High-Resolution Hardware-based Packet Capture with Higher-Layer Pass-Through on NetFPGA Card

Yaovi E. Kwasi and Roberto Rojas-Cessa
Networking Research Laboratory
Department of Electrical and Computer Engineering
New Jersey Institute of Technology
Newark, NJ 07102
{yk73, rojas}@njit.edu

***Abstract*—Packet capture for measurement of several network parameters on high-speed networks requires high clock resolution and the capability of analyzing different protocol functions for behavioral functionality. Herein, we present a practical packet-capture tool that time-stamps packet arrival at the data-link layer with nanosecond resolution and with the capability of forwarding packets to the application layer in real time. Our tool can be used to analyze both network performance and behavioral functionality for high-speed networks. We build this tool on a NetFPGA card and with minimum dependencies on off-line data processing. Different from existing specialized packet-capture cards, our card allows packets to be time-stamped at both the data-link and application layers. We present the design and evaluation of our proposed tool. We tested the accuracy of our tool and compared it to that of a commercial specialized packet-capture card, and show that they are equivalent. Additionally, our tool provides different functionality.**

***Index Terms*—packet capture, hardware speed, software sniffer, high resolution, pass through, high speed**

I. Introduction

Packet capture has shown to be a fundamental tool for observing and analyzing network performance and behavioral functionality. Most popular versions of packet capture tools are implemented in software [1], [2]. These software-based tools provide a very complete classification of protocols. This feature makes it simple and flexible to use in a large number of applications for network traffic analysis. These tools largely use the libcap library, which lists a large number of protocols to which captured packets may be classified into [3].

The performance of software-based packet capture tools greatly depends on the speed in which the operating system is executed. Therefore, it depends on the processing speed of the host workstation [4]. The application of such tools may then be limited to analyze traffic on links with modest speed rates and functional operation of protocols. High-performance workstations are then needed to keep up with the rapid increase of the transmission rates. Direct Memory Access (DMA) is widely used to speedup packet capture [5]. In efforts to increase the performance, software-based implementations with smaller dependence on the OS have been considered [6]–[8]. Software based solutions are widely used for behavior analysis (e.g., protocols monitoring, network forensics) [9].

The elimination of kernel livelock while processing interrupts [6] and the use of device polling [7] have significantly improved the performance of software-based packet capture, and at the same time, eliminated the risk of significantly slowing the speed of the operating system during packet capture at high transmission speeds. Other solutions adopt new application programming interfaces (APIs), such as nCAP [8], to minimize kernel intervention and enable a straight path from the network adapter to user space, increasing flexibility and performance.

In the measurement of network parameters, such as available bandwidth, one-way delay, and jitter among others, the time resolution in packet capture is critical [10], [11]. Operating systems have a clock resolution of up 1 μs and fast rates may require clock resolution in the nanosecond range. Furthermore, a workstation may take additional time to perform time-stamping as part of the packet processing time [12] in response to a system interrupt.

While DMA releases the central processing unit (CPU) from spending resources and time on storing packet descriptors, access to shared buses may still add some processing delays, as assignment of time stamps may take a number of instructions to execute [13]. Therefore, hardware-based packet capture has been considered as an alternative to high-resolution clock time-stamping [14]–[16]. However, access to the packet capture (and to the time stamps) on hardware-based capture tools is performed off-line as the application layer is fully disassociated from participating in the capture. One of the reasons is that most packets are encapsulated into another packet to preserve the high-resolution time stamp (provide by hardware) and the original packets. Then the application layer cannot provide useful time stamps. While the approach provides high-resolution time stamps, it also disables the application layer from real-time analysis. Moreover, this approach also requires additional software for data adaptation.

In this paper, we introduce a different approach, a packet capture tool that time-stamps packet arrival with high-resolution, and yet, packets are forwarded to the applications layer, without delays, for real-time analysis. Our packet capture tool can be used in combination with any software packet capture (e.g., Tcpdump [1] or Wireshark [17]), and with

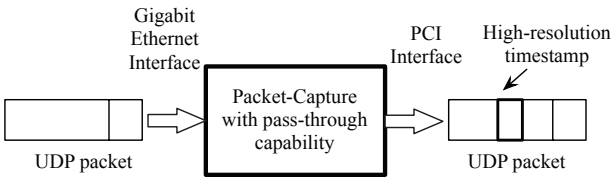

Fig. 1. High-resolution time-stamping packet-capture tool with pass-through capability.

minimum off-line data processing. Our tool allows users get to observe time-stamping at the data-link and application layers. We implement our packet capture tool on a programmable card, NetFPGA [18]. We present time-stamping accuracy tests of our packet capture tool and show that our tool provides equivalent accuracy to existing hardware-based packet capture cards, and it also provides additional functionality.

The remainder of this paper is organized as follows. Section II introduces our proposed and developed packet capture tool. Section III presents the experimental performance evaluation of the proposed hardware-capture tool. Section IV presents our conclusions.

II. Pass-through and Concatenating Hardware-based Packet Capture

The objective of our packet-capture tool is to achieve the maximum accuracy in time-stamping packet arrivals, and at the same time, to allow packets continue their trip to the application layer. We call the latter feature as *pass through* capability. This feature allows the user to time-stamp a packet arrival at both the data-link layer (actual arrival time) and that at the application layer (software arrival time). We perform the capture and time-stamping at the data-link layer, executed in hardware. At the same time, the pass-through feature makes our tool independent of requiring coupling software to convert the capture information into a readable format [19]. We use a NetFPGA card for the implementation of our packet-capture tool. Our implementation comprises hardware design for the NetFPGA card, using two modules: a time-stamp module and a new module, called *monitoring module*. In the general data path, packets are received at the data-link layer, time-stamped, and then forwarded to the application layer of the hosting workstation for identification of protocol and fields. In order to show the high-resolution time stamp and to keep the original expected application-layer time stamp, the packet length must remain as in the original packet. We, therefore, swap 64 bits of payload of the packet for the high-resolution time stamp. If payloads need to be analyzed, our card can provide an additional output with the original packets for further analysis. However, that is out of the scope of this paper.

Once packets are received, they are forwarded to the application layer. We use Wireshark [17] to display captured packets and to display the time stamp assigned by the NetFPGA card.

The NetFPGA card is a network hardware accelerator for network applications that require specialized packet handling or high performance processing [18], [20]. These cards are offered in two models: 1 and 10 Gb/s. In this paper, we use the 1 Gb/s version. A NetFPGA card comes in the form of a plug-in card with four ports, a local Static RAM (SRAM), and Dynamic RAM (DRAM) for local processing. The NetFPGA card attaches to the Peripheral Communication Interconnect (PCI) bus of the host workstation. This card has four full duplex 1-Gb/s ports, where each of them has receiving and transmission queues. Internally, there is an input arbiter block, used to determine the port being accessed, an output-port lookup block, to determine which output is used to forward a packet, and an output-queues block, where packets that are forwarded to an output are stored while waiting for service, if needed. In our packet capture implementation, we design the different modules of our design as a single block that is inserted between the output-port look and output-queue blocks, as Figure 2 shows.

A. Time-stamp Module

We make use of the time-stamp module of the Monitoring System Project [19], [21]. The time-stamp module is placed before the receiving (RX) queues so that incoming packets are timestamped as soon as they arrive. The reference clock of the Time-stamp Module is the one the NetFPGA board uses for local clocking. This approach reduces inaccuracy in the time stamp generated for the received packets. Because the Ethernet preamble has variable length, the packets are timestamped as soon as the start-of-frame delimiter is detected. Time stamps are generated from a 64-bit counter, driven by the 125 MHz NetFPGA clock, giving one count increment every 8 ns. Therefore, we divide our time stamps by 8 to obtain a 1 ns resolution. Because a division operation is clock-consuming and complex to implement in hardware, we recur to Direct Digital Synthesis (DDS) [22], [23]. DDS rate is a 32-bit number used to generate a *synthetic frequency* (f_s) from the *crystal frequency* (f_c), as (1) indicates:

$$f_s = \frac{f_c \; DDS}{2^{32}} \tag{1}$$

At each edge of the crystal-frequency signal, a 32-bit adder adds the DDS rate number to the accumulated value. The overflow output from the adder is used to toggle the chip-enable input on the time-stamp counter, incrementing it. The output of the counter is f_s. As an example, if the desired output frequency is exactly half the input frequency, the DDS rate is set to 0x80000000. On the first edge, the accumulated value increases from 0 to 0x80000000 and the counter is disabled. On the next signal edge, the accumulated value wraps around to zero and the counter is enabled, causing the counter to count at half the input frequency; thereby dividing the input signal by two. As in the monitoring system project [19], the value 0x8970_5F41 is selected as the DDS rate. This value makes the clock to increment the time-stamp counter, one count every 14.9 ns. By shifting left by 6 bits, each time stamp becomes a fixed-point, and the 64-bit representation of time holds the seconds in the upper 32 bits, and fraction of a second in the lower 32 bits.

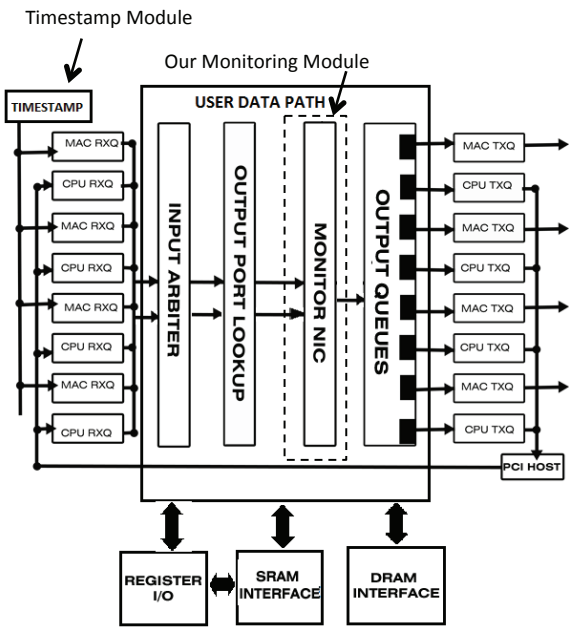

Fig. 2. NetFPGA architecture and our proposed monitoring module.

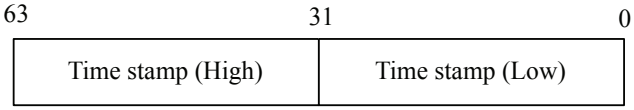

Fig. 3. Time stamp format.

B. Monitoring Module

The monitoring module identifies the payload of the UDP packets and replaces the first 64 bits of data with the time stamp of the current packet. The data path of the NetFPGA is 64 bits wide (i.e., 64 bits of packet data are forwarded from module to module every clock cycle). Control words of 8 bits are also sent in parallel through the control bus to identify the type of data being sent. In order to detect the payload field of a packet, we count the number of clock cycles of the monitoring module to identify the current portion of the passing packet. Figure 4 shows where the time stamp is placed in a User Defined Protocol (UDP) datagram, and the portions of the frame accessed each clock cycle (see right-side labels in the figure).

The counter has five states (the first state is in Figure 5) to analyze the data as it passes through the monitoring module. NetFPGA appends a module header to the frame whenever it moves from one module to another. If the control word (in_fifo_ctrl_dout in the code) is 255 (or 0xFF), the part of the data currently in the module represents the module header appended to the frame. A counter indicates the portion of a frame and the fields that pass through the module. For example, the start-of-frame is detected at the zero count. At this time, we store the time stamp of the packet in the dedicated registers of the module. At this time, the first 64 bits of ethernet header pass through the module.

When the count is 1, the 14 bytes of Ethernet header and

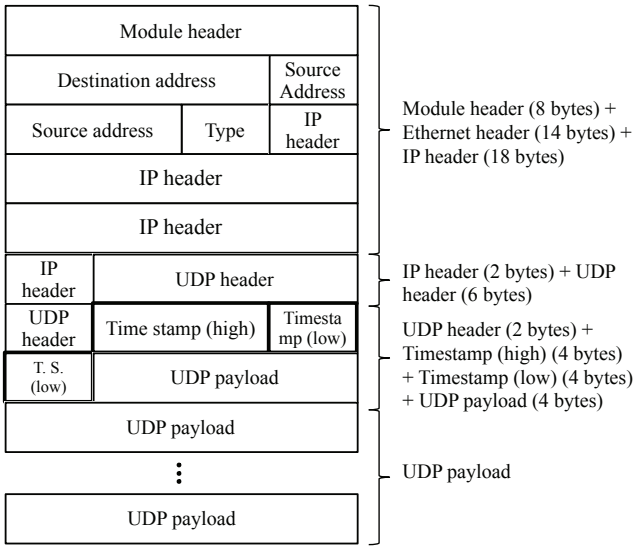

Fig. 4. Example of structure of UDP datagram after adding the time stamp.

```
case(state)

    PROCESS_CTRL_HDR: begin
    // Pass all control headers through unmodified
        if (control_word == 0) begin
         store_timestamp_from_previous_module()
         next_state = ETH_IP_HDR
         count = count + 1
        end
    end // case: PROCESS_CTRL_HDR
```

Fig. 5. Pseudo code of pass-through analysis section.

the first 16 bits of the IP header traverse the module. When the count becomes two, the following 64 bits of the IP header pass through. Similarly, when the count becomes 3, 64 more bits of the IP header data pass through module, unmodified. When the counter is 4, 16 bits of IP header and 48 bits of UDP header pass through the module. Once the counter becomes 5, the 16 bits of UDP header and 48 bits of payload pass through the module. Thus, the state of the monitoring state moves to the UDP_HDR_PAYLOAD state, as the pseudo-code in Figure 6 shows.

At this time, the time stamp is placed in the first 64 bits of the UDP payload, and those substituted bits are discarded. This is done through two successive stages as illustrated in the pseudo-code Figure 7 shows. The rest of the payload passes unmodified. Once the monitoring module identifies a

```
ETH_IP_HDR: begin
    count= count + 1
    if (count == 5) begin
        next_state = UDP_HDR_PAYLOAD
    end
end // case: ETH_IP_HDR
```

Fig. 6. Pseudo code of state change at identification of field for time stamp.

```
UDP_HDR_PAYLOAD: begin
      forward_16bit_of_udp_header_unmodified()
      forward_first_48_bit of timestamp()
      next_state = PAYLOAD_TWO
end // case: UDP_HDR_PAYLOAD

   PAYLOAD_TWO: begin
      forward_remaining_16bit_timestamp()
      forward_48bit_of_payload()
     next_state = PAYLOAD_THREE
end // case: PAYLOAD_TWO
```

Fig. 7. Pseudo code of placement of time stamp.

```
     PAYLOAD_THREE: begin
              //Detect new frame
            if (control_word != 0) begin
               next_state = PROCESS_CTRL_HDR
               count = 1
            end
    end // case: PAYLOAD_THREE
end case
```

Fig. 8. Pseudo code of return to start state.

new frame, the state machine goes into the initial processing state and continue as before (see Figure 8).

C. Supporting Software

The software part of the system, a Perl program, is used to adjust the DDS rate without making reference to an external time base. The main difference between our implementation and the implementation in [19] is that we don't create a new packet from each captured packet. The new packets are generated to carry the time stamp appended to the original packet. In that approach, the user must first patch the pcap library [24] in order for the packet capturing software (e.g., Tcpdump) to identify the packets format. There are additional correction measures added to that, and these may change according the Linux distribution used. In contrast, our approach is simple and easy to use as we obtain the time stamp from the packet (through Wireshark). It is compatible with any distribution of Linux hosting the NetFPGA platform and packet-capture software. While Wireshark puts its own time stamp on the packets, the hardware time stamp is found in the payload. Furthermore, the software extracts the hardware time stamp, corrects it according to the identification of each packet, and display it. In this way, our proposed tool can be used to observe inter-frame gaps assigned by hardware (data-link layer) and software (application layer).

III. Experimental Evaluation

We tested the accuracy of our monitoring tool by comparing the time stamp of packet captured at different rates to those obtained with a specialized hardware capture card, Endace DAG line card [15]. We generated UDP traffic using Iperf [25]. To obtain copies of the packets transmitted between Iperf and the packet capture cards, one for NetFPGA card and another for the DAG card, we use a gigabit network tap (1000-Mb/s Black Box copper tap [26]). The experimental setup is connected as Figures 9 and 10 show. We generated

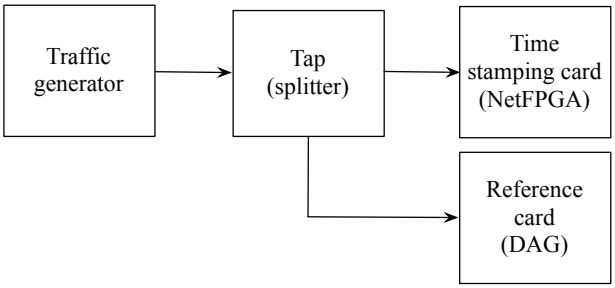

Fig. 9. Diagram of the experimental test setup.

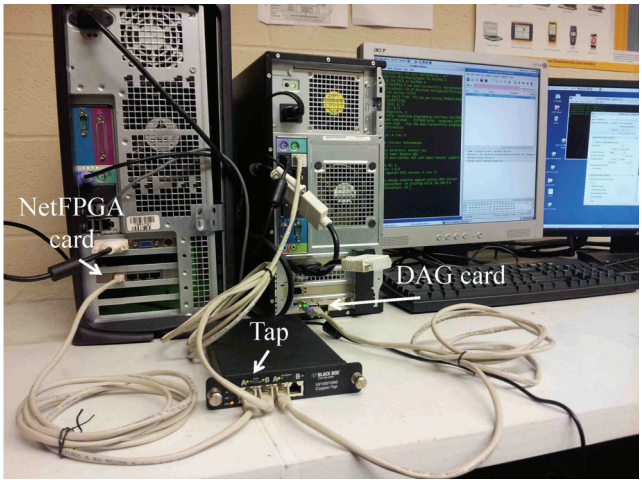

Fig. 10. Picture of the experimental test setup.

UDP packets, each with a length of 1500 bytes, and the inter-packet gaps were modified to achieve different data rates (R), such as $R = \{100,\ 200,\ 400,\ 600,\ and\ 800\}$ Mb/s. Because Iperf is a software-based packet generation tool (and therefore, its performance greatly depends on the specifications of the host workstation), we calibrated the actual data rate that Iperf generates with the DAG card, and noticed that 1000 Mb/s is not achievable on our available workstations. Therefore, we use 800 Mb/s as the maximum data rate. All the measurements are referred to the DAG card capture.

We present the distribution of time stamp variations achieved with our tool, one per each tested data rate. The x axis in the following graphs represents the discrepancies in nanoseconds between the capture time of our tool and the DAG card. The y axis represents the number of packets over the thousand that fall into each discrepancy bin. It should be noted that we use the inter-packet gaps to rescind clock synchronization.

A. Average discrepancy and standard deviation

We list the measured average discrepancies (in reference to the DAG card) and the standard deviations of the collected samples for each tested data rate. The summary of the collected values are presented in Table I.

1) 100 Mb/s flows: The histogram in Figure 11 shows that most of the time stamps of the captured packets are equivalent.

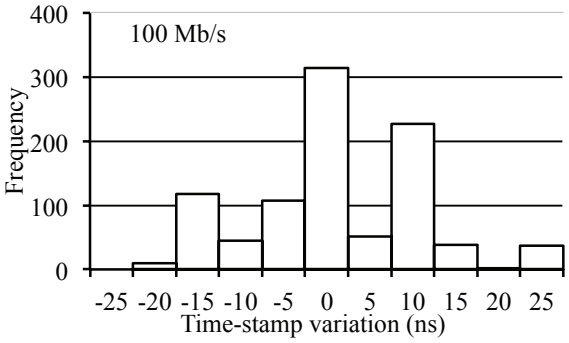

Fig. 11. Distribution of time-stamp discrepancies at 100 Mb/s.

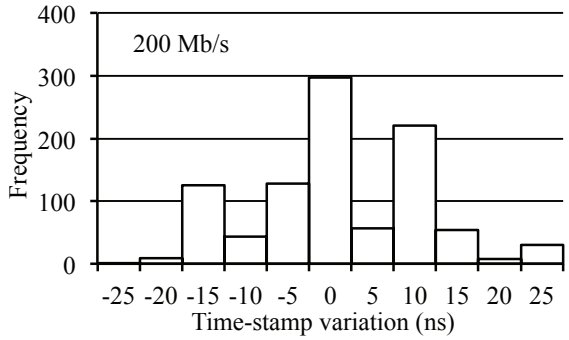

Fig. 12. Distribution of time-stamp discrepancies at 200 Mb/s capture.

About 20% of the time stamps are about 10 nanoseconds longer on the NetFPGA than those of the DAG card, while 4.5% of the time stamps are 10 nanoseconds shorter. The largest discrepancies are $\pm$25 ns. But the number of packets that falls into the ranges of $\pm$25 and $\pm$20 ns are negligible. The average time-stamp difference is -3.49 ns and the standard deviation is 18.8 ns.

2) 200 Mb/s flows: At 200Mb/s, the results obtained are very similar to that at 100 Mb/s, as Figure 12 shows. Although there are more packets that have the same (inter-packet gap) time stamp, 22 % of the packets were captured with a 10-ns discrepancy. In this case the average difference increases to -2 ns and the standard deviation becomes 13.7 ns.

3) 400 Mb/s flows: The distribution of discrepancy shows consistency. However, the distribution around 0 increases. The average delay is -1 ns and the standard deviation is about 11.12 ns.

4) 600 and 800 Mb/s flows: The distribution at 600 and 800 Mbps remains with high consistency. Interestingly, the average discrepancy approaches zero as the transmission rate increases. The average discrepancy approaches to zero. The standard deviations measured at 600 and 800 Mb/s are 9.8 and 9.9 ns, respectively.

B. Transmission rate

We also compared the transmission rate measured with our packet-capture system with the reference DAG card. The measured rates (estimated from the time stamps from the first captured packet to the last) show very similar results. Figure 16 shows the interpolation between the rate measured by the DAG card (x axis) and the rate measured by our packet capture tool (y axis). From the graph, we noticed that the rates are slightly overestimated over 400 and 600 Mb/s. This is expected as the average discrepancies measured are smaller than zero. The measured rates at 600 and 800 Mb/s are practically equivalent, as the average discrepancies approaches to zero, as discussed above.

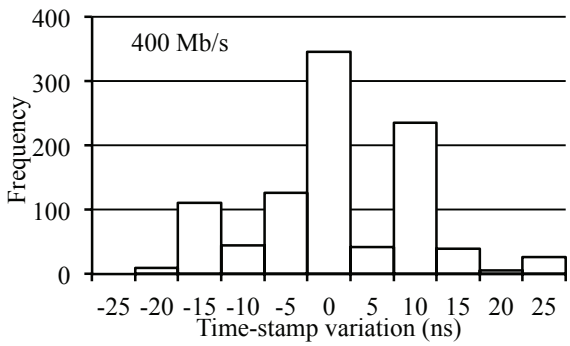

Fig. 13. Distribution of time-stamp discrepancies at 400 Mb/s capture.

TABLE I
TRANSMISSION TIME DISCREPANCY

Data rate (Mb/s)	Avg. discrepancy (ns)	Std (ns)
100	-3.9	18.8
200	-2.0	13.7
400	-1	11.1
600	-0.7	9.8
800	-0.6	9.9

To provide a close-up view of the differences on the measured rates, we graph the error, taking the measurement of the DAG card as reference. Figure 17 shows these results. The graph shows that the error is nevertheless, smaller than 0.005%, which is insignificantly small. The recorded average error between these two systems is 0.0037 %.

IV. CONCLUSIONS

We presented the design and implementation of a high-resolution hardware-based packet capture tool. This tool has the properties of providing data-link- and application-layer time stamps. These features are achieved by providing time stamps through hardware-based capture and by allowing packets continue their trip to the application layer, without de-

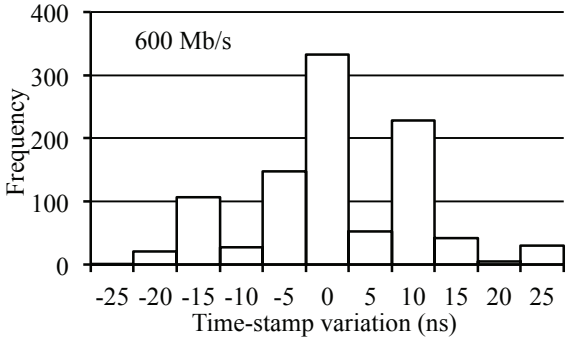

Fig. 14. Distribution of time stamp values at 600 Mb/s capture.

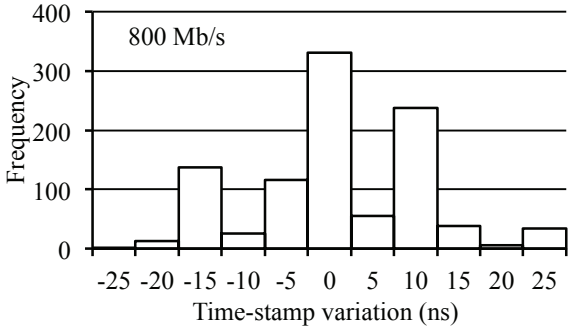

Fig. 15. Distribution of time stamp values at 800 Mb/s capture.

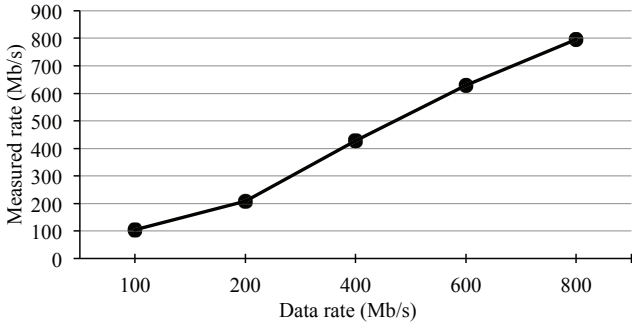

Fig. 16. Achievable maximum rates in packet capture of proposed monitoring tool.

lays, with unmodified packet lengths. We implemented our approach on a NetFPGA card and tested its accuracy. The results show that our tool achieves equivalent time stamping resolution to that of specialized packet cards in the market. Furthermore, it provides additional functionality; desynchronized time stamping at the data-link and application layers.

REFERENCES

[1] V. Jacobson, C. Leres, and S. McCanne, "The tcpdump manual page," *Lawrence Berkeley Laboratory, Berkeley, CA*, 1989.

[2] G. Combs *et al.*, "Wireshark," *Web page: http://www. wireshark. org/last modified*, pp. 12–02, 2007.

[3] V. Jacobson, C. Leres, and S. McCanne, "libpcap, lawrence berkeley laboratory, berkeley, ca," *Initial public release June*, 1994.

[4] F. Schneider and J. Wallerich, "Performance evaluation of packet capturing systems for high-speed networks," in *Proceedings of the 2005 ACM conference on Emerging network experiment and technology*. ACM, 2005, pp. 284–285.

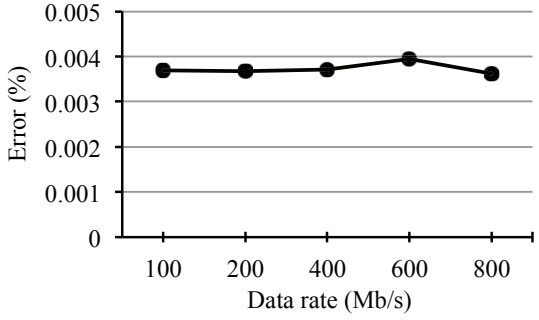

Fig. 17. Error of rate measurement.

[5] R. W. Herrell and T. P. Morrissey, "User scheduled direct memory access using virtual addresses," Apr. 5 1994, US Patent 5,301,287.

[6] J. C. Mogul and K. Ramakrishnan, "Eliminating receive livelock in an interrupt-driven kernel," *ACM Transactions on Computer Systems*, vol. 15, no. 3, pp. 217–252, 1997.

[7] L. Rizzo, "Device polling support for freebsd," in *BSDConEurope Conference*, 2001.

[8] L. Deri, "ncap: Wire-speed packet capture and transmission," in *End-to-End Monitoring Techniques and Services, 2005. Workshop on*. IEEE, 2005, pp. 47–55.

[9] V. Corey, C. Peterman, S. Shearin, M. S. Greenberg, and J. Van Bokkelen, "Network forensics analysis," *Internet Computing, IEEE*, vol. 6, no. 6, pp. 60–66, 2002.

[10] K. M. Salehin and R. Rojas-Cessa, "Active scheme to measure throughput of wireless access link in hybrid wired-wireless network," *Wireless Communications Letters, IEEE*, vol. 1, no. 6, pp. 645–648, 2012.

[11] ——, "Ternary-search-based scheme to measure link available-bandwidth in wired networks," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. IEEE, 2010, pp. 1–5.

[12] ——, "Measurement of packet processing time of an internet host using asynchronous packet capture at the data-link layer," in *IEEE International Conference on Communications*, 2013, p. 5pp.

[13] G. Iannaccone, C. Diot, I. Graham, and N. McKeown, "Monitoring very high speed links," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. ACM, 2001, pp. 267–271.

[14] J. Micheel, S. Donnelly, and I. Graham, "Precision timestamping of network packets," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. ACM, 2001, pp. 273–277.

[15] Endace, "DAG, network monitoring interface." [Online]. Available: http://www. endace. com

[16] T. Wolf, R. Ramaswamy, S. Bunga, and N. Yang, "An architecture for distributed real-time passive network measurement," in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006. 14th IEEE International Symposium on*. IEEE, 2006, pp. 335–344.

[17] G. Combs *et al.*, "Wireshark," *Web page: http://www. wireshark. org/last modified*, pp. 12–02, 2007.

[18] NetFPGA, http://www.NetFPGA.org.

[19] G. Antichi, D. J. Miller, and S. Giordano, "An open-source hardware module for high-speed network monitoring on netfpga," in *European NetFPGA Developers Workshop*, 2010.

[20] G. Watson, N. McKeown, and M. Casado, "Netfpga: A tool for network research and education," in *2nd workshop on Architectural Research using FPGA Platforms (WARFP)*, vol. 3, 2006.

[21] G. Antichi, S. Giordano, D. J. Miller, and A. W. Moore, "Enabling open-source high speed network monitoring on netfpga," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 2012, pp. 1029–1035.

[22] S. F. Donnelly, "High precision timing in passive measurements of data networks," Ph.D. dissertation, Citeseer, 2002.

[23] P. Saul, "Direct digital synthesis," *Circuits and systems tutorials*, p. 393, 1996.

[24] V. Jacobson, C. Leres, and S. McCanne, "pcap-packet capture library," *UNIX man page*, 2001.

[25] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "Iperf: The tcp/udp bandwidth measurement tool," 2005. [Online]. Available: http://dast.nlanr.net/Projects

[26] Black Box Network Services, "10/100/1000 Black Box copper tap." [Online]. Available: http://www. blackbox. com