establishment of multiple MPTCP connections introduces extra latencies and CPU overheads and decreases throughput. Besides, the changing of packet header violates the end-to-end semantics of the original TCP connection. Considering the different usage scenario of PTM and that of MPTCP Tunnel, we do not compare them in our evaluation.

Multipath networks [12, 13] constitute another access network aggregation scheme that also uses MPTCP. Proxy is adopted in multipath networks. All TCP traffic is first

intercepted by a modified home router equipped with the Linux MPTCP kernel, then forwarded to a server in the cloud over MPTCP connection, and finally sent to the destination in regular TCP. However, in our architecture, original packet headers are preserved and the TCP flows are tunneled in a single MPTCP connection, which maintains the end-to-end TCP semantics. However, we cannot find additional technological details of multipath networks, so the performance evaluation of it is not presented in this paper.

There are other substantial studies that address the bundling of access networks belonging to the second type—mobile-host-based mechanisms. In this class, bandwidth aggregation is carried out at the mobile end with multiple network interfaces [14–17], especially in wireless access networks. Nevertheless, most of the hosts connected to the

fixed access network do not have multiple-network-access capability. Considering the scenario discussed in this paper, the mobile-end-based studies are out of the scope of this paper.

3. Design Requirement

From the above discussion we know that both network-layer-based and MPTCP-based mechanisms have their flaws, respectively. A truly effective mechanism must mitigate the bandwidth shortage of fixed access network and tackle these problems in the meantime. Our architecture must meet such requirements.

3.1. Aggregating Bandwidth of Multiple Access Networks Efficiently and Less Costly. The approach used to aggregate the bandwidth of access networks must be efficient and less costly. Because of incapability of multipath support for most of the end hosts, network-assisted hybrid access architecture is an optimal choice. In order to make a balance between high QoS and low operating costs, cellular access network is used when the bandwidth of the fixed access network is insufficient.

3.2. Making No Changes to End Hosts. No matter whether they are servers or clients, the end hosts are inaccessible to operators. So any architecture cannot raise any requirement to end hosts. Moreover, the architecture must be adaptable to any type of user traffic and any number of TCP flows.

3.3. Addressing Reordering Problem. The network state is varying dynamically, especially in cellular network. The heterogeneity resulting from network states can cause packet arriving out of order at the other side of the access network, so a method is required to address the reordering problem.

3.4. Adapting to Multiple Traffic Types and TCP Flows. Different clients have distinct application demands. And different servers provide diverse services. Being an operator-based design, our architecture must adapt to multiple traffic types and TCP flows. Moreover, in the scenario discussed in this paper, proxies are intermediate nodes of a TCP connection. Contents in the packet header should not be changed when the packet is forwarded through them. A specially designed mechanism is needed to maintain the end-to-end semantics of TCP connections.

According to these requirements, we propose new architecture, *MPTCP Tunnel*, to aggregate the bandwidth of fixed and cellular access networks. The schematic illustration of MPTCP Tunnel is shown in Figure 2(c). The word *Tunnel* means that our architecture is an encapsulation mechanism. However, what distinguishes our architecture from HYA is the use of MPTCP, which is a transport-layer protocol with reliability assurance. In MPTCP Tunnel, we utilize MPTCP to bundle two access networks. Thus, MPTCP Tunnel can solve the reordering problem resulting from the heterogeneity of access networks naturally and adapt to the dynamic variation of network states. We propose to receive the whole IP packet (payload together with IP and TCP headers) when it arrives at the proxy. After being forwarded through the access network, the IP packet is retrieved and sent to the link layer

directly. In this process, the content of the packet header is unchanged, which maintains the end-to-end semantics of TCP connection. In our architecture, only one MPTCP connection is set up. Packet from any TCP flow is transmitted in this MPTCP connection in the same way, which means that MPTCP Tunnel can adapt to any traffic type.

4. Detailed Design

4.1. Overview of the Architecture. The components of MPTCP Tunnel are illustrated in Figure 3. (Our architecture can dispose the two-way TCP. For the sake of the simplicity, only a downlink TCP connection is shown in Figure 3. The solid line modules are used in downlink TCP, while the dashed ones are used in uplink TCP.) There are four components in our architecture. The two ends (server and client) are the source and destination of user traffic, and the two components in between are proxies.

The proxy located in the home network of a client is a CPE, and the one located at the Internet side may be a border gateway or specialized equipment, which can attach multiple access networks simultaneously. Three core modules are included in each proxy. The PtoS (packet to stream) module is responsible for processing the data to be sent to the access network, the StoP (stream to packet) module is responsible for processing the data received from the access network, and the MPTCP module handles the data transmission through multiple heterogeneous access networks.

Before the establishment of any TCP connection between the two end hosts, an MPTCP connection has been previously established between the two proxies with the help of MPTCP modules. No matter how many links exist between two proxies, only one MPTCP connection is established. The congestion control of the original TCP connection is still carried out by end hosts, and that of the MPTCP connection is controlled by MPTCP modules. Any packet loss or out-of-order among access networks is settled by MPTCP, which is transparent to the end hosts. The source and destination of user traffic are unaware of the existence of the MPTCP connection. Within the MPTCP connection, MPTCP packets are forwarded over the fixed or cellular access network according to the packet distribution policies and link states of access networks. The setting of link priority and adding or removing subflows are all managed by the MPTCP modules.

After being transmitted from a sender, the original IP packet is first received by the PtoS module. Then the PtoS module sends the IP packet to the MPTCP module in the style of data stream. The MPTCP module in the home proxy forwards the data stream through multiple access networks according to their network states. The MPTCP module in the remote proxy aggregates the data streams coming from different access networks and sends them to the StoP module. The StoP module parses the original IP packet from the data stream according to the format of the packet header. Eventually, StoP module sends the IP packet to the link layer directly.

In normal TCP, the packet header of a lower layer will be thrown away when forwarded to a higher layer. In PTM, a new MPTCP option is added in packet header to piggyback the destination address and a binding entry is added at the

FIGURE 3: Components of MPTCP Tunnel. MPTCP connection is established previously before the establishment of any TCP flow. IP packets with different color mean that they come from different TCP flows. They do not have to be differentiated when transmitted through access networks. After that, they are forwarded according to their destination address in the packet header.

FIGURE 4: Evolution of the packet header in MPTCP Tunnel. The header of the original packet is preserved together with the payload when an IP packet is sent through the access network. After leaving access network, the IP packet is forwarded to link layer directly.

CPE and the concentrator. According to the situation in our architecture, all of them violate the end-to-end semantics of a TCP connection. We avoid this violation by preserving the original IP packet header in our architecture. The evolution of packet header in our architecture is shown in Figure 4, which is different from that in normal TCP, HYA, and PTM. The header of the original packet is preserved and served as the payload of MPTCP connection. When arriving at another side of the access networks, the original IP packet is parsed and forwarded to link layer directly, that is, IP packet sent by end hosts, whose header content is not changed by middle nodes.

4.2. PtoS Module. When user traffic is sent by a sender, it is fragmented into data segments with the size of the minimum MSS of the links. Each data segment is then appended to a TCP, IP, and link layer headers, respectively, when it is forwarded through the transport layer, network layer, and link layer. The headers will be thrown away reversely when the packet is submitted to the application layer at the receiver. From the concept of end-to-end protocol semantics, the content of the packet header should not be changed before it arrives at the destination.

In MPTCP Tunnel user traffic is forwarded through the access network under the control of MPTCP modules in proxies. MPTCP is a transport-layer protocol, and it needs end-to-end information of its connection, but the proxy is just an intermediate node of a TCP connection. The changing of packet headers violates the end-to-end semantics of a TCP connection. A method to preserve the original end hosts' information is needed in this circumstance. PtoS module (depicted in Figure 5(a)) achieves this function.

4.2.1. Data Receiving. In our architecture, we preserve IP packet header by directly receiving the IP packet from the interface of the link layer. The Recv_Unit in PtoS module retrieves data packet directly from the link layer and stores the whole IP packet in receive buffer, which also completes the transformation of data style from data packet to data stream. The packet headers of network layer and transport layer are preserved because of the bypassing of corresponding layers. The data stream will be served as the data payload of MPTCP connection.

4.2.2. Data Batching. In our architecture, we use MPTCP to fully utilize the network resources of multiple access

FIGURE 5: The design of PtoS and StoP modules. (a) PtoS module: whole IP packet is received by PtoS module. When batching size reaches batching threshold, the batched data stream is sent to MPTCP module. (b) StoP module: after forwarded through access networks, IP packet is retrieved from data stream and sent to link layer directly.

networks. MPTCP is an end-to-end protocol, and it addresses traffic in the style of data stream. Before forwarded into access networks, the data stream will be fragmented into data packets each with a new TCP header attached with an MPTCP option as well as a new IP header when transferred through the transport layer and the network layer, respectively. If the size of the data stream is just equal to that of an original IP packet, the data stream will be fragmented into two packets, and the payload of that small packet is much smaller than that of the large one. The maximum size of an IP packet that can be transmitted without fragment varies for different transmission media, such as 1500 bytes for Ethernet (including the packet header and data). The maximum size of an MPTCP option is 28 bytes [10]. Then the maximum of the MPTCP data payload is 1432 bytes (1500 bytes minus 28 bytes of the MPTCP option and 40 bytes of the IP and TCP headers), which also equals the minimum MSS of the MPTCP connection. When a 1500-byte original IP packet is forwarded by the MPTCP module, the payload of that small packet is only 68 bytes. When there are many such original data packets arriving at the proxy, there will be many small packets to be transmitted through access networks. With the increase in the number of small packets, the goodput will decrease greatly and the CPU load will increase accordingly.

To tackle the aforementioned problem, a Batching Manager is deployed in the PtoS module, as shown in Figure 5(a). It monitors the queue length in the receive buffer. As long as the queue length reaches to a threshold named *Threshold_B_size*, the Batching Manager informs Send_Unit to send data. In our architecture, we set *Threshold_B_size* to be the integer multiple of the minimum data payload size of an MPTCP packet, which averts fragment for MPTCP packets when they are forwarded through access networks. To avoid wasting bandwidth or violating the ACK-clocking of TCP, the waiting time of the first segment of data stream in the receive buffer cannot be too long. That is to say, *Threshold_B_size* cannot be too big. On the other hand, there also exist light-load network conditions, where it takes a very long time for the queue length to reach the batch threshold. To avoid excessive latencies, we also introduce a timeout mechanism: if a batch with the size *Threshold_B_size* is not made up within the timeout threshold, all current packets will be directly sent to the MPTCP module without waiting for future packets.

4.3. MPTCP Module. The batched data stream will be forwarded to access networks by MPTCP module according

to their network states. Inspired by research [18, 19], data traffic is first delivered through fixed access network for the economy and reliability considerations. Cellular access network is used when the bandwidth of fixed access network is insufficient. In our architecture, the policies of selecting, adding, and removing subflows are made by operators and are out of the scope of this paper.

When an MPTCP data packet arrives at a proxy located at the other side of the access network, it will be first received by the MPTCP module. The possible out-of-order problem caused by heterogeneity of access networks is right settled by the MPTCP module with the congestion control and data sequence mapping mechanisms. The newly added TCP and IP headers of the in-sequence packet are thrown away and the payload is saved in a data stream style in the MPTCP receive buffer. The data stream will be sent to the StoP module subsequently.

4.4. StoP Module. The StoP module also has a Recv_Unit and a Send_Unit to receive and send data, respectively, as shown in Figure 5(b). Data stream received at the StoP module contains the header of the original packet. The Packet Parser in the StoP module has to parse out the original IP packet from the data stream. The IP packet is then forwarded to the link layer directly.

4.4.1. Data Receiving. What the MPTCP module sends to the StoP module is a piece of data stream. The Recv_Unit in the StoP module just receives the piece and saves it in the receive buffer of the StoP module.

4.4.2. IP Packet Parsing. In our architecture, the whole IP packet is preserved in the data stream. It already contains IP and TCP headers, so it needs neither to be clipped nor to be appended to additional IP and TCP headers. What we only need to do is to retrieve the original packets from the data stream. The Packet Parser in the StoP module is right in charge of this work.

Each IP packet has the same standard header structure. According to the format of the IP header, the Packet Parser gets the packet size from the *total_length* field. From the start of the data stream, Packet Parser fetches *total_length* of data each time.

There is another problem to address. The size of the data stream sent to the MPTCP module is *Threshold_B_size*. It is possible that the *Threshold_B_size* is not the integer multiple

FIGURE 6: Implementation of PtoS and StoP modules. A pair of receive and send threads is created in each module, respectively. A receive buffer is allocated to cache data in receive thread, as well as a send buffer in send thread. A shared ring buffer is allocated to share data between receive and send threads in each module.

of IP packet's size, so an IP packet may be clipped into two parts and fall in two adjacent pieces of the data stream. In this context, the Packet Parser cannot get a whole IP packet all at once. Therefore, the Packet Parser has to wait for the next piece of data stream to retrieve the remaining data of that IP packet.

4.4.3. Data Sending. Similar to disposing the packet size, the destination address and port number can also be retrieved from the data stream by the Packet Parser. After that, Send_Unit sends the IP packet directly to the interface of the link layer.

5. Implementation

We implement a prototype of our architecture with 982 lines of C code (source code is available at <https://github.com/dfshian/mptcp-tunnel>). Two main modules (PtoS and StoP) are included in the prototype, as is shown in Figures 6(a) and 6(b), respectively. Though more complicated than one thread in structure, a pair of threads is created in each module. The reason for it is that both of the two modules are deployed in each proxy and there are two I/O operations in each module: read and write. When data volume is huge, frequent reading or writing occupies much CPU time. If the read and write operations are carried out in one single thread, one operation must be blocked when the other is on. Two separate threads can speed up the overall reading and writing speed of system. Furthermore, two threads can utilize the advantage of multiple core of CPU.

In the receive thread, a receive buffer is used to cache packets received from the link layer. Correspondingly, a send buffer is used to cache packets that are available for sending. Considering the reliability and memory access speed, we allocate a shared ring buffer for data transmission between receive thread and send thread in each module. Data synchronization between two threads is a Producer-Consumer problem. Generally speaking, the memory access speed is much faster than the network speed, so the ring buffer only brings very little extra overhead to system and will not be the bottleneck.

Network communication is usually achieved with sockets [20] that abstract out the complicated processing of TCP/IP protocol stacks into a couple of socket APIs. There is a special socket, *raw socket* [21, 22], which allows access to lower layers of the TCP/IP protocol stack. In our architecture, we need to preserve the header of an IP packet. Therefore, we use raw socket to receive (send) IP packets from (to) the link layer directly.

In following subsections we elaborate the implementation of PtoS and StoP, respectively.

5.1. PtoS Module. Implementation of the PtoS module includes the data receiving from the link layer and sending to the MPTCP module. In this process, the data style is changed from packet to stream.

5.1.1. Receive Data from the Link Layer. We maintain the end-to-end protocol semantics by reserving the original IP packet header in our architecture. Therefore, we bypass the transport

and network layers and directly receive the IP packet from the interface of the link layer. We use raw socket to achieve this objective, as shown in step ① in Figure 6(a). The raw socket is created in following format:

```
socketfd = socket(PF_PACKET, SOCK_RAW,
                  htons(ETH_P_IP)). (1)
```

PF_PACKET means the data type we get from the link layer is a packet. ETH_P_IP means that we only receive an IP packet from the link layer. When the IP packet is received from link layer, it is appended to the receive buffer and saved as a piece of data stream, so the data style is transformed from packet to stream in this process.

5.1.2. Push Data to Ring Buffer. The data stream in the receive buffer is sent to ring buffer by the receiving thread, as shown in step ② in Figure 6(a). In order to avoid frequent read-write operations to the ring buffer, the data stream is pushed to the ring buffer only when the queue length is larger than a threshold. In our prototype, this threshold is set to the size of two IP packets. When the queue length of the receive buffer exceeds the threshold, the receive thread sends the data stream to the ring buffer with `push()` function.

5.1.3. Pull Data from the Ring Buffer. The send thread detects the state of the ring buffer periodically and retrieves the data stream from the ring buffer with `pull()` function (step ③ in Figure 6(a)). The retrieved data stream is then stored in the send buffer.

5.1.4. Send Data to the MPTCP Module. When the queue length in the send buffer is equal to or larger than *Threshold_B_size*, the send thread will send the data stream to the MPTCP module with `send()` function (step ④ in Figure 6(a)). In our architecture we set *Threshold_B_size* to the size of one MPTCP packet payload, which is the MTU of the MPTCP connection minus the sum of the IP header, TCP header, and MPTCP option.

5.2. StoP Module. Implementation of the StoP module includes the data receiving from the MPTCP module and sending to the link layer. In this process, the data style is changed from stream to packet. IP packets are parsed from the data stream in the StoP module.

5.2.1. Receive Data from MPTCP Model. The receive thread receives the data stream from MPTCP with `receive()` function and saves it in receive buffer (step ① in Figure 6(b)).

5.2.2. Push Data to the Ring Buffer. The data stream in the receive buffer is sent to the ring buffer by the receive thread (step ② in Figure 6(b)). The receive thread writes the data stream into the ring buffer with `push()` function.

5.2.3. Pull Data from the Ring Buffer. The process to pull data from the ring buffer in the StoP module (step ③ in Figure 6(b)) is the same as that in the PtoS module.

5.2.4. Send Data to the Link Layer. After data stream is pulled into the send buffer, IP packets need to be retrieved by the Packet Parser. If an intact IP packet is parsed, it will be sent to the link layer directly with raw socket (step ④ in Figure 6(b)). Raw socket used in data sending in the StoP module is different from that used in PtoS module. It is created in following format:

```
socketfd = socket(AF_INET, SOCK_RAW,
                  htons(IPPROTO_RAW)). (2)
```

The protocol type IPPROTO_RAW tells the interface of the link layer that the type of the packet to be sent is an IP packet, which can be sent directly.

6. Evaluation

In this section we show the following: (1) Our architecture is feasible and reliable in aggregating the bandwidth of multiple access networks. (2) Our architecture is more efficient than network-layer scheme under the dynamic variation of network conditions.

6.1. Experiment Setup. We build a testbed with 4 servers representing the server, two proxies, and the client, respectively, as shown in Figure 7. Each server is a Dell OptiPlex 7010 desktop running CentOS 7.2, equipped with a 4-core Intel Core i3-3240, 3.40 GHz CPU, 4 GB memory, 500 GB hard disk, and one Intel 82576 and one Intel 82576 Gigabit Ethernet NICs. We deploy the latest MPTCP implementation (v0.93) in GNU/Linux 4.9.60 in proxy servers.

In the following subsections, we will evaluate the performance of MPTCP Tunnel and compare it with HYA in terms of throughput under the varying network states.

6.2. Performance Comparison between Fixed Access Network, Cellular Access Network, and Our Architecture. In this subsection, we mainly consider three scenarios: (1) only one fixed access network, (2) only one cellular access network, (3) aggregated fixed and cellular access networks. Our MPTCP Tunnel is used in scenario 3. In our testbed, fixed access network is connected with one NIC. The bandwidth and RTT of fixed network is 10 Mbps and 50 ms, respectively. Cellular access network is connected with the other NIC, which is tethered with a 4G LTE USB modem. The RTT of LTE is about 50 ms. In each experiment, the TCP connection is a persistent flow. We compare the achieved throughput under the above three scenarios independently and separately. The results are shown in Figure 8.

All of throughput curves increase sharply at the early stage because of the empty link. After 4 seconds, the throughput of fixed access network keeps stable at the full capacity. Because of the stable bandwidth and negligible packet loss rate, the throughput variation of fixed access network is negligible. As for cellular access network, the announced bandwidth of 4G LTE is between 50 Mbps and 100 Mbps, but the actual bandwidth is only about 5 Mbps averagely in our measurement. And the throughput significantly fluctuates because of the varying bandwidth of the wireless access link.

FIGURE 7: Testbed setup. PtoS, StoP, and MPTCP modules are deployed in two proxies.

FIGURE 8: Throughput comparison between fixed access, cellular access network, and MPTCP Tunnel.

MPTCP Tunnel aggregates the bandwidth of the above two access networks. Therefore its throughput approximately equals the bandwidth sum of two access networks. The results prove that our architecture is feasible and efficient in aggregating the bandwidth of multiple heterogeneous access networks.

6.3. Throughput Comparison with the Network-Layer Scheme.

In order to compare the performance of HYA and MPTCP Tunnel under varying network conditions, we emulate fixed and cellular networks with two NICs. The bandwidth and RTT of cellular network are 5 Mbps and 50 ms, respectively, as is measured in the first experiment. The bandwidth of fixed network is 10 Mbps. We use `tc-tbf` and `tc-netem` tools in Linux to emulate the bandwidth variation, RTT, and packet loss rate of cellular network and fixed network, similar to the experimental method in [23]. TCP flows are generated by `iperf`. We transmit a 50 MB file in HYA and MPTCP Tunnel independently and compare their performance under the varying RTT and packet loss rate. Each experiment in this subsection is repeated for 100 times. Within HYA, we use JUGGLER [24, 25] to mitigate the impact of packet reordering on TCP performance. In JUGGLER, `of_timeout` is set to 10 ms, and `inseq_timeout` is set to 1 ms. The traffic scheduling policy in HYA is the proportional scheduling with different ratios.

The first experiment is the throughput comparison under varying RTT. In order to identify the effect of RTT variation on throughput, we keep the downlink bandwidth of cellular network stable. The packet loss rates are 0 and 0.01% in fixed access network and LTE access network, respectively. These parameters are set according to the previous researches [26, 27]. The RTT in cellular network varies from 30 ms to

FIGURE 9: Average and standard deviation of throughput of MPTCP Tunnel and HYA (with two traffic scheduling ratios) under varying RTT.

80 ms, increasing by 10 ms each time. The results are shown in Figure 9.

From Figure 9, MPTCP Tunnel achieves a better throughput under the RTT variation. As RTT increases, the throughputs of MPTCP Tunnel and HYA all decrease. However, the throughput decrease of MPTCP Tunnel is slight, and the throughput fluctuation of HYA is greater than that of MPTCP Tunnel. HYA can achieve the same or a slightly better throughput with MPTCP Tunnel only if the traffic scheduling can perfectly adapt to the proportion and variation of network bandwidth.

We also compare the throughput of HYA and MPTCP Tunnel in the circumstance of varying packet loss rate. The downlink bandwidth of cellular network is still 5 Mbps, and the packet loss rate is 0.01%. The RTTs are configured as default values. The packet loss rate in the fixed network varies from 0.01% to 20%.

It is shown clearly in Figure 10 that the throughputs of HYA and MPTCP Tunnel both decrease with the increasing of the packet loss rate. Throughput of HYA (ratio 1:1) decreases by 82.6% when packet loss rate increases from 0.01% to 5%, while that of MPTCP Tunnel decreases by 59.3%. Furthermore, MPTCP Tunnel just uses the better link and achieves the throughput no worse than the bandwidth of the better link in the circumstance of heavy packet loss. However, the throughput of HYA decreases greatly, even with a suitable traffic scheduling ratio.

From the aforementioned experiments, we can draw a conclusion that MPTCP Tunnel is more efficient than HYA in addressing the heterogeneity of access networks. HYA requires a specially designed traffic scheduling policy and reordering mechanism deployed on BGW to deal with the

FIGURE 10: Throughput comparison between MPTCP Tunnel and HYA (with two traffic scheduling ratios) under varying packet loss rate.

heterogeneity of access networks. Even so, the performance of HYA is heavily affected under the extreme network conditions. From above experiments, we can draw a conclusion that MPTCP Tunnel is more efficient than HYA in bundling access networks.

7. Discussion

Being a TCP extension, MPTCP owns all characteristics of TCP. Tunneling TCP in MPTCP may encounter the dilemma discussed in tunneling TCP in TCP [28, 29]. However, there is a distinct difference between our work and previous ones. In tunneling TCP in TCP, the congestion control concurrency problem mainly occurs when the in-between TCP connection is slow or unreliable [28]. However, in our scene the MPTCP connection is fast and reliable. Thus this problem is not serious. In our evaluation, we basically never encounter this problem in our architecture.

8. Conclusion

In this paper, we consider the bandwidth shortage of fixed access network in certain scenarios and design new architecture, *MPTCP Tunnel*, to aggregate the bandwidth of multiple heterogeneous access networks from the perspective of operators. MPTCP Tunnel leverages MPTCP to bundle multiple access networks and forwards original IP packets through access networks. In this way, MPTCP Tunnel solves the packet reordering problem and maintains the end-to-end TCP semantics. We implement a prototype and build a testbed to evaluate the performance of MPTCP Tunnel, taking the recent HYA scheme as reference. The experimental results show that MPTCP Tunnel can indeed aggregate the bandwidth of fixed and cellular networks and achieve up to 80% higher throughput than that of HYA. Furthermore, MPTCP Tunnel is also more adaptable to the increased heterogeneity of multiple access networks than HYA.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is supported in part by 863 Plan under Grant no. 2015AA020101, NSFC under Grant no. 61225011, and Suzhou-Tsinghua Special Project for Leading Innovation.

References

- [1] "Fixed network data traffic worldwide: forecasts and analysis 2015-2020, 2016," <https://www.analysismason.com/Research/Content/Reports/Fixed-data-forecast-May2015-RDTW0/>.
- [2] "Bell labs metro network traffic growth: An architecture impact study, 2013," <http://www.doc88.com/p-0953164628991.html>.
- [3] "White paper: Optimizing bandwidth: why size doesnot always matter, 2014," <https://insight.nokia.com/optical-transport-networks-and-bandwidth-demand>.
- [4] N. Leymann, C. Heidemann, M. Wesserman, L. Xue, and M. Zhang, "Hybrid access network architecture," *Network Working Group, draft-lhwxz-hybrid-access-network-architecture-02 (work in process)*, 2015.
- [5] M. Boucadair, C. Jacquenet, D. Behaghel et al., "An mptcp option for network-assisted mptcp deployments: Plain transport mode," *Network Working Group*, 2016.
- [6] Y.-C. Chen and D. Towsley, "On bufferbloat and delay analysis of multipath TCP in wireless networks," in *Proceedings of the 2014 IFIP Networking Conference, IFIP Networking 2014*, pp. 1-9, IEEE, Trondheim, Norway, June 2014.
- [7] K.-C. Leung, V. O. Li, and D. Yang, "An overview of packet reordering in transmission control protocol (TCP): Problems, solutions, and challenges," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 4, pp. 522-535, 2007.
- [8] A. Gurtov and S. Floyd, "Modeling wireless links for transport protocols," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 85-96, 2004.
- [9] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural Guidelines for Multipath TCP Development," RFC Editor RFC6182, 2011.
- [10] "Tcp extensions for multipath operation with multiple addresses, 2013," <https://tools.ietf.org/html/rfc6824.html>.
- [11] C. Raiciu, C. Paasch, S. Barre et al., "How hard can it be? designing and implementing a deployable multipath tcp," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pp. 29-29, USENIX Association, San Jose, CA, USA, 2012.
- [12] Mptcp., http://blog.multipath-tcp.org/blog/html/2015/12/25/commercial_usage_of_multipath_tcp.html.
- [13] Multipath networks, <https://web.archive.org/web/20150325055000/http://www.multipathnetworks.com>.
- [14] K.-H. Kim and K. G. Shin, "Improving TCP performance over wireless networks with collaborative multi-homed mobile hosts," in *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services, MobiSys 2005*, pp. 107-120, ACM, New York, NY, USA, June 2005.
- [15] K. Evensen, D. Kaspar, C. Griwodz, P. Halvorsen, A. Hansen, and P. Engelstad, "Improving the performance of quality-adaptive video streaming over multiple heterogeneous access networks," in *Proceedings of the 2nd Annual ACM Conference on Multimedia Systems (MMSys '11)*, pp. 57-68, ACM, Santa Clara, Calif, USA, February 2011.
- [16] A. Popovska Avramova and L. Dittmann, "Modeling of bandwidth aggregation over heterogeneous wireless access networks," *OPNETWORK*, 2012.

- [17] K. Habak, K. A. Harras, and M. Youssef, "Bandwidth aggregation techniques in heterogeneous multi-homed devices: A survey," *Computer Networks*, vol. 92, pp. 168–188, 2015.
- [18] B. Arzani, A. Gurney, S. Cheng, R. Guerin, and B. T. Loo, "Deconstructing MPTCP performance," in *Proceedings of the 22nd IEEE International Conference on Network Protocols, ICNP 2014*, pp. 269–274, IEEE, Raleigh, NC, USA, October 2014.
- [19] S.-Y. Park, C. Joo, Y. Park, and S. Bank, "Impact of traffic splitting on the delay performance of MPTCP," in *Proceedings of the 2014 1st IEEE International Conference on Communications, ICC 2014*, pp. 1204–1209, IEEE, Sydney, NSW, Australia, June 2014.
- [20] "Socket - linux socket interface, 2016," http://www.man7.org/linux/man-pages/man7/socket.7.html#top_of_page.
- [21] "Raw - linux ipv4 raw sockets, 2016," <https://linux.die.net/man/7/raw>.
- [22] "Tcp/ip raw sockets, 2016," [https://msdn.microsoft.com/en-us/library/windows/desktop/ms740548\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms740548(v=vs.85).aspx).
- [23] Y. Cui, Z. Lai, and N. Dai, "A first look at mobile cloud storage services: Architecture, experimentation, and challenges," *IEEE Network*, vol. 30, no. 4, pp. 16–21, 2016.
- [24] Y. Geng, V. Jeyakumar, A. Kabbani, and M. Alizadeh, "JUGGLER: A practical reordering resilient network stack for datacenters," in *Proceedings of the 11th European Conference on Computer Systems, EuroSys 2016*, p. 20, ACM, New York, NY, USA, April 2016.
- [25] Y. Geng, "Juggler," <https://github.com/gengyl08/juggler>.
- [26] "G. T. 36.213, 3rd generation partnership project; technical specification group radio access network; evolved universal terrestrial radio access (e-utra); physical layer procedures, 2013. Release 14, version 14.1.0," <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2427>.
- [27] N. Becker, A. Rizk, and M. Fidler, "A measurement study on the application-level performance of LTE," in *Proceedings of the 2014 IFIP Networking Conference, IFIP Networking 2014*, pp. 1–9, IEEE, Trondheim, Norway, June 2014.
- [28] O. Titz, "Why tcp over tcp is a bad idea," <http://sites.inka.de/bigred/devel/tcp-tcp.html>.
- [29] O. Honda, H. Ohsaki, M. Imase, M. Ishizuka, and J. Murayama, "Understanding TCP over TCP: Effects of TCP tunneling on end-to-end throughput and latency," in *Proceedings of the Performance, Quality of Service, and Control of Next-Generation Communication and Sensor Networks III*, vol. 6011, p. 60110H, SPIE, Boston, MA, USA, October 2005.