Computer Networks 91 (2015) 548–563

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Constraint-based proactive scheduling for MPTCP in wireless networks

Bong-Hwan Oh, Jaiyong Lee*

UbiNet Lab, Department of Electrical and Electronic Engineering, Yonsei University, 134 Shinchon-Dong, Seodaemoon-Gu, Seoul 120-749, Republic of Korea

ARTICLE INFO

Article history:
Received 20 March 2015
Revised 8 July 2015
Accepted 3 September 2015
Available online 10 September 2015

Keywords:
Multipath TCP
Packet scheduling
Buffer blocking problem

ABSTRACT

Multipath TCP (MPTCP) is one of the leading protocols that support multipath operation in a transport layer. However, depending on the network and the receiver buffer, the original MPTCP can experience throughput degradation, underutilizing the network capacity compared to the regular TCP. Furthermore, MPTCP can result in a large packet interval. In this paper, we propose a new scheduling scheme for MPTCP that performs packet scheduling according to the receiver buffer and network delay. Our scheme estimates out-of-order packets according to performance differences between subflows and assigns data packets to subflows by comparing the estimated out-of-order packets and the buffer size. Moreover, our scheme can adjust the trade-off between throughput and delay performance using a delay constraint. We implement the proposed scheduling in the Linux kernel and evaluate its performance over a virtual network framework using NS-3 and real networks. The results show that the proposed scheduling scheme performs efficient packet transmission regardless of the performance differences of multiple paths and buffer size. Moreover, the proposed scheduling can complement and cooperate with an existing non-scheduling-based solution.

1. Introduction

New approaches for using access networks have been proposed with the rapid advancement of portable devices. Unlike older devices, which used only a selected network, current portable devices are equipped with multiple radio interfaces that can select and use a specific access network. In addition to the selection of a specific network, new initiatives to concurrently use multiple access networks seek to provide better network services. One instance such work is the announcement by SK Telecom and KT (the main network operators in South Korea) that they will launch the commercial services Hybrid Network Integration System (SK telecom) and KT GIGA path, which use long-term evolution (LTE) and WiFi networks simultaneously.

Unfortunately, the current dominant transport layer protocols such as TCP and UDP cannot use and control multiple paths autonomously. The main problem with the existing protocols is that applications cannot change a communication session to another path. Thus, in order to manage multiple paths with current protocols, applications should perform multiple interface management, which increases the workload of applications.

One promising solution to this problem is the application of multipath TCP (MPTCP) [1], which is a TCP extension version that is being standardized by the Internet Engineering Task Force (IETF). MPTCP provides concurrent data transmission over multiple paths and supports compatibility with a single TCP. Thus, by using MPTCP, applications can utilize multiple interfaces without additional burden.

* Corresponding author. Tel.: +82 221232873.
E-mail addresses: crusader27@yonsei.ac.kr (B.-H. Oh), jyl@yonsei.ac.kr (J. Lee).

http://dx.doi.org/10.1016/j.comnet.2015.09.002
1389-1286/© 2015 Elsevier B.V. All rights reserved.

One of the concerns caused by the extension from single TCP to MPTCP is packet scheduling for multiple paths. This affects the reordering problem, which is one of the challenges of multipath transmission. Thus, in order to provide efficient transmission in MPTCP, a scheduling scheme should be considered in order to maintain the number of reordering packets, which influences the overall transmission performance. Currently, two basic scheduling cases have been proposed to fully utilize all paths [1,3]. The first is a round-robin scheduler that utilizes subflows in sequence. The second is lowest-round-trip time (RTT)-first scheduling, which first assigns packets to the fastest subflow until its congestion window is filled with data, after which packets are then allocated to the other subflows. However, these cases present problems when applied to current wireless networks, and the results might not be viable. The current scheduling methods can generate a lower throughput (goodput) performance than single TCP when MPTCP with a limited buffer simultaneously uses multiple paths that have different delay characteristics [4,6]. This phenomenon occurs frequently, because promising wireless networks, including 3G (LTE) and WiFi, have quite different delay performances, and mobile devices do not always have sufficient memory. Furthermore, a large delay difference can generate a large packet interval, which affects the quality of service (QoS) of real-time applications.

In this paper, we propose a new scheduling scheme for MPTCP, which performs packet scheduling according to the receiver buffer and the network delay. The proposed scheme can prevent performance degradation by using multiple asymmetric paths and sufficiently utilizes the capacity of multiple symmetric paths. As a result, MPTCP with the proposed scheduling can perform efficient packet transmission regardless of the performance difference of multiple paths and buffer size. For the delay performance, the proposed scheme can maintain the packet interval by restricting the utilization of multiple paths. Moreover, the proposed scheduling scheme can be used with other solutions that are not scheduling-based and is sufficiently simple for practical implementation.

The rest of this paper is organized as follows. Section 2 describes a data transfer problem in MPTCP and discusses related work. Section 3 introduces the proposed scheduling scheme. Section 4 presents the results of performance evaluation, robustness of the proposed scheme, and cooperation with an existing solution. Section 5 presents the measurement results in real networks. Finally, we offer our conclusions in Section 6.

2. Background and related work

Fundamentally, MPTCP consists of two sub-layers: the MPTCP layer and the subflow layer. The MPTCP layer provides reliability and application compatibility by preserving the TCP-like semantics of global ordering of the application data, whereas the subflow layer provides network compatibility by appearing and behaving as a TCP flow in the network [2]. First, the MPTCP layer establishes the connection over the subflow layer and discovers available paths. The MPTCP layer receives a byte stream from an application and allocates application data to the subflows. Each subflow transmits the allocated data segment and performs congestion control according to its network condition. Although the data segment allocated to a subflow is managed in the subflow, all of the application data is managed using a single buffer and data sequence number (DSN) at the MPTCP connection in order to support ordered and reliable delivery. In other words, although each subflow operates independently regarding network-dependent functions, it can affect the other subflows through their combination with application-dependent functions.

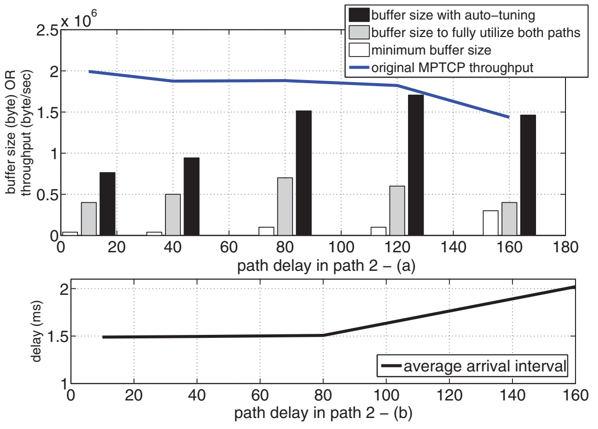


Fig. 1. Buffer and arrival interval consideration in original MPTCP.

Flow control is one of the application-dependent functions that are influenced by all subflows according to an extension from single TCP to MPTCP. Basically, the flow control operation in single TCP is the solution to one of the problems caused by a difference in processing performance between end devices. However, in MPTCP, flow control can be performed due to the difference in network delay performance, because the packets transmitted to a slow path hinder transmission in a fast path. Although packets transmitted to a fast path are sent later than packets transmitted to other paths, they can be out-of-order due to the difference in delay among multiple paths, which reduces the receiver window (RWND). Consequently, the sender performs flow control, which interrupts transmissions on the fast path, which is called the buffer blocking problem in this paper. Although this phenomenon does not occur when MPTCP has a sufficient buffer [4], the required buffer size can be extremely large when MPTCP simultaneously uses multiple paths that have different delay characteristics [5,6].

One of the main considerations for solving the buffer blocking problem is the required buffer size needed to utilize multiple paths. Fig. 1a shows the effect of delay asymmetry on MPTCP buffer size. In this simulation environment, the original MPTCP uses two paths: path 1 has a fixed performance (8 Mbps bandwidth and 10 ms path delay), and path 2 has a fixed bandwidth (8 Mbps) but a path delay that varies between 10 ms and 160 ms. The original MPTCP does not employ additional buffer-blocking solutions such as opportunistic retransmission or penalization [6,7]. The gray bar indicates the required buffer size needed to fully utilize both paths, and the white bar shows the minimum buffer size needed to achieve the same or better throughput than single TCP over the fastest path. The black bar indicates the saturated buffer size according to an auto-tuning algorithm [8] when the maximum buffer size is 4 MB.

Generally, the buffer size in all cases tends to increase with increasing path delay in path 2 because of the increase in out-of-order packets due to the delay difference. Furthermore, this phenomenon can be more severe with a difference in throughput performance. If the buffer size in MPTCP is less than the minimum buffer size, MPTCP can have a lower throughput performance than single TCP over the fastest path. This phenomenon can be critical in wireless networks because variation in wireless path performance is inevitable, and mobile devices have restricted memory. However, although MPTCP cannot have a sufficient buffer to utilize multiple paths, if MPTCP can determine whether to use multiple paths or a single path on the basis of the estimated minimum buffer size, MPTCP can achieve the same or better throughput compared to single TCP over the fastest path, which satisfies the MPTCP functional goals [2].

Similar to TCP, MPTCP employs an auto-tuning algorithm [8], which sets the buffer size to $2 * \sum_i^n bw_i * \mathrm{RTT}_{\max}$ in order to fully utilize all of the subflows. bw_i indicates the bandwidth in subflow i, n represents the number of subflows, and $\mathrm{RTT}_{\max}$ refers to the RTT of the slowest subflow. As shown in Fig. 1a, MPTCP with an auto-tuning algorithm can achieve a sufficient buffer size with an adequate maximum buffer size (4 MB). Although the auto-tuning algorithm can disrupt the increasing congestion window, which has already been studied [7], MPTCP with auto-tuning seems to sufficiently increase the buffer size with a long transmission period. However, because the auto-tuning algorithm operates within a maximum buffer size, if the maximum buffer size is insufficiently bounded due to a memory limit, the problem mentioned above is a concern.

The other consideration in multipath transmission is the delay issue. Fig. 1b shows the results of the average arrival interval in MPTCP according to the path delay in path 2. The simulation environment is the same as in Fig. 1a. The arrival interval is the time period between the arrival of in-order packets at the receiver. As shown in Fig. 1b, the arrival interval increases with increases in delay difference between paths, although MPTCP has a sufficient buffer (4 MB) and does not create network reordering or packet losses. This effect is only caused by multipath transmission with large delay differences between paths. Because delay can be more important than throughput in real-time traffic, in special cases, delay performance should be maintained even though MPTCP cannot improve the throughput performance.

Recently, several schemes based on MPTCP have been proposed to prevent goodput degradation [6,7,9–12]. [6,7] propose opportunistic retransmission and a penalizing congestion window method that can solve transmission interruption due to asymmetric multiple paths. In [9], systematic coding is applied to MPTCP in order to improve not only the throughput, but also the delay jitter performance. In [10], congestion window adaptation and a proactive scheduler are proposed to prevent goodput degradation. However, the weak point in these previous solutions is that they require additional operations with network resources or modification of network-dependent functions. In [6,7] and [10], the congestion window can be modified regardless of network congestion. Additional retransmissions can be performed without packet loss [6,7]. In [9], additional operations for applying the coding scheme are required. Although some transmission strategies have been proposed in terms of delay [3,29], these schemes mainly focus on the buffer blocking phenomenon.

In the Stream Control Transmission Protocol (SCTP) [18], a problem similar to that mentioned above can be generated when SCTP provides multipath transmission [19,20]. To solve the problem, a retransmission-based solution [19,22], path selection method [21], bandwidth estimation scheme [22], and rescheduling scheme [23] have been proposed. In [19], several retransmission policies are proposed to solve the problem caused by spurious retransmission. [21] proposes a path-selection method that selects a data transmission path or a retransmission path according to the receiver buffer and path conditions. However, it is not a sufficient solution when all of the subflows are fully used. In [22], partial reliability retransmission and bandwidth estimation based on Westwood congestion control [24] are proposed for real-time traffic. However, that method does not consider a limited buffer case. In [23], the chunk to block the sending window can be rescheduled in order to eliminate the effect of asymmetric paths. However, this method might be applicable in concurrent multipath transfer (CMT)-SCTP because a selective acknowledgement (SACK) [25] operation is required at the data sequence level [23]. It is expected (but not mandated) that SACK be used at the subflow level in order to improve the efficiency of the current MPTCP [1].

One solution is to schedule application data to subflows in order to independently operate both application-dependent functions and network-dependent functions. In [3,13], the impact of scheduling in MPTCP is evaluated, and MPTCP is shown to improve performance by scheduling strategies considering network conditions. Scheduling is a secure methodology because it is not affected by any network-dependent functions such as congestion control or loss detection; in other words, it can be applied without any modification of network-dependent functions. Because of the independence of network-dependent functions, the scheduling method can be employed with other solutions that are not associated with scheduling. Moreover, the scheduling methodology does not require network resources. Some scheduling schemes [11,12] are proposed to prevent goodput degradation. However, in [11], subsequent packets can be sent before the current packet, which means that sufficient data should be buffered beforehand by an application in MPTCP. Although performance degradation can be eliminated by disabling the underperforming subflow, as shown in [12], the decision rules for utilization additional paths and path probing are static and are only based on the throughput, which cannot deal with the various degradation cases.

Thus, our goal is to propose a new scheduling scheme that schedules application data to subflows according to network delay performance and residual buffer size in order to achieve efficient concurrent transmission over multiple paths regardless of the difference in network performance and MPTCP buffer size. Furthermore, a scheduling strategy for delay performance is considered according to the delay difference between paths.

3. Constraint-based proactive scheduling (CP scheduling)

To summarize the problem mentioned in the previous section, the buffer blocking problem occurs when a receiver (sender) buffer is filled with out-of-order packets, and the utilization of the path with a large delay can cause degradation of delay performance in the MPTCP layer. Thus, the main feature of the proposed scheme is that it schedules application data to subflows according to network delay performance and residual buffer size in order to prevent the buffer from filling with out-of-order packets. With regard to delay performance, the proposed scheme can adjust the utilization of multiple paths according to the path delay. The questions that arise from buffer blocking are the extent to which the out-of-order packets cause transmission interruptions, and with this information, assignment of the application data to subflows. In response to these questions, a basic scheduling strategy is assigned first, because the number of out-of-order packets is dependent on the scheduling. We then estimate the number of out-of-order packets based on the scheduling and congestion control. Finally, we propose a new scheduling scheme based on the estimation. For delay performance, the proposed scheme additionally determines the utilization of subflows according to the delay constraint.

3.1. Basic scheduling strategy in CP-scheduling

In the proposed scheduling method, the lowest-RTT-first scheduling, which is the default scheduling strategy of the original MPTCP [1,3], is employed as the basic scheduling strategy. The advantage of this scheduling scheme is that it is less influenced by slow subflows than the round-robin scheduling scheme. Note that the basic scheduling strategy in the proposed scheme performs the same operation as the original MPTCP only when MPTCP can fully utilize all of the paths.

In the lowest-RTT-first scheduling, only the subflow with the minimum delay performance is identified. However, in order to estimate the number of out-of-order packets, all subflows are first ranked in ascending order according to the RTT. In the proposed scheme, the estimated RTT values in subflows are used to determine the rank order; if some subflows do not have any RTT estimation values, packets are assigned to them according to the default scheduling in the original MPTCP. The new parameter N_i is defined to notify subflows according to the RTT rank order and indicates the subflow index of the ith order in the RTT rank.

3.2. Determination of required buffer size

In this scheduling scheme, out-of-order packets occur when packets that have been transmitted in fast subflows arrive at the receiver before packets transmitted in the slow subflow. Thus, when data from subflow i to subflow j are used, the expected out-of-order data, $E_{i,j}$, are represented by

$$E_{i,j} = \sum_{i=1}^{j-1} T_i * \frac{\mathrm{RTT}_j - \mathrm{RTT}_i}{2}, \tag{1}$$

where T_i refers to the throughput of subflow i, and RTT_i indicates the RTT of subflow i. We assume that all subflows are

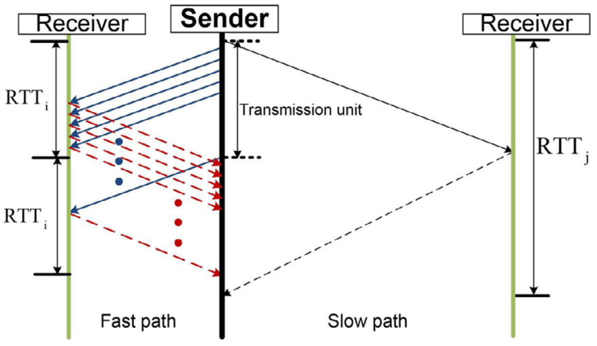


Fig. 2. Estimation of the expected out of order packets when transmission occurs in the slow subflow.

sorted in ascending order according to the RTT for ease of expression, although CP scheduling can be performed with an arbitrary arrangement of subflows. The one-way delay is briefly calculated as half of the RTT.

If the buffer size is $E_{i,j}$, although buffer blocking cannot arise at the receiver, the sender cannot send additional data because the sender does not know that the blocking phenomenon is solved at the receiver until receiving the ACK that was transmitted after the blocking was resolved. Therefore, in the worst case, the required buffer size to prevent performance degradation is

$$B_{i,j} = \sum_{i=1}^{j-1} T_i * \mathrm{RTT}_j. \tag{2}$$

To calculate $B_{i,j}$, we determine the throughput per time in each subflow, which is affected by the congestion control. Thus, we calculate $B_{i,j}$ according to the congestion control method.

3.2.1. Using uncoupled and coupled congestion control

Instead of estimating the throughput in each subflow, we calculate the number of out-of-order packets based on the congestion control behavior. The new parameter $L_{i,j}$ is defined to briefly estimate the number of out-of-order packets in subflow i during RTT_j,

$$L_{i,j} = \mathrm{Floor}\left(\frac{\mathrm{RTT}_j}{\mathrm{RTT}_i}\right) - 1, \text{ when } \frac{\mathrm{RTT}_j}{\mathrm{RTT}_i} \geq 2$$
$$L_{i,j} = 0, \text{ when } \frac{\mathrm{RTT}_j}{\mathrm{RTT}_i} < 2 \tag{3}$$

when $j > i$. For a rapid estimation, the number of transmitted packets during RTT is assumed to be one transmission unit, as shown in Fig. 2. With this assumption, $L_{i,j}$ represents the expected number of transmission units that are associated with out-of-order packets in subflow i during RTT_j. To avoid an excessive transmission restriction, $L_{i,j}$ is set to zero when the number of out-of-order packets during RTT_j is less than the number of transmitted packets during RTT_i.

On the basis of $L_{i,j}$, when subflows i and j are used concurrently, an expected out-of-order packet $D_{i,j}$ to subflow i during RTT_j can be generated according to the congestion phase,

$$D_{i,j} = \sum_{m=1}^{L_{i,j}} P_i * 2^{m-1}, \text{ when } L_{i,j} \geq 1$$
$$D_{i,j} = 0, \text{ when } L_{i,j} = 0 \quad (4)$$

in slow start phase

$$D_{i,j} = \sum_{m=1}^{L_{i,j}} P_i + \text{ICA}, \text{ when } L_{i,j} \geq 1$$
$$D_{i,j} = 0, \text{ when } L_{i,j} = 0 \quad (5)$$

$$\text{ICA} = m - 1 \text{ in uncoupled congestion control}$$
$$\text{ICA} = (m-1) * \frac{\alpha * \text{cwnd}_i}{\text{cwnd}_{\text{total}}} \text{ in coupled congestion control}$$
$$\text{ICA} = 0, \text{ when } P_i < \text{CWND}_i$$

in congestion avoidance phase.

P_i indicates the outstanding packets in subflow i, cwnd_i is the congestion window of subflow i, and $\text{cwnd}_{\text{total}}$ is the sum of the congestion windows of all subflows in the connection [14]. ICA is the window increment according to the congestion control in the congestion-avoidance phase. When some subflows cannot utilize their entire congestion window, ICA is set to zero. Eventually, when using data from subflow i to subflow j, the required buffer size, $B_{i,j}$, can be calculated as follows:

$$B_{i,j} = \sum_{i=1}^{j-1} D_{i,j} * \text{MSS}_i \quad (6)$$

where MSS_i is the maximum segment size on subflow i.

3.2.2. Using other congestion control methods

The proposed scheme can be utilized with other congestion control methods by estimating the required buffer size for a specific congestion control scheme. As an example, a congestion window can be estimated without packet loss events during the RTT of a slow subflow for conservative operation. In ACK-based congestion control [15], $D_{i,j}$ can be estimated using the same method as in the previous subsection. An additional consideration is determining ICA for a specific congestion-control method. In rate-based congestion control [16], the throughput in each subflow can be obtained with MSS_i and the transmission rate of each subflow. Thus, $B_{i,j}$ can be estimated using (2). In time-based congestion control [17], when the packet sending time is t_0, $D_{i,j}$ can be estimated using $L_{i,j}$,

$$D_{i,j} = \sum_{m=0}^{L_{i,j}} w_i(t_0 + m * \text{RTT}_i) \quad (7)$$

where $w_i(t)$ is the congestion window size at time t. Finally, $B_{i,j}$ is obtained by substituting (7) into (6).

3.3. Operation of CP scheduling

The main feature of CP scheduling is to schedule packets to subflows in order to prevent buffer blocking according to network delay performance and residual buffer size. To ensure this feature, the CP scheduler assigns the packets to subflows on the basis of $B_{i,j}$. If the required buffer size for path j, $B_{1,j}$, is greater than the available buffer size, the current buffer is insufficient for utilizing path j with other paths with better delay performance. In other words, the delay performance in path j is insufficient for utilizing path j with the other paths.

Basically, the required buffer size ($B_{i,j}$) is the predictive value under the current path conditions, which means $B_{i,j}$ can be overstated or understated. Because CP scheduling is operated based on $B_{i,j}$, there are two methods to employ CP scheduling: the radical method and the conservative method. Radical CP scheduling can calculate the required buffer with underestimation, which can prevent underutilization of the aggregation bandwidth. In contrast, the conservative CP scheduling might overestimate the required buffer, which can prevent buffer blocking due to excessive transmission.

First, the basic strategy in radical CP Scheduling is summarized as follows.

1. The radical CP scheduler assigns packets in the sending buffer to the fastest subflow, (N_1), until the congestion window of the fastest subflow, (N_1), is filled with packets (the same operation as the lowest-RTT-first scheduling).
2. Packets are assigned in the next path, j, only when the available buffer $> B_{1,j}$ (radical method).

Because $B_{i,j}$ is estimated in order to avoid an excessive transmission restriction, the radical CP scheduling is performed only with $B_{i,j}$.

Second, the basic strategy in conservative CP scheduling is represented as follows.

1. The conservative CP scheduler assigns packets in the sending buffer to the fastest subflow, (N_1), until the congestion window of the fastest subflow, (N_1), is filled with packets (the same operation as the lowest-RTT-first scheduling).
2. If the fastest path is in the slow start phase or 2*outstanding packets in the fastest path $>$ RWND, the fastest path is only employed for packet transmission.
3. If 2*outstanding packets in the fastest path $\leq$ RWND, the packets are assigned in the next path, j, only when the available buffer $> B_{1,j}$ + total outstanding bytes in MPTCP (conservative method).

In order to perform conservative transmission, the conservative CP scheduler only utilizes the fastest path when the fastest path is in the slow start phase. If the receiver has a smaller buffer size than 2*outstanding packets in the fastest path, the conservative CP scheduler also utilizes only the fastest path because the receiver buffer size might be insufficient even for fastest path in the auto-tuning perspective. Furthermore, in order to conservatively utilize multiple paths, the conservative CP scheduling uses the outstanding bytes in MPTCP as a guard factor.

Although CP scheduling based on buffer size can prevent buffer blocking, it cannot guarantee delay performance. In order to consider delay performance, a delay constraint, D_c, can be optionally applied. For example, if path *j* is not fastest path and RTT_j is greater than the delay constraint, CP scheduling using the delay constraint does not assign packets to path *j* although MPTCP has a sufficient buffer to utilize path *j*. The delay constraint (D_c) can be set to a static value or a dynamic value, as for the RTT in other paths.

1. The packets are assigned in the next path, *j*, only when $\text{RTT}_j < D_c$ (optional).

3.4. Path probing scheme for CP scheduling

Although the above operations can prevent throughput degradation, path probing is required for network dynamics. If the CP scheduler without path probing prohibits packet transmission in path *j*, path *j* can be continuously disabled even if the RTT in path *j* becomes sufficiently small for utilizing other paths. This is because the RTT estimation in path *j* is not performed after packet transmission is restricted. In order to estimate the RTT in path *j*, packet transmission is periodically allowed for path *j* after it is restricted. When CP scheduler decides to deny packet transmission in path *j*, the CP scheduler records the current time (T_j^{pause}), the first sequence in the current sending window (S_j^{pause}), and the number of transmission pauses in path *j* (N_j^{pause}). Then, the CP scheduler cannot assign packets to path *j* during T_j^{guard}, which represents the guard time needed to enable path *j*,

$$\begin{aligned} T_j^{\text{guard}} &= \beta * \text{RTT}_j \quad \text{(static probing)} \\ T_j^{\text{guard}} &= \beta * \text{RTT}_j * N_j^{\text{pause}} \quad \text{(dynamic probing)} \qquad (8) \\ T_j^{\text{guard}} &= T_{\max}^{\text{guard}}, \quad \text{when} \quad T_j^{\text{guard}} \geq T_{\max}^{\text{guard}}. \end{aligned}$$

There are two path-probing schemes: static path probing and dynamic path probing. Static path probing estimates the path RTT in a static period. In contrast, dynamic probing can increase the probing period. In dynamic probing, T_j^{guard} increases whenever transmission pauses occur in path *j* due to CP scheduling. β determines the initial probing period, which should be greater than one, and $T_{\max}^{\text{guard}}$ indicates the maximum probing period. After T_j^{guard}, if the available buffer size is still less than $B_{1,j}$ and the current first sending sequence in the sending window (S_j^{first}) is greater than S_j^{pause}, the CP scheduling assigns packets to path *j* for path probing, which does not mean that path *j* is recovered. $C_j^{\text{estimation}}$ indicates the number of packets for path probing in path *j*, and the default value of $C_j^{\text{estimation}}$ is set to initial CWND. Then, the T_j^{pause}, S_j^{pause}, and N_j^{pause} values are updated. If the current first sending sequence in the sending window (S_j^{first}) is equal to or less than S_j^{pause}, the sending window remains blocked. In this case, path *j* is consistently prohibited, although the current time is greater than $T_j^{\text{pause}} + T_j^{\text{guard}}$. When the available buffer becomes greater than $B_{1,j}$ or $B_{1,j}$ + total outstanding bytes of MPTCP in the radical and conservative methods, respectively, the CP scheduling assigns a packet to path *j* for packet transmission. Fig. 3 shows the overall algorithm of CP scheduling.

```
for all subflows $j$ do
    if $cwnd$ in fastest subflow is not filled then
        fully utilize fastest subflow
    else
        if phase in fastest subflow = slow start || 2*outstanding packets in
            fastest subflow > $RWND$ then
            (only for conservative scheduling)
            do not use the subflow $j$
        end if

        calculate $B_{1,j}$

        if $B_{1,j}$ > available buffer then
            if current time > $T_j^{pause} + T_j^{guard}$ && $S_j^{pause} < S_j^{first}$ then
                update the values ($C_j^{estimation}$, $T_j^{pause}$, $S_j^{pause}$, $N_j^{pause}$)
                do not assign packets to subflow $j$
            else if $N_j^{pause} > 1$ then
                (path probing scheme)
                if $C_j^{estimation} \leq 0$ then
                    stop the path probing in path $j$
                else
                    send a packet for path probing
                    $C_j^{estimation} = C_j^{estimation} - 1$
                end if
            else
                do not assign packets to subflow $j$
            end if

        else if $B_{1,j}$ + total outstanding bytes > available buffer then
            (only for conservative scheduling)
            do not use the path $j$
        end if

        if $RTT_j > D_c$ then
            (only for delay constraint)
            do not use the path $j$
        end if

        utilize subflow $j$

    end if
end for
```

Fig. 3. Constraint-based proactive scheduling algorithm.

4. Measurement results

In this section, we evaluate the performance of MPTCP with CP scheduling using measurements over a virtual network framework. We implement CP scheduling in the MPTCP Linux kernel implementation with version 0.88 [26]. CP scheduling is executed when the MPTCP path manager is set to ‘fullmesh’, which produces a full mesh of subflows among all of the available subflows. For CP scheduling, the radical method and conservative method are employed. The radical CP scheduler uses dynamic probing and a β value of four. The conservative CP scheduler uses static probing and a β value of four. The maximum probing period ($T_{\max}^{\text{guard}}$) is set to 2 s. For comparison, the original MPTCP (version 0.88) uses the default scheduling strategy when the path manager is set to ‘fullmesh’. In order to only consider the effect of scheduling, the opportunistic retransmission and penalization method [6,7] is disabled. Both MPTCP with CP scheduling and the original MTPCP use uncoupled congestion control with Reno. We generate a virtual network topology using NS-3 with direct code execution (DCE), which is a framework for executing the real kernel and application code from within ns-3 [27]. The network topology is shown in Fig. 4. The client and server are connected through two disjoint routers, and the links between routers and the server have sufficient performance (100 Mbps and a 1-ns delay). One link between the

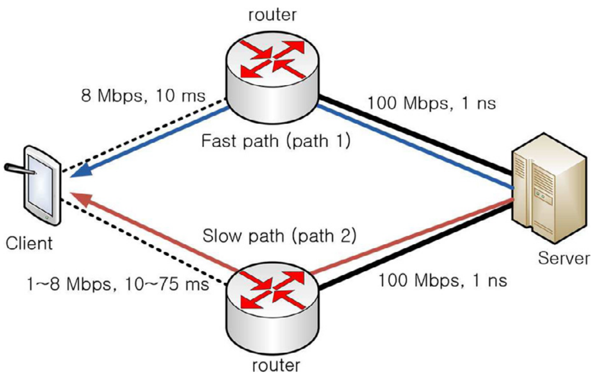


Fig. 4. Virtual network topology.

Table 1
Buffer size parameters.

Maximum buffer size (KB)	40, 60, 100, 200, 250, 300, 350, 400, 500, 600, 800 ,1000

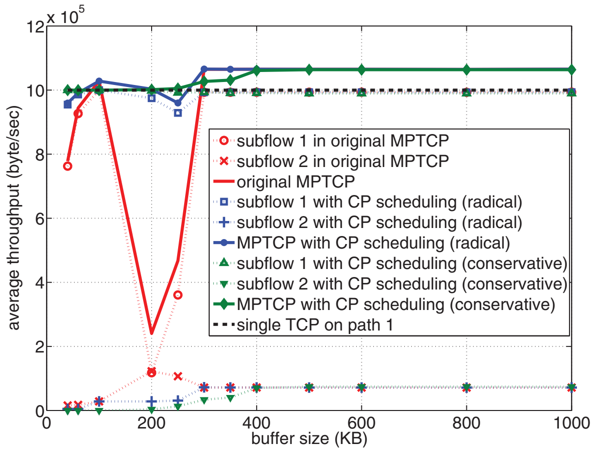


Fig. 5. Average throughput over an 8 Mbps-10 ms and a 1 Mbps-75 ms paths.

client and the router has fixed link performance (8 Mbps with a 10-ms delay and no error rate), and the other link has variable link performance (1–8 Mbps with a 10–75-ms delay). The queue type and size in each link are Drop-Tail and 100 packets, respectively. The application runs for 100 s to allow the flows to reach equilibrium. An auto-tuning algorithm is enabled, and the maximum buffer size for buffer auto-tuning varies between 40 KB (the required buffer size for single TCP running on the path with 8 Mbps and a 10-ms delay) and 1 MB. The parameters for buffer size are shown in Table 1.

4.1. Performance evaluation in constraint-based proactive scheduling

4.1.1. Buffer consideration with symmetric and asymmetric paths

First, we investigate the effect of buffer size on the original MPTCP and MPTCP with CP scheduling. In these scenarios, the buffer size varies between 40 KB and 1 MB. In the first scenario, MPTCP uses multiple paths that have quite different delay characteristics; then, the scenario in which MPTCP uses multiple symmetric paths is considered.

Fig. 5 shows the average throughput of the original MPTCP and the MPTCP with CP scheduling over an 8 Mbps-10 ms path and a 1 Mbps-75 ms path according to buffer size. In Fig. 5, the solid lines indicate the original MPTCP results; the solid lines with dots represent the results of MPTCP with radical CP scheduling, and the solid lines with diamonds indicate the results of MPTCP with conservative CP scheduling. Dotted lines with circles indicate the results of path 1 in the original MPTCP. The dotted lines with squares correspond to path 1 in MPTCP with radical CP scheduling, and the dotted lines with triangles represent path 1 in MPTCP with conservative CP scheduling. Dotted lines with crosses show the results of path 2 in the original MPTCP; the dotted lines with plus signs correspond to path 2 in MPTCP with radical CP scheduling, and the dotted lines with inverted triangles represent path 2 in MPTCP with conservative CP scheduling. Dotted lines indicate the single TCP results of path 1. The results show that the original MPTCP has a lower average throughput than the single TCP running on path 1 when the buffer size is less than 300 KB, which is caused by the throughput degradation in path 1 of the original MPTCP. The reason for the throughput degradation in path 1 is that the transmission is disturbed by a transmission to path 2, as mentioned in Section 2. This phenomenon can be generated in a small buffer case (40 KB), which is the required buffer size for the single TCP on path 1. This is because the original MPTCP attempts to use multiple paths despite its excessively limited buffer size. In this case, the throughput in path 1 decreases without any increase in the throughput in path 2.

On the other hand, MPTCP with CP scheduling eliminates the throughput degradation when the buffer size is less than 300 KB by assigning the packet to the fastest path (path 1) and not assigning a packet to the slow path (path 2) if there is not have a sufficient buffer to use both paths. MPTCP with radical CP scheduling fully achieves the throughput performance in path 1 with a negligible impact, and the negligible impact can be eliminated in MPTCP with conservative CP scheduling. If the buffer size is greater than 300 KB, MPTCP with CP scheduling can achieve a throughput performance similar to that of the original MPTCP, which surpasses the single TCP throughput. This is because MPTCP with CP scheduling assigns the packet to the slow path if MPTCP has a sufficient buffer size with consideration of the RTT difference.

Fig. 6 shows the average RTT difference in the original MPTCP and MPTCP with CP scheduling over an 8 Mbps-10 ms path and a 1 Mbps-75 ms path according to buffer size. The average RTT difference is defined as the difference in average RTTs of the paths. The solid lines indicate the original MPTCP results. The solid lines with dots represent the results of MPTCP with radical CP scheduling, and the solid lines with diamonds represent the results of MPTCP with conservative CP scheduling. The dotted lines indicate the acceptable RTT difference, which is the maximum allowable RTT difference according to the buffer size in order to use both paths. The acceptable RTT difference is calculated with the maximum throughput in the fast path and buffer size. The average RTT difference in the original MPTCP increases

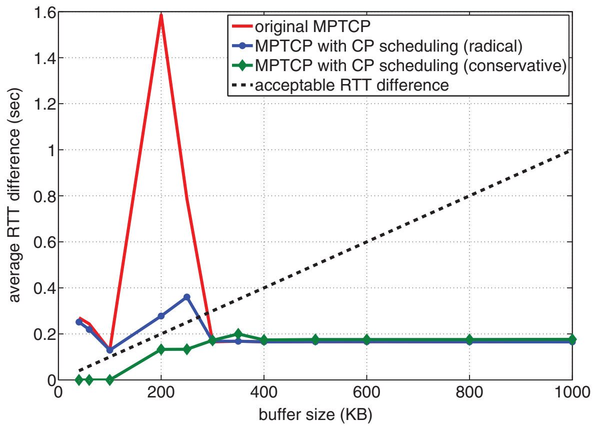


Fig. 6. Average RTT difference over an 8 Mbps-10 ms and a 1 Mbps-75 ms paths.

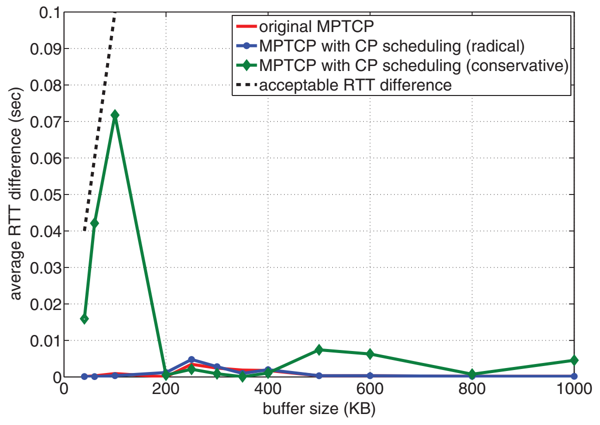


Fig. 8. Average RTT difference over two 8 Mbps-10 ms paths.

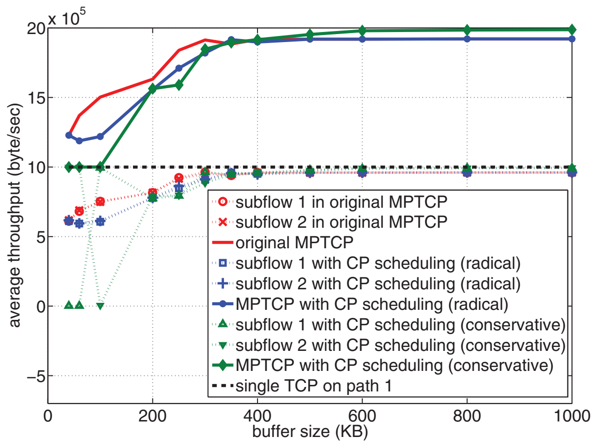


Fig. 7. Average throughput over two 8 Mbps-10 ms paths.

rapidly when the buffer size is greater than 100 KB, which is the starting point of throughput degradation in the original MPTCP. This increase in RTT difference is caused by transmission on the slow path. The average RTT difference of the original MPTCP exceeds the acceptable RTT difference when the buffer size is less than 300 KB, which means that the buffer size is insufficient for using both paths from a delay perspective. On the other hand, MPTCP with CP scheduling gradually increases the RTT difference. Although MPTCP with radical CP scheduling also exceeds the acceptable RTT difference when the buffer size is less than 300 KB, it is sufficiently smaller than that of the original MPTCP. MPTCP with conservative CP scheduling is less than the acceptable RTT difference in all cases. This means that MPTCP with CP scheduling cannot perform packet transmission to a slow path when the buffer size is insufficient for using both paths according to the RTT difference.

Fig. 7 shows the average throughput over symmetric paths (8 Mbps-10 ms path) according to buffer size. In contrast to the previous results, the original MPTCP has no throughput issues, regardless of the buffer size. Although each path in the original MPTCP cannot fully utilize the path bandwidth with a small buffer size, the original MPTCP outperforms the single TCP running on path 1, which satisfies the design goal of MPTCP [2]. MPTCP with CP scheduling also outperforms the single TCP running on path 1 regardless of the buffer size, although it can utilize less path bandwidth than the original MPTCP in some cases. MPTCP with conservative CP scheduling can only utilize a single path for conservative transmission when the buffer size is less than 100 KB. When the buffer size is sufficient for using both paths, MPTCP with conservative CP scheduling can achieve a throughput performance similar to that of the original MPTCP, which fully utilizes both paths.

Fig. 8 shows the average RTT difference over symmetric paths (8 Mbps-10 ms path) according to buffer size. In this case, the results of the average RTT difference in both the original MPTCP and MPTCP with CP scheduling are less than the acceptable RTT difference, which means that they have a sufficient delay performance for both paths. In conservative CP scheduling, the average RTT difference is relatively large compared to other cases due to the queueing delay in the only path utilized.

4.1.2. Performance effect according to variation in throughput and delay

Second, we investigate the effect of throughput and delay in path 2 on the original MPTCP and MPTCP with CP scheduling. Fig. 9 shows the effect of bandwidth in path 2 on the original MPTCP and MPTCP with CP scheduling. The performance of path 1 is 8 Mbps-10 ms. The bandwidth of path 2 varies between 1 Mbps and 8 Mbps, and the path delay is the same as that of path 1; the buffer size is 200 KB. The bar graphs indicate the average throughput in the connection; the dotted line graphs with marks represent the average throughput in path 1. The dotted line indicates the results of the single TCP in path 1. The results show that the average throughput in all cases tends to increase with increasing throughput in path 2. The original MPTCP can achieve better throughput than MPTCP with CP scheduling when the throughput in path 2 is greater than 3 Mbps. This is because continuous transmission can utilize more of the path bandwidth when the buffer blocking is negligible. However, the original MPTCP shows throughput degradation in path 1 due to buffer blocking when the throughput in path 2 is less than

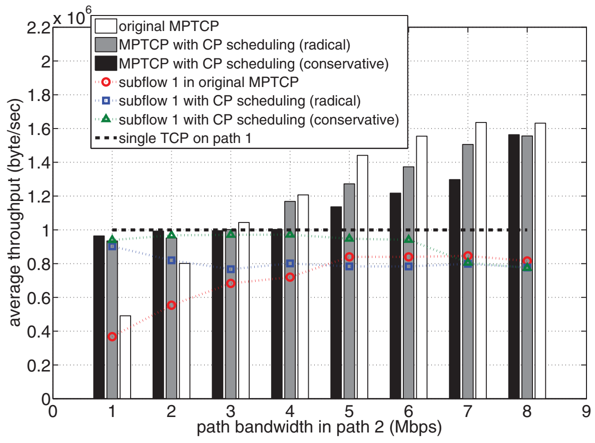


Fig. 9. Average throughput versus path bandwidth in path 2.

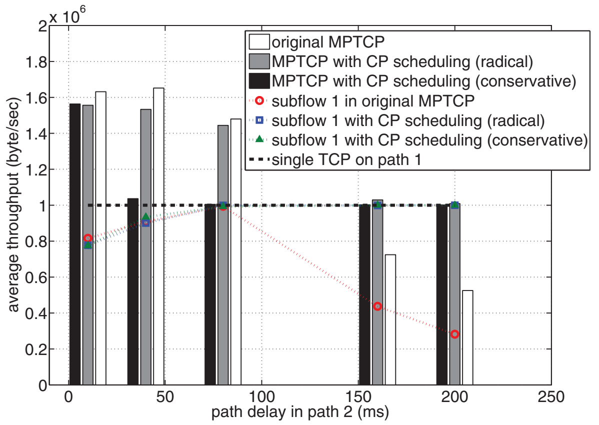


Fig. 11. Average throughput versus path delay in path 2.

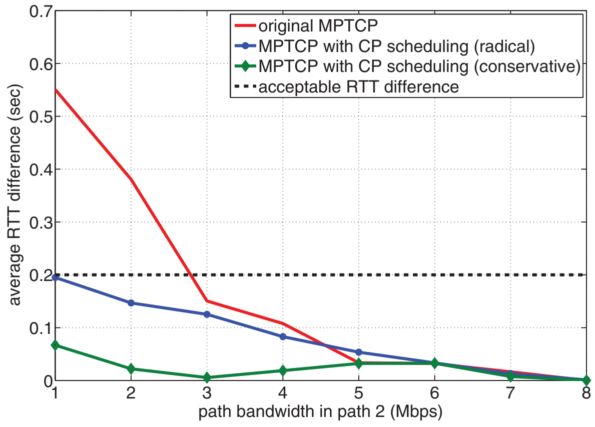


Fig. 10. Average RTT difference versus path bandwidth in path 2.

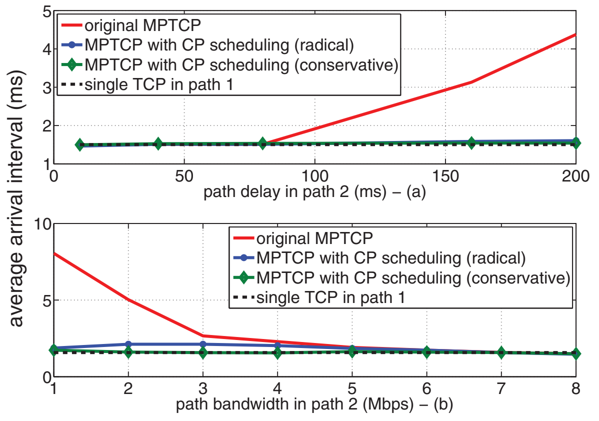


Fig. 12. Average arrival interval versus path delay (path bandwidth) in path 2.

3 Mbps, which causes the original MPTCP to have a poorer throughput performance than the single TCP in path 1. This is because the degradation of throughput in path 2 increases the delay in path 2, which increases the RTT difference between paths, as shown in Fig. 10. In these cases, the RTT difference between paths exceeds the average RTT difference, which means the RTT difference is insufficient for using both paths with this buffer size. Although MPTCP with CP scheduling has an inferior throughput performance compared with the original MTPCP when the throughput degradation is negligible, it nearly eliminates throughput degradation when the throughput in path 2 is less than 3 Mbps. As shown in Fig. 10, the average RTT difference in MPTCP with CP scheduling does not exceed the average RTT difference, which means that MPTCP with CP scheduling utilizes the additional path only when the RTT difference is acceptable for using both paths. Therefore, MPTCP with CP scheduling can at least achieve the throughput performance of the single TCP in path 1, regardless of the throughput difference between paths.

Fig. 11 shows the effect of path delay in path 2 on the original MPTCP and MPTCP with CP scheduling. The path 1 performance is 8 Mbps-10 ms. The delay in path 2 varies between 10 and 200 ms, and the bandwidth is the same as that of path 1; the buffer size is 200 KB. The representation of the results is the same as in Fig. 9. When the delay difference between paths is relatively small (the delay in path 2 is less than 80 ms), the original MPTCP has better throughput than MPTCP with CP scheduling. In this case, MPTCP with conservative CP scheduling might be vulnerable to fully utilize multiple paths. This is because MPTCP with conservative CP scheduling does not utilize additional paths although MPTCP can improve the throughput performance with the negligible buffer blocking. Although the two paths are symmetric, buffer blocking can be generated instantaneously in a small buffer case, which is the reason why MPTCP with radical CP scheduling has less throughput than the original MPTCP when the path delay in path 2 is 10 ms. When the delay difference between paths is relatively large (the delay in path 2 is greater than 80 ms), MPTCP with CP scheduling eliminates the throughput degradation by nearly disabling path 2, although the original MPTCP has less throughput than the single TCP in path 1, which is caused by buffer blocking due to the delay difference.

4.1.3. Delay performance according to variation in throughput and delay

In this section, we investigate the delay performance according to throughput and delay variations. Fig. 12 shows

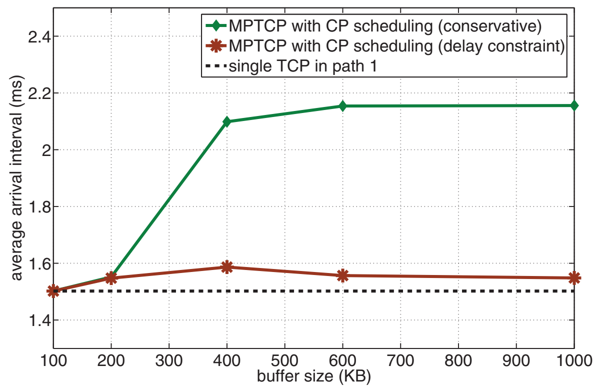


Fig. 13. Average arrival interval versus buffer size.

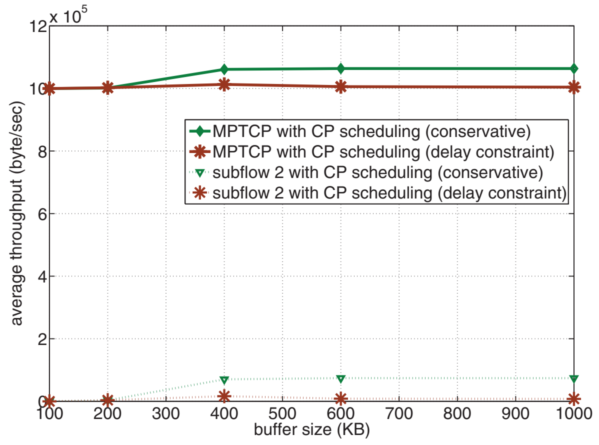


Fig. 14. Average throughput versus buffer size.

the results of average arrival interval according to path delay and bandwidth in path 2. The simulation environment is the same as in Figs. 9 and 11. Although the original MPTCP experiences not only a buffer blocking (in Figs. 9 and 11) but also a relatively large arrival interval when the path delay in path 2 is greater than 80 ms (Fig. 12a) or the path bandwidth is less than 3 Mbps (Fig. 12b), MPTCP with CP scheduling can maintain the arrival interval performance. This is because CP scheduling with a limited buffer restricts the utilization of a slow path (path 2) when the delay difference between the paths is large, which prevents buffer blocking and degradation of delay performance. As shown in Fig. 10, a reduction in path bandwidth also causes an increase in path delay. Although MPTCP with radical CP scheduling can generate a certain abnormal arrival interval when bandwidth between paths is large, this interval is relatively small compared to that of the original MPTCP.

Figs. 13 and 14 evaluate the effect of delay constraint in CP scheduling. Fig. 13 shows the results of average arrival interval according to buffer size. The effect of the delay constraint is observed using conservative CP scheduling, which has a better delay performance than radical CP scheduling. The simulation environment is the same as in Fig. 5. The solid lines with diamonds represent the results of MPTCP with conservative CP scheduling without a delay constraint, and

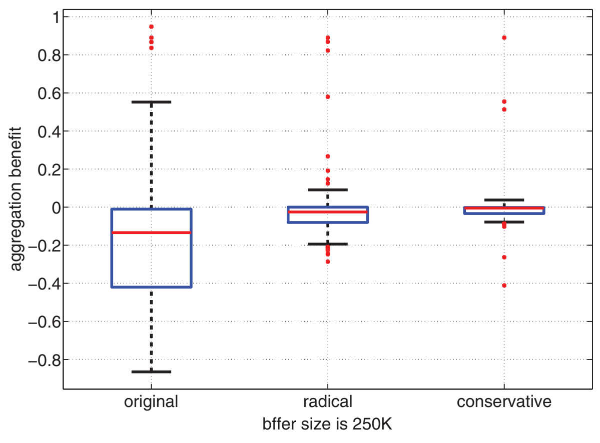


Fig. 15. Throughput in various network scenarios (buffer size is 250 KB) .

the solid lines with asterisks indicate the results of MPTCP with conservative CP scheduling with a delay constraint. The delay constraint is set to twice the RTT in the fastest path (path 1). In MPTCP with conservative CP scheduling without a delay constraint, the average arrival interval increases with increases in buffer size because it utilizes the slow path (path 2) without buffer blocking, as shown in Fig 14. In contrast, MPTCP with conservative CP scheduling with a delay constraint maintains the average arrival interval regardless of buffer size because it underutilizes the slow path (path 2) although the buffer size is sufficient for utilizing both paths, as shown in Fig 14.

To summarize the above results, when the RTT difference between paths is relatively large, the original MPTCP with a limited buffer size can suffer from throughput degradation caused by throughput degradation in the fast path (path 1) in the original MPTCP. The RTT difference can be generated by the throughput difference between paths. The degradation in throughput in a specific path seems to increase the RTT of the path, which can generate an RTT difference between paths. If the RTT difference between paths is relatively small, the original MPTCP outperforms the single TCP running on the fast path (path 1), regardless of the buffer size. MPTCP with CP scheduling achieves approximately the same or better throughput compared to the single TCP running on the fast path, regardless of buffer size, throughput difference, and RTT difference between paths. This is because MPTCP with CP scheduling fully utilizes an underperforming path only when MPTCP has a sufficient buffer or the RTT difference is sufficiently small for both paths to be used, which prevents throughput degradation in the fast path (path 1). Furthermore, with a delay constraint, CP scheduling can adjust the trade-off between throughput and delay performance.

4.2. Robustness in constraint-based proactive scheduling

4.2.1. Performance evaluation in wide network scenarios

Figs. 15 and 16 show the results with various network scenarios for the original MPTCP and MPTCP with CP scheduling. The results are represented by box-and-whisker plots. The simulation environment is organized based on [7]. In order to cover various scenarios, a space-filling design is

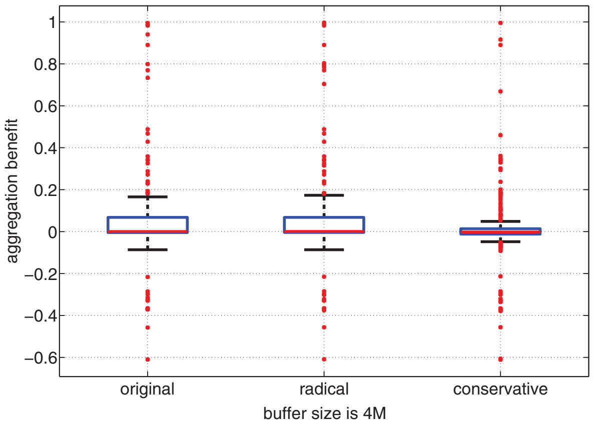


Fig. 16. Throughput in various network scenarios (buffer size is 4 MB).

Table 2
Parameter setting.

Parameter	Min	Max
Capacity (Mbps)	1	100
Path delay (ms)	1	75
Queueing (ms)	1	1500

employed using the SED Toolbox [28]. Each path is influenced by three factors, as shown in Table 2, indicating the need for 6-dimensional parameters. In order to attain reliable results, 200 individual parameters are applied, and the application runs for 60 s. Based on the aggregation benefit defined in [7], the modified aggregation benefit, Ben$_m$ (S), is employed to evaluate the throughput improvement

$$\mathrm{Ben}_m(S) = \frac{g - C^S_{\max}}{\sum_{i=1}^{n} C^S_i - C^S_{\max}},\ \text{when } g \geq C^S_{\max}$$
$$\mathrm{Ben}_m(S) = \frac{g - C^S_{\max}}{C^S_{\max}},\ \text{when } g < C^S_{\max}, \tag{9}$$

where S indicates a multipath aggregation scenario, n is the number of paths, C^S_i represents the single TCP goodput in path i, and $C^S_{\max}$ is the maximum TCP goodput among all paths. g indicates the measured MPTCP goodput.

Fig. 15 shows the modified aggregation benefit of the original MPTCP and MPTCP with CP scheduling when the buffer size is 250 KB. The original indicates the original MPTCP results. The radical and conservative represent the results of MPTCP with radical CP scheduling and MPTCP with conservative CP scheduling respectively. In a limited buffer case, the original MPTCP cannot typically obtain bandwidth aggregation gain because it tries to fully utilize all paths regardless of path performance, which can cause throughput degradation even though it might achieve an aggregation benefit in some cases. MPTCP with CP scheduling experiences less throughput degradation than the original MPTCP, which means CP scheduling can minimize the effect of throughput degradation. However, MPTCP with CP scheduling also hinders the effect of throughput increment.

Fig. 16 shows the modified aggregation benefit of the original MPTCP and MPTCP with CP scheduling when the buffer size is 4 M. In this case, the original MPTCP and MPTCP

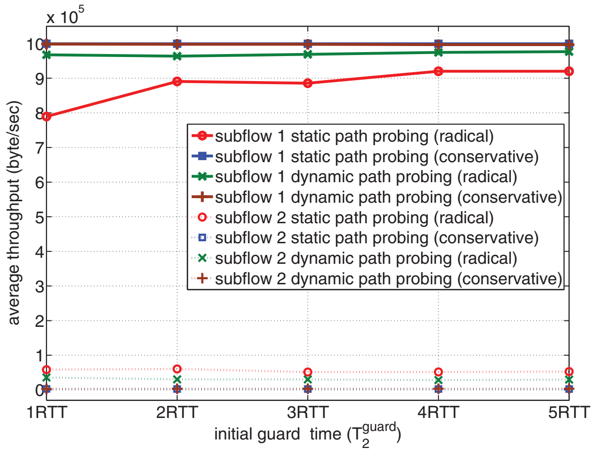


Fig. 17. Average throughput according to path probing schemes.

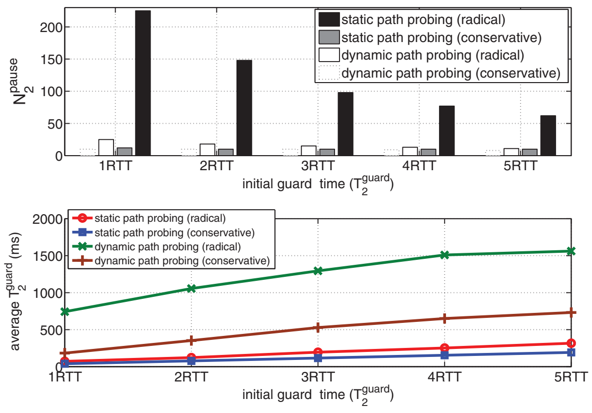


Fig. 18. Guard time and number of transmission pause according to path probing schemes.

with radical CP scheduling have similar results, which means they can utilize the same bandwidth in the fastest path or obtain bandwidth aggregation. MPTCP with conservative CP scheduling has similar results with small buffer cases, which means it might be vulnerable to obtain bandwidth aggregation. The results in Figs. 15 and 16 show that CP scheduling generally performs a conservative transmission, which minimizes the throughput degradation when MPTCP has a limited buffer. When the buffer size is sufficient, CP scheduling can achieve a similar performance to the original MPTCP. Based on these results, radical CP scheduling is more adaptable than conservative CP scheduling in terms of utilization of bandwidth in all paths, although it does not prevent throughput degradation as well as does the conservative method in limited buffer cases.

4.2.2. *Effect of path probing scheme in constraint-based proactive scheduling*

Figs. 17 and 18 illustrate the effect of path probing scheme on MPTCP with CP scheduling. Fig. 17 shows the throughput performance according to initial guard time and Fig. 18 shows the average guard time and the number of transmission pauses according to initial guard time. These results occur in the same simulation environment shown in Fig. 5, and the buffer size is set to 200 KB to show the effect of path probing scheme when buffer blocking occurs. Solid lines with dots and squares indicate the subflow 1 results of MPTCP with CP scheduling using static path probing, while solid lines with crosses and plus signs indicate the subflow 1 results of MPTCP with CP scheduling using dynamic path probing. Solid lines with dots and crosses indicate the results using radical CP scheduling, and solid lines with squares and plus signs represent the results using conservative CP scheduling. Dotted lines indicate the path 2 results associated with the above path 1 results, which are classified by the same marks. In the radical CP scheduling case, the throughput performance is changed based on the probing scheme as shown in Fig. 17. Radical CP scheduling with dynamic probing is better for eliminating the throughput degradation in subflow 1 than is radical CP scheduling with static probing. This is because dynamic probing increases the guard time (T_2^{pause}) and reduces the number of transmission pauses (N_2^{pause}), as shown in Fig. 18, which can minimize the blocking phenomenon due to path probing. Although radical CP scheduling with static probing has less throughput in subflow 1 than other cases, it greatly improves the throughput performance compared to the original MPTCP in subflow 1. In the radical CP scheduling with static path probing, the increase of initial probing period can eliminate the throughput degradation in subflow 1 because the increase of initial probing period also increases T_2^{pause} and decreases N_2^{pause}. In the conservative CP scheduling case, the throughput performance is almost the same regardless of the probing scheme and initial probing period. This is because conservative CP scheduling almost prevents the buffer blocking phenomenon. Consequently, T_2^{pause} and N_2^{pause} experience a smaller increase than in radical CP scheduling using static path probing, as shown in Fig. 18.

4.3. Cooperation with opportunistic retransmission and penalizing [6,7]

In this section, we investigate the effect of cooperation between CP scheduling and the opportunistic retransmission and penalizing method (OR&P). As mentioned in Section 2, CP scheduling can cooperate with other solutions that are not associated with scheduling. The OR&P method is a solution based on retransmission and congestion reduction, retransmitting the packet previously sent on another subflow and reducing another subflow's congestion window when the receiver buffer is blocked with packets sent on other subflows. Because no correlation exists between CP scheduling and OR&P, the two can cooperate with each other. Thus, we study CP scheduling as a complementary and cooperative method instead of a substitute method. In this section, the original MPTCP and MPTCP with CP scheduling are tested in conjunction with OR&P.

Fig. 19 shows the average throughput of MPTCP with OR&P and of MPTCP with OR&P and CP scheduling over an 8 Mbps-10 ms path and a 1 Mbps-75 ms path versus buffer size. The representation of the results is the same as in Fig. 5. The results show that MPTCP with OR&P outperforms the single TCP running on path 1 except in the small buffer case (40 KB), and that it can fully utilize both paths when the

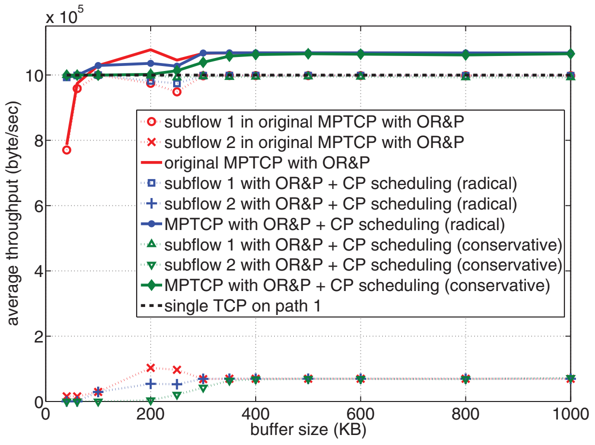


Fig. 19. Average throughput over 8 Mbps-10 ms and 1 Mbps-75 ms paths.

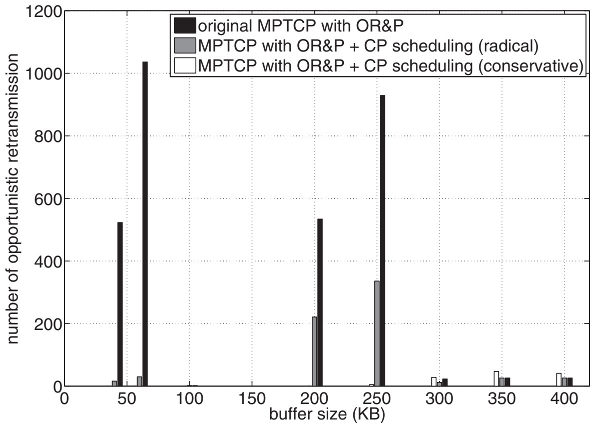


Fig. 20. Number of opportunistic retransmissions over 8 Mbps-10 ms and 1 Mbps-75 ms paths.

buffer size is greater than 200 KB. There is throughput degradation in the small buffer for the same reason as in the original MPTCP, although MPTCP uses OR&P. Moreover, in the small buffer case, OR&P can aggravate the throughput degradation. The degradation in the small buffer can be eliminated through cooperation with CP scheduling. The results show that MPTCP with OR&P and radical CP scheduling can better utilize both paths than MPTCP with radical CP scheduling because of the use of OR&P. Although MPTCP with OR&P and radical CP scheduling utilizes path 2 less than MPTCP with OR&P in some cases, this is not the result of throughput degradation. The reason for this is that radical CP scheduling conservatively performs the data transmission in path 2 to prevent throughput degradation, which affects the opportunistic retransmission, as shown in Fig. 20. MPTCP with OR&P and conservative CP scheduling has similar results to those in Fig. 5, which means the OR&P is rarely used, which is shown in Fig. 20.

Fig. 20 shows the number of opportunistic retransmissions over an 8 Mbps-10 ms path and a 1 Mbps-75 ms path versus buffer size. The black bar graph indicates the results of the MPTCP with OR&P. The gray bar graph depicts the results of MPTCP with OR&P and radical CP

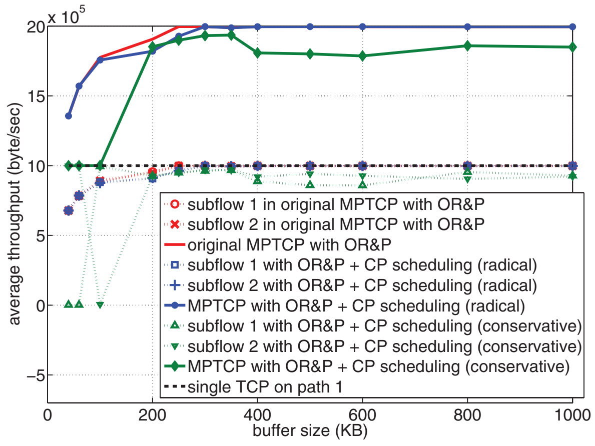


Fig. 21. Average throughput over two 8 Mbps-10 ms paths.

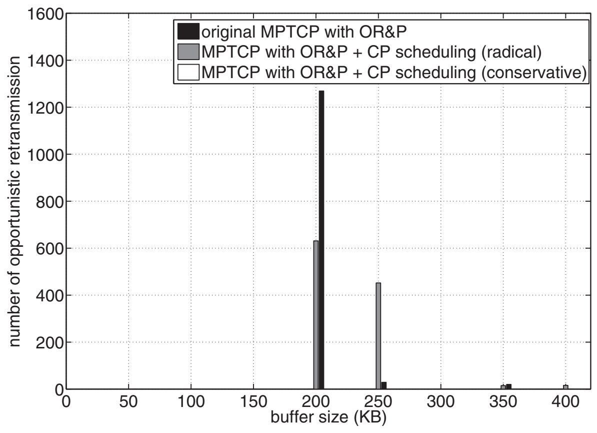


Fig. 22. Number of opportunistic retransmissions over two 8 Mbps-10 ms paths.

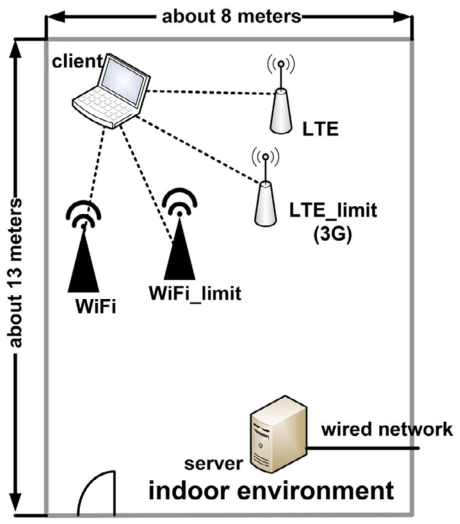


Fig. 23. Measurement environment.

scheduling, and the white bar graph depicts the results of MPTCP with OR&P and conservative CP scheduling. The results show that MPTCP with OR&P and radical CP scheduling performs opportunistic retransmission less often than MPTCP with OR&P in most cases because radical CP scheduling decreases the buffer blocking phenomenon. In other words, because radical CP scheduling can reduce the blocking phenomenon, and opportunistic retransmission is less frequent. MPTCP with OR&P and conservative CP scheduling almost eliminates the opportunistic retransmission with negligible impact, which means it can prevents the blocking phenomenon.

Fig. 21 shows the average throughput of MPTCP with OR&P and MPTCP with OR&P and CP scheduling over symmetric paths (8 Mbps-10 ms path) versus buffer size. MPTCP with OR&P and MPTCP with OR&P and CP scheduling suitably utilize multiple paths and can achieve the full capacity of both paths. For the same reason as in the previous results, MPTCP with OR&P and radical CP scheduling can reduce opportunistic retransmission as shown in Fig. 22, although it marginally underutilizes both paths compared to MPTCP with OR&P in some cases. MPTCP with OR&P and conservative CP scheduling also eliminates opportunistic retransmission with negligible impact over symmetric paths; it performs OR&P once when the buffer size is 200 KB.

To summarize the results in this section, CP scheduling and OR&P can complement each other. OR&P can improve the utilization of both paths, and CP scheduling can complement the robustness when the buffer size is too small. Although MPTCP with OR&P and radical CP scheduling can marginally underutilize both paths in some cases, it can reduce opportunistic retransmission, which can reduce the network resource usage. In conservative CP scheduling, OR&P is rarely performed because it almost eliminates the buffer blocking phenomenon.

5. Performance measurement in real networks

5.1. Throughput performance according to wireless networks

In this section, we evaluate the performance of MPTCP with CP scheduling in real networks. Reno congestion control is applied, and the buffer size is set to 200 KB in both the original MPTCP and MPTCP with CP scheduling. The OR&P method is disabled. The measurement environment is shown in Fig. 23. The WiFi network belongs to the Yonsei University network, and the LTE modem belongs to LG U+ Telecom (one of the main network operators in South Korea). WiFi represents the WiFi network without a throughput limitation and WiFi_limit denotes the WiFi network with a bandwidth limitation (10 Mbps). LTE signifies the normal LTE network, and LTE_limit is a kind of 3G service using an LTE network, which offers an analogous service to 3G by using a limited bandwidth in LTE. All wireless access points are located within five meters of the client, and the server is connected to the Yonsei University wired network. During this experiment, data is transferred over multiple paths, and the packet transmission runs for 100 s to allow the flows to reach equilibrium. The measurements are repeated 10 times.

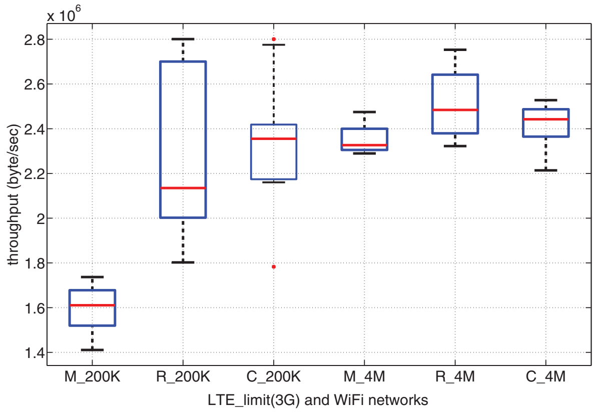


Fig. 24. Throughput in WiFi and LTE_limit (3G) networks.

Table 3
Wireless network performance.

Wireless link	Average RTT (ms)	Average throughput (Mbyte/s)
WiFi	16	2.25
WiFi_limit	107.6	1.12
LTE	131.4	1.57
LTE_limit	91.8	0.12

Fig. 24 shows the throughput of the original MPTCP and MPTCP with CP scheduling in WiFi and LTE_limit (3G) networks, and the results are represented as box-and-whisker plots. In these results, M indicates the original MPTCP. R and C represent the MPTCP with radical CP scheduling and MPTCP with conservative CP scheduling, respectively. The buffer size in MPTCP is 200 K and 4 M. The results show that the original MPTCP has a poorer throughput performance than others when the buffer size is set to 200 KB. The first reason for this is that the original MPTCP has throughput degradation due to the delay difference between the LTE_limit path and the WiFi path, as shown in Table 3. The other reason is that the queueing delay can be accelerated due to the limited bandwidth in LTE_limit. In the limited buffer case, MPTCP with CP scheduling has a better throughput performance than the original MPTCP. This is because CP scheduling conservatively utilizes the bandwidth of LTE_limit in order to eliminate the throughput degradation due to the delay difference between the paths, which maintains a similar throughput as the single WiFi. In this case, conservative CP scheduling can better buffer the throughput performance than radical CP scheduling because it can better eliminate the throughput degradation than radical CP scheduling. When the buffer is set to 4 MB, all cases have a similar throughput performance, which is almost the same or better throughput than the average throughput in the single TCP over a WiFi network.

Fig. 25 shows the throughput of the original MPTCP and MPTCP with CP scheduling in WiFi and LTE networks. The representation of the results is the same as in Fig. 24. As shown in Fig. 25a and b, the original MPTCP and MPTCP with radical CP scheduling almost equally utilize both the LTE and WiFi networks while MPTCP with conservative CP scheduling mainly utilizes the WiFi networks. Nevertheless, all cases have a similar throughput performance regardless of buffer size. In the limited buffer case (200 KB), the throughput degradation in WiFi (subflow 1) is almost the same as the throughput gain from using LTE. When the buffer size is set to 4 MB, no case can fully utilize both the WiFi and LTE networks. It appears that the bandwidth in wireless networks might not create the bottleneck, which means that the gain in bandwidth aggregation is not a main factor of throughput in a connection.

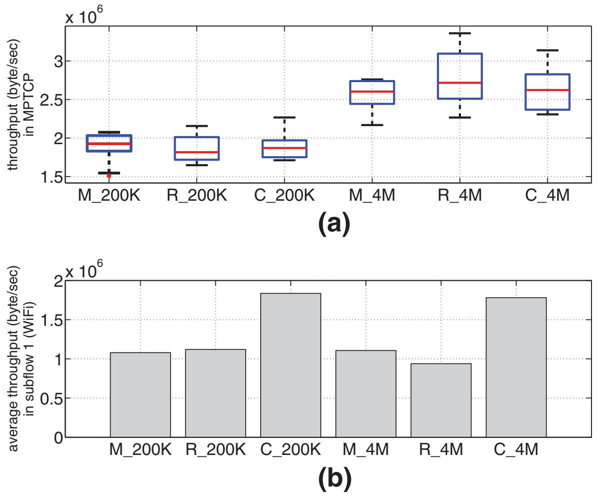


Fig. 25. Throughput in WiFi and LTE networks.

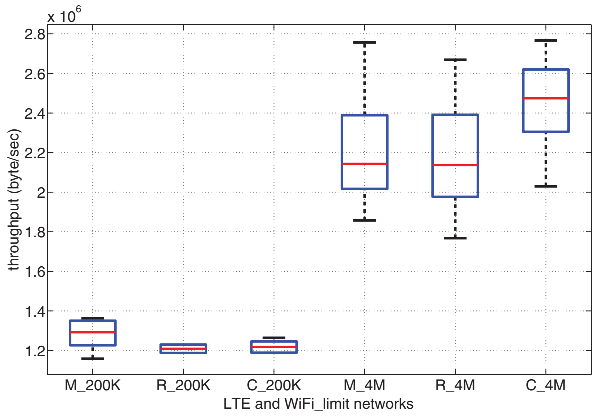


Fig. 26. Throughput in WiFi_limit and LTE networks.

Fig. 26 shows the throughput of the original MPTCP and MPTCP with CP scheduling in WiFi_limit and LTE networks. The representation of the results is the same as in Fig. 24. In this scenario, wireless networks seem to be the bottleneck. When the buffer size is set to 200 KB, all cases mainly utilize the WiFi_limit network due to the limited buffer size, and the throughput performance in all cases is almost the same as the single TCP in the WiFi_limit network. When the buffer size is set to 4 MB, all cases can achieve bandwidth aggregation, which means they fully utilize both the WiFi_limit and LTE networks.

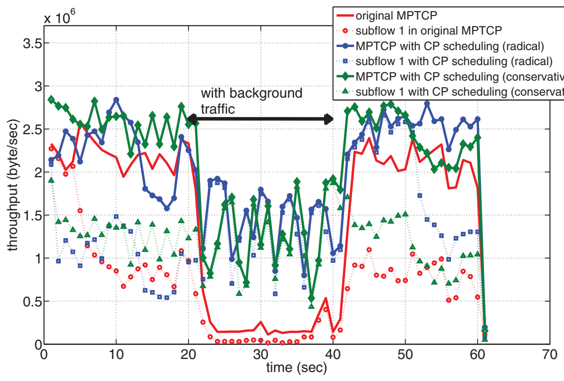


Fig. 27. Throughput according to the network dynamics.

5.2. *Performance effect of network dynamics*

Fig. 27 shows the effect of network congestion according to background traffic. In this scenario, the WiFi (subflow 1) and WiFi_limit (subflow 2) networks are employed, and the buffer size is set to 200 KB. During the interval between 20 s and 40 s, UDP traffic (9 Mbps) is applied to the WiFi_limit network (subflow 2) as background traffic, which causes the bandwidth of WiFi_limit is constricted. The representation of the results is the same as in Fig. 5. When the background traffic is applied in WiFi_limit (subflow 2), the original MPTCP experiences throughput degradation in subflow 2 and subflow 1. This is because the original MPTCP invariably tries to fully utilize all paths, although WiFi_limit (subflow 2) is unsuitable for transmission with WiFi (subflow 1), which is vulnerable to network dynamics. In contrast, MPTCP with CP scheduling adequately utilizes both paths before 20 s and can maintain throughput performance in the WiFi (subflow 1) after 20 s, although WiFi_limit (subflow 2) experiences the performance degradation. This is because the CP scheduling estimates the path performance and assigns the packets according to the estimated performance, which enables MPTCP to adjust the network dynamics. After background traffic disappears, MPTCP with CP scheduling can improve the throughput by utilizing the bandwidth of WiFi_limit (subflow 2).

6. Conclusion

In this paper, we present a new scheduling scheme, CP scheduling, for MPTCP in order to solve the throughput degradation problem that arises due to the performance difference of paths in a limited buffer. CP scheduling is a proactive solution that schedules data packets to the paths according to the estimation of out-of-order packets based on buffer size and the performance difference between paths. When a relatively large performance difference exists between paths, the MPTCP with CP scheduling achieves approximately the same or better throughput compared to a single TCP over the fastest path, although the standard MPTCP can suffer from throughput degradation. When the performance is the same between paths, MPTCP with CP scheduling sufficiently utilizes the capacity of the paths. Thus, MPTCP with CP scheduling transmits packets efficiently regardless of the buffer size and performance difference among multiple paths. Furthermore, CP scheduling maintains a stable packet interval with delay constraints. CP scheduling can complement and cooperate with a non-scheduling solution such as the OR&P and can be robustly operated despite network dynamics in a slow path. Finally, the simplicity of CP scheduling allows for practical implementation.

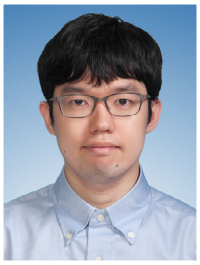
Acknowledgments

This work was supported by ICT R&D program of MSIP/IITP. [1491104001, Development of PS-LTE System and Terminal for National Public Safety Service].

References

[1] A. Ford, C. Raiciu, M. Handley, O. Bonaventure, January 2013, TCP Extensions for Multipath Operation with Multiple Addresses, RFC 6824.
[2] A. Ford, C. Raiciu, M. Handley, S. Barre, J. Iyengar, March 2011, Architectural guidelines for multipath TCP development, RFC 6182.
[3] C. Paasch, S. Ferlin, O. Alay, O. Bonaventure, Experimental evaluation of multipath TCP schedulers, in: Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop, 2014. ACM
[4] Y.C. Chen, Y.S. Lim, R.J. Gibbens, E.M. Nahum, R. Khalili, D. Towsley, A measurement-based study of multipath TCP performance over wireless networks, in: Proceedings of the 2013 Conference on Internet Measurement Conference, 2013. ACM
[5] S. Deng, R. Netravali, A. Sivaraman, H. Balakrishnan, WiFi, LTE, or Both?: measuring multi-homed wireless internet performance, in: Proceedings of the 2014 Conference on Internet Measurement Conference, 2014 ACM.
[6] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, M. Handley, How hard can it be? Designing and implementing a deployable multipath TCP, in: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, 2012. USENIX Association
[7] C. Paasch, R. Khalili, O. Bonaventure, On the benefits of applying experimental design to improve multipath TCP, in: Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, 2013 ACM.
[8] S. Barre, C. Paasch, O. Bonaventure, Multipath TCP: from theory to practice, NETWORKING 2011, Springer, Berlin Heidelberg, 2011, pp. 444–457.
[9] M. Li, A. Lukyanenko, S. Tarkoma, Y. Cu, A. Yla-Paaski, Tolerating path heterogeneity in multipath TCP with bounded receive buffers, Comput. Netw. 64 (2014) 1–4.
[10] D. Zhou, W. Song, M. Shi, Goodput improvement for multipath TCP by congestion window adaptation in multi-radio devices, in: Consumer Communications and Networking Conference (CCNC), 2013 IEEE, 2013.
[11] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, R. Boreli, DAPS: Intelligent delay-aware packet scheduling for multipath transport, in: Communications (ICC), 2014 IEEE International Conference on IEEE, 2014.
[12] S. Ferlin, T. Dreibholz, O. Alay, Multi-path transport over heterogeneous wireless networks: does it really pay off? in: Global Communications Conference (GLOBECOM), 2014 IEEE, 2014.
[13] B. Arzani, A. Gurney, S. Cheng, R. Guerin, B.T. Loo, Impact of path characteristics and scheduling policies on MPTCP performance, in: Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on. IEEE, 2014.
[14] C. Raiciu, M. Handly, D. Wischik, 2011, Coupled congestion control for multipath transport protocols, RFC 6356 (Experimental).
[15] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, J.Y.Le Boudec, MPTCP is not pareto-optimal: performance issues and a possible solution, IEEE/ACM Trans Netw 21 (5) (2013). 1651, 1665
[16] Y. Cao, M. Xu, X. Fu, Delay-based congestion control for multipath TCP, in: Network Protocols (ICNP), 2012 20th IEEE International Conference on. IEEE, 2012.
[17] T. Le, H. Rim, C. Hong, L. Sungwon, A multipath cubic TCP congestion control with multipath fast recovery over high bandwidth-delay product networks, IEICE Trans. Commun. 95 (7) (2012) 2232–2244.
[18] R. Stewart, Sep. 2007, Stream control transmission protocol, IETF RFC.

[19] J. Iyengar, P. Amer, R. Stewart, Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths, IEEE/ACM Trans. Netw. 14 (5) (2006). 951,964
[20] J. Liao, J. Wang, X. Zhu, cmpSCTP: an extension of SCTP to support concurrent multi-path transfer, in: Proceedings of the IEEE International Conference on Communications (ICC), May 2008, pp. 5762–5766.
[21] Y. Qiao, E. Fallon, J. Murphy, L. Murphy, Z. Shi, A. Hanley, Transmission scheduling for multi-homed transport protocols with network failure tolerance, Telecommun. Syst. 43 (1–2) (2010) 39–48.
[22] M. Fiore, C. Casetti, G. Galante, Concurrent multipath communication for real-time traffic, Comput. Commun. 30 (17) (2007) 3307–3320.
[23] T. Dreibholz, M. Becke, E.P. Rathgeb, M. Tuxen, On the use of concurrent multipath transfer over asymmetric paths, in: Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM 2010), 1, Dec. 2010, pp. 6–10. no. 6
[24] S. Mascolo, L.A. Grieco, R. Ferorelli, P. Camarda, G. Piscitelli, Performance evaluation of Westwood+ TCP congestion control, Performance Evaluat. 55 (4) (2004) 93111.
[25] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, October 1996, TCP Selective Acknowledgment Options, RFC 2018.
[26] C. Paasch, S. Barre, et al., Multipath TCP in the Linux Kernel, available from http://www.multipath-tcp.org.
[27] NS-3 Project, Direct code execution, http://www.nsnam.org/overview/projects/direct-code-execution/.
[28] Surrogate Modeling Lab at Ghent University, Sequential Experimental Design Toolbox, available from http://sumo.intec.ugent.be/SED.
[29] S. Ferlin-Oliveira, T. Dreibholz, O. Alay, Tackling the challenge of bufferbloat in Multi-Path Transport over heterogeneous wireless networks, in: Quality of Service (IWQoS), 2014 IEEE 22nd International Symposium of. IEEE, 2014.



Bong-Hwan Oh. Bong-Hwan Oh received B.S. and M.S. in Electrical and Electronic Engineering from Yonsei University in 2009, and 2011, respectively. He is currently a Ph.D. candidate in Electrical and Electronic Engineering from Yonsei University, Korea. His research interests include network architecture and network protocols.

Jaiyong Lee. Jaiyong Lee has been a professor at electrical and computer engineering department since 1987, 7 years at POSTECH and 20 years at Yonsei University, Seoul, Korea. As a director of the Advanced RFID/USN Technology center from 2004 to 2012, which was sponsored by the government, he led the technology development in related areas by producing many patents, journal papers and promoted academy-industry collaboration by transferring technologies. As a member of advisor of KIOT(Korea IoT Association) since 2005 and as the president of KICS (Korea Institute of Communication and Information Sciences) in 2012, he led government policies and activated the industries in ICT and especially in IoT areas, and led many international workshops such as WF(world Forum)-IoT 2014 as a co-chair. As a president of Giga Korea Foundation since 2013, he is leading the technology development for next generation ICT technology including 5G and IoT for future services in Korea. He was the dean of college of engineering and the president of industry-academy collaboration foundation of Yonsei University. He holds the Ph.D. in computer engineering from Iowa State University , USA.