

# An improved method for experimental evaluations of TCP Congestion Control Algorithms

Harhalakis Stefanos

Department of Informatics, TEI of Thessaloniki, Greece

v13@priest.com

Samaras Nikolaos, Fragiadaki Eleni

Department of Applied Informatics, University of Macedonia, Greece

samaras@uom.gr, eleni.fra@gmail.com

## Abstract

Benchmarks are a vital part of most of the performed research activity. Internet is an extremely complex internetwork composed of many smaller but still complex networks based on the Internet Protocol (IP). Transmission Control Protocol (TCP) is the most common protocol for IP networks and one of the toughest to perform realistic tests on. It is assumed that it is impossible to emulate all of Internet's behavior under lab conditions and thus accurate benchmarking and testing of TCP consists a difficult task. We present a review of existing tools that are being used for evaluating TCP Congestion Control Algorithms. We also include a new method for performing detailed benchmarks of TCP Congestion Control Algorithms. The paper presents a custom made utility with more abilities than existing tools and a set of scripts for automating benchmarks and graphing collected data.

## Index Terms

TCP, Benchmark, Congestion Control Algorithms, Experimental Evaluation, Linux

## I. INTRODUCTION

Transmission Control Protocol [1] (TCP) is the most used Transport Layer protocol of the Internet. TCP dominates Internet traffic and is a crucial part of the worlds networking infrastructure. It is a connection-oriented, full-duplex, reliable transport layer protocol that has the ability to provide flow control for its transfers. TCP has evolved in parallel with the Internet, and underwent a lot of adjustments to fit the constant changing needs since its inception. Its ability to adjust one of its parameters, namely window size, is the basis of its flow control feature.

Early versions of TCP known as TCP Tahoe[2] implemented a simplistic method to adjust the window size. TCP Reno [3] was the first improvement proposed by V. Jacobson in an e-mail to end2end-interest@venera.isi.edu and introduced a better approach to window resizing. After that a plethora of proposals were made and many of them were actually implemented and are currently being used by major operating systems.

Research activity on TCP Congestion Control Algorithms seems like a horse race where evaluations and proposals supersede each other in the lead. A large part of the evaluations is performed in network simulators like NS2, OpNet and Omnet++ but tier-1 Internet Service Providers (ISPs) provide us with the most valuable experimental results by taking advantage of their ability to stress existing algorithms under real conditions using intercontinental-scale test suites. Simulator-based performance evaluation is being used primarily for new algorithm research while experimental evaluations are best suited for actual algorithm comparison. Some proposals have been made for

experimental based benchmarking [4], [5], [6] and quite a lot of evaluations have been published. Many research groups monitor common network traffic using a set of well known packet analyzers. Others prefer specialized benchmarking tools like TTCP, Netperf, Netspec, Nettek or the more sophisticated DBS[7].

The results of the contacted studies have lead well recognized scientists like Van Jacobson and Sally Floyd to new TCP Congestion Control proposals [8], [9], [10], [11], [12], [13], [14] that formed today's behavior of Internet TCP traffic. Even though there is a plethora of benchmarking methods we believe that there is still room for improvement. TCP behavior depends on multiple factors including the multitude of used TCP congestion control algorithms, the different network technologies that data traverse, the type of queuing algorithms that are being used along the data path, the unpredictable rate of continuous or momentary packet loss and the number of parallel data transfers.

This paper proposes a methodology for experimental evaluation of the existing TCP congestion control algorithms based on a tool designed and programmed by ourselves, designed to provide extremely detailed results. We use one computer with two Network Interface Cards (NICs) and thus we are able to achieve timings of high detail. For our purpose we took advantage of the Linux operating system in conjunction with the Netfilter packet filtering framework and the iproute2 suite.

The remaining of the paper is organized as follows. We introduce the Transmission Control Protocol in section II. The available benchmarking tools are presented in section III. Then we describe the proposed benchmarking method in section IV followed by the implementation details in section V. Next we provide an overview of our benchmarking suite in section VI and conclude with a brief summary.

## II. TCP OVERVIEW

Transport Layer Protocols transmit data either as a stream or datagrams. A stream is a continuous data flow while a datagram is a method for transmitting individual data chunks. TCP is Reliable Transport layer Protocol (RTP) that sends data as a stream. An essential requirement for RTPs is that received data must be retransmitted until they are acknowledged. The method used by TCP is known as Positive Acknowledgment and Retransmission (PAR) and is implemented by sending positive cumulative acknowledgments (ACKs) upon reception of data (i.e there is no negative acknowledgment). The window size is the amount of data that can be transmitted by each end without receiving an acknowledgment from the other. The whole window and PAR idea is commonly referred to as the "Sliding Window".

Because of the delay that signal propagation and data processing introduce, the maximum speed of TCP transmission at any time is limited by its Current Window Size (cwnd). We refer to the sum of these delays as the Data Propagation Delay. One of the fundamental parameters of TCP transmissions is the Round Trip Time (RTT). RTT is the time a data segment requires to traverse the network and its acknowledgment be received, as shown in Figure 1.

Since a host cannot transmit more than cwnd number of bytes without receiving an acknowledgment, the maximum transmission speed is limited to:

$$MaxSpeed = cwnd/RTT$$

where

$$RTT = SendingPropagationDelay + ProcessingDelay + ReceivingPropagationDelay$$

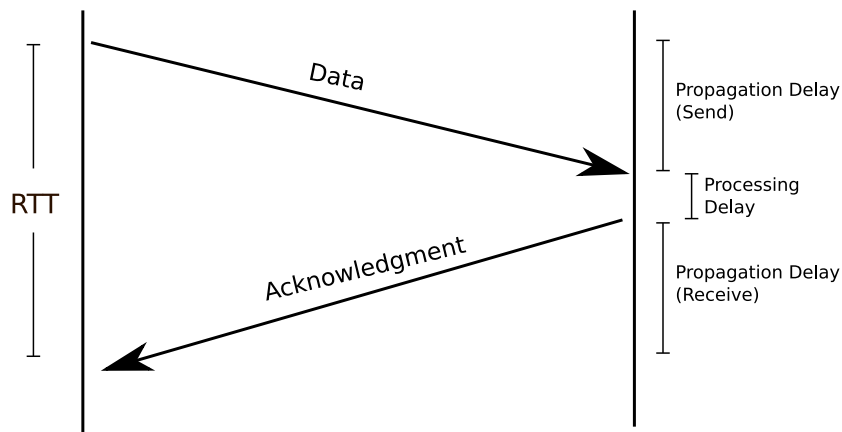


Figure 1. Round Trip Time

In most papers and books it is assumed that:

$$ProcessingDelay = 0$$

and

$$SendingPropagationDelay = ReceivingPropagationDelay = RTT/2$$

Knowing that, we implement flow control algorithms that manage the TCP speed at any time by changing the cwnd. An ideal TCP implementation would adjust its speed to match the exact available bandwidth along the transmission path. Since the data transmission rate cannot be predefined and it has to vary during the connection lifetime, it is essential that TCP adjusts its cwnd and thus its speed every moment. Congestion Control Algorithms try to estimate the RTT and accordingly adjust cwnd and perform packet retransmissions. By measuring packet loss and RTT they are able to guess the available bandwidth. Since RTT is a very crucial measurement, a TCP option that directly measures it with very low processing overhead was proposed in RFC 1323 “TCP Extensions for High Performance” [15] and it is known as RTT Measurement (RTTM).

It must be noted that all this information is of great importance to TCP algorithm designers but for the actual adoption of a TCP algorithm the world needs experimental results. The list of proposed improvements that failed during experimental evaluations is quite long. A well accepted belief dictates that it is practically impossible to represent all of the real world’s conditions using simulator parameters.

### III. EXISTING TCP BENCHMARKING TOOLS

Experimental evaluations are conducted using either real network traffic that is being analyzed by custom or well known IP traffic analyzers like wireshark (former ethereal), or by using specialized benchmarking tools. Here we present an overview of the most widely used.

#### A. Test TCP (TTCP)

TTCP was a simple tool that tried to measure the speed of a link by transmitting data. Now known as nttcp (New TTCP) it is the simplest tool available and it only measures the bandwidth of a connection.

### B. Netperf

Netperf is another network performance tester and it is best described by its manual page:

“Netperf is a benchmark that can be used to measure various aspects of networking performance. Currently, its focus is on bulk data transfer and request/response performance using either TCP or UDP, and the Berkeley Sockets interface.”

### C. Netspec

Netspec is an advanced benchmarking tool, superior to TTCP and Netperf. It supports multiple connections in parallel or serial and is able to emulate the following traffic types:

- FTP (bulk download)
- TELNET (low latency, small packet size)
- Variable Bit Rate (VBR) Video (MPEG, Video-Teleconferencing) (low latency, variable packet size, moderate to high traffic)
- Constant Bit Rate (CBR) Voice (low latency, constant packet size, low to moderate traffic)
- HTTP (World Wide Web) (bursty traffic, large packet size, low initial latency)

Netspec is best suited for tier-1 Internet Service Providers (ISPs) for measuring the actual throughput of their backbone connections.

### D. Nettest (Iperf)

Nettest uses the iperf tool to test a network connection. Using the User Datagram Protocol (UDP) it can provide more information about the underlying network than the former tools. For TCP it can only measure the maximum bandwidth.

### E. DBS

DBS seems the most powerful tool and it can actually monitor TCP delay and bandwidth. Unfortunately the last version was published at 1998 and it seems abandoned.

## IV. A METHOD FOR BENCHMARKING TCP

During our research we figured out of a new method for performing accurate experimental TCP benchmarks. We used one computer with two NICs that emulated a two computer environment. To achieve this we implement the topology shown in figure 2.

Using the Linux operating system and the Network Address Translation (NAT) capabilities of Netfilter we managed to create a virtual topology where each interface card behaves as if it was connected with another computer. Then we create a TCP connection with source IP address 10.20.1.1 and destination IP address 10.30.1.1 and transfer data. The operating system is configured to assume that it can reach 10.30.1.1 using NIC1 and thus it transfers the data through the actual cable connection. During the transmission the following take place:

- An IP packet is prepared to be send to 10.30.1.1 using 10.20.1.1 as source IP address.
- The system uses NIC1 to transmit the data.
- Just before an IP packet is send to the cable it is modified and its source IP address is changed to 10.30.2.1.
- NIC 2 receives an IP packet with source IP address 10.30.2.1 and destination IP address 10.30.1.1.
- Just before the IP packet is processed it is modified again and its destination IP address is changed to 10.20.1.2.
- Finally, the operating system kernel receives an IP packet destined for itself and it handles it as needed.

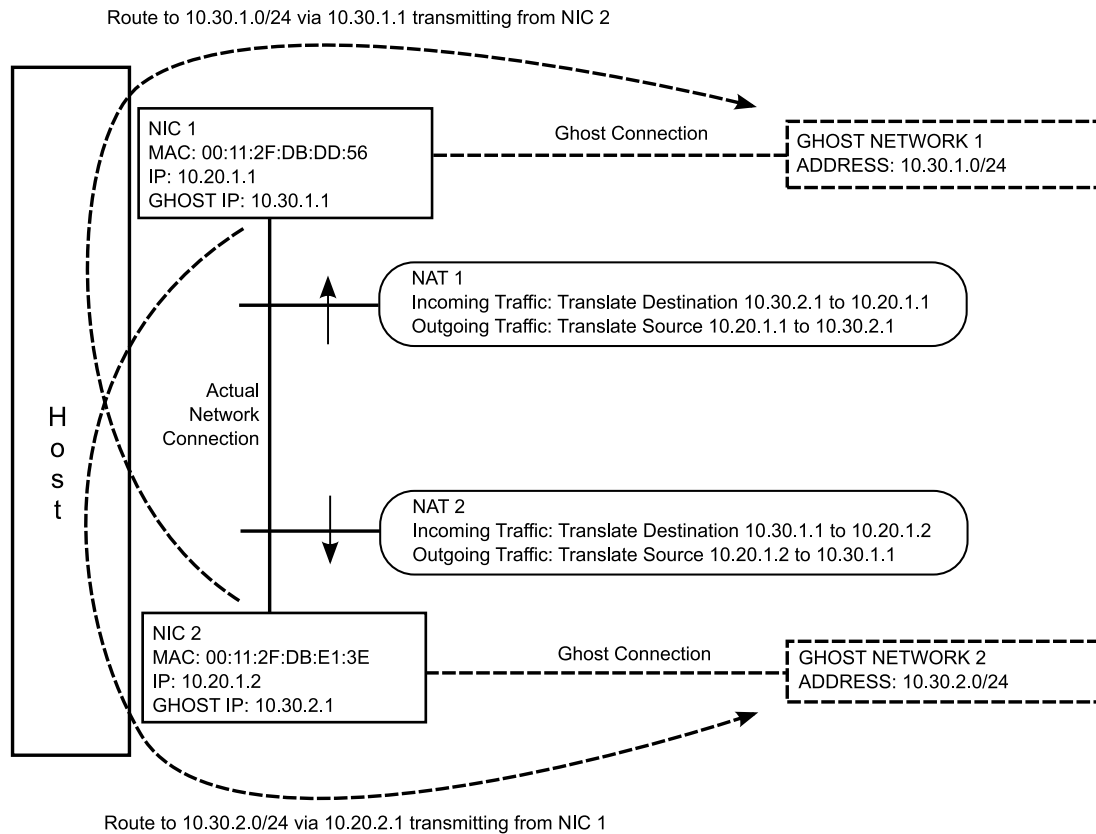


Figure 2. Topology

This way we're able to perform benchmarks using only one PC. By including time information in the transmitted data we're able to evaluate the application layer delay with great precision. We should note that this is not possible using two computers since it is very hard to completely synchronize their clocks. On 100Mbps Ethernet links, the largest IP packet that can be send unfragmented is 1500 bytes and can be transmitted in 15 microseconds ( $1.5 * 10^{-5}$ ). On 1Gbps links the time is reduced to one tenth and on 10Gbps links even more. Therefore the conducting of time sensitive benchmarks using two different stations consists an extremely difficult task since it is nearly impossible to completely synchronize their clocks. The only viable solution is to set their time from a directly connected GPS device.

Apart from the topology we take advantage of the netem (Network Emulator) packet scheduler which can be programmed to emulate a variety of network conditions by applying it to the outbound traffic of each interface.

## V. IMPLEMENTATION

The required configuration can be split down to the following steps:

- Perform standard IP configuration.
- Configure IP routing.
- Configure static Address Resolution Protocol (ARP) mappings.
- Configure NAT.
- Configure netem.

### A. Standard IP configuration

We set up the IP addresses of the two NICs

### B. Configure IP routing

We add the following routes using the “ip route” command:

- Route to 10.30.1.0/24 via 10.30.1.1 using interface NIC2 and forcing the system to believe that this is a directly attached network using the 'onlink' keyword.
- Route to 10.30.2.0/24 via 10.30.2.1 using interface NIC1 and forcing the system to believe that this is a directly attached network using the 'onlink' keyword.

### C. Configure static ARP mappings

We need to avoid ARP requests for non existing addresses and thus we manually set up two static, non expiring, ARP cache entries:

- Map 10.30.1.1 to the MAC address of NIC1
- Map 10.30.2.1 to the MAC address of NIC2

### D. Configure NAT

We perform the following address translations:

- Change the destination address of all traffic entering NIC1 and destined for 10.30.1.0/24 to the actual IP address of NIC1 (10.20.1.1).
- Change the destination address of all traffic entering NIC2 and destined for 10.30.2.0/24 to the actual IP address of NIC2 (10.20.1.2).
- Change the source address of all traffic leaving NIC1 to 10.30.1.1
- Change the source address of all traffic leaving NIC2 to 10.30.2.1

### E. Configure netem

Netem can be configured to emulate a variety of network problems. It can

- increase the delay by an amount of time and apply delay jitter for more realistic results.
- use four different distributions for random number generation.
- drop a percentage of transmitted packets.
- corrupt a percentage of transmitted packets.
- duplicate a percentage of transmitted packets.
- reorder a percentage of transmitted packets using a configurable reordering gap.

The actual netem configuration is related to the performed benchmark and not to the benchmarking method.

## VI. CUSTOM BENCHMARKING SUITE

Apart from the topology we created a custom benchmarking suite. We wrote a custom program in C/C++ that consists of a client and a server part. The client connects to the server and performs bulk data transfers using a custom number of parallel connections. All transmitted data contain a timestamp that is used to determine the application layer delay. The client part records statistics like delay and throughput in custom intervals and as a whole and it keeps distinct information for each TCP connection. This way we are able to calculate the delay and bandwidth jitter.

For data storage we selected the excellent opensource PostgreSQL database. The schema of the database is shown in figure 3.

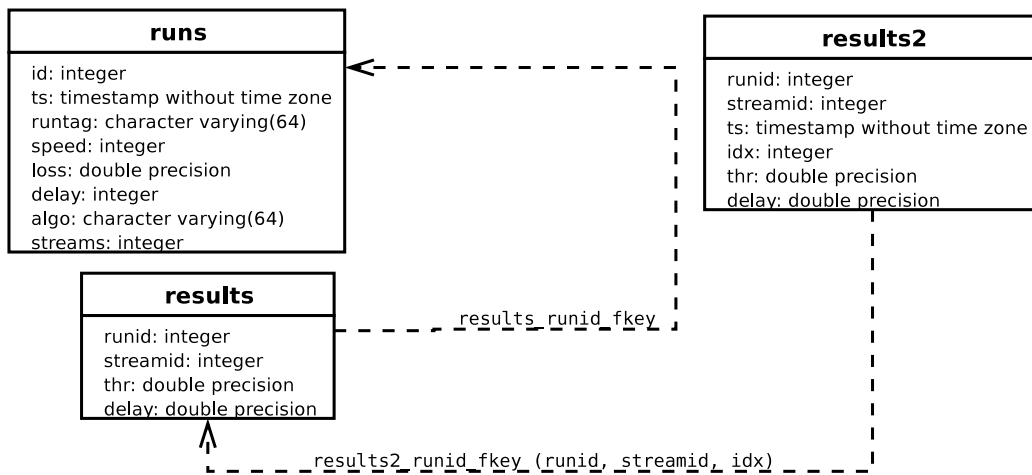


Figure 3. Database Schema

We also created a set of scripts to automate the tests. They are used to automatically alter network parameters according to our needs and perform multiple runs for each test case. Finally we wrote a data visualization program in python that reads the recorded data from the database and draws graphs like the one in figure 4.

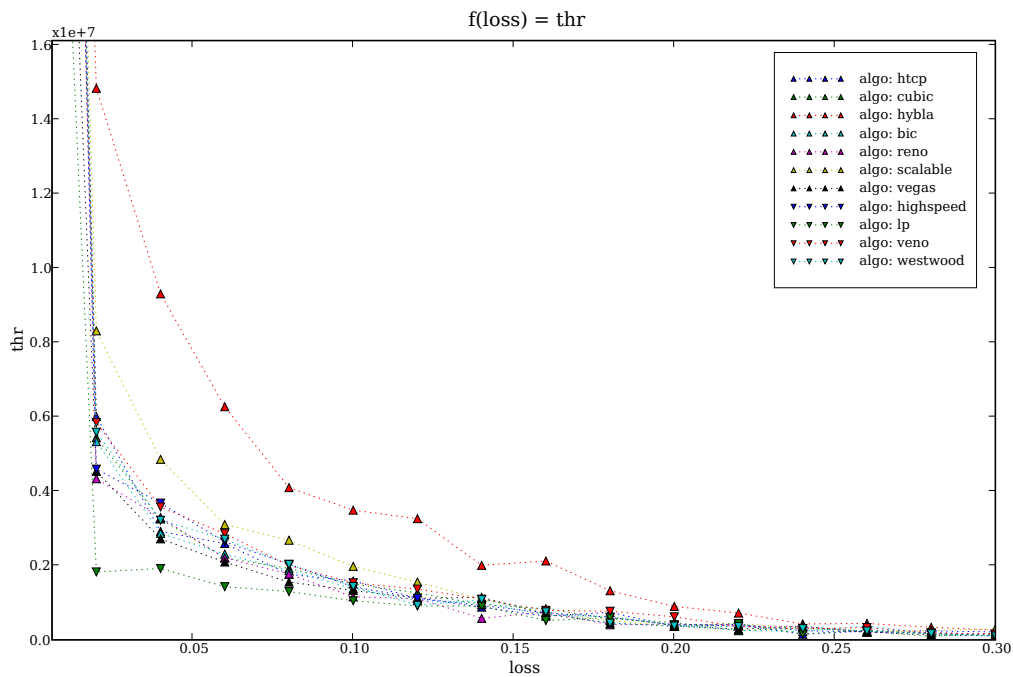


Figure 4. Sample graph of throughput over packet loss

## VII. CONCLUSIONS

We believe that our method is unique among others because of its ability to be run with only one computer with two Network Interface Cards (NICs). Since we operate on one computer we are able to get detailed transmission delay information that other methods are not capable of. Since Gigabit Ethernet sends one full sized Ethernet frame in 1.5 usec it is crucial that there are no clock differences between the sender and the receiver. Having this in mind, existing methods may be able to provide this kind of results only when stations equipped with GPS powered clocks are being used.

## REFERENCES

- [1] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFC 3168.
- [2] Van Jacobson. Congestion avoidance and control. In *ACM SIGCOMM '88*, pages 314–329, Stanford, CA, August 1988.
- [3] Van Jacobson. Modified tcp congestion avoidance algorithm, April 1990.
- [4] David Wei Pei. Time for a tcp benchmark suite?
- [5] Yee-Ting Li, Douglas Leith, and Robert N. Shorten. Experimental evaluation of tcp protocols for high-speed networks. Technical report, Hamilton Institute, NUI Maynooth, 2005.
- [6] Sangtae Ha, Yusung Kim, Long Le, Injong Rhee, and Lisong Xu. A step toward realistic performance evaluation of high-speed tcp variants. Technical report, Department of Computer Science, North Carolina State UniversityDepartment of Computer Science, University of Nebraska, 2006.
- [7] Y. Murayama and S. Yamaguchi. Dbs: A powerful tool for tcp performance evaluations, 1997.
- [8] Lawrence S. Brakmo and Larry L. Peterson. TCP vegas: End to end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, 1995.
- [9] S. Floyd. The NewReno Modification to TCP's Fast Recovery Algorithm. *RFC 2582*, 1999.
- [10] Douglas Leith and Robert Shorten. H-tcp: Tcp for high-speed and long-distance networks, 2004.
- [11] Carlo Caini and Rosario Firrincieli. Tcp hybla: a tcp enhancement for heterogeneous networks. *International Journal of Satellite Communications and Networking*, 22(5):547–566, August 2004.
- [12] T. Kelly. Scalable tcp: Improving performance in highspeed wide area networks, 2003.
- [13] A. Kuzmanovic and E. Knightly. Tcp-lp: A distributed algorithm for low priority data transfer, 2003.
- [14] *Binary Increase Congestion Control for Fast, Long Distance Networks*, 2004.
- [15] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC 1323 (Proposed Standard), May 1992.