(CUBIC White Paper V. 1.0)

Injong Rhee and Lisong Xu
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7634

June 21, 2004

1. CUBIC – New High Speed TCP Protocol

CUBIC is an enhanced version of BIC, one of many TCP variants that improve TCP performance under a specialized case of high-speed networks (high-speed, but long-RTT networks). CUBIC further improves on the TCP friendliness and RTT-fairness of BIC. We are currently conducting experiments to verify the performance of CUBIC.

The following new features are added:

1. A new window growth function – as you can imagine from the name of the protocol, it is a cubic function. It is a generalization of the original BIC growth function. A more scalable, but more TCP friendly version. Also the function is a real-time driven. The BIC growth function (like HSTPC and STCP) is driven by RTT. CUBIC’s growth function grows the window based on the time elapsed from the last congestion epoch. Thus, the protocol can be highly RTT-fair since the window growth rate is determined more by the elapsed time. Since it is less dependent on RTT, the protocol can afford to be more scalable without diminishing RTT-fairness. Also, this feature allows more TCP-friendliness: as the protocol is designed to scale well in long-RTT networks, its growth rate in a short RTT network is in fact slower than TCP which is designed to perform well in short-RTT networks, and its growth rate in a long RTT network is in fact faster than BIC.

2. A new TCP mode – Since the cubic growth function is much slower than TCP when RTT is small and the available bandwidth is small, we make CUBIC as aggressive as TCP under these environment. We accomplish this by emulating TCP to estimate the throughput of TCP. If TCP throughput is larger than CUBIC’s, then CUBIC runs in a “TCP mode” where the window grows at the rate comparable to TCP.

3. Low utilization detection – It adopts the low utilization detection of TCP-LP. Thus, when the network is highly under-utilized due to the departure of some high-speed flows (or any events that disturbs the steady state of the network), it can increase its window more aggressively.

Disclaimer: This document is an informal white paper so that the claims made in the document are given without scientific proof as it focuses on providing insights into the new protocol. We will follow up this paper with a formal research paper.
1.1 BIC growth function
Before delving into CUBIC, let us examine the features of BIC. The main feature of BIC-TCP is its unique window growth function.

![Fig. 1. The CWND curve of BIC](image)

Fig. 1 shows the growth function of BIC. When it gets a packet loss event, BIC reduces its window by a multiplicative factor ($\beta$). The window size just before the reduction is set to the *maximum* and the window size just after the reduction is set to the *minimum*. Then, BIC performs a binary search using these two parameters – by jumping to the “midpoint” between the maximum ($W_{max}$) and the minimum ($W_{min}$). Since packet losses have occurred at $W_{max}$ the window size that the network can currently handle without loss must be somewhere between these two numbers.

However, jumping to the midpoint could be too much increase within one RTT, so if the distance between the midpoint and the current minimum is larger than a fixed constant, called $S_{max}$, BIC increments the current window size by $S_{max}$ (linear increase). If BIC does not get packet losses at the updated window size, that window size becomes the new minimum. If it gets a packet loss, that window size becomes the new maximum. This process continues until the window increment is less than some small constant called $S_{min}$ at which point, the window is set to the current maximum. So the growing function after a window reduction will be most likely to be a linear one followed by a logarithmic one (marked as "additive increase" and "binary search" in the figure).

If the window grows past the maximum, the equilibrium window size must be larger than the current maximum and a new maximum must be found. BIC enters a new phase called "max probing." The growth function during max probing is the inverse of those in binary search and additive increase. It grows exponentially (which is very slow at the beginning) and then linearly. Max probing uses a
window growth function exactly symmetric to those used in binary increase (additive increase and binary search) – only in a different order: it uses the inverse of binary search (which is logarithmic; its reciprocal will be exponential) and then additive increase. Fig.1 shows the growth function during max probing. During max probing, the window grows slowly initially to find the new maximum nearby, and after some time of slow growth, if it does not find the new maximum (i.e., packet losses), then it guesses the new maximum is further away so it switches to a faster increase by switching to additive increase where the window size is incremented by a large fixed increment.

The good performance of BIC comes from the slow increase (see “plateau” in the figure) around $W_{max}$ and linear increase during additive increase and max probing.

1. It gives TCP friendliness. During this time, the window growth is slower than TCP so that the plateau gives time for competing TCP flows to grow their windows.

2. It enhances the stability of the protocol and the network. As the window gets closer to the maximum, this plateau keeps the window near that maximum a long time. This has an effect of reducing window fluctuations. In fact, the window increment just before the maximum is the smallest (this feature contrasts with other protocols where the increment is the largest near the maximum). BIC tends to overshoot over the network capacity by a smaller amount than other protocols. This tends to reduce the impact of the overshoot to other competing flows.

3. The plateau increases the utilization of the network. In steady state, the network is in the full utilization at $W_{max}$. Since BIC keeps the network in the full utilization for a longer time (even with a small buffer at the router), it tends to achieve higher network utilization than other protocols.

4. The linear increase during additive increase and max probing also helps improve the fairness of the protocol. It makes the protocol behave like AIMD. AIMD achieves bounded RTT fairness where the throughput ratio of two competing flows with different RTTs is bounded by the square of the RTT ratio. It also achieves the fair bandwidth share among competing BIC flows with the same RTTs. BIC inherits these features of AIMD.
1.2 New window growth function (cubic)

Although BIC achieves pretty good scalability, fairness, and stability during the current high speed environments, the BIC’s growth function can still be too aggressive for TCP, especially under short RTT or low speed networks. Furthermore, the several different phases of window control adds a lot of complexity in analyzing the protocol. We have been searching for a new window growth function that while retaining most of strengths of BIC (especially, its stability and scalability), simplifies the window control and enhances its TCP friendliness.

In the new release of BIC, we introduce a new window growth function – a cubic function. Figure 2 shows a cubic function whose shape is similar to the BIC window curve in Figure 1. The function grows much slower than binary increase (which is the logarithmic) near the origin (or plateau).

We set the origin to be $W_{\text{max}}$. So after a window reduction, the window grows very fast, but as it gets closer to $W_{\text{max}}$, it slows down its growth. At $W_{\text{max}}$, its increment becomes zero. After that, the window grows slowly as its growth accelerates as it moves away from $W_{\text{max}}$. It has the same plateau as in BIC’s window curve, but its growth rate accelerates much more slowly than BIC’s. This slow growth significantly improves the TCP friendliness of BIC. Furthermore, the function extremely simplifies the window control since there are only one function to use and no multiple phases.

![Graph showing the new window growth function](image)

**Fig. 2.** The new CWND curve of CUBIC
Two issues need attention: RTT-fairness and intra-protocol fairness. Since AIMD is no longer used, we need special mechanisms to ensure these fairness properties. Our strategy for ensuring these properties is to allow the window to grow at a rate dependent on the elapsed time after the last packet loss event. This strategy is also adopted by SQRT [cite] and HTCP [cite]. More specifically, the window size $W$ is determined by the following function.

$$W_{cubic} = C(T - K)^3 + W_{max}$$  \hspace{1cm} (Eqn. 1: the window growth function of CUBIC)

$C$ is a scaling factor, and $K = \sqrt[3]{W_{max} \beta / C}$. $W_{max}$ is the window size just before the last window reduction, $T$ is the elapsed time from the last window reduction, and $\beta$ is the multiplicative decrease factor after a packet loss event.

This function ensures the intra-protocol fairness among the competing flows of the same protocol. To see this, suppose that two flows are competing on the same end-to-end path. The two flows converge to a fair share since they drop by the same multiplicative factor $\beta$ – so a flow with larger $W_{max}$ will reduce more, and the growth function allows the flow with larger $W_{max}$ will increase more slowly – $K$ is larger as $W_{max}$ is larger. The function also offers a good RTT fairness property because the window growth rate is dominated by $T$, the elapsed time. This ensures bounded RTT fairness since any competing flows with different RTT will have the same $T$ after a synchronized packet loss.

To further enhance the fairness and stability, we clamp the window increment to be no more than $S_{max}$ per second. This feature keeps the window to grow linearly when it is far away from $W_{max}$. This makes the growth function very much in line with BIC’s as BIC increases the window additively when the window increment per RTT becomes larger than some constant. The difference is that we ensure this linear increase of the CUBIC window to be real-time dependent.

The real-time increase of the window enormously enhances the TCP friendliness of the protocol. Note that the window growth function of other RTT dependent protocols (TCP being a good example), grows proportionally faster (in real time) in shorter RTT networks whereas CUBIC will grow independently of RTT. Since TCP gets more aggressively while CUBIC being unchanged, short RTT will make CUBIC more friendly to TCP. In short RTT networks, CUBIC’s window growth is slower than TCP.

The cubic function is very TCP friendly as well. Note that when the window size is small, its $W_{max}^3\beta$ is small. This means that $K$ is small and the growth rate of the cubic function is very slow – even slower than TCP’s. Thus, when the BDP of the network is small, CUBIC becomes very friendly to TCP.
We empirically set $C$ to 0.4 and $\beta$ to 0.2. Based on our analysis, these values give reasonable TCP friendliness, fairness, scalability, and convergence speed.

Note that the entire window growth function is described by just one function – in CUBIC, we don’t need different phases of window control like additive increase, binary search, and max probing in BIC. This immensely simplifies our analysis of CUBIC. We omit the formal analytical evaluation of CUBIC here and defer it to a future research paper.

1.3 New TCP mode

We mentioned that the window growth rate of CUBIC can be slower than TCP under short RTT and/or BDP networks. In order to achieve comparable performance to TCP, we allow the window of CUBIC to grow at least at the speed of TCP. We accomplish this by adding a “TCP” mode.

Most high-speed TCP variant protocols have some form of “TCP mode” during which a protocol behaves in the same way as TCP. HSTCP and STCP enter their TCP mode when the window size is less than some small cutoff constant (typically around 30 packets). This cutoff value is determined by the intersection point between the response function of a high speed protocol and that of TCP. Note that a response function of a protocol is the sending packet rate per RTT (or window size) in terms of packet loss rates.

BIC also adopts this approach. However, this approach has a major drawback. TCP may give very good performance even if the window size gets larger than 30. This happens when RTT is very short, but big enough to make its BDP larger than the cutoff value. For instance, 200Mb/s bandwidth and 10 ms RTT gives BDP around 200 but TCP performs very well in this environment, utilizing the full bandwidth. If used in this environment, this approach can make BIC (also HSTCP and STCP) too aggressive for competing TCP flows sometimes.

In fact, our observation is that the regime where TCP performs well is defined by the congestion epoch size (not by the window size) – the period between two consecutive loss events. For instance, if RTT is 1 ms, TCP can grow its window by 1000 per second (with delayed ack, by 500), and this speed should be fast enough to fully utilize the capacity in most of large bandwidth networks (if there are enough buffer space in the bottleneck link). On the other hand, if RTT is 200 ms, then TCP can grow its window by 5 per second. This could be too slow for a large bandwidth network. Controlling the TCP mode using only the window size may not be applicable for all situations. On the other hand, the congestion epoch can tell whether the network environment is good for TCP. The congestion epoch tells the time period that TCP takes to reach the full capacity of the network.

CUBIC solves this problem quite elegantly. Since the window growth rate of CUBIC is independent of RTT, as TCP changes aggressiveness based on RTT,
the real time duration that CUBIC grows more slowly than TCP after the previous epoch is determined by the aggressiveness of TCP. As TCP becomes very aggressive, this “TCP friendly period” becomes longer.

In order to estimate the growth rate of TCP, we emulate TCP after a packet loss event. To be more accurate, we need to compare the throughput of TCP and CUBIC. However, we believe that estimating the window growth rate would be a close approximation to the throughput.

Since CUBIC reduces its window by a factor of $\beta$ after a loss event, the TCP fair rate would be $3((1-\beta)/(1+\beta))$ per RTT. This is because the average sending rate of AIMD is $\frac{1}{\text{RTT}} \sqrt{\frac{\alpha + \beta}{2 - \beta - p}}$, where $\alpha$ is the additive window increment and $p$ is the loss rate. Therefore, TCP will get $\frac{1}{\text{RTT}} \sqrt{\frac{3 - \beta}{2 - \beta - p}}$ since $\alpha=1$ and $\beta=1/2$. To achieve the same average sending rate as TCP with an arbitrary $\beta$, we need $\alpha$ equal to $3((1-\beta)/(1+\beta))$. Since we set $\beta$ to 0.2, the additive increase factor is 0.5. Given this growth rate per RTT, the window size of TCP at time $t$ (after the last epoch) is $W_{\text{tcp}} = W_{\text{max}} \beta + 3((1-\beta)/(1+\beta)) \frac{t}{\text{RTT}}$. If $W_{\text{tcp}}$ is larger than clamped $W_{\text{cubic}}$ (in eqn. 1), then we set the window size to $W_{\text{tcp}}$. Otherwise, clamped $W_{\text{cubic}}$ is the current congestion window size. Note that clamped $W_{\text{cubic}}$ is obtained by clamping $W_{\text{cubic}}$ to grow no faster than $S_{\text{max}}$ per second.

1.4 Low utilization detection

When the network is highly under-utilized, it is unnecessary for the protocol to achieve fairness. At this time, a protocol can increase its transmission rate very aggressively. After the network gets to the full utilization or the “efficiency” line, fairness becomes critical. The network can be at a low utilization if some high-speed flows or there are no high-speed flow users in the network. If we can detect whether the network utilization is low or not, we can increase the scalability of the protocol. This idea is used in TCP-LP (or HSTCP-LP) which increases its rate rapidly using HSTCP when there are not competing traffic. HSTCP-LP reverts to the regular TCP mode if there is cross traffic. Since HSTCP-LP is designed for a low-priority file transfer, this behavior works well for such a goal.

CUBIC uses a similar idea; low utilization triggers CUBIC to increase its window growth rate, and when there are competing traffic, it simply reverts to the regular CUBIC operation (which ensures fairness). During low utilization, CUBIC simply increases the window by the cubic function without the clamping – this increases the window growth rate.
Our low utilization detection is much more conservative than HSTCP-LP. We measure RTT (instead of one-way delay) and check whether a weighted average of RTT is larger than 10% of the difference between $\text{minRTT}$ and $\text{maxRTT}$. $\text{minRTT}$ and $\text{maxRTT}$ are the minimum and maximum RTTs observed in the transmission. This condition is checked only during the max probing phase of CUBIC (only after $\text{CWND} > W_{\text{max}}$). If this condition is true, CUBIC does not apply the linear clamping and the window increases by Eqn. 1 with $C$ set to 0.4. During the max probing phase, the network is more likely to be in non-steady state, meaning that the network state might have changed from the last epoch since the network does not incur packet losses at the same window size where packet losses occur at the end of the last epoch. Our experiment suggests that this technique works fairly well. Especially when competing high-speed flows left the network, CUBIC flows can quickly grab the unused bandwidth.

1.5 CUBIC in action

Figure 3 shows the window curve of CUBIC over the running time. This graph is obtained by running an NS simulation experiment on a dumbbell network configuration with significant background traffic in both directions to remove the phase effect. The bottleneck capacity is 500Mbps and the RTT is set to 100ms. Drop Tail routers are used. Note that the curves have plateaus around $W_{\text{max}}$ which is the window size at the time of the last packet loss event.

Figure 4 shows the same NS simulation but with two flows of CUBIC and two TCP flows; these flows have the same RTT and bottleneck. We observe that the two flows of CUBIC converge to a fair share nicely around 220 seconds. Their $\text{CWND}$ curves are very smooth and do not cause much disturbance to competing TCP flows. In this experiment, the total network utilization is around 98%: the two CUBIC flows take about 78% of the total bandwidth, the two TCP flows take 11%, and the background flows take up around 9%.
Fig. 3: CUBIC window curve (NS simulation in a network with 500Mb/s and 100ms RTT), $C = 0.04, \beta = 0.8$.

Fig. 4: CUBIC window curve with competing flows (NS simulation in a network with 500Mb/s and 100ms RTT), $C = 0.04, \beta = 0.8$. 
2. TCP Friendliness

Below we present some performance results regarding the TCP friendliness of CUBIC and other high-speed TCP variants. We test these protocols in two different environments. First we test them in short-distance (aka, short RTT) networks in which we vary the bottleneck bandwidth while keeping RTT short (less than 20 ms). Second, we test them in long-distance (aka long RTT) networks in which we again vary the bottleneck bandwidth while keeping RTT between 100ms to 200ms.

Below, Tables 1 shows the bandwidth shares of high speed protocols, and long-term TCP over various network conditions. For each simulation run, we run four flows of a high-speed protocol and four flows of regular TCP SACK (or Long-term TCP flows) over the same end-to-end paths for the entire duration of the simulation; their starting times and RTTs are slightly varied to reduce the phase effect. The same amount of background traffic as in the previous experiment is used for both forward and backward directions of the dumbbell setup. For all the experiments, the buffer size of Drop Tail routers is set to 100% of BDP. It shows the total utilization of the link capacity, the utilization of a high speed TCP variant, and that of TCP. The remainder of the link capacity (total utilization – the total utilization of TCP and high speed flows) is used by the background traffic on the forward link.

**Experiment 1: TCP Friendliness in Short RTT Networks**

(Experiment script available in the BIC web site)

We run four TCP long-term flows along with four flows of a high-speed protocol. We test four high speed TCP variants: CUBIC, HSTCP, Scalable TCP, and HTCP. We set RTT of the flows to be around 15 ms and vary the bottleneck bandwidth from 20 Mbps to 2.5 Gbps. Table 1 summarizes the results. It shows, for each run, the total utilization of the bottleneck link capacity (for the second half of the simulation of 300 seconds), the link utilization by the four flows of a high-speed TCP variant, that of the long-term TCP flows (the TCP SACK flows running for the same duration as the high-speed flows), and that of the (forward) background TCP traffic. Figure 8 shows the utilization ratio of the long-term TCP flows over the high-speed flows (or TCP friendly ratio) measured from these runs.

Over 20 Mbps and 100 Mbps networks, all protocols show good ratios. But as the bottleneck bandwidth increases, the ratios for HSTCP, STCP, and HTCP drop dramatically indicating unfair use of bandwidth with respect to TCP. Under these environments, regular TCP can still use the full bandwidth as evidenced by the CUBIC experiment where long-term TCP running on the same end-to-end path with CUBIC uses about the same amount of bandwidth as CUBIC while maintaining high total utilization. Scalable TCP shows the worst TCP friendliness in these tests followed by HSTCP and HTCP. We note that HTCP tends to show significantly lower network utilization than other protocols.
Table 1: The summary of TCP friendly simulation runs over short-RTT networks.

<table>
<thead>
<tr>
<th>Speed</th>
<th>CUBIC Total Util. (%)</th>
<th>CUBIC Long-term TCP (%)</th>
<th>HSTCP Total Util. (%)</th>
<th>HSTCP Long-term TCP (%)</th>
<th>STCP Total Util. (%)</th>
<th>STCP Long-term TCP (%)</th>
<th>HTCP Total Util. (%)</th>
<th>HTCP Long-term TCP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20Mbps</td>
<td>95</td>
<td>30</td>
<td>25</td>
<td>90.5</td>
<td>20.64</td>
<td>23.51</td>
<td>96.41</td>
<td>48.34</td>
</tr>
<tr>
<td>100Mbps</td>
<td>97</td>
<td>31</td>
<td>26</td>
<td>97.71</td>
<td>42.16</td>
<td>34.39</td>
<td>98.64</td>
<td>57.96</td>
</tr>
<tr>
<td>300Mbps</td>
<td>95</td>
<td>38</td>
<td>32</td>
<td>95.83</td>
<td>42.16</td>
<td>34.39</td>
<td>98.64</td>
<td>57.96</td>
</tr>
<tr>
<td>500Mbps</td>
<td>96</td>
<td>45</td>
<td>33</td>
<td>94.61</td>
<td>42.16</td>
<td>34.39</td>
<td>98.64</td>
<td>57.96</td>
</tr>
<tr>
<td>1Gbps</td>
<td>97</td>
<td>54</td>
<td>29</td>
<td>95.83</td>
<td>42.16</td>
<td>34.39</td>
<td>98.64</td>
<td>57.96</td>
</tr>
</tbody>
</table>

Fig. 8: TCP Friendly Ratio for Short RTT networks

Experiment 2: TCP Friendliness in Long RTT Networks
(Simulation script available in the BIC web site)

Although the TCP mode improves the TCP friendliness of the protocol, it does so mostly for short RTT situations. When the BDP is very large with long RTT, the aggressiveness of the window growth function (more specifically, the congestion epoch length) has more decisive effect on the TCP friendliness. As the epoch gets longer, it gives more time for TCP flows to grow their windows.

An important feature of BIC and CUBIC is that it keeps the epoch fairly long without losing scalability and network utilization. Generally, in AIMD, a longer congestion epoch means slower increase (or a smaller additive factor). However, this would reduce the scalability of the protocol, and also the network would be underutilized for a long time until the window becomes fully open (Note that it is true only if the multiplicative decrease factor is large; but we cannot keep the multiplicative factor too small since that implies much slower convergence to the equilibrium).

CUBIC solves this problem quite elegantly. It increases the window to (or its vicinity of) $W_{max}$ very quickly and then holds the window there for a long time.
This keeps the scalability of the protocol high, while keeping the epoch long – of course, it also keeps the utilization high. This feature is unique both in BIC and CUBIC.

Below, Table 2 shows the summary of our experiments over long-RTT networks. We vary the bottleneck bandwidth from 20Mbps to 1Gbps and also RTT from 100ms to 200ms. Figure 9 shows the utilization ratio of long-term TCP over high-speed TCP variants. Over 20 Mbps, all the high speed protocols show reasonable friendliness to TCP. As the bandwidth gets larger than 20 Mbps, the ratio drops quite rapidly. Overall, CUBIC shows a good friendly ratio comparable to HSTCP and HTCP. HSTCP also shows pretty good friendliness to TCP over high-bandwidth environments. STCP again shows the worst TCP friendliness. The utilization of HTCP has improved from the short-RTT cases, but is still the worst.

<table>
<thead>
<tr>
<th>Bottle Size</th>
<th>CUBIC Total Util. (%)</th>
<th>HSTCP Total Util. (%)</th>
<th>STCP Total Util. (%)</th>
<th>HTCP Total Util. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20Mbps, 100ms</td>
<td>98 15 12</td>
<td>97 13.8 16</td>
<td>98 16 12</td>
<td>98 13 11</td>
</tr>
<tr>
<td>20Mbps, 200ms</td>
<td>98 21 9</td>
<td>98 13 9.9</td>
<td>98 15 11.6</td>
<td>97.8 22.8 7.7</td>
</tr>
<tr>
<td>100Mbps, 100ms</td>
<td>96 28 15</td>
<td>96 45 12</td>
<td>99 57 10</td>
<td>94.5 42.6 12</td>
</tr>
<tr>
<td>100Mbps, 200ms</td>
<td>99 59 9</td>
<td>98 56 11</td>
<td>99 87 7.5</td>
<td>95 47.6 6</td>
</tr>
<tr>
<td>300Mbps, 100ms</td>
<td>99 70 12</td>
<td>97.8 70.8 9.5</td>
<td>99.9 82 4.7</td>
<td>94 45 6</td>
</tr>
<tr>
<td>300Mbps, 200ms</td>
<td>100 77 7</td>
<td>99.6 74 11</td>
<td>99.5 82 4</td>
<td>93.7 72 6</td>
</tr>
<tr>
<td>500Mbps, 100ms</td>
<td>99 77 9</td>
<td>99 78 8.5</td>
<td>99.9 85 4.5</td>
<td>95 73 9</td>
</tr>
<tr>
<td>500Mbps, 200ms</td>
<td>100 85 5</td>
<td>99.8 81 9</td>
<td>100 88.6 2</td>
<td>94 78.5 5</td>
</tr>
<tr>
<td>1Gbps, 100ms</td>
<td>99 85 7</td>
<td>97.7 80.9 6.7</td>
<td>100 90.7 2</td>
<td>93 78 6.6</td>
</tr>
<tr>
<td>1Gbps, 200ms</td>
<td>100 89 4</td>
<td>100 87 6.9</td>
<td>99.9 90 2.7</td>
<td>97 85.8 4.6</td>
</tr>
</tbody>
</table>

Table 2: The summary of TCP friendly simulation runs over short-RTT networks.
4. Scalability

We measure the scalability of CUBIC by estimating its response function (which is the relation between the packet loss rate $p$ and its sending rate – $cwnd$). It is very easy to analyze the response function of CUBIC (unlike BIC) since its growth function consists of only one well-defined function. Here we omit the details on the analysis, but present the result. Our analysis shows that the response function of CUBIC is very close to AIMD when the window size is large. This is because when the window size is large, the window growth is determined by the clamped increment (which is a linear increase). AIMD has the following response function:

$$R = \sqrt{\frac{\alpha + \beta}{2(1 - \beta)} \frac{1}{p}}$$

Since CUBIC is a real-time protocol, its growth rate (per RTT) varies based on RTT (while remaining constant in real time). For $S_{max} = 160$ per second, for 200ms RTT, it grows 32 packets per RTT and for 100ms RTT, 16 packets per RTT.

The typical bit error rate of high-speed networks is around 1e-15. Given that a packet size is 1e-5, the packet loss rate is around 1e-10. At this loss rate, CUBIC can have average window size around 489,000 packets. This is equivalent to approximately 30Gbps. We are currently investigating increasing the clamping (or even removing it). With the cubic function alone (without the linear clamping), we can achieve around 1.2 Tera bps around 1e-10.

Under low utilization, CUBIC can increase its window by the cubic function alone. Thus, it can grow its window by more than 10,000 packets within 30 seconds, and more than 80,000 within one minute. This would make short-term flows (the network transfer with a small amount of data to transfer) utilize the full bandwidth quickly under low network utilization.

4. Stability

Experiment 3: Long-RTT high-speed flows vs. short-RTT TCP flows.
(Simulation script available in the BIC web site)

We run four flows of a high-speed TCP variant over a long-RTT network path (~200ms) and four flows of long-term TCP-SACK flows over a short-RTT path (~20ms). These two paths share a bottleneck link of 2.5Gbps. In this experiment, to see how stable different protocols become as the buffer space of the bottleneck router varied, we vary the buffer space of the bottleneck router from
20% to 5% of the BDP of the long-RTT path. The background TCP traffic is added to the bottleneck link. Figure 10 illustrate our simulation setup (slightly modified for clarity). The actual simulation setup can be found in the script above.

Below, we show the CWND graphs from experiments with 5% and 20% buffer in Figures 14 and 15.

By inspecting the raw data, we can tell that HTCP has some stability issues (this needs to be confirmed with the original authors of HTCP). The oscillation of CWND for STCP and HSTCP gets worse as the buffer size reduces. The high oscillation occurs over various time scales.

There is no well-defined measure of stability. Earlier work focuses more on the smoothness of transmission rate variations. However, the real measure of stability as defined in control theory is whether the transmission rate of a protocol can converge to a “fair” transmission rate. The fairness is defined by the utility function of the protocol. Typically, congestion control for the Internet follows min-max fairness. However this notion of stability only tells whether a protocol can converge to that fair share or not, but since protocols run in various unpredictable network environments (so some of essential assumptions used in the traditional congestion control theory do not hold), it does not say “decisively” whether a protocol is stable or not.

We are looking for a relative measure that can tell whether a protocol is more stable than others. We are currently investigating techniques to measure the average fairness index (by Jain) at various time scales and compare those of various protocols. We will report the results later as they become available.

For a less satisfactory measure, we present the coefficients of variance (CoV) of throughput in Figure 16. We can see that CUBIC shows superior stability over the other protocol (measured by CoV).
5. Conclusion.

In this document, we provide some insights into the protocol of CUBIC and its behavior. We also provide some details on simulation experiment and their results. Our analysis and experimental results indicate that CUBIC works reasonably well. More thorough analysis and performance evaluations are required to confirm the claimed benefits of CUBIC. We are continuing our effort to achieve that. Currently, BIC is being evaluated at quite a few different research and commercial sites.
Fig 15. The CWND graphs from the stability tests (20% buffer)

Fig 16. Coefficients of variance of throughput from the stability test.