Proceedings of the 8th International Conference on Applied Informatics
Eger, Hungary, January 27–30, 2010. Vol. 2. pp. 237–245.

Timestamp-resolution Problem of Traffic Analysis on High Speed Networks

Peter Orosz, Tamas Skopko

Faculty of Informatics, University of Debrecen, Debrecen, Hungary
e-mail: `oroszp@unideb.hu, skopkot@unideb.hu`

Abstract

Tcpdump and Wireshark are commonly used tools for analyzing network traffic between two communicating endpoints over the internet. Capture capabilities of these tools are based on the libpcap library. This library's 1.0.x version only supports 10^{-6} second native timestamp resolution, however on Gigabit Ethernet and faster network speeds nanosecond resolution would be preferred.

Timestamp generation precision depends on other factors too, like kernel packet handling, network driver operation, clock source etc.

In our paper we will show why nanosecond resolution is necessary on Gigabit Ethernet or higher speed networks and the crucial parts of the Linux kernel infrastructure in point of timestamp generation.

Keywords: libpcap; timestamp resolution; inter-arrival time; linux kernel; high speed network

1. Introduction

As network speeds get faster more and more studies show that packet timestamp resolution should be more precise [1][2][3]. Because of the lack of the necessary software infrastructure and/or computing power most of the implementation ended up with hardware-based capturing solutions that provide enough precision for todays 1Gbps and 10Gbps link speeds [1][3].

Libpcap library - the heart of widely used capture applications like tcpdump and Wireshark - currently supports microsecond precision for sub-second information. By targeting the nanosecond precision, this paper is focusing on the possibility of a universal, purely software-based solution for computers running Linux OS and capture utilities based on the libpcap library [4].

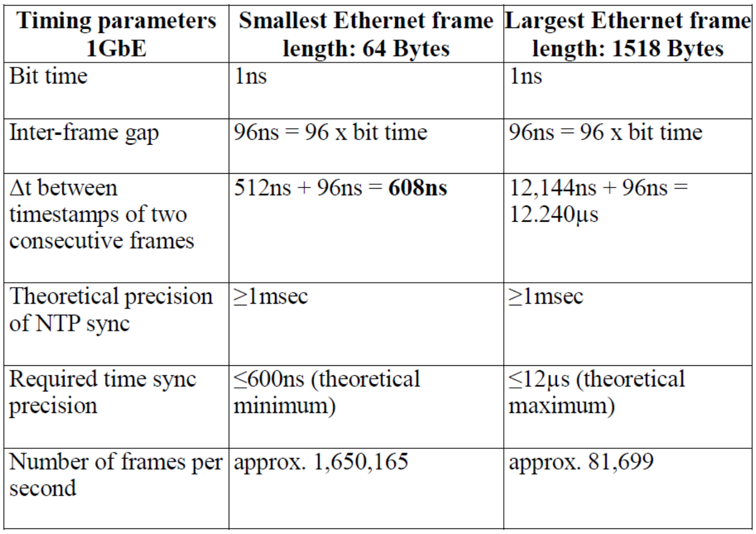
2. Background

One of the most important trace parameters is timestamp that represents a unique time moment when a frame is transmitted or arrives. Packet delay of the consecutive frames could be easily determined. However, inter-arrival time (Δt) must conserve the correct time domain relation between two consecutive frames [7].

3. Problem Definition

3.1. Link speed

The higher the transfer rate the higher time resolution is required for timestamping packets. Table 1 shows accuracy requirements at 1 Gbps link speed at full rate. We can consider that timestamp deltas between two 1518-byte consecutive frames are in the microsecond domain, however transmission of 64-byte frames claims nanosecond precision resolution, notably 608 ns.



Timing parameters 1GbE	**Smallest Ethernet frame length: 64 Bytes**	**Largest Ethernet frame length: 1518 Bytes**
Bit time	1ns	1ns
Inter-frame gap	96ns = 96 x bit time	96ns = 96 x bit time
Δt between timestamps of two consecutive frames	512ns + 96ns = **608ns**	12,144ns + 96ns = 12.240μs
Theoretical precision of NTP sync	≥1msec	≥1msec
Required time sync precision	≤600ns (theoretical minimum)	≤12μs (theoretical maximum)
Number of frames per second	approx. 1,650,165	approx. 81,699

Table 1: Gigabit Ethernet time parameters

As shown by Table 2 at 10 Gbps transmission rate these prerequisites are even higher: approx. 60 ns resolution for 64-Byte frames.

Timing parameters 10GbE	**Smallest Ethernet frame length: 64 Bytes**	**Largest Ethernet frame length: 1518 Bytes**
Bit time	.1ns	.1ns
Inter-frame gap	9.6ns = 96 x bit time	9.6ns = 96 x bit time
Δt between timestamps of two consecutive frames	51.2ns + 9.6ns = **60.8ns**	1,214.4ns + 9.6ns = 1.224µs
Theoretical precision of NTP sync	≥1msec	≥1msec
Required time sync precision	≤60.0ns (theoretical minimum)	≤1.2µs (theoretical maximum)
Number of frames per second	approx. 16,501,650	approx. 816,990

Table 2: Ten Gigabit Ethernet time parameters

3.2. NIC driver architecture

A network device driver can notify the kernel about packet reception using interrupts. This can be done after every received packets or after receiving a specified number of packets. At low network traffic this method might be efficient enough but when network load gets higher the system could get overloaded. Linux kernel provides another method for getting received packets from device driver to kernel space called polling mode. This eliminates the need for using interrupts by querying the driver about received packets time to time.

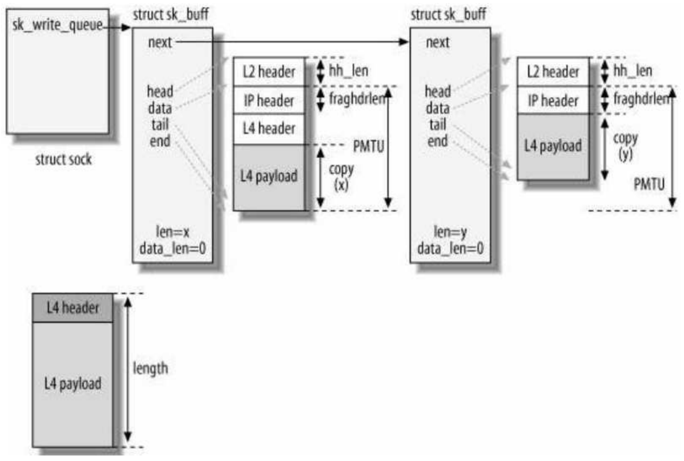


Figure 1: Sk_buff structure of the linux kernel

An intelligent network driver design combines this two modes using the kernel feature NAPI: at lower traffic it uses interrupts, at higher load it switches to polling mode.

3.3. The OS kernel

Received packets by the network driver are enqued by the kernel queue handler. The *sk_ buff* data structure is used to store information about the packets enqueued. This stucture has the tstamp member - a 64-bit integer - to store enqueuing time information. Its higher 32-bit part is used to store the number of seconds, the lower 32-bit part is used to store subsecond information. Its length makes possible to represent time in the nanosecond domain.

Timestamps can be assigned at two different points: at physical reception by the device driver or at enqueueing. The first case is out of our scope because it is based on jiffies. They are the kernel heartbeats with a maximum of 1000 Hz therefore they can provide time information in 1 ms resolution only. The second method seems to be more relevant to us because it represents when packets are made available for further processing.

3.4. Packet Delay Variation

IP Packet Delay Variation (IPDV) is an IETF RFC 3393 proposal [6][7].

$$\mathrm{dT}_2 - \mathrm{dT}_1 = \mathrm{ddT}$$

where

d_{T1} ... delay of a packet sent at time T1
d_{T2} ... delay of a packet sent at time T2
d_{dT} ... type-P-one-way-ipdv from Src to Dst

Delay-per-hop

$$d_H = d_t + d_p + d_q$$

where

d_H ... delay per hop
d_t ... transmission delay
d_p ... processing delay (in the router)
d_q ... queuing delay (in the router)

$$\mathrm{dT} = \sum_{i=1}^{n} d_{H_i}$$

End-to-end delay:
where

d_T ... end-to-end delay
d_H... delay per hop

n . . . number of hops
We assume that IPDV has a Gamma distribution function

$$f(x;k;\theta) = x^{k-1}\frac{e^{-x/\theta}}{\theta^k\Gamma(k)} \text{where } x, k, \theta > 0$$

64-byte packets have been generated according to the Gamma distribution on a 1 Gbps connection (Fig. 2).

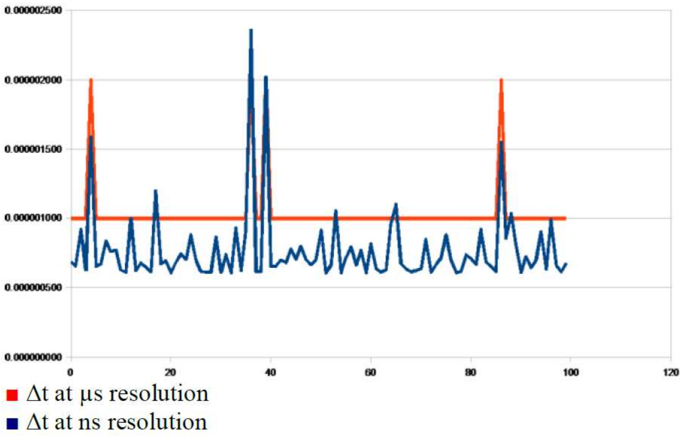


Figure 2: Gamma distributed PDV of 64-byte frames at 1Gbps transmission rate

It can easily be shown out that microsecond time resolution could be insufficient to describe the time domain relation between packet arrivals on Gbps or higher speed network path [5][8].

3.5. Libpcap

Commonly used network traffic capture programs like tcpdump are based on libpcap. This library implements functions to get packets dequeued from the kernel but it handles time information in the microsecond time domain only.

Under Linux, it can use Linux MMAP support (memory mapping) for doing a more "virtual tapping" by reducing memory block copies. When this feature is not available, the SIOCGSTAMP IOCTL call is used to query timestamp information which is less efficient.

3.6. Time synchronization

Let us assume that we monitor network traffic at both end of a 1Gbps path. In order to precisely reproduce time domain relation between consecutive packets, endpoints clocks have to be time-syncronized within a 600 ns precision during the capture. 10Gbps requires 60 ns precision accordingly.

Requirement for end-to-end time syncronization precision is proportional to the connection speed of the link.

Alternative sync mechanisms for the 1000 to 10 ns time domain, as shown by Fig 3.:

- Network Time Protocol. At its best NTP could keep the time sync between within 1 ms for two IP host [12]. Not suitable for nanosecond resolution. A kernel level implementation can be subject of study.
- National Semiconductor Phyter (IEEE 1588 V1 and V2 compatible). The PTP protocol with hardware support [12]. It is suitable since its 10 ns accuracy.
- GPS: Endace boards use this method. Their 7.5 ns accuracy make them suitable.

When measurement scenario makes it possible, custom sync frame could be used over a dedicated low latency LAN environment built between the nodes and a time server for time synchronization. Theoretical conditions for accurate syncing:

- The nodes are connected via a network hub to the time server.
- All cabling to be the same length and quality.
- All NIC cards used for syncing should by the same type.
- Hardware and OS environments preferrably uniform for possibly equivalent packet paths and processing times.

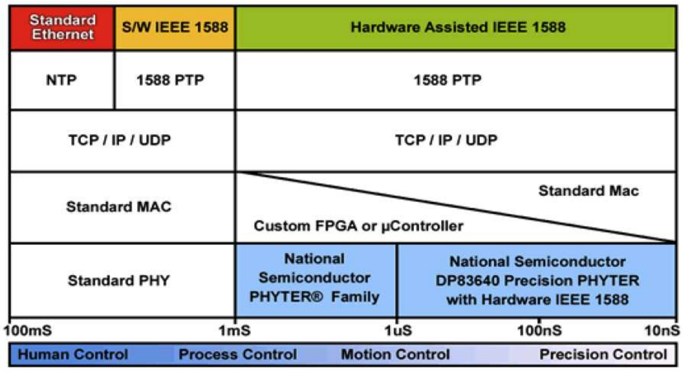


Table 3: Efficiency of different time synchronization methods

4. How to take high resolution traces

FPGA-based capture cards are very efficient. Hardware solutions like Endace DAG boards provide rather good resolution but are expensive. They provide very accurate clock synchronization by using GPS signals.

Interface cards featuring the National Semiconductor DP83640 Phyterare are good values for the money too since they are IEEE 1588 compatible.

Optical only applications can tap the connection physically by splitting the fiber.

Software solutions are cheap and easily accessible if nanosecond resolution could be worked out. That's why our research moves that direction.

5. Benefits of higher resolution

More accurate timestamping helps to better review data traversal between network layers, adjust required buffer sizes to multimedia applications. Moreover, IPDV could be observed more precisely, especially on 10Gbps or faster connections, if timestamp accuracy is in correlation with the hardware-based solutions.

6. An advanced software-based approcach

Our main aim was to reveal and test all of the kernel functions and features that will be essential elements in our project to enhance libpcap to a nanosecond-capable capture library.

It is resonable to reach nanosecond resolution purely on software-based tapping:

- *tstamp* member of *sk_ buff* structure is capable of nanosecond resolution
- Linux kernel function *ktime_ get_ real()* to query system clock in nanosecond resolution
- This function is adequate to fill up nanosecond *tstamp* fields in *sk_ buff.*
- Accordingly user-space applications (such as libpcap-based ones) could display/process 10^{-9} second resolution timestamps.
- For cost-efficient time synchronization dedicated LAN interfaces and a PTP timing protocol could be used within a low latency wired environment.

Input queue handler within the 2.6 kernel puts a 64-bit timestamp to each enqueued frame.

Linux kernel API features *ktime_ get_ real()* function that enables us to retrieve nanosecond resolution timestamps from the kernel.

For ns resolution we assume that main CPU is operates at least at 1GHz master clock speed.

```
include/linux/sk_buff.h:

struct sk_buff {
        /*These two members must be first.*/
        struct sk_buff          *next;
        struct sk_buff          *prev;

        struct sock             *sk;
        ktime_t                 tstamp;
        struct net_device       *dev;
(...)
}

include/linux/ktime.h:

union ktime {
        s64     tv64;
#if BITS_PER_LONG != 64 &&
!defined(CONFIG_KTIME_SCALAR)
        struct {
# ifdef __BIG_ENDIAN
        s32     sec, nsec;
# else
        s32     nsec, sec;
# endif
        } tv;
#endif
};

typedef union ktime ktime_t;
```

7. Summary

High resolution traces help to analyze high performance networking protocols and multimedia applications. Our key achievement is to collect all the corresponding kernel functions and features that are essential to enhance libpcap a nanosecond-capable capture library. However this software-only method has to be validated against hardware-based solutions by making simultaneous traces.

References

[1] The DAG project, http://dag.cs.waikato.ac.nz

[2] Attila Pásztor, Darryl Veitch, PC based precision timing without GPS, Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, June 15-19, 2002, Marina Del Rey, California, USA

[3] Cace TurboCap network interface card, http://www.cacetech.com/products/turbocap.html

[4] Libpcap, a common open source packet capture library for Unices, `http://www.tcpdump.org/`

[5] Gianluca Iannaccone, Christophe Diot, Ian Graham, Nick McKeown, Monitoring very high speed links, Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, November 01-02, 2001, San Francisco, California, USA

[6] IETF RFC2679, A one-way delay metric for IPPM, `http://www.ietf.org/rfc/rfc2679.txt`

[7] IETF 3393, IP Packet Delay Variation Metric for IPPM, `http://www.ietf.org/rfc/rfc3393.txt`

[8] Jörg Micheel, Stephen Donnelly, Ian Graham, Precision timestamping of network packets, Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, November 01-02, 2001, San Francisco, California, USA

[9] TSC, `http://en.wikipedia.org/wiki/Time_Stamp_Counter`

[10] Wireshark, http://www.wireshark.org/

[11] Christian Benvenuti, Understanding Linux Network Internals, O'Reilly, 2006

[12] Precision PHYTER - IEEE 1588 Precision Time Protocol Transceiver, `http://www.national.com/pf/DP/DP83640.html`