Multipath IP Routing on End Devices: Motivation, Design, and Performance

Liyang Sun*, Guibin Tian*, Guanyu Zhu*, Yong Liu*, Hang Shi†, and David Dai†
*Electrical & Computer Engineering, New York University, Brooklyn, NY, 11201, USA
† Huawei Technology, Santa Clara, CA, 95050, USA

Abstract—**Most end devices are now equipped with multiple network interfaces. Applications can exploit all available interfaces and benefit from multipath transmission. Recently Multipath TCP (MPTCP) was proposed to implement multipath transmission at the transport layer and has attracted lots of attention from academia and industry. However, MPTCP only supports TCP-based applications and its multipath routing flexibility is limited. In this paper, we investigate the possibility of orchestrating multipath transmission from the network layer of end devices, and develop a Multipath IP (MPIP) design consisting of signaling, session and path management, multipath routing, and NAT traversal. We implement MPIP in Linux and Android kernels. Through controlled lab experiments and Internet experiments, we demonstrate that MPIP can effectively achieve multipath gains at the network layer. It not only supports the legacy TCP and UDP protocols, but also works seamlessly with MPTCP. By facilitating user-defined customized routing, MPIP can route traffic from competing applications in a coordinated fashion to maximize the aggregate user Quality-of-Experience.**

I. Introduction

Contemporary end devices are normally equipped with multiple network interfaces, ranging from datacenter blade servers to user laptops and handheld smart devices. Exploiting all available interfaces, applications can adopt multipath transmissions to achieve higher and smoother aggregate throughput, resilience to traffic variations and failures on individual paths, and seamless transition between different networks. While each application can implement its own multipath transmission at the application layer, it is more desirable to provide multipath transmission services from the lower network protocol stack so that all applications can benefit. Recently, Multipath TCP (MPTCP) has been proposed and attracted lots of attention from academia and industry [1], [2], [3], [4], [5]. MPTCP allows all TCP-based applications enjoy the multipath gain in a *transparent* fashion. However, UDP-based applications cannot benefit from multipath transmissions.

In this paper, we share our experience of orchestrating multipath transmission from the network layer on end devices, and present a complete design of Multipath IP Transmission (MPIP). There are several advantages of implementing multipath transmission at the network layer:

Broader Coverage. MPIP can transmit IP packets generated by any TCP or UDP based application. Being transparent to the upper layers, MPIP can benefit all user applications without changing the application and transport layer protocols.

ISBN 978-3-903176-08-9 © 2018 IFIP

Better View and Coordination. The network layer can directly measure network status and promptly capture various dynamic events, such as interface and network changes. Since all application traffic go through the network layer, MPIP can adjust the transmission strategies for all applications in a coordinated fashion to maximally satisfy the diverse application and user needs.

More Flexible Routing. With MPTCP, traffic allocated to a path is determined by the rate achieved by the TCP subflow on that path, i.e., routing is simply determined by congestion control along multiple paths. This is too rigid and limited for applications with different throughput and delay requirements, and users with different resource and economic constraints. MPIP instead can implement any customized multipath routing.

Lower Complexity. MPIP can eliminate redundant network probings and routing adjustments attempted by individual applications and sessions. From the implementation point of view, similar to MPTCP, MPIP only requires changes on end devices. MPTCP has to work with the complexity resulted from the stateful TCP implementation. The legacy IP protocol is stateless and its implementation is much simpler than the legacy TCP. This leaves more design space for MPIP.

Meanwhile, MPIP also faces additional challenges. First of all, due to the stateless nature of IP, there is no existing session and path management mechanisms at network layer. Secondly, to work with multiple paths, MPIP constantly needs feedbacks about the availability and performance of each path. However, the legacy IP does not provide end-to-end feedbacks. Thirdly, various middle-boxes, e.g., NAT routers, are *by-no-means transparent*. They change and verify IP and TCP headers, and drop packets which they believe are "unorthodox" according to the legacy TCP/IP protocol. Multipath transmission unavoidably leads to out-of-order packet delivery. This will cause problem for running legacy TCP over MPIP. Finally, MPIP design and implementation should minimize the overhead and complexity added to the network layer. We address those challenges in our MPIP design and implementation. The contribution of our work is three-fold:

1) We develop a complete design to implement multipath transmission at the network layer, consisting of signaling, session and path management, multipath IP source routing, and NAT traversal. Our MPIP design not only can be used by the legacy TCP and UDP protocols, but also works seamlessly with MPTCP.

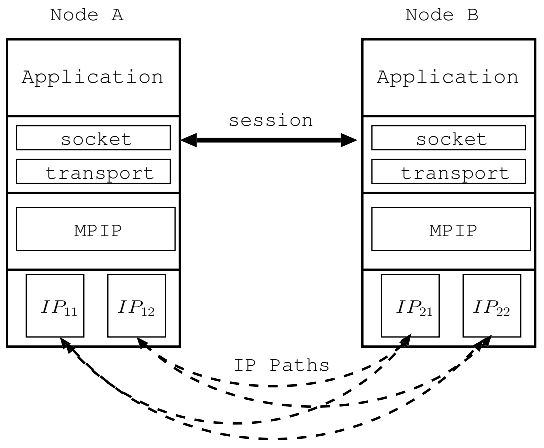


Fig. 1. Example of MPIP Transmission

2) MPIP supports diverse multipath routing strategies. For *all-paths* mode, we design a delay-based routing algorithm for MPIP to balance the loads of available paths. We also develop a user-defined multipath routing framework, through which customized routing strategies, such as *selected-paths* and *single-path*, can be realized by MPIP to satisfy diverse application/user needs.
3) We implement MPIP in Linux and Android kernels. We evaluate its performance using controlled lab experiments and Internet experiments. We demonstrate that MPIP can transparently achieve various multipath gains at the network layer. It works seamlessly with legacy transport layer protocols and popular applications. It can significantly improve user Quality-of-Experience (QoE) using easily configurable multipath routing strategies.

The rest of the paper is organized as follows. The semantics of MPIP is presented in Section II. The complete MPIP design is developed in Section III. Special issues related to TCP are addressed in Section IV. In Section V, we report the experimental results. Related work is summarized in Section VI. The paper is concluded in Section VII.

II. Semantics

MPIP works at the network layer on end devices. The basic building blocks are: *Node, Session, and Path.*

- *Node* refers to an end device with potentially multiple network interfaces, each of which gets assigned with a private or public IP address. MPIP also works with nodes with single network interface.
- *Session* is a transport layer flow between two nodes served by MPIP. A session is established at the transport layer, using the legacy TCP or UDP protocol, or even the new MPTCP protocol.
- *Path* is an end-to-end IP route available for a session. For each session, MPIP can use any interface on one node to transmit packets to any interface on the other node. If the two nodes have m and n interfaces respectively, the number of possible paths is mn.

With the legacy IP, each session is associated with only one IP (interface) and one port number on each node. The routing decision is based on destination IP address. MPIP employs customized *session-based* routing, and transmits packets of each session using any combination of the available paths. For the example in Figure 1, node A and node B are MPIP-enabled. They use the legacy application layer and transport layer. Each node has two interfaces (and the associated IP addresses). There are four end-to-end IP paths, as illustrated in Figure 1. When an application on node A opens a TCP/UDP connection to node B, MPIP will treat this connection as a new session. For each packet going from A to B, MPIP will choose one of the four available paths to send it out. To do that, MPIP will change the source and destination IP addresses as well as the port numbers of the packet so that it can be forwarded to the corresponding interface of the chosen path on node B. When node B receives the packet, it will first check which session it belongs to, then modify the IP address and port number back to the original values of the session. Finally, the packet will be passed to the corresponding TCP/UDP socket. The whole process is transparent to TCP/UDP session. If MPIP can simultaneously utilize the four paths by dispatching different packets to different paths, TCP /UDP throughput can be improved. Also the session can work normally as long as one path is available. Consequently, a TCP/UDP session will not be interrupted even if the default interfaces assigned to the session by the OS are disconnected. This makes handovers between different networks seamless and transparent to the transport and application layers. In general, MPIP routes packets from one session using several modes: 1) *all-paths mode:* packets are dispatched concurrently to all the available paths. Each packet will be transmitted along one of the paths. *MPIP Routing* determines the traffic splitting ratios among paths; 2) *selected-paths mode:* packets are routed on a subset of paths that meet the requirements of the application. Selected-paths mode avoids the inclusion of bad paths that will drag down the application performance. Path selection is application-specific and can be adapted by MPIP based on both application and network dynamics; 3) *single-path mode:* at any time, packets are only routed over one selected path, which can change during the course of the session. MPIP will handle seamless handover between paths, without interrupting the session. Single-path mode eliminates path quality disparity, such as out-of-order packet delivery, by sacrificing the throughput gain; 4) *protected-path mode:* a mission-critical packet is simultaneously transmitted on multiple paths. The receiver will pass the first arrived copy to the upper layer and discard the subsequent redundant copies. It sacrifices bandwidth for resilience.

III. MPIP Design

A. Workflow of Sending/Receiving Packets

Before diving into the design details, we present the MPIP workflow in Figure 2. When an outbound packet arrives at network layer from transportation layer, given the destination IP address and port number in header, MPIP checks whether

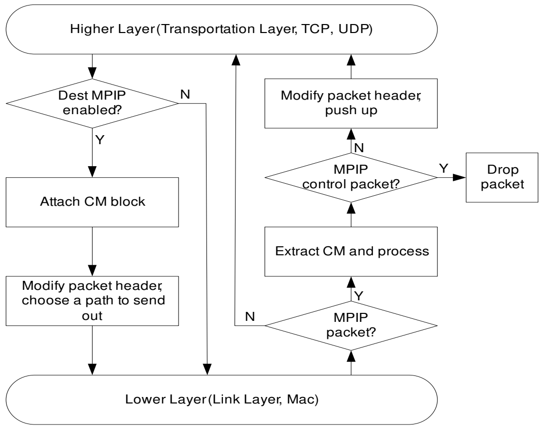

Higher Layer(Transportation Layer, TCP, UDP)
Dest MPIP enabled?
N
Y
Attach CM block
Modify packet header, choose a path to send out
Lower Layer(Link Layer, Mac)
Modify packet header, push up
N
MPIP control packet?
Y
Drop packet
Extract CM and process
Y
N
MPIP packet?

Fig. 2. MPIP Work Flow of Sending and Receiving Packets.

the destination node is MPIP enabled. If not, the packet will be processed by the regular IP stack and sent to the data link layer. If the destination is MPIP-enabled, MPIP will append a MPIP control message (CM) block to the end of the packet, change the IP and port addresses in packet header so that it will be sent to a chosen IP path. When receiving an inbound packet, MPIP processes the CM block to find the transport layer socket that the packet belongs to. Then MPIP reverts the IP and port addresses in packet header to the original values before pushing the packet to the transport layer. The major MPIP design components are: *Signaling Channel*, *Handshake*, *Session Management*, *Path Management*, *MPIP Routing*, and *NAT Traversal*.

B. Signaling Channel

TABLE I
CONTROL MESSAGE BLOCK

Source Node ID	*Session ID*	*Local IP Address List*	*CM Flags*
Path ID	*Feedback Path ID*	*Packet Timestamp*	*Path Delay*

MPIP needs realtime information about the availability and performance of end-to-end paths. Due to its connectionless design, legacy IP protocol doesn't have its built-in end-to-end feedback channel. We need a signaling channel for MPIP. Instead of transmitting extra signaling packets, we piggyback MPIP control information to each MPIP packet. For each packet sent out by MPIP, we add an additional control message (CM) data block at the end of user data. The size of the CM block is 25 bytes, a small overhead for typical data packets of $1000+$ bytes. Considering the throughput gain and robustness brought by MPIP, the overhead of CM block is well acceptable. Packet size may exceed the link MTU after attaching the CM block. We force the transport layer to reduce the size of each segment, e.g. decreasing the MSS value for TCP connection, to make sure the CM block fits within the MTU limit. The information contained in a CM block of a packet is shown in Table I.

Source Node ID is a globally unique ID of the sending node of this packet. Since each node has multiple interfaces, and their IP addresses may change over time, to have a semi-static node ID, we use the MAC address of a NIC (preferable more static ones) on the node to be its ID.

Local IP Address List carries all local IP addresses on the sending node. This list will be used to construct MPIP paths.

CM Flags encodes the MPIP functionality of the packet. With different values of *CM Flags*, different actions will be operated when the packet is received.

Other fields will be explained in the following sections.

C. Handshake and Session Management

As an extension of IP, MPIP needs to be backward compatible. To take advantage of MPIP, both end nodes of a session need to be MPIP-enabled. Locally, every MPIP-enabled node maintains a table to record the availability of MPIP on remote nodes. A node can query the MPIP availability of a remote node by sending out a MPIP packet with $Flags\_Enable$ in CM. If the remote node is MPIP-enabled, it will send back confirmation. Both nodes will update their MPIP availability table accordingly. Please refer to our technical report [6] for the detailed handshake process, After the MPIP handshake, a node can start to learn the interfaces available on each MPIP-enabled remote node. Each node maintains a node ID to IP address and port number mapping table. Every time a MPIP packet is received, the receiver extracts the sender's node ID from the packet's CM block, and IP address and port number from the packet header. The three tuple is then written into the mapping table.

MPIP conducts session-based routing. Session management takes care of the addition and removal of TCP and UDP sessions. At the transport layer, each session is identified by the traditional 5-tuple: source and destination IP addresses and port numbers, and protocol type. Since MPIP can transmit a packet of a session using source and destination IP address/port numbers different from the session's original ones, we can no longer use IP addresses/port numbers to associate a MPIP packet with a transport layer session. Instead, we will use session ID and node ID carried in the CM block to identify the session of a MPIP packet. We need a table to correlate the two different session mapping schemes employed by MPIP and the legacy transport layer. This is achieved through the session information table, as in Table II. The table maintains one entry for each session to each remote node. For each entry, the socket information, namely IP addresses and port numbers, are the original ones from the transport layer.

After the MPIP availability handshake has been successfully completed, when sending out a packet, the sender checks Table II to see whether a proper session entry has been generated. If not, MPIP generates a new session ID and adds a new entry to Table II. After this, all packets belong to the session will carry the session's ID in its CM block. On the receiver end, whenever a MPIP packet is received, the receiver extracts the source node ID and session ID from its CM block. If there is no entry found in its session information table, it

TABLE II
SESSION INFORMATION TABLE

Dest. Node ID	Session ID	Source IP	Source Port	Destination IP	Destination Port	Protocol Type	Next Sequence No	Update Time
ID_1	SID_1	SIP_1	$SPORT_1$	DIP_1	$DPORT_1$	TCP	S_1	T_1
ID_1	SID_2	SIP_1	$SPORT_2$	DIP_1	$DPORT_2$	UDP	0	T_2
ID_2	SID_1	SIP_2	$SPORT_3$	DIP_2	$DPORT_3$	TCP	S_2	T_3
ID_2	SID_2	SIP_2	$SPORT_4$	DIP_2	$DPORT_4$	UDP	0	T_4

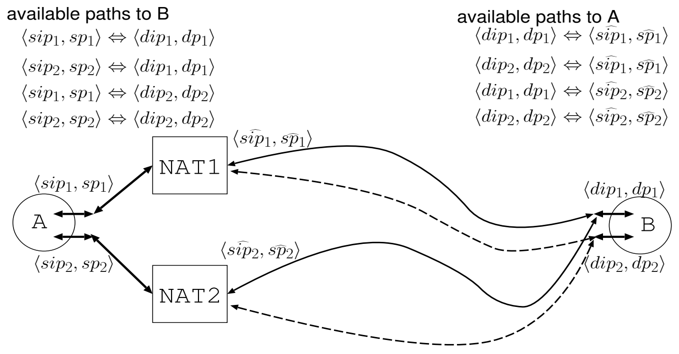

available paths to B
$\langle sip_1, sp_1\rangle \Leftrightarrow \langle dip_1, dp_1\rangle$
$\langle sip_2, sp_2\rangle \Leftrightarrow \langle dip_1, dp_1\rangle$
$\langle sip_1, sp_1\rangle \Leftrightarrow \langle dip_2, dp_2\rangle$
$\langle sip_2, sp_2\rangle \Leftrightarrow \langle dip_2, dp_2\rangle$
available paths to A
$\langle dip_1, dp_1\rangle \Leftrightarrow \langle \widehat{sip}_1, \widehat{sp}_1\rangle$
$\langle dip_2, dp_2\rangle \Leftrightarrow \langle \widehat{sip}_1, \widehat{sp}_1\rangle$
$\langle dip_1, dp_1\rangle \Leftrightarrow \langle \widehat{sip}_2, \widehat{sp}_2\rangle$
$\langle dip_2, dp_2\rangle \Leftrightarrow \langle \widehat{sip}_2, \widehat{sp}_2\rangle$
$\langle \widehat{sip}_1, \widehat{sp}_1\rangle$
NAT1
$\langle sip_1, sp_1\rangle$
A
$\langle sip_2, sp_2\rangle$
$\langle \widehat{sip}_2, \widehat{sp}_2\rangle$
NAT2
$\langle dip_1, dp_1\rangle$
B
$\langle dip_2, dp_2\rangle$

Fig. 3. MPIP Path Establishment with NAT

will generate a new entry and populate it with the source node ID, session ID, and socket information carried in the packet header, with swapped source and destination IP/port addresses. This will make sure that both sides of the same session use the same session ID. Removal of a session is done by expiration based on the session's *Update Time* in Table II. The column *Next Sequence No* is used for TCP out-of-order process which will be explained in Section IV-B.

D. Path Management

After a session is registered with MPIP, the next step is to explore all the available paths for the session. One simple solution is to have each node send their local IP addresses to the other end using the *Local Address List* in CM block. Then any pair of IP addresses on the two ends can be used as a path for MPIP transmission. However, this only works if all interfaces on both ends have public IP addresses. If one node is behind a NAT, its local IP addresses cannot be used directly to establish IP paths. To solve this problem, we have to identify paths using a combination of IP address and port number on both ends. Consequently, the path management has to be done for each session individually.

1) Establishment: MPIP maintains a path information table on each node, as in Table III, to record the available paths for each session. Each entry contains the ID of the remote node and the session ID. Each path is allocated with a path ID, which is unique on the local node. The source and destination IP and port addresses are the addresses carried in the header of MPIP packet, NOT necessarily the same as those allocated to the session at the transport layer.

Given m and n interfaces at each end node, there are totally mn possible paths. After the MPIP handshake, each node tries to send out packets from each of its local interfaces to each of the known interface on the remote node. If a packet with a certain combination of source and destination IP/port addresses can get through, the node will add the path to path information table. Let's explain the process through the example in Figure 3. Node A initiates a session with node B. The IP and port addresses allocated to the session at the transport layer are $\langle sip_1, sp_1\rangle$ and $\langle dip_1, dp_1\rangle$ on A and B respectively. Without loss of generality, let's assume the session can be established correctly with legacy IP. Then on both ends, MPIP records the new session, and adds the default path between $\langle sip_1, sp_1\rangle$ and $\langle dip_1, dp_1\rangle$ for the session in Table III. Since A knows B is MPIP-enabled, it also tries to send the same packet from its other local interface with IP address sip_2 by changing its source addresses to $\langle sip_2, sp_2\rangle$. When B receives the packet, possibly due to NAT, the source IP and port addresses in the packet might be different from $\langle sip_2, sp_2\rangle$, say $\langle \widehat{sip_2}, \widehat{sp_2}\rangle$. Then B examines the *Source Node ID* and *Session ID* in the packet's CM block, it knows this is a MPIP transmission for the same session but from a different interface. B adds a new path with destination address of $\langle \widehat{sip_2}, \widehat{sp_2}\rangle$ in its path information table. Now B will also send back packets to A's second interface, using destination addresses $\langle \widehat{sip_2}, \widehat{sp_2}\rangle$. When A receives the packet, it confirms the connectivity of its local path between $\langle sip_2, sp_2\rangle$ and $\langle dip_1, dp_1\rangle$, and adds it to its path information table. Similarly, if B has another interface with public address dip_2, A will obtain the new address from the *Local Address List* in the CM block of packets from B to A. Then A can establish more IP paths to this new address using a similar process.

2) Monitoring: To facilitate path selection, MPIP continuously monitors the performance of active paths. Given that packet losses in the current Internet are rare, we mainly focus on path delay in our current design. Due to asymmetric routing and unequal congestion levels along two directions of the same path, instead of measuring the round-trip delay of a path, we measure the one-way path delay to infer the path quality on each direction. When node A sends out a packet, it chooses a path from Table III and sets *Packet Timestamp* with its local system time T_1. After node B receives this packet, it calculates the one-way delay for the path as $T_2 - T_1$, where T_2 is B's local time when receiving the packet. In practice, the absolute value of path delay calculated here isn't the real delay value because of the clocks on node A and B are not synchronized. But our path selection algorithms depend on the relative ordering of path delays, instead of their absolute values. Clock difference between nodes has little impact. B then sends back the path delay information in the CM block

TABLE III
PATH INFORMATION TABLE

Dest Node ID	Session ID	Path ID	Src IP	Src Port	Dest IP	Dest Port	Minimum Path Delay	Real-Time Path Delay	Real-Time Queuing Delay	Maximum Queuing Delay	Path Weight
ID	SID_1	PID_{11}	sip_1	sp_1	dip_1	dp_1	D_{min11}	D_{11}	Q_{11}	Q_{max11}	W_{11}
ID	SID_1	PID_{12}	sip_2	sp_2	dip_1	dp_1	D_{min12}	D_{12}	Q_{12}	Q_{max12}	W_{12}
ID	SID_2	PID_{21}	sip_1	sp_1	dip_2	dp_2	D_{min21}	D_{21}	Q_{21}	Q_{max21}	W_{21}
ID	SID_2	PID_{22}	sip_2	sp_2	dip_2	dp_2	D_{min22}	D_{22}	Q_{22}	Q_{max22}	W_{22}

of the next packet going back to A, which records the path delay value into the column *Real-Time Path Delay* in Table III. Path delay values are smoothed using a simple moving average algorithm. More details can be found in our technical report [6]

3) Dynamic Path Management: MPIP supports dynamic addition and removal of paths from Table III. When IP address change happens on one node, it sets $Flags\_IP\_Change$ in the CM block of its next outgoing packet. After receiving a packet with this flag, the receiver knows that IP address on the sender has changed, it removes all path entries related to the changed IP address in Table III. Meanwhile, the entry for this session in Table II remains unchanged. The path that sends out the IP change notification will be added back to the aforementioned tables as the only path of the session. Also, the sender does the same reset for this session. After all these resets, there is only one path left for this session, all the other available paths will be added back through the procedure in Section III-D1. Similarly, when a new interface becomes available, new IP paths from it can be added using the the mechanism in Section III-D1. Table III should be updated continuously on both sides. The updates are piggybacked on MPIP packets. For sessions with one-way traffic, such as some UDP sessions, a periodical heartbeat mechanism is introduced to keep Table III fresh. More details can be found in our technical report [6].

E. Multipath IP Source Routing

Given all paths available for a session, every time one node needs to send out a packet, it chooses the most suitable path from Table III. MPIP offers different routing strategies to satisfy the diverse needs of applications.

1) All-paths Mode: Many applications, e.g., web, file transfer, and video streaming, can benefit from high-throughput transmissions. MPIP can concurrently transmit packets along multiple paths to achieve higher throughput than the traditional single path routing. Since MPIP works under rate control schemes from transport and application layers, it will be redundant and possibly conflicting to implement fine-grained rate control for each MPIP path at the network layer. Instead, the main design goal of MPIP routing is to balance load among concurrent paths using end-to-end path delay feedback and probabilistic packet dispatching algorithm. As in Table III, we maintain a *Path Weight (W)* for each active path. Each packet will be dispatched to a path k with the probability $P(k)$, which is calculated as:

$$P(k) = \frac{W_k}{\sum_{i=1}^{N} W_i}. \tag{1}$$

We use realtime one-way path delay to dynamically update path weights. End-to-end path delay consists of propagation delay, transmission delay, processing and queueing delay. While propagation delay and transmission delay are mostly constant, processing and queue delay are time-varying and increase with congestion level. We maintain the minimum path delay to represent the constant portion of end-to-end path delay, and use the difference between real-time and minimum delay to infer the queuing delay, which reflects the congestion level along the path. We then adjust the weight of each path using the real-time queuing delay. When a new delay sample D is received, the other three delay metrics are updated:

1) *Minimum Path Delay:* $D_{min} = \min\{D_{min}, D\}$;
2) *Real-Time Queuing Delay:* $Q = D - D_{min}$;
3) *Maximum Queuing Delay:* $Q_{max} = \max\{Q_{max}, Q\}$.

We adjust the weights of all paths together based on their queueing delay variations as in Algorithm 1. N is the number

Algorithm 1 Path Weight Adjustment.

```
1:  $Q_{avg} = \frac{\sum_{i=1}^{N} Q_i}{N}$; //average delay among all paths
2:  if $Q_i \leq Q_{avg}$ then
3:      $W_i = W_i + S$; //increase weight for low delay path
4:      if $W_i > 1000$ then
5:          $W_i = 1000$; //upper bound for path weight
6:      end if
7:  else
8:      $W_i = W_i - S$; //decrease weight for high delay path
9:      if $W_i < 1$ then
10:         $W_i = 1$; //lower bound for path weight
11:     end if
12: end if
13: return ;
```

of paths that belong to one session, Q_i and W_i are queuing delay and weight of path i, and S is the adjustment granularity. Initially, every path has the same path weight of $\frac{1000}{N}$. In each iteration, the path weight increases or decreases by S based on whether its queuing delay is higher or lower than the average delay. The maximum weight is 1000, and the minimum is 1. This way, we keep all live paths in consideration. Heavily congested paths will not be completely eliminated. Instead they will have the minimum weight, and their weights will be increased after congestion is relieved. Algorithm 1 is executed periodically, the length of each period is defined as a configurable system parameter T.

2) User-defined Multipath Routing: Not all applications take throughput as the first priority. To address the diverse needs of applications, we design MPIP to support user-defined routing schemes, including *selected-paths*, *single-path* and *protected-path*. Users can inform MPIP of their desired multi-path routing policies by configuring a routing table as illustrated in Table IV. Each line of the table is a customized routing rule for outgoing packets. Each rule matches a set of packets and the routing priority for the matched packets. Packet matching is done using destination IP address, port number, protocol, and the range of packet length. We currently define two types of routing priorities: *throughput-first* T_f, and *responsiveness first* R_f. Outgoing packets with T_f priority will be dispatched to available paths using the *all-paths* mode presented in Section III-E1. Outgoing packets with R_f priority will always be sent to path with the lowest delay using the *single-path* mode. For example, based on the first row of Table IV, for any TCP connection with destination port 22 (ssh session), if the packet length is smaller than 200 bytes, the packet will be forward to the lowest delay path. The second row defines that all UDP packets going to a remote host with packet size larger than 200 bytes should be forwarded using *all-paths* mode. The third row specifies that for a UDP packet going to the same remote host, but a different port number, if the packet size is less than 500, it will be forwarded to the lowest delay path instead. We will extend this basic framework to incorporate more flexible and more user-friendly packet matching rules and more diverse routing policies with finer granularity in our future work.

TABLE IV
USER-DEFINED MULTIPATH ROUTING TABLE

IP Address	Port Number	Protocol Type	Start Size	End Size	Routing Priority
$*$	22	TCP	0	200	R_f
192.168.1.2	5222	UDP	200	$*$	T_f
192.168.1.2	5221	UDP	0	500	R_f

IV. TCP-RELATED ISSUES

By deviating from the default single-path transmission, MPIP also brings some new issues for the upper layer protocols, especially TCP, such as NAT checking and out-of-order packet delivery. It is also intriguing to explore the co-existence of MPIP with multi-path transmissions at upper layers, such as MPTCP. We now present solutions to TCP-related issues.

A. NAT Checking

Based on our experiments and other studies, e.g. [1], NAT devices are by no means transparent, and conduct all kinds of mapping, verification, and dropping to end-to-end sessions, especially TCP. One immediate obstacle introduced by NAT to MPIP is that many NAT devices drop a TCP packet if they don't have a record about the TCP connection that the packet belongs to. In MPIP, if we transmit TCP packets on a path different from the original one through which the TCP connection is established, NAT devices along the path are not aware of the connection and will drop these packets before they arrive at the destination. We provide two solutions. One solution is to construct a fake TCP three-way hand-shake on the NAT's path before sending packets over. When a client receives the IP address list of the server, it sends out a SYN packet along each possible path to the server except the original one which was used to initiate the real TCP connection. After the fake three-way handshake is completed successfully, NAT routers along the path have a record about this fake TCP connection, will pass TCP packets assigned to the path. Another solution is UDP wrapper. During our experiments, most NAT devices don't verify socket information of UDP packets. We make use of this feature and wrap a TCP packet inside a UDP packet to pass NAT checking. Whenever MPIP chooses for a TCP packet a path different from its original path, it encapsulates the TCP packet into an UDP packet by adding a forged UDP header using the corresponding IP addresses and port numbers of the chosen path. At the receiver end, MPIP removes the UDP header and extract the original socket information from Table II to be filled into the TCP and IP headers.

B. Out-of-order Packet Processing

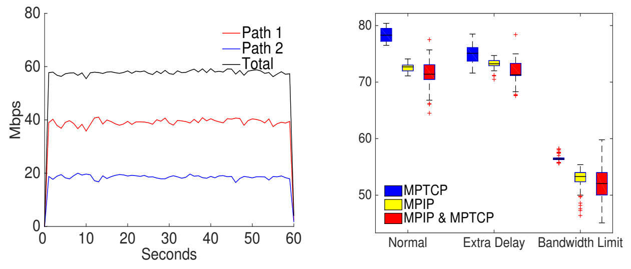
Packets sent over multiple interfaces/paths can arrive at the destination node out of order. When TCP works over MPIP, if the delay difference between multiple paths is significant, we can expect a lot of out-of-order packets. To resolve this problem, for each session in Table II, if it is TCP protocol, MPIP maintains the sequence number S of the next in-order packet of the session to be received. MPIP also maintains a separate re-sequencing buffer B for each active session to store out-of-order packets. Whenever a new packet is received, if the sequence number is larger than S, it will be stored in B; if the sequence number equals to S, MPIP pushes all consecutive packets in B to the transport layer and update S accordingly. To avoid blocking introduced by a lost packet, we limit the size of re-sequencing buffer. All the packets in the buffer will be pushed up once the buffer is full. In our prototype, we set the maximum buffer size to 100 packets.

C. MPTCP over MPIP

A MPTCP session employs multiple subflows, each of which is a legitimate TCP connection over a single IP path. When MPTCP runs over MPIP, each TCP subflow can now utilize multiple paths. For the example in Figure 1, a MPTCP session can have 4 subflows. MPIP will treat each subflow as an independent TCP session, and will create 4 paths for each subflow. As a result, there are totally 4 sessions and 16 paths managed by MPIP. When congestion accumulates on one path, MPIP will first notice the high queuing delay on that path, reduce the path weight and shift packets to less congested paths. The load balancing conducted by MPIP at the network layer makes the congestion variations along different paths less perceivable for MPTCP subflows so that MPTCP can make better use of subflows to achieve higher throughput. We will demonstrate this using MPTCP+MPIP experiments in Section V-A1.

V. Performance Evaluation

To evaluate the performance of the proposed design, we implement MPIP in Linux kernel 3.10.11 in Ubuntu system for IPv4. The main MPIP functions are implemented with more than $5,000$ lines of code. MPIP is also implemented into Android system 6.0.1 with kernel version 3.10.73. For all TCP experiments, we use CUBIC-TCP [7]. MPTCP version 0.92 is used in our evaluation. We use *Iperf/Iperf3* to generate traffic.



(a) Load Balancing (b) MPIP and MPTCP

Fig. 4. TCP over MPIP Performance

A. Controlled Lab Experiments

In our lab, we install the prototype on two desktop computers, which are connected directly to a router. Each desktop has two 100Mbps NICs, leading to 4 paths with aggregate capacity of 200Mbps. We use *tc (traffic control)* tool in Linux to control bandwidth and delay on each path.

1) TCP over MPIP: To test the effectiveness of MPIP load-balancing, we enable only two parallel paths between the two desktops so that they don't share any NIC to prevent traffic coupling. To make it more intuitive, we limit the bandwidth of path 1 to 40Mbps and path 2 to 20Mbps. From the throughput trend in Figure 4(a), both paths converged close to their capacities and remained stable for the whole experiment. We then compare path failure response time of TCP/MPIP and MPTCP/IP by disconnecting then reconnecting one path. MPTCP always suffers a $10 \sim 20$ seconds delay to re-establish the subflow. MPIP promptly detects the re-activated path at the network layer to ramp up the throughput.

As mentioned in Section IV-C, MPIP should be compatible with MPTCP. Three groups of experiments are conducted for different combinations of multipath transmission at transport and network layers, namely, MPTCP/IP, TCP/MPIP, and MPTCP/MPIP. For the first group (normal), two available paths with 40Mbps bandwidth each are configured; for the second group (extra delay), an extra 10ms delay is added to path 1; at last, bandwidth of path 1 is limited to 20Mbps. In Figure 4(b), the boxplots for throughputs of all combinations are plotted. MPTCP/IP throughput is stable and close to the capacity in all cases. TCP/MPIP and MPTCP/MPIP throughputs are little lower but still close to the capacity. Their throughput variances are also larger than MPTCP. The interaction between MPIP load balancing and upper layer congestion control needs further study and fine-tuning.

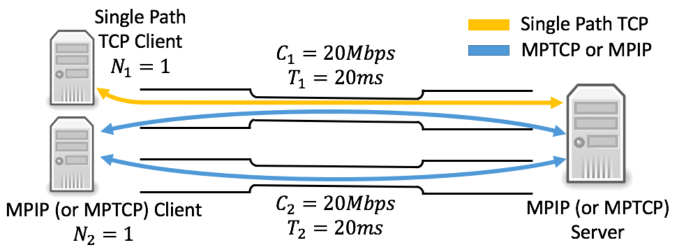
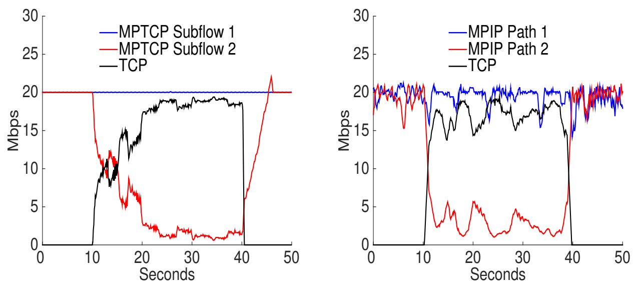


Fig. 5. MPTCP/MPIP Compete with Single Path TCP

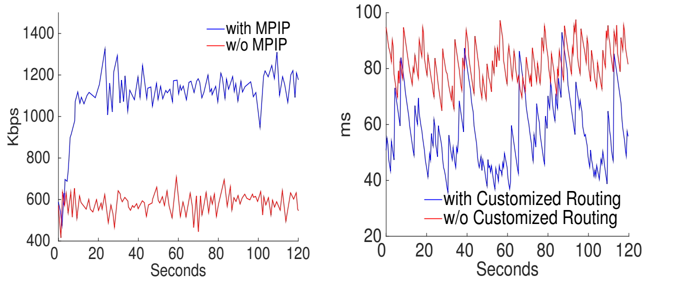


(a) MPTCP/IP and TCP/IP (b) TCP/MPIP and TCP/IP

Fig. 6. Fairness with Legacy TCP/IP

2) Fairness with Legacy TCP/IP: We next conduct experiments to study how TCP/MPIP co-exists with legacy TCP/IP sessions, and compare it with MPTCP. Consider a network containing three types of sessions, TCP/IP, MPTCP/IP, and TCP/MPIP, illustrated in Fig 5. Similar to the MPTCP fairness study in [8], two paths are set up with two bottleneck links of $20Mbps$. The upper path is shared by the TCP/IP session and MPIP (or MPTCP) session. The MPIP (MPTCP) session starts first. The TCP/IP session follows after ten seconds, and lasts for thirty seconds. Fig 6(a) illustrates how MPTCP with BALIA congestion control (CC) co-exists with TCP. MPTCP gradually reduces its traffic on the shared path to leave space for the single-path TCP, which eventually gets comparable throughput as MPTCP. When TCP session is done, it takes a while for MPTCP to reclaim the capacity on the shared path. Meanwhile, from Fig 6(b), MPIP reacts much faster than MPTCP to make space for single-path TCP, which obtains nearly all the available bandwidth of the shared link. After single-path TCP completes, MPIP also reclaims the available bandwidth faster than MPTCP. This demonstrates that MPIP's load balancing at the network layer can facilitate fair bandwidth sharing at the transport layer.

3) UDP over MPIP: To evaluate how UDP-based applications, such as Real Time Communications, can benefit from MPIP, we run WebRTC video chat over MPIP and collect application-level performance by capturing the statistics windows of WebRTC-internals embedded in Chrome, then extracting data from the captured windows using WebPlotDigitizer. We first configure two IP paths between two lab machines without bandwidth limit, and then run WebRTC video call between the two machines. To test the robustness of MPIP against path failures, one path is disconnected in the middle of experiment. If WebRTC video chat is running over legacy IP, when the original path is disconnected, video freezes for few seconds before video flow migrates to the other path. This demonstrates that while WebRTC can recover from path failure at the application layer, its response is too sluggish and user QoE is significantly degraded by a few seconds freezing. With MPIP, video streams continuously without interruption. In addition, to demonstrate how WebRTC benefits from MPIP multipath throughput gain, we limit the bandwidth of each path to 1Mbps. Comparison presented in Figure 7(a) illustrates that with the help of MPIP, WebRTC video throughput improves from 600Kbps to 1200Kbps. We then introduce additional delays of 50ms and 80ms to the two paths respectively. MPIP then use *single-path* mode to route audio packets to the path with shorter delay, while video packets are routed using *all-paths* mode. Figure 7(b) shows clearly that audio delay is reduced by 30ms while the video quality is not affected.



(a) Video Rate under B.W. Limit (b) WebRTC Audio Delay

Fig. 7. WebRTC Performance over MPIP: (a) all-paths mode; (b) single-path mode for audio, all-paths mode for video.

B. Internet Experiments

Besides the controlled lab experiments, we also conduct experiments on the Internet to evaluate MPIP's compatibility with real applications and various middle boxes, e.g. NAT routers inside ISP and CSP networks.

1) Coordinated Routing between Applications: We study coordinated MPIP routing for Youtube video streaming and file downloading applications using the testbed in Figure 8. Since we cannot install MPIP on YouTube servers, we configure a MPIP proxy using Squid on Ubuntu. Three NICs are installed on the proxy server: one NIC is connected to Internet, and the other two are connected to a MPIP client through two paths in an emulated network. We setup 2Mbps bandwidth limit for each path and introduced 20ms extra delay to one path.

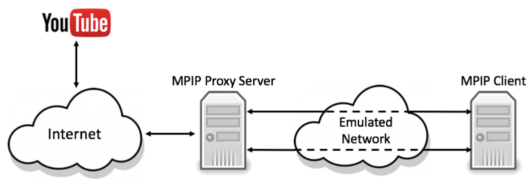
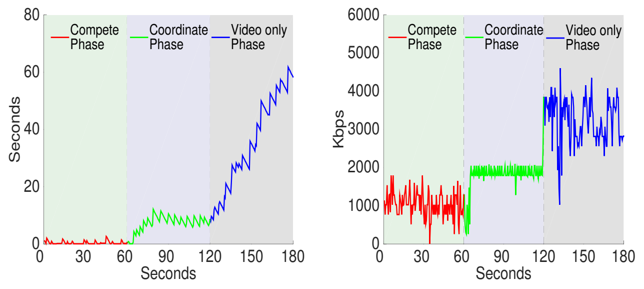


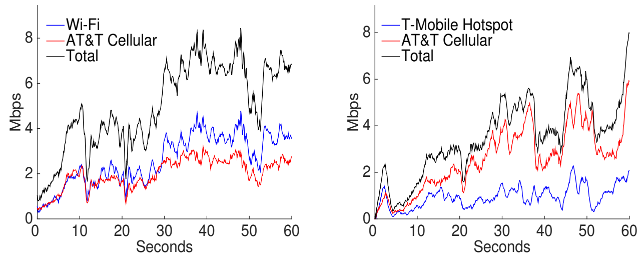
Fig. 8. MPIP works with YouTube through Proxy



(a) Video Buffer Health (b) Video Throughput

Fig. 9. Youtube 720p Video Streaming with Coordinated MPIP Routing

At the beginning, besides the YouTube video session, another file downloading session is added to transmit data from MPIP proxy server to client. Initially MPIP operates in the *all-paths* mode and establishes two paths for each session to acquire more bandwidth. Due to the path delay difference, out-of-order packet deliveries limit the TCP throughput for both sessions. Sixty seconds into the experiment, MPIP implements *coordinated routing*: both sessions are routed using the *single-path* mode, with Youtube session assigned to the path with shorter delay and the file downloading session assigned to the other path. In Figure 9, coordinated routing significantly improve the performance of the video session: video throughout increases by 400Kbps (from $1,500$Kbps to $1,900$Kbps), and buffer length accumulates to 10 seconds without freezing. Meanwhile, the average throughput of the downloading session drops from 2.51Mbps to 1.89Mbps. Since users are more sensitive to video quality than the file downloading throughput, the coordinated routing presumably improves the overall user experience. Sixty seconds later, we terminated the downloading session. From Figure 9(a) and 9(b), we observe that both the video throughput and preload buffer length increase significantly.



(a) WiFi and Cellular (b) Two Cellular Networks

Fig. 10. MPIP over Wireless

2) Android Experiments: We use a Nexus 5X phone located in California to test Android MPIP. The phone is equipped with one cellular interface and one WiFi interface. We use it to download data from a server located in New York City with one public IP address. We first connect the phone to a corporate ISP through WiFi and AT&T CSP through 4G cellular. Without MPIP, the phone can achieve average bandwidth of 4.5Mbps through WiFi and 4.3Mbps through cellular respectively. The average RTTs of WiFi and cellular are 76.2ms and 155.9ms respectively. When MPIP is enabled, as illustrated in Figure 10, Android MPIP can concurrently transmit data on both paths going through different ISP/CSP and reach aggregate throughput of 7.5Mbps in the face of large delay disparity. Next we replace the corporate WiFi router with a hotspot hosted by another phone connected to T-Mobile cellular network. As all data through the hotspot are forwarded by another phone, the average RTT on the T-Mobile path increases dramatically to 349.2ms and the average bandwidth is only 1.52Mbps. Figure 10(b) demonstrates that even when one cellular path has bad performance, MPIP still manages to multiplex bandwidth from two CSPs to achieve higher aggregate throughput.

VI. Related Work

The growing popularity of multi-homed devices makes it possible to initiate multipath transmission from end devices. Back to 2001, Hsieh et al proposed pTCP[9] that effectively performs bandwidth aggregation on multi-homed mobile hosts. In [10], the authors investigated the potential benefits of coordinated congestion control for multipath data transfers. In [11], Dong et al implemented concurrent TCP(cTCP) in FreeBSD to improve throughput. Also, the Stream Control Transmission Protocol (SCTP)[12], [13] is an early protocol designed for multihoming to support failover and simultaneous transmission. In 2010, Barre et al published experimental results of using multiple paths simultaneously in TCP transmission [4], [1]. IETF RFC 6182 [14] for Multipath TCP was published in in 2011. In [2], Chen et al did a thorough measurement of MPTCP over wireless links. Different from those multipath protocols at the transport layer, MPIP is a transparent multipath solution at the network layer of end devices. As bandwidth of cellular network becomes comparable with the wired Internet, switching among WiFi and cellular becomes practical for mobile devices, e.g. [15], [16]. All these solutions require significant changes and coordination at multiple layers. In [17], a pure user-level solution, called msocket, was proposed for seamless handover between different mobile networks. Different from these previous work, MPIP realizes path selection and seamless handover by only changing the network layer. It has long been observed that routing for applications on the same device needs to be coordinated [18], [19]. MPIP serves as a light-weight framework to implement coordinated routing for multiple applications.

VII. Conclusions and Future Work

In this paper, we developed MPIP, a complete design of multipath transmission at the network layer of end devices. MPIP consists of signaling, session and path management, multipath routing, and NAT traversal. MPIP can be used by both TCP and UDP-based applications. It also works seamlessly with MPTCP, and supports user-defined routing strategies. We implemented MPIP in Linux and Android kernels. Through extensive lab and Internet experiments, we demonstrated that MPIP can transparently support flexible and coordinated routing for diverse applications to achieve multipath gains. MPIP is only our first attempt for implementing multipath transmission at the network layer. The signaling and feedback mechanisms can be further optimized to reduce its overhead and improve its robustness. The delay-based load balancing algorithm can be improved to better address path heterogeneity, especially for WiFi, LTE, and the emerging 5G Cellular links. We will extend the user-defined routing framework to support finer routing granularity and more flexible forwarding actions. We will also port MPIP to IPv6. Finally, we will further study the efficiency, fairness and stability of the vertical and horizontal interactions of MPIP with legacy TCP and IP protocols through analysis, simulations and prototype experiments.

References

[1] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How hard can it be? designing and implementing a deployable multipath tcp," in *NSDI*, 2012.

[2] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley, "A measurement-based study of multipath tcp performance over wireless networks," in *IMC*, 2013.

[3] Y.-C. Chen, D. Towsley, E. M. Nahum, R. J. Gibbens, and Y.-s. Lim, "Characterizing 4g and 3g networks: Supporting mobility with multipath tcp," *School of Computer Science, University of Massachusetts Amherst, Tech. Rep*, vol. 22, 2012.

[4] S. Barre, C. Raiciu, O. Bonaventure, and M. Handley, "Experimenting with multipath tcp," in *SIGCOMM 2010 Demo*, September 2010.

[5] C. Paasch, R. Khalili, and O. Bonaventure, "On the benefits of applying experimental design to improve multipath tcp," in *CoNEXT*, 2013.

[6] L. Sun, G. Tian, G. Zhu, Y. Liu, H. Shi, and D. Dai, "Multipath IP Routing on End Devices: Motivation, Design, and Performance," Tandon Engineering School, New York University, Tech. Rep., 2017, available at http://eeweb.poly.edu/faculty/yongliu/docs/MPIP_Tech.pdf.

[7] S. Ha, I. Rhee, and L. Xu, "Cubic: A new tcp-friendly high-speed tcp variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, Jul. 2008.

[8] Q. Peng, A. Walid, J.-S. Hwang, and S. H. Low, "Multipath tcp algorithms: Theory, design and implementation," *IEEE/ACM Transactions on Networking*, 2016.

[9] H.-Y. Hsieh and R. Sivakumar, "A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts," in *MobiCom*, 2002.

[10] P. Key, L. Massoulié, and D. Towsley, "Path selection and multipath congestion control," *Commun. ACM*, vol. 54, no. 1, Jan. 2011.

[11] Y. Dong, D. Wang, N. Pissinou, and J. Wang, "Multi-path load balancing in transport layer," in *Next Generation Internet Networks, 3rd EuroNGI Conference on*, May 2007.

[12] L. Ong, C. Corporation, and J. Yoakum, "An introduction to the stream control transmission protocol (sctp)," IETF RFC 3286, 2002.

[13] I. Joe and S. Yan, "Sctp throughput improvement with best load sharing based on multihoming," in *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*, Aug 2009.

[14] A. Ford, C. Raiciu, M. Handley, S. Barre, U. C. D. Louvain, and J. Iyengar, "IETF RFC 6182: architectural guidelines for multipath tcp development," 2011.

[15] P. Nikander, T. Henderson, C. Vogt, and J. Akko, "End-host mobility and multi-homing with host identity protocol," IETF RFC 5206, 2008.

[16] A. Singh, G. Ormazabal, H. Schulzrinne, Y. Zou, P. Thermos, and S. Addepalli, "Unified heterogeneous networking design," in *IPTComm*, 2013.

[17] A. Yadav and A. Venkataramani, "msocket: System support for mobile, multipath, and middlebox-agnostic applications," in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, 2016.

[18] H. Balakrishnan, H. S. Rahul, and S. Seshan, "An integrated congestion management architecture for internet hosts," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '99, 1999.

[19] H. Balakrishnan and S. Seshan, "Ietf rfc 3124: The congestion manager," 2001.