A Multipath Cubic TCP Congestion Control with Multipath Fast Recovery over High Bandwidth-Delay Product Networks*

Tuan Anh LE†(a), Rim HAW(b), Nonmembers, Choong Seon HONG†(c), Member, and Sungwon LEE†(d), Nonmember

SUMMARY Cubic TCP, one of transport protocols designed for high bandwidth-delay product (BDP) networks, has successfully been deployed in the Internet. Multi-homed computers with multiple interfaces to access the Internet via high speed links will become more popular. In this work, we introduce an extended version of Cubic TCP for multiple paths, called MPCubic. The extension process is approached from an analysis model of Cubic by using coordinated congestion control between paths. MPCubic can spread its traffic across paths in load-balancing manner, while preserving fair sharing with regular TCP, Cubic, and MPTCP at common bottlenecks. Moreover, to improve resilience to link failure, we propose a multipath fast recovery algorithm. The algorithm can significantly reduce the recovery time of data rate after restoration of failed links. These techniques can be useful for resilient high-bandwidth applications (for example, tele-health conference) in disaster-affected areas. Our simulation results show that MPCubic can achieve stability, throughput improvement, fairness, load-balancing, and quick data rate recovery from link failure under a variety of network conditions.

key words: multipath congestion control, Cubic TCP for multiple paths, load-balancing, resource pooling, fairness, resilience, disaster

1. Introduction

Cubic TCP*†[3] is one of the transport protocols over high bandwidth-delay product (BDP) networks which has successfully been deployed in the Internet. Its congestion window growth is implemented by a cubic function, which increases the congestion window in terms of concave and convex curves. Such congestion window growth depends on the time between two consecutive congestion events. Hence Cubic not only improves performance in high BDP networks but also ensures fairness at common bottlenecks with heterogeneous round trip time (RTT) Cubic flows.

In addition, multi-homed computers with multiple interfaces to access the Internet via high speed links will become more popular. This would promise a huge opportunity to improve the performance and resilience via transport protocol where multiple links are used simultaneously [4].

Communication infrastructure has to meet up challenges like fast information diffusion, failure recovery, bandwidth-availability in a disaster-affected region [5], [6].

An instance of it is tele-medicine application that becomes vital in an area just struck with natural or man-made disaster [7], [8]. Caregivers at those areas take assistance of remote hospitals or clinicians by video-conference or by uploading X-ray report of victims [7]. We have to ensure high-bandwidth and failure recovery mechanism in communication. For example, once any link along path 1 as shown in Fig. 1 fails, user’s session can be maintained by using path 2. In contrast to multipath protocol, user’s session using single-path protocol is interrupted. When a router at ISP 1 detects such a link failure, it switches the route to an alternative link, so multipath protocol can aggregate more network bandwidth. However, when some links along paths are intermittent due to a mild crisis, many retransmission timeout events occur. So, the performance of any multipath protocol degrades significantly.

A multipath TCP protocol allows the creation of simultaneous multiple sub-flows amongst two end hosts through multiple network interfaces [9] where each sub-flow maintains sending data packets over a path. Coupled congestion control (MPTCP) [10] moves more traffic on the less-congested paths as a load-balancing mechanism. MPTCP was designed to be backward-compatible with regular TCP (TCP Reno) [11], which has proven to have low utilization of bandwidth available in high BDP networks. So far, there is no transport protocol supporting multiple paths over high BDP networks. Therefore an urgent requirement is to design or extend a multipath transport protocol for high BDP networks.

In this work, we introduce an extended version of Cubic† for multiple paths, called MPCubic. The extension process is approached from the analysis model of Cubic. By establishing coordinated congestion control between paths

* Translated from the English original.
** The previous version of Cubic, called BIC-TCP, [2] has been the default TCP algorithm in Linux since 2006.
† The short version [1] of this paper was presented at the International Conference on ICT Convergence in 2011.
and constraining fairness to regular Cubic, MPCubic is able to move traffic away from congested paths to uncongested paths, and fairly share bandwidth with regular Cubic and TCP at common bottleneck. Moreover, to improve resilience to link failure, we propose a multipath fast recovery algorithm. The algorithm can significantly reduce the recovery time of data rate after restoration of failed links.

The rest of this paper is organized as follows: Sect. 2 gives related work which summarizes previous works relevant to multipath congestion control and retransmission. We describe the Cubic background in Sect. 3, and then the details of the extension process towards MPCubic in Sect. 4. In Sect. 5, we describe the details of the fast recovery algorithm. Section 6 presents the results of simulation in terms of congestion window evolution, fairness, throughput, resource pooling, and resilience to link failure. Finally, we offer our conclusions and future work in Sect. 7.

2. Related Work

Major issues in multipath transport protocol are as follows: Spreading sequence number space across paths per sub-flow and/or per multipath flow, ordering/scheduling packet across paths, congestion control (for sub-flow and multipath flow), packet retransmission at sub-flow and/or multipath connection-level, and etc. In this paper, we summarize several solutions for multipath congestion control and retransmission problems.

Multiple paths TCP (mTCP) [12] focuses on striping data packets across multiple paths, and detecting common bottleneck to alleviate unfair sharing by computing correlation between fast retransmit intervals.

A sub-flow in Parallel TCP (pTCP) [13] as well as concurrent transfer multipath (CMT) over SCTP [14] sends data packets independent of each others because it uses uncoordinated congestion control algorithm (using regular TCP). These protocols cannot fairly share with regular TCP flows at common bottleneck since they do not handle/detect the common bottleneck. Moreover, CMT over SCTP utilizes five retransmission policies for packet losses and retransmission timeout as follows: RTX-SAMe - All retransmissions are sent to the same path; RTX-ASAP - A retransmission of data packet depends on congestion window available at the time of retransmission; RTX-CWND - The choice of path for retransmission is based on the largest congestion window; RTX-SSTHRESH - The path having the largest slow-start threshold is chosen for retransmission; RTX-LOSSRATe - The lowest loss path rate is chosen for retransmission.

Reliable Multiplexing Transport Protocol (R-MTP) [15] was designed for wireless networks. R-MTP’s sub-flow tracks packet arrival time to estimate the available bandwidth for rate control, and for congestion detection.

To improve fairness to regular TCP at shared bottleneck, the weighted TCP [16] uses a fixed weight in allocating bandwidth to each sub-flows. The weight is chosen by $1/N^2$, where $N$ is the number of paths in a multipath TCP session. Obviously the weighted TCP could fairly share if RTTs of flows are equal. This mechanism takes advantage of the simplicity of deployment and fair allocation without any shared bottleneck detection.

MPTCP [10] collects resources on all paths as a single resource, as suggested in the resource pooling principle [17]. The design goals for MPTCP are (i) fairness to regular TCP when several sub-flows compete with TCP flows at a shared bottleneck link, (ii) performance improvement, and (iii) load-balancing - A multipath flow should shift more traffic from the more congested paths onto the less congested paths. Following the design technique in [18], at the first step, the multipath increase and decrease rules on path $s$ aim to satisfy: the throughput of MPTCP is equivalent to that of the single-path TCP (with the same RTT). The rules are as follows

$$\text{Inc.}: w_s(t) \leftarrow w_s(t) + a^{\text{TcP}} w_{\text{total}}(t),$$

$$\text{Dec.}: w_s(t) \leftarrow w_s(t) - b^{\text{TcP}} w_{\text{total}}(t),$$

(1)

where $w(t)$ is the congestion window size at time $t$; $w_{\text{total}}(t) = \sum w_s(t)$; $a^{\text{TcP}} = 1$ and $b^{\text{TcP}} = 0.5$ are the increase and decrease factors, respectively. $[\cdot]^+ = \max(\cdot, 0)$. Such a multipath algorithm will fairly share with regular TCP because regular TCP also increases its congestion window $w$ by $a^{\text{TcP}} w$ per ACK. The multipath protocol as in (1) gives more fluctuation, because when $w_s$ decreases by $b^{\text{TcP}} w_{\text{total}}$ and if $w_s \leq b^{\text{MPTC}} w_{\text{total}}$, then $w_s$ would be truncated to 0. The fluctuation phenomenon occurs for any multipath transport protocol when the multipath protocol sends packets on one path for a moment, then it switches sending on the other, and so on [18], [19]. So resource pooling of any multipath transport protocol is adversely affected in such condition [18].

In the next step, the increase and decrease terms are multiplied by $w_s/w_{\text{total}}$ in order to avoid the above problems. Unfortunately, the fluctuation in the modified multipath protocol is not much alleviated. Therefore, the increase term is slightly modified by using $a^{\text{TcP}} w_{\text{total}} = (aw_s/w_{\text{total}})^{1-\phi}$ where the $\phi$ parameter is aggressiveness of MPTCP. To guarantee the fairness goal, the window increase is limited to that of regular TCP (i.e., $a^{\text{TcP}} w$) by using $\min(\cdot)$. Finally, each MPTCP’s sub-flow on path $s$ controls its window as

$$\text{Inc.}: w_s(t) \leftarrow w_s(t) - 1$$

$$\text{Dec.}: w_s(t) \leftarrow w_s(t) - b^{\text{TcP}} w_s(t),$$

(2)

where

$$a = \hat{w}_{\text{total}} \frac{\alpha^{1-\phi}}{\alpha^{1-\phi}/\alpha^{1-\phi}}$$

(3)

From now, we prefer Cubic or regular Cubic as Cubic TCP, and regular TCP as TCP Reno.
\( \hat{w}_s \) and \( \hat{w}_{total} \) are the equilibrium value of \( w_s(t) \) and \( w_{total}(t) \), respectively. \( \phi \) is set to 1. Our proposed multipath Cubic protocol is designed from the single-path Cubic algorithm, instead of starting with the multipath algorithm as in the design of MPTCP.

### 3. Cubic Background

In this section, we briefly describe the Cubic algorithm for single-path transmission, and then its performance analysis.

Cubic uses a cubic function for congestion window growth in terms of concave and convex curves as follows:

\[
w(t) = C(t - K)^3 + W_{\text{max}},
\]

where \( C \) denotes a constant in regular Cubic. \( t \) denotes the time elapsed from the last packet loss event, \( K = (\beta W_{\text{max}}/C)^{1/3} \) denotes a period of time between two consecutive packet loss events (called a congestion epoch), and \( W_{\text{max}} \) is the congestion window size where a packet loss occurs. When detecting packet loss via duplicated ACKs, Cubic decreases its congestion window by a constant factor of \( \beta \). Moreover, Cubic can operate in TCP-friendly mode as described in [3]. The average congestion window [3] in the steady state for the concave window curve is

\[
\bar{w} = \left( \frac{C(4 - \beta)RTT^3}{4\beta p^2} \right)^{1/3},
\]

where \( p \) denotes the packet loss rate during a Cubic session.

### 4. Extension Process Towards MPCubic

Applying the design technique for MPTCP into the extension of Cubic towards multiple paths results in complication in analysis and computation. The details are described in the appendix. In this section, we describe the extension towards MPCubic using our technique, where MPCubic is designed from the single-path Cubic algorithm instead of from the multipath algorithm as in the design of MPTCP.

Now, we execute a simple method that starts with the single-path Cubic algorithm. We agree that any multipath transport protocol should satisfy three above goals (except MPCubic should be fair to single-path Cubic, rather than regular TCP). We consider sub-flow’s congestion window growth on path \( s \) from the single-path Cubic model as follows

\[
w_s(t) = \min \left( \delta C(t_s - K_s)^3, C(t_s - K_s)^3 \right) + W_{s, \text{max}},
\]

where \( t_s \) denotes the time elapsed from the last packet loss event on path \( s \); and \( K_s \) denotes a period of time between two consecutive packet loss events; \( \delta \) indicates the aggressiveness level of the congestion window growth so that MPCubic can fairly share to single-path Cubic as several sub-flows compete a Cubic flow at a shared bottleneck link, while it can improve total throughput as paths are disjoint. To ensure fairness, MPCubic’s window increment per path should not exceed that of regular Cubic by using the right argument in \( \min(\cdot) \).

We assume that a packet loss event is occurred on path \( s \) as the congestion window reaches \( W_{s, \text{max}} \). It is then decreased by factor \( \beta \) as used in regular Cubic [3]. Therefore, we calculate \( K_s \) such that

\[
\min \left( -\delta CK_s^3, -CK_s^3 \right) + W_{s, \text{max}} = (1 - \beta)W_{s, \text{max}},
\]

\[
\Rightarrow K_s = \left( \frac{\beta W_{s, \text{max}}}{\min(\delta, 1)C} \right)^{1/3}.
\]

To determine \( \delta \), we should analyze MPCubic performance model in the steady state. We consider MPCubic operating in the congestion window increase phase of the concavity mode as described in [3]. The number of round trip times in a congestion epoch on path \( s \) is \( K_s/RTT_s \), then the number of packets sent in that congestion epoch is

\[
\min \left( \sum_{i=0}^{K_s} [\delta C(iT_s - K_s)^3 + W_s], \sum_{i=0}^{K_s} [C(iT_s - K_s)^3 + W_s] \right)
\]

\[
= \sum_{i=0}^{K_s} [\min(\delta, 1)C(iT_s - K_s)^3 + W_s] + \frac{W_sK_s}{T_s} - \min(\delta, 1)C \sum_{i=0}^{K_s} (K_s - iT_s)^3
\]

\[
= \frac{W_sK_s}{T_s} - \min(\delta, 1)C \sum_{j=0}^{K_s} (jT_s)^3; jT_s \leq K_s - iT_s
\]

\[
= \frac{W_sK_s}{T_s} - \min(\delta, 1)C T_s^3 \left( \frac{K_s^4}{4} + \frac{K_s^2}{2} \right)
\]

\[
= \frac{W_sK_s}{T_s} - \min(\delta, 1)C T_s^3 \left( \frac{K_s^4}{4} \right)
\]

\[
= \frac{W_sK_s}{T_s} - \min(\delta, 1)CK_s^3 \frac{T_s}{4T_s},
\]

where \( iT_s \) denotes the \( i \)th \( RTT_s \) of a sub-flow in a congestion epoch on path \( s \), \( W_s \) denotes the last congestion window size before a packet loss event. When the congestion window reaches \( W_s \) at the end of the congestion epoch, one of these packets on path \( s \) will be dropped with the packet loss probability \( p_s \). Therefore, (7) must be
\[
\frac{W_s K_s}{T_s} - \min(\delta, 1) \frac{C K_s^4}{4T_s^3} = \frac{1}{p_s}. \tag{8}
\]

By substituting (6) into (8), we obtain \( W_s \) as follows
\[
W_s = \left( \min(\delta, 1) \frac{C}{\beta} \right)^{1/4} \left( \frac{4T_s}{(4 - \beta)p_s} \right)^{3/4}. \tag{9}
\]

For the deterministic model, the average congestion window \( \overline{w}_s \) on path \( s \) is calculated according to the number of packets sent over the number of RTTs for one congestion epoch as follows
\[
\overline{w}_s = \left( \frac{W_s K_s}{T_s} - \min(\delta, 1) \frac{C K_s^4}{4T_s^3} \right) \left( K_s / T_s \right) = \left( \frac{4 - \beta}{4} \right) W_s = \left( \min(\delta, 1) \frac{C(4 - \beta)T_s^3}{4\beta p_s^3} \right)^{1/4}. \tag{10}
\]

If a Cubic flow ran on path \( s \), its average window \( \overline{w}^{\text{cubic}}_s \) (from (4)) would be
\[
\overline{w}^{\text{cubic}}_s = \left( \frac{C(4 - \beta)T_s^3}{4\beta p_s^3} \right)^{1/4}. \tag{11}
\]

Equations (10) and (11) give the relationship between MPCubic and Cubic as follows
\[
\overline{w}_s = \overline{w}^{\text{cubic}}_s \min(\delta^{1/4}, 1),
\]
or
\[
\overline{w}^{\text{Cubic}}_s = \max \left( \overline{w}_s, \overline{w}^{\text{cubic}}_s \right). \tag{12}
\]

The improved throughput and fairness goals suggest that the total throughput of a multipath Cubic flow should equal that of a Cubic flow on the best path for it [19], i.e.,
\[
\sum_r \frac{\overline{w}_r}{RTT_r} = \max_r \left( \frac{\overline{w}^{\text{Cubic}}_r}{RTT_r} \right). \tag{13}
\]

By substituting (12) into (13), we obtain
\[
\sum_r \frac{\overline{w}_r}{RTT_r} = \max_r \left( \max \left( \frac{\overline{w}_r}{RTT_r}, \frac{\overline{w}_r}{\delta^{1/4}RTT_r} \right) \right).
\]

Since \( \overline{w}_s \) never equals zero, equality of above equation cannot occur, therefore
\[
\sum_r \frac{\overline{w}_r}{RTT_r} = \max_r \left( \frac{\overline{w}_r}{\delta^{1/4}RTT_r} \right).
\]

By solving for \( \delta \), we obtain
\[
\delta = \max_r \left( \left( \frac{\overline{w}_r}{RTT_r} \right)^4 / \sum_r \left( \frac{\overline{w}_r}{RTT_r} \right)^4 \right).
\]

\( \delta \) in the equation above is shared between the paths and just depends on the maximum throughput among sub-flows. However, the congestion window sizes on the congested paths are always smaller than that on the lightly loaded paths. To perform load-balancing between paths, the congestion windows on the congested paths should be increased more gradually than that on the better paths [18]. This means that \( \delta \) for the worse paths would be smaller. Additionally, \( \delta \) should be based on the congestion window size in order to compensate for window increments for the lightly loaded and longer RTT paths because the longer RTT paths expect the larger congestion windows. Therefore we slightly modify \( \delta \), and then replace \( \delta \) with \( \delta_s \) as
\[
\delta_s = \overline{w}_s \max \left( \frac{\overline{w}_r}{RTT_r^2} \right) / \left( \sum_r \frac{\overline{w}_r}{RTT_r} \right)^4,
\]

where \( k \) is a trade-off parameter between resource pooling and protocol fluctuation, \( 0 \leq k \leq 4 \). Here is an example of the role of \( \delta_s \) in the case where all the paths have the same RTT. In this case, a sub-flow on path \( s \) will grow its congestion window based on its window size and \( \delta_s \), which reflect the path’s available bandwidth. If a path is more congested, the congestion window on that path will be smaller, so \( \delta_s \) on that path gets the value smaller than those on other paths.

In this paper, some results are obtained by comparing MPCubic with a uncoordinated multipath Cubic (called un-MPCubic), where the congestion control algorithm (using regular Cubic) on each path operates independently each other. For two-path unMPCubic flow, we set the window increase scale factor of \((1/2)^k\) so that each sub-flow gets half the throughput of what a regular Cubic gets.

The following experiment results present the choice of \( k \) such that it achieves good resource pooling but less fluctuation. Experiments are run on the network configuration as shown in Fig. 5(a). The exogenous drops are regarded as the uniform random process. We use the histogram to represent the evolution of congestion windows. Each point \((cwnd_{1}, cwnd_{2})\) shown in Fig. 2 represents the congestion window size on paths 1 and 2 at time \( t \), and a darker area implies that \( cwnd_{1} \) and \( cwnd_{2} \) spend more time in that area. The top plots in Fig. 2 show that when \( p_1 = p_2 \), unMPCubic demonstrates non-variation in increment of both windows; MPCubic with \( k \leq 2 \) gives the slight variations, and for \( k = 3 \), pairs of window increment move from one axis to another, and so on. Whereas when \( p_1 > p_2 \), as shown in the bottom plots in Fig. 2, MPCubic demonstrates the capability of less aggressive increase in the congestion window on path 1 and the more aggressive increase on path 2 as \( k \) value increases. There is a reasonable trade-off between fluctuation and load-balancing when we choose \( k = 2 \).

For MPCubic operating in TCP-friendly region, we must separately calculate \( k_{\text{TCP}} \), such that MPCubic still exists with not only Cubic flows but also TCP (TCP Reno) flows. The average congestion window \( \overline{w}_s^{MTCP} \) of TCP flow on path \( s \) is

\[\text{The congestion window size is set to the smallest value of 1 as initiating, resetting, or occurring a timeout.}\]
where $\alpha^{TCP} = 1$ denotes the increase factor for each RTT, $\beta^{TCP} = 1/2$ denotes the multiplicative decrease factor for a packet loss event. The congestion window of multipath TCP on path $s$ increases by $\min\left(\delta^{MPTCP} \alpha^{TCP}, 1\right)$ for each RTT, and decreases as the single-path TCP does. Hence, $\overline{w}_s^{MPTCP}$ in the steady state is

$$
\overline{w}_s^{MPTCP} = \left(\min(\delta^{MPTCP}, 1) \frac{\alpha^{TCP}(2 - \beta^{TCP})}{2 \beta^{TCP} p_s}\right)^{1/2}.
$$

(14)

By substituting (14) into (15) and then solving for $\delta^{MPTCP}$ with the constraint similar to (13) but for $\overline{w}_s^{MPTCP}$, we obtain

$$
\delta^{MPTCP} = \max_r \left\{ \left( \frac{\overline{w}_r}{RTT_r} \right)^2 \left( \sum_r \frac{\overline{w}_r}{RTT_r} \right)^2 \right\}.
$$

(15)

Although our extension approach is different from MPTCP design [10], we should choose $\delta^{MPTCP}$ in TCP-friendly region, such that the throughput of both approaches in the steady state are the same (i.e., $\overline{w}_s = \alpha^{TCP} \max_r \left( \frac{\overline{w}_r}{RTT_r} \right) / \beta^{TCP} p_s (\sum_r \overline{w}_r/RTT_r)^2$ if neglecting $1/w_s(t)$ in (min())). Therefore, we adopt $\delta_s^{MPTCP}$ as

$$
\delta_s^{MPTCP} = \overline{w}_s \max_r \left\{ \frac{\overline{w}_r}{RTT_r} \right\} \left( \sum_r \frac{\overline{w}_r}{RTT_r} \right)^2.
$$

(16)

Regular TCP in the steady state gives $\overline{w}_s^{TCP} = (3/2)^{1/2}$ [20]. Therefore, multipath TCP’s increase factor will be equivalent to that of regular TCP if it is

$$
\frac{\alpha^{TCP}(2 - \beta^{TCP})}{2 \beta^{TCP} p_s} = \left(\min(\delta_s^{MPTCP}, 1) \frac{3}{2 p_s} \right)^{1/2}.
$$

(17)

Finally, we present multipath TCP’s window growth function in MPCubic with respect to the elapsed time $t$ on path $s$ as

$$
\overline{w}_s^{MPCubic}(t) = (1 - \beta^{TCP})W_{max} + \min(\delta_s^{MPCubic}, 1) \frac{3 \beta^{TCP}}{2 - \beta^{TCP} RTT_s} \frac{t}{RTT_s}.
$$

The calculations of values of $\delta_s$ and $\delta_s^{TCP}$ are based on the average congestion window sizes and round trip times, which are estimated by smoothing their current values. The smoothed RTT is estimated similarly to regular TCP. The average congestion window $^*$ is estimated by using the exponentially weighted moving average technique (EWMA), with weight factor $\gamma = 0.125$.

5. Multipath Fast Recovery Algorithm

In this section, we present a simple algorithm to help fast recovery in multipath transport protocol when the link along paths fails. In this paper, we consider a situation, where the time between link failure and resilience is long enough so that transport protocol is triggered by a retransmission timer.

$^*$MPTCP [10] uses instantaneous congestion window in alpha parameter calculation. However, we estimate the average value in MPCubic because MPCubic’s window increase may be large (due to the characteristic of cubic function), instead of linear increase in MPTCP.
When a temporary or permanent path failure occurs to a sub-flow, retransmission timeouts are inevitable. As a result, sub-flow’s congestion window is set to 1 segment and its $ssthresh$ is decreased by factor of $\beta$ (the number of $ssthresh$ decrease times equal the number of timeout events), and multipath flow may be stalled due to out-of-order delivery to application. To improve recovery of end-to-end multipath transmission to path failure, we propose a multipath fast recovery algorithm shown in Alg. 1. When a timeout event occurs on path $s$, the sub-flow on that path finds the path $r$ that has the highest throughput ($x_r$) and not in the retransmission timeout state ($RTO$), denoted by $x_{snRTO}$, as shown at (b.1). The pseudocode at (b.2) describes that unacked data portions on path $s$ (denoted by $D_{snacked}$) are then moved into the sub-flow’s extra queue on path $r$, denoted by $q_r$. The unacked data in $q_r$ is immediately retransmitted on the best path $r$ as soon as possible. For a temporary link failure, transport protocol may suffer several consecutive timeouts, $ssthresh$ is reduced to the minimum value of 2; after link recovery and the end of timeouts, transport protocol will quickly enter the congestion avoidance phase to probe the network bandwidth at such a very low threshold. Hence, transport protocol is slow to respond to resilience of the network in its expected rate. Especially, this strongly affects multipath transport protocol’s performance because the sub-flow on bad path has both congestion window and $\delta$ very small.

To prevent from reducing unreasonably in multipath transport protocol, we use the bandwidth estimate technique proposed in [21] to update $ssthresh$ to the estimated bottleneck bandwidth, as shown at (b.3), where $eBW_s$ denotes the estimated available bandwidth, $\minRTT_s$ is the minimum RTT observed during sub-flow session, and $MSS_s$ denotes the length of the TCP segment on path $s$. Because the estimated bottleneck window represents the available bandwidth at bottleneck, $ssthresh$ will be updated to the optimal value if timeouts are not caused by heavy network congestion, and to the small value otherwise. So, Alg. 1 helps multipath transport protocol to quickly respond to resilience of the network by increasing congestion window exponentially to $ssthresh$ in the slow-start phase. If timeouts are caused by heavy network congestion, Alg. 1 does not send more data packets to the heavy congestion link. We do not change the exponential backoff of retransmission timeout of each sub-flow to avoid collapse of the network. Therefore, Alg. 1 can improve throughput performance and can reduce out-of-order delivery to applications due to timeouts, while preserving transport protocol backoff. Moreover, the algorithm can be applied to any multipath transport protocol.

### 6. Performance Evaluation

In this section, we evaluate fairness, throughput, load-balancing, and resilience of MPCubic. In some evaluations, we compare to unMPCubic. Simulation experiments are run on NS-2 [22]. In all simulations, the transport protocols is set by the selective acknowledgment (SACK) option [23] and a 1000-byte data packet. In order to easily implement connection-level acknowledgements and retransmissions, we use two levels of sequence spaces as described in [24]. Alg. 1 is not set as a default option in simulations. We use the droptail discipline to demonstrate the evolution of congestion window as shown in Fig. 3(a). The other experiments use the random early discard (RED) queue management [25] at bottleneck routers to prevent them from synchronized packet drops. For RED parameters, we set the minimum threshold to 0.2 $\times$ bandwidth-delay product, and the maximum threshold to triple the minimum value. Data rates are sampled every 5 seconds.

#### 6.1 MPCubic Congestion Window Evolution

In this section, we demonstrate MPCubic congestion window evolution compared with that of a regular single-path Cubic flow. In our experiment, a two-path MPCubic flow is run on separate paths with the same network parameters with FIFO queue as shown in Fig. 3(a). A regular single-path Cubic flow is separately simulated on one path. We define a window growth cycle including two consecutive
congestion epochs: a concave region and a concave-convex region. Figure 4 shows that the regular Cubic flow is more aggressive in window growth than two-path MPCubic flow. A window growth cycle of sub-flow in MPCubic is approximately 2.5 times as long as that of regular Cubic. This result matches the theory above as follows: From (6), and neglecting argument 1 in \( \min(\cdot) \), \( K_s = \frac{K_{Cubic} s}{\delta_1^{1/3}} \). Because of using two paths with the same RTT, \( \delta_s = (1/2)^4 \). Therefore \( K_s = 2^{4/3} K_{Cubic}^s \). Since two paths’ parameters are equal, evolution of two congestion windows in Fig. 4 overlaps.

6.2 Fairness

In this section, we investigate MPCubic’s fairness to regular Cubic, TCP (TCP Reno), and MPTCP at common bottleneck. In the first experiment, a two-path MPCubic flow shares with a regular Cubic flow at a link as shown in Fig. 3(b), with link’s capacity of 50 Mbps and propagation delay of 100 ms. Figure 5 plots congestion window evolution and throughput of two sub-flows and one regular Cubic flow, as a two-path MPCubic flow competes against a regular Cubic flow. Figure 5(a) shows that there are slight oscillation in congestion window evolution of two sub-flows and regular Cubic’s window size is doubled. As a result, regular Cubic’s average throughput is 23.60 Mbps, and MPCubic gets 23.74 Mbps (their instantaneous rates are plotted in Fig. 5(b)). MPCubic’s fairness is achieved by adjusting \( \delta_1 \) and \( \delta_2 \). For the same RTT paths, the optimal values of \( \delta_1 \) and \( \delta_2 \) are 0.0625, the average values of \( \delta_1 \) and \( \delta_2 \) in the experiment are approximately 0.079 and 0.083, respectively. The differences in calculating \( \delta_s \) are due to large variance in congestion window increase in concave-convex profile, and oscillations of RTT.

Next we run the same simulations above on Fig. 3(b) but replacing Cubic with a regular TCP with link’s capacity of 10 Mbps and propagation delay of 16 ms. We choose a short RTT (i.e., 32 ms), such that MPCubic operates in the TCP-friendliness mode as found in [3]. Figure 6(a) shows that each sub-flow takes half data rate of a regular TCP flow. So, two-path MPCubic is not greedy to capture more bandwidth than the regular TCP, as shown in Fig. 6(b). The average values of \( \delta_1^{TCP} \) and \( \delta_2^{TCP} \) are observed as around 0.31 and 0.28, respectively, compared with the same ideal value of 0.25. Since MPCubic’s congestion window growth in the TCP-friendliness mode is linear, the factor causes the error in calculating \( \delta_s \) due to oscillations of RTT.

Finally, we examine fairness of two-path MPCubic flow competing against a two-path MPTCP on two disjoint links as shown in Fig. 3(c). When MPCubic is in the TCP-friendliness region, as shown in Fig. 7(a), each sub-flow of a multipath transport protocol fairly shares with each of the other multipath transport protocol. Hence, throughput of two multipath transport protocols are roughly equal, as shown in Fig. 7(b). The fair allocation is resulted from the same throughput model for both multipath transport protocols.

In summary, the above results imply that MPCubic can fairly share to regular Cubic at the joint bottleneck, and can preserve friendliness to both regular TCP and MPTCP on operating in the TCP-friendliness region.

6.3 Throughput

In this section, we demonstrate MPCubic’s the throughput improvement, compared to unMPCubic in a scenario shown in Fig. 3(d), where two links have the same configuration with link’s capacity of 100 Mbps and propagation delay of 50 ms; the background traffic is generated by fifteen Cubic flows on each link \( (n = m = 15) \). Figure 8(a) shows that two
Fig. 6  (a) Data rate and (b) the total data rate of a two-path MPCubic flow vs. a regular TCP flow at the shared bottleneck.

Fig. 7  (a) Data rate and (b) the total data rate of a two-path MPCubic flow and a two-path MPTCP flow competing against on two separate links.

Fig. 8  (a) and (b) Data rate of a two-path MPCubic flow vs. fifteen Cubic flows on each path; (c) and (d) data rate of two-path unMPCubic flow vs. fifteen Cubic flows on each path.
MPCubic sub-flows have slight fluctuation while obtaining network bandwidth. As expected, the sum of their throughput is higher than the average throughput of fifteen Cubic flows on each path, as shown in Fig. 8(b). In contrast, two-path unMPCubic flow, as shown in Fig. 8(c), gives less fluctuation in rate control in each path. And Fig. 8(d) shows that unMPCubic achieves the expected throughput, but lower than MPCubic. Note that the variance in estimation of $\delta_1$ and $\delta_2$ causes fluctuation in two MPCubic sub-flows. The peak rates are surged up by accelerating MPCubic’s window growth to probe network bandwidth after passing the last maximum window, as described in [3]. Because MPCubic gets the average values of $\delta_1 \approx 0.37$ and $\delta_2 \approx 0.3$ greater than unMPCubic (i.e., $\delta_1 = \delta_2 = 0.0625$), MPCubic is more aggressive in rate control than unMPCubic. Therefore, multipath transport protocols can yield more benefits as more paths are used.

In the next evaluation, we want to compare the performance between MPCubic and MPTCP in two disjoint paths. Each path has RTT of 200 ms and a bottleneck link’s capacity of 20 Mbps. Packet loss probability on both paths varies from 0.0005% to 1%. Figure 9 plots the average goodput at the receiver according to the packet loss rate on both paths. MPCubic can outperform MPTCP for the loss rate lower than 0.1%, and has the similar performance for loss rate greater than 0.1%. These results match the behaviors of single-path Cubic compared with regular TCP as described details in [3].

### 6.4 Resource Pooling

First, we evaluate resource pooling in terms of throughput and packet loss rate with a scenario shown in Fig. 3(e), where each multipath transport protocol has two paths. Simulations are run for 400 s. As known, a perfect resource pooling should produce the equal throughput for each multipath flow. Table 1 shows average throughput of MPCubic and unMPCubic. Each MPCubic flow appears more fair allocation to each other than unMPCubic flows, and MPCubic’s sum of throughput is higher than that of unMPCubic as well. Links 2 and 3 are shared between three flows, and the other links are used by only one sub-flow. So, there are more packet drops at links 2 and 3 than that at links 1 and 2. Hence, MPCubic flows A and C do not send their data to links 2 and 3 because MPCubic algorithm increases $cwnd$, more gradually when more packet drops appear on path s. As a result, the middle MPCubic flow has chance to get more bandwidth available at links 2 and 3. On the contrary, unMPCubic’s algorithm is independent of congestion control. So, links 2 and 3 suffered more congestion than links 1 and 4, since links 2 or 3 is connected by two unMPCubic sub-flows, instead of only one sub-flow for link 1 or link 4. Therefore, the middle unMPCubic flow using links 2 and 3 obtains the lowest throughput.

To evaluate the capability of traffic shifting from the congested link, we change link 1’s capacity from 50 Mbps to 25 Mbps. Table 2 shows that the packet loss rates always increase as link 1’s capacity is reduced, unMPCubic, in particular, produces increasing the packet loss rate at link 1 five times, while the packet loss rate caused by MPCubic is doubled. As known, the throughput $x = \pi/RTT$ of a transport protocol is always inversely proportional to the packet loss rate $p$\(^*\). When link 1’s capacity is decreased, flow A shifts its traffic from link 1 to link 2, and than flow B does the same manner from link 2 to link 3; after that flow C reduces its load at link 3 in order to move more load to the least congested link (link 4). Therefore, the packet loss generated by MPCubic is lower than that generated by unMPCubic.

Next, we investigate the impact of the different network congestion levels and the delay along paths on resource pooling. For such purpose, we focus on a scenario with a heterogeneous configuration, as shown in Fig. 3(d), where capacity and propagation delay of links 1 and 2 are set to (100 Mbps, 50 ms) and (80 Mbps, 200 ms), respectively. Heavy load at link 1 and light load at link 2 are generated by ten and five Cubic flows ($n = 10, m = 5$), respectively. Figure 10(a) shows that MPCubic produces throughput less aggressive on path 1, but more aggressive on path 2. So, the sum of throughput of MPCubic reaches the target throughput, which is equivalent to the average throughput of five Cubic flows on the best path (path 2), as shown in Fig. 10(b). We find that $\delta_2 \approx 0.725$ is very large, compared with $\delta_1 \approx 0.0271$. The large value of $\delta_2$ helps the MPCubic sub-flow to be more aggressive as compensating for window $\delta_1$.

\[^*\]For example, Cubic’s throughput is $x = (\frac{C_{\text{4R}} - \beta_1 - \beta_2}{4\pi \Delta T T_{\text{RTT}}})^{1/4}$ from (4). So, if a Cubic flow runs a link, the packet loss increases once the link’s capacity is decreased.

### Table 1: Throughput of MPCubic vs. unMPCubic.

<table>
<thead>
<tr>
<th>Flow ID</th>
<th>MPCubic</th>
<th>unMPCubic</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>63.7 Mbps</td>
<td>64.1 Mbps</td>
</tr>
<tr>
<td>B</td>
<td>43.7 Mbps</td>
<td>40.6 Mbps</td>
</tr>
<tr>
<td>C</td>
<td>64.2 Mbps</td>
<td>63.3 Mbps</td>
</tr>
<tr>
<td>Sum</td>
<td>171.6 Mbps</td>
<td>168 Mbps</td>
</tr>
</tbody>
</table>

### Table 2: Packet loss rate vs. link’s capacity.

<table>
<thead>
<tr>
<th>Link ID</th>
<th>MPCubic</th>
<th>unMPCubic</th>
</tr>
</thead>
<tbody>
<tr>
<td>link 1 = 50 Mbps</td>
<td>0.0029%</td>
<td>0.0029%</td>
</tr>
<tr>
<td>link 1 = 25 Mbps</td>
<td>0.0032%</td>
<td>0.0155%</td>
</tr>
</tbody>
</table>
increment for the lightly loaded and long RTT path (path 2). Whereas, there is no coordination of congestion control in unMPCubic, Fig. 10(c) shows that unMPCubic does not reduce its rate much on the congested path (path 1), and does not accelerate rate on the lightly loaded path (path 2) because unMPCubic just uses a fixed factor. Hence, unMPCubic gets the total throughput lower than five Cubic flows on the best path, path 2, as shown in Fig. 10(d).

6.5 Resilience from Link Failure

In this section, we demonstrate our multipath fast recovery algorithm to recover quickly after restoration of failed links, while preserving transport protocol backoff to help the network recover from heavy congestion. The simulations are run on two disjoint paths, as shown in Fig. 3(a), but replacing FIFO with RED, and RTT = 200 ms on path 2 is double of that on path 1. We want to manage links 1 and 2 to turn-off at time 100 s and 50 s, respectively, and to turn-on at the same time 150 s, shown in Fig. 11. In the absent of multipath fast recovery algorithm, MPCubic is very slow to recover after restoration of failed link to the expected operating point, as represented in evolution of the congestion window in Fig. 11(a). Note that the sub-flow on the longer RTT path (path 2) experiences longer exponential backoff timeout than the other on the shorter RTT path (path 1), and so it takes a longer time to start retransmission. In fact, the timeout of the sub-flow on path 2 is ended at time 179 s, and it then enters the slow-start phase. After retransmitting one packet successfully, the sub-flow on path 2 goes into the congestion avoidance phase using the cubic function for window growth. Since $cwnd_2 \ll cwnd_1$ at that time, $\delta_2 \ll \delta_1$. As a result, $cwnd_2$ growth is slow and prolonged. This makes another trade-off between resilience and load-
balancing. By contrast, Fig. 11(b) shows that Alg. 1 supports MPCubic to retransmit the unacked data on path 2 to on path 1 after a timeout event during link 1 turn-off, and to update $ssthresh$ to the optimal bottleneck bandwidth. When ending backoff times, two sub-flows can quickly capture more available bandwidth via exponential window growth in the slow-start phase.

To investigate the preservation of transport protocol backoff to help the network to recover from the heavy congestion, we repeat the above experiment but use background traffic generated by constant bit rate (CBR) at 50 Mbps, rather than turn-off the links. We find that after routers are congested as CBRs are turned on, the evolution of congestion windows with Alg. 1, as shown in Fig. 12, is similar to that without Alg. 1, as shown in Fig. 11(a). Because the background traffic do not completely block MPCubic traffic as in Fig. 11(a), resilience in Fig. 12 takes place earlier than that in Fig. 11(a). When heavy congestion occurs in such a case, Alg. 1 updates $ssthresh_1$ to the minimum value of 2, rather than the very small estimated value.

7. Conclusion

We propose a multipath Cubic congestion control algorithm with multipath fast recovery algorithm that can naturally provide performance improvement, load-balancing, and resilience from link failure over high BDP networks.

Through simulation results, we found that the proposed protocol can outperform MPTCP, improve throughput performance, and can move its traffic away from a congested link, while it can preserve fairness to single-path Cubic, regular TCP and MPTCP (with short RTTs). Moreover, the proposal can quickly recover data rate after restoration of failed links.

We believe that our technique can be generalized for extending current transport protocols for multiple paths. In the future, we will implement MPCubic algorithm in Linux for testbeds, and will try to extend other high performance transport protocols such as Compound TCP and FAST TCP.

Acknowledgment

This research was supported by the IT R&D program of KCA (10913-05004: Study on Architecture of Future Internet to Support Mobile Environments and Network Diver-

References

[21] T.A. Le and C.S. Hong, “TCP BaLDE for improving TCP per-

Fig. 12 The preservation of transport protocol backoff in Alg. 1.


Appendix

In this appendix, we explain the shortcomings of the design technique for MPTCP [18] when applying that method into the extension of Cubic.

At the first design step for MPTCP, we have to find out a multipath algorithm (i.e., both increase and decrease rules for multiple paths transmission) for Cubic. Such a candidate multipath algorithm is

\[
\text{Inc. } W_s(t) = \theta C(t - K)^3 + W_s, \\
\text{Dec. } W_s(t) = [W_s - \beta W_{\text{total}}]^+,
\]

(A-1)

where \(W_s\) and \(W_{\text{total}}\) denote as \(W_{s, \text{max}}\) and \(W_{\text{total}}\), respectively; \(W_{\text{total}} = \sum_s W_s; [.]^+ = \max(., 0)\). The decrease rule in (A-1) is similar to that in (1), hence we need to find \(\theta\) in the increase rule so that \(\theta W_{\text{total}} = W_s\) (because such algorithm as (A-1) will use only the least congested path due to truncation of the decrease rule)\(^1\). By solving for \(\theta\), we obtain

\[
\theta = \left(\frac{4W_s - \beta W_{\text{total}}}{4 - \beta W_{\text{total}}}\right)^3.
\]

(A-2)

Applying the second MPTCP design step into multipath Cubic, we multiply the increase term by \(W_s/W_{\text{total}}\) and the decrease term by \(\alpha(W_s/W_{\text{total}})^3\). The modified multipath algorithm becomes

\[
\text{Inc. } W_s(t) = \min \left\{ \alpha C(t - K)^3 \frac{4W_s - \beta W_{\text{total}}}{4 - \beta K^3} \right\}^+ \frac{W_s^\phi}{W_{\text{total}}^\phi}, \\
\text{Dec. } W_s(t) = W_s - \beta W_s,
\]

(A-3)

where \(0 \leq \phi \leq 4\). We attempt to solve (A-3) (neglecting the right argument of \(\min(.)\)) for \(W_s\) in the same way as in (9), where we have

\[
(4 - \beta^3)^{\phi} \frac{\sum_s W_s^\phi}{\sum_s W_{\text{total}}^\phi} = 16T_s^\phi (\alpha C)^{1/3} W_s^\phi.
\]

\(^1\)We can apply this way into multipath TCP with window increase by \(\theta W_{\text{total}}/\alpha W_{\text{max}}\) (i.e., using single-path increase rule) and window decrease by \(\beta W_{\text{total}}\), then obtain \(\theta = w_s/w_{\text{total}}\). This result matches to (1).

\(^3\)MPTCP’s increase rule uses \(\alpha(w_s/w_{\text{total}})^{1-\phi}\) because each sub-flow becomes a regular TCP once \(\phi = 2\).

Obviously, from the equation (A-4) we cannot find a general solution of \(W_s\). Hence, we cannot solve for a general parameter \(\alpha\) as well. However, given a particular value of \(\phi\) we can determine \(W_s\) and then the \(\alpha\) parameter. Because using \(\sum_s w_s/RTT_s = \max_s \left\{ \frac{W_s^C}{RTT_s} \right\}\) (as described in (13)) in both two design methods, multipath Cubic protocols designed by two different design methods will yield the similar performance, but their load-balancing degrees are dependent on choosing the values of \(k\) and \(\phi\).

Tuan Anh Le received his B.S. degree in Information Technology from Natural Sciences University, Ho Chi Minh City, Vietnam in 2002. During 1997–2003, he worked as research assistance in R&D section of Faculty of Information Technology, Posts and Telecommunications Institute of Technology (PTITT), Ho Chi Minh City Campus, Vietnam. He received M.S. in Computer Engineering from Kyung Hee University, Korea in 2005. He is a lecturer (now on leave) in the Faculty of Information Technology at PTITT, Ho Chi Minh City, Vietnam. He is currently a Ph.D. Candidate in the Department of Computer Engineering at Kyung Hee University, Korea. His research interests are traffic engineering, flow control, congestion control, and resource management.

Rim Haw received his B.S. and M.S. degrees from Department of Computer Engineering at Kyung Hee University, Korea, in 2008 and 2010, respectively. Since March 2010, he is working on his Ph.D. degree in Department of Computer Engineering at Kyung Hee University, Korea. He is a student member of KIPS, and KICS. His research interests include advanced wireless network protocols, mobility management, and P2P Network.

Choong Seon Hong received his B.S. and M.S. degrees in electronic engineering from Kyung Hee University, Seoul, Korea, in 1983, 1985, respectively. In 1988 he joined KT, where he worked on Broadband Networks as a member of the technical staff. From September 1993, he joined Keio University, Japan. He received the PhD degree at Keio University in March 1997. He had worked for the Telecommunications Network Lab, KT as a senior member of technical staff and as a director of the networking research team until August 1999. Since September 1999, he has been working as a professor of the School of Electronics and Information, Kyung Hee University. He has served as a Program Committee Member and an Organizing Committee Member for international conferences such as NOMS, IM, APNOMS, E2EMON, CCNC, ADMSN, ICPP, DIM, WISA, BeN and TINA. His research interests include ad hoc networks, network security and network management. He is a member of IEEE, IPSJ, KIPS, KICS and KIIE.
Sungwon Lee received the Ph.D. degree from Kyung Hee University, Korea. He is a professor of the Computer Engineering Department at Kyung Hee University, Korea. Dr. Lee was a senior engineer of Telecommunications and Networks Division at Samsung Electronics Inc. from 1999 to 2008. He is a editor of the Journal of Korean Institute of Information Scientists and Engineers: Computing Practices and Letters.