An Efficient MPTCP-Based Congestion Control Scheme for HBDP Networks

Kwangsue CHUNG, Junyeol OH
Department of Electronics and Communications Engineering,
Kwangwoon University, Seoul, South Korea
kchung@kw.ac.kr

***Abstract*—With the widespread distribution of devices with multiple network interfaces, interest in multi-path transmission techniques has increased. The Internet Engineering Task Force (IETF) published Multi-path TCP (MPTCP) as a standard for multi-path transmission techniques and many researchers have studied multipath means of transmitting data efficiently, with each path having different characteristics. However, today's networks have been shown to exhibit high bandwidth-delay product (HBDP) characteristics but MPTCP does not match the requirements of HBDP networks. Many researchers have proposed solutions to overcome this problem, but the solutions have had the drawbacks of ineffective load balancing mechanisms and a trade-off problem between improving throughput and preventing loss events. In this paper, we propose an efficient MPTCP-based congestion control scheme in HBDP networks. Our scheme consists of two main mechanisms. One is to mitigate trade-off problems observed in previous works and the other is to enhance traffic migration according to the conditions of each path. Simulation results have shown that our scheme achieves those goals and enhance performance in HBDP networks.**

***Index Terms*—data transfer, packet loss, quality of service, TCPIP, transport protocols.**

I. Introduction

Along with the increased interest in the multi-path transmission techniques due to the widespread distribution of devices with multiple network interfaces, path diversity has increased because Internet environments have become more complex. So, researches have studied multipath transmission techniques in order to transmit data efficiently in each path having different characteristics [1-3].

The Internet Engineering Task Force (IETF) published Multi-path TCP (MPTCP) as a standard for multi-path transmission techniques. MPTCP can increase the throughput by transmitting data simultaneously through different paths, with multiple TCP subflows [4]. However, MPTCP is not friendly with single-path TCP or have mechanisms for load balancing.

To solve such problems and to take full advantage of multipath transmission techniques, many kinds of congestion control mechanisms have been studied. The previous schemes can be categorized as being either MPTCP improvement schemes or MPTCP variant schemes. MPTCP improvement schemes have been studied to guarantee the friendliness with single-path TCP and provide loading balancing by controlling increment of congestion windows [5-10]. In these regards those schemes have been shown to improve performance compared to that of bare MPTCP. However, they work based on the AIMD mechanism of legacy TCP. Therefore they spend too much time occupying available bandwidth in high bandwidth-delay product (HBDP) networks. With MPTCP variant schemes, designs for high-speed TCP were extended to the multi-path environment, to improve bandwidth utilization in HBDP networks [11-13]. They can guarantee not only load balancing but also friendliness with single-path TCP. However, they work based on high-speed TCP mechanisms that increase congestion window size aggressively, which results in frequent congestion in HBDP networks.

These new congestion control schemes show improved performance with regard to friendliness with single-path TCP and load balancing. However the trade-off problem between improving throughput and preventing loss events remained unsolved. Also, it is difficult to provide efficient load balancing in accordance with path characteristics because of the dependence on particular parameters.

In this paper, we propose an efficient MPTCP-based congestion control scheme for HBDP networks. Our scheme consists of congestion control and load balancing mechanisms which are respectively based on router buffer states and expected throughput. The purpose of the former mechanism is to adapt the congestion window by mitigating the trade-off problem of previous works and the purpose of the latter mechanism is effective traffic migration through consideration of the characteristics of each path.

The rest of this paper is structured as follows. Section 2 is a brief discussion of some related works and summarizes the problems and requirements of HBDP networks. Then, we describe details of our scheme in Section 3. In Section 4, we quantify how well our scheme satisfies RTT fairness by using a mathematical model. Section 5 presents the results of a performance evaluation. Finally, Section 6 gives some conclusions from our current work.

II. Related Work

In this section, we introduce MPTCP and two representative implementations of it, and point out the problems that are encountered in HBDP networks.

A. Multi-path TCP (MPTCP)

MPTCP is a modified version of TCP that allows the concurrent use of multiple available paths for data transmission, in a manner that is transparent to the application. Multi-path transport has been shown to be beneficial for bandwidth aggregation and to increase end-host robustness by creating subflows across the potentially disjoint paths. However, there is no congestion control

Digital Object Identifier 10.4316/AECE.2018.02006

mechanism in the MPTCP level and all subflows perform the congestion control mechanism independently; there are no considerations about load balancing and friendliness with single-path flows. Therefore it is hard to effectively utilize the innate benefits of multi-path flows.

To utilize benefits of MPTCP, the IETF defined the following three requirements to capture the desirable properties of a multi-path congestion control mechanism [6], [14]: first, improving throughput; a multi-path TCP user should perform at least as well as a TCP user that uses the best available path. Second, friendliness with single-path flow; a multi-path TCP user should never take up more capacity from any of its paths than a regular TCP user. Third, load balancing; a multi-path TCP algorithm should transfer traffic away from congested paths to less congested ones whenever possible. To satisfy the three requirements, many congestion control mechanisms have been proposed. These schemes can be categorized as being either MPTCP improvement schemes or MPTCP variant schemes.

B. Linked Increase Algorithm (LIA)

Several congestion control algorithms have been proposed for MPTP. Four of them are included in the Linux kernel implementation: LIA [7], OLIA [15], BALIA [16] and wVegas [17]. LIA is the default congestion control scheme. LIA is one of the major MPTCP improvement schemes that originated from TCP Reno's additive increase/multiplicative decrease (AIMD) mechanism. The goal of LIA is to satisfy the three MPTCP requirements. LIA regulates traffic by adjusting the sending win of each subflow. Assume that a connection consists of set of subflows R where each subflow may take a different route through the Internet. Each subflow $r \in R$ maintains its own congestion window w_r and $\hat{w}_r$ represents the value at equilibrium. During congestion avoidance MPTCP will:

Upon reception of each ACK on subflow r, increase the window on that subflow by $\min(\alpha/w_{total}, 1)$, where w_{total} denotes the total congestion window across all subflows.

Upon each loss on subflow r, reduce the window on that subflow by $1/w_r$.

Here

$$\alpha = \hat{w}_{total} \frac{max\left\{\frac{\hat{w}_r}{RTT_r^2} \mid r \in R\right\}}{\left(\sum_r \frac{\hat{w}_r}{RTT_r}\right)^2} \tag{1}$$

The parameter α controls the aggressiveness for satisfying the three MPTCP requirements. We describe how to satisfy these requirements by using α. To meet the two requirements, throughput and friendliness with single-path flow, the LIA incorporates the following relationship:

$$\sum_r \frac{\hat{w}_r}{RTT_r} = max\left\{\frac{\hat{w}_r^{TCP}}{RTT_r} \mid r \in R\right\} \tag{2}$$

$\hat{w}_r^{TCP}$ denotes congestion window size of regular TCP at equilibrium. To get the parameter α which satisfies Eq. 2, the LIA uses balance equations. At equilibrium, the balance is expressed by:

$$(1 - p_r)\frac{\alpha}{\hat{w}_{total}} = p_r \frac{\hat{w}_r}{2} \tag{3}$$

p_r denotes a probability of a dropped packet. Making the approximation that $1 - p_r \approx 1$, and writing $\hat{w}_r^{TCP}$ equal to square root of $2/p_r$,

$$\sqrt{\frac{\hat{w}_{total}\hat{w}_r}{\alpha}} = \hat{w}_r^{TCP} \tag{4}$$

By substituting Eq. 4 in Eq. 2, LIA derives parameter α as shown in Eq. 1. Hence, LIA can guarantee the throughput and single-path friendliness requirements.

Regarding the last of the requirements, load balancing, LIA assumes that a congested path may suffer from packet loss events more frequently than an uncongested path. Also, RTT is one of the components determining increments of the congestion window for each path. Therefore, the LIA can transfer traffic by taking RTT and loss events into account.

C. Multi-path Cubic (MPCUBIC)

MPCUBIC is one of the major MPTCP variant schemes which extended HBDP TCP for application to multiple paths, to improve bandwidth utilization in HBDP networks [12]. To improve throughput, MPCUBIC adapted the regular Cubic TCP mechanism [18] which uses concave-convex growth functions. During congestion avoidance, MPCUBIC regulates traffic by adjusting the sending window of each subflow. MPTCP will:

Upon reception of each ACK on subflow r, increase the window on that subflow by $\min(\delta C(t_r - K_r)^3 + W_r^{max}, C(t_r - K_r)^3 + W_r^{max})$, where C denotes a constant in regular Cubic, δ is a linking parameter between paths, t_r denotes the time elapsed from the last packet loss event on path r, and K_r denotes the period of time between two consecutive packet loss events.

Upon each loss on subflow r, reduce the window on that subflow by the factor β as used in regular Cubic.

To ensure friendliness with regular Cubic, MPCUBIC's window increment per path does not exceed that of regular Cubic by using the right argument in min(.). If MPCUBIC's window increment does not exceed that of regular Cubic, subflows of MPCUBIC perform the congestion control mechanism of Cubic. On the other hand, MPCUBIC has to set δ to use min(.).

Like the LIA, MPCUBIC also incorporates conditions for satisfying the requirements of throughput and single-path friendliness. In this case, MPCUBIC matches its friendliness to that of regular Cubic TCP:

$$\sum_r \frac{\hat{w}_r}{RTT_r} = max\left\{\frac{\hat{w}_r^{Cubic}}{RTT_r} \mid r \in R\right\} \tag{5}$$

To derive a δ which can satisfy Eq. 5, we summarize the relationship between $\hat{w}_r$ and $\hat{w}_r^{Cubic}$ in [12]:

$$max\left(\hat{w}_r, \frac{\hat{w}_r}{\delta^{1/4}}\right) = \hat{w}_r^{Cubic} \tag{6}$$

By substituting $\hat{w}_r^{Cubic}$ in Eq. 5, MPCUBIC derives the parameter δ:

$$\delta = \frac{max\left\{\left(\frac{\hat{w}_r}{RTT_r}\right)^4 \mid r \in R\right\}}{\left(\sum_r \frac{\hat{w}_r}{RTT_r}\right)^4} \tag{7}$$

To meet the third IETF requirement, for load balancing, MPCUBIC assumes that not only the congested path suffers from packet loss events more frequently but also that uncongested paths have larger window sizes. Therefore, δ must be based on both the congestion window size and

throughput (i.e. $\hat{w}_r/RTT_r$). Therefore MPCUBIC modifies Eq. 7 to Eq. 8, and then replaces δ with δ_r.

$$\delta_r = \hat{w}_r^k \frac{max\left\{\frac{\hat{w}_r^{4-k}}{RTT_r} \mid r \in R\right\}}{\left(\sum_r \frac{\hat{w}_r}{RTT_r}\right)^4} \quad (8)$$

MPCUBIC considers congestion window size and throughput for transferring traffic with consideration of the congestion levels of paths. Therefore, MPCUBIC can enhance bandwidth utilization and satisfy the three MPTCP requirements.

D. Problems of previous works and requirements for HBDP networks

- *Adaptiveness*

LIA and almost all MPTCP improvement schemes have inherited the properties of TCP Reno. They show low packet loss rates compared to HBDP TCP schemes because of the AIMD mechanism of TCP Reno. However, they spend too much time occupying the available bandwidth [19]. On the other hands, MPCUBIC and most MPTCP variant schemes have inherited the properties of high-speed TCPs. They do not spend much time occupying the available bandwidth. However, they cause lots of packets to be dropped because of the aggressiveness of high-speed TCP. MPTCP improvement schemes and MPTCP variant schemes have the trade-off problem between improving throughput and preventing loss events.

HBDP networks have the properties of high bandwidth and large buffer sizes [20]. A proper algorithm should guarantee aggressive increase of the congestion window to quickly occupy available bandwidth. Plus, the buffer overflow should be prevented especially in HBDP networks because these networks produce many dropped packets due to large buffer size when the overflow occurs. To guarantee both properties, a congestion control scheme should intelligently adapt the congestion level to mitigate the trade-off problem that was encountered in previous works.

- *Load balancing*

For effective load balancing, the congestion condition of each path has to be estimated accurately and the traffic should be transferred to uncongested paths as soon as possible. For determining the congestion condition, the LIA uses RTT only whereas MPCUBIC uses RTT and the congestion window of the subflow. These parameters can lead to wrong decisions because their use does not apply path conditions accurately. To prevent wrong decisions for dealing with congestion conditions, the sender has to consider the property of the path only, and exclude consideration of the properties of the subflow.

For transferring traffic, RTT fairness is important, to transfer traffic quickly from one path to the other path. Window-based congestion control schemes update their window size after receiving an acknowledgement packet. This update period depends on the RTT of the path. RTT unfairness may cause poor traffic transfer. For example, there are two paths A and B. Path A has a large RTT and is an uncongested path whereas path B has a short RTT but is a congested path. Sender wants to transfer traffic from path B to path A. But sender has an RTT unfairness and update period of path A is too long compared to that of path B. The difference of the congestion window size between both paths increases slowly and the speed of traffic transfer is also slow. Therefore, to transfer traffic quickly after making a decision, sender has to guarantee RTT fairness.

III. Efficient MPTCP-based Congestion Control Scheme

After analyzing the problems of the schemes of the previous works, we studied the design of an efficient MPTCP-based congestion control scheme that fulfills the three IETF requirements for MPTCP that are listed in Section 2. We propose two key ideas. First, the sending rate should be increased adaptively according to the congestion level of each path to prevent packet loss and enhance throughput. For example, if the link is uncongested, increasing the sending rate should be aggressive to obtain available bandwidth quickly. However, once the link is fully utilized, the sending rate should be increased conservatively to prevent overshooting. To control aggressiveness according to the congestion level, we apply a congestion control mechanism based on the router buffer states. This mechanism can adjust its aggressiveness based on the capacity of the bottleneck buffer and the existence of cross traffic by observing packet delay. Second, load balancing should be performed quickly and accurately among paths that can be used by applying path conditions, not subflow conditions. To perform load balancing quickly and accurately, we apply an expected throughput based load balancing mechanism. This mechanism can transfer traffic based on relationships among paths. Eq. 9 shows our proposed subflow congestion window growth function on paths.

$$\Delta w_r(n+1) = min\left(\frac{\rho_r(n)\alpha_r(n)}{w_r(n)}, \frac{\delta_r(n)\rho_r(n)\alpha_r(n)}{w_r(n)}\right) \quad (9)$$

Δw_r denotes the variation of congestion window size on path r, ρ_r is a load balancing parameter on path r with respect to the others, α_r is an increment of the congestion window size on path r and $\delta_r(n)$ is a friendliness parameter on path r with single-path Cubic TCP on path r. ρ_r is calculated from the expected throughput based load balancing mechanism and δ_r is calculated from friendliness using the Cubic TCP mechanism. Both ρ_r and δ_r can be derived in the MPTCP level. α_r is calculated from a router buffer-based congestion control mechanism and can be derived in the subflow level. After deriving the three parameters, Δw_r can be selected by using min(.). It allows an upper threshold to keep friendliness with single-path Cubic TCP and performs in congested states as a decision of MPTCP.

A. Router buffer based congestion control mechanism

As explained earlier, a congestion control mechanism based on router buffer states has to derive a proper value of α_r to control aggressiveness according to the congestion level. To derive α_r, the proposed scheme firstly checks router buffer occupancy. By analyzing router buffer occupancy and its variance, we predict the congestion level and existence of cross traffic.

- *Measuring router buffer occupancy*

For checking congestion level of each path, we use *diff* which denotes the number of packets backlogged in the bottleneck queue and $diff_{max}$ which denotes the bottleneck queue size. *diff* is defined in TCP Vegas [21]. Fig. 1 shows how to estimate *diff* and $diff_{max}$.

On each ACK:
Expected=win/baseRTT
Actual=win/RTT
diff=(Expected-Actual)baseRTT
If packet loss **then**
$diff_{max}$=diff
Go to fast recovery phase
Else
Go to congestion avoidance phase

Figure 1. Pseudo-code for calculating $diff_{max}$

In Fig. 1, a state variable, called *baseRTT*, is maintained as an estimation of the transmission delay of a packet over the network path. *Expected* gives the estimation of the throughput that we get if we do not overrun the network path. *Actual* stands for the throughput that we really get. Then, (*Expected-Actual*) is the difference between the expected throughput and the actual throughput. When multiplied by *baseRTT*, the product stands for the amount of data backlogged in the bottleneck queue, *diff*. We assume that all packet loss events occur because of overflow in the bottleneck queue. Therefore $diff_{max}$ sets the value of *diff* calculated at the time that the packet loss event occurs.

- *Control of the increment of congestion window size as network load*

Previous congestion control mechanisms of the subflow cannot guarantee high bandwidth utilization and prevent overflow simultaneously because of the trade-off relationship between both. To overcome this trade-off, our scheme controls the increment of congestion window size by considering relationship between *diff* and $diff_{max}$. Fig. 2 illustrates the two-phase growth function of a subflow in our scheme to guarantee high bandwidth utilization and prevent overflow.

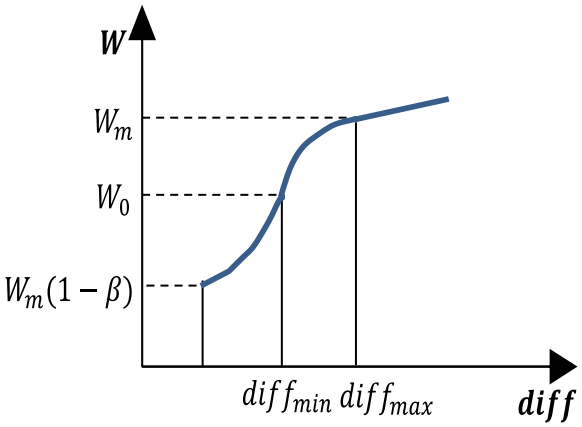


Figure 2. Growth function of the congestion window size

We denote W_m as the window size just before the last window reduction which occurred due to overshooting. After losing packets, window size decreases from W_m to $W_m(1-\beta)$. β is a constant multiplication decrease factor applied for window reduction at the time of the loss event. At that time, there are no backlogged packets in the bottleneck queue. Therefore, the window size increases multiplicatively until it reaches W_0, to occupy bandwidth quickly (i.e. $diff > diff_{min}$), where W_0 denotes the window size which equals the capacity of the bottleneck queue. After reaching W_0, the window size slows down its growth logarithmically as it gets closer to W_m (i.e. $diff \cong diff_{max}$), to prevent buffer overflow and keep bandwidth utilization. For example, when a delay-based scheme competes with a loss-based scheme, their bandwidth utilization falls down as the window size is reduced because of their early congestion detection policy. If queueing delay increases, the increment of window size is set to zero or becomes negative. But, our growth function keeps increasing the window size by considering the bottleneck queue limit. We designed the algorithm as an increment function, *f*(*diff*) as in Eq. 10:

$$f(diff) = \Delta w(n+1) = \begin{cases} \sqrt{\frac{w_r(n)}{4}}, if\ diff \le diff_{min} \\ \frac{c_1}{c_2 + diff}, else \end{cases}$$

$$c_1 = \frac{(diff_{max} - diff_{min})\alpha_{max}\alpha_{min}}{\alpha_{max} - \alpha_{min}} \quad (10)$$

$$c_2 = \frac{(diff_{max} - diff_{min})\alpha_{min}}{\alpha_{max} - \alpha_{min}} - diff_{min}$$

w_r is the current window size, α_{max} and α_{min} are thresholds of *f*(*diff*). If $diff \le diff_{min}$, the multiplicative increase phase will be performed. In order to occupy available bandwidth quickly after reducing window size, *f*(*diff*) is set to the square root of w_r. In the second condition, the logarithmic increase phase will be performed. In order to prevent overshoot and increase bandwidth utilization, *f*(*diff*) is decreased as *diff* gets closer to $diff_{max}$, as shown in Fig. 3 which shows the relationship between the congestion level and the increment of window size.

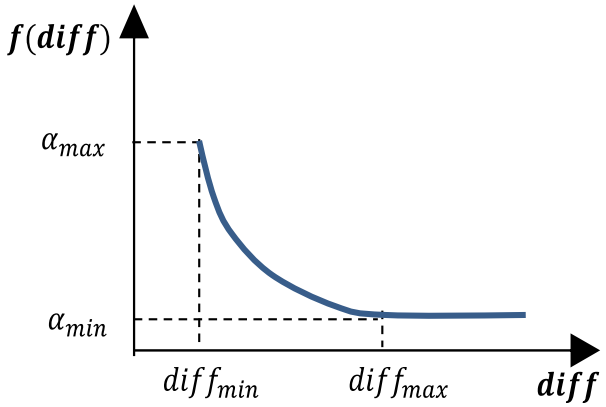


Figure 3. Graph for relationship between *diff* and α

- *Setting the threshold of f(diff)*

In real networks, both injecting and ejecting cross traffic are general cases. The congestion level of a path varies with the variation in the cross traffic. Therefore, an adaptive scheme which can control the increment of congestion window according to the variation in cross traffic is needed. Previous schemes controlled the congestion window by applying a fixed increment rate or range. With such schemes

it is difficult to perform adaptive congestion control with the variation in the competing flows.

To solve this problem, our scheme proposes a mechanism to set the threshold of *f*(*diff*) based on the expected fairness index, $\dot{F}$ [22]:

$$\dot{F}_i = \frac{\Delta diff_i}{\Delta w_i} \tag{11}$$

Consider a single link bottleneck traversed by n flows with the same RTT when the total throughput of competing flows $\sum_{i=0}^{n-1} w_i(t)$ is equal to bottleneck link capacity C where i denotes the flow number. If any flow increases its congestion window, the excess packets will be stacked in the bottleneck queue. Suppose that all flows increase the congestion window by the same Δw simultaneously. At the end of one RTT, the bottleneck queue will have $n \cdot \Delta w$ packets queued up. However, the actual number of backlogged packets belonging to each flow will not be the same unless all flows have the same congestion window size and congestion control policy. Specifically, the number of new packets of each flow contributed to the queue is $w_i(t) / \sum_{i=0}^{n-1} w_i(t)$. This observation can be used to estimate the fairness ratio. If a flow i increases the congestion window when the link is fully utilized, it expects to see an increase in the queuing delay. If the link is shared perfectly fairly, this increase should be the same as if the flow had been alone in a bottleneck link of capacity equal to $w_i(t)$. Therefore, by comparing the actual delay increase with the expected one, the flow can deduce the status of the link. If the observed delay increase is greater than expected, the flow is currently using more than its fair share. And conversely, a smaller than expected delay increase indicates a throughput below the fair share of the flow.

Fig. 4 shows the pseudo-code to set the threshold of *f*(*diff*) for fairness. We assume that the current bottleneck link is busy and that there are backlogged packets in the bottleneck queue. Therefore the pseudo-code of Fig. 4 only performs in the logarithmic increase phase, during which the value of *diff* is positive. If the value of $\dot{F}_i$ is bigger than $\dot{F}_{max}$ or smaller than $\dot{F}_{min}$, *f*(*diff*) of flow i decreases or increases its threshold values α_{max} and α_{min} by multiplying them by $\dot{F}_i$; otherwise, flow i maintains its increment threshold of *f*(*diff*) because flow i works fairly with the others. This operation works continuously. So, all flows competing for the same bottleneck link can be converged to a fair share range. Our scheme can ensure fairness with competing flows and efficiency for bandwidth occupancy by monitoring the effects of its aggressiveness.

If $\hat{F}_{min} \le \hat{F} \le \hat{F}_{max}$ **then**
  $\alpha_{max} = \alpha_{max}$
  $\alpha_{min} = \alpha_{min}$
Else
  $\alpha_{max} = \alpha_{max} \cdot \hat{F}$
  $\alpha_{min} = \alpha_{min} \cdot \hat{F}$

Figure 4. Pseudo-code to set the threshold of *f*(*diff*) for fairness

B. Expected throughput-based load balancing mechanism

As explained earlier in Section 2, the load balancing mechanism has to consider the properties of paths only and not those of the subflows. With previous approaches, the load balancing mechanism is included in the mechanism for friendliness with single-path TCP as an additional function. Therefore, its load balancing mechanism performance is limited by its use of subflow parameters such as w and RTT. So, both mechanisms have to be separated to properly achieve their purposes. To check the properties of paths only, we derive an expected throughput which is based on the AIMD throughput in equilibrium states. It is a function of the decrease/increase parameters α and β, RTT, and the packet drop rate p. AIMD throughput in equilibrium states is expressed as follows [23]:

$$Th = \frac{\sqrt{2-\beta}\sqrt{\alpha}}{\sqrt{2\beta}RTT\sqrt{p}} \tag{12}$$

Parameter α and β are properties of the subflow. So, we initialize α to 1, β to 2/5, for removing the effects of subflow. The equation can then be rewritten as the following:

$$Th = \frac{\sqrt{\frac{2}{p}}}{RTT} \tag{13}$$

After checking the expected throughput of each path by using Eq. 13, our proposed mechanism sets the load balancing factor ρ to transfer traffic efficiently. Fig. 5 shows relationship between *Th* and ρ.

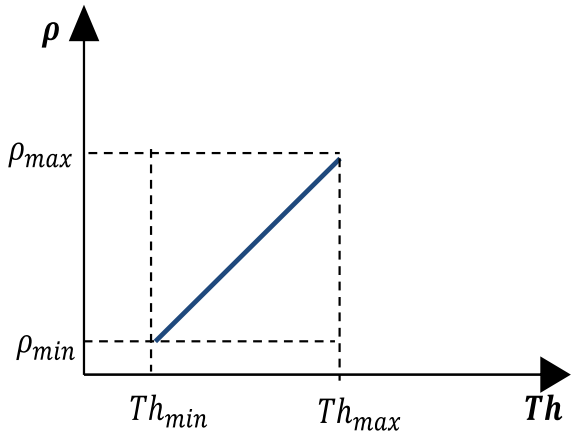


Figure 5. Graph for relationship between *Th* and ρ

After calculating the *Th* of each path, we can derive the average expected throughput Th_{avg}. Also, Th_{max} and Th_{min} are the maximum and minimum expected throughputs. The values of ρ_r, ρ_{max} and ρ_{min} can be expressed as follows:

$$\begin{aligned} \rho_r &= \frac{\rho_{max} - \rho_{min}}{Th_{max} - Th_{min}}\left(Th_r - Th_{min}\right) + \rho_{min} \\ \rho_{max} &= 1 - \frac{Th_{avg} - Th_{max}}{Th_{avg}} \\ \rho_{min} &= 1 - \frac{Th_{avg} - Th_{min}}{Th_{avg}} \end{aligned} \tag{14}$$

C. Fairness with regular Cubic TCP

Previous schemes suggest a mechanism that makes the total congestion window across all subflows equal to a targeted single-path flow to guarantee friendliness with

single-path flow. Their total window size cannot exceed the single-path flow's own. With such a mechanism MPTCP cannot increase its window size even though the occupancy of the bandwidth is low or the available bandwidth is not fully utilized. Then, it degrades the utilization of bandwidth. To overcome this limitation, we propose a mechanism for partial friendliness with single-path flows. It limits the increment of MPTCP's window size to single-path flow by using a threshold parameter δ in Eq. 9, only when the network is fully utilized. Our target is to match regular Cubic TCP which is commonly used on the web. We can derive δ as follows:

To achieve our goal, δ has to satisfy the IEFT's MPTCP requirements for throughput and single-path friendliness in Eq. 5. So, we express our scheme's throughput in equilibrium states using Eq. 9:

$$(1 - p_r)\frac{\delta_r \rho_r \alpha_r}{\hat{w}_r} = p_r \frac{\hat{w}_r}{\beta} \tag{15}$$

If we assume that $1\text{-}p_r \approx 1$, we can write $\hat{w}_r$ and p_r,

$$\begin{aligned} \hat{w}_r &= \sqrt{\frac{\delta_r \rho_r \alpha_r \beta}{p_r}} \\ p_r &= \frac{\delta_r \rho_r \alpha_r \beta}{\hat{w}_r^2} \end{aligned} \tag{16}$$

The average throughput of regular Cubic TCP is as follows [11]:

$$\hat{w}_r^{Cubic} = \left(\frac{C(4 - \beta) RTT_r^3}{4 \beta p_r^3} \right)^{\frac{1}{4}} \tag{17}$$

By simultaneously substituting the expression for p_r in Eq. 16 into Eq. 17, we can express the average throughput of Cubic TCP:

$$\hat{w}_r^{Cubic} = \left(\frac{C(4 - \beta) RTT_r^3}{4\beta \left(\frac{\delta_r \rho_r \alpha_r \beta}{\hat{w}_r^2} \right)^3} \right)^{\frac{1}{4}} \tag{18}$$

Finally we substitute Eq.18 into Eq. 5 and derive δ_r which satisfies friendliness with single-path Cubic TCP,

$$\delta_r = \frac{\hat{w}_r^2 max\left\{ \left(\frac{C(4-\beta) RTT_r^3}{4\beta} \right)^{\frac{1}{3}} \mid r \in R \right\}}{\rho_r \alpha_r \beta \left(\sum_r \frac{\hat{w}_r}{RTT_r} \right)^{\frac{4}{3}}} \tag{19}$$

δ_r is a parameter to make the total window size across all subflows equal to that of single-path Cubic TCP. It is only applied in the logarithmic increase phase. When the network is underutilized, our scheme increases the sending rate quickly. After entering the logarithmic increase phase, δ_r performs as a threshold value to help occupy the bandwidth of single-path flow. Before entering the logarithmic increase phase, our scheme does not harm single-path flow because of our congestion control mechanism.

IV. Analysis for RTT Fairness

In this section, we want to quantify how well the proposed scheme satisfies RTT fairness over multiple subflows with different RTTs. In the following analysis, we use a synchronized loss model. There is much evidence showing that synchronized loss is common in high-speed networks [23]. We verify our scheme based on a simple network topology as shown in Fig. 6. Our scheme has two window growth shapes, multiplicative increase and logarithmic increase. We prove RTT fairness of our scheme separately.

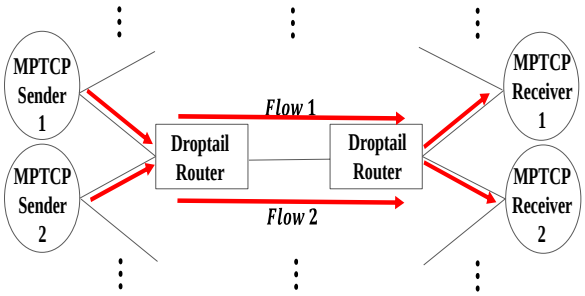


Figure 6. Network topology for analysis

A. Assumption

Before proving the RTT fairness, we demonstrate several assumptions and equations needed for verification as below.

- Network topology

In Fig. 6, each MPTCP sender has two subflows and a bottleneck link that is occupied by a subflow from different sender. With this topology, the RTT of a subflow consist of a propagation delay p_d and queueing delay q_d. If the total throughput of subflows that occupy a bottleneck link do not reach link capacity, q_d =0, then q_d will be increased when their total throughput exceeds link capacity. There are two flows: $flow_1$ is a flow which has short RTT, RTT_1, and $flow_2$ is a flow which has large RTT, RTT_2. We can summarize the RTTs of these flows:

$$\begin{aligned} RTT_1 &= p_d + q_d \\ RTT_2 &= \varepsilon \cdot p_d + q_d (\varepsilon > 1) \end{aligned} \tag{20}$$

- Parameter tuning

For simplification of this proof, we do not use parameters, ρ and δ in this section. We do not consider the case of competing with single-path flow and transferring traffic for load balancing. We are only interested in fair sharing between flows with different RTTs. By excluding unnecessary parameters, we can derive a suitable congestion window growth function. Its increase factor α will be determined by $f(diff)$:

$$w_r(n+1) = w_r(n) + \frac{\alpha_r(n)}{w_r(n)} \tag{21}$$

- Verification

We prove the RTT fairness over multiple subflows with different RTTs in the multiplicative increase phase and logarithmic increase phase separately. In this verification, we use Jain's fairness index [25], which is defined as:

$$F(x) = \frac{(\sum x_i)^2}{n(\sum x_i^2)} \tag{22}$$

Theorem 1: Two flows converge to their fair share under the network model of Fig. 6 with different RTTs in the multiplicative increase phase.

Proof. Let's consider two flows, whose sending rates are

x_1 and x_2. We assume that $x_1 > x_2$. In Eq. 22, we see $\Delta F \geq 0$, if and only if $\Delta x_1/x_1 \leq \Delta x_2/x_2$. Additionally we can express $x_i = w_i/RTT_i$ and $\Delta x_i = x_i(n+1) - x_i(n) = (w_i+\Delta w_i)/(RTT_i+\eta) - w_i/RTT_i$; η denotes the increment of queueing delay after updating the window size. So we summarize $\Delta x_i/x_i$ as below:

$$\frac{\Delta x_i}{x_i} = \frac{\frac{\Delta w_i RTT_i - \eta w_i}{RTT_i(RTT_i-\eta)}}{\frac{w_i}{RTT_i}} \tag{23}$$

The multiplicative increase phase works only when the network state is not busy, $diff \leq 0$. We know that $RTT >> \eta$ and $\eta \cong 0$, and also $q_d \cong 0$. In this phase, $\Delta x_i = f(diff) = \sqrt{w_i}/2$ using Eq. 10. Therefore, we rewrite Eq. 23: $\Delta x_1/x_1 = \sqrt{w_1}/2w_1$ and $\Delta x_2/x_2 = \sqrt{w_2}/2w_2$. The relationship between $\Delta x_1/x_1$ and $\Delta x_2/x_2$ can be shown by using a graph $f(t) = \sqrt{t}/2t$. If $t_i < t_{i+n}$, the value of $f(t_i)$ is greater than $f(t_{i+n})$. That means that $\Delta x_1/x_1 \leq \Delta x_2/x_2$ because x_1 is bigger than x_2; $flow_1$ has a short RTT and a larger w than $flow_2$. Therefore, $\Delta F \geq 0$ in the multiplicative increase phase.

Theorem 2: Two flows converge to their fair share under the network model of Fig. 6 with different RTTs in the logarithmic increase phase.

Proof. Like Theorem 1, we prove that when $x_1 > x_2$, the logarithmic increase phase can satisfy $\Delta F \geq 0$ or not. Note that $q_d > 0$ and $\eta \geq 0$; the value of η depends on $\sum_{i=0}^{n-1} \Delta w_i$ in a shared bottleneck. Fig. 1 shows that Δw_i is determined by $f(diff)$, and also $f(diff)$ is determined in inverse proportion to $diff$ in Fig. 3. Fig. 3 shows how to determine $diff$ and confirms the relationship between $f(diff)$ and parameters such as the window size and delay components. We summarize this relationship as below:

$$f(diff) \propto \frac{1}{diff} \quad and \quad diff \propto \frac{w_i}{\varepsilon_i p_d} \tag{24}$$

In Fig. 1, *Actual* can be written as $w_i/(\varepsilon_i \cdot p_d + q_d + \eta)$ and *Expected* can be written as $wi/\varepsilon i \cdot p_d$ because $\varepsilon_1 = 1$ and $\varepsilon_2 > 1$. Then, $diff = w_i(q_d+\eta)/(\varepsilon_i \cdot p_d + q_d + \eta)$. At this point, note that all subflows which occupy the same bottleneck link have the same q_d and η. Both parameters depend on the number of backlogged packets in the queue. So, we ignore effects of both to show RTT fairness easily. In our assumption, $RTT_1 < RTT_2$, and $w_1 > w_2$. Therefore, $flow_1$ encounters a congestion level higher than $flow_2$. That means that Δw_2 is higher than Δw_1. This situation may continue until $flow_1$ and $flow_2$ have similar conditions. That means that $\Delta x_1/x_1 \leq \Delta x_2/x_2$ because $flow_2$ estimates that its congestion level is lower than that of $flow_1$. Therefore $\Delta F \geq 0$ in the logarithmic phase increase.

By showing Theorem 1 and 2 and proving its RTT fairness, we confirm that the difference between flows which occurred owing to RTT will decrease with time.

V. Experiments

To evaluate the performance of the proposed scheme in comparison with LIA and MPCUBIC, we use NS2 for the performance evaluation. The simulation topologies are shown in Fig. 7. Our simulations are divided into three parts which include load balancing, adaptiveness according to the congestion level and friendliness with single-path flow.

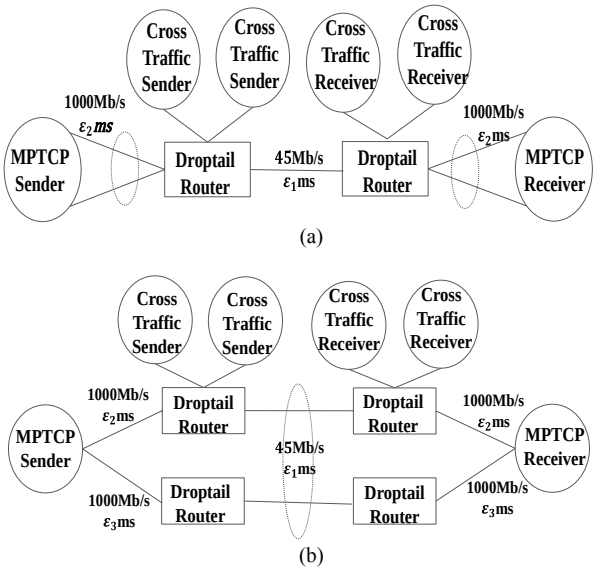


Figure 7. Simulation topologies

A. Adaptiveness

In order to know how the proposed scheme adapts as the congestion level varies, we consider the topology in Fig. 7(a). We set $\varepsilon_1 = 20$ and $\varepsilon_2 = 10$. Fig. 8 shows the congestion window fluctuation of congestion control schemes when competing with cross traffic. In this scenario, we use constant bit-rate (CBR) flow as cross traffic and number of CBR flows vary as shown in Table I.

TABLE I. The Number of Cross Traffic Flows According to Time

Time (s)	0-10	10-20	20-25	25-30	30-40	40-50
Number of Flows	0	2	0	1	2	0

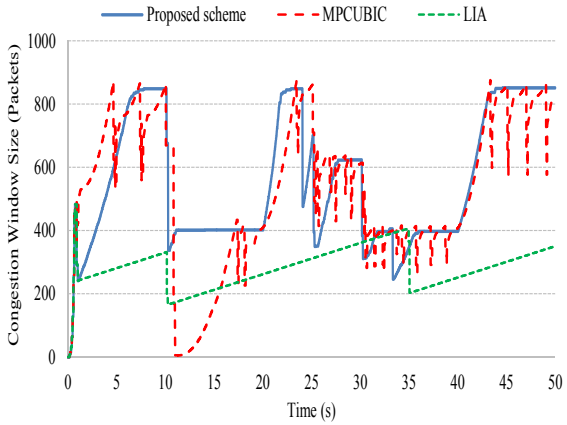


Figure 8. Congestion window with cross traffic

Fig. 8 shows that all schemes enter the congestion avoidance phase at 1 second after overshoot in the slow start phase. As we said in Section 2, the LIA spends a lot of time occupying capacity because of the AIMD property and it cannot control the increment rate based on network conditions whereas MPCUBIC does not spend much time occupying bandwidth available after packet loss events, but it overshoots which causes lots of packet drops because it is too aggressive. Even its congestion window size resets to 1

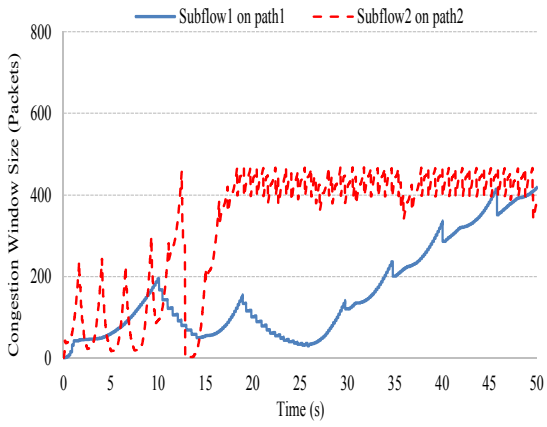
because of retransmission time-out (RTO) at 11 seconds. We confirm that both schemes show trade-off problems between preventing packet loss and throughput. With the proposed scheme, it performs based on bottleneck capacity and controls the increment rate according to the congestion level. When the network is congested, it increases the congestion window conservatively; with lower congestion levels, the congestion window is increased aggressively. Therefore, we know that it can reduce packet loss events and occupy available bandwidth rapidly by considering network conditions.

B. Load balancing

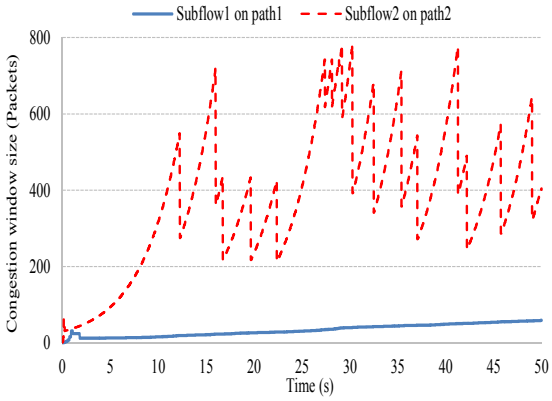
In order to evaluate the load balancing property of the proposed scheme based on path properties and network conditions, we consider the topology of Fig. 7(b). We use two scenarios to evaluate the efficiency of load balancing.

The first scenario is to evaluate the ability to determine path properties. We set $\varepsilon_1 = 10$, $\varepsilon_2 = 40$, $\varepsilon_3 = 1$, $p_1 = 0.001$ and $p_2 = 0.003$ where p_i denotes the random loss rate of path i. It is not related to congestion loss. According to the parameters of each path, we know that the expected throughputs for $path_1$ and $path_2$ are respectively 223 pkt/s and 1075 pkt/s. Ideally traffic should transfer from $path_1$ to $path_2$.

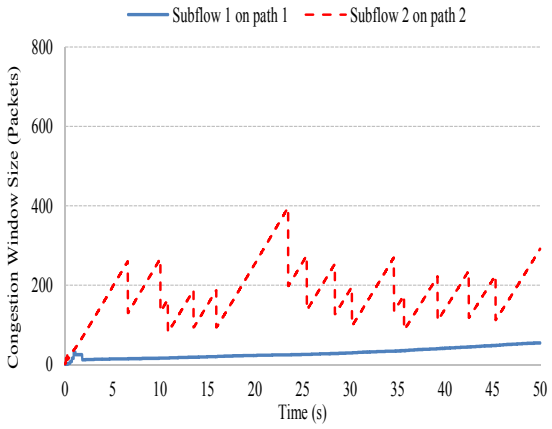
Fig. 9 shows the congestion window fluctuations of congestion control schemes in an environment over multiple paths with different properties. MPCUBIC transfers its traffic from $path_2$ to $path_1$ gradually because it determines the path condition according to each path's congestion window size: $path_1$ which has a low random loss rate seldom experiences loss events and the congestion window size of $path_1$ increases continuously. Our sche-me transfers traffic from $path_1$ to $path_2$ even though $path_2$ experiences random loss because it determines the path condition according to expected throughput and allocates traffic per path. Plus, the proposed scheme reduces packet loss events better than MPCUBIC because of its control of the increment of congestion level. The LIA seems to work properly like ours. But it uses a fixed increment and suffers from RTT unfairness. This means that the update period of window size depends on the RTT. Therefore, it is difficult to transfer from one path with a short RTT to another path with a long RTT.



(a)



(b)



(c)

Figure 9. Congestion window of the MPTCP scheme over multiple paths with different properties: (a) MPCUBIC (b) Proposed scheme (c) LIA

Table II shows the average throughput of each scheme over multiple paths with different properties. Table II shows that even though MPCUBIC transfers its traffic to $path_1$, the throughput of $path_1$ is too low and does not take the advantage of multi-path transmission. The proposed scheme can achieve higher total throughput than the others.

TABLE II. COMPARISON OF THROUGHPUT OVER MULTIPLE PATHS WITH DIFFERENT PROPERTIES

Protocol / Path	LIA	MPCUBIC	Proposed scheme
$path_1$	2.28 Mb/s	12.39 Mb/s	2.64 Mb/s
$path_2$	89.56 Mb/s	151.22 Mb/s	187.92 Mb/s
Total	91.85 Mb/s	163.61 Mb/s	190.57 Mb/s

The second scenario evaluates the ability to transfer traffic quickly. We set $\varepsilon_1 = 20$, $\varepsilon_2 = 10$; both paths have the same properties. To check how well the MPTCP schemes transfer traffic from congested paths to uncongested when the congestion level of each path is changed continuously, we control the congestion level of the path by inserting cross traffic into $path_1$ as shown in Table III. Ideally, if we insert cross traffic into $path_1$, traffic transfers to $path_2$. After finishing the insertion of cross traffic to $path_1$, the same amount of traffic is inserted into each path gradually.

TABLE III. THE NUMBER OF CROSS TRAFFIC FLOWS ACCORDING TO TIME

Time (s)	0-10	10-30	30-45
Number of flows	0	2	0

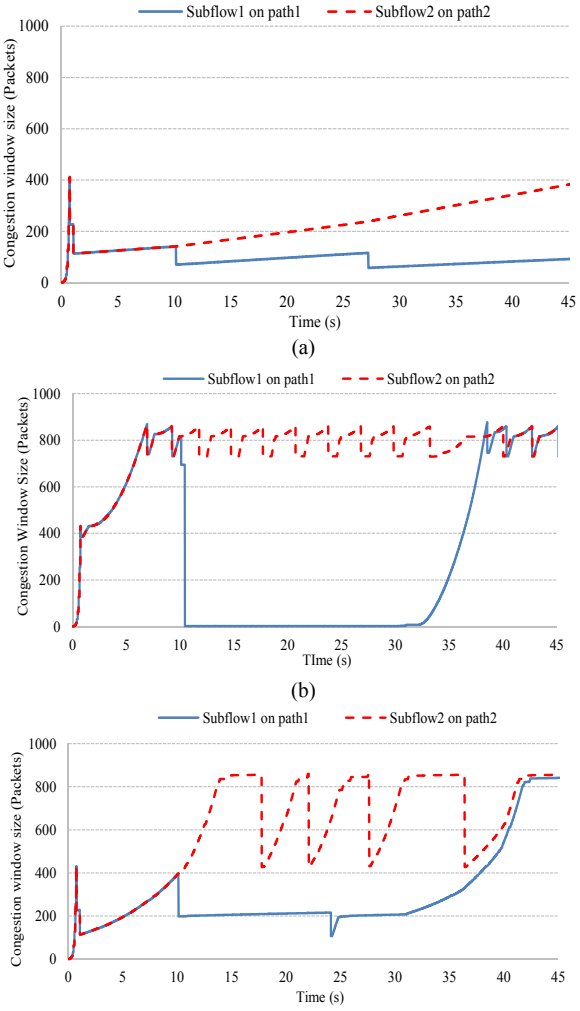


Figure 10. Congestion window of MPTCP schemes versus the congestion level of the path: (a) LIA (b) MPCUBIC (c) Proposed scheme

Fig. 10 shows the congestion window fluctuation of congestion control schemes versus the congestion level of the path. In Fig. 10, with the LIA, after 10 seconds it transfers traffic from $path_1$ to $path_2$. But, after finishing inserting cross traffic, the increase rate of $path_2$ is higher than that of $path_1$ even though the congestion level of $path_2$ is higher than that of $path_1$. After 30 seconds, both paths have same condition and much traffic is inserted to $path_2$. So, we confirm that the LIA could not transfer traffic adaptively as the congestion level varies. MPCUBIC shows a dramatic shape on the graph. When cross traffic is inserted to $path_1$ suddenly, RTO occurs and the window size of that path is initialized to 1. MPCUBIC determined the network's condition by using the current window size. Therefore, MPCUBIC cannot use $path_1$ until 30 seconds have passed. With the proposed scheme, a lot of traffic is transferred from $path_1$ to $path_2$ in 20 seconds. After 30 seconds, traffic transfers from $path_2$ to $path_1$ and after 43 seconds, both paths have the same traffic and work fairly.

C. *Friendliness with single-path flow*

In this section, we investigate proposed scheme's friendliness with regular Cubic TCP at a common bottleneck. Two subflows share the bottleneck with a regular Cubic TCP flow at a link as shown in Fig. 2(a). We set $\varepsilon_1=20$ and $\varepsilon_1=10$. Fig. 11 shows that the proposed scheme can share the bottleneck with regular Cubic TCP at the bottleneck. The proposed scheme captures bandwidth quickly until 15 seconds. But, when regular Cubic TCP increases its congestion window size greedily to capture more bandwidth, the proposed scheme increases its congestion window size conservatively to guarantee bandwidth occupancy of regular Cubic TCP. So, regular Cubic TCP does not suffer from congestion loss events due to competing flows. After 20 seconds, the proposed scheme keeps its congestion window size below that of regular Cubic TCP.

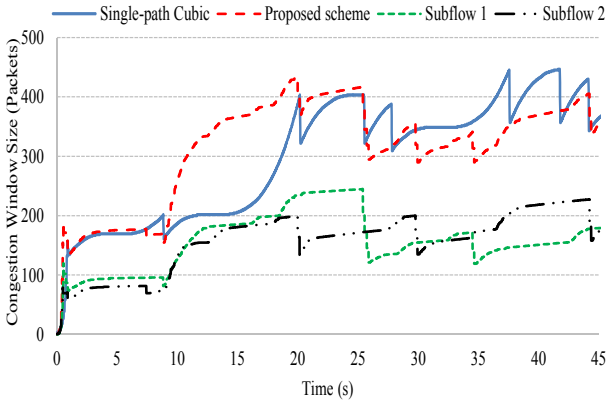


Figure 11. Comparison of congestion window between proposed scheme and single-path Cubic TCP

VI. Conclusion

We have proposed an efficient MPTCP-based congestion control scheme in HBDP networks. The goal of the proposed scheme is to ensure that the three IEFT requirements for MPTCP are met, while especially overcoming the trade-off problems that were encountered in previous works and enhancing the efficiency of load balancing. Our proposed scheme can be divided into two main mechanisms, the router buffer based congestion control mechanism and the expected throughput based load balancing mechanism. The former mechanism checks congestion levels and its variation accurately by monitoring router buffer states. After that it determines an increment of the congestion window so as to occupy bandwidth without wasting available bandwidth, by using a graph for the relationship between the congestion level and the increment of the congestion window. The latter mechanism estimates the properties of each path and transfers traffic from one path to the other properly by calculating the expected throughput in each path. Simulations show that the proposed scheme performs well in comparison with those of previous works and achieves the above mentioned goals.

Acknowledgment

The work reported in this paper was conducted during the sabbatical year of Kwangwoon University in 2017.

References

[1] R. Stewart, "Stream control transmission protocol," IETF RFC 4960, Sept. 2007.

[2] J. Iyengar, P. D. Amer, and R. Stewart, "Concurrent multipath transfer using SCTP multi-homing over independent end-to-end paths," IEEE/ACM Trans. Netw., vol. 14, no. 5, pp. 951–964, Oct. 2006. doi:10.1109/TNET.2006.882843

[3] P. Vo, T. Le, S. Lee, C. Hong, B. Kim, and H. Song, "mReno: A Practical Multipath Congestion Control for Communication Networks," Computing, vol. 96, no. 3, pp. 189–205, Mar. 2014. doi:10.1007/s00607-013-0341-1

[4] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses," IETF RFC 6824, Jan. 2013.

[5] M. Honda, Y. Nishida, L. Eggert, P. Sarolahti, and H. Tokuda, "Multi-path congestion control for shared bottleneck," Proc. 7th PFLDNet Workshop, 2009.

[6] C. Raiciu, M. Handley, and D. Wischik, "Coupled congestion control for multi-path transport protocols," IETF RFC 6356, Oct. 2011.

[7] D. Wischik, C. Raiciu, A. Greenhalgh, M. Handley, "Design, implementation and evaluation of congestion control for multi-path TCP," Proc. 8th USENIX NSDI Conf., vol.11, pp.8–22, Mar. 2011.

[8] J. Zhao, C. Xu, J. Guan, H. Zhang, "A fluid model of multipath TCP algorithm: Fairness design with congestion balancing," Proc. IEEE Int. Conf. on Commun., London, 2015, pp. 6965–6970. doi:10.1109/ICC.2015.7249436

[9] C. Xu, J. Zhao, G. Muntean, "Congestion Control Design for Multipath Transport Protocols: A Survey," IEEE Commun. Surveys & Tutorials, vol. 18, no. 4, pp. 2948–2969, Apr. 2016. doi:10.1109/COMST.2016.2558818

[10] R. Gonzalez, J. Pradilla, M. Esteve, C. E. Palau, "Hybrid delay-based congestion control for multipath TCP," Proc. IEEE Mediterranean Electrotechnical Conf., Limassol, 2016, pp. 1–6. doi: 10.1109/MELCON.2016.7495389

[11] T. Le, C. Hong, and S. Lee, "Multi-path binomial congestion control algorithms," IEICE Trans. Commun., vol. E95-B, no. 6, pp. 1934–1943, Jun. 2012. doi:10.1587/transcom.E95.B.1934.

[12] T. Le, C. Hong, and S. Lee, "MPCubic: an extended Cubic TCP for multiple paths over high bandwidth-delay networks," Proc. Int. Conf. on ICT Convergence, Seoul, 2011, pp. 34–39. doi:10.1109/ICTC.2011.6082546

[13] B. P. Ha, B. Y. Tran, T. A. Le, C. H. Tran, "A hybrid multi-path congestion control algorithm for high speed and/or long delay networks," Proc. 2014 Advanced Tech. Commun., Hanoi, 2014, pp. 452–456. doi:10.1109/ATC.2014.7043430

[14] S. Ferlin, Ö. Alay, T. Dreibholz, D. A. Hayes , M. Welzl, "Revisiting congestion control for multipath TCP with shared bottleneck detection," Proc. IEEE INFOCOM, San Francisco, 2016, pp. 1–9. doi: 10.1109/INFOCOM.2016.7524599

[15] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, J.-Y. Le Boudec, "MPTCP is not pareto-optimal: performance issues and a possible solution," Proc. 8th Int. Conf. on Emerging Netw. Experiments and Technologies, Nice, 2012, pp. 1–12. doi: 10.1145/2413176.2413178

[16] Q. Peng, A. Walid, J. Hwang, S. Low, "Multipath TCP: analysis, design, and implementation," IEEE/ACM Trans. Netw., vol. 24, no. 1, pp. 596–609, Feb. 2015. doi:10.1145/2413176.2413178

[17] Y. Cao, M. Xu, X. Fu, "Delay-based congestion control for Multipath TCP," Proc. 20th IEEE Int. Conf. on Network Protocols, Austin, 2012, pp. 1–10.

[18] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," ACM SIGOPS Operating System Review, vol. 42, no. 5, pp. 64–74, Jul. 2008. doi:10.1145/1400097.1400105

[19] V. Konda and J. Kaur, "RAPID: shrinking the congestion control timescale," Proc. IEEE INFOCOM, Rio de Janeiro, 2009, pp. 1-9. doi:10.1109/INFCOM.2009.5061900

[20] J. Gettys and K. Nichols, "Bufferbloat: dark buffers in the Internet," Commun. ACM, vol. 55, no. 1, pp. 57–65, Jan. 2012. doi:10.1145/2063176.2063196

[21] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: new techniques for congestion detection and avoidance," Proc. ACM SIGCOMM Symposium, vol. 24, no. 4, pp. 24–35, Oct. 1994. doi:10.1145/190314.190317

[22] H. Jung, S. Kim, and S. Kang, "Adaptive delay-based congestion control for high bandwidth-delay product networks," Proc. IEEE INFOCOM, Shanghai, 2011, pp. 2885–2893. doi:10.1109/INFCOM.2011.5935127

[23] S. Floyd, M. Handley, and J. Padhye, "A comparison of equalion-based and AIMD congestion control," AT&T Center for Internet Research, 2000.

[24] L. Xu, K. Harfoush and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," Proc. IEEE INFOCOM, vol. 4, pp.2514–2524, Mar. 2004.

[25] D. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks,"Comput. Netw., ISDN Syst., vol. 17, no. 1, pp. 1–14, Jun. 1989. doi:10.1016/0169-7552(89)90019-6