Tolerating path heterogeneity in multipath TCP with bounded receive buffers

Ming Li,⇑ Andrey Lukyanenko, Sasu Tarkoma, Yong Cui, Antti Ylä-Jääski

Article history:
Received 8 April 2013
Received in revised form 14 November 2013
Accepted 28 January 2014
Available online 6 February 2014

Keywords:
TCP
MPTCP
Linear systematic coding
Head-of-line blocking
NS-3

ABSTRACT

With bounded receive buffers, the aggregate bandwidth of multipath transmission degrades significantly in the presence of path heterogeneity. The performance could even be worse than that of single-path TCP, undermining the advantage gained by using multipath transmit. Furthermore, multipath transmission also suffers from delay and jitter even with large receive buffers.

In order to tolerate the path heterogeneity when the receive buffer is bounded, we propose a new multipath TCP protocol, namely SC-MPTCP, by integrating linear systematic coding into MPTCP. In SC-MPTCP, we make use of coded packets as redundancy to counter against expensive retransmissions. The redundancy is provisioned into both proactive and reactive data. Specifically, to send a generation of packets, SC-MPTCP transmits proactive redundancy first and then delivers the original packets, instead of encoding all sent-out packets as all the existing coding solutions have done. The proactive redundancy is continuously updated according to the estimated aggregate retransmission ratio. In order to avoid the proactive redundancy being underestimated, the pre-blocking warning mechanism is utilized to retrieve the reactive redundancy from the sender.

We use an NS-3 network simulator to evaluate the performance of SC-MPTCP with and without the coupled congestion control option. The results show that with bounded receive buffers, MPTCP achieves less than 20% of the optimal goodput with diverse packet losses, whereas SC-MPTCP approaches the optimal performance with significantly smaller receive buffers. With the help of systematic coding, SC-MPTCP reduces the average buffer delay of MPTCP by at least 80% in different test scenarios. We also demonstrate that the use of systematic coding could significantly reduce the arithmetic complexity compared with the use of non-systematic coding.

1. Introduction

In today’s Internet, users expect to concurrently utilize multiple access interfaces to get higher bandwidth and robustness. The latest large-scale data centers use redundant paths to achieve the same goal. Popular servers are multi-homed to more than one Internet provider to handle partial failure. Although the Internet was designed to be multipath to satisfy these requirements, TCP/IP, the dominant Internet Protocol Suite which is always single-path, seems ill-suited for the future Internet.

The idea of concurrently making use of multiple paths for communication is not new. This topic has been
considered widely in the literature in various contexts. The earliest reference to concurrent multipath transmit, referred to as dispersity routing, is in Maxemchuck’s dissertation [1] in 1975. After that a great amount of research [2–18] on multipath transmission has been published.

Although delivering data over multiple paths has been discussed since the beginning of the Internet, the reordering issue associated with multipath transmission is still a challenge today. Most existing multipath solutions and frameworks try to boost the aggregate bandwidth and offer higher robustness with an assumption that the receive buffers are unbounded. Few of them really consider the fact that large receive buffers may prevent multipath transmission from being used on busy servers and memory-scarce devices. In regular TCP, the reordering problem comes up, because networks may reorder the packets or lose them, whereas, in multipath transmit, the problem becomes serious in the context of path heterogeneity. In Multipath TCP (MPTCP) [14], for example, large receive buffers (e.g., a few MB) could always accommodate all of the out-of-order data. But if the receive buffers are bounded, the out-of-order data may overflow them so that the flow control mechanism is triggered, i.e., the next ACK contains zero in the advertised window to stop transmission for a non-transient time period, significantly degrading the aggregate bandwidth.

Other issues of multipath transmission are buffer delay and jitter. Buffer delay is the time period during which an out-of-order packet is stored in the receive buffer until the missing packet arrives. Jitter is the variation of the end-to-end delay. Both the buffer delay and jitter can impact the quality of the real-time applications. In single-path transmission, buffer delay and jitter are caused by network reordering or packet losses, whereas, in multiple path transmission, path heterogeneity also introduces non-trivial buffer delay and jitter. In our study, we observe that large receive buffers could not mitigate the buffer delay and jitter.

In order to improve the reordering problem as well as the delay characteristics, we propose a linear systematic coding [19] enabled multipath transmission protocol, SC-MPTCP. To counter expensive retransmissions, SC-MPTCP uses the systematic coding at the transport layer and adaptively provisions proactive redundancy according to current path characteristics. For example, we use systematic coding to keep the original packets uncoded and only encode the redundancy. For each generation, SC-MPTCP sends the proactive redundancy first and then transmits the uncoded packets. Moreover, in order to counter bursty losses where proactive redundancy may be underestimated, we propose a pre-blocking warning mechanism to allow the receiver to retrieve missing packets without waiting for retransmissions. Moreover, the use of systematic coding could also significantly improve the delay characteristics and CPU consumption.

The performance of SC-MPTCP is promising. It mitigates the buffer delay and jitter significantly. With bounded receive buffers, SC-MPTCP also achieves almost the same aggregate bandwidth of MPTCP with large receive buffers. Our contribution includes: (i) using linear systematic coding to mitigate the buffer delay and jitter with minor encoding/decoding complexity, (ii) using a pre-blocking warning mechanism to retrieve reactive redundancy to compensate for the missing packets without incurring the flow control, and (iii) aggregating available bandwidth of all subflows even if their available bandwidth is small.

The rest of the paper is organized as follows. In Section 2, we review the existing solutions. In Section 3, we introduce a few key components of SC-MPTCP and systematic coding. In Section 4, we mathematically model the key components of SC-MPTCP and also present the design of key algorithms. In Section 5, we evaluate the performance of SC-MPTCP with and without the coupled congestion control option. At last, we conclude in Section 6.

2. Related work

The topic of concurrently using multiple paths to transmit data between two entities has been discussed widely in the literature in various contexts. A challenge associated with multipath transmission is that of connection-level packet reordering. For example, when the traffic between a typical pair of end nodes follows different paths, each path may have diverse bandwidth, loss ratio and delay characteristics. Therefore, packets delivered over different paths may arrive at the receiver out of order. According to the TCP rules, the out-of-order packets have to wait in the receive buffer before they can be delivered in order to a certain application. Once the out-of-order packets overflows the receive buffer, the buffer would be blocked so that the overall performance will degrade significantly.

Lee et al. [2] try to address the reordering issue in multipath transmission by tuning a few TCP parameters (e.g., increasing the fast retransmission threshold and enabling delayed ACKs) and making use of flow-aware routers. The authors in [3,4] propose to schedule packets on multiple paths to offer aggregate bandwidth while minimizing delay. However, they all ignore packet losses on unstable paths.

In order to improve the fault tolerance over lossy paths, Chen et al. [5] propose to transmit multiple copies of the same packet on different paths. But the performance degrades sharply as the loss rate increases beyond 20%. To effectively counter packet losses, Sharma et al. [6] consider using FEC (Forward Error correction) to compensate for lost packets. They use a fixed-rate FEC coding to improve the goodput and reduce packet recovery latency in wireless mesh networks. However, they do not consider receive buffer constraint (i.e., they assume the receive buffer is large enough to handle the out-of-order data). In addition, FEC will introduce extra buffer delay and jitter anyway because of the block decoding process. Cui et al. [7] propose to use Fountain Code to mitigate the impact of path heterogeneity. However, the use of coding also leads to unnecessary delay and jitter, and an analysis on the receive buffer size is missing. Mirani et al. [9] propose an RTT-aware scheduling algorithm to reduce the receive buffer. Unfortunately, their solution is brittle because any unexpected packet loss or RTT variation could disrupt the reordering. Zhuoqun et al. [10] propose to use network coding in wireless mesh networks to minimize reordering and timeout at the receivers. However, they only give some general description on how network coding can be integrated into multipath TCP but lack detailed analyses.
on how network coding could be used to solve the reordering problem with bounded receive buffers.

Hsieh and Sivakumar [11] propose to allow the application to provide large aggregate buffers to handle the out-of-order data. However, they do not solve the problem itself but just move the problem from the transport layer to the application layer.

Stream Control Transmission Protocol (SCTP) [12] has been designed with multihoming support. A few SCTP extensions [13,15] support multipath transmission. To solve the reordering problem, the authors in [13,20] propose several retransmission policies. However, these policies can only solve the reordering problem caused by spurious retransmissions but cannot handle reordering in lossy networks. Furthermore, the policies did not take bandwidth and packet loss factors into consideration.

Multipath TCP (MPTCP) [14,21], being developed within the Internet Engineering Task Force (IETF), is a major extension to TCP. It supports multihoming and is backward compatible with the regular single-path TCP. Barre et al. [16] evaluate the impact of different delays and packet losses of multiple paths on receive buffer and aggregate throughput. The results show that losses on one subflow have limited impact on the performance of the other subflow. However, they assume a large receive buffer (e.g., a few MB) that can handle the out-of-order data when packet losses happen. The authors in [8,17] evaluate the performance of MPTCP in terms of load sharing and throughput optimization with and without the coupled congestion control option respectively. The results show that the context of heterogeneous networks (e.g., Ethernet, Wifi, and 3G) has a great impact on the performance of MPTCP with bounded receive buffers. Raiciu et al. [18] propose a series of mechanisms at the sender side to avoid the reordering problem using bounded receive buffers. Although these mechanisms could help avoid a serious reordering issue, they use the mechanisms of opportunistic retransmission and penalizing slow subflows, which prevents the MPTCP from making use of all available bandwidth, violating the original design goal of MPTCP. In our previous work [22], we use a coding-based approach to counter against path heterogeneity. Like the other coding approaches, it fails to solve the buffer delay and jitter issues.

To summarize, some key limitations of the existing works on this topic are: (i) they may not scale well to heterogeneous, lossy networks, (ii) most of them assume large receive buffers, (iii) they suffer from performance degradation with bounded receive buffers in heterogeneous networks, (iv) non-coding solutions cannot mitigate the buffer delay and jitter, and (v) coding solutions will introduce extra buffer delay and jitter because of the decoding process.

To address these limitations, we propose a systematic coding enabled MPTCP, called SC-MPTCP.

3. Our approach

There has been a good deal of work on building multipath transport protocols [12,13,21,23,24]. In our work, we use MPTCP as the base multipath transmission protocol. Because (i) it is being developed in IETF [14,21], (ii) it starts to be used in large-scale data centers and big server farms [25,26], (iii) it provides a resource pool [27] feature which is very useful for traffic engineering in networks.

In this section, we present the key components in the SC-MPTCP architecture and introduce systematic coding to be used in SC-MPTCP with the added advantage of requiring fewer and simpler operations during the coding process. In the next section, we present the mathematical model and design of SC-MPTCP in detail.

3.1. Architecture

Since SC-MPTCP uses MPTCP as its base protocol, they have the same basic components in the architecture. In this section, we mainly introduce the extra components used in SC-MPTCP. But before that, in order to make the paper self-contained, we first give a brief introduction to the coupled congestion control (CCC) option of MPTCP [28]. We evaluate the performance of MPTCP with and without the CCC option in Section 5.

The CCC algorithm has been discussed in [28]. It applies to the additive increase phase of the congestion avoidance state specifying how the window inflates upon receiving an ACK. For each ACK received on the ith subflow, its congestion window (cwd) is increased by

$$\min \left( \frac{\alpha \cdot \text{bytesacked} \cdot \text{mssi}}{\sum \text{cwd}_i}, \frac{\text{bytesacked} \cdot \text{mssi}}{\sum \text{cwd}_i} \right).$$

where cwi and mssi are the congestion window and maximum segment size respectively on the ith subflow, $\sum cwi$ is the sum of the congestion windows of all available subflows. Parameter $\alpha$, which controls the aggressiveness of the multipath traffic flow, is calculated by

$$\alpha = \sum \text{cwd}_i \frac{\max \text{cwd}}{\text{rtti}^2},$$

where rtti is the round trip time of the ith subflow. MPTCP with the CCC option could move traffic away from the most congested paths and provide bottleneck fairness to legacy traffic, whereas MPTCP without the CCC option cannot offer such benefits and behaves like multiple standalone TCP connections sharing the same aggregate send and receive buffers. For more information about the CCC option as well as the MPTCP architecture, please refer to [14,28].

We now introduce the extra components used in SC-MPTCP. In order to integrate the systematic coding into the transport layer of MPTCP, the Encoder and Decoder components are added in the sender and receiver respectively. The former generates redundant packets by linearly combining a generation (block) of packets according to a generator matrix (is introduced in the next section). Instead of merely delivering plain packets as MPTCP does, each subflow may transmit the coded packets. The Decoder is used to recover the packets when it receives enough linearly independent coded packets.

One of the main purposes of making use of systematic coding is to compensate for expensive retransmissions by provisioning redundancy. However, the redundancy comes at the cost of traffic overhead. In order to balance the traffic
overhead and the aggregate bandwidth gained by using multipath transmit, we add two other components: Proactive Redundancy Estimation and Pre-blocking warning mechanism. The former provides proactive redundancy adaptively to the estimated current path characteristics, and the latter allows the receiver to retrieve reactive redundancy to compensate for the missing packets without waiting for expensive retransmissions. The pre-blocking warning mechanism is usually triggered if the proactive redundancy is underestimated in burst networks. We discuss these two components in Section 4.2 and 4.3 respectively.

To the best of our knowledge, all existing coding solutions of multipath transmission incur non-trivial decoding delay and heavy computational overhead, because the receiver has to wait for a considerable number of packets and then performs computationally heavy Gaussian elimination algorithms to recover the data. These two issues are not specific to multipath transmission but are two of the key concerns when applying coding approaches to networking problems, whereas our work differs from them in that we make use of a systematic coding approach to improve the delay characteristics and CPU consumption. We introduce the basic idea of systematic coding in SC-MPTCP in the next section. Mathematical models of systematic coding on decoding delay and computation complexity are given in Section 4.4.

3.2. Systematic coding

In this section, we briefly introduce how systematic coding could be used in SC-MPTCP and discuss its benefits by illustrating an example of a simplified transmission procedure using different protocols.

In coding theory, a systematic coding is any error-correcting code in which the input data is embedded in the encoded output. In contrast, in a non-systematic coding the output does not contain the input symbols. All existing coding solutions for multipath transmission, to the best of our knowledge, use non-systematic coding. In SC-MPTCP, we divide the data stream into generations (i.e., blocks). In each generation (we use \( \theta \) to denote its size), every original packet is encoded to itself and only the redundancy is the linear combinations of the original packets. Specifically, we interpret each packet as a symbol over the field \( GF(2^s) \), where \( s \) indicates the packet size (we assume each packet has the same size). The coded packets are generated by combining the packets from the same generation using random coefficients over the field \( GF(2^s) \). To simplify this procedure without loss of generality, we use a predefined linear independent coefficient matrix \( A_k \) to generate the coded packets and use an identity matrix \( I_s \) to encode the original packets to themselves. Thus, the generator matrix \( M \) for a generation is

\[
M = [A_k | I_s], \tag{3}
\]

where \( k \) (the number of redundant packets) is dynamically determined by the current path characteristics and is further discussed in Section 4.2.

We now analyse an example of a few simplified transmission procedures of MPTCP and SC-MPTCP to highlight the compensation property of coding schemes as well as the decoding delay improvement the systematic coding offers. Moreover, we analyse how the buffer occupation could be affected by the sending order of the redundancy and the uncoded packets.

We assume the sending window size is 5 and consider that a sender transmits five packets and one of them is lost. In the MPTCP transmission procedure, shown in Fig. 1a, \( p_4 \) is lost and will be retransmitted later. Before the arrival of the retransmitted \( p_4 \), the receiver needs to accommodate all the out-of-order packets. If \( p_4 \) cannot arrive before the out-of-order packets overflow the receive buffer, the transmission will be blocked due to flow control. In this paper, we name this phenomenon head-of-line blocking (HLB) and name the packet (e.g., \( p_4 \)) introducing the out-of-order data head-of-line packet.

Under the same network context, Fig. 1b shows the transmission procedure of MPTCP using non-systematic coding, where the generation size is 4 (i.e., \( \theta = 4 \)) and one more packet (i.e., \( k = 1 \)) is needed to compensate for the lost packet. The sender sends out five packets and each of them is a linear independent combination of the original packets. During the transmission, the \( 4_{th} \) packet is lost. However, the receiver can still make use of the redundancy to decode the original packets and forward them up to the upper layer without waiting for the retransmission. Moreover, if the sender receives the acknowledgment of \( \text{Ack} = 6 \) before retransmitting the lost packet, it will just ignore the retransmission, which saves the traffic without breaking the sequence number integrity.

By comparing Fig. 1a and b, we notice that in MPTCP transmission, the in-order arriving packets (e.g., \( p_1, p_2 \) and \( p_3 \)) can be forwarded up immediately, whereas, in the transmission of MPTCP using non-systematic coding, all the packets from the same generation are buffered until enough coded packets arrive, not only introducing unnecessary buffer delay and jitter but also increasing buffer burden. Fig. 1c presents the transmission procedure of SC-MPTCP, where systematic coding is used. As shown in the figure, the redundancy is sent at the beginning of the generation and the uncoded packets are sent later. We observe that systematic coding preserves the compensation property of non-systematic coding while keeping the advantage of MPTCP that in-order arriving original packets (\( p_1 \) and \( p_2 \)) can be forwarded up immediately without incurring any delay.

In Fig. 1c, we send the redundancy initially, and then transmit the original packets in subsequent transmissions. Fig. 1d shows the transmission procedure by reversing the sending order, i.e., sending the original packets first and redundancy later, so that in the case of loss-free reception, the receiver can do the data delivery without having to do any decoding at all. Note that our target is to tolerate path heterogeneity. Therefore, we need to release the receive buffer as soon as possible. In Fig. 1d, although the original packets can be forwarded up to the upper layer, they cannot be released until enough linearly independent packets arrive. Because the loss-free reception of a generation is unpredictable, the receiver must buffer all the original packets just in case some packets are lost.

Conversely, in Fig. 1c, we could release the in-order arriving original packets immediately and only buffer the redundancy. For example, when \( p_1 \) arrives and is forwarded up to the upper layer, the coded packet \( p_1 + p_2 + p_3 + p_4 \) could be recoded to \( p_2 + p_3 + p_4 \) by...
removing $p_1$ arithmetically, so that the buffer space of $p_1$ could be released.

In our SC-MPTCP design, we choose to send the redundancy at the beginning and transmit the uncoded packets later.

4. Mathematical model and design

In this section, we model the key components of SC-MPTCP mentioned in Section 3 and give a theoretical justification for the key parameters used in SC-MPTCP.

4.1. Receive buffer models in multipath transmission

In regular, single-path TCP, it is usually recommended to use

$$\text{Buf} = 2 \cdot BW \cdot RTT$$

(4)

to tune the receive buffer to ensure that twice the Bandwidth-Delay Product (BDP) of the path can be stored. In the context of multipath transmission, the story is a bit more complicated. To ensure in-order delivery, MPTCP must use a connection level aggregate receive buffer, where packets are stored until they are in order before they are extracted by the application. The goal is that a subflow packet loss or delay should not affect the throughput of the other working subflows; the receiver should have enough buffer to store out-of-order data until the missing packet successfully reaches the receiver. To achieve this goal, Ford et al. [14] recommended the aggregate receive buffer in MPTCP should be dimensioned as

$$\text{Buf} = 2 \cdot \sum_{i=1}^{N} BW_i \cdot RTT_{\text{max}}$$

(5)

where $N$ is the number of subflows, $BW_i$ is the available bandwidth of the $i$th subflow, and $RTT_{\text{max}}$ is the largest RTT across all subflows. Through our simulation study, we observe that the HLB problem does not disappear even if the buffers are set according to (5). A real implementation of MPTCP in [18] also reaches the same conclusion. The reason is that (5) is a conservative buffer-tuning algorithm. It only takes into account reordering by the network and fast retransmit but ignores timeout retransmit, where the sender retransmits the unacked packets after a timeout. As discussed in [14], in order to avoid stalling in the presence of timeouts, the minimum required buffer should be

$$\text{Buf} = 2 \cdot \sum_{i=1}^{N} BW_i \cdot RTO_{\text{max}}$$

(6)

where $RTO_{\text{max}}$ is the largest RTO (retransmission timeout) across all subflows. In TCP implementations, the timeout threshold is set according to a function of the estimated RTT and should be greater than the Minimum RTO ($RTO_{\text{min}}$), $RTO_{\text{min}}$ is used to protect against spurious expirations when the Delayed ACK mechanism [29] is enabled. The Delayed ACK mechanism allows the receiver to generate one ACK for every other packet for as long as the Delayed ACK timer suggests (usually no less than 200 ms). Thus, $RTO_{\text{min}}$ is set to no less than 200 ms. For example, $RTO_{\text{min}}$ is set to 200 ms and 1 s in Linux and Windows OS respectively as default.

As pointed out by [14], the receive buffer set according to (6) requires an order of magnitude more than the receive buffer required for a single-TCP connection, and many times larger than the required buffer set according to (5). It is too expensive for practical purposes. For example, if $RTT_{\text{max}}$ is less than half of $RTO_{\text{max}}$ (e.g., $RTT_{\text{max}} = 20$ ms and $RTO_{\text{max}} = 200$ ms), which covers most of the normal scenarios in practice, the receive buffer set according to (5) would certainly be underestimated. Moreover, it be-
comes even worse if the default \( RTO_{\text{min}} \) is set to 1 s which is recommended by IETF in [29].

In SC-MPTCP, we use (5) to tune the aggregate receive buffer to provision optimal aggregate bandwidth.

### 4.2. Proactive redundancy estimation

In this section, we discuss the proactive redundancy estimation algorithm. We first define the retransmission ratio as follows.

**Definition 1 (Retransmission ratio).** The retransmission ratio is the ratio between the number of retransmitted packets and the number of sent-out packets during a certain time period.

Let \( G \) denote a generation of packets. We still use \( \theta \) to represent the size of \( G \). We use \( r_i \) to denote the retransmission ratio on the subflow \( f_i \) and use \( n_i \) to denote the number of packets delivered over \( f_i \) for generation \( G \). We assume there are \( N \) subflows. In order to guarantee that the redundant packets can compensate for retransmissions, the following equations must be satisfied

\[
\begin{align*}
\sum_{i=1}^{N} n_i \cdot (1 - r_i) & = \theta \\
\sum_{i=1}^{N} n_i & = \theta + k, 
\end{align*}
\]

where \( h \) denotes the number of redundant packets for \( G \) as before. According to (7), under the same network conditions, choosing a large enough value for \( k \) can always avoid recovery latency, but an overestimated \( k \) could incur unnecessary traffic overhead undermining the throughput advantage gained by using multipath transmission. Thus, \( k \) must be carefully selected to maintain a reasonable trade-off between performance and overhead.

In order to calculate \( k \), (7) requires an exhaustive search that does not scale well with the number of paths. In our algorithm, instead of estimating the parameters of each individual subflow, we treat all the subflows as a whole and use aggregate retransmission ratio to help in averaging out the volatility of each individual subflow, which can lead to a smoother and more stable estimation. Let \( r_a \) denote the aggregate retransmission rate across \( N \) subflows. \( r_a \) can be estimated as follows

\[
r_a = \frac{\sum_{i=1}^{N} n_i r_i}{\theta + k}.
\]

Taking (7) into (8), we get

\[
k \geq \frac{\theta \cdot r_a}{1 - r_a}.
\]

As long as (9) is satisfied, the receiver could decode the generation without waiting for retransmission. However, in practical networks, the network quality varies over time. It is impossible to precisely predict the redundancy rate in the next time period. If more than the expected number of packets are retransmitted, the receiver can only wait for the retransmissions which would become a potential risk of blocking the receive buffer. In our algorithm, in order to cover the normal variance of retransmission ratio, we determine \( k \) according to the mean aggregate retransmission rate \( r_a \) as well as its variance \( \sigma_r \) as follows

\[
k \geq \theta \cdot \frac{r_a + \sigma(r_a)}{1 - r_a - \sigma(r_a)}.
\]

We predict \( r_a \) by sampling the history value of \( r_a \) periodically and averaging those samples (using the EWMA method). With each new sample \( r_a(t), r_a(t + 1) \) is computed as

\[
r_a(t + 1) = (1 - \alpha) \cdot r_a(t) + \alpha \cdot r_a(t),
\]

where \( \alpha \) is a variable between 0 and 1 controlling how rapidly \( r_a \) adapts to new change. We set \( \alpha \) to 0.8 to allow \( r_a \) to adapt more swiftly to dynamic network conditions.

In IP networks, packet losses are bursty and a retransmission may happen suddenly without being able to predict it in advance. In order to avoid expensive retransmission, especially timeout retransmission, a reactive strategy alone would not be effective enough. Therefore, we make use of a default proactive redundancy to handle the retransmit. Specifically, we set the default proactive redundancy to \( K_{\text{min}} \) for each generation, i.e., the lower bound of \( k \) is \( K_{\text{min}} \). In our simulations we set \( K_{\text{min}} = 1 \), because we assume only one subflow may enter timeout retransmission while delivering the same generation. In practice, this assumption is feasible because timeout retransmission would not happen frequently, especially during the delivering of the same generation.

Because the proactive redundancy is provisioned adaptively to the path characteristics, it may be overestimated in some cases. We believe that the overestimated redundancy would not be wasted but could compensate for fast retransmissions as well as delayed packets to boost the aggregate bandwidth.

### 4.3. Pre-blocking warning mechanism

In the previous section, we discussed the redundancy used to compensate for retransmissions proactively. In this section, we assume that the proactive redundancy is underestimated. In such a case, the reactive redundancy for a certain generation is required to compensate for missing packets. In order to avoid spurious reactive redundancy, where packets may simply be delayed slightly with no need of more redundancy, a trigger condition must be set carefully to save traffic overhead. In this section, we introduce a pre-blocking warning mechanism to obtain reactive redundancy without waiting for expensive retransmissions and explore an appropriate condition to trigger it.

We use \( D_{\text{arr}} \) to denote the arrival delay of a packet and use \( BW_{agg} \) to denote the aggregate throughput during \( D_{\text{arr}} \). In SC-MPTCP, the receiver needs to buffer the out-of-order data until the head-of-line packet reaches the destination. Therefore, the required volume of buffer to accommodate the out-of-order data is \( BW_{agg} D_{\text{arr}} \). According to (5) (see Section 4.1), if

\[
BW_{agg} D_{\text{arr}} > 2 \cdot BW_{agg} \cdot RTT_{max},
\]

where \( BW_{agg} \) is usually calculated according to the observed aggregate throughput in history, then the...
out-of-order data will overflow the receive buffer. In order to avoid it, we have two basic methods:

1. Penalizing certain subflows during $D_{agg}$ to reduce $BW_{agg}$.
2. Shortening $D_{agg}$ so that the missing packet could arrive before the buffer overflow.

The former needs to degrade the aggregate throughput deliberately. In SC-MPTCP, we adopt the latter because it requires no throughput degradation.

We now try to extract the idea of the pre-blocking warning mechanism by considering a buffer reordering process in the case of subflow packet loss. To simplify the process without loss of generality, we assume there are 2 subflows, one of which is a lossy path and the other has no packet loss. As shown in Fig. 2, when a packet is lost on the lossy path, the following arriving packets on the same path would become out-of-order data and be buffered in the subflow receive buffer, while on the lossless path, all the packets would arrive in order on the subflow level. When the packets delivered on the lossless path move from the subflow receive buffer to the aggregate buffer, the packets become out-of-order packets, because the aggregate buffer is waiting for the head-of-line packet, the same packet the lossy subflow is waiting for. If a subflow in-order arriving packet fails to be inserted into the aggregate buffer, it implies that the current subflow will be blocked with high probability. We use this failed insert operation as the trigger of the pre-blocking warning mechanism. When the mechanism is triggered, an ACK carrying the required number of packets for a certain generation is sent by the receiver to the sender. The sender then sends the required packets (also taking into account packet loss) immediately in response on the same subflow.

We now investigate whether this pre-blocking warning mechanism is effective in avoiding flow control on the lossless path. We still use Fig. 2 as an example. Let $BW_{loss}$ and $RTT_{loss}$ denote the available bandwidth and RTT of the lossless path respectively. When the pre-blocking warning mechanism is triggered on the lossless path, the missing packet could arrive at the destination in $RTT_{loss}$ during which $BW_{loss} \cdot RTT_{loss}$ size of buffer is required to accommodate the packets arriving in order on the lossless path. Note that according to (4), the subflow receive buffer is tuned to $BW_{loss} \cdot RTT_{loss}$ which is big enough to avoid buffer overflow. Thus, the pre-blocking warning mechanism is flow control safe.

In the discussion above, we have assumed that the missing packet arrives at the destination in one round of RTT. However, in practice, the retransmitted missing packet may be lost again even if we send redundant packets on multiple paths for safe consideration. We argue that this extreme case rarely happens and there exists no effective method to solve it. Thus, the pre-blocking warning mechanism is flow control safe in normal cases where the missing packet could arrive at the destination in an expected time period.

4.4. Systematic coding model

As discussed in Section 3.2, the desirable properties of non-systematic coding come at the expense of a potentially high decoding delay and heavy computational overhead. In this section, we use a simplified mathematical model to show that systematic coding could significantly reduce both the decoding delay and the computational overhead.

In the existing coding solutions of multipath transmission, the sender encodes a generation of packets and the receiver decodes the generation if it receives enough number of linearly independent coded packets. Before the receiver is able to decode the generation, all coded packets are stored at the buffer, thereby extra buffer delay is introduced, which further causes jitter. We use $p_1, p_2, \ldots, p_n$ to denote the received coded packets from a generation $G$. Note that in any non-systematic coding, there is no concept of packet sequence within a generation. The sequence of the subscript in $p_i, p_2, \ldots, p_n$ only indicates the arrival sequence. We assume the packets arrive at the receiver at a fixed time interval. In practice, this assumption is over-simplified, because the packets may arrive randomly without obeying any known distribution. We make this assumption here only to get a feeling for the benefits of using systematic coding. Please see Section 5.5 for simulations which demonstrate the buffer delay improvement due to the use of systematic coding.

Let $D_d(i)$ denote the decoding delay of $p_i$, and $BW_{agg}$ denote the aggregate available bandwidth. Then we have

$$D_d(i) = \frac{(\theta - i) \cdot MSS}{BW_{agg}},$$

where we assume each packet uses the same segment size denoted by MSS. Based on (13), we get the expected packet decoding delay in the generation as follows

$$E[c(D_d)] = \frac{(\theta - 1) \cdot MSS}{2 \cdot BW_{agg}}.$$  

Eq. (14) shows that we could get a small decoding delay by either shortening the generation size or using subflows offering large aggregate available bandwidth.

We now model the decoding delay of systematic coding. In systematic coding, the in-order packets of a generation have the concept of sequence and can be forwarded up to an application immediately, whereas the out-of-order packets might not arrive in sequence.
denote the out-of-order packets as a non-systematic generation. In order to differentiate the expected decoding delay between non-systematic codeling and systematic codeling, we use $G^*$ to denote the generation using systematic codeling. The expected packet decoding delay for $G^*$ could be expressed as

$$E_{Dd}^*(D_g) = \sum_{i=0}^{\theta-2} P(|G_1| = i)E_{G_i}(D_g) = \left(\frac{\theta-1}{\theta} \cdot \frac{\theta-2}{\theta} \cdot \frac{MSS}{4 \cdot BW_{agg} \cdot \theta}\right).$$

We take (14) into the equation above and then get

$$E_{Dd}^*(D_g) = \left(\frac{1}{2} - \frac{1}{\theta}\right) \cdot E_{Dd}(D_g).$$

Eq. (16) shows that in the worst case, systematic codeling reduces the decoding delay of non-systematic codeling by at least 50% in theory. In our simulations conducted in Section 5.5, we show that the systematic codeling enabled SC-MPTCP could improve the buffer delay by at least 80% compared to MPTCP and at least 90% compared to MPTCP using non-systematic codeling.

In addition to the improvement of decoding delay, the systematic codeling also reduces the number of required encoding and decoding operations considerably. In non-systematic codeling, the arithmetic complexity of generating $\theta + k$ codelized packets is $O((\theta + k)^2)$ and the arithmetic complexity using Gaussian elimination to decode a generation of size $\theta$ is $O(\theta^3)$, whereas, in systematic codeling, the computation complexity of generating $k$ codelized packets and decoding $k$ codelized packets are reduced to $O(\theta k)$ and $O(\theta k^2)$ respectively. Thus, systematic codeling could significantly reduce the computational overhead involved in the encoding and decoding operations. For example, in this paper, we set $\theta$ to 50, which is explained later in the next section, and $k$ is usually 1 or 2 in most cases. The arithmetic complexity of generating codelized packets is reduced by 1 order of magnitude. And the arithmetic complexity of decoding a generation is reduced by 2 or even 3 orders of magnitude.

In summary, the use of the systematic codeling in SC-MPTCP greatly reduces the computation overhead and incurs negligible processing delay of codelizing operations.

5. Performance evaluation

In this section, we evaluate the performance of SC-MPTCP with and without the coupled congestion control option. To differentiate the protocols used in this section, we name the MPTCP and SC-MPTCP without the coupled congestion control option MTCP and SC-MTCP respectively, and name the MPTCP and MTCP using non-systematic codeling NSC-MPTCP and NSC-MTCP respectively.

The simulation platform used in this paper is Network Simulator NS-3.11. We use a N-path topology, shown in Fig. 3, to abstract the physical routes where different paths correspond to different routes. It is a typical topology used in most of the MPTCP literature. For every set of parameters we consider, we repeat the measurements 10 times with different global random seeds, and without explicit specification we present the mean value in the simulation results. In each simulation, we transmit a file of 100 MB and set $RTO_{min}$ to 200 ms. The RTT and bandwidth on each subflow, without explicit specification, are set to 20 ms and 100 Mbps as default respectively. In our simulation, we have used various generation size to test the performance of SC-MPTCP and SC-MTCP. We found that they performed the best when the generation size was 50. Thus, we choose $\theta = 50$ for all the simulations conducted in this section. We simulate the path heterogeneity by keeping the parameters of one subflow fixed while varying the parameters of the other subflows.

5.1. Performance metric

In our study, we have been interested in assessing the performance in terms of the following metrics: minimum required receive buffer, aggregate bandwidth, buffer delay and jitter.

In this paper, we ignore the computational delay which may be introduced by codeling operations, because first we focus only on the form of systematic codeling to reduce codeling cost instead of measuring it and secondly, as we have demonstrated in Section 4.4, systematic codeling introduces few codeling operations per generation, which does not impact the overall performance. In addition, in the simulations we observe that the performance comparison between SC-MPTCP and MPTCP may be similar to that between SC-MTCP and MTCP. In this case, if it is not necessary to compare the performance of SC-MPTCP and SC-MTCP, we will only present one of the comparisons.

In this section, we define the throughput of MPTCP using an infinite receive buffer as the optimal throughput, because under the same network context, MPTCP achieves...
the maximum throughput if the receive buffer is big enough, e.g., infinite.

5.2. Receive buffer size analysis

As discussed in Section 4.1, one of the main issues affecting MPTCP performance is that MPTCP may require large receive buffers in heterogeneous networks. In this section, we analyse the minimum receive buffer required to achieve the optimal performance in multipath transmission.

We use a 2-path topology and fix the packet loss ratio and RTT of one subflow to 0.1% and 20 ms respectively. We increase the path heterogeneity by setting the packet loss ratio and RTT of the other subflow from 1% to 5% and from 40 ms to 60 ms respectively. Note that in this simulation, we set the receive buffer of MPTCP infinite to obtain the maximum occupied receive buffer as the minimum required receive buffer, whereas, it is not fair to do the simulation of SC-MPTCP using the infinite receive buffer, because the pre-blocking warning mechanism will not take effect if the receive buffer is infinite. Therefore, in the simulation of SC-MPTCP with pre-blocking warning mechanism, we set the receive buffer according to (5).

Fig. 4 shows the minimum required buffer to achieve the maximum aggregate throughput. We use (5) to calculate the theory value of the required receive buffer as benchmark. We observe that the required buffer increases when either the packet loss ratio or the RTT of the other subflow grows. We deduce that the larger the path heterogeneity is, the more receive buffer is needed. Moreover, under the same network conditions, SC-MPTCP without the pre-blocking warning mechanism requires much less receive buffer than MPTCP. However, without the pre-blocking warning mechanism, SC-MPTCP needs more receive buffer than the benchmark. In Fig. 4, we do not draw the curves of SC-MPTCP with the pre-blocking warning mechanism enabled, because, when we set the receive buffer according to the theory value (5), it could always approach the same performance as MPTCP with large receive buffers. This result implies that (5) is a good algorithm to tune the receive buffer of SC-MPTCP with the help of the pre-blocking warning mechanism. The effectiveness of the pre-blocking warning mechanism is further demonstrated in the next section.

In the rest of the simulations, we set the receive buffer according to (5).

5.3. Effectiveness of pre-blocking mechanism and proactive redundancy

In this section, we evaluate the effectiveness of the pre-blocking warning mechanism and the proactive redundancy by analysing the runtime aggregate throughput in SC-MPTCP and SC-MTCP. Through the simulations, we want to demonstrate that either the pre-blocking warning mechanism or the proactive redundancy could mitigate the reordering issue, but these two strategies are not orthogonal, i.e., they must work together to avoid the HLB problem completely.

In the simulation results presented next, we use a 2-path topology. The loss ratio on each subflow is set to 0.1% and 2% respectively. Fig. 5 presents the variance of the runtime aggregate throughput of MPTCP, where we enable and disable the pre-blocking warning mechanism respectively. We observe that the pre-blocking warning mechanism could improve the aggregate throughput of MPTCP, but it cannot avoid the HLB completely. Fig. 6 shows the aggregate throughput of SC-MPTCP, where we disable the proactive redundancy (PR), and enable and disable the pre-blocking warning mechanism respectively. We observe that SC-MPTCP which has disabled both of the pre-blocking warning mechanism and the proactive redundancy has a very similar performance to MPTCP shown in Fig. 5. When we enable the pre-blocking warning mechanism but still disable the proactive redundancy, the performance of SC-MPTCP improves a lot in most of the time. However, it still cannot avoid the HLB completely. Therefore, we deduce that with the pre-blocking warning mechanism alone, MPTCP and SC-MPTCP could mitigate the HLB problem but could not kill it.

We now enable the proactive redundancy. Fig. 7 shows the simulation results, where the pre-blocking warning mechanism is enabled and disabled respectively. As shown in the figure, at the beginning of the transmission, two curves completely overlap. Then the curve of SC-MPTCP without pre-blocking warning mechanism goes down significantly due to HLB and goes up in a few seconds. This up and down process repeats until the end of the transmission. Every time the curve recovers, we observe that SC-MPTCP with and without the pre-blocking warning mechanism keeps a similar performance. This phenomenon implies that the pre-blocking warning mechanism with the help of the proactive redundancy is able to avoid HLB. According to the discussion above, we believe that the pre-blocking warning mechanism must work together...
with the proactive redundancy in order to counter against HLB. In the following simulations, we enable the proactive redundancy as default as most existing coding-based MPTCP solutions did. We now further analyse the packet arrival time of SC-MPTCP with and without the pre-blocking warning mechanism to explore the reason for the performance degradation of SC-MPTCP without pre-blocking warning mechanism. Fig. 8 draws the curves of packet arrival time of SC-MPTCP during the first time when the curve of SC-MPTCP without pre-blocking enabled in Fig. 7 starts to go down. As shown in Fig. 8, at the beginning of the transmission, two curves also completely overlap in consistence with the curves in Fig. 7, and then one packet is lost. In SC-MPTCP without pre-blocking warning enabled, during the retransmission time period (i.e., $D_1$), the out-of-order data overruns the receive buffer, whereas, with pre-blocking warning enabled, SC-MPTCP allows the receiver to retrieve the required packet using less time (i.e., $D_2$), which eliminates the possibility of HLB.

To compare the performance of SC-MPTCP and SC-MTCP, we present the aggregate throughput of SC-MTCP in Fig. 9. By comparing the curves in Figs. 7 and 9, we observe that during the unblocking period, the average aggregate throughput of SC-MTCP outperforms that of SC-MPTCP, because SC-MTCP inherits the coupled congestion control option from MPTCP. Therefore, SC-MTCP provisions conservative throughput boost in order to provide fairness to legacy TCP. Another observation is that during the blocking period, SC-MPTCP performs worse than SC-MTCP. This is because, when the subflow HLB occurs, the aggregate throughput reduces to the throughput of the live subflow which has more packet losses. Affected by the resource pooling feature of MPTCP, the live subflow moves its traffic away to another subflow even though it is blocked. This problem does not come from SC-MPTCP but inherits from MPTCP.

5.4. Bandwidth aggregation

In this section, we study the capability of SC-MPTCP and SC-MTCP to aggregate bandwidth from heterogeneous subflows in wired networks and wireless networks respectively. In the rest of the simulations, we enable both the pre-blocking warning mechanism and the proactive redundancy in SC-MPTCP and SC-MTCP as default.

5.4.1. Wired multi-homing scenario

Multipath bandwidth aggregation typically results in large gains in available bandwidth. However, due to path heterogeneity and bounded receive buffers, it may degrade significantly. To demonstrate that SC-MPTCP and SC-MTCP could avoid the degradation of the overall performance with bounded receive buffers, we use the average goodput of MPTCP with large receive buffers (8 MB) and single-path TCP (on the path having the best quality) as benchmark and evaluate the performance of MTCP, MPTCP, SC-MPTCP and SC-MTCP. In the following simulations, we use a 2-path topology and keep the loss ratio of one subflow fixed to 0.1% and increase the path heterogeneity by increasing the loss ratio of the other subflow from 1% to 5%.

Fig. 10 shows the simulation results. We observe that SC-MPTCP and SC-MTCP with bounded receive buffers could approach almost the same average goodput as MPTCP and MTCP respectively with large receive buffers, even if the path heterogeneity is huge, whereas, under the same path heterogeneity, the performance of MPTCP and MTCP...
is poor, even worse than that of a single-path TCP connection. Furthermore, we observe that the MPTCP degrades more significantly than MTCP, which is consistent with what we have discussed in Section 5.3 on the aggregate runtime throughput of them. Therefore, we deduce that MPTCP degrades more seriously than MTCP on heterogeneous paths with bounded receive buffers.

We now study the impact of the number of subflows on the aggregate bandwidth. In the simulations, we start from using two subflows, keep the packet loss ratio of one subflow at 0.1% and vary the loss ratio of the other subflow from 1% to 5%. Each time we add one more subflow having a medium path quality (e.g., packet loss ratio is set to 0.5% and other parameters keep the same). Fig. 11 presents the simulation results. In this figure we use the average goodput achieved by MPTCP with large receive buffers (e.g., 8 MB) as benchmark. The Y axis indicates the goodput percentage of it. As shown in Fig. 11, MPTCP always achieves less than 20% of the optimal average goodput. Its performance further degrades when the path heterogeneity grows. Furthermore, the curves of MPTCP using a different number of subflows almost overlap with each other. We deduce that more subflows of MPTCP with bounded receive buffers cannot boost the percentage of the optimal goodput, whereas SC-MPTCP always approaches the optimal performance in all circumstances.

We now study the impact of different delivery delays on the aggregate bandwidth. We use the average goodput of MPTCP with large receive buffers (8 MB) as benchmark and evaluate the average goodput percentage of it the MPTCP and SC-MPTCP could achieve respectively. We keep the RTT of one subflow at 20 ms and vary the RTT of another subflow from 20 ms to 80 ms. Each time we add one more subflow which has a medium RTT. All the sub-flows have the same packet loss ratio of 0.1%. Fig. 12 shows the simulation results. We observe that using the same number of subflows, the average goodput of MPTCP with bounded receive buffers decreases when the RTT diversity grows. And more subflows could help achieve a relatively higher percentage of the optimal goodput, whereas SC-MPTCP with bounded receive buffers always performs as well as MPTCP with large receive buffers.

5.4.2. Wireless multi-homing scenario

Until now we restrict our attention on wired networks. We relax this constraint to study the performance of the protocols in wireless multi-homing scenarios.

5.5. Buffer delay improvement

In this section, we study the performance of buffer delay of multipath transmission protocols. First, we approach the reason behind the HLB problem from the arrival delay viewpoint and demonstrate that in multipath transmission, high arrival delay could cause HLB. And then we present the buffer delay as well as jitter performance of NSC-MTCP and SC-MTCP. In this section, we only present the performance of SC-MTCP, because SC-MPTCP has similar performance.

We first analyse a simulation result of MTCP with large receive buffers (e.g., 8 MB). We use a 2-path topology and set the packet loss ratio of each subflow to 0.1% and 1% respectively. Fig. 14 shows the simulation results. Fig. 14a shows the arrival delay of each packet. We observe that high arrival delays always appear suddenly and randomly. Its impact on buffer delay is shown in Fig. 14b. We observe that high arrival delays always appear suddenly and randomly. Its impact on buffer delay is shown in Fig. 14b. We observe that high buffer delays always start to appear...
at the same time when there are high arrival delays and recover gradually, which makes the curve look like a right triangle. The length of the bottom edge of each triangle indicates the size of the buffers required to handle the out-of-order data. The gradient of the bevel edge represents the aggregate throughput during the out-of-order status. Under the same network conditions, the larger the gradient is, the longer the bottom edge will be, which implies larger receive buffers.

Fig. 15 shows the packet buffer delay of MTCP, NSC-MTCP and SC-MTCP all using bounded receive buffers. All the other parameters as well as the global seed are the same as above. Fig. 15a shows the packet buffer delay of MTCP. We observe that at the beginning of the transmission, the curve in Fig. 15a is the same as the curve in Fig. 14b. We deduce that though large receive buffers can accommodate more out-of-order data and achieve the desired throughput, it cannot improve the buffer delays. Moreover, the right triangle in Fig. 14b becomes a right trapezoid in Fig. 15a. This phenomenon implies that one of the subflows is blocked.

Fig. 15b shows the packet buffer delay of NSC-MTCP. We observe that high buffer delays do not appear any more, but medium buffer delays come up regularly and the number of packets having buffer delays even exceeds that of MTCP. According to the shape of the curves, e.g., trapezoidal curves appear regularly, we deduce they are incurred by generation decoding. Fig. 15c shows the packet buffer delay of SC-MTCP. We observe that most of the packets could be forwarded up to the application without introducing any delay in the buffer.

Jitter is a variation in packet transit delay caused by queuing, contention and serialization effects on the path. In general, large jitter is more likely to occur on either slow or heavily congested paths, whereas, when multiple paths are concurrently used, buffer delays also incur jitter which has significant impact on real-time applications. We now study the jitter incurred by the buffer delay. We still use the same parameters as above. The graph in Fig. 16 shows the variance of jitter over time. Fig. 16a and b present the jitter of a single-path TCP connection using the best quality subflow and MTCP respectively. We observe that the TCP connection has moderate jitter with few spikes, whereas MTCP makes it serious. Specifically, the number of spike jitters increases when multiple paths are used. Fig. 16c shows the jitter of NSC-MTCP. We observe that SC-MTCP could eliminate the spike jitter but it introduces much more relatively medium-sized jitter (in the level of 20 ms and 40 ms) incurred by the decoding process. Fig. 16d shows the jitter of SC-MTCP. We observe that it not only eliminates the spike jitter but also improves the jitter incurred by decoding. Furthermore, the jitter performance of SC-MTCP outperforms that of the single-path TCP.

In the following simulations, we further study the impact of path heterogeneity on buffer delay in more test scenarios while demonstrating the effectiveness of systematic coding on improving buffer delays.

Fig. 17 shows the buffer delay under the impact of packet loss and RTT. In this simulation, we use a 2-path topology. We fix the packet loss ratio and RTT of one subflow to 20 ms and 0.1% respectively, and vary the packet loss ratio of the other subflow from 1% to 5% using different RTT. We observe that the average buffer delay of MTCP increases when the path heterogeneity (caused by either packet loss ratio or RTT) grows. Although NSC-MTCP could eliminate spike buffer delays, it introduces non-trivial decoding delays, which is consistent with our previous discussion. In some test scenarios (packet loss ratio is above 3%), the average buffer delay of NSC-MTCP is higher than that of MTCP, whereas SC-MTCP is able to tolerate large path heterogeneity. For example, according to the curves in Fig. 17, SC-MTCP reduces the average buffer delay of MTCP and NSC-MTCP by at least 80% and 90% respectively, which is more effective than the theoretical evaluation discussed in Section 4.4.
Figs. 16. Jitter: (a) single-path TCP, (b) MTCP, (c) NSC-MTCP, and (d) SC-MTCP.

Fig. 17. Average buffer delay.

6. Conclusion

MPTCP is a major extension of TCP aiming to achieve bandwidth aggregation. In a heterogeneous network context, where subflows may have diverse characteristics in terms of bandwidth, delay and packet loss, reordering at the receiver becomes a challenge. In order to avoid the HLB due to the fact that the out-of-order data may overflow bounded receive buffers, we propose SC-MPTCP by incorporating systematic coding into MPTCP.

In this paper, we have simulated the performance of MPTCP and SC-MPTCP with and without the coupled congestion control option under the impact of various path heterogeneity. The results demonstrate that in a heterogeneous network context, MPTCP requires large receive buffers to reach the maximum aggregate throughput. And the receive buffer requirement further increases when the path heterogeneity grows, whereas SC-MPTCP requires much smaller receive buffers to approach the maximum aggregate throughput and it could also tolerate a wide range of path heterogeneity, i.e., the path heterogeneity has limited impact on the minimum required receive buffers. A surprising observation in the simulation results is that although more subflows could help boost the aggregate bandwidth, if the receive buffers are bounded, the boost is limited, especially when the path heterogeneity is caused by different subflow packet losses. The simulation results also demonstrate the effectiveness of a few key components of SC-MPTCP. For example, when the proactive redundancy is underestimated, a pre-blocking warning mechanism could retrieve the missing packets without incurring the HLB. Additionally, the systematic coding itself improves different characteristics: it reduces both the delay and the computational overhead considerably.

Acknowledgment

This work was supported in part by the HIIT Distributed and Mobile Cloud Systems research programme, Tekes within the ITEA2 project 10014 EASI-CLOUDS, and NSFC project (61120106008).

References


Ming Li received his M.S. degree in Computer Science from the Beijing University of Posts and Communications, China in 2008 and is currently a Ph.D. candidate in Computer Science and Engineering at the Aalto University, Finland. His research interests are in the areas of network security, network architecture and network protocols.

Andrey Lukyanenko a postdoctoral researcher at Aalto University. His primary duties are to instruct, guide and help the Master and Ph.D. students with their research and studies. At the department he is a responsible teacher of a methodological course: Methodology for computer science research. He has also produced and improved a small self-educatory course on Practical research tools. His research interests are Information-centric networking, Cloud computing and datacenter networking.

Sasu Tarkoma received his Ph.D. in Computer Science from the University of Helsinki. He has managed and participated in national and international research projects at the University of Helsinki, Helsinki University of Technology, and Helsinki Institute for Information Technology. He has lectured several courses on middleware and data communications. His research interests include middleware and distributed computing.

Yong Cui received his Ph.D. in Tsinghua University, China in 2004. He is currently a professor in Tsinghua University and the Deputy Director of Network Technology Institute. He teaches some courses including Discrete Mathematics, Wireless Network and Mobile Computing. His major research interests include computer network architecture, the Next Generation Internet and mobile computing.

Antti Ylä-Jääski received his Ph.D. in ETH Zurich 1993 and is a professor in the Department of Computer Science and Engineering of Aalto University since 2002. He has worked with Nokia 1994–2004 in several research and research management positions with focus on future Internet, mobile networks, applications, services and service architectures. He was a Research Fellow on Internet Technologies in Nokia Research Center until 2008. His research interests include Internet technologies, application and service development in Internet, and network architectures.