Computer Networks 56 (2012) 2677–2689

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Data uploading time estimation for CUBIC TCP in long distance networks

Nobuyoshi Tomita [a], Shahrokh Valaee [b,*]

[a] Sony Corporation, Japan
[b] The Department of Electrical and Computer Engineering, University of Toronto, Canada

A R T I C L E   I N F O

Article history:
Received 20 September 2010
Received in revised form 11 April 2012
Accepted 18 April 2012
Available online 28 April 2012

Keywords:
TCP CUBIC
Bandwidth estimation

A B S T R A C T

CUBIC is a TCP-friendly algorithm that uses a cubic curve, independent of the round-trip time, to rapidly recover from a packet loss. New releases of Linux use CUBIC for the TCP protocol. In this paper, we show that if the socket buffer size of a sender TCP is small compared with the bandwidth-delay product, Linux TCP window size drops to almost zero every time a packet loss occurs. Using this fact, we estimate data uploading time in long distance networks with packet loss. Also we discuss the improvement of the uploading time by increasing cumulative socket buffer size in two ways: large buffer size or parallel connections.

1. Introduction

New models of digital still cameras and digital camcorders are equipped with Wi-Fi and B3G/4G capability that allows them to upload photos and movies directly onto servers or databases. Users of these products, for instance, may want to upload movies and photos captured by their digital camcorders from overseas hotel rooms to video sharing websites. In this case, data transfer in long distance networks with large delay is usually required. Furthermore, some networks may have comparatively large delay even for domestic transfers in heavy traffic condition. This paper investigates the following question: *Under the given round-trip delay and loss conditions, how long does it take to upload a certain volume of data stored in a camera with the new Linux kernel?*

File transfer protocols use the reliable service of TCP (Transmission Control Protocol) to transfer data among hosts. The presence of TCP at the transport layer may reduce the effective bandwidth even if the underlying network has enough capacity. This is due to the fact that the throughput of TCP is related to the round-trip time (RTT) of the network. That is, a sender TCP is allowed to transmit new packets only after it has received acknowledgements (ACKs) for previously transmitted packets.

To improve the TCP throughput, the application layer can increase the "cumulative" socket buffer size of the sender TCP by increasing the TCP *socket buffer* size, or increasing the *number* of TCP connections. This approach uses the conventional TCP, such as TCP Reno/New Reno, and only adjusts its parameters, or uses multiple instances of TCP to give an aggregated service to the application. The two approaches have different effects on the throughput in lossy networks. That is, a packet loss can reduce the throughput more severely if the socket buffer size is large and might have a moderate or small impact if multiple TCP connections with small socket buffer size are used. It is generally understood that multiple TCP connections maintain a higher throughput in reaction to a packet loss since the loss only reduces the throughput of a single TCP connection sparing the rest of the TCP connections to operate at their favorable rate [1].

There has been a host of proposals on enhancing throughput using multiple parallel TCP connections. An enhanced FTP protocol, called GridFTP, has been proposed in [2] to reduce the file downloading time in grid environment. In [1], the concept of parallel TCP is explored and a performance model is suggested to study the improvement

* Corresponding author.
E-mail addresses: Nobuyoshi.Tomita@jp.sony.com (N. Tomita), valaee@comm.utoronto.ca (S. Valaee).

1389-1286/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comnet.2012.04.010

achieved by the application of multiple parallel TCP connections. The *MultiTCP* system [3] is a receiver-driven multimedia streaming system that uses multiple TCP connections. Multihomed hosts can also benefit from running multiple parallel TCPs on different networks; Hsieh and Sivakumar [4] proposes an end-to-end transport protocol *pTCP*, which allows connections to enjoy the aggregate bandwidth obtained through the multiple paths. All these works use TCP Reno/NewReno and adjust the parameters of the protocol to achieve enhanced throughput.

The second approach to throughput enhancement is to replace the conventional TCP with a new transport algorithm with appropriate congestion control mechanism. A new breed of protocols, such as HSTCP [5], Scalable TCP [6], H-TCP [7], XCP [8], TCP Vegas [9] and TCP Veno [10], has been proposed in the literature to produce transport layers with enhanced throughput. Since these protocols can co-exist with the conventional TCP protocol in a network, a major question is how "friendly" these protocols are to the TCP protocol. BIC [11] and CUBIC [12] are two new TCP-friendly protocols that can realize high throughput even in networks with large RTT and are used as default protocols in the new versions of Linux in particular after Linux 2.6.8 and Linux 2.6.19, respectively. Detailed Linux TCP implementation is explained in [13]. Also [14] surveys various congestion control methods used in Linux implementations, and shows their strengths and weaknesses.

In this paper, we study the congestion control mechanism of CUBIC TCP, which is used in the latest versions of Linux. We closely examine the code of Linux 2.6.27 to understand the congestion control algorithm of this operation system. In particular, we find a lower bound for the total data uploading time for given file size, RTT, and probability of packet loss.

There are some works in the literature that relate to our research. An initial experimental evaluation of CUBIC TCP algorithm is presented in [15], which highlights a number of practical issues. Miras et al. [16] presents experimental results and compares New Reno, HSTCP, H-TCP and CUBIC with different views of fairness such as TCP-friendliness, RTT-fairness, intra- and inter-protocol fairness. In [17], active measurement results of New Reno, Vegas, Veno and CUBIC in a commercial IEEE 802.16/WiMAX-based network reveal several issues such as limited bandwidth for TCP, high RTT and jitter, and unfairness during bidirectional transfers.

This paper is organized as follows. In the next section Linux TCP behavior is explained in detail. We estimate the uploading time in Section 3. In Section 4, we add an adjustment for the estimates with the actual Linux TCP implementation. Experimental results in Section 5 show the validity of the estimates and we conclude the paper in Section 6.

2. Behavior of Linux TCP

Although the main purpose of this paper is to estimate data uploading time in long distance networks with packet loss, in this section, we study the behavior of the CUBIC algorithm in Linux TCP. We will show in this paper that TCP and its congestion control has a direct impact on the uploading time. One of the important findings of this paper is that increasing the socket buffer size does not necessarily lead to a shorter uploading period. This finding is against the conventional wisdom that the rate of TCP is controlled by the congestion window, and that increasing the socket buffer size results in a smaller data uploading time. We need to review the details of the TCP congestion control to realize how the total uploading rate is affected by the socket buffer size and the round trip time (RTT).

There are two important buffers in the networking stack: one is the "socket buffer," which is used to store data in the kernel, the other is the "application buffer," which is used in the application layer. When the application calls the `send()` function with a pointer to the application buffer, the sender TCP copies data from the application buffer to its socket buffer and then uses a congestion control algorithm to transmit the data. We assume that the socket buffer size of the sender TCP is smaller than that of the receiver TCP. This is a reasonable assumption because the default value of the socket buffer size for the sender TCP in Linux 2.6.27 is 16 Kbytes (16,384 bytes), whereas it is 85 Kbytes (87,380 bytes) for the receiver TCP.

Fig. 1a shows the typical throughput curve for CUBIC TCP with a single packet loss. The vertical axis shows the "congestion window" size, which is the number of packets allowed to be transmitted without any acknowledgement. The maximum size of the congestion window depends on the socket buffer size of the sender TCP. If the buffer size

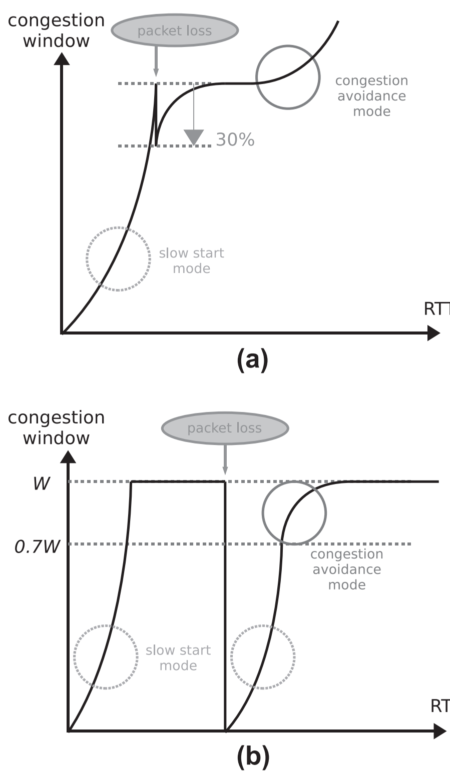

Fig. 1. Throughput curve for CUBIC TCP with a packet loss when the congestion window size is (a) less than *W* and (b) equal to *W*.