Analysis of TCP BIC Congestion Control Implementation

Wu Hua
School of Computer Science & Engineering
Southeast University
Nanjing, China
hwu@njnet.edu.cn

Gong Jian
School of Computer Science & Engineering
Southeast University
Nanjing, China
jgong@njnet.edu.cn

***Abstract*—TCP has the congestion control algorithm, the behavior of TCP flows can reflect the network status. It is possible to know the network status through the flow statistics such as NetFlow. But different TCP congestion control algorithms take different methods to the same network state. The fully understanding of congestion control mechanism is the precondition of inspecting network status through TCP flow behavior. Although there have a long-term studies on TCP Reno, little has done to those new congestion control algorithms which are implemented for the current fast long-distance networks. The implementation of the default TCP congestion control algorithm in Linux kernel 2.6.8 is studied in this paper. The original algorithm is introduced firstly, then the implementation of BIC in Linux kernel is analyzed through the traces. The results of this paper provide the basis of inspecting network status through TCP BIC flow behavior.**

Keywords- network status;congestion control;TCP BIC; flow behavior

I. INTRODUCTION

Many network applications use TCP as their transfer protocol. The characteristics of TCP flows are affected not only by the network transmission capacity and the background traffic, but also the congestion control algorithm. TCP congestion control algorithm is the most important factor. The sender adjusts the data sending mode according to the network congestion status. There are more than ten kinds of TCP congestion control algorithms, which kind of congestion control algorithm is used depend on the protocol stack implementation of the operating system. Even the same kind of congestion control algorithm, there are different implementation in different operating system. For example, RFC2581 defines TCP's four main congestion control algorithms, even such a standard recommendation, there are different optimization method when they are implemented. In order to infer the network status through TCP flows' behavior, the first thing is to consider the impact of the special congestion control algorithm used by the TCP flow.

There are two main kinds of operating system on the Internet, Microsoft operating systems and Linux operating system.

Microsoft operating systems use the improved version of TCP originated from RFC2581. Because Linux system is fully open, under the principle of the GNU GPL General Public License, everyone can be free access, dissemination, and even modify the source code. Many new technologies are timely reflected in Linux system. TCP congestion control algorithm is a prime example. Linux system allows the user to choose TCP congestion control algorithm, the system provides a number of congestion control modules for user to select, such as BIC(Binary Increase Congestion control for TCP) [1],CUBIC(Binary Increase Congestion control for TCP V2.0) [2],High Speed TCP, HTCP(Hamilton TCP), HYBLA (TCP HYBLA), RENO(TCP NewReno), VEGAS, WESTWOOD. But normally users do not configure the default congestion control module. BIC is the default TCP congestion control algorithm in Linux kernel 2.6.8, in Linux Kernel 2.6.19, BIC was improved into CBUIC.

This paper analyzed the implementation and optimization of the TCP BIC congestion control algorithm in Linux kernel 2.6.18-4-686. The result is helpful for inferring the network congestion status through TCP flows' characteristic. The second section of this paper gives the related work, the third section introduced the original BIC algorithm. In section 4, through the analysis to TCP flow traces, the implementation of TCP BIC in the Linux kernel 2.6.18-4-686 is analyzed. The last part is the conclusion.

II. RELATED WORKS

There are a lot of analysis on the TCP Reno algorithm [3][4], In paper[3], Mathis gave the model between TCP Reno flow transfer performance and round-trip delay, packet loss and other factors. Padhye gave the theoretical model between TCP flow throughput and delay, packet loss. But these theoretical results are based on low-bandwidth links, and no longer fit for the current high-bandwidth network environment.

Cwnd(congestion window) is a important factor that determine the transfer performance of TCP flow. If cwnd increased to a large number, the transfer performance will be high, but the possibility of packet loss will be increased which will trigger the congestion process. If cwnd is too small, the throughput of the TCP flow will be very low. TCP congestion control algorithm tries to find the most suitable cwnd for the current network load status.

TCP Reno congestion control algorithm can not fully utilize the bandwidth in high-speed Internet. In order to find the value, after entering the congestion avoidance phase, the sender slowly increases cwnd by one packet per RTT time, if packet loss or timeout happens, cwnd will be decreased by half, and then slowly rise until the next loss or timeout event

978-1-4673-0719-2/12/$26.00 ©2012 IEEE

happens. In most of the time, cwnd will be much more less than the suitable value.

To find a suitable cwnd is a search process. TCP Reno uses the traverse search algorithm, the search time is long. From the perspective of research performance, linear traversal algorithm is the lowest algorithm. Since one RTT has to go through in the course of one research step, the more steps of the search process need, the more bandwidth is wasted. Especially in the high bandwidth, long delay network circumstance. Since cwnd is large when the packet loss happens, according to TCP Reno, cwnd will decreased to half the current value, a long time will be needed to increase cwnd to the maximum number. Thus the traditional TCP Reno congestion control algorithm is not suitable to be applied on the current high speed long delay network.

For these reasons, a lot of new TCP congestion control algorithms were put forward., such as BIC,CUBIC,High Speed TCP, HTCP, HYBLA,RENO), VEGAS, WESTWOOD, High Speed TCP, Scalable TCP, XCP and etc. Since Linux is open source system, a lot of new algorithms is implemented on Linux, a wide variety of new congestion control algorithms are integrated on it. It provides the configuration interface which allows the users to choose which kind of TCP congestion algorithms is used. Normally the users will not modify the default configuration, so study on the behavior of the default congestion algorithms is useful in understanding the behavior of network.

BIC and CUBIC are the default TCP congestion control algorithms of Linux kernel. They are designed to make full use of available resources in the high-speed long-distance path, while maintaining fairness and stability of TCP.

BIC algorithm is proposed in [1] in 2004. The a series of articles carried out a comparative assessment of experimental [5] [6] [7] on BIC and other high-speed TCP congestion control algorithms, but there are little analysis about the implementation of TCP BIC in the kernel. The reason is the time of the algorithm is relatively short compared to TCP Reno, more important reason is that more freedom in the optimization algorithm in the Linux system, the analysis of the algorithm is difficult.

In this paper, the behavior characteristics of the BIC is analyzed, because BIC and Reno is quite different. CUBIC is an improved version of BIC, it has a good improvement on the TCP fairness, the basic behavioral characteristics are very similar with BIC.

III. BASIC ALGORITHM OF TCP BIC

BIC congestion control is complicated, and the algorithms can be found in [1].Parameters used in the algorithm are defined as follows:

low_window: low window threshold, if cwnd is greater than this threshold, then use BIC, if cwnd is less or equal to this threshold, using ordinary TCP congestion avoidance algorithms of TCP Reno;

Smax: the maximum increasement of window during a round;

Smin: the minimum decreasement of window during a round;

β : multiplier factor to reduce window;

default_max_win: the default largest window.

The above parameters are constant in the algorithm implementation, there are some variables are defined below:

max_win: the maximum window;

min_win: the minimum window;

pre_win: the maximum window value before the current maximum window is set;

target_win: the middle point between max_win and min_win.

The following outlines the four main control algorithm of BIC.

(1)Binary search increase

Packet loss indicating that the best window value should be smaller than cwnd , BIC set current cwnd as max_win, the cwnd after the packet loss is multiplied by β and set to be min_win.Then BIC began to perform binary search between max_win and min_win. But if max_win is relatively large, to adjust the window to the midpoint of max_win and min_win may be a larger increasement and it will cause jitter in the transmission. BIC took another two reference values Smax and Smin. If the difference between the midpoint target_win and the current cwnd value is greater than Smax, then cwnd only have a growth Smax. If there is no packet loss, reset min_win as current window, if there is a packet loss, to set max_win as the current window.. The process continues until the difference between target_win and current cwnd is less than Smin. This process is a two statuses combination of search and increase.

(2) Slow Start

The Slow start here is not the same with the slow start in Reno. If cwnd is more than max_win, assume that the current state is stable, the window in the steady state should be larger than the current window, it is needed to search out new max_win, the search method is similar to the slow start. Fistly,max_win is set to be a very large value, the current cwnd is set to be min_win, BIC then take a similar slow start strategy, after each RTT window is set to be cwnd +1, cwnd +2, cwnd +4 ... cwnd + Smax, until it equals to Smax, then enter to the binary search increase status.

(3) Fast convergence

In order to ensure different TCP flows have the same convergence rate, when packet loss happens and cwnd has a decrease trend, take the mid-point of max_win and min_win as the target cwnd, if cwnd has a increase trend, take the maximum window as the target cwnd

(4) Reno

Because the above algorithm has no superiority on a small bandwidth path, BIC has the provisions that if the current window is less than low_window , use TCP Reno congestion control algorithm.

The reason that TCP BIC has good performance is cwnd growth slowly near max_win, that is cwnd increases the smallest particle size within the vicinity of the maximum window, the window can last a long time in a place near the maximum allowable window. While TCP Reno increases the largest particle size within the vicinity of the maximum window. Compare TCP BIC with TCP Reno, TCP BIC reduces the jitter of the cwnd, increases the stability of the TCP transport. Besides, in this relatively stable phase, beacuse TCP BIC keep cwnd of the TCP stream to make full

use of the network bandwidth, it can get the higher resource utilization.

IV. TCP BIC IMPLEMENTATION ANALYSIS

Through the observation of the actual trace, TCP BIC behaviors far from the above algorithm. This is because the implementation does not adhere to the original algorithm, the developer has done a lot of improvements. We analyzed many traces to find out the implementation of TCP BIC.

A. Data analysis methods

This paper focus on the typical operating system kernel Linux the kernel 2.6.18-4-686.

In order to observe the actual performance of TCP BIC in different network paths, we need to analyze the trace data across different backbone. We set measurement points at Education Network, Telecom and China Netcom. Every measurement point uses the same operating system platform, and Apache 2.2.3 Web server. Because we need to study the bulk data transfer performance of TCP BIC, for each Web server, we put 20MB and 30MB of data files for download .

When the client requests to download, use tcpdump to sniffer the packet trace on the server side, and according to the IP address and port, filtering out the special TCP stream packet trace. By analyzing the packets trace to obtain the characteristics of TCP BIC.

We analyzed about 50 samples. The method of analysis is to observe the relationship between cwnd and packet loss events, combined with the existing documentation, summed up the TCP BIC implementation, as well as the behavioral characteristics of different congestion conditions.

B. Analysis results

In the Linux TCP BIC implementation, compared to the original algorithms, some changes were done. These changes had a greater influence on the behavior of TCP BIC Flow. When we study on the characteristics of TCP BIC, we must consider the impact of these optimization algorithms. In this section, we first study optimizations in the implementation of TCP BIC, then analysis packet loss of TCP BIC flows based on these implementation details.

In the specific implementation of the BIC in Linux kernel 2.6.18-4-686, after the packet loss, the window transform into four statuses: slow start, additive increase, binary search and maximum detection. Following we will describe the different selection criteria, as well as the various congestion control strategy in the four statuses. In the implementation, there have different parameters and variables. The new parameters are as defined as follows:

BICTCP_B: constant which give the range of binary increase, usually is 4;

Smooth_part: constant which is set to slow down the binary incrase,usually is 20;

linux_min_win: variable that when packet loss happens cwnd will be set;

$\beta\_win : \beta\_win = \max\_win \times \beta$.

In [1], when a packet loss happens, $\min\_win = \max\_win \times \beta$, but in the implementation, it is not set in this way. The new minimum window is called linux_min_win to distinguish with min_win in [1].

Assume cwnd is max_win when a packet loss happens, then after the packet loss set cwnd as linux_min_win. If cwnd has a downward trend, then the target cwnd can be calculated by $target\_win = \max\_win \times ((1+\beta)/2)$, β is a constant factor, the default value is 0.8. If cwnd has a upward trend, then target_win equals to max_win. Following gives the boundary conditions and the algorithms in the system implementation:

(1) After a packet loss happens, cwnd is set to be linux_min_win, if $linux\_\min\_win < \beta\_win$, use slow start as the congestion control algorithm, then every time it receives an ACK packet, it will sends three data packets until $cwnd = \beta\_win$ .At this time, if $target\_win - cwnd > BICTCP\_B$, the status will enter into step 2,if $|target\_win - cwnd| < BICTCP\_B$, enter into step 3.

(2) If $target\_win - cwnd > BICTCP\_B$, it will enter into additive increase, in this process, for every RTT time, cwnd will be increased by $\frac{target\_win - cwnd}{BICTCP\_B}$ packets, that is, after $\frac{cwnd * BICTCP\_B}{target\_win}$ packets have been sent, cwnd will increased by one packet.

(3) If $|target\_win - cwnd| < BICTCP\_B$, it will enter into binary increase, for every RTT time, cwnd will be increased by $\frac{BICTCP\_B}{Smooth\_part}$ packets, that is, after $\frac{cwnd * Smooth\_part}{BICTCP\_B}$ packets have been sent, cwnd will increased by 1 packets. Considering the default value BICTCP_B equals to 4, Smooth_part equals to 20, that is in binary increase, after five times cwnd packets have been sent, cwnd is increased by one. Binary increase is a very slow process, reflects the basic idea of TCP BIC algorithm, assuming that the target window is where the available bandwidth of the network path may be fully utilized without packet loss, the slowly increase in this range is to ensure the TCP flow can maximize use of available bandwidth without occurrence of a packet loss. This process continues until cwnd is more than target_win plus BICTCP_B; but in this status if cwnd is less than 15, the congestion control algorithm is Reno, that is for every ACK packets, cwnd will increased by 1/cwnd packets, after two time cwnd packets have been sent, cwnd will be increased by one packet.

(4) If $cwnd - target\_win > BICTCP\_B$, enter into maximum detection. In this status, for every RTT time, cwnd will be increased by

$\frac{cwnd - target\_win}{BICTCP\_B - 1}$ packets, that is, after $\frac{cwnd(BICTCP\_B - 1)}{cwnd - target\_win}$ packets have been sent, cwnd will increased by one packet.

In the implementation of the Linux kernel, when cwnd is greater than 15 if a packet loss happens, due to the above algorithm, if using the default algorithm parameters, and no packet loss encountered in the recovery process, for each occurrence of the slow start, additive increase, binary additive and maximum detection, the start point is fixed. Analysis on the traces also confirmed this. The following figure shows variation of cwnd and transition of state when pakcet loss happens at different cwnd. (algorithm constants are set to take the system default value = 0.8, Smax = 16, Smin = 4, low_window = 14).

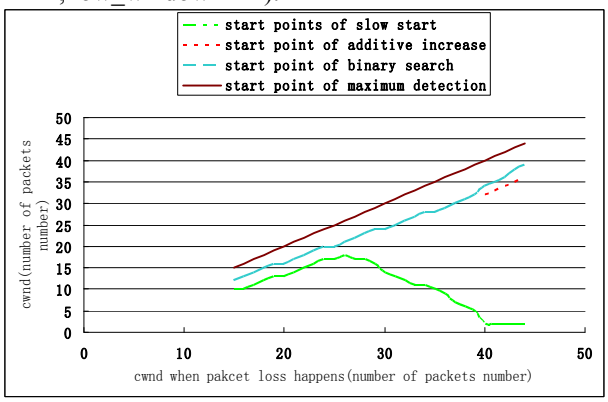

Figure 1. Variation of cwnd and transition of state when pakcet loss happens at different cwnd.

In all the experiments, the parameters are default value. The abscissa cwnd is the congestion window when packet loss occurs. Because BIC is enabled when cwnd is greater than 14, in the figure the curve begin from the abscissa equals to 15. After a packet loss, the window decrease to linux_min_win:the slow start point, the larger cwnd is when packet loss happens, the lower cwnd will go. When the window decrease to the slow stat point, the window will increase according to the slow start algorithm until cwnd goes to the situation in (2), if in the additive increase process, no packets was lost, cwnd will go to the situation in (3) and change to the binary search process. Sometimes, situation can go straightly from (1) to (3). It can be seen from the figure, if we use the default control parameters, additive increase occurs only when the packet loss window is greater than 39. In the binary search process, because the rate of increase is very slow, it requires a long time. If there are no packet loss in the binary search process, it will goes into the maximum detection process, in this status, the algorithm used to increase cwnd is similar with additive increase process.

According to the above analysis, there are two trends when packet loss happens: if the packet loss happens when cwnd is less than 39, the change process is binary search, maximum detection, there will be no additive increase; if the packet loss happens when cwnd is more than 39, after the slow start, it will goes into additive increase, binary search, and maximum detection.

V. CONCLUSION

Due to the existence of congestion control mechanisms, the behavior of TCP flow is a reflection of the network status. So analysis of the network status through the TCP behavior must have a full understanding of the TCP congestion control algorithm. Although there has been a long-term research on TCP Reno, but there are little research on some of the new algorithm for high-bandwidth long-delay networks. This paper studies the default TCP congestion control algorithm in Linux kernel 2.6.8, its implementation of the Linux kernel were analyzed by analysis of the actual traces. The analysis result was given through a large number of data, the study provides a basis for obtaining network status through BIC flow behavior.

ACKNOWLEDGMENT

This work was sponsored by he National Grand Fundamental Research 973 program of China under Grant No.2009CB320505, the National of Nature Science Foundation of China under Grant No. 60973123, the Technology Support Program (Industry) of Jiangsu Province under Grant No.BE2011173, Jiangsu Province Key Laboratory of Network and Information Security under Grants No.BM2003201 and Key Laboratory of Computer Network and Information Integration (Southeast University) of Ministry of Education.

REFERENCES

[1] Xu Lisong, Harfoush Khaled, Rhee Injong. Binary increase congestion control (BIC) for fast long-distance networks. IEEE INFOCOM. 2004, Vol 4: pp.2514-2524

[2] Sangtae Ha, Injong Rhee, Lisong Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variant. ACM SIGOPS Operating Systems Review. 2008 42(5): pp.64-74.

[3] MATHIS M, SEMSKE J, MAHDAVI J. The macroscopic behavior of the TCP congestion avoidance algorithm. Computer Communication Review, 1997, 27(3):pp.67-82I.

[4] Padhye J, Firoiu V, Towsley D, Kurose J. TCP Throughput:A Simple Model and its Empirical Validation. Computer Communication Review, 1998, 28(4): pp.303-314

[5] Escheikh M., Barkaoui K. Performance analysis of high-speed TCP protocols BIC and cubic with AQM in lossy networks. International Conference on Communication Systems, Networks, and Applications, CSNA 2007. pp.31-35.

[6] Mbarek R, Hahar Ben Othman M, Nasri S. Fairness of high-speed TCP protocols with different flow capacities. Journal of Networks,,2009 4(3): pp. 163-169

[7] Nabeshima M, Yata K. Performance evaluation and comparison of transport protocols for fast long-distance networks: IEICE Transactions on Communications. 2006,Vol E89-B, Vol4: pp.1273-1283