Analysis of TCP/IP Protocol Processing in Gigabit Networks

RALF LEHMANN, MIRKO BENZ
Institute for System Architecture, Chair for Computer Networks
Dresden University of Technology
01062 Dresden
GERMANY
{lehmann, benz}@rn.inf.tu-dresden.de http://www.rn.inf.tu-dresden.de

Abstract: - Gigabit networks cannot be fully utilized even by today's high end systems. The processing requirements of applications and advanced services like quality of service or security processing aggravate this situation which will get critical with next generation networks. To analyze these shortcomings, this paper presents the measurement results of processing and queuing times of TCP packets in Gigabit networks using the Linux network protocol stack. Finally, possible directions towards protocol acceleration are outlined.

Key-Words: - TCP/IP, Local Area Network, Gigabit network, measurement, processing time, queuing time

1 Introduction

With the growth of the Internet and the involved increase of data volume, the processing of network protocols requires an increasing amount of overall computing of host processors. Therefore, even today's high end systems cannot fully utilize gigabit networks and support advanced services at the same time.

The majority of currently transmitted network data use protocols based on TCP/IP. Beside the most frequently used protocol – HTTP for web based applications – network storage solutions using iSCSI gain an increasing relevance. Most transmission protocols based on TCP use request-response handshaking with only different message sizes for the request and response, respectively. Thus, for a significant speedup of such data transmissions, the TCP protocol processing has to be accelerated. For such an acceleration approach, fundamental knowledge of protocol processing and its bottlenecks is required.

In the following chapter, we give an overview of the fundamental operation of the Linux TCP protocol processing. Afterwards, we describe our protocol processing measurements and the results for the send and the receive path of the network interface, the IP and the TCP layer.

2 Linux TCP Stack Overview

2.1 Receive Path

Figure 1 shows a schematic overview of the Linux TCP receive path.

Every network packet is received by the device driver usually initiated by an interrupt started by the network interface card. The interrupt handler puts the network packets into the input queue of the interface layer and allocates new memory for the receive buffers.

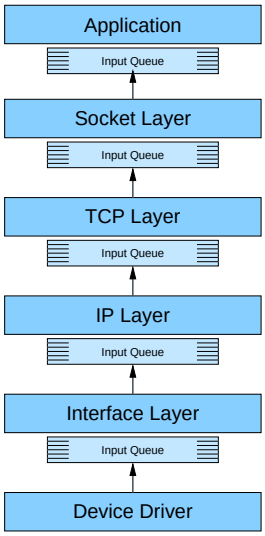

Figure 1: Linux TCP Stack – Receive Path

After that, Ethernet header information of the network packet will be extracted and handled by the interface layer. The Ethernet header is used for filtering and selection of the next protocol queue – in case of IP the input queue of the IP layer.

The IP layer implementation extracts the IP header information and forwards TCP packets either directly to the TCP layer or to the input queue of the TCP layer depending of the state of utilization of the TCP engine.

Finally, the TCP header information of the packets will be handled by the TCP layer. The extracted application data will be sent to the input queue of the socket layer.

2.2 Send Path

A schematic overview of the send path is shown in figure 2.

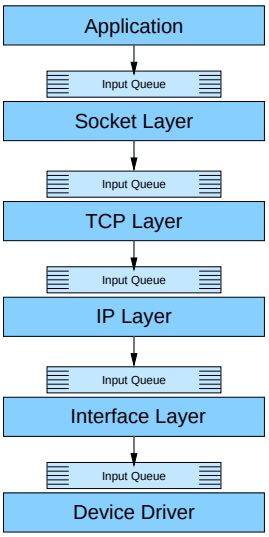

Figure 2: Linux TCP Stack – Send Path

Like receive path processing, every network protocol layer has an input queue, some with configurable size. The application data is delivered to the socket layer and then to the TCP layer. Big messages are split into smaller packets. Afterwards, the checksum is computed and the TCP protocol headers are built. The IP layer is responsible for the ascertainment of routing information necessary for network interface selection.

2.3 Generic TCP data processing

Due to the wide utilization spectrum, the TCP protocol is very complex. However, in today's local area networks only a small amount of the whole protocol implementation is used for data transmission. All other parts are responsible for signalling and error handling, but modern local highspeed networks have very low error probability.

3 TCP Stack Processing Measurement

3.1 Experimental Setup

Since every measurement influences the results, its impact on the measurement should be minimized. A tool with these characteristics and the possibility to insert dynamic measurement probes without reboots offers DProbes [5] in conjunction with Linux Trace Toolkit [10]. By using DProbes, it is possible to insert watchpoints dynamically to monitor the flow of the network packets and log the processing time of every packet at the watchpoints. Another possible method for measurements is the Fast Kernel Trace kit, specified in [11]. In comparison to DProbes, only static measurement points with a circumstantial way of analysis can be used. Therefore, the DProbes solution was preferred.

For measurement of the data packet processing times, the following measurement points were inserted into the receive and the send path of Linux TCP processing (see figures 1 and 2):

TCP receive path

- Start of device independent Ethernet processing
- End of Ethernet processing – enqueuing to IP layer
- Dequeue of IP queue – start of IP layer processing
- End of IP processing – enqueuing or handover to TCP layer
- Dequeuing of TCP queue – start of TCP layer processing
- End of TCP processing – enqueuing to Socket queue

TCP send path

- Start of TCP processing – for packet tracing after split of message into transmittable parts
- End of TCP processing – handover to IP layer queue
- Start of IP processing
- End of IP processing – handover to Ethernet processing queue
- Start of Ethernet device processing
- End of Ethernet device processing – triggering the transmission interrupt

Furthermore, different watchpoints were located in the error handling part of the protocol layers to validate the data path in a local network.

The only modification of the Linux kernel was adding an identifier to the network packet buffer structure *sk_buff* [9] to distinguish the various network packets especially at the queues and allow packet tracing during protocol stack processing. Thus, every buffer structure got an unique number at creation time.

The measurement environment consists of

- Pentium III (800MHz) running Kernel 2.4.18 with DProbes patch applied
- Dual Athlon (1,4GHz) running Kernel 2.4.18
- DLink DGE-550T network cards (Gigabit Ethernet network card)

For data transmission the performance measurement software *netperf* [4] was used. To involve a wide range of transmission scenarios, the measurement was made using request-response data transmissions with a constant request message size of 128 bytes and a variable response size between 256 and 65536 bytes with steps of 256 bytes. For comparable results the transmissions were done in both directions.

3.2 TCP Stack Processing Measurement Results

3.2.1 Processing Time by Message Size

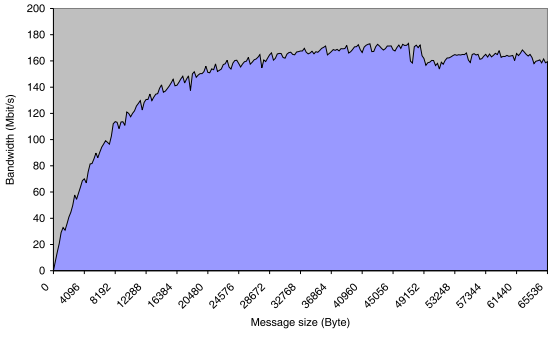

Figure 3: Bandwidth by response message size

The results of the measurement of the average packet processing and queue time in the receive path are shown in figure 4. It illustrates the increasing queue time depending on the response message size due to a slower packet processing in comparison to the packet receive rate. With increasing message

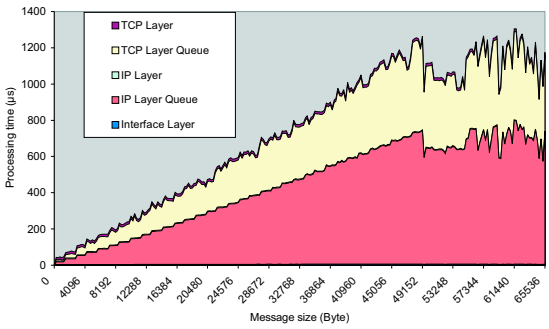

Figure 4: Processing time by response message size (receive path)

size, the request-response scenario will almost become a streaming scenario with at last maximum bandwidth usage as shown in figure 3. Due to the relatively slow host processor, only a small amount of the line speed can be used. If the maximum of usable transmission speed is reached, the queuing times of the packets stay at a relatively constant maximum level even with increasing message sizes.

The Ethernet MTU size of 1500 bytes causes peaks in the graphs every 1448 bytes of the message size since every message is split into parts of 1448 bytes plus 20 bytes for the IP and 32 bytes for the TCP header information. Due to the usage of a request-response transmission, the last packet of every message is mostly not fully filled with application data.

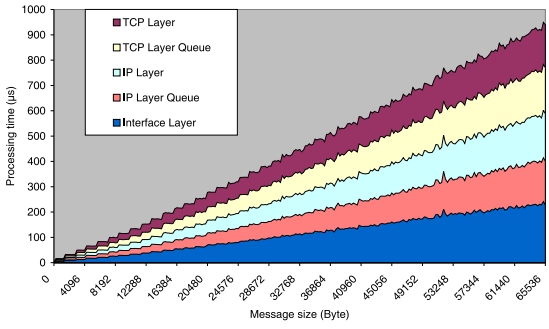

Figure 5: Processing time by response message size (send path)

The measurement results of the TCP send path in dependency to the used message size are demonstrated in figure 5. It shows the processing time for every transmitted messages. All messages are split

into parts of 1448 bytes. As seen in figure 3, the used measurement environment cannot fully utilize Gigabit Ethernet. Thus, the queue times between the protocol layers for every network packet are at a relatively constant low level.

3.2.2 Queuing times

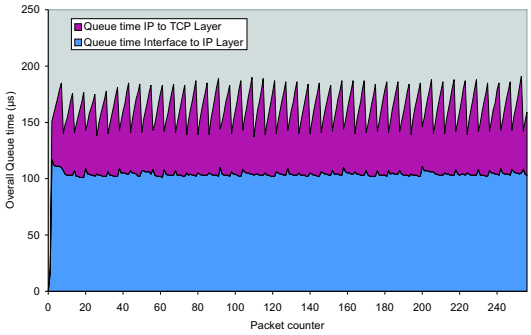

Figure 6: Increase of queuing time (response message size 8192 bytes)

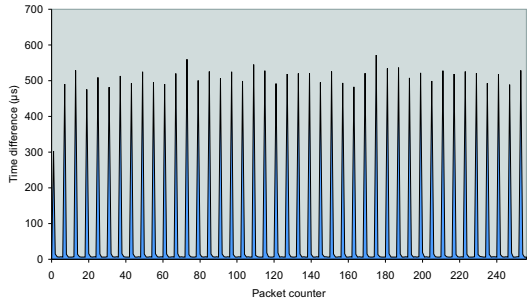

Figure 7: Time difference between sequenced packets (response message size 8192 bytes)

To validate the assertion of too slow protocol processing and the therefore increasing queue times, the receiving of data with selected response message sizes is considered next.

Figure 6 shows the periodically increasing queue times using a response message size of 8192 bytes every 6th transmitted packet (the message of 8192 bytes is split into 6 messages of 1448 bytes). As demonstrated in figure 7, the queue times are minimized when waiting for the next request message to be sent. Since the IP layer queue delay time is relatively constant at a level of 100 μs, the TCP layer queue delay time of every packet differs heavily.

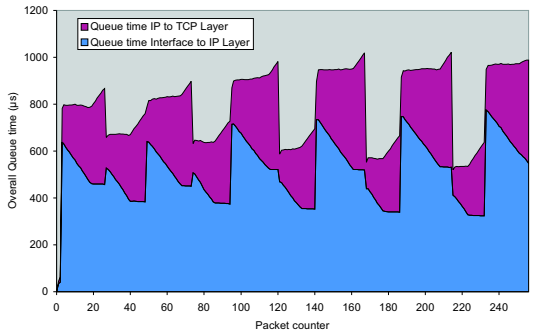

Figure 8: Increase of queuing time (response message size 65536 bytes)

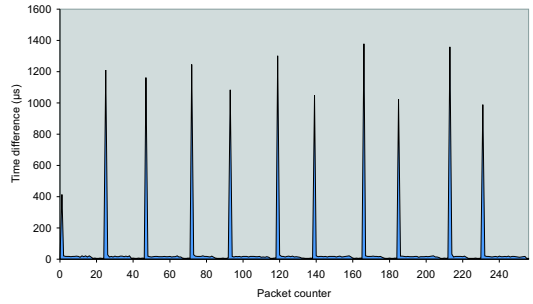

Figure 9: Time difference between sequenced Packets (response message size 65536 bytes)

The queue times when transmitting response messages of a size of 65536 bytes is shown in figure 8. Since the response message size is larger than the TCP send window size of the sender, an additional TCP acknowledgement packet has to be transmitted between the send bursts for every response message. Therefore, the queue times are increased in two steps. This is also demonstrated in figure 9, which illustrates the time difference between receiving two sequential network packets. Besides the request delay, a delay for acknowledgement processing is shown. In comparison to the queue times of receiving response messages of 8192 bytes, the increase of the layer queue times of the IP is contrary to the queue time of the TCP layer. This is caused by the different processing time for the IP and the TCP layer as illustrated in figure 10.

3.2.3 Bottleneck search

Because of the increasing queue times as shown in figures 4, 6, and 8, the real packet processing times

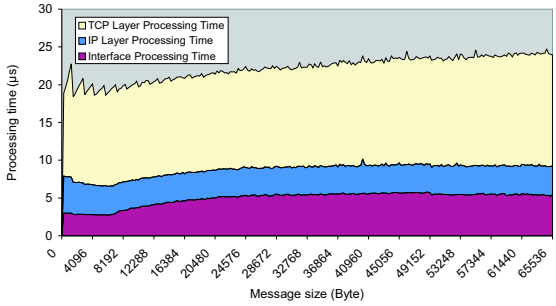

Figure 10: Real packet processing time by message size (receive path)

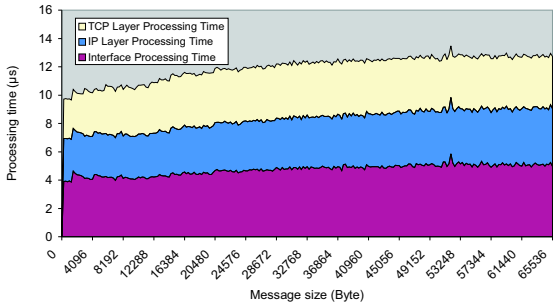

Figure 11: Real packet processing time by message size (send path)

have to be considered. Figure 10 illustrates only the processing time of the protocol receive path and figure 11 shows the processing time for the send path.

As shown, the processing times are relatively constant, only slightly increasing. The receive path takes about 20 to 23 μs computing time, the send path only 10 to 12 μs per network packet. For real send processing, the computing time for the message splitting has to be added. Thus, the TCP protocol processing takes the largest slice of the computing time. It is caused especially by the extensive checksum and window computing.

4 Acceleration Approach

As a result of the measurement of the protocol data path, the processing of the network packets has to be accelerated to get gigabit line speed. Assuming an ideal implementation of a protocol stack with no transmission overhead, the maximum processing time per Ethernet packet is about 12,11 μs (1GBit = 125000000 Bytes = ca. 82563 Ethernet packets of 1514 bytes to be transmitted every second).

Because of the needs of standard conformity to the network protocol RFCs [2, 3], the only way for an acceleration on today's end systems is the usage of special hardware. As shown in [1], the extraction of the TCP receive data path and the send path from the software stack and the implementation onto specialized hardware is necessary to get line speed and relieve the host CPU.

5 Related Work

Much attention has been focused on the design of interconnection networks, network interfaces, and even fast messaging layers. As a consequence of poor TCP performance, many transport layer alternatives were proposed as well. However, due to success of TCP those alternatives are not widely used. Hence, understanding TCP behaviour and performance characteristics is an important step towards reducing overheads in software implementations and optimizing utilization of today's high speed networks. An important aspect is to precisely measure required CPU cycles and the delays due to protocol processing.

A light weight messaging layer is proposed in [6]. It presents a detailed analysis and a correlation of the cost analysis with network hardware features which identifies the key cost components and attributes them to specific user communication services. It suggests that the majority of protocol processing could be avoided if the network could provide properties like in order delivery. However, such assumptions are only valid for local area networks.

A performance analysis for a FDDI network is described in [7]. It classifies overhead categories into data touching and non-data touching overheads. Furthermore, it presents UDP and TCP processing overhead times for a Berkeley Unix derived software stack implementation. Timing measurements are provided for protocol processing functions like checksum, data movement, error checking, and so forth. In contrast, our measurements provide timing details based on communication layers including queuing times. The MTU for FDDI is 4352 bytes. This is significantly larger than a standard Ethernet MTU. Furthermore, we evaluated higher speed networks and PC systems. In [8] the authors extend their measurements by providing optimization approaches especially for copy operations and checksum computations.

6 Conclusions and Future Work

Supporting high performance networks and advanced services at the same time presents a challange for today's protocol implementation architectures. Even high end systems cannot fully utilize Gigabit networks. This was proven by real time measurements of the protocol processing of the receive data path as well as the send data path. To reduce the overall processing time, especially, when receiving data, the TCP layer has to be accelerated particulary.

Based on these results, we will analyze additional network protocols based on TCP and/or IP like iSCSI and IPSec.

References:

[1] Mirko Benz and Ralf Lehmann. TCP Acceleration based on Network Processors. In *SDA 2002 – Workshop on System Design Automation*, pages 91–100, Pirna, Germany, 2002.

[2] Defense Advanced Research Project Agency. *Internet Protocol*, 1981. RFC 791.

[3] Defense Advanced Research Project Agency. *Transmission Control Protocol*, 1981. RFC 793.

[4] HP Information Networks Division. *Netperf: A Network Performance Benchmark – http://www.cup.hp.com/netperf/NetperfPage.html*. Internet WWW document.

[5] IBM Linux Technology Center. *Dynamic Probes for Linux – http://oss.software.ibm.com/developer/opensource/linux/projects/dprobes/*, 2002. Internet WWW document.

[6] V. Karamcheti and A. A. Chien. Software overhead in messaging layers: Where does the time go? *ACM SIGPLAN Notices*, 29(11):51–60, Nov. 1994.

[7] J. Kay and J. Pasquale. The Importance of Non-Data Touching Overheads in TCP/IP. In *1993 SIGCOMM*, pages 259–268, San Francisco, CA, 1993.

[8] Jonathan Kay and Joseph Pasquale. Profiling and Reducing Processing Overheads in TCP/IP. *IEEE/ACM Transactions on Networking*, 4(6):817–828, 1996.

[9] Linux Kernel Source. *skbuff.h, skbuff.c*, 2001.

[10] Opersys. *The Linux Trace Toolkit – http://www.opersys.com/LTT/*, 2002. Internet WWW document.

[11] R. Russell and M. Chavan. Fast Kernel Tracing: A Performance Evaluation Tool for Linux, 2001.