Multipath IP Transmission: Motivation, Design, and Performance

***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 attentions from academia and industry. However, MPTCP only supports TCP based applications and traffic rate allocated to each path is totally determined by TCP congestion control. In this paper, we implement multipath transmission at the network layer and develop a Multipath IP (MPIP) design consisting of signaling, session and path management, multipath routing, and NAT traversal. MPIP supports flexible multipath routing strategies to satisfy diverse application needs. We implement MPIP in the latest Linux kernel. Through controlled lab experiments and Internet experiments, we demonstrate that MPIP can effectively obtain multipath gains at the network layer. MPIP not only can be used by the legacy TCP and UDP protocols, but also works seamlessly with MPTCP. For TCP-based applications, MPIP's performance is comparable to MPTCP, and the best performance can be obtained when they work together.**

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 from it. Recently, Multipath TCP (MPTCP) has been proposed and got lots attention from academia and industry [1], [2], [3], [4]. In MPTCP, if a pair of nodes have multiple end-to-end IP paths, each TCP session is carried by multiple subflows, each of which is an independent regular TCP connection and is routed to one available path. TCP Packets are dispatched to different subflows along different paths. At the receiver end, all packets coming from different subflows/paths are put back for reconstructing the original TCP data stream. MPTCP allows all TCP-based applications enjoy the multipath gain in a transparent fashion. However, UDP-based applications cannot benefit from it. Fundamentally, the question of *"at which layer(s) should multipath transmission be implemented?"* is still largely open. On one hand, routing is intrinsically a network layer function; on the other hand, end-to-end multipath transmission can be conveniently initiated and managed by end nodes at transport and application layers.

In this paper, we share our experience of implementing multipath transmission at the network layer of end nodes and present our design of Multipath IP Transmission (MPIP). MPIP has several advantages over MPTCP:

- *Broader Coverage.* While MPTCP can only be used by TCP based applications, MPIP can transmit IP packets generated by both TCP and UDP based applications.
- *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.
- *More Flexible Routing.* With MPTCP, traffic allocated to each path is determined by the rate achieved by the TCP subflow on that path, i.e., routing is simply a result of congestion control. It is too rigid for applications with diverse throughput and delay needs. MPIP instead can implement any customized multipath routing strategy to satisfy application needs at the network layer.
- *Lower Complexity.* MPTCP has to work with complicated implementations and constraints imposed by the legacy TCP implementations. To the contrast, the legacy IP protocol is stateless and much simpler than TCP. It gives lots of freedom for MPIP design and implementation.

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 the 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. They pose significant challenges to MPIP. 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 to achieve multipath gains 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 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.

2) We design a simple delay-based routing algorithm for MPIP to balance the loads of available paths. We also propose a user-defined mutlipath routing framework, through which customized routing strategies can be realized by MPIP to satisfy diverse application needs.
3) We implement MPIP in the latest Linux kernel. We evaluate its performance using controlled lab experiments and Internet experiments. We demonstrate that MPIP can effectively achieve various multipath gains at the network layer. For TCP sessions, MPIP's performance is comparable to MPTCP, and the best performance can be obtained when they work together.

The rest of the paper is organized as follows. Section II describes the related work. The design of our implementation is introduced in Section IV. In Section VI, we report the experimental results for our multipath IP design. We conclude the paper with summary and future work in Section VII.

II. Related Work

Multipath transmission has been an active research area for quite some time. Back to 2001, Hsieh et al proposed pTCP[5] that effectively performs bandwidth aggregation on multi-homed mobile hosts. In [6], the authors investigated the potential benefits of coordinated congestion control for multi-path data transfers. In [7], Dong et al implemented concurrent TCP(cTCP) in FreeBSD to improve throughput of connections through balancing traffic load on multiple end-to-end paths. As bandwidth of cellular network becomes comparable with the wired Internet, switching among WiFi and cellular becomes practical for mobile devices. In [8], the authors designed a complete system that supports smooth transfer among different networks. Shuo et al proposed a transport framework of mobile network selection named Delphi in [9]. In 2010, Barre et al published experimental results of using multiple paths simultaneously in TCP transmission [10]. After two years, based on IETF RFC 6182 for multipath TCP in 2011, the same team implemented a complete prototype of multipath TCP in Linux and Android system [1]. They also explored many other aspects of MPTCP in [3], [11], [12], [4]. This prototype is used by other researchers for measurement and improvement of MPTCP. In [2], based on WiFi and cellular networks, Chen et al did a thorough measurement of MPTCP over wireless links. In [13], a delay-based congestion control algorithm, a variation of TCP Vegas [14], was proposed for multipath TCP. In industry, Apple implements multipath TCP in iOS 7.0 for SIRI, a cloud-based natural language voice command and navigation service, which is the first large-scale industry deployment of multipath TCP.

III. Semantics

Before describing the detailed design, we first define basic elements in MPIP and present work flow of MPIP.

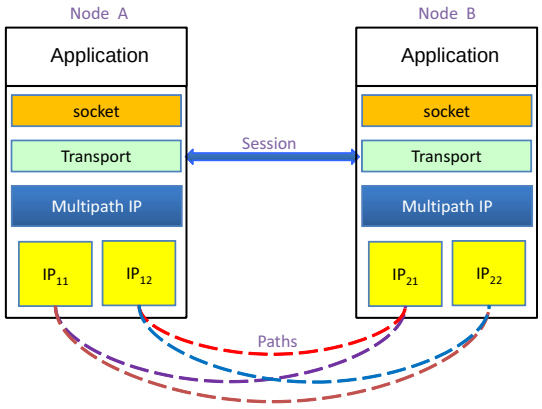


Fig. 1: Example of MPIP Transmission

- *Node* refers to an end device with potentially multiple network interfaces, each of which gets assigned with a private or public IP address.
- *Session* is the communication session between two nodes. A session is established at the transport layer, using the legacy TCP or UDP protocol.
- *Path* refers to an end-to-end path available for a session. For each session, MPIP can use any network 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 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 it changes the IP address and port number back to the original values of the session. Then the packet will be passed to TCP. The whole process is transparent to TCP. If MPIP can simultaneously utilize the four paths by dispatching different packets to different paths, TCP throughput can be improved. Also the session can run as long as one path is available. This means the TCP connection will not be interrupted even if the default interfaces assigned to the TCP connection by the OS are disconnected. This makes hand-overs between different networks seamless and transparent to the transport and application layers.

To realize the gain of MPIP, there are several major steps.

1) *Signaling Channel:* MPIP needs coordination between the two end nodes. We implement a signaling channel by adding a control message block to each IP packet.
2) *Handshake:* MPIP works only if both ends are MPIP enabled. We design a handshake mechanism for a MPIP-enabled node to probe the MPIP availability of a remote node and configure MPIP transmission.
3) *Session Management:* MPIP conducts session-based routing. Session management takes care of the addition and removal of TCP and UDP sessions.
4) *Path Management:* The available paths between two end nodes are explored at the beginning of each MPIP session. During the session, MPIP also continuously monitors the availability and quality of all paths.
5) *MPIP Routing:* MPIP dispatches packets to different paths to take advantages of multi-path gain, seamless transition between networks, as well as application-specific customized routing.
6) NAT Traversal: The existence of NAT routers significantly increases the complexity of MPIP. All components are designed to work with NATs.

We present the detailed design for each component in the following section.

IV. Detailed Design

MPIP only changes the network layer and is transparent to the transport and application layers. To keep the simplicity of IP protocol, MPIP is still implemented as connectionless, while maintaining some feedback information of the available paths necessary for MPIP routing. We achieve this by simply keeping track of several key tables. As a result, MPIP implementation does not add too much overhead to the kernel.

A. Signaling Channel

Due to its connectionless design, IP protocol doesn't have its built-in end-to-end feedback channel. MPIP routing algorithms do need realtime information about the availability and performance of end-to-end paths. It needs a signaling channel. Instead of transmitting extra signaling packets, we piggyback MPIP control and feedback information to each MPIP packet. We add an additional control message (CM) data block at the end of user data, as shown in Table I. The size of the CM block is around 25 bytes, which is negligible for typical data packets of 1000+ bytes.

TABLE I: Control Message Structure

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

Source Node ID is a globally unique identification of the sending node of this packet. To have a semi-static node ID, we use the MAC address of one selected NIC (preferably a more static one) to be its ID. The value of node ID is initiated when the system starts and remains unchanged until it exits.

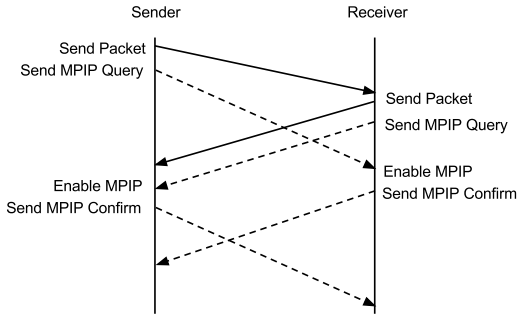


Fig. 2: MPIP Handshake

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

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

Checksum is used to verify the integrity of the CM. Checksum is calculated using all other fields in CM. Receiver verifies a CM by recalculating its checksum.

Other fields will be explained in the following sections.

B. Handshake

TABLE II: MPIP Availability

Dest. IP Address	Dest. Port Number	MPIP Availability	Query Count
IP_1	P_1	True	2
IP_2	P_2	False	5

To take advantage of MPIP, both end nodes of a session need to be MPIP enabled. Locally, every node maintains a table (Table II) to record the availability of MPIP on remote nodes. Due to the existence of NAT, two different remote nodes might share the same IP address, this is why we have to index each entry using the combination of IP address and port number.

The MPIP handshake process is illustrated in Figure 2. When a node receives a packet from the transport layer, it first checks locally whether the destination address and port number has an entry in Table II. If yes and MPIP availability is true, then the packet will be sent out using MPIP; if MPIP availability is false, it will be sent out as a normal IP packet to be backward-compatible. If there is no entry found in the table, besides sending out the packet as a normal IP packet, MPIP makes a copy of the packet and inserts the CM block with $Flags\_Enable$ for MPIP query. When the packet is received by a MPIP-enabled node, the receiver adds the sender's IP address and port number into its own MPIP availability table with value of $True$, then sends back a confirmation packet to the sender with $Flags\_Enabled$. When the sender receives the confirmation, it will add the receiver's address and port number to its local MPIP availability table.

With the handshake flow above, the smallest number of query packet that will be sent for the whole process is only one. But sometimes because of packet loss or synchronization

issues, there can be multiple query packets sent out by both ends. This is not a problem because the design of the system allows receiving more than one query packets and confirmation packets. In Table II, the column *Query Count* maintains the number of query messages that have been sent out to each destination. If the number is larger than a threshold, it assumes that the destination doesn't support MPIP, and mark the availability in the table as False. In our experiments, we set the threshold to five.

TABLE III: Node ID vs IP address and Port

Node ID	IP Address	Port Number
ID_1	IP_{11}	P_{11}
ID_1	IP_{12}	P_{12}
ID_2	IP_{21}	P_{21}
ID_2	IP_{22}	P_{22}

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, as in Table III. 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. Note, if there are multiple sessions from the same sender, even if the sessions share the same network interface on the sender, they will use different port numbers. As a result, the number of entries for a remote node can be larger than the number of interfaces available on that node.

C. Session Management

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 different source and destination IP/port addresses than the session's original addresses, we can no longer use IP/port numbers to associate a received MPIP packet to 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 IV. 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. A session's socket information will not be changed even if the IP addresses and port numbers that are initially assigned to the session are not active anymore. As will be shown later, this is to guarantee seamless hand-overs between networks.

After the MPIP availability handshake has been successfully completed, when sending out a packet, the sender checks Table IV 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 IV. The IP addresses, port numbers and protocol are extracted from the packet header, and the destination node ID is obtained from Table III. 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 in its CM block. If there is no entry found in its session information table, it 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. Note that, due to NAT, for the same session, the IP addresses and port numbers seen by a remote node might be different from the values on a local node. This won't cause any confusion as long as the session ID and node ID combination is unique.

Removal of a session is done by expiration. At each node, every time it sends or receives a packet, it updates the session's *Update Time* column in Table IV. For an active session, this time stamp should be updated frequently. If a session is not updated over some time interval, the session is considered to be obsolete and will be removed from Table IV. In our system, this threshold value is set to 120 seconds. After all information related to this session is removed, if there are still more packets coming from that session, MPIP handshake will start over.

The *Next Sequence No* is used for TCP out-of-order packet processing, which will be explained in Section V-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 each end can be used as a path for MPIP transmission. However, this only works if all interfaces on both ends have public IP addresses. To solve this problem, we again 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 V, 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 NOT necessarily the same as those allocated to the session at the transport layer. They are the actual IP and port addresses in the header of MPIP packet.

Given M and N interfaces at each end node of a session, there are totally $M * N$ possible paths that can be added to the session. Let's explain the process through the example in Figure 1. 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. 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 V. Since A knows B is MPIP-enabled, it also tries to send the same packet from its another local interface with IP address sip_2 by changing its source addresses to $\langle sip_2, sp_1 \rangle$. When B receives the packet, possibly due to NAT, the source

TABLE IV: Session Information

Dest. Node ID	Session ID	Source IP	Source Port	Destination IP	Destination Port	Protocol	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

TABLE V: Path information

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_1	dip_1	dp_1	D_{min12}	D_{12}	Q_{12}	Q_{max12}	W_{12}
ID	SID_2	PID_{21}	sip_1	sp_2	dip_1	dp_1	D_{min21}	D_{21}	Q_{21}	Q_{max21}	W_{21}
ID	SID_2	PID_{22}	sip_2	sp_2	dip_1	dp_2	D_{min22}	D_{22}	Q_{22}	Q_{max22}	W_{12}

IP and port addresses in the packet might be different from $\langle sip_2, sp_1\rangle$, say $\langle \widehat{sip_2}, \widehat{sp_1}\rangle$. 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 for the session a new path with destination address of $\langle \widehat{sip_2}, \widehat{sp_1}\rangle$ in its path information table. Now B will also send back packets to A's second interface, using destination addresses $\langle \widehat{sip_2}, \widehat{sp_1}\rangle$. When A receives the packet, it confirms the connectivity of its local path between $\langle sip_2, sp_1\rangle$ and $\langle dip_1, dp_1\rangle$, and adds it to its path information table. Similarly, if B has another interface with public address, 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. Packet loss and delay are two important path performance measures. Given that packet losses in the current Internet are rare, we only 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 quality of a path along each direction.

TABLE VI: Path Delay Feedbacks

Source Node ID	Path ID	Path Delay	Feedback Time
ID_{11}	PID_{11}	D_{11}	T_{11}
ID_{12}	PID_{12}	D_{12}	T_{12}
ID_{21}	PID_{21}	D_{21}	T_{21}
ID_{22}	PID_{22}	D_{22}	T_{22}

In Table V, all fields related to network delay will be calculated by one-way path delay feedback from the remote node. When node A sends out a packet, it chooses a path from Table V, sets *Path ID* of the packet's CM block, and sets *Packet Timestamp* with its local system time T_1. After node B receives this packet, it extracts *Source Node ID*, *Path ID* and *Timestamp* from the CM block. The node ID and path ID are directly used to identify records in its local path delay feedback table (Table VI). The one-way delay for the path from A to B is calculated as $T_2 - T_1$, where T_2 is B's local system time when receiving the packet. Node B checks whether the path that identified by the source node ID and path ID already exists in Table VI, if yes, it updates the path's delay with $T_2 - T_1$; otherwise, it adds a new entry into Table VI. In practice, the absolute value of path delay calculated here isn't the real delay value due to the clock difference between node A and node B. But our path selection algorithms depend on the relative ordering and variations of path delays, instead their absolute values. Clock difference between nodes has little impact.

When node B needs to send a packet back to node A, B piggybacks path delay information with the packet. It chooses the record with the earliest (oldest) feedback time from Table VI, sets the field *Feedback Path ID* and *Path Delay* in the packet's CM block, and updates the column *Feedback Time* with its local system time. When node A receives the packet, it extracts the path ID and path delay value from its CM block, and fills the path delay value into the column *Real-Time Network Delay* in Table V.

3) Dynamic Path Addition and Removal: As multipath feature enabled on a device, IP addresses of interfaces change dynamically. A mobile device can connect to different access points (WiFi hotspot/Cellular Tower) during a session. Its IP addresses can be changed, removed or added back dynamically. To make the changes transparent to the session, MPIP supports dynamic addition and removal of paths from Table V. When IP address change happens on one node, it will set $Flags\_IP\_Change$ in the CM block of its next outgoing packet. After receiving a packet with this flag, the receiver knows that IP address change happened on the sender, it removes all entries of all paths of this session in all tables, including Table II, Table III, Table V and Table VI. At the mean time, the entry for this session in Table IV remains unchanged. The path that sends out the IP change notification packet 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 into the system through the procedure in Section IV-D1.

E. Multipath Routing

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

1) Concurrent Transmission: 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. As in Table V, we maintain a *Path Weight* for each active path. Each packet will be dispatched to a path k with the probability $P(k)$, which is calculated as:

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

Path weight is the only criterion for path selection and determines the performance of MPIP load balancing. In our prototype, we use realtime path feedback delay to dynamically update path weights.

The difference between the real-time delay and minimum delay along a path can be used to infer the queuing delay which reflects the congestion level along the path. We can adjust the weight of each path using the real-time queuing delay. In Table V, *Real-Time Path Delay* D is collected using receiver feedbacks as described in Section IV-D2. Every time a new path delay sample D is received, the other three delay metrics are updated as follows.

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 use a simple heuristic iterative algorithm to adjust path weights W_k based on their queueing delays. At each iteration, we first calculate the average queueing delay among the active paths. For a path with delay larger than the average, we decrease its weight by a small amount Δ; for a path with delay smaller than the average, we increases its weight by Δ. The weight saturated at minimum of 1 and maximum of $1,000$. The updated weights will be used to update dispatching probability as in (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.

2) User-defined Multipath Routing: Not all applications take throughput as the first priority. For a live video streaming session, as long as the throughput is higher than the video rate, delay is more critical for the streaming quality. Even for the same application, different data may have different QoS requirements. In the example of video calls, such as FaceTime and Skype, audio stream has low volume but are very sensitive to delay, video stream has high volume and can be less sensitive to delay than audio. To address the diverse needs of applications, we design MPIP to support user-defined routing schemes.

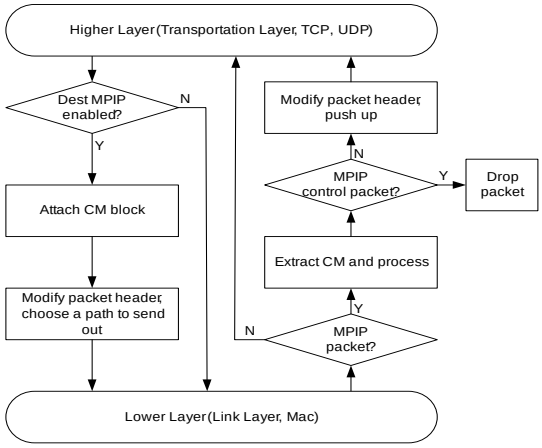


Fig. 3: Work flow of sending and receiving a packet.

A user can inform MPIP of its desired multi-path routing policy by configuring a routing priority table as illustrated in Table VII. Each line of the table is a customized routing rule

TABLE VII: User-defined Multipath Routing Table

IP Address	Port Number	Protocol	Min Length	Max Length	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

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 concurrent routing scheme presented in Section IV-E1. Outgoing packets with R_f priority will always be sent to paths with the lowest delay. Table VII shows an example of the customized routing table.

To conclude this section, Figure 3 presents the workflow of how MPIP sends and receives packets.

V. TCP Related Issues

MPIP changes the default single-path transmission to achieve higher throughput and robustness. It 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. In this section, we present our solutions to some TCP related issues.

A. NAT Checking

Based on our experiments and other previous studies [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 posed by NAT to MPIP is that many NAT devices will drop a TCP packet if they don't have a record about the TCP connection

that the packet belongs to. 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.

1) Fake TCP Connection: To work around a NAT device that drops packets of a TCP connection established on a different path, we construct a fake TCP connection along the path traversing the NAT before sending packets over that path. Instead of constructing a real TCP connection, we mimic TCP's three-way handshake connection establishment at the network layer and make sure this connection information won't go up to TCP layer. All handshake packets have *CM Flags* set to $Flags\_Hs$, these packets are dropped after being processed by MPIP. As shown in Table I, the field *Local Address List* carries all local IP addresses. Also, the node that initiates the connection is considered as the client. When the 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. When the server receives a SYN packet, it replies with a SYN-ACK packet through the same path. After the client sends out the final ACK packet to the server, the three-way handshake for a fake TCP connection is completed successfully. After this, NAT routers along the path have a record about this fake TCP connection, will pass TCP packets assigned to the path.

2) UDP Wrapper: 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 transmit a TCP packet inside a UDP packet to pass NAT checking. At the sender side, every time the network layer gets a TCP packet from transport layer, MPIP chooses a path to send the packet out. If the chosen path isn't the original path, it encapsulates the whole TCP packet into a 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 can tell this UDP packet is a carrier for a TCP packet instead of a regular UDP packet by checking the *Protocol* field of the path in Table IV. After removing the UDP header, the original socket information will be extracted from Table IV to be filled into the TCP and IP headers.

B. Out-of-order Packet Processing

When TCP works over MPIP, if the delay difference between multiple paths is significant, we can expect a lot of out-of-order packets, which will significantly degrade its performance. We try to mitigate this out-of-order problem at the network layer using a simple algorithm.

For each session in Table IV, 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 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 buffer B; if the sequence number equals to S, MPIP pushes all consecutive packets stored in B to the transport layer and update the next sequence number S accordingly. If one packet is lost or delayed for a long time, all subsequent packets will get stuck in the buffer. As a result, TCP layer will assume that all packets are lost, this will be devastating for the connection. To avoid this, we set up the maximum size of the buffer. All the packets in the buffer will be pushed up once the buffer is full. In our prototype, we set the maximum size to 10 packets.

C. MPTCP over MPIP

MPTCP exploits the multi-path gain at the transport layer. A MPTCP session employs multiple subflows, each of which is a legitimate TCP connection over a single IP path. MPIP exploits the multi-path gain at the network layer. When MPTCP runs over MPIP, MPIP treats each TCP subflow as an independent session, which can now utilize multiple paths. For the example in Figure 1, a MPTCP session can have 4 subflows. MPIP will create 4 paths for each subflow. As a result, there are totally 4 sessions and 16 paths managed by MPIP. Now MPTCP and MPIP work together to adapt the traffic allocated to each path. 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.

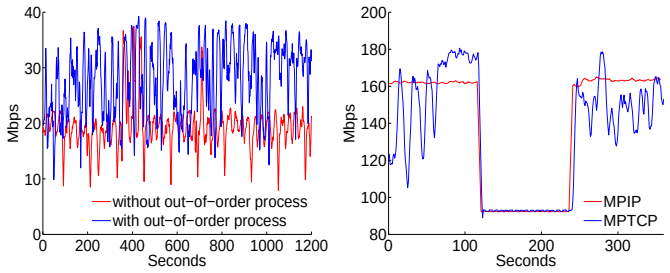
VI. Performance Evaluation

To evaluate the performance of the proposed design, we implemented MPIP in Linux kernel $3.12.1$ in Ubuntu system. The main functions are implemented in three new files with about $5,000$ lines of code. We modified the file of "ip_input.c" and "ip_output.c" under IPv4 folder to embed our MPIP feature into the existing TCP/IP stack. We then evaluated the performance of the system in both controlled lab environment and on the Internet. In all experiments, we tried to keep the configuration of each node unchanged after installation. We conducted side-by-side comparison with TCP and MPTCP. For all TCP experiments, we use CUBIC-TCP [15] as the congestion control algorithm. MPTCP version 0.88 is used in our evaluation. *Iperf3* is used to generate traffic.

A. Controlled Lab Experiments

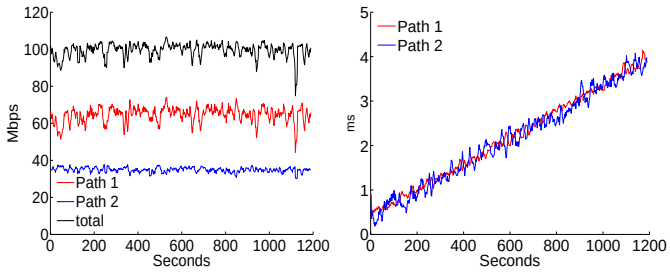
In our lab, we installed the prototype on two desktop computers, which are connected directly to a router without any middle-box. Each desktop has two $100Mbps$ NICs, leading to 4 paths with aggregate capacity of $200Mbps$ between the two nodes. We used *Netem* tool in Linux to control bandwidth and delay on each path to generate various multi-path scenarios. Wireless connections were also used in our evaluation.

1) Impact of Out-of-order Packet Processing: To test out-of-order packet processing in Section V-B, we replaced one wired NIC on one desktop with a WiFi NIC. As a result, the RTT of one path with two wired NICs is about 0.1ms while the paths from the wireless NIC is about 0.5ms. The RTT difference is large enough to generate out-of-order packet



(a) (b)

Fig. 4: a) TCP throughput w./w.o. out-of-order packet processing; b) Dynamic Addition and Removal of Paths



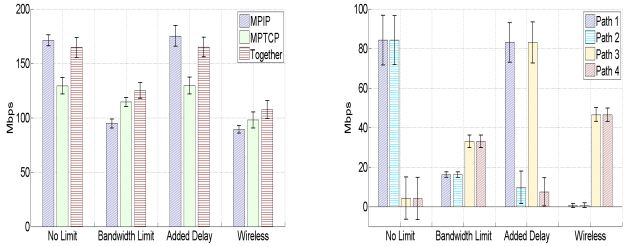
(a) Path Throughput (b) Queuing Delay

Fig. 5: Load Balancing between Available Paths

deliveries. Also, to make sure that there are significant load of packets to be assigned to the wireless NIC card, instead of using the standard path selection algorithm in Section IV-E, we fixed the weight of all the four paths to be the same. Then we made sure that 50% of outbound packets will be assigned to the wireless NIC. With this configuration, we ran a regular TCP session that lasts for 20 minutes with MPIP out-of-order processing enabled and disabled respectively. The result is shown in Figure 4(a). With all the other configurations being the same, the average throughput achieved by TCP/MPIP with and without out-of-order process were 28.2Mbps and 19.4Mbps respectively. This shows that TCP throughput is vulnerable to out-of-order packet delivery, and the proposed solution at MPIP is effective in mitigating the negative impact. In all the following experiments, we enable out-of-order process by default.

2) Resilience to Path Addition and Removal: We conducted experiments to test MPIP's capability to handle dynamic path addition and removal. We conducted one experiment for TCP over MPIP, another experiment for MPTCP over IP, in which MPTCP handles the addition and removal of paths. As illustrated in Figure 4(b), each experiment lasted for six minutes with three stages. We started an experiment with the four paths fully functional. After two minutes, we unplugged the wire of one NIC on the client to reduce the number of paths to two. The wire was plugged back for the last two minutes. In both cases, MPIP and MPTCP adapted to the changes smoothly. The transmission of Iperf3 didn't stop even though its throughput varied with the number of available paths. Because of the overhead of closing and opening new TCP subflows, when the available paths changed, MPTCP's adaption was slower than MPIP, especially when paths were added back. Also MPTCP throughput has larger oscillations than MPIP even when the paths were stable. We will come back to this issue later.

3) TCP Throughput Improvement: 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. We limited the bandwidth of path 1 to 80Mbps and path 2 to 40Mbps. From the throughput trend in Figure 5(a), both paths converged close to their capacities and remained stable for the whole experiment. Figure 5(b) shows the inferred one-way queuing delay along each path using the feedback mechanism in Section IV-D2. The linear drift is due to the clock difference between the two nodes. Since both paths experience the same clock difference, and the path weight adjustment in Section IV-E is based on relative delays, MPIP was able to allocate packets to the two paths proportionally so that their queue lengths remained stable.



(a) (b)

Fig. 6: Lab Experiments: a) Session Throughput Comparison; b) MPIP Traffic Allocation

Next we conducted side-by-side comparison between TCP/MPIP, MPTCP/IP and MPTCP/MPIP. Figure 6(a) shows the average throughput comparison results for multiple configurations. The error stick on top of each bar is the standard deviation over 20 minutes experiment. To avoid unnecessary errors in the initial transition phase, we didn't include throughput data in the first 100 seconds. We first conducted experiments without any bandwidth limiting. The aggregate capacity for the four paths is 200Mbps. We can see that TCP/MPIP achieved the highest throughput of 171Mbps, MPTCP/IP got 129.5Mbps. When we combine MPIP and MPTCP together as stated in Section V-C, we got a throughput of 164.6Mbps. To increase path capacity diversity, we limited the bandwidth of one client NIC so that its capacity to each server NIC is 20Mbps. The total capacity along all four paths became 140Mbps. TCP/MPIP got the lowest bandwidth here, followed by MPTCP/IP. This suggests that as the capacity diversity among paths increases, MPTCP does better than MPIP. When we combined MPTCP with MPIP, the highest throughput was achieved. In our local testbed, with wired connections,

the round trip time is trivial (about 0.1ms). To emulate a connection with longer delay, we manually added 5ms delay to each NIC on the client and plotted the results as the third bar group in Figure 6(a). The results are consistent with the first group with short delays. By replacing one NIC on the client by a WiFi NIC, we evaluated the performance of MPIP with hybrid wireless and wireline connections. In this case, MPTCP achieves higher throughput than MPIP, but still, when they worked together, the highest throughput was achieved.

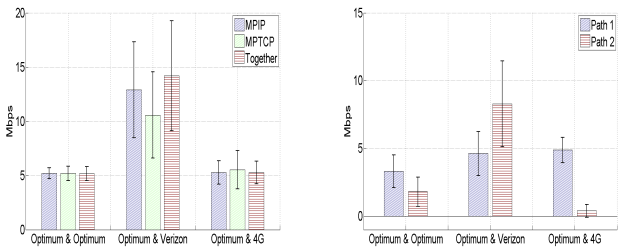
In Figure 6(b), we plot the throughput shares of all paths employed by MPIP. For group 1 and group 3, even though all paths have identical network configuration, the achieved throughputs can be quite different. This is due to the traffic coupling of paths. Different from the independent paths in Figure 5, the four paths are no longer independent and any pair of paths share a common NIC, either on the client side, or the server side. As a result, the queuing delay generated by one path may also affect the queuing delay on another path. The throughput order among homogeneous paths are not predictable. When we limited bandwidth of one NIC (group 2), as expected the two limited paths(1 and 2) have much lower throughput than those two unlimited paths (3 and 4). Finally, for the wireless experiment, the two paths from the wireless interface almost got nothing. This is due to high and unstable delays on these two paths. The throughput variations on individual paths in Figure 6(b) are larger than the variations of the aggregate throughputs of MPIP sessions in Figure 6(a). Other than higher average throughput, the improved smoothness is another advantage of multi-path transmissions.

B. Internet Experiments

Besides the controlled lab experiments, we also conducted Internet experiments to evaluate MPIP's robustness and compatibility with middle boxes, including NAT routers. We set up our server on Emulab [16] which is located in Utah while the client is in a major city in east coast. Because the Emulab server only has one NIC, there are only 2 paths in this case.

We conducted three sets of experiments with different bottleneck placement. First, we connected both NICs of our local client to the same NAT router that connects to Optimum Cable with 15Mbps bandwidth. Secondly, we connected one NIC to Optimum and connected the other NIC to Verizon FIOS with 25Mbps bandwidth. Finally, we connected one NIC to Optimum and connected the other wireless NIC to T-Mobile $4G$ network through an iPhone 6 plus's hotspot.

In our experiments, MPIP can always successfully traverse the local NAT router and potential middle-boxes inside ISP and cellular networks and established two paths to the Emulab server as expected. The end-to-end bandwidth between our local client and the Emulab server is about 5Mbps through Optimum, 10Mbps through FIOS, and 900Kbps through T-Mobile 4G. The results of Iperf3 transmissions over different transport and network layer combinations are compared in Figure 7(a) and Figure 7(b). When two paths share the same Internet access, all three sessions fully utilized the capacity of the connection with little throughput difference. The two



(a) (b)

Fig. 7: Internet Experiments: a) Session Throughput Comparison; b) MPIP Traffic Allocation

paths didn't have equal throughput share mostly because of the traffic coupling between them. When Optimum and FIOS were used, MPTCP got the lowest throughput among the three sessions, followed by MPIP. Still, when MPIP and MPTCP worked together, the highest throughput was achieved. In Figure 7(b), the throughput share along each path is close to its capacity. This proved that the delay-based path selection algorithm in Section IV-E1 also works well in real Internet.

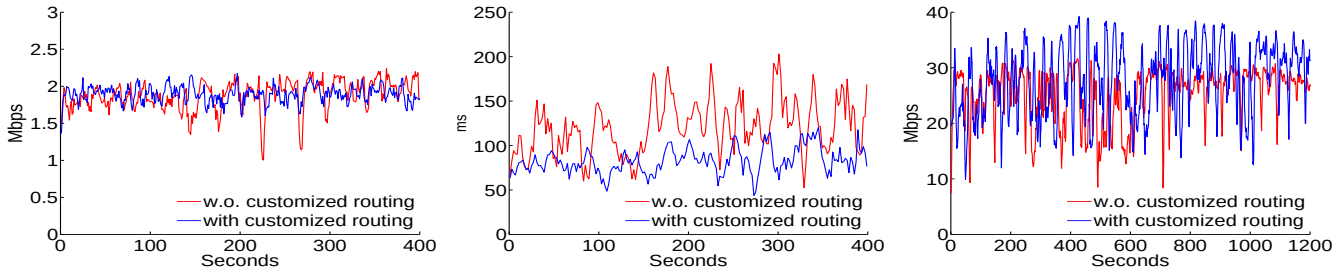
For experiments with a 4G connection, MPTCP got the highest throughput, the throughput of the other two sessions were slightly lower. According to our observation, compared with Optimum and FIOS, the delay of the 4G connection was very unstable, mostly due to varying signal strength and interference on the 4G link. According to the delay-based path selection algorithm, the weight of the 4G path was very small. This can be shown by the throughput share for the 4G path in Figure 7(b). When MPIP and MPTCP worked together, both TCP subflows have two paths at the IP level, neither made good use of the 4G path because of the unstable delay.

C. UDP and Customized Routing

In this section, we demonstrate the gain of UDP over MPIP and customized routing. We don't include any throughput improvement results for UDP here since the result is straightforward. We got nearly twice the throughput by running UDP over MPIP in our lab experiments. Instead, we use Skype as an example application to demonstrate the existing applications (TCP or UDP) can benefit from MPIP's customized routing.

Skype uses direct connection for two-party video calls [17]. All the video and audio packets are transmitted between the two ends using UDP. During a video call, users are more sensitive to audio delay than video delay. According to our experiments, almost all Skype audio packets are less than 200 bytes while video packets are generally larger than $1,000$ bytes. To achieve short audio delay, we added one entry to Table VII to assign packets smaller than 200 bytes with responsiveness-first priority. In addition to the Skype call, we also run an Iperf3 TCP session between the two nodes to see how TCP and UDP coexist over MPIP.

We used only one NIC on the server and two NICs on the client. For path 1, we set the bandwidth to 2Mbps and the delay to 50ms. For path 2, we set the bandwidth to 300Kbps



(a) Iperf3 TCP Throughput (b) Skype Audio Delay (c) Throughput with ACK Optimization

Fig. 8: Responsiveness-first Routing for Skype UDP Packets and TCP ACK Packets

and the delay to 20ms. Then we have one high bandwidth-delay product path and one low bandwidth-delay product path. By enabling and disabling customized routing, we got the results in Figure 8. Figure 8(a) shows the throughput of Iperf3 TCP session; Figure 8(b) shows Skype's audio round trip time which was extracted from Skype's real-time technical report window during the experiments. It shows the amount of delay the user experienced during the call. From Figure 8(b), we see the huge reduction of audio delay. The average audio RTT is 82ms and 119 ms with and without responsiveness-first routing for short packets. Meantime, from Figure 8(a), we see that the TCP session got roughly the same throughput in both cases.

Similar to Skype audio packets, TCP ack packets are normally short, but are delay sensitive. If a ACK packet is overly delayed or lost, the TCP congestion control will be triggered and degrade the throughput performance. We can also apply customized routing and assign ACK packets to paths with short delay and low loss rate. To demonstrate this, we conducted an experiment using the same configuration as in Figure 4(a), by replacing one wired NIC with wireless NIC. Without customized routing, we assign the same weight to all paths. In Figure 8(c), we compare the throughput of the same TCP session when enabling and disabling responsive-first routing for small packets in MPIP. We can see obvious throughput improvement. Customized routing in MPIP improved the average throughput from 24.2 Mbps to 28.5 Mbps.

VII. Conclusions and Future Work

In this paper, we developed MPIP, a complete design of multipath transmission at the network layer. MPIP consists of signaling, session and path management, multipath routing, and NAT traversal. MPIP can be used by both TCP and UDP applications. It also works seamlessly with MPTCP, and supports user-defined customized routing. We implemented MPIP in the latest Linux kernel. Through lab and Internet experiments, we demonstrated that MPIP can effectively achieve various multipath gains at the network layer. The best performance can be obtained when MPIP and MPTCP work together. MPIP is just 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 routing algorithm can be improved to better address path diversity, especially for WiFi and cellular paths. MPIP makes it possible to coordinate multipath routing for a mix of TCP and UDP traffic. We are also interested in extending the user-defined multipath routing framework to support finer routing granularity and more flexible forwarding actions. We are currently working on the Android implementation of MPIP. IPv6 support is another feature on its way.

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] C. Paasch, G. Detal, F. Duchene, C. Raiciu, and O. Bonaventure, "Exploring mobile/wifi handover with multipath tcp," in *Cellnet*, 2012.

[4] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental evaluation of multipath tcp schedulers," in *ACM SIGCOMM Capacity Sharing Workshop (CSWS)*, 2014.

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

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

[7] 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.

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

[9] S. Deng, A. Sivaraman, and H. Balakrishnan, "All your network are belong to us: A transport framework for mobile network selection," in *HotMobile*, 2014.

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

[11] G. Detal, C. Paasch, S. van der Linden, P. Merindol, G. Avoine, and O. Bonaventure, "Revisiting flow-based load balancing: Stateless path selection in data center networks," *Computer Networks*, vol. 57, no. 5, April 2013.

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

[13] Y. Cao, M. Xu, and X. Fu, "Delay-based congestion control for multipath tcp," in *ICNP*, 2012.

[14] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "Tcp vegas: New techniques for congestion detection and avoidance," in *SIGCOMM*, 1994.

[15] 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.

[16] Emulab-Team, "Emulab - Network Emulation Testbed Home," http://www.emulab.net/.

[17] Y. Xu, C. Yu, J. Li, and Y. Liu, "Video telephony for end-consumers: Measurement study of google+, ichat, and skype," in *ACM Internet Measurement Conference*, 2012.