Debugging end-to-end performance in commodity operating systems.

Sven Ubik, <ubik@cesnet.cz> and Pavel Cimbal, <P.Cimbal@sh.cvut.cz>
CESNET, Prague, Czech Republic

I. Introduction

The Internet infrastructure is very diverse, but when it comes to end hosts interfacing the Internet to the users, the PC hardware with one of a few so called commodity operating systems, such as Microsoft Windows or Linux is prevalent.

It turns out that many performance problems in network communication, particularly in high-speed long-distance networks, are caused by improper behaviour of end hosts, rather than by the network itself. Understanding of the behaviour of commodity operating systems, networking applications, their mutual interaction and their interaction with a computer network is required to identify and resolve these problems. Our goal in this paper is to identify some of these problems and suggest possible solutions. Many advanced Internet users, requiring high network performance, use the Linux operating system. Therefore, we decided to concentrate on networking support in Linux. Particularly, we will show that just setting large TCP buffers is not suf£cient to reliably achieve high throughput.

II. Using commodity operating systems in high-speed long-distance networks

Based on net¤ow statistics from backbone routers, over 95% of data currently transferred over the Internet is carried in the TCP protocol. The current state of networking support in all commodity operating systems after the default installation procedure and system boot is that they perform very poorly over high-speed long-distance networks. The primary reason are small default TCP buffers resulting in TCP connections being slowed down by small ¤ow control windows.

In most cases, increasing the size of TCP buffers from the default values helps to increase throughput. Several tools have been developed in the form of kernel patches or daemons to help £nd the optimal buffer sizes for particular network conditions [1], [2], [3]. Using these tools is a procedure for advanced users requiring a lot of manual installation and con£guration. Simply setting a proper size of TCP buffers allows to achieve throughput at most a few hundreds of megabits per second. The primary limitation is in TCP congestion avoidance based on AIMD(1, 1/2) algorithm [12], [11]. There are proposals to modify TCP congestion avoidance by adjusting its aggresiveness and responsiveness according to the current TCP window size [4] or to use another congestion control algorithm [5].

III. Gap between research, specifications and implementation

It appears that some commodity operating systems include their own modi£cations to TCP implementation with respect to the speci£cation in RFCs and other documents. These modi£cations should be considered in studies of new mechanisms designed for the use in public Internet. In¤uence of these modi£cations and of implementation problems in some commonly used networking applications on the network performance can be stronger that various subtle improvements to TCP congestion control proposed in research papers.

IV. Linux TCP implementation specifics

Linux kernel 2.2 behaved closely to the speci£cation in RFCs. Beginning with the Linux kernel 2.4 a lot of differences from the speci£cation have been introduced. Most differences are very poorly documented only within the kernel source code. This is surprising considering how widely used is the Linux kernel 2.2 for network performance experiments.

A. Sizes of TCP buffers

Actual sizes of sending and receiving TCP buffers (and rcwd advertised by the receiver) are different from values supplied to `setsockopt()` call for `SO_SNDBUF` and `SO_RCVBUF` socket options. This must be *taken into account when doing performance tests* and when setting buffer sizes for proper TCP operation. The supplied values are £rst multiplied by 2 and stored in internal variables. The content of these variables is returned by `getsockopt()` call. If `setsockopt()` was not called, system default values (see below) are copied to internal variables. In Linux 2.2 the content of these internal variables is again divided by two before setting the buffer sizes or advertised rwnd. Division by two is probably legacy to deal with old TCP implementations that used 16-bit signed variables for window arithmetics. And multiplication by two is probably legacy of patching the previous patch to behave as expected. In Linux 2.4 division by two was removed. The real buffer sizes are therefore twice as speci£ed by `setsockopt()`. When an application does not explicitely request buffer sizes by calling `setsockopt()`, the kernel uses heuristics to choose the initial buffer sizes according to `net/ipv4/tcp_[rw]mem` kernel variables and current memory consumption.

B. Application buffer clamping

The advertised rwnd is computed from the internal variable described in the previous section using `tcp_adv_winscale` kernel parameter as: internal variable $*(1 - 1/2^{tcp\_adv\_winscale})$. The remaining part of the receiving buffer is clamped as application buffer. The purpose is to smooth out advertised rwnd by setting aside part of the received data waiting to be read by application out of the receiving buffer.

C. Runtime window moderation

During the connection, the kernel tries to moderate rwnd advertised by the receiver as well as cwnd computed by the sender. The advertised rwnd grows from a small initial value to the size of the receiving buffer (without the clamped application buffer). This growth is driven by data being received. If little data is received, rwnd grows slowly. The computed cwnd is moderated so that it does not grow too much over the "¤ight size". The purpose of this moderation is to prevent large packet loss, which could be caused by a sudden burst coming after a period of low sender bandwidth, when cwnd was allowed to grow unimpeded and rwnd was too high.

D. Fast path, slow path

The kernel can process packets in two modes - fast path and slow path. If the connection is purely unidirectional, that is only pure ACKs are sent in one direction and data segment in the other direction, fast path is used. If the connection is bidirectional (one data segment sent in the other direction is suf£cient), the kernel switches to slow path.

E. TCP parameter cache

Certain TCP runtime parameters such as ssthresh are cached for 10 minutes for individual destination IP addresses and used for subsequent connections to the same destinations. This can in¤uence performance of subsequent nconnections.

V. Observations in performance debugging

Low and unreliable throughput was observed between two machines connected at different points of the European Géant network. One PC was located in Uninett, Norway and the other was located in Cesnet, Czech Republic.

Setting large TCP buffers helped to get only about 25% of available capacity and throughput began to decrease when buffer sizes were increased over certain limit.

Both PCs were PIII/850 MHz with Gigabit Ethernet adapters in 64-bit/ 66 MHz slots. Terrestrial distance along the wires was approximately 1530 km. The connection consisted of 10 routers and Gigabit Ethernet, OC-48 and OC-192 links (only 1.2 Gb/s was enabled on OC-192 links by £lters due to cost limitations). The round-trip time was 40 ms. The slowest links were running Gigabit Ethernet. Therefore the capacity of the empty pipe was 1000 Mb/s * 40 ms = 5 MB. We enabled window scaling option and set system-wide limits on TCP buffer sizes to 16 MB.

We tried to check available capacity by brute force with RUDE/CRUDE [7] utilities. RUDE sent a stream

of 1500-byte UDP packets at the rate ranging from 300 Mb/s to 1000 Mb/s in 20 Mb/s increments to be captured by CRUDE on the other side. Throughput stopped increasing at 457 Mb/s and was probably limited by the speed of the receiving PC. However, packet losses were occuring beginning at about 340 Mb/s, which would in¤uence TCP congestion control.

We also tried to check available bandwidth with modi£ed pathload [8] tool. It is distributed with built-in constants limiting its operation to 120 Mb/s. These constants can be tweaked to allow operation to the full Gigabit Ethernet speed. However, our experience is that the pathload output is not reliable. Depending on network conditions, the produced results can be divided into three cases: i) it correctly iterates with variable-rate packet chains to the realistic available bandwidth estimated with 50 Mb/s precision (when that happen on our network path, the indicated available bandwidth was in the range of 850-1000 Mb/s, probably periods of lighter load), ii) it looses lots of packets even in low-rate chains and falsely states the available bandwidth in the range of 50-100 Mb/s and iii) it does not detect any delay increase even in high-rate chains and states an unlikely available bandwidth of 1000 Mb/s.

We then used iperf to measure TCP throughput and scp to copy a 100 MB £le. Averages of £ve measurements for different window sizes are shown in Table I. The indicated window sizes are the real ones after considering buffer size arithmetics and clamping as described in section III. Note that throughput was decreasing when buffer sizes were increased over 2 MB. Throughput over a back-to-back connection of the two same PCs as those used in the measurement was 531 Mb/s.

We checked for packet losses during connection. Neither PC reported any link layer errors. We captured the connection trace by tcpdump run on a third PC connected to a switch port con£gured to mirror outgoing and incoming packets of the sender PC to prevent in¤uence of tcpdump on a monitored connection. The monitoring PC was able to capture almost all packet headers including TCP options with minimal losses. We found there were occassional losses in the monitored connection. Most of them were dealt with by the Fast Recovery mechanism. It looked like lightly-loaded network path. For some reason, the sender was not able to utilize the available capacity although its CPU load was low.

TCP window	iperf throughput
1 MB	135 Mb/s
2 MB	230 Mb/s
4 MB	153 Mb/s
8 MB	115 Mb/s

TABLE I
Throughput over Géant network

	iperf
1 MB	138 Mb/s
2 MB	244 Mb/s
4 MB	357 Mb/s
8 MB	262 Mb/s
16 MB	149 Mb/s

TABLE II
Throughput over Géant network with txqueuelen set to 1000

We began to suspect the end hosts. We found that increasing the txqueuelen parameter set by the ifcon£g utility signi£cantly reduced the effect of throughput decrease with large TCP windows. Measured throughput averaged over £ve measurements is shown in Table II. In all cases, throughput increased intil 4 MB TCP windows. It began to decrease with either 8 MB or 16 MB TCP windows. Our theory is that with small lower-level kernel queues limited by the txqueuelen parameter, these queues can be £lled up too early, the CPU context is switched to another process and by the time it is switched back to the sender process, the queue is empty and the network adapter has nothing to send. To deal with Gigabit Ethernet speed and 100 Hz default rate of the Linux system timer, we need to increase txqueuelen to about 1000 packets (1000*1500 bytes* 8 bits / 10^9 bits/second = 0.12 second). We found that setting it to higher values did not have any further effect on resulting throughput.

By further inspection of tcptrace diagrams we found

that large-window connections (at 8 MB and 16 MB) exhibited signi£cantly more losses than small-window connections. These losses frequently resulted in repeated slow starts, reducing ssthresh limit with very slow cwnd increase by congestion avoidance afterwards. All these connections exhibited round-trip times ¤uctuating up to 120 ms, that is much higher than the empty pipe round-trip of 40 ms. Our theory is that this effect is caused by two factors. First, although the TCP connection is self-regulating in a sense that a larger round-trip time means slower cwnd increase, it shows that with a large window advertised by the receiver, the sender can overshoot available bandwidth too much and the decrease in sender speed comes too late to prevent congestion and loss. Second, utilizing buffered pipe capacity instead of empty pipe capacity does not increase throughput, but £lled-up queues increase probability of congestion caused by ¤uctuating cross traf£c. Therefore, we recommend not to set receiver advertised window too higher. Setting it only slightly more than the empty pipe capacity allows the best performance TCP connections in most cases.

We also observed that in some cases cwnd was increasing very slowly from the beginning of the connection and failed to open enough to use available nadwidth during the connection time. It turned out that ssthresh was set too low at the beginning of the connection as a result of caching from previous connections and prematurely ending the slow start phase. Although we found that this phenomenon was not common, it seems that TCP parameters cache is more likely to be harmfull than useful and we recommend to ¤ush its content before opening every long-distance connection. It can be done with `echo 1 > /proc/sys/net/ipv4/route/flush` command.

VI. PRT initiative

In order to help users who need high network performance and do not have expertise or time to perform debugging procedures themselves, the European *Performance Response Team (PRT)* initiative has started. Initiated by European NRENs (National Research and Educational Networks), its mission should be [13]: “To provide a support structure for end user to help solve performance issues when using applications over a computer network”. Roughly speaking it should be comparable to what CERT means for computer security. A kick off meeting will take place in Amsterdam on December 17, 2002. As part of the PRT project, the “End-to-end performance cookbook” should be developed to share the necessary expertise. It is proposed to be an online, well structured resource supporting semi-automatable updates from users to keep the content up-to-date and the amount of work manageable.

VII. SCAMPI - A Scaleable Monitoring Platform for the Internet

In order to provide advanced programmable monitoring capabilities for high-speed networks, SCAMPI, a two-and-a-half-year European project is currently developing a scaleable monitoring platform for the Internet [9]. The architecture will use both an intelligent, high-speed network adapter, running at 10 Gb/s and developed as part of the project, as well as a commodity Fast Ethernet or Gigabit Ethernet network adapters. The middleware will provide modules for reading packets from speci£ed named ¤ows and applying functions on them, accessible to programmers in the form of MAPI (Monitoring API). On top of MAPI, multiple concurrent monitoring applications will be executed, such as packet capture, QoS monitoring or intrusion detection. SCAMPI is primarily designed for passive monitoring, but it will also function as a programmable packet generator for active monitoring. Moving time-critical operations, such as various counters and pattern recognition functions into hardware will allow to perform monitoring of speci£ed phenonema affecting performance in high speed.

VIII. Future work

We are now working on investigation of performance problems of some commonly used network applications such as scp.

We plan to study implementation of more appropriate receiver advertised buffer moderation.

We will work on QoS and performance monitoring application running on top of the SCAMPI architecture that will help to identify certain performance related problems.

We will participate in the PRT initiative.

IX. Conclusion

- The in¤uence of the networking subsystem con£guration on end hosts and the in¤uence of modi£cations to TCP implementation already present in commodity operating systems on the resulting throughput and TCP behaviour can very signi£cant and probably stronger than the in¤uence of various enhancements of the TCP congestion control described in literature.
- Analytical studies and simulations about TCP performance should take into account a huge installed base of TCP implementation using different congestion control from what is described in RFCs.
- Lots of research has been performed, new mechanisms have been explored by simulations and speci£ed in RFCs and Internet Drafts but commodity operating systems are being developed by different people.
- PRT initiative kicked off, can it be the place for interaction of research people with software developers, equipment manufacturers and other communities involved in high-performance networking?

References

[1] W. Feng, M. Fisk, M. Gardner, E. Weigle. *Dynamic Right-Sizing: An Automated, Lightweight, and Scalable Technique for Enhancing Grid Performance*, PfHSN 2002, Berlin, Germany.

[2] Tom Dunigan, Matt Mathis, Brian Tierney. *A TCP Tuning Daemon*, SC2002, Baltimore, Maryland, USA.

[3] *Web100, version 2.1*, `http://www.web100.org`.

[4] Sally Floyd. "HighSpeed TCP for Large Congestion Windows", *draft-¤oyd-tcp-highspeed-00.txt*, Internet Engineering task Force, June 2002.

[5] Dina Katabi, Mark Handley, Charlie Rohrs. *Internet Congestion Control for Future High Bandwidth-Delay Product Environments*, ACM SIGCOMM 2002.

[6] "NIST Net", Internetworking Technology Group (ITG), National Institute of Standards and Technology (NIST), `http://snad.ncsl.nist.gov/itg/nistnet`.

[7] RUDE/CRUDE utilities, `http://rude.sourceforge.net`.

[8] Manish Jain, Constantinos Dovrolis. *Pathload: a measurement tool for end-to-end available bandwidth*, PAM 2002.

[9] SCAMPI project. `http://www.ist-scampi.org`.

[10] Sven Ubik, Josef Vojtech. *Using end-to-end bandwidth estimation tools in high-speed networks*, CESNET Technical Report, work in progress.

[11] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi. "The Macroscopic Behaviour of the TCP Congestion Avoidance Algorithm", *Computer Communications Review*, Vol. 27, No. 3, July 1997.

[12] Jitendra Padhye, Victor Firoiu, Don Towsley, Jim Kurose. "Modeling TCP Throughput: A Simple Model and its Empirical Validation", *IEEE/ACM Transactions on Networking*, April 2000.

[13] Prezentation of Victor Reijs, 9th TF-NGN meeting, October 18, 2000, Budapest, Hungary.