# FMTCP: A Fountain Code-Based Multipath Transmission Control Protocol

Yong Cui, *Member, IEEE*, Lian Wang, Xin Wang, *Member, IEEE*, Hongyi Wang, and Yining Wang

***Abstract*—Ideally, the throughput of a Multipath TCP (MPTCP) connection should be as high as that of multiple disjoint single-path TCP flows. In reality, the throughput of MPTCP is far lower than expected. In this paper, we conduct an extensive simulation-based study on this phenomenon, and the results indicate that a subflow experiencing high delay and loss severely affects the performance of other subflows, thus becoming the bottleneck of the MPTCP connection and significantly degrading the aggregate goodput. To tackle this problem, we propose Fountain code-based Multipath TCP (FMTCP), which effectively mitigates the negative impact of the heterogeneity of different paths. FMTCP takes advantage of the random nature of the fountain code to flexibly transmit encoded symbols from the same or different data blocks over different subflows. Moreover, we design a data allocation algorithm based on the expected packet arriving time and decoding demand to coordinate the transmissions of different subflows. Quantitative analyses are provided to show the benefit of FMTCP. We also evaluate the performance of FMTCP through ns-2 simulations and demonstrate that FMTCP outperforms IETF-MPTCP, a typical MPTCP approach, when the paths have diverse loss and delay in terms of higher total goodput, lower delay, and jitter. In addition, FMTCP achieves high stability under abrupt changes of path quality.**

***Index Terms*—Fountain code, multipath TCP, rateless coding, scheduling.**

## I. Introduction

CURRENTLY, the majority of data transmissions go through TCP. In a network with high loss and delay, such as a wireless network, the performance of TCP degrades significantly due to frequent retransmissions of lost or erroneous packets. In addition, a user may want to transmit data at a higher aggregate throughput when having multiple access links to the network. However, conventional TCP cannot enjoy the multihoming feature.

In order to solve these problems, Multipath TCP (MPTCP) [1]–[3] has been proposed to transmit TCP simultaneously over multiple paths to improve the goodput and reliability. When all the paths are good, subflows can transmit as usual, and the goodput of MPTCP is high as expected. However, if the paths have high diversity in quality (i.e., with different loss or delay), the goodput of MPTCP degrades sharply. When a receiver waits for a packet sent on a low-quality path, the receiver buffer may be filled up. Thus, even if other paths have good quality, they cannot send more packets, and the low-quality paths become the bottlenecks of MPTCP. Some studies [1], [4] show that the goodput of MPTCP can be even worse than that of an ordinary TCP in some cases. In Section III, we provide some performance studies to further illustrate the problems.

To solve the bottleneck problem, some attempts [5], [6] have been made to improve the throughput over lossy networks. However, if a large number of packets need to be retransmitted due to the high loss rate, a high overhead to schedule packet retransmissions would be incurred. It is also very difficult to coordinate transmissions among all paths.

The fundamental problem associated with path diversity lies in the slow transmission on low-quality paths. The retransmission, however, cannot address this problem and may even exacerbate the problem. Alternatively, if the transmissions can be made more reliable, it would help to alleviate the above problems. Instead of simply relying on TCP retransmissions, in this paper, we propose a Fountain code-based Multipath Transport Control Protocol (FMTCP) where we introduce fountain code into MPTCP to improve the throughput and reduce the bottleneck impact due to path heterogeneity.

Fountain code [7] is a linear random code for channels with erasures, and its elemental data unit is *symbol* formed with a certain number of bits. An encoding symbol is generated based on random linear combination of the source symbols in a data block. With its low complexity and redundancy, different fountain codes are considered for use in different transmission standards. The most advanced version of a practical fountain code has been standardized in IETF RFC6330 [8] to provide reliable delivery of data objects. As a key advantage, the original data can be encoded into an arbitrary number of symbols based on the transmission quality and the estimated number of symbols to use for data recovery. The receiver can recover the original data with high accuracy after obtaining enough number of the encoded symbols.

Taking advantage of fountain codes, a sender in FMTCP generates *new encoded symbols* for a block based on the remaining number of symbols needed for reliable decoding at the receiver. Instead of retransmitting the lost packets along the same path on which packet loss is detected, new symbols from the same or different blocks are put into one or multiple packets. Packets are flexibly allocated to different TCP subflows for transmissions based on the packet arrival time estimated according to

Manuscript received September 30, 2012; revised June 13, 2013 and June 13, 2013; accepted January 05, 2014; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Allman. Date of publication January 30, 2014; date of current version April 14, 2015. This work was supported by the NSFC project of China under Grant No. 61120106008, the National 863 project of China under Grant No. 2013AA010401, and the NSF of US under Grants CNS-1247924 and ECCS-1231800.

Y. Cui, L. Wang, H. Wang, and Y. Wang are with Tsinghua University, Beijing 100084, China (e-mail: cuiyong@tsinghua.edu.cn; wanglian12@mails.tsinghua.edu.cn; wanghongyi09@mails.tsinghua.edu.cn; wangyn10@mails.tsinghua.edu.cn).

X. Wang is with the Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794 USA (e-mail: xwang@ece.sunysb.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2014.2300140

the transmission quality of each flow; as a result, the low-quality paths will no longer be the bottlenecks of the overall multipath TCP transmission. As only randomly generated symbols are needed for decoding, there is no need for FMTCP to coordinate transmissions on different paths, which not only significantly reduces the complexity of scheduling, but also reduces the gap in transmission time on diverse paths. This will in turn significantly improve the TCP performance. We provide quantitative analysis of our scheduling algorithm and derive an upper bound of the ratio of transmission time on diverse paths. The simulation results demonstrate that FMTCP achieves much higher and more stable performance than the IETF standardized MPTCP (IETF-MPTCP).

Another issue in designing a transport-layer protocol is fairness, i.e., whether it is TCP-friendly. TCP-friendliness mainly focuses on the behaviors when different flows share a common bottleneck link and is generally achieved through the congestion control algorithm. Loss-based congestion control algorithms for multipath transfer have been explored in [1], [9], and [10]. The Vegas has also been extended for multipath transfer to obtain fine-grained load balancing [11] and achieve reasonable tradeoff between latency and throughput in DCNs [12], providing more options for multipath TCP. However, the definition of *fairness* for multipath transmission protocols is still disputable. Becke *et al.* [13] investigate the existing congestion control approaches and try to extend the definition of fairness from single-path transport to multipath transport. In this paper, we mainly focus on packet encoding and data allocation, and our framework can adopt any congestion control mechanisms in the work mentioned above. Moreover, as we only use disjoint paths in the simulations, the selection of congestion control mechanisms would not influence the results.

The rest of the paper is organized as follows. We discuss the related work in Section II and study the limitation of existing MPTCP approaches through simulations in Section III. Then, we present our FMTCP design and data allocation algorithm along with quantitative performance analysis in Sections IV and V, respectively. Finally, we evaluate the performance of the proposed FMTCP in Section VI and conclude the work in Section VII.

## II. Related Work

In a network with high loss and/or delay, such as wireless networks, conventional TCP suffers from performance degradation due to frequent retransmissions and reordering. On mobile phones, the problem may be exacerbated due to limited power [14]. Some solutions [15], [16] optimize congestion control to compensate for the performance degradation caused by packet loss. Other efforts [17]–[19] have been made to introduce network coding into TCP over wireless networks. Huang *et al.* [17] show that network coding helped reduce packet loss probability and decrease the retransmissions, thus improving the throughput. Sundararajan *et al.* [19] propose a scheme to combine random linear network coding with TCP. Compared to the literature work, FMTCP employs coding mechanisms that fundamentally avoid retransmissions and reduces reordering without influencing the fairness of transmission.

Many efforts have been made to explore use of multiple interfaces on a portable device simultaneously to improve the goodput and stability of transmissions [3], [9], [20]–[23]. The studies indicate that transmissions over multiple paths can improve the transmission capacity and quality. An existing multipath transport protocol that supports multistreaming and multihoming is Stream Control Transmission Protocol (SCTP) [24]. However, in the basic SCTP design, other paths are only considered as the backups of a primary path, and Concurrent Multipath Transfer (CMT) is not supported. CMT-SCTP [9] extends SCTP by adding CMT support, which is in the process of IETF standardization. Raiciu *et al.* attempt to design and implement a deployable multipath TCP that is backward-compatible with TCP. They evaluate the behaviors of various middleboxes in relation to TCP extensions through a large number of Internet measurements [25]. Based on the observations, they propose architectural guidelines [26] and extension mechanisms [2], [4] for Multipath TCP development. Their TCP extension for multipath operation has been standardized in [3], and we denote it as *IETF-MPTCP* to differentiate it from general MPTCP schemes. IETF-MPTCP has been used to save mobile energy [27], improve datacenter performance [28], provide opportunistic mobility [29], etc. However, as mentioned earlier and studied in Section III, poor paths can significantly affect the performance of Multipath TCP and hence become the bottleneck. We focus on this performance degradation problem and propose FMTCP based on those schemes above.

The path diversity problem is a common issue in multipath transport protocols. Iyengar *et al.* [30] indicate that in CMT-SCTP, the low-quality paths degrade the overall throughput because the limited receive buffer blocks the transmission, which is consistent with our observations for MPTCP. Dreibholz *et al.* [31] try to address this problem by scheduling. They compare different stream scheduling policies and prove that mapping streams to a specific path performs better than round-robin scheduling. This work aims for multistreaming and provides no guidance for transmitting a single stream over paths with quality diversity. The authors also analyze the buffer blocking problem in [32] and propose to split the buffer among paths to tackle this problem. However, the splitting of the overall buffer into equal-size segments for all paths may waste buffer spaces on low-quality paths. Also, although it may work for disordered transmissions, a large buffer would still be needed to support ordered transmissions commonly ensured by TCP.

The authors of IETF-MPTCP propose to retransmit the first unacknowledged packet on subflows being blocked and penalize subflows that cause buffer blocking by halving the congestion window in [4]. Although the two schemes are simple and help to mitigate the buffer blocking problem, they are only triggered after the buffer blocking occurs, and retransmissions are still needed, which would impact the overall performance. In addition, the cross-path retransmission and the window reduction of the inferior path will waste the capacity of both types of path. Moreover, even if buffer blocking issues are mitigated, the problems associated with the diversity in path qualities remain, which will be further explained in Section III. In contrast, we propose a novel mechanism that exploits rateless coding and path-quality-aware transmission scheduling to improve the performance of MPTCP in lossy and heterogeneous networks. Our proposed scheme avoids the need of retransmitting lost packets and largely reduces the reordering, which in turn avoid buffer blocking and reduce the buffer delay.

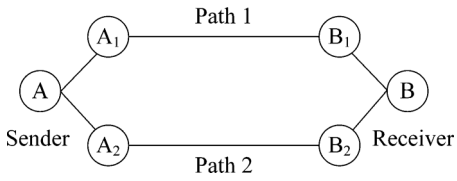



Fig. 1. Simulation topology.

Network Coding has also been considered to improve the performance of multipath TCP in MPLOT [33] and HMTP [34]. MPLOT [33] takes advantage of current diverse paths to improve the goodput of wireless mesh networks and reduce packet recovery latency. It uses a fixed-rate coding scheme, which does not have good performance when the path quality decreases sharply. Moreover, the scheduling scheme is too simple and could not support the variation of transmission quality from different networks. In our paper, we take a step further by introducing a practical rateless code into Multipath TCP, and we design a novel data allocation algorithm. Similar to our solution, HMTP [34] also takes advantage of Fountain Code. However, in HMTP, the sender continues encoding and sending packets until the receiver completes decoding and sends a stop message. This stop-and-wait mechanism is obviously inefficient and possibly increases transmission redundancy. We address this issue by introducing a prediction-based mechanism in scheduling the number of encoded symbols and further optimizing the data allocation among different subflows based on the estimation of path quality.

## III. Limitation of MPTCP

Before presenting our proposed FMTCP scheme, we first provide some analysis on the limitation of MPTCP through simulation studies using ns-2 on a typical multipath TCP protocol, IETF-MPTCP [2].

Generally, MPTCP contains several subflows, with each subflow performing TCP transmission separately. In our simulation, we employ a topology with two nodes connected by two disjoint paths, as shown in Fig. 1, in which the term "disjoint" means that the link parameters (e.g., bandwidth, delay, loss rate, etc.) of two paths are independent from each other. One of the paths has fixed quality, and the other one has its QoS parameters varied based on different simulation settings. To evaluate the performance of IETF-MPTCP, we compare the performance of one IETF-MPTCP flow between two nodes to that of two independent TCP flows transmitting over the two paths.

In Fig. 2, we set up the subflow 1 with 100 ms delay and 0.1% packet loss, while varying the loss rate and delay of the subflow 2 based on the parameters in Table I. The buffer size of each TCP flow is set to the default value, 64 kB, and we study the throughput of IETF-MPTCP under several different buffer sizes. We can observe that the goodput of IETF-MPTCP is always lower than that of two independent TCP flows, denoted as "2-TCP" in the figure.

When the buffer size is 64 kB, the total goodput of IETF-MPTCP is much lower than that of two independent TCP flows, and the difference increases as the quality of subflow 2 gets worse. In Test Cases 1–5, as the loss rate of subflow 2 varies from 1% to 50%, the goodput decreases up to 65.6%. As IETF-MPTCP needs to aggregate the packets from two subflows at the receiver and sends them to the application layer in order, the

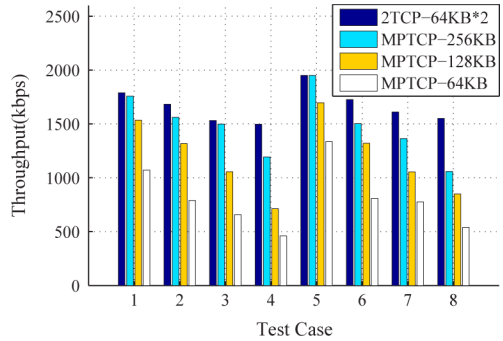



Fig. 2. Goodput comparison between IETF-MPTCP and 2-TCP.

TABLE I
Path Parameters of Subflow 2

| Test Case | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Delay (ms) | 100 | 100 | 100 | 100 | 100 | 200 | 300 | 500 |
| Loss Rate (%) | 2 | 5 | 20 | 50 | 1 | 1 | 1 | 1 |

loss from one subflow will impact the overall goodput of IETF-MPTCP. In Test Cases 5–8, as the delay of subflow 2 varies from 100 to 500 ms, we observe a 59.7% goodput reduction. When the buffer size gets larger, the throughput gets closer to that of 2-TCP. However, in extreme scenarios where there are large losses or delays such as Test Cases 4 and 8, MPTCP still experiences 38.8% and 45.7% goodput reduction, respectively, even when the buffer size is 256 kB.

These results are mainly caused by the heterogeneity in quality of paths over which different subflows go through. MPTCP uses a connection-level receive buffer where packets are stored until they are ready to be delivered to application layer sequentially, and lost packets from a subflow are generally retransmitted through the same subflow. We assume the very first packet in the receiving window happens to be transmitted via an inferior subflow and has not been successfully received yet. When the receiver buffer is full, other subflows cannot send packets and are thus blocked by the inferior subflow. Therefore, the existence of inferior flows limits the total goodput. The higher the loss rate is, the more severe the blocking problem will be.

One possible solution is to simply enlarge the receiver buffer size. We first analyze the minimum receiver buffer size needed. When a packet loss occurs, the receiver has to buffer data from all subflows for the duration of the fast retransmission, which would double the maximum round-trip time $2RTT_{\max}$ if the loss is on the slowest path. In a setup of $n$ paths with bandwidth $BW_i$ and RTT $RTT_i$ of path $P_i$, the minimum buffer size $B_{\min}$ for the sender as well as for the receiver buffer [26] is

$$B_{\min} = 2 * \sum_{i=1}^{n} BW_i * \max_{1 \le i \le n} RTT_i. \quad (1)$$

The worst-case scenario would be when the subflow with the highest round-trip time or retransmission timeout (RTT/RTO) experiences a timeout. In this case, the minimum buffer size $B_{\min}$ to avoid stalling will be

$$B_{\min} = \sum_{i=1}^{n} BW_i * (\max_{1 \le i \le n} RTT_i + \max_{1 \le i \le n} RTO_i). \quad (2)$$

The bandwidth of IEEE 802.11a is 54 Mb/s, and the rate can be 300 Mb/s and even higher in IEEE 802.11g and future standards. With the default minimum RTO being 200 ms in Linux, the minimum buffer size $B_{\min}$ can be more than 100 MB, which is costly, and the cost would be very high when there are more transmission flows. In IETF-MPTCP, the suggested sender or receiver buffer size is calculated as (1), which may still be very expensive when paths with high bandwidth work together with applications that have high delay. On the other hand, as we have shown in our example above, even when the buffer size is large enough, the overall performance may still be constrained by inferior flows. All the packets will be aggregated at the receiver and submitted to the application layer sequentially. Therefore, even if other subflows can send in the packets, they cannot be passed to the application layer in time if some earlier packets from the inferior flows have not arrived. This will introduce a large end-to-end delay and jitter, as will be seen in Section VI.

Note that although IETF-MPTCP attempts to be TCP-friendly so it would not be more aggressive than a single-path TCP, it is not the key reason for the performance degradation in our simulation setting. TCP-friendliness mainly focuses on the behaviors when different subflows share a common bottleneck link. In fact, even if each subflow of IETF-MPTCP follows the conventional TCP congestion control mechanism, it still suffers from the bottleneck problem caused by subflows with lower path quality as long as packets transmitted on different subflows need to be aggregated into an in-order byte stream before transmitting to the upper layer.

## IV. FMTCP Design

Since the degradation of transmission quality of individual subflows could significantly impact the total goodput of MPTCP, we propose to introduce fountain code to improve the transmission quality. Compared to a fixed-rate coding scheme, the rateless fountain code has the benefit of changing the coding rate on the fly based on the receiving quality while introducing very low overhead. In this section, we first introduce the basic architecture of our proposed FMTCP, and then present our reasoning of code selection in FMTCP.

### A. Architecture of FMTCP

The sender-side architecture of FMTCP is illustrated in Fig. 3. We introduce the fountain code into the transport layer and transmit encoded data via multiple paths. A byte stream from applications is divided into blocks, which are taken as the input of the encoding module inserted on top of the data allocation module. After the encoding, each block is converted to a series of encoded symbols, which are carried in packets and transmitted to the receiver.

On the receiver side, encoded symbols are converted back to the original data through a decoding module appended on top of the data aggregation module. Once decoded, the data can be transmitted to the application layer, and the corresponding symbols can be removed from the receiving buffer.

Upon a subflow gets a transmission opportunity, the sender needs to generate encoded symbols from the pending blocks and combine these symbols into a packet for the subflow. A receiver will extract encoded symbols from packets and aggregate the symbols from different subflows. If the received symbols are enough to recover a block, these symbols can be sent to the decoding module.

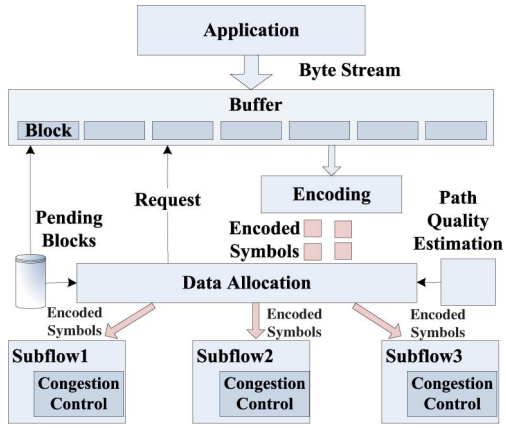



Fig. 3. FMTCP sender architecture.

The main challenges are to determine the number of symbols to transmit for a block and which subflows the symbols of a block should be assigned to in order to improve the goodput. If the receiver has obtained enough information to recover a block, it is redundant to send more symbols for that block. On the other hand, if the received symbols of a block are inadequate to be decoded, they would occupy the receiving buffer and further influence the receiving of latter blocks. We will handle this issue in later sections.

We design our FMTCP based on the architectural guidelines for Multipath TCP development [26] and adopt the IETF-MPTCP stack [3] to support transport-layer coding as well as provide smart data scheduling among multiple subflows. IETF-MPTCP provides the same interface as TCP to the application and manages multiple TCP subflows below it. It consists of four functions: path management, packet scheduling, subflow (single-path TCP) interface, and congestion control.

We focus on the packet scheduling part that breaks the byte stream received from the application into segments to transmit on different available subflows. Before transmission, the coding module encodes the segment, and the data allocation module will determine which subflow the segment will be assigned to based on the path quality estimation. FMTCP inherits the other functions of MPTCP. All MPTCP operations are signaled using optional TCP header fields, so does our FMTCP. To support coding, we can design a new MPTCP option in place of the Data Sequence Signal (DSS) option in MPTCP [3], where an 8-bit source block number and a 24-bit encoding symbol ID are used according to RFC6330 [8] to identify the data in the packet instead of the data sequence number used in MPTCP.

Another issue in designing and implementing a deployable multipath TCP is to support middleboxes. Operations performed by middleboxes in the Internet can be generally categorized as NAT, sequence number rewriting, removing TCP options, segment splitting, segment coalescing, payload modification, and pro-active acking [4], [25].

The main difference between the packet coding of FMTCP and IETF-MPTCP is that FMTCP uses block number and symbol ID instead of data sequence number in IETF-MPTCP. FMTCP can perform as well as IETF-MPTCP when dealing

with NATs and middleboxes that rewrite sequence numbers, and for removing TCP options or modifying the payload. Segment splitting will not be a problem either, as FMTCP can use block number to do data sequence mapping in a way similar to MPTCP and avoid duplicate mappings caused by segment splitting. Segment coalescing would keep only one DSS option due to the limited option space, and thus induce data loss in both MPTCP and FMTCP.

Middleboxes with pro-active acking are mainly TCP proxies that cache the original segment and resend it when retransmission is needed. We retransmit new data in FMTCP. If the original packet and the retransmission are from the same data block, it does not matter which one arrives. Otherwise, it may lead to some wrong estimation of the remaining symbols needed. At this time, we can use acknowledgment to estimate the number of symbols sent for corresponding blocks, at the cost of possibly higher redundancy and delay.

### *B. Code Selection for FMTCP*

Since we are transmitting data via different paths and these paths may have very different quality, it is possible that the low-quality paths block the high-quality ones, causing increasing transmission delay and low goodputs. Therefore, we introduce Forward Error Correction (FEC) code for channels with erasures into the transport layer to alleviate this problem. Here, FEC is not used to correct bit errors, but to recover data in lost packets that will be introduced later. In this way, even if packets are lost on some low-quality paths, the receiver may still be able to recover the original data, and thus the low-quality paths will not block the high-quality ones. There are basically two categories of FEC codes: fixed-rate and rateless.

For a fixed-rate code, a block containing $k_b$ symbols is encoded into $\hat{k}_b$ encoded symbols, where $\hat{k}_b/k_b$ is the coding rate. It is guaranteed that any $k_b$ of the $\hat{k}_b$ encoded symbols can recover the original file. Therefore, the receiver can successfully decode the data if the number of symbols lost is no larger than $\hat{k}_b - k_b$. Otherwise, the sender is forced to retransmit some packets, which may happen if the loss rate is underestimated. This coding scheme works well on channels whose loss rate does not change rapidly. However, in a practical scenario, the loss rate of channels may vary over a large range. This would make a fixed-rate coding scheme no longer suitable since it is hard to alter the coding rate (i.e., the number of encoded symbols generated for one block) during the course of data transmissions.

To solve the problem of fixed-rate coding, rateless coding scheme is introduced. In rateless coding, an arbitrarily number of symbols can be generated for a block. If the receiver does not receive enough number of symbols due to dynamics of the channels, additional new symbols can be generated to form a lower-rate encoding instead of retransmitting the lost ones. Therefore, rateless coding has no fixed rate and its rate adapts between 1 and 0. This feature is also referred as the *prefix property* [35] as an encoding with a higher rate can be considered as a prefix of any lower-rate encoding. With rateless codes, we can adapt the rate according to the dynamics of the channels to improve the throughput of different paths.

In our FMTCP, we use special kinds of rateless codes, called Fountain Codes [7], which are based on random linear coding as illustrated in Fig. 4. In the coding process, a data block is divided into $\hat{k}_b$ equal-length source symbols $\{\rho_1, \rho_2, \cdots, \rho_{\hat{k}_b}\}$,

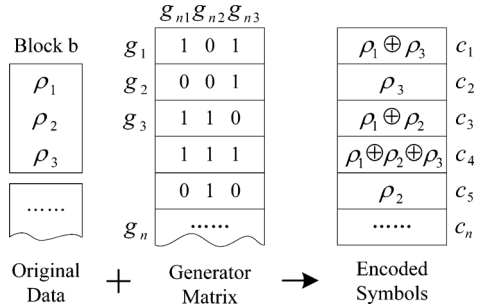



Fig. 4. Rateless coding.

each of which has $x_b$ bits. At each time instant $n$, the sender will generate a $\hat{k}_b$-bit random vector $(g_{nk})$. The corresponding encoded symbol $c_n$ is then set to bitwise sum (modulo 2) of the source symbols for which $g_{nk}$ is 1 by

$$c_n = (\sum_{k=1}^{\hat{k}_b} \rho_k \cdot g_{nk}) \bmod 2. \tag{3}$$

The encoded symbol is also $x_b$ bits. The number of 1 in a random vector is called degree $d_n$. By randomly generating $(g_{nk})$'s that form a $n$-by-$\hat{k}_b$ generator matrix, the sender can produce new symbols whenever needed.

The symbols are put into packets and transferred through different paths. Each packet carries the random vector or a key based on which the decoder can generate the random vector using the same random number generator in the header. The receiver could get the $\hat{k}_b$-bit vector $(g_{nk})$ of the encoded symbols along with the received packets. When the receiver gets $\hat{k}_b$ linear independent random vectors that form a $\hat{k}_b$-by-$\hat{k}_b$ invertible (modulo 2) matrix $G_n$, the original data can be decoded.

In our earlier work [36], the simplest fountain code called *random linear fountain code* is used for simulation. In random linear codes, as the random vectors are completely randomly generated, the average degree is $\hat{k}_b/2$, and the encoding and decoding complexity is $O((\hat{k}_b)^2)$, which has a high computational cost. In [37], Luby Transform Codes (LT Codes) were introduced to reduce the density of the generator matrix by carefully designing the degree distribution $\rho(d)$ that represents the probability of degree $d$ ranging from 1 to $\hat{k}_b$. As shown in [7], the average degree of encoded symbols needs to be at least $\ln(\hat{k}_b)$ in order to recover all the original symbols at a low redundancy level, and this is achieved with LT Code. The encoding and decoding complexity of LT Code reduces to $O(\hat{k}_b \ln(\hat{k}_b))$.

In this paper, we adapt a more practical fountain code called *Raptor Code* [38], which further extends LT Code and reduces the encoding and decoding complexity to $O(\hat{k}_b)$. It combines the feature of rateless code and fix-rate code to achieve a good performance by concatenating a weakened LT code with an outer code. More specifically, the internal code is an LT code with the average degree constrained to a constant value about 3. The weakened LT code is not enough to recover the entire source symbols, but can recover most of them (i.e., 95%). To achieve a higher recovery rate, Raptor Code adds a fix-rate outer code. Shokrollahi [38] uses an irregular Low-Density Parity-Check code (LDPC) [39], [40] that can correct erasures at a known rate of 5% with a low decoding complexity for the outer code.

In Raptor Codes, a block containing $k_b$ symbols is first encoded into $\hat{k}_b$ symbols using LDPC code where $\hat{k}_b$ is slightly

larger than $k_b$. Next, the $\hat{k}_b$ encoded symbols are further encoded into $\overline{k}_b$ symbols using LT code. The number of encoded symbols, $\overline{k}_b$, is arbitrary and can change according to the dynamics of the channel. If the receiver has received $\hat{k}_b$ encoded symbols of LT code, the receiver is able to recover at least $k_b$ symbols at a very high possibility, which can be further decoded into the original data according to the nature of fix-rate coding. Note that the value of $\hat{k}_b$ depends only on the nature of LT code, not the dynamics of the channel.

Raptor Codes have been standardized to provide reliable delivery of data objects in RFC5053 [41], which has been further extended in RFC6330 [8]. In most cases, any set of encoding symbols with the cardinality equal to the number of source symbols are sufficient to recover the source block; in rare cases, a slightly larger number of encoding symbols are required. A Source Block Number (SBN) and an Encoding Symbol ID (ESI) of a symbol are enough for the receiver to recover the original data. The symbol size ranges from 1 to 65 535 B, and a wide range of block sizes (number of symbols) between 10 and 56 403 are also supported [8]. The symbol size and block size are constant for a block and can be negotiated between the sender and the receiver to achieve better efficiency.

In determining the size of a block that is the product of the symbol size $x_b$ and the number of symbols $k_b$ in a block, several constraints should be considered.

- *Maximum segment size*: Similar to TCP, FMTCP also employs the maximum segment size, MSS, to limit the size of a packet. The total size of the encoded symbols in a packet should be smaller than MSS to avoid the fragmentation of packets.
- *Buffer size*: Since all symbols of a block should be buffered until the block is decoded, the size of a block should be no more than the buffer size of the receiver.
- *End-to-end delay*: As data are delivered to the application by block, the end-to-end delay for applications grows with the increase of the block size.

When the data size from the application is small, we can choose a small symbol size and block size. Addition to that, additional padding symbols can be used as introduced in [8], which do not need to be actually sent. Similarly, if a source block cannot be divided into $k_b$ equal-length symbols, some padding bits are added to the last source symbol, which does not need to be included in the packet either.

## V. Data Allocation

In FMTCP, data allocation is needed to choose symbols from proper data blocks to generate a sending packet for the subflow requesting for packets. Both the number of encoded symbols generated for a block and the distribution of each subflow should be considered to improve the performance. In the following section, we first discuss the objectives and constraints of data allocation. We then provide a scheme to determine the redundancy level thus the additional number of symbols to send in a block to ensure more timely data transmissions. Finally, we present a data allocation algorithm.

### A. Analysis on Data Allocation

We use $F$ to represent the set of all the subflows. Each subflow $f$ maintains its congestion window size $\hat{w}_f$ and the retransmission timeout $RTO_f$. We denote $p_f$ and $RTT_f$ as the statistic loss probability and round-trip time of the subflow $f$, respectively.

Intuitively, to improve the efficiency of FMTCP, data allocation is expected to achieve the following objectives.

- *Low redundancy*: In order to increase the goodput, the sender should not transmit redundant encoded symbols to receiver once the receiver can recover the original block.
- *Decoding in order*: In order to ensure the delivery order, if a block $b_i$ is sent before a block $b_{i+1}$, $b_i$ should be recovered by the receiver before $b_{i+1}$. This is because only by recovering the first pending block can the receiver release its buffer in time and thus avoid the restriction on the aggregation data rate caused by the flow control.
- *High decoding probability*: In order to reduce the delivery delay for a block, the receiver should be able to recover the original data with a high probability. If the receiver is not able to recover the data, it needs to inform the sender to send more encoded symbols of the block, which will increase the delivery delay for the block and make the protocol not suitable for real-time applications.

However, it is very hard to achieve all the objectives mentioned above at the same time. For instance, in order to ensure that the receiver has a high probability of successful decoding, the sender should send more redundant data on a lossy channel, which conflicts with the first objective that forbids sending redundant data to save bandwidth. In order to ensure the delivery order, the sender should wait for all ACKs of a block $b_i$ before sending symbols of the block $b_{i+1}$, which is inefficient because of the stop-and-wait manner.

To handle these problems, we should set more practical objectives for FMTCP. First, instead of forbidding the sender from sending redundant data, the sender is allowed to send some redundant data on lossy channels as long as it can help significantly reduce the delivery delay of a block. Second, the sender should not work in a stop-and-wait manner. Once a block is completed, the sender should directly move on to the next block no matter whether all the ACKs of the last block have been received. FMTCP also employs a data allocation algorithm to adjust the order of arrival for different blocks.

To represent the quality for a receiver to recover a block $b$ once some symbols of $b$ are sent, we employ the following definition of *completeness* of a block.

*Definition 1:* A block $b$ is complete if and only if the number of symbols received for $b$ equals $\hat{k}_b$; otherwise, the block is pending.

The completeness of a block means that the receiver can recover the block with a very high probability. From Section IV, we known that $\hat{k}_b$ encoding symbols are sufficient to recover the source block. By the first objective of the data allocation scheme, the sender should not send more symbols once a block becomes complete.

However, in reality it is impractical to assume all paths are reliable; therefore, the sender should send more than $\hat{k}_b$ symbols to ensure a high decoding probability. Intuitively, one might suggest the sender send $\hat{k}_b/(1-p)$ symbols, assuming the loss rate of paths is $p$, as this would make the expected number of successfully-delivered symbols equal $\hat{k}_b$. However, this is not actually the case, because in most cases the sender cannot send all symbols of a block in one burst. Usually the sender waits for the ACK of previously sent symbols and then decides how

many more symbols should be sent for the block. This behavior makes $\hat{k}_b/(1-p)$ a pessimistic estimation and thus wastes the bandwidth.

To determine the number of appending symbols (i.e., the amount of redundant data sent for a block), we need to further analyze the behavior of TCP congestion control when a packet loss is detected. We model TCP congestion control in a way similar to [42].

TCP opens window and sends new packets upon receipt of an ACK that acknowledges new data. During continuous transmissions of data, most symbols lost will be detected by three duplicate ACKs after a round-trip time and then be supplemented. For delay-tolerant applications, the block size can be set much larger than the congestion window size of lossy channels to achieve a higher goodput, and symbols of a block would be sent during several round-trip time. Under this condition, symbols lost except those sent during the last round-trip time can be detected and supplemented in time. As symbols of the same block are mutually independent and without order, the lost symbols will not cause reordering and do little harm to our transmission of the whole block. Thus, we only need to append extra packets for symbols sent during the last round-trip time, which will be no more than the congestion window size. To achieve the minimum transferring delay, the block size can be set to be smaller than the congestion window size. In either case, we could consider the loss to be within the expected congestion window size. Therefore, we calculate the number of appending packets based on the congestion window size $\hat{w}_f$ instead of the block size $\hat{k}_b$.

We first give the definition of $\delta$-completeness of a block as follows.

*Definition 2:* A block $b$ is $\delta$-complete if and only if the decoding failure probability of $b$, $\delta_b$, is no more than $\delta$.

Note that the reason for decoding failure is not the nature of Raptor codes, but due to the random loss of packets on the channel.

We also employ *maximum acceptable decoding failure probability*, denoted by $\hat{\delta}$, as the threshold to predict whether a block can be successfully decoded by the receiver. If a block $b$ is $\hat{\delta}$-complete, the receiver is expected to be able to decode it, and thus the sender can stop sending symbols for $b$. For the estimation of decoding failure probability, we have the following theorem.

*Theorem 1:* If $l_b$ extra packets are sent for block $b$, the probability of decoding failure can be computed using

$$\delta_b(l_b) = \sum_{j>l_b} \binom{\hat{w}_f + l_b}{j} p_f^j (1-p_f)^{\hat{w}_f + l_b - j}. \tag{4}$$

*Proof:* When $l_b$ extra packets are appended for block $b$, the decoder can get no less than $\hat{k}_b$ packets, which is sufficient for decoding successfully if no more than $l_b$ packets out of $\hat{w}_f + l_b$ ones are lost.

Supposing the number of lost packets within $\hat{w}_f$ is $j$, then the probability of losing $j$ packets is

$$P = p_f^j (1-p_f)^{\hat{w}_f + l_b - j}. \tag{5}$$

Considering all combination of any $j$ packets out of $\hat{w}_f + l_b$ packets, the probability that the number of lost packets is $j$ is

$$P(j) = \binom{\hat{w}_f + l_b}{j} p_f^j (1-p_f)^{\hat{w}_f + l_b - j}. \tag{6}$$

When more than $j$ packets are lost, the decoding would fail, and extra packets are needed for transmission. By adding the probability of all cases in which $j > l_b$, we get (4), which represents the decoding failure probability. Hence, the proof is completed. ■

In order to ensure the block has the maximum acceptable decoding failure probability $\hat{\delta}$ and reduce the redundance, the value of $l_b$ should be set to the smallest integer, $l_b^*$, such that $\delta_b(l_b)$ is no more than $\hat{\delta}$.

The redundancy (i.e., the overhead of the coding scheme), $r_f$, is given using

$$r_f = \frac{\hat{k}_b + l_b}{k_b}. \tag{7}$$

The redundancy consists of both coding redundancy and extra packets appended on lossy channels. It is a measurement of the waste of bandwidth. We should make $r_f$ as small as possible while reducing delivery delay at the same time.

### *B. Data Allocation Algorithm*

An intuitive approach of data allocation is to assign encoded symbols in a greedy manner, i.e., transmit the first pending block until its expected failure probability gets lower than the acceptable failure threshold and then handle the next block.

Although this approach is able to meet the above requirements in a single-path transmission, it will not work efficiently for transmissions through multiple heterogeneous paths. For example, if the delay of the pending flow is very large compared to other flows, it would be unreasonable to assign the symbols of the first pending block to it. Thus, the performance of data allocation would be improved by predicting the transmission time of the subflows.

*Definition 3:* The *Expected Delivery Time* ($EDT$) for a path $f$ is defined as the expected time to send a packet successfully to the other end, using only path $f$.

The $EDT$ of flow $f$ with loss rate $p_f$, one-way delivery time $DT_f$, round-trip time $RTT_f$, and retransmission timeout $RTO_f$ can be estimated as

$$\begin{aligned} EDT_f &= \sum_{j=0}^{\infty} p_f^j (1-p_f) j RTO_f + DT_f \\ &= \frac{p_f RTO_f}{1-p_f} + DT_f. \end{aligned} \tag{8}$$

Here, we used $RTT_f/2$ to approximately estimate $DT_f$ for simplicity.

*Definition 4:* When a packet is sent on a path $f$, the time elapsed before it is acknowledged or its timer expires is defined as the *Response Time* ($RT$).

When a packet is sent, if it arrives successfully, $RT$ equals RTT; otherwise, $RT$ equals RTO. Thus, we can compute the expected $RT$ of subflow $f$ according to

$$RT_f = (1-p_f) RTT_f + p_f RTO_f \tag{9}$$

where $p_f$ is the loss rate of $f$.

*Definition 5:* After a packet arrives at the transport layer and is scheduled for transmission, the time elapsed before the allocated packet is successfully delivered to the other end on subflow $f$ is defined as the *Expected Arriving Time* ($EAT$) of the subflow $f$.

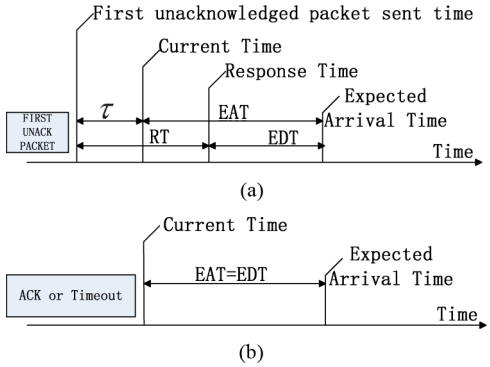



Fig. 5. EAT evaluation. (a) $w_f = 0$. (b) $w_f > 0$.

The $EAT$ of subflow $f$ can be calculated based on two cases. If the congestion window is not full, $EAT_f$ equals $EDT_f$. Otherwise, $EAT_f$ consists of $EDT_f$ and $RT_f$. Moreover, it should subtract $\tau_f$ (time elapsed since the first unacknowledged packet was sent). Thus, the $EAT$ of subflow $f$ can be expressed as

$$EAT_f = \begin{cases} EDT_f, & \text{if } w_f > 0 \\ EDT_f + RT_f - \tau_f, & \text{otherwise} \end{cases} \quad (10)$$

which is also illustrated in Fig. 5. Here, $w_f$ represents the *remaining* window space.

When some subflows request to send packets, $EAT$ is applied to compare the transmission time of all subflows and determine to which subflows to allocate symbols.

Based on the definition of $EDT$ and $EAT$, we design a data allocation algorithm as depicted in Algorithm 1. Blocks that can be simultaneously transmitted on the fly are called the pending blocks, denoted as a set $B = \{b_1, b_2, \ldots, b_n\}$. The number of pending blocks $n$ is the maximum number of blocks that can be stored in the receive buffer at the same time. The symbol size of block $b_i$ is denoted as $\hat{x}_{b_i}$. A subflow with remaining window space to send data is identified as the pending flow $f_p$ when it requests for a packet. Instead of fetching the earliest data in the buffer that has not been sent, Algorithm 1 is triggered to decide which symbols to send in the packet. The output vector $V = \{v_1, v_2, \ldots, v_n\}$ describes the data contents inside the allocated packet, in which $v_i$ denotes the number of symbols from block $b_i$. Then, the corresponding symbols are generated by the coding module and sent on subflow $f_p$ in a packet.

To achieve the objectives of data allocation in the algorithm, the flow with the smallest $EAT$ is *virtually* assigned to a packet (i.e., not really assigned the packet, and this process is only applied to facilitate scheduling) in each iteration until the pending flow is assigned a packet for transmission. After each iteration, the $EAT$ of the corresponding subflow is *virtually* updated. The packet description vector for $f_p$ is taken as the output of the algorithm to guide the packet construction , and only the packet allocated to $f_p$ is physically generated. Thus, when a loss is detected on a flow, the lost symbol(s) will be supplemented on the fastest flow that has the smallest $EAT$.

The allocation is virtual for subflows except $f_p$, as it is not necessary to physically generate symbols and update the EAT for other subflows. When one of these subflows has the transmission opportunity later, it will trigger the allocation algorithm and be assigned the symbols from the same or different blocks if the EAT change makes the allocation result different.

**Algorithm 1:** AllocatePacket $(f_p, F, B)$

**Input:** the pending subflow $f_p$; the set of subflows $F$; the set of pending blocks $B = \{b_1, b_2, \ldots, b_n\}$, and the $RT$, $\tau$ for each subflow;
**Output:** the description vector $V$ for the packet to send;

```
repeat
    f ← argmin_{g∈F} EAT_g
    X ← 0
    s ← 0
    i ← 0
    while (i < n) and (s ≤ MSS_f) do
        i ← i + 1
        v_i ← 0
        while (δ̃_{b_i} ≥ δ̂) and (s ≤ MSS_f) do
            v_i ← v_i + 1
            s ← s + x̂_{b_i}
        end while
    end while
    EAT_f ← EDT_f + RT_f − τ_f
until f = f_p
return V
```

Virtual allocation tries to get data received in order (in granularity of blocks). It uses an EDT to evaluate the overall performance of a subflow based on delay and loss rate, and further estimates the EAT of packets sent on a subflow by combining the state of the sending window. The appending subflow would send data from the appending blocks in the appropriate position instead of always the first pending block. With coding, packets within a data block are mutually independent from each other and can be transmitted out of order without having to wait until earlier packets have arrived. Hence, the very first block will not occupy the receiving buffer too long to block the transferring of following blocks. The lost data can also be recovered in time as the most urgent packet will be sent on the subflow with the smallest EAT that is expected to deliver the packet in the shortest time. Virtual allocation can thus help make better use of transmission capacity of all subflows without their blocking of transmissions from each other.

Our algorithm has good tolerance for RTT measurement inaccuracies and variations by adapting the transmissions accordingly. Subflows with similar qualities have the same priority in transmissions (i.e., their order will not affect the performance). Virtual allocation will work properly as long as the inaccuracy of RTT measurement is smaller than the difference in the subflow quality.

### C. Performance Analysis of the Data Allocation Scheme

In this section, we provide analysis to show that our data allocation scheme helps to reduce side effects due to low-quality transmission paths and path diversity, which will in turn improve the overall transmission quality of multipath TCP. We use $r_f$, $R_f$ to denote the round-trip time and RTO of a subflow $f$, respectively.

First, we want to show that symbols lost on a subflow will not be appended on a subflow with a lower quality. Thus, FMTCP will not be blocked by frequent loss on low-quality subflows. More specifically, we have the following theorem.

*Theorem 2:* If $EDT_i < EDT_j$, then symbols lost on the subflow $i$ will not be appended on the subflow $j$.

*Proof:* When a subflow $i$ requests to append symbols for a data block, its congestion window has space. Based on (10), $EAT_i = EDT_i < EDT_j \leq EAT_j$, thus this symbol will not be immediately appended on subflow $j$. If the appended symbols were not transmitted on subflow $i$, then there exists a subflow $k$ with $EAT_k < EAT_i$, where

$$EAT_k = EDT_k + RT_k - \tau_k < EAT_j. \tag{11}$$

Note that (11) remains valid until either subflow $j$ or subflow $k$ is updated with a packet transmission. Therefore, those appended symbols will not be transmitted on subflow $j$. ■

Next, we want to show that the $EDT$ used in the data allocation algorithm can reflect the actual path quality to some extent. We define *Expected Delivery Time of Packet* ($PEDT$) to estimate the transmission time of a packet.

*Definition 6:* When a packet is generated and scheduled for transmission, the expected time needed to successfully transmit the packet is defined as the $PEDT$ of the packet.

$PEDT$ reflects the expectation of the time needed to successfully transmit a packet in MPTCP which contains retransmission on the original path in IETF-MPTCP and supplement on the best path in our FMTCP. Thus, $PEDT_f$ equals $EDT_f$ in IETF-MPTCP, but is no larger than $EDT_f$ in FMTCP.

*Theorem 3:* If $EDT_i < EDT_j$, then $PEDT_i < PEDT_j$.

*Proof:* We use induction to prove this theorem. If there is only one path, then the theorem is obviously correct. We assume the theorem is correct for $n$ paths. Next, we want to prove the theorem is also correct for $n+1$ paths.

Without the loss of generality, we assume

$$EDT_1 < EDT_2 < \cdots < EDT_{n+1}.$$

Using the induction hypothesis on subflow 2 to $n+1$, we have the following inequality:

$$PEDT_2 < PEDT_3 < \cdots < PEDT_{n+1}. \tag{12}$$

Since $EDT_1 < EDT_2$, according to Theorem 1, symbols lost on subflow 1 will only be appended on subflow 1, while symbols lost on subflow 2 may be appended on either subflow 2 or subflow 1. Thus, $PEDT_1 = EDT_1$. When symbols lost on subflow 2 are appended on subflow 2, $PEDT_2 = EDT_2$. When symbols lost on subflow 2 are appended on subflow 1, $PEDT_2$ is

$$\begin{aligned} PEDT_2 &= (1-p_2)\frac{r_2}{2} + p_2(R_2 + EDT_1) \\ &= (1-p_2)(\frac{r_2}{2} + \frac{p_2 R_2}{1-p_2}) + p_2 EDT_1 \\ &= (1-p_2)EDT_2 + p_2 EDT_1. \end{aligned}$$

In both cases we can see $PEDT_2 > EDT_1$, therefore

$$PEDT_1 < PEDT_2. \tag{13}$$

Combining (12) and (13), we complete the proof. ■

Theorem 3 shows that $EDT$ is a proper estimation on the quality of subflows in FMTCP.

Finally, we give an analysis to show that the allocation scheme of FMTCP helps to reduce the time difference of symbols transmitted on different flows, which further reduces the disorder of packet arrival. Our FMTCP's advantage over conventional multipath TCP increases as the diversity of path grows.

We assume there are two subflows. The round-trip time and loss rate of the two paths are $r_1, p_1, r_2, p_2$, respectively. We also assume $r_1 \approx R_1$, $r_2 \approx R_2$, where $R_1, R_2$ are RTO of both subflows. Without the loss of generality, we assume that flow 2 is the inferior flow. More precisely, $EDT_2/EDT_1 = m$, and $m > 1$. $m$ represents the diversity of paths.

*Lemma 1:* If the following equation is satisfied:

$$m > \frac{3-p_1}{1+p_1} \tag{14}$$

then symbols lost on subflow 2 will only be appended on subflow 1.

*Proof:* Using (8) and supposing that $r_2 \approx R_2$, we can estimate $EDT_2$ with

$$EDT_2 = \frac{p_2 R_2}{1-p_2} + \frac{r_2}{2} \approx \frac{1+p_2}{2(1-p_2)} r_2. \tag{15}$$

With (10), $EAT_1$ can be bounded as follows:

$$\begin{aligned} EAT_1 &\leq EDT_1 + RT_1 - \tau_1 \\ &\leq EDT_1 + R_1 \\ &\approx \frac{EDT_2}{m} + r_1. \end{aligned}$$

Hence, if (14) is satisfied, we can deduce that

$$EAT_1 < EDT_2$$

which means the symbols lost on subflow 2 will only be transmitted on subflow 1. ■

*Theorem 4:* Let $T_1$, $T_2$ denote the time elapsed before the receiver successfully receives one symbol from a block, which is initially transmitted on subflow 1 and subflow 2, respectively. Then, the ratio of $E(T_2)$ and $E(T_1)$ is bounded as follows when (14) is satisfied:

$$\frac{E(T_2)}{E(T_1)} \leq p_2 \frac{3-p_1}{1+p_1} + (1-p_2)m. \tag{16}$$

*Proof:* $E(T_2)$ can be bounded as

$$\begin{aligned} E(T_2) &= (1-p_2)\frac{r_2}{2} + p_2(R_2 + E(T_1) + E(RT_1 - \tau_1)) \\ &\leq (1-p_2)\frac{r_2}{2} + p_2(R_2 + E(T_1) + R_1). \end{aligned} \tag{17}$$

Using (15), we can estimate $m$ as follows:

$$m = EDT_2/EDT_1 \approx \frac{(1+p_2)(1-p_1)r_2}{(1+p_1)(1-p_2)r_1}.$$

Plugging $E(T_1) = EDT_1 = PEDT_1$ and $m$ into (17), we get

$$\frac{E(T_2)}{E(T_1)} \leq p_2 \frac{3-p_1}{1+p_1} + (1-p_2)m.$$

■

In IETF-MPTCP, a packet will continuously be transmitted over the same subflow until it is successfully received, so the ratio is exactly $m$. When (14) is not satisfied, the ratio of FMTCP is not larger than MPTCP. Otherwise, the ratio of FMTCP is bounded as (16), which is smaller than $m$. Therefore,

as the diversity of path ($m$) grows, FMTCP shows its advantage over the conventional multipath TCP.

## VI. Simulation Results

In this section, we evaluate our FMTCP solution through ns-2 simulations. In our simulation, we employ a topology with two nodes connecting with each other through two disjoint paths as the same in Section III.

We first conduct a series of experiments to determine the packet appending rate. Then, we study the impact of buffer size and compare the overall performance of FMTCP to that of IETF-MPTCP [26]. Finally, we make further analysis on the effect of rateless coding and scheduling separately.

The metrics we take to evaluate the performance include goodput, delivery delay, and jitter. Note that the delivery delay of each packet is not a good metric to evaluate the performance of FMTCP because the receiver cannot decode with the data from a single packet. What is more, the transfer delay on the path just reflects the quality of the transmission links, while the performance of transmission protocols should be evaluated though continuous transmission of some amount of data. Therefore, we measure the delay by block and define the delivery delay of a block as the duration from the transmission of its first encoded symbol to the time when the block is decoded successfully by the receiver and ready to be handed to the application layer. The jitter is also measured by the block. For a fair comparison, we partition the data streams transmitted by IETF-MPTCP into blocks of the same length as that of FMTCP and measure the delay and jitter accordingly.

### A. Appending Packets

As explained earlier, it is crucial to estimate the number of packets lost when transmitting a block in order to determine the number of packets needed to append for each block (i.e., the redundancy in transmission) according to the loss rate of the paths. If the redundancy is too small, the receiver cannot recover the original block with a high probability, and the delivery delay for a block increases. If the redundancy is too large, the bandwidth is wasted, and the total goodput decreases. In (4), we give a formula to estimate the probability of the decoder being unable to recover the original block, given the number of appending packets for block $b$, $l_b$, and the loss rate of paths, $p_f$. In this section, we use simulation to display the number of lost packets when transmitting a block under different loss rates.

We record the number of lost packets of blocks and present the cumulative distribution (i.e., the percentage of blocks that lose no more than $x$ packets) in Fig. 6. In the implementation of the protocol, we set the value of $\hat{\delta}$ to be 5%. In this setting, using (4), and $\delta_b(l_b^*) < \hat{\delta}$, we obtain that $l_b^* = 1$ for loss rate from 1% to 5% and $l_b^* = 2$ for loss rate 10%. This result is consistent with Fig. 6, where we observe that when the loss rate of the path 2 is not larger than 5%, putting $l_b = 1$ we can obtain $\delta_b(l_b)$ smaller than $\hat{\delta}$. However, when the loss rate increases to 10%, we need to set $l_b$ to 2 in order to ensure the failure probability of delivery is smaller than $\hat{\delta}$.

We then study the impact of different appending rates of a block on the average delivery delay of blocks and the goodput in FMTCP. If the rate is low, the receiver will not be able to recover the original block in most cases. Therefore, the sender should resend symbols, and the delivery delay is large. However, if the appending rate is large, the bandwidth will be wasted because the sender sends a large number of redundant packets and the goodput will reduce. From Fig. 7(a), we can see that when the appending rate is small, the average delivery delay decreases sharply as we increase the appending rate. However, when the appending rate is large, the average delivery delay remains almost the same despite us increasing the appending rate. Fig. 7(b) shows the goodput of FMTCP under different appending rates. For each loss rate, the goodput first increases with the appending rate and reaches a peak point, beyond which the goodput will reduce as the number of appending packets increases further. The optimal appending rate corresponding to the maximum goodput increases as the loss rate increases.

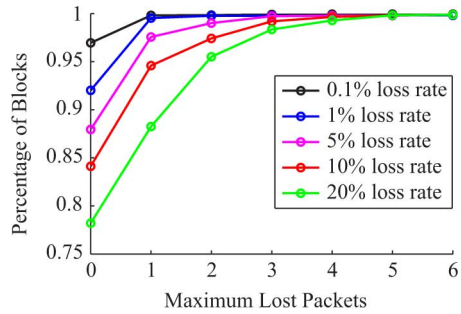



Fig. 6. Average number of lost packets on the fly for one block. Loss rate of one flow is constant (0.1% packet loss), while the loss rate of the other flow changes according to the legend.

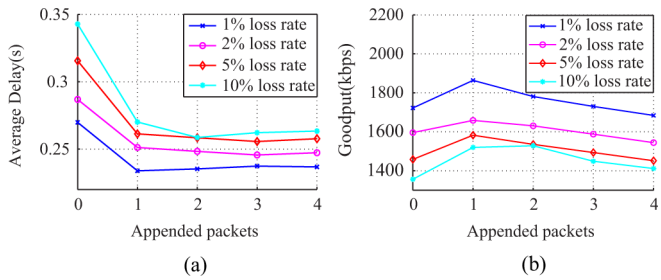



Fig. 7. Average delivery delay and goodput under different appending rate settings. The parameters of the subflow 1 remain the same (100 ms delay, 0.1% packet loss). The path delay of subflow 2 is 100 ms, while the loss rate changes according to the legend. (a) Delay. (b) Goodput.

In practice, we need to choose the proper appending rate under different path-loss rates. As introduced earlier in Section V, a possible choice of appending rate is $l_b^*$, which is the smallest integer such that $\delta(l_b^*) < \hat{\delta}$, where $\hat{\delta}$ is the maximum acceptable decoding failure probability. In the implementation, we set $\hat{\delta}$ to be 5%, which is reasonable because Fig. 7(a) shows that when $l_b$ is larger than 2, the average delivery delay remains almost unchanged while the goodput decreases as we increase the appending rate.

### B. Buffer Size

In the following sections, we employ the same methodology as in Section III, i.e., to simulate the heterogeneity of different subflows by adjusting the quality of one path, while keeping the parameters of the other one constant.

As explained in Section III, in a network with heterogeneous quality of links, packets lost on inferior flows (with higher loss rate or delay) may block the transmissions in good paths, which

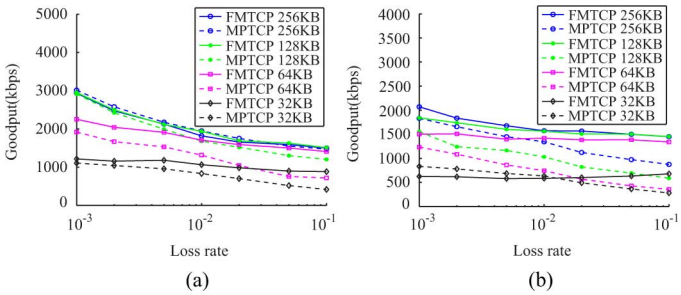



Fig. 8. Total goodput of FMTCP and MPTCP under various buffer limitations. The parameters of the subflow 1 remain the same (100 ms delay, 0.1% packet loss). The delay of the subflow 2 is set to 100 ms/300 ms, and the loss rate is between 0.1% and 10%. (a) $Delay2 = 100$ ms. (b) $Delay2 = 300$ ms.

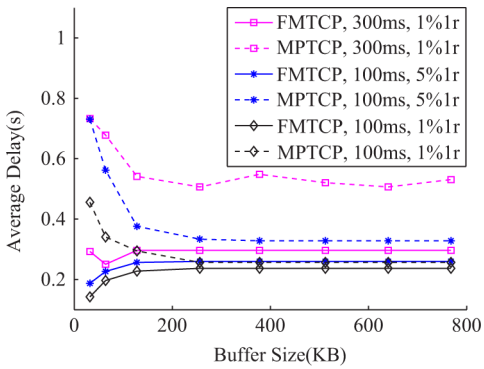



Fig. 9. Average delay of FMTCP and MPTCP under various buffer limitations. The parameters of the subflow 1 remain the same (100 ms delay, 0.1% packet loss). The delay and loss rate of the subflow 2 change according to the legend.

consequently significantly degrades the aggregate goodput of the multipath connection. The impact on goodput is more severe when the buffer size is limited. To store all incoming packets on all subflows of the same connection during the retransmission or timeout period of the subflow with the largest RTT, the minimum buffer size needed may be large according to (1) and (2). We introduce Fountain Codes to reduce the reordering and avoid waiting for urgent packets that need to be retransmitted over the original path with the highest RTT/RTO as observed in IETF-MPTCP.

We present the goodput of FMTCP and IETF-MPTCP under different buffer sizes in Fig. 8. To set the block size 32 kB, we keep the buffer size no less than 32 kB. We can see that FMTCP can work well with much smaller buffer size. When the buffer size is slightly bigger than the expected congestion window size, it is enough for FMTCP to achieve as high goodput as IETF-MPTCP with a large buffer size. From Fig. 9, we can see that the delay of FMTCP does not increase, while the delay of IETF-MPTCP increases sharply as the receiving buffer limit goes down. When path delays of subflows are comparable as the cases in Fig. 8, MPTCP can achieve as good performance as FMTCP and even better if the buffer size is large enough and the loss rates are small. However, in reality, multipath TCP would need to handle various and varying transmission qualities from different networks, especially wireless networks. Hence, FMTCP can be useful in many scenarios.

When the buffer size is extremely small, the goodputs of IETF-MPTCP and FMTCP are rather low even when the loss rate is very small. This is because the buffer size is even smaller

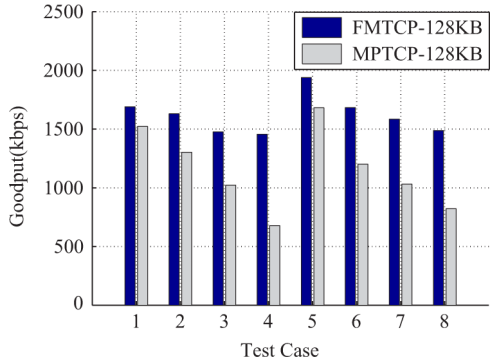



Fig. 10. Goodput comparison between FMTCP and IETF-MPTCP with the variation of subflow 2. The parameters of the subflow 1 remain the same (100 ms delay, 0.1% packet loss). The delay and loss rate of the subflow 2 are shown in Table II.

TABLE II
PATH PARAMETERS OF SUBFLOW 2

| Test Case | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Delay (ms) | 100 | 100 | 100 | 100 | 100 | 200 | 300 | 500 |
| Loss Rate (%) | 2 | 5 | 20 | 50 | 1 | 1 | 1 | 1 |

than the expected congestion window size, and the window size becomes the main restriction of the actual achieved bandwidth instead of the loss rate. When delay of subflow 2 is 300 ms, the goodput of FMTCP is even smaller than IETF-MPTCP under some conditions. When the block size is set equal to the buffer size, new blocks cannot be sent until the last block can be decoded. In this scenario, delay of a block may be smaller as shown in Fig. 9, but as it works in a stop-and-wait model, the total goodput decreases. Even so, when the loss rate gets bigger, FMTCP performs better than IETF-MPTCP, and we can see that the total goodput of FMTCP is not affected much by the limitation on the receiving buffer, while the total goodput of IETF-MPTCP degrades significantly as the receiving buffer limit goes down, especially when the diversity of delay among subflows is distinct such as in Fig. 8(b).

### C. Goodput, Delay, and Jitter Comparison

To study the impact of path diversity, we set the delay of subflow 1 to 100 ms and packet loss to 0.1%, while varying the loss rate and delay of subflow 2 according to Table II. The buffer size is set to 128 kB under all these conditions.

In Fig. 10, we observe that FMTCP outperforms IETF-MPTCP in all cases even when the buffer size is 128 kB. When the difference of either delay or loss rate of the two paths increases, the goodput difference between FMTCP and IETF-MPTCP also becomes larger. For instance, when the loss rate of subflow 2 varies from 1% to 50% in Test Cases 1–5, the goodput of IETF-MPTCP has up to 57.9% degradation, while the goodput of FMTCP only decreases slightly. When the delay of subflow 2 varies from 100 to 500 ms in Test Cases 5–8, FMTCP also significantly outperforms IETF-MPTCP.

The high degradation of IETF-MPTCP indicates that the performance of high-quality subflows is seriously influenced by the low-quality ones under IETF-MPTCP scheme. As the delay and loss rates go up, the low-quality subflows become the bottleneck of IETF-MPTCP, and the total goodput is reduced.

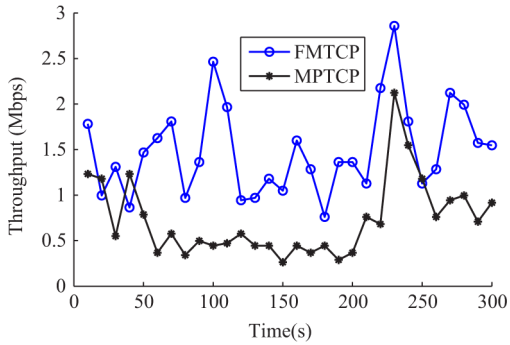



Fig. 11. Comparison between FMTCP and IETF-MPTCP. The loss rate of the subflow 2 surges to $p$ at 50 s and returns to 1% at 200 s.

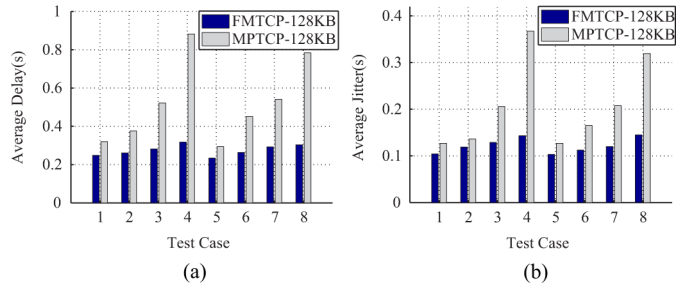



Fig. 12. Average delivery delay and jitter of blocks of FMTCP and IETF-MPTCP. The parameters of the subflow 1 remain the same (100 ms delay, 0.1% packet loss). The delay and loss rate of the subflow 2 are shown in Table II. (a) Delay. (b) Jitter.

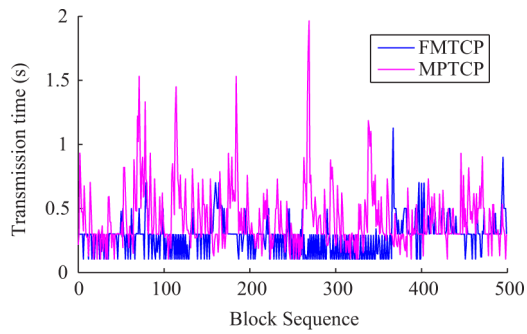



Fig. 13. Delivery delay of blocks of FMTCP and IETF-MPTCP. The parameters of subflow 1 remain the same (100 ms delay, no packet loss). The delay of subflow 2 is 100 ms, while the loss rate is 10%.

Fig. 8(a) and (b) gives more detailed pictures to show how the total goodput varies when the loss rate of the subflow 2 increases. We observe that the total goodput of FMTCP reduces only slightly, while the goodput of MPTCP has a sharp reduction as the lost rate goes up.

There are two main reasons for FMTCP to outperform the conventional IETF-MPTCP. First, when some packets are lost, IETF-MPTCP subflows have to retransmit lost packets. If some necessary packets have not arrived, other subflows must stop and wait for them. However, FMTCP only needs to append some new symbols of that block instead of retransmitting the lost symbols and all the symbols sent after it. Therefore, transmissions from inferior subflows will not block those from good ones, and the total goodput is improved. Second, FMTCP employs a data-allocation algorithm that predicts the arriving order of different symbols. The scheduling algorithm helps to reduce the disorder of packets that are transmitted on different paths. Moreover, the scheduling algorithm together with the redundant coding scheme helps to alleviate the effect as a result of the inferior subflows blocking the good ones. In Fig. 8(a) and (b), the quality of the inferior subflow does not have much impact on the total goodput.

We further study the stability of the goodput by using a connection with two subflows for both IETF-MPTCP and FMTCP. The loss rate of subflow 2 varies over time, thus the performance of FMTCP under bursty packet losses is studied here. In Fig. 11, the delay of both paths is set to 100 ms. The initial loss rate of both paths is 1%. At 50 s, the loss rate of the subflow 2 surges to 35%. At 200 s, the loss rate of subflow 2 returns to 1%.

We observe that IETF-MPTCP also performs worse than FMTCP when the quality of subflows changes rapidly. The total goodput rate of IETF-MPTCP fluctuates severely under the loss surge of subflow 2. However, FMTCP can adapt to this variation well, and the goodput rate is much more stable.

We then study the delivery delay and jitter of FMTCP, which are used to measure the efficiency and stability. These metrics are especially important for applications such as real-time multimedia transmissions.

Fig. 12(a) shows the average delivery delay of the blocks of FMTCP and IETF-MPTCP in different scenarios. It is clear that the blocks transmitted with IETF-MPTCP experience higher delivery delays than those transmitted by FMTCP. When the difference of either delay or loss rate of the two paths increases, the delay of a block increases sharply in IETF-MPTCP, but only changes slightly in FMTCP. This is because FMTCP, with its data allocation mechanism, is able to send the most urgent packets as soon as possible. In other words, if the receiver needs more symbols to decode the very first pending block, FMTCP would transmit the required symbols via the path with the lowest $EAT$, which reduces the waiting time for decoding the block. Therefore, the delivery delay of a block is reduced. Particularly, when the path quality falls, the delivery delay of IETF-MPTCP grows considerably as the impact of those low-quality subflows is amplified.

We also investigate the jitter of the blocks of the two protocols, and the results are in Fig. 12(b). The difference in jitter performance is even larger than that of the delay difference, especially when the quality of one subflow is very low. The reason is similar to the delay comparison case. FMTCP can assign the urgent packets to the high-quality subflow, while IETF-MPTCP does not consider the path quality. More specifically, assigning an urgent packet to different subflows leads to different delivery delay, and the gap becomes larger as the quality of one flow gets lower. FMTCP always try to send the packets through the high-quality subflow so the delivery delay remains stable; IETF-MPTCP does not have this feature, and therefore the delivery delay varies according to the path selected.

We make a more detailed examination of the results through another simulation. Fig. 13 shows the delivery delay over time where the loss rate of subflow 2 is high (10%). We can see extremely high fluctuations on the curve of IETF-MPTCP, with the highest delivery delay five times that of the average value, while the delay of FMTCP is much more stable. Clearly, those high delay bursts are caused by the large loss rate of the subflow 2. Because IETF-MPTCP cannot avoid allocating urgent

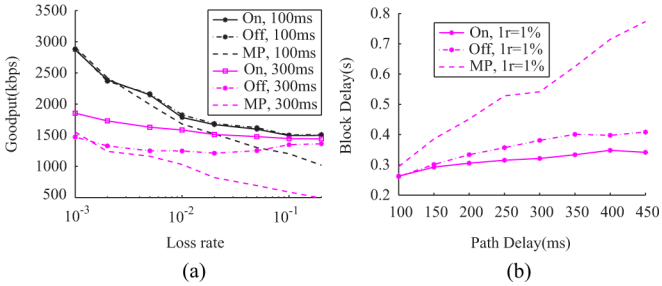



Fig. 14. Goodput gain and delay reduction of rateless coding and scheduling in FMTCP. The buffer size is set to 128 kB. The parameters of subflow 1 remain the same (100 ms delay, 0.1% packet loss). The loss rate and delay of subflow 2 change according to the legend. (a) Goodput. (b) Delay.

packets to the subflow 2, it often suffers from the high delivery delay, which eventually increases its jitter.

### *D. Performance Impact of Rateless Coding and Scheduling*

To further understand the different impacts on performance due to rateless coding and our scheduling algorithm, we run the simulation with and without scheduling and compare their performance.

Fig. 14(a) shows the goodput of FMTCP with the scheduling on and off. When the delay of the subflows is equal, i.e., at 100 ms, the goodput of FMTCP with scheduling on and off is roughly the same at different loss rates, which indicates that scheduling will not play the major role when there is no difference in subpath delay. This can be further proved when the delay of the path 2 is 300 ms, which is different from that of path 1, in which case FMTCP scheduling is observed to effectively increase the throughput. From the performance results, rateless coding can help mitigate the problems due to packet losses and reduce the impact of inferior subflows on the overall multipath TCP performance as we mentioned before, but it cannot make full use of different paths when they have very different delay. Our scheduling algorithm further helps reduce the block delay to achieve a higher goodput. Instead of sending data in order, our scheduling attempts to make data received on different paths in order. Due to the Additive Increase Multiplicative Decrease (AIMD) feature of loss-based TCP congestion algorithm, the average window size as well as the throughput of subflows decrease sharply as the loss rate increases. When the loss rate of subflow 2 is over 10%, the low goodput of TCP makes the goodput gain from scheduling small.

We further study the block delay with results shown in Fig. 14(b). When the path delay difference increases, the block delay of FMTCP with scheduling is obviously lower than that without scheduling as expected, which demonstrates the effectiveness of our scheduling algorithm in improving the transmission quality in the presence of path delay difference.

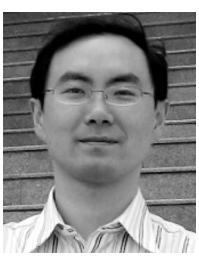
## VII. Conclusion

In this paper, we propose FMTCP, an extension to TCP, to enable efficient TCP transmissions in multi-interface networks. We employ fountain code to encode transmission data and take advantage of its random coding scheme to avoid retransmissions in MPTCP. Taking advantage of the features of fountain code, we propose an allocation algorithm to flexibly allocate encoded symbols to different subflows based on the expected packet arrival time over different paths. Both theoretical analysis and simulation results demonstrate that FMTCP alleviates the bottleneck problem due to the quality diversity of multiple paths and outperforms MPTCP in various scenarios. In addition, FMTCP has stable performance when there is a burst of packet loss. The low transmission delay and jitter of using FMTCP help to better support multimedia transportation and other types of real-time applications.

## References

[1] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proc. 8th USENIX NSDI*, 2011, p. 8.
[2] S. Barré, C. Paasch, and O. Bonaventure, "Multipath TCP: from theory to practice," in *Proc. Netw.*, 2011, pp. 444–457.
[3] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses," RFC6824, 2013 [Online]. Available: http://tools.ietf.org/html/rfc6824
[4] 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 *Proc. 9th USENIX NSDI*, 2012, vol. 12, p. 29.
[5] H. Hsieh and R. Sivakumar, "A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts," *Wireless Netw.*, vol. 11, no. 1-2, pp. 99–114, 2005.
[6] K. Kim, Y. Zhu, R. Sivakumar, and H. Hsieh, "A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces," *Wireless Netw.*, vol. 11, no. 4, pp. 363–382, 2005.
[7] D. MacKay, "Fountain codes," *IEE Proc., Commun.*, vol. 152, no. 6, pp. 1062–1068, 2005.
[8] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder, "Raptorq forward error correction scheme for object delivery," RFC6330, 2011 [Online]. Available: http://tools.ietf.org/html/rfc6330
[9] J. Iyengar, P. Amer, and R. Stewart, "Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 951–964, Oct. 2006.
[10] T. Dreibholz, M. Becke, H. Adhari, and E. Rathgeb, "On the impact of congestion control for concurrent multipath transfer on the transport layer," in *Proc. 11th IEEE ConTEL*, 2011, pp. 397–404.
[11] Y. Cao, M. Xu, and X. Fu, "Delay-based congestion control for multipath TCP," in *Proc. 20th IEEE ICNP*, 2012, pp. 1–10.
[12] Y. Cao, M. Xu, X. Fu, and E. Dong, "Explicit multipath congestion control for data center networks," in *Proc. 9th ACM CoNEXT*, 2013, pp. 73–84.
[13] M. Becke, T. Dreibholz, H. Adhari, and E. Rathgeb, "On the fairness of transport protocols in a multi-path environment," in *Proc. IEEE ICC*, Ottawa, ON, Canada, 2012, pp. 2666–2672.
[14] Y. Cui, X. Ma, H. Wang, I. Stojmenovic, and J. Liu, "A survey of energy efficient wireless transmission and modeling in mobile cloud computing," *Mobile Netw. Appl.*, vol. 18, no. 1, pp. 148–155, 2013.
[15] C. Lai, K. Leung, and V. Li, "Enhancing wireless TCP: a serialized-timer approach," in *Proc. IEEE INFOCOM*, 2010, pp. 1–5.
[16] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, and R. Wang, "TCP westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proc. 7th Annu. ACM MobiCom*, 2001, pp. 287–297.
[17] Y. Huang, M. Ghaderi, D. Towsley, and W. Gong, "TCP performance in coded wireless mesh networks," in *Proc. 5th Annu. IEEE SECON*, 2008, pp. 179–187.
[18] Y. Li, Y. Zhang, L. Qiu, and S. Lam, "Smarttunnel: Achieving reliability in the Internet," in *Proc. 26th IEEE INFOCOM*, 2007, pp. 830–838.
[19] J. Sundararajan, D. Shah, M. Médard, M. Mitzenmacher, and J. Barros, "Network coding meets TCP," in *Proc. IEEE INFOCOM*, 2009, pp. 280–288.
[20] Y. Lee, I. Park, and Y. Choi, "Improving TCP performance in multipath packet forwarding networks," *J. Commun. Netw.*, vol. 4, no. 2, pp. 148–157, 2002.
[21] J. Chen, K. Xu, and M. Gerla, "Multipath TCP in lossy wireless environment," in *Proc. 3rd Annu. IFIP Med-Hoc-Net*, 2004, pp. 263–270.
[22] G. Kwon and J. W. Byers, "ROMA: Reliable overlay multicast with loosely coupled TCP connections," in *Proc. 23rd Annu. IEEE INFOCOM*, 2004, vol. 1.

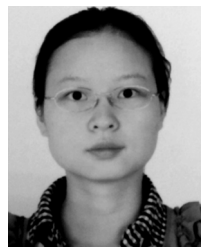
[23] H. Han, S. Shakkottai, C. Hollot, R. Srikant, and D. Towsley, "Multi-path TCP: a joint congestion control and routing scheme to exploit path diversity in the Internet," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1260–1271, Dec. 2006.
[24] R. Stewart, "Stream control transmission protocol," RFC4960, 2007.
[25] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?," in *Proc. ACM SIGCOMM IMC*, 2011, pp. 181–194.
[26] A. Ford, C. Raiciu, M. Handley, S. Barré, and J. Iyengar, "Architectural guidelines for multipath TCP development,," RFC 6182, 2011.
[27] C. Pluntke, L. Eggert, and N. Kiukkonen, "Saving mobile device energy with multipath TCP," in *Proc. 6th ACM MobiArch*, 2011, pp. 1–6.
[28] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," *Comput. Commun. Rev.*, vol. 41, no. 4, pp. 266–277, 2011.
[29] C. Raiciu, D. Niculescu, M. Bagnulo, and M. J. Handley, "Opportunistic mobility with multipath TCP," in *Proc. 6th ACM MobiArch*, 2011, pp. 7–12.
[30] J. Iyengar, P. Amer, and R. Stewart, "Performance implications of a bounded receive buffer in concurrent multipath transfer," *Comput. Commun.*, vol. 30, no. 4, pp. 818–829, 2007.
[31] T. Dreibholz, R. Seggelmann, M. Tüxen, and E. Rathgeb, "Transmission scheduling optimizations for concurrent multipath transfer," in *Proc. PFLDNeT*, Nov. 2010.
[32] T. Dreibholz, M. Becke, E. Rathgeb, and M. Tuxen, "On the use of concurrent multipath transfer over asymmetric paths," in *Proc. IEEE GLOBECOM*, 2010, pp. 1–6.
[33] V. Sharma, S. Kalyanaraman, K. Kar, K. Ramakrishnan, and V. Subramanian, "MPLOT: A transport protocol exploiting multipath diversity using erasure codes," in *Proc. 27th IEEE INFOCOM*, 2008, pp. 121–125.
[34] Y. Hwang, B. Obele, and H. Lim, "Multipath transport protocol for heterogeneous multi-homing networks," in *Proc. ACM CoNEXT Student Workshop*, 2010, p. 5.
[35] P. A. Iannucci, J. Perry, H. Balakrishnan, and D. Shah, "No symbol left behind: a link-layer protocol for rateless codes," in *Proc. 18th Annu. ACM MobiCom*, 2012, pp. 17–28.
[36] Y. Cui, X. Wang, H. Wang, G. Pan, and Y. Wang, "FMTCP: A fountain code-based multipath transmission control protocol," in *Proc. 32nd ICDCS*, 2012, pp. 366–375.
[37] M. Luby, "LT codes," in *Proc. 43rd Annu. IEEE FOCS*, 2002, pp. 271–280.
[38] A. Shokrollahi, "Raptor codes," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2551–2567, Jun. 2006.
[39] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.
[40] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 638–656, Feb. 2001.
[41] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer, "Raptor forward error correction scheme for object delivery," RFC5053, 2007 [Online]. Available: http://tools.ietf.org/html/rfc5053
[42] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," *Comput. Commun. Rev.*, vol. 28, no. 4, pp. 303–314, 1998.



**Yong Cui** (M'10) received the B.E. and Ph.D. degrees in computer science and engineering from Tsinghua University, Beijing, China, in 1999 and 2004, respectively.

He is currently a Full Professor with Tsinghua University and Co-Chair of IETF IPv6 Transition WG Softwire. He has published more than 100 papers in refereed journals and conferences, holds more than 40 patents, and authored three Internet standard documents, including RFC 7040 and RFC 5565, for his proposal on IPv6 transition technologies. His major research interests include mobile wireless Internet and computer network architecture.

Prof. Cui serves on the Editorial Board of both the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS and the IEEE TRANSACTIONS ON CLOUD COMPUTING. He received the National Award for Technological Invention in 2013, the Influential Invention Award of China Information Industry in both 2012 and 2004, and the Best Paper Award in ACM ICUIMC 2011 and WASA 2010.



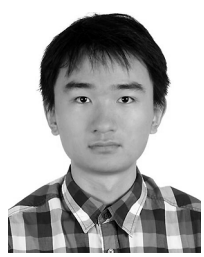
**Lian Wang** received the B.E. degree in computer science from Tsinghua University, Beijing, China, in 2012, and is currently pursuing the master's degree in computer science and technology at Tsinghua University, supervised by Prof. Yong Cui.

Her research interests include computer network architecture and wireless networks.

**Xin Wang** (M'01) received the B.S. and M.S. degrees in telecommunications engineering and wireless communications engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 1990 and 1993, respectively, and the Ph.D. degree in electrical and computer engineering from Columbia University, New York, NY, USA, in 2001.

She is currently an Associate Professor with the Department of Electrical and Computer Engineering, State University of New York at Stony Brook, Stony Brook, NY, USA. Before joining Stony Brook, she was a Member of Technical Staff in the area of mobile and wireless networking with Bell Labs Research, Lucent Technologies, Holmdel, NJ, and an Assistant Professor with the Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, NY, USA. Her research interests include algorithm and protocol design in wireless networks and communications, mobile and distributed computing, as well as networked sensing and detection.

Dr. Wang has served in executive committees and technical committees of numerous conferences and funding review panels and is the referee for many technical journals. She received the NSF Career Award in 2005 and the ONR Challenge Award in 2010.



**Hongyi Wang** received the B.E. degree in computer science from Tsinghua University, Beijing, China, in 2009, and is currently pursuing the master's degree in computer science and technology at Tsinghua University, supervised by Prof. Yong Cui.

His research interests include data center networking, wireless networks, and mobile system.

Mr. Wang won the Best Paper Award at ACM ICUIMC 2011.

**Yining Wang** is an undergraduate student with the Institute of Interdisciplinary Information Sciences, Tsinghua University, Beijing, China.

His research interests include wireless networks, artificial intelligence, and machine learning and its application in natural language processing.