Computer Communications 33 (2010) 1013–1029

Contents lists available at ScienceDirect

Computer Communications

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

CODE TCP: A competitive delay-based TCP

Yi-Cheng Chan [a,*], Chia-Liang Lin [a], Chia-Tai Chan [b], Cheng-Yuan Ho [c]

[a] Department of Computer Science and Information Engineering, National Changhua University of Education, No. 2, Shi-Da Road, Changhua City 500, Taiwan
[b] Institute of Biomedical Engineering, National Yang-Ming University, No. 155, Sec. 2, Linong Street, Taipei City 112, Taiwan
[c] Department of Computer Science, National Chiao Tung University, No. 1001 Ta Hsueh Road, Hsinchu City 300, Taiwan

ARTICLE INFO

Article history:
Received 19 September 2008
Received in revised form 9 January 2010
Accepted 11 January 2010
Available online 18 January 2010

Keywords:
Congestion control
TCP Reno
TCP Vegas
Fairness
Transport layer protocol

ABSTRACT

TCP Vegas is a well-known delay-based congestion control mechanism. Studies have indicated that TCP Vegas outperforms TCP Reno in many aspects. However, Reno currently remains the most widely deployed TCP variant in the Internet. This is mainly because of the incompatibility of Vegas with Reno. The performance of Vegas is generally mediocre in environments where it coexists with Reno. Hence, there exists no incentive for operating systems to adopt Vegas as the default transport layer protocol. In this study, we propose a new variant of Vegas called COmpetitive DElay-based TCP (CODE TCP). This variant is compatible with Reno and it can obtain a fair share of network resources. CODE is a sender-sided modification and hence it can be implemented solely at the end host. Simulations and experiments confirm that CODE has better fairness characteristics in network environments in which it coexists with Reno while retaining the good features of Vegas.

1. Introduction

Transmission Control Protocol (TCP) is a connection-oriented, end-to-end, and reliable protocol. Nowadays, a majority of Internet traffic is carried by TCP. Therefore, the behavior of TCP is tightly coupled with the overall Internet performance. To improve network efficiency, many TCP variants have been proposed. Two of these variants are noteworthy. One is Reno [1], which has been widely deployed in the Internet; the other is Vegas [2], which claims to have a throughput that is 37–71% greater than that of Reno.

TCP Vegas is a delay-based congestion control mechanism. Unlike TCP Reno which uses a binary congestion signal, packet loss, to adjust its window size, Vegas adopts a more fine-grained signal, queuing delay, to avoid congestion. Vegas can detect network congestion in the early stage and successfully prevent periodic packet loss that usually occurs in Reno. Delay-based congestion control schemes have attracted considerable attention because of their innovative control policy [3–10].

As compared to Reno, Vegas generally performs better with respect to overall network utilization [2], stability [11,12], fairness [11,12], throughput, packet loss [2], and burstiness [13] in homogeneous environments. Vegas also outperforms TCP Newreno [14]. However, studies have shown that when Reno and Vegas connections coexist in the same network, Reno obtains a greater amount of bandwidth as compared to Vegas [12,15,16]. Consequently, although Vegas has been available for a few years, it has not been adopted widely because of its perceived incompatibility with Reno.

To deal with the fairness problem, we propose a new mechanism called COmpetitive DElay-based TCP (CODE TCP), which is a variant of Vegas. Most of the operations in CODE TCP are similar to those in Vegas except that the two thresholds α and β are adaptive to the state of the network. When CODE senses the occurrence of network congestion and it does not consider itself to be responsible for the congestion, it begins to increase α and β instead of reducing its own rate. This makes CODE behave more similarly to Reno. Therefore, if the competing source is Reno, CODE reacts against its aggressiveness and its performance does not decrease. Conversely, if the competing source is Vegas, after a transition period, CODE recovers to a stable status as a Vegas source by reducing α and β. Changing the values of α and β instead of directly altering the value of its congestion window (*CWND*) should allow CODE to preserve the properties of Vegas in reaching an operating point.

The remainder of this paper is organized as follows. We describe the related work in Section 2. Section 3 presents a detailed description of the proposed mechanism, CODE TCP. Section 4 presents and discusses the results of both NS-2 simulations and experiments on a Linux platform. Finally, the conclusions are presented in Section 5.

2. Related work

To ensure network efficiency, TCP controls its sending rate based on feedback from the network. In order to control the

* Corresponding author. Tel.: +886 4 7232105x7044; fax: +886 4 7211284.
E-mail addresses: ycchan@cc.ncue.edu.tw (Y.-C. Chan), 94612005@mail.ncue.edu.tw (C.-L. Lin), ctchan@bme.ym.edu.tw (C.-T. Chan), cyho@csie.nctu.edu.tw (C.-Y. Ho).

0140-3664/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2010.01.007

sending rate, TCP estimates the available network bandwidth via a bandwidth estimation scheme. In TCP Reno, packet losses are used to detect network congestion while in TCP Vegas, the queuing delay is used to estimate the network condition. In this section, we present a summary of these two congestion control mechanisms and explain why they are incompatible. Then two algorithms, NewVegas [17] and Vegas-A [18], that attempt to solve the fairness problem between Vegas and Reno are described.

2.1. TCP reno

TCP Reno uses a congestion window (*CWND*) to control the amount of data transmitted in a round-trip Time (*RTT*) and a maximum window (*MWND*) that is set by the receiver to limit the maximum value of *CWND*. The congestion control scheme of Reno can be divided into three phases: slow-start, congestion avoidance, and fast retransmission and fast recovery. In the interest of conciseness, we only describe the congestion avoidance phase since it is most closely related to our work. The descriptions of the other two phases can be found in [1].

Since the window size in the slow-start phase expands exponentially, packets sent at this increasing speed would quickly lead to network congestion. To avoid this, the congestion avoidance phase begins when *CWND* exceeds a preset slow-start threshold (*ssthresh*). In this phase, *CWND* is incremented by $1/CWND$ packet every time an *ACK* is received in order to make *CWND* grow linearly. This process continues until a packet loss is detected; subsequently the scheme switches to the fast retransmission and fast recovery phase.

2.2. TCP vegas

TCP Vegas adopts a more sophisticated bandwidth estimation scheme that attempts to avoid congestion rather than react to it. It uses the measured *RTT* to accurately calculate the number of data packets that a source can send. Vegas features three improvements as compared to TCP Reno: (1) a modified slow-start mechanism, (2) an improved congestion avoidance mechanism, and (3) a new retransmission mechanism. Its window adjustment algorithm also consists of three phases. The *CWND* is updated based on the currently executing phase. Fig. 1 shows the state transition diagram of TCP Vegas. A connection begins with the slow-start phase. The window-adjustment phase transition is attributable to specific events, as depicted along the edges.

2.2.1. Slow-start

During the slow-start phase, Vegas allows a connection to quickly ramp up to the available bandwidth. However, in order to detect and avoid congestion during this phase, Vegas doubles its *CWND* only every other *RTT*. In between, the *CWND* remains fixed so that a valid comparison of the *Expected* and *Actual* sending rates can be made. Vegas estimates a suitable amount of extra data to be maintained in the network pipe and controls the *CWND* accordingly. It records *RTTs* and sets the *BaseRTT* to the minimum *RTT* value measured. The amount of extra data ($\varDelta$) is estimated as follows:

$$\varDelta = (Expected - Actual) \times BaseRTT, \tag{1}$$

where *Expected* rate is the current *CWND* size divided by the *BaseRTT*, and *Actual* rate is the *CWND* divided by the newly measured smoothed *RTT*.

This detection mechanism is applied during the slow-start phase to decide when to switch the phase. If the estimated amount of extra data is greater than the threshold γ, Vegas reduces its *CWND* by one-eighth and transitions from the slow-start phase to the congestion avoidance phase.

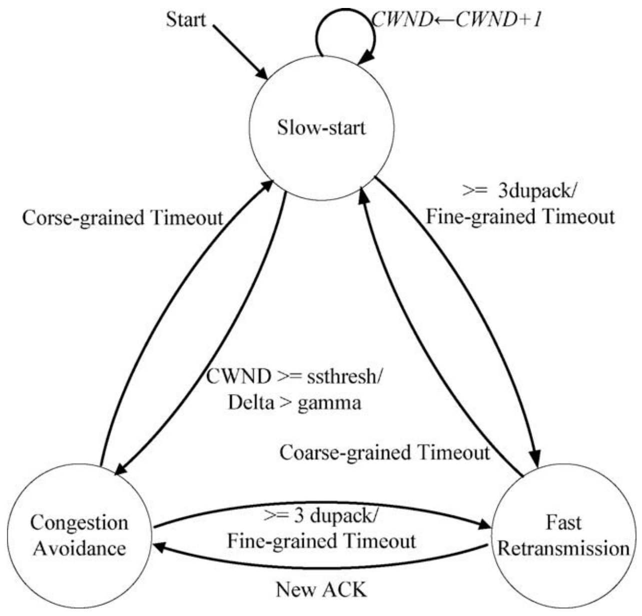

Fig. 1. State transition diagram of Vegas.

2.2.2. Congestion avoidance

During the congestion avoidance phase, Vegas does not continually increase the *CWND*. Instead, it attempts to detect incipient congestion by comparing the *Actual* rate with the *Expected* rate. The *CWND* is kept constant when $\varDelta$ lies between two thresholds α and β. If $\varDelta$ is greater than β, it is considered to indicate incipient congestion and thus the *CWN*D is reduced. On the other hand, if $\varDelta$ is lesser than α, the connection may be underutilizing the available bandwidth and thus the *CWND* is increased. *CWND* is updated on a per-*RTT* basis. The rule for updating *CWND* can be expressed as follows:

$$CWND = \begin{cases} CWND + 1, & \text{if } \varDelta < \alpha \\ CWND - 1, & \text{if } \varDelta > \beta \\ CWND, & \text{if } \alpha \leqslant \varDelta \leqslant \beta \end{cases}. \tag{2}$$

2.2.3. Fast retransmission and fast recovery

TCP Vegas measures the *RTT* for every packet sent based on fine-grained timer values. By using fine-grained *RTT* measurements, a timeout period is computed for each packet. When a duplicate *ACK* is received, Vegas checks whether or not the timeout period of the oldest unacknowledged packet has expired. If it has, the packet is retransmitted. This modification leads to packet retransmission after just one or two duplicate *ACKs*. When a non-duplicate *ACK* that is the first or second *ACK* after a fast retransmission is received, Vegas again checks for the expiration of the oldest unacknowledged packet following which it may retransmit another packet.

After a packet retransmission is triggered by a duplicate *ACK* and the *ACK* of the lost packet is received, the *CWND* will be reduced to alleviate the network congestion. Vegas sets the *CWND* in two cases. If a lost packet has been transmitted just once, the *CWND* will be three-fourth of the previous window size. Otherwise, it is considered to indicate more serious congestion, and *CWND* is set to one half of the previous window size. It should be noted that if multiple packet losses occur during one round-trip time and trigger more than one fast retransmission, the *CWND* will be reduced only for the first retransmission.