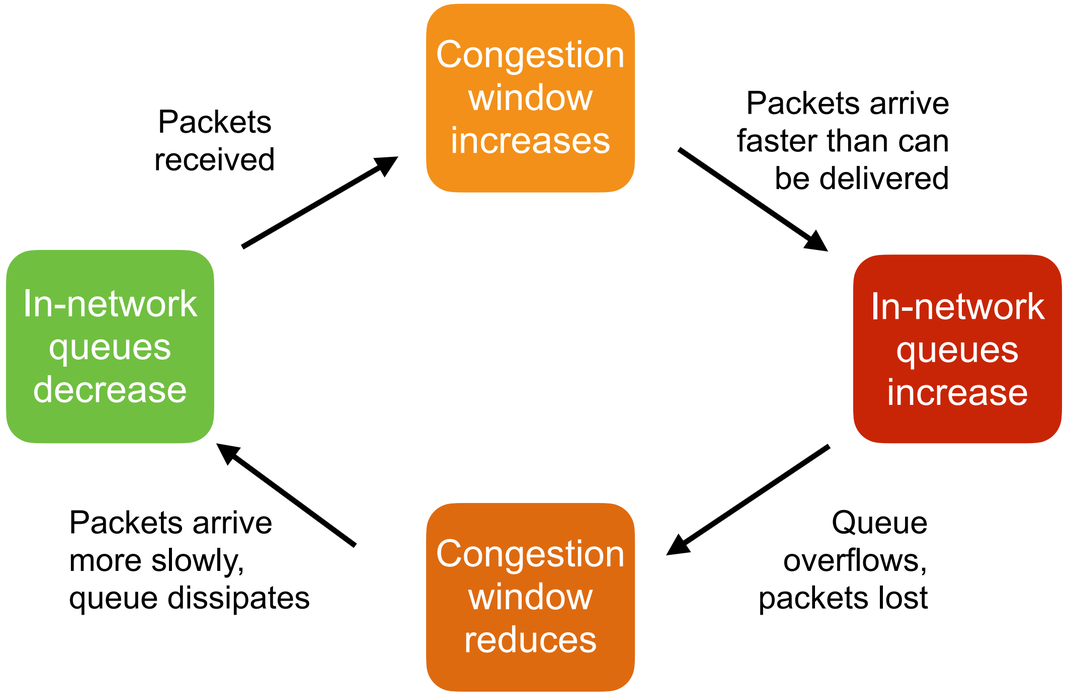
Delay-based Congestion Control

- Traditional TCP causes latency
- TCP Vegas
- TCP BBR

Impact of TCP on Latency

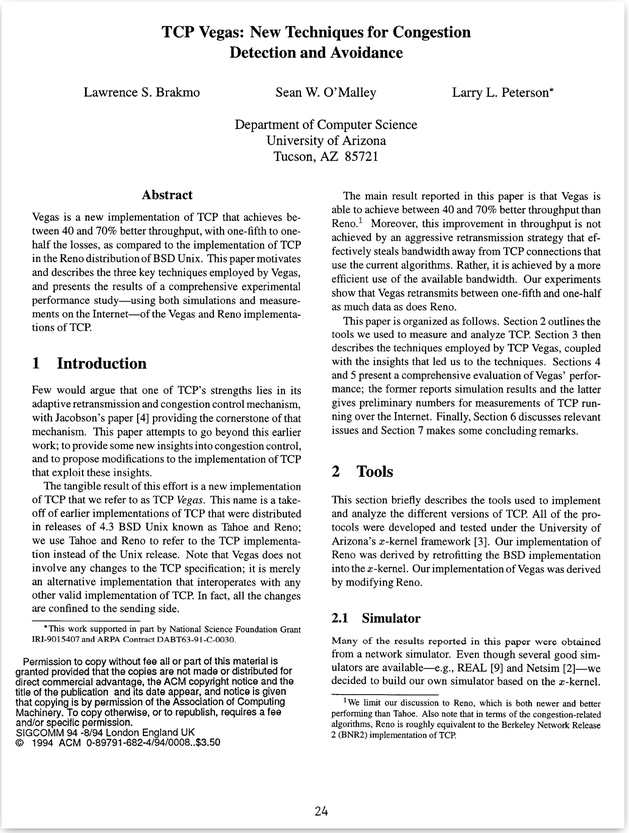
- TCP Reno and TCP Cubic **both** aim to fill the network



- TCP Reno and Cubic use packet loss as congestion signal → increase congestion window until queue overflows and packet is lost
- No matter how much big the queue, TCP Reno or Cubic **will** cause it to overflow, if given enough data to send
- Packets waiting in queues within network adds latency → increase RTT

TCP Vegas: Reducing Latency

- Key insight: if sending faster than the network can deliver, packets will be queued
  - TCP Reno and Cubic wait until the queue overflows and packets are lost before slowing down
  - **TCP Vegas** watches for the increase in delay as the queue starts to fill up → slows down before the queue overflows
    - Queues are smaller → lower latency
    - Packets are not lost, so don't need retransmission
  - Only affects congestion avoidance; slow start unchanged



TCP Vegas: New Techniques for Congestion Detection and Avoidance

Lawrence S. Brakmo Sean W. O'Malley Larry L. Peterson*

Department of Computer Science
University of Arizona
Tucson, AZ 85721

Abstract

Vegas is a new implementation of TCP that achieves between 40 and 70% better throughput, with one-fifth to one-half the losses, as compared to the implementation of TCP in the Reno distribution of BSD Unix. This paper motivates and describes the three key techniques employed by Vegas, and presents the results of a comprehensive experimental performance study—using both simulations and measurements on the Internet—of the Vegas and Reno implementations of TCP.

The main result reported in this paper is that Vegas is able to achieve between 40 and 70% better throughput than Reno.[1] Moreover, this improvement in throughput is not achieved by an aggressive retransmission strategy that effectively steals bandwidth away from TCP connections that use the current algorithms. Rather, it is achieved by a more efficient use of the available bandwidth. Our experiments show that Vegas retransmits between one-fifth and one-half as much data as does Reno.

This paper is organized as follows. Section 2 outlines the tools we used to measure and analyze TCP. Section 3 then describes the techniques employed by TCP Vegas, coupled with the insights that led us to the techniques. Sections 4 and 5 present a comprehensive evaluation of Vegas' performance; the former reports simulation results and the latter gives preliminary numbers for measurements of TCP running over the Internet. Finally, Section 6 discusses relevant issues and Section 7 makes some concluding remarks.

1 Introduction

Few would argue that one of TCP's strengths lies in its adaptive retransmission and congestion control mechanism, with Jacobson's paper [4] providing the cornerstone of that mechanism. This paper attempts to go beyond this earlier work; to provide some new insights into congestion control, and to propose modifications to the implementation of TCP that exploit these insights.

The tangible result of this effort is a new implementation of TCP that we refer to as TCP *Vegas*. This name is a take-off of earlier implementations of TCP that were distributed in releases of 4.3 BSD Unix known as Tahoe and Reno; we use Tahoe and Reno to refer to the TCP implementation instead of the Unix release. Note that Vegas does not involve any changes to the TCP specification; it is merely an alternative implementation that interoperates with any other valid implementation of TCP. In fact, all the changes are confined to the sending side.

*This work supported in part by National Science Foundation Grant IRI-9015407 and ARPA Contract DABT63-91-C-0030.

SIGCOMM 94 -8/94 London England UK
© 1994 ACM 0-89791-682-4/94/0008..$3.50

2 Tools

This section briefly describes the tools used to implement and analyze the different versions of TCP. All of the protocols were developed and tested under the University of Arizona's *x*-kernel framework [3]. Our implementation of Reno was derived by retrofitting the BSD implementation into the *x*-kernel. Our implementation of Vegas was derived by modifying Reno.

2.1 Simulator

Many of the results reported in this paper were obtained from a network simulator. Even though several good simulators are available—e.g., REAL [9] and Netsim [2]—we decided to build our own simulator based on the *x*-kernel.

[1] We limit our discussion to Reno, which is both newer and better performing than Tahoe. Also note that in terms of the congestion-related algorithms, Reno is roughly equivalent to the Berkeley Network Release 2 (BNR2) implementation of TCP.

24

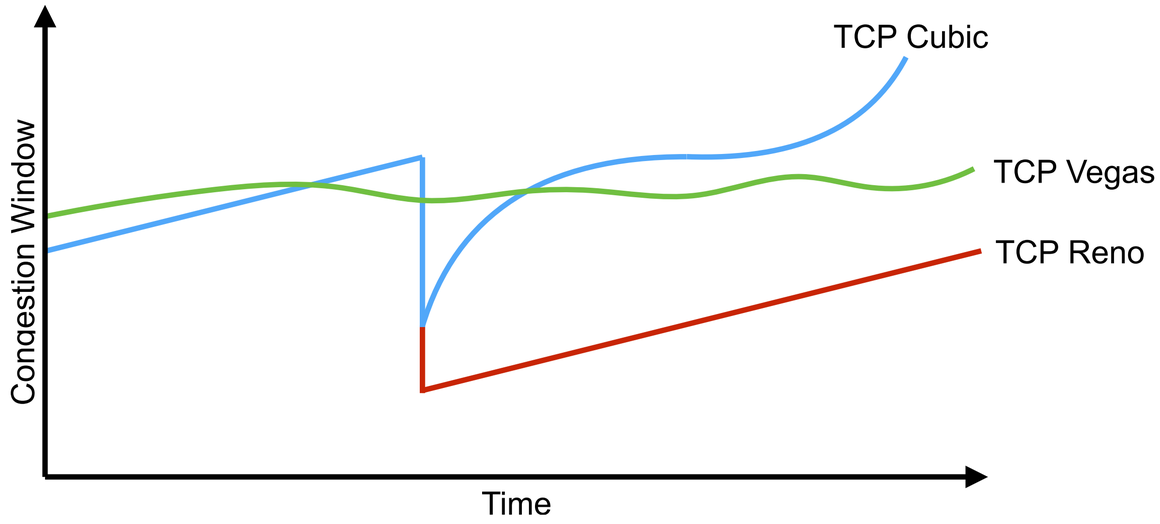
L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance", ACM SIGCOMM Conference, London, August 1994. https://dx.doi.org/10.1145/190314.190317

TCP Vegas Congestion Control (1/2)

- Measure **BaseRTT**
  - The smallest time between sending a packet and getting its acknowledgement
  - BaseRTT will be from a packet delivered without much queueing
- Calculate **ExpectedRate** = $W/_{BaseRTT}$
  - The congestion window, W, determines how many packets are sent each RTT
  - If the network can support this sending rate, the complete window should be delivered within the RTT
- Measure **ActualRate**
  - Bytes sent divided by actual RTT, for each packet
  - Measure how fast packets are actually received, based on acknowledgements returned
  - ActualRate ≤ ExpectedRate since packets can't be delivered faster than they're sent

TCP Vegas Congestion Control (2/2)

- Compare ExpectedRate with ActualRate and adjust window:
  - If ExpectedRate - ActualRate < R_1 then additive increase to window
    - Data arriving at close to expected rate, can likely send faster
  - If ExpectedRate - ActualRate > R_2 then additive decrease to window
    - Data arriving slower than it's being sent, slow down
  - Parameters R_1 and R_2, where $R_1 < R_2$, critical to performance



Additive increase and decrease in size window → smoother changes in sending rate than Reno or Cubic

TCP Vegas: Limitations

- Delay-based congestion control is a good idea in principle
  - Reduces latency
  - Reduces packet loss
- But loss- and delay-based congestion control don’t cooperate
  - TCP Reno and TCP Cubic aggressively increase queue sizes
  - Increases RTT and reduces ActualRate as measured by TCP Vegas
  - Forces TCP Vegas to slow down; cycle repeats → TCP Vegas sending rate drops to zero over time

- TCP Vegas is not used, because it can’t be deployed alongside TCP Reno or TCP Cubic

TCP BBR

- Is it possible to develop a delay-based congestion control algorithm for TCP that can be deployed on a network alongside TCP Reno and TCP Cubic?
- **Maybe** – TCP BBR ("Bottleneck Bandwidth & RTT") is one attempt
  - Proposal from Google – measures RTT and bandwidth of the bottleneck link, directly sets congestion window
  - Used by YouTube in some cases, but **highly experimental**
    - Recent work by Ranysha Ware and Justine Sherry shows serious fairness problems with TCP BBR v1
      - https://dl.acm.org/doi/10.1145/3355369.3355604
      - https://vimeo.com/showcase/6531379/video/369121357#t=990s
  - TCP BBR v2 in development, might solve these problems – this is an active research area

BBR Congestion-Based Congestion Control

MEASURING BOTTLENECK BANDWIDTH AND ROUND-TRIP PROPAGATION TIME

NEAL CARDWELL
YUCHUNG CHENG
C. STEPHEN GUNN
SOHEIL HASSAS YEGANEH
VAN JACOBSON

By all accounts, today's Internet is not moving data as well as it should. Most of the world's cellular users experience delays of seconds to minutes; public Wi-Fi in airports and conference venues is often worse. Physics and climate researchers need to exchange petabytes of data with global collaborators but find their carefully engineered multi-Gbps infrastructure often delivers at only a few Mbps over intercontinental distances.[6]

These problems result from a design choice made when TCP congestion control was created in the 1980s—interpreting packet loss as "congestion."[1,3] This equivalence was true at the time but was because of technology limitations, not first principles. As NICs (network interface controllers) evolved from Mbps to Gbps and memory chips from KB to GB, the relationship between packet loss and congestion became more tenuous.

Today TCP's loss-based congestion control—even with the current best of breed, CUBIC[1]—is the primary cause of these problems. When bottleneck buffers are large,

acmqueue | september-october 2016 20

N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. BBR: Congestion-based congestion control. ACM Queue, 14(5), September 2016. https://dl.acm.org/doi/10.1145/3012426.3022184

Delay-based Congestion Control

- Traditional TCP causes latency
- TCP Vegas
- TCP BBR