ACCESS METHODS FOR SOCIAL INCLUSION[*]

Sagar Sumit

February 2019

1 Introduction

Poor sections of the society who lack an internet connection should be able to share the internet facility belonging to a richer community nearby for free. However, the real challenge is to make this privilege without denting the bandwidth allocated for the owner of the connection. We analyze existing congestion control mechanisms, and we see fit in a new experimental protocol called as LEDBAT (Low Extra Delay Background Transport), a delay based congestion control mechanism, which is extensively used in torrent applications[1] to solve problems with similar goals. Our work implements the latest version of LEDBAT[2] and develops a Linux kernel module for the same. We have implemented the latest version in a well-known network simulator NS2, by performing extensive experiments. In order to ensure only a genuine poor user participates, we have fingerprinted the LEDBAT traffic based on utilization and classified flows to detect non-LEDBAT users.

2 Problem Definition

The implementation setup diagram on the following page describes the base model that we have considered to simulate and analyze to derive conclusions about our problem. Given this setup, the problem is to identify the most suitable mechanism by studying the existing access methods for creating a system of sharing of network resources and the impact of implementing the access methods.

[*]This is an excerpt of my undergraduate thesis submitted in May 2012 under the guidance of Dr. R. Leela Velusamy, Dept. of Computer Science and Engineering, NIT Trichy.

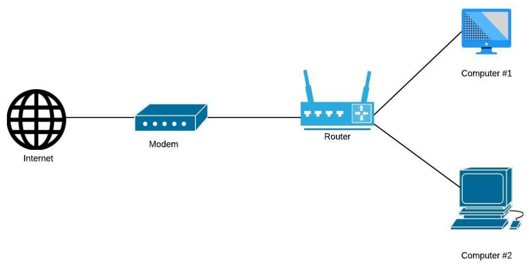


Figure 1: Implementation setup

The main actors in figure 1 are as follows:

- A rich user (Computer #1), who owns the internet connection.
- A poor user (Computer #2), who shares the connection of the rich user.
- A central router.
- A modem, which connects the users to the internet.

The goals of the experiment are as follows:

1. To utilize the free bandwidth of the rich user effectively for the poor user.
2. To make sure that the Quality of Service (QoS) parameters of the rich user are not affected substantially by the presence of another flow.

3 Setup and Analysis

Analyzing congestion control algorithms is not new. A comparative analysis of TCP congestion control algorithms has been done by S. Patel et al[3], which involves active queue management techniques. Window-based congestion control algorithms have been tested by S. Jin et al[4], which tend to behave *TCP-like*. Cui-Qing Yang[5] provides a taxonomy for existing algorithms, treats the internet as a huge distributed system where congestion control schemes are implemented in every node. While many of these actually appreciate the working of the algorithms themselves, they do not clearly examine the scenario required for the problem. Hence, an analysis is required, which considers a specific traffic pattern, with which the choice of the congestion control mechanism can be concluded.

This experiment proves our very choice to use LEDBAT as a congestion control mechanism to solve the given problem. We analyze a set of well established TCP control algorithms by simulating them for a scenario described below, and hence infer that LEDBAT is the best suited mechanism.

3.1 Connection Setup

Two nodes, namely $Node_1$ and $Node_2$, are connected to a bottleneck link, whose link capacity is 512 Kilobytes (Kb) with a 100 milliseconds (ms) latency. $Node_1$ runs TCP-Cubic[6] congestion control algorithm with an FTP source attached to it. $Node_2$ will run TCP-Sack[7], TCP-BIC, TCP-Reno, TCP-NewReno[8], TCP-Vegas[9], and finally TCP-LEDBAT. The individual link capacities are 1 Megabytes per second (Mbps) each with a latency of 10ms. $Node_1$ and $Node_2$, which can be compared to rich user and poor user respectively in figure 1, are connected to $Node_3$, which basically acts like a router that is start of the bottleneck link and forwards the packets to $Node_4$, which is the sink. $Node_3$, which acts as a router, has a *droptail* queue with a fixed queue length. The packet size considered is 512 bytes, and the throughput is calculated for each flow every 10 seconds, and the graph is plotted as Time in seconds (s) vs Throughput in bytes per second (bps).

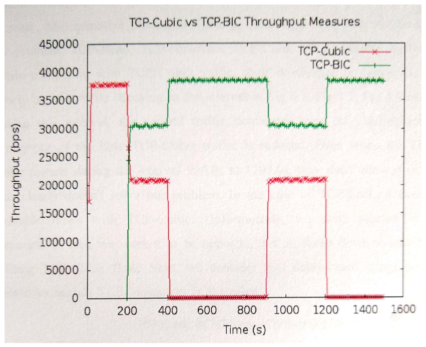
3.2 Traffic Setup

The entire duration of simulation is for 1500s. The FTP_0 traffic, originating from $Node_1$, runs from 5.0s to 400.0s and again restarts from 900.0s and runs up to 1200.0s. The FTP_1 traffic, originating from $Node_2$, runs from 200.0s till the end of the simulation. The reason we chose such a traffic pattern is that it correctly predicts how an existing traffic (from $Node_1$) reacts in the presence of new traffic (from $Node_2$). This is done by analyzing what happens at 200.0s when FTP_1 begins, and continues till 400.0s. Moreover, this pattern also predicts how the FTP_1 traffic exploits the additional bandwidth when it becomes becomes available after the FTP_0 traffic is removed between 400.0s and 900.0s.

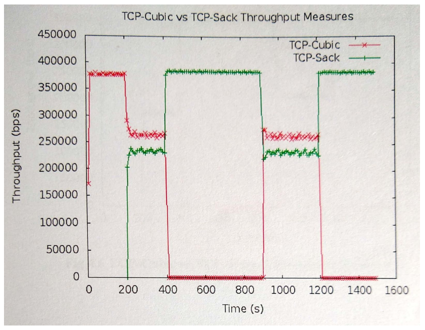
Paraphrasing to our problem, basically, we can analyze what congestion control algorithm to use to test how the rich user's traffic is affected when a poor user's traffic is introduced (in the interval from 200.0s to 400.0s), and how effective the bandwidth utilization is when the poor user alone utilizes the extra bandwidth (in the interval from 400.0s to 900.0s). Overall, this experiment should tell us which algorithm is the most effective one to be considered for the given problem.

3.3 Analysis

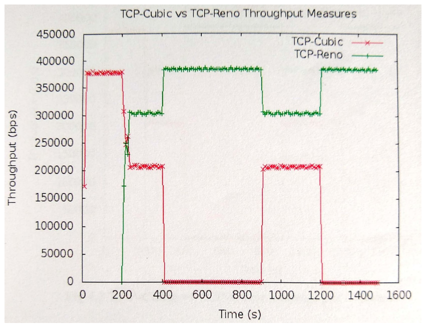
Throughput of algorithms depicted in figures 2(a) through 2(d) are **acknowledgement** based congestion control mechanisms. Based on the number of outstanding packets unacknowledged, they increase or decrease the congestion window. TCP-BIC, TCP-Reno and TCP-NewReno are aggressive by themselves. While competing with TCP-Cubic traffic, their dominance is visible in their respective plots in the interval from 200.0s to 400.0s. Even when the TCP-Cubic traffic restarts during the interval from 900.0s to 1200.0s, they do not allow it to progress. In the case of TCP-Sack, it does provide equitable flows with TCP-Cubic.



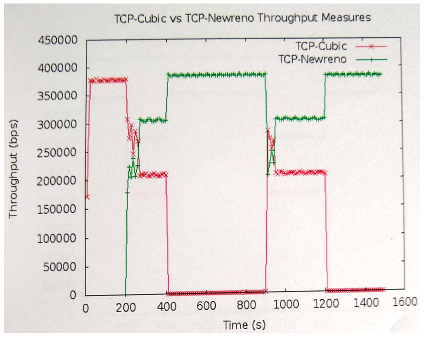
(a) TCP-Cubic vs TCP-BIC



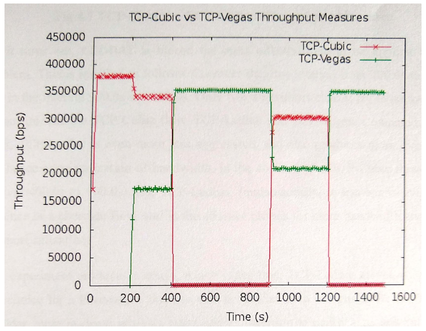
(b) TCP-Cubic vs TCP-Sack



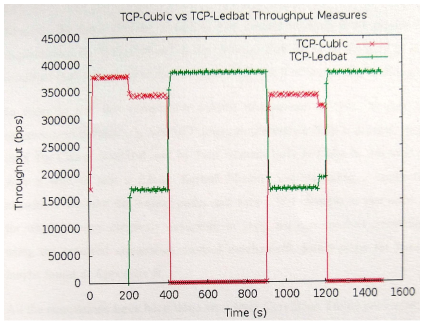
(c) TCP-Cubic vs TCP-Reno



(d) TCP-Cubic vs TCP-NewReno



(e) TCP-Cubic vs TCP-Vegas



(f) TCP-Cubic vs TCP-LEDBAT

Figure 2: Comparison of throughput of TCP congestion control algorithms

Next, we consider two **delay** based congestion control algorithms, namely TCP-Vegas and TCP-LEDBAT. As it turns out, LEDBAT is the most effective mechanism for our problem. Consider the time interval from 200.0s to 400.0s and the interval from 900.0s to 1200.0s in figures 2(e) and 2(f). Both, TCP-Vegas and TCP-LEDBAT, try to back off and not impede the TCP-Cubic flow. However, TCP-LEDBAT has a higher throughput, and hence more utilization of bandwidth in the absence of TCP-Cubic flow (in the interval from 400.0s to 900.0s).

4 Conclusion

Among different congestion control mechanisms, TCP-LEDBAT is not only less aggressive in the presence of an alternate TCP-Cubic flow, but also utilizes the bandwidth efficiently in the absence of the alternate flow. Thus, TCP-LEDBAT can be considered as a choice of network protocol for the poor user so that it satisfies both the goals in our problem definition. Future scope of the work may include further analysis of LEDBAT protocol's variation to packet size as we kept this parameter fixed at 512 bytes. We could also increase the load by increasing the number of nodes in the setup and see how the throughput of the protocol varies. Overall, the outcome of the experiment has provided a concrete proof and a general working model of the LEDBAT protocol in a real-world setup of one poor user sharing network resources per rich user.

5 References

1. Dario Rossi, Claudio Testa, Silvio Valenti, Luca Muscariello, LEDBAT: The new BitTorrent Congestion Control Protocol, International Conference on Computer Communications and Networks, August 2010.

2. S. Shalunov, G. Hazel, J. Iyengar, M. Kuehlewind, Low Extra Delay Background Transport (LEDBAT), Internet Engineering Task Force RFC (https://tools.ietf.org/html/draft-ietf-ledbat-congestion-09), October 2011.

3. Sanjeev Patel, P. K. Gupta, Arjun Garg, Prateek Mehrotra, Manish Chhabra, Comparative Analysis of Congestion Control Algorithms Using NS-2, International Journal of Computer Science Issues, Volume: 8, Issue: 5, September 2011.

4. Shudong Jin, Liang Guo, Ibrahim Matta, Azer Bestavros, A spectrum of TCP-friendly window-based congestion control algorithms, IEEE/ACM Transactions on Networking, Volume: 11 , Issue: 3, June 2003.

5. Cui-Qing Yang, A taxonomy for congestion control algorithms in packet switching networks, IEEE Network, Volume: 9 , Issue: 4 , Jul/Aug 1995.

6. Sangtae Ha, Injong Rhee, Lisong Xu, CUBIC: A New TCP-friendly High-Speed TCP Variant, ACM SIGOPS Operating Systems Review 42(5):64-74, July 2008

7. M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, TCP Selective Acknowledgement options, Internet Engineering Task Force (https://tools.ietf.org/html/rfc2018), October 1996.

8. S. Floyd, T. Henderson, A. Gurtov, The NewReno Modification to TCP's Fast Recovery Algorithm, Internet Engineering Task Force RFC (https://tools.ietf.org/html/rfc3782), April 2004.

9. L.S. Brakmo, L.L. Peterson, TCP Vegas: end to end congestion avoidance on a global Internet, IEEE Journal on Selected Areas in Communications, Volume: 13 , Issue: 8 , October 1995.