BicTCP Implemenation in Linux Kernels

Yee-Ting Li*and Doug Leith†
Hamilton Institute, NUI Maynooth

15th February 2004

Abstract

This document describes a bug in BicTCP, which has been implemented into the 2.6.6+ kernels.

1 Background

TCP is a protocol that regulates network congestion and best effort fairness. Many attribute the success of the internet to the congestion avoidance algorithms in the original TCP. However, TCP has performance deficiencies [R03] and many New-TCP protocols have also been developed, amongst which HSTCP, ScalableTCP, HTCP and FAST have been proposed and discussed amongst the scientific network research communities. Presently, only BicTCP [XHR] has been implemented into a standard release of the Linux 2.6 kernels [Linux][1]. In fact, the BicTCP algorithm is actually switched on by default in 2.6.8 kernels and above [2.6.8].

Given that TCP plays such an important role in the current internet, thorough analysis and testing of all New-TCP protocols should be conducted and well understood as global widespread usage of such New-TCP protocols may lead to undesirable effects on network fairness and stability.

We are currently in the process of developing generic tests along these lines. As part of this work, we have identified an error in the implementation of BicTCP in the standard release of the Linux kernels. The objective of this note is to describe this error and outline how it can be fixed.

2 BicTCP Implementation

BicTCP is based on the binary increase of the fundamental TCP variable *cwnd* that controls the rate at which data is to sent into the network.

The BicTCP algorithm defines a threshold target value $last\_max\_cwnd$ for *cwnd*. This threshold is fundamental to the BicTCP algorithm as BicTCP has differing data send rates depending on the relation between

*Yee-Ting.Li@may.ie
†Doug.Leith@may.ie

[1] TCP Westwood and TCP Vega have also been implemented but are not considered here.

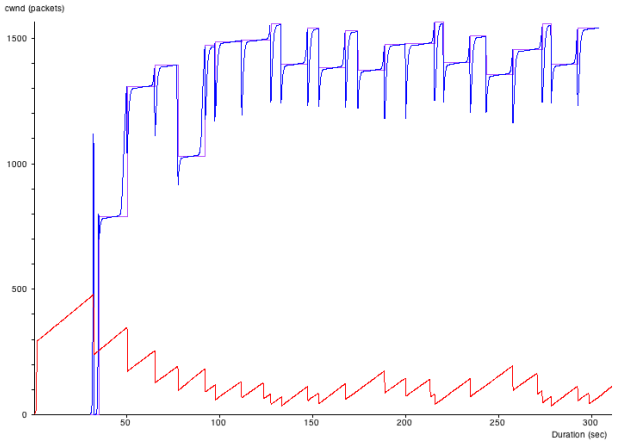

Figure 1: *cwnd* histories of BicTCP version 1.1 (blue) competing against Standard TCP (red). *last_max_cwnd* of BicTCP version 1.1 shown in purple with 100Mbit/sec common bottleneck and 162msec RTT experienced by both flows. Bottleneck queuesize is 270 packets. Details of the experimental set-up is given in Appendix A.

cwnd and *last_max_cwnd*; when *cwnd* is smaller than *last_max_cwnd*, BicTCP regulates traffic into the network through an aggressive algorithm until *cwnd* nears the target value at which point it steadies off its sending rate [XHR]. Upon congestion, BicTCP backs off *cwnd* according to $cwnd \leftarrow \beta \times cwnd$, where β is a constant parameter of the algorithm.

A *cwnd* history from a correct implementation of this algorithm is shown in Figure 1. It is from the Linux 2.4.25 patch which is publicly available from the authors of BicTCP [BicD]. This will been referred to as BicTCP version 1.1 in this note. As described in [XHR], following congestion the target value *last_max_cwnd* is set to halfway between the *cwnd* just before congestion is detected and that after congestion. The defaults for BicTCP version 1.1 define that the back-off factor β should be 0.8. Therefore, the *last_max_cwnd* value after congestion should be set to 0.9, i.e. $(1 - \frac{1-0.8}{2})$.

In the various Linux implementations of BicTCP, this value is set *only* in the function *tcp_recalc_ssthresh* which calculates appropriate TCP parameters upon packet loss detection. In the BicTCP version 1.1, this is implemented as;

```
if ((tp->snd_cwnd < tp->bictcp_last_max_cwnd) &&
            (sysctl_bictcp_fast_convergence))
      /* Wmax and fast convergence */
      tp->bictcp_last_max_cwnd
            = (tp->snd_cwnd
                  * (1024+sysctl_bictcp_1024times_beta))>>11;
```

where `sysctl_bictcp_1024times_beta` is set to `819` (representing 0.8 when bit shifted 10 times right). Therefore, the above code represents

the necessary calculation of $last\_max\_cwnd = (1 - \frac{1-0.8}{2}) \times cwnd$ of the BicTCP proposal when β is 0.8.

3 Linux 2.6.6+ Kernel Implementation

The actual implementation of BicTCP in the standard release Linux 2.6 kernel is referred to here as the Linux 2.6.6+ implementation. This is also applicable for all current Linux kernel versions including 2.6.6 and above[2].

The Linux 2.6.6+ implementation of BicTCP appears to be based on the INFOCOM paper [XHR]. Differences from the version implemented in BicTCP version 1.1 include:

- The back-off factor β is 0.875 in Linux 2.6.6+ rather than 0.8 in BicTCP version 1.1
- An early slow-start exit (*ssthresh* 100) is present in BicTCP version 1.1
- A 'low utilisation detection mode' based on a difference in ratio of the measured minimum and maximum latencies experienced by the TCP flow is present in BicTCP version 1.1 but is absent from Linux 2.6.6+
- Provision to account for delayed acking is present in the BicTCP v1.1 release for Linux 2.4.25 but not in the Linux 2.6.6+ version.

The Linux 2.6.6+ implementations incorporates a reduction in the number of *sysctl*'s through the introduction of a series of static `#ifdef`'s. The corresponding define for the `sysctl_bictcp_1024times_beta` in the Linux 2.6.6+ kernels is `BICTCP_1_OVER_BETA` which is set to `8` (corresponding to a β of 0.875).

The kernel code (again in the function *recalc_recalc_ssthresh*) in the Linux 2.6.6+ implementation is as follows:

```
if (sysctl_tcp_bic_fast_convergence &&
            tp->snd_cwnd < tp->bictcp.last_max_cwnd)
      tp->bictcp.last_max_cwnd
            = (tp->snd_cwnd * (2*BICTCP_1_OVER_BETA-1))
                  / (BICTCP_1_OVER_BETA/2);
```

This results in a *last_max_cwnd* value which is approximately 4 times ($\frac{15}{4}$) larger than the *cwnd* value at loss. The effects of this code can be see in Figure 2. The large dips in the *cwnd* in Figure 2 (left) are due to various issues in the Linux networking kernel as outlined in [DL04]. An implementation of these various bug-fixes with BicTCP is shown in Figure 2 (right) which is zoomed to make the structure of the linear increase more apparent.

[2] Analysis shows that the differences between the initial release of BicTCP on Linux 2.6.6 and that of newer kernels (up to 2.6.11-rc4) show no difference in the BicTCP code - the exception being an implementation to 'keep track of last time congestion was computed, and recompute if *cwnd* changes or every 1/32 of a second' [2.6.8]. These changes do not affect the calculation of *last_max_cwnd*.

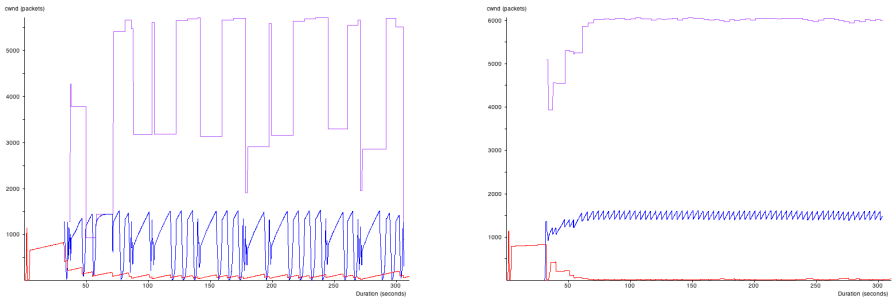

Figure 2: *cwnd* (blue) and *last_max_cwnd* (purple) histories of BicTCP Linux 2.6.6+ competing against Standard TCP (red) with 100Mbit/sec common bottleneck and 162msec RTT experienced by both flows. Bottleneck queuesize is 270 packets. Plot on left is Linux 2.6.10 and the plot on the right is Linux 2.6.10 with [DL04] modifications.

The effect of setting *last_max_cwnd* to $\frac{15}{4} \times cwnd$ is that the TCP connection will spend more time in BicTCP's aggressive increase regime where the *cwnd* is increased by 32 packets each round-trip time (as opposed to an increase of 1 packet each round-trip time in Standard TCP).

4 Linux Kernel Fix

As the value of *last_max_cwnd* specified in [XHR] is halfway between the value of *cwnd* before and after congestion, the above bug can be easily fixed the correction:

```
if (sysctl_tcp_bic_fast_convergence &&
            tp->snd_cwnd < tp->bictcp.last_max_cwnd)
      tp->bictcp.last_max_cwnd
            = (tp->snd_cwnd * (2*BICTCP_1_OVER_BETA-1))
                  / (BICTCP_1_OVER_BETA*2);
```

which will set the value of *last_max_cwnd* to $\frac{15}{16} \times cwnd = 0.9375 \times cwnd$ which is the correct value for a the back-off factor of $\frac{1}{8}$. However, this would not function correctly for values of `BICTCP_1_OVER_BETA` other than 8, and therefore we recommend that it is modified to:

```
if (sysctl_tcp_bic_fast_convergence &&
            tp->snd_cwnd < tp->bictcp.last_max_cwnd)
      tp->bictcp.last_max_cwnd
            = tp->snd_cwnd
                  - ( tp->snd_cwnd / (BICTCP_1_OVER_BETA*2) );
```

The impact of the above fix is shown in Figure 3. They clearly show the appropriate settings of *last_max_cwnd* similar to the design of BicTCP version 1.1. This patch is available at [Bic-Patch] and is compatible will all versions of Linux 2.6.6 and above.

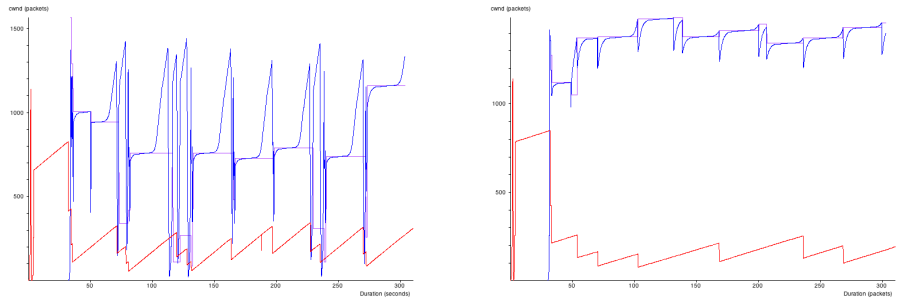

Figure 3: *cwnd* and *last_max_cwnd* (purple) histories of Linux 2.6.6+ BicTCP with patch competing against Standard TCP (red). 100Mbit/sec common bottleneck and 162msec RTT experienced by both flows. Bottleneck queuesize is 270 packets. Results shown for Linux 2.6.10 patched (left) and Linux 2.6.10 patched with [DL04] modifications (right).

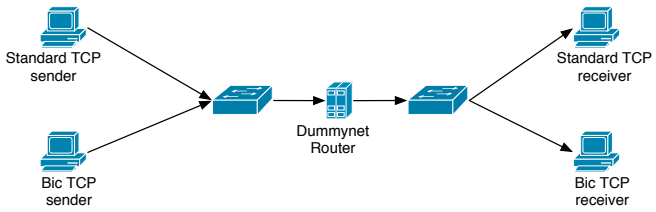

Figure 4: Experimental set-up of all tests. The BicTCP kernel is configured with differing kernel versions as shown in Table 2.

A Experimental Setup

All tests performed in this paper were conducted through a dummynet [R98] router connected via two gigE switches as shown in Figure 4. The dummynet router was configured with 100Mbit/sec bottleneck and an round trip latency of 162ms. The bottleneck queuesize was set to 20% of the Bandwidth Delay Product (405,000 Bytes or 270 packets).

Two flows were injected into the testbed. The first is a standard TCP (SACK) flow using a 2.6.6 kernel with [DL04] performance modifications. The second flow was injected into the network 30 seconds after the initial flow and was configured with various BicTCP kernel stacks as defined in Table 2. All sending and receiving machines have identical hardware configurations as shown in Table 1 and were connected to the GigE switches at 1Gb/sec.

Traces of *cwnd* and *last_max_cwnd* were gathered through web100 [Web100] by overloading a `GAUGE32` variable (in the above cases `MinSsthresh`).

	Description
CPU	Intel(R) Xeon(TM) CPU 2.80GHz
Memory	256Mbytes
Motherboard	Dell PowerEdge 1600SC
Network Interface Card	Intel Corp. 82540EM Gigabit Ethernet Controller (rev 02)
NIC Driver Version	Standard Linux distribution versions with default configurations

Table 1: Experimental kernel versions used in experimental tests.

Name	Kernel Version	Web100 Version [Web100]
BicTCP Version 1.1 [BicD]	2.4.25	2.3.5
BicTCP Linux 2.6.6+ [Linux]	2.6.10	2.5.2
BicTCP Linux Patched [Bic-Patch]	2.6.10	2.5.2

Table 2: Experimental kernel versions used in experimental tests.

B Protocol Stacks

Tests were performed on the kernel stacks as shown in Table 2. Note that the BicTCP differences between Linux 2.6 distributions are minimal and do not affect the decrease parameter nor calculation of $last\_max\_cwnd$.

The Web100 implementations provide per flow TCP logging facilities to allow debugging of TCP state variables.

References

[Linux] *The Linux Kernel Archives*. http://www.kernel.org/.

[2.6.8] *Kernel.org - 2.6.8 Changelog*, http://www.kernel.org/pub/linux/kernel/v2.6/ChangeLog-2.6.8.

[XHR] L. Xu, K. Harfoush, and I. Rhee, *Binary Increase Congestion Control for Fast, Long Distance Networks*. Infocomm 2004.

[Bic] I. Rhee, *BIC TCP - Publications*, http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/.

[BicD] I. Rhee, *BIC TCP - Downloads*, http://www.csc.ncsu.edu:8080/faculty/rhee/export/bitcp/index_files/Page546.htm

[DL04] D. J. Leith, *Linux TCP Implementation Issues in High-Speed Networks*, http://www.hamilton.ie/net/LinuxHighSpeed.pdf.

[R03] S. Ravot, *TCP transfers over high latencybandwidth networks & Grid DT*, Protocols For Long Distance Networks 2003.

[R98] L. Rizzo, *Dummynet: A Simple Approach to the Evaluation of Network Protocols*, ACM Computer Communication Review, vol. 27, no. 1, pp. 31-41, 1998.

[Web100] M. Mathis, J. Heffner, R. Reddy, R. Raghunarayan & J. Saperia, *Web100 Project*, http://www.web100.org/.

[Bic-Patch] Y. Li, *Linux 2.6.6+ BicTCP last_max_cwnd Patch*, http://www.hamilton.ie/net/.