International Journal of Innovative
Computing, Information and Control
Volume **16**, Number **3**, June **2020**

 ISSN 1349-4198
pp. **1035–1046**

PACKET LENGTH AWARE TIMESTAMPING MECHANISM FOR DPDK BURST-ORIENTED PACKET PROCESSING

Chuanhong Li[1,2], Xuewen Zeng[1,2] and Lei Song[1,2]

[1]National Network New Media Engineering Research Center
Institute of Acoustics, Chinese Academy of Sciences
No. 21, North 4th Ring Road, Haidian District, Beijing 100190, P. R. China
{ lich; zengxw; songl }@dsp.ac.cn

[2]School of Electronic, Electrical and Communication Engineering
University of Chinese Academy of Sciences
No. 19(A), Yuquan Road, Shijingshan District, Beijing 100049, P. R. China

Received October 2019; revised February 2020

Abstract. *DPDK burst-oriented packet processing boosts up the performance of packet capture at the cost of packet timestamp accuracy. Being blind to packet timestamp deviates from the goal of monitoring since it has a significant effort on network traffic monitoring and analysis. In this paper, we analyze packet timestamp of DPDK burst-oriented packet processing and model the cumulative time gaps of packet timestamp in the same burst. Then, we demonstrate that timestamping packet in a 1-by-1 fashion after acquiring system time is not suitable for DPDK as the cumulative time gaps depending on packets number in a burst and subsequent operations on each packet make it uncontrollable. At last, a packet length aware timestamping mechanism is proposed and demonstrated that it can conform to the theoretical model we give. Experiments show the rationality and effectiveness of our proposed timestamping mechanism.*
Keywords: Timestamp accuracy, DPDK burst-oriented packet processing, Timestamping mechanism, Cumulative time gaps

1. **Introduction.** To provide good quality of service and ensure users to access the Internet safely, network traffic monitoring and analysis are becoming more and more significant. Networked control systems are widely used in the industrial Internet as the monitoring and control system [1]. In the last few years, network traffic has been increasing exponentially, which presents big challenges to network traffic monitoring and analysis. A few years ago, traffic monitoring at rates ranging from 100 Mb/s to 1 Gb/s was considered a challenge, while now 10 Gb/s NIC is a common feature of routers [2]. Packet Capture is the core activity of network monitoring [3]. It is not an easy task to capture packets at 10 GbE network based on general-purpose operating systems without specific hardware support since they are optimized for compatibility and stability rather than high performance or high precision [4,5].

Fortunately, the emergence of some novel packet I/O frameworks adopting new technologies such as zero-copy, kernel bypass, I/O batch processing makes the performance of packet capture engines improved greatly. DPDK is an outstanding representative of them. With burst-oriented functions used, DPDK boosts up the performance of packet I/O engines.

With the performance of packet capture improved, however, most of the packet I/O frameworks choose to pay no attention to packets' timestamp, although, it is significant

DOI: 10.24507/ijicic.16.03.1035

in traffic monitoring and analysis. For example, passive network monitoring requires not only capturing packets but also labeling them with their arrival timestamps [14]. Actually, the timestamp accuracy is important to many monitoring applications, especially for those services with a temporal pattern. Timestamping the arrival of packets allows the investigation to obtain the original traffic time-profile on the network link [6] and when an event occurs, the more accuracy the timestamp of the packet is, the better it is for locating the problem [7]. Besides, when cooperated with software packet generators for latency measurements, the timestamp accuracy of packets plays a critical role in packet I/O frameworks [8]. Even worse, batch processing attributes to timestamp inaccuracy. For DPDK, burst-oriented functions are used to speed up packet processing and how to timestamp packets in a burst is essential for subsequent analysis. Unexpectedly, this phenomenon has not received attention to date.

Therefore, in this paper, we focus on DPDK burst-oriented packet processing and the timestamping mechanism of packets in a burst. Based on the practical implements in our project, a packet length aware timestamping mechanism is proposed which is proved a rational and effective method to solve the above problem. In summary, the contribution of our work includes:

- We analyze the burst-oriented packet processing for DPDK and a theoretical time gap upper bound of the packets' timestamp in the same burst is modeled;
- We prove that timestamping packet in a 1-by-1 fashion after acquiring system time is not suitable for DPDK since the cumulative time gaps depending on packets number in a burst and subsequent operations on each packet make it uncontrollable;
- A packet length aware timestamping mechanism is proposed, and it is demonstrated that it always conforms to the theoretical model we give.

The rest of this paper is organized as follows. Section 2 describes some related work. Section 3 gives some descriptions about DPDK and presents the problem statement, which analyzes the timestamping mechanism for DPDK burst-oriented packet processing. The packet length aware timestamping mechanism is proposed in Section 4. Section 5 shows the result of the proposed method. Finally, we conclude our paper with some discussions in Section 6.

2. **Related Work.** The role the packet's timestamp plays is becoming more and more significant in modern communication networks [7]. To timestamp packets as close as it arrives is useful to maintain the original traffic time-profile on the network link which plays an important role in traffic monitoring and analysis [6]. As network functions become more and more complex, the time resolution should be higher which has been realized in the last few years [9-11]. However, most of them are based on specialized hardware support to timestamp packets, such as NetFPGA [12], Endace DAG cards [9] and Natpatch [13].

The expensive price and poorly scalable for specialized hardware-based methods make the research community start to explore the use of commodity hardware together with only-software solutions as a more flexible and economical choice [14].

With the advent of open source, new novel packet I/O engines have sprung up leading to capture packets on 10GbE or more network based on commodity hardware [19]. Netmap [20], a new packet I/O engine proposed by Rizzo, enables packet processing at 10 Gbit/s link on the commodity operating system. MoonGen [21], a flexible high-speed packet generator developed by Emmerich et al., uses DPDK as the underlying framework. It can saturate 10 GbE links with minimum sized packets using only one core. However, all of them do not pay more attention to the packet timestamp.

A software solution based on the libpcap [27] packet capture library and high precision kernel-based timestamp generation is studied by Orosz and Skopko [16]. By some modifications and additions to the original code of libpcap, they can provide nanosecond precision timestamping, but the authors do not consider timestamp inaccuracy incurred by batch processing.

Moreno et al. propose two different approaches to distribute inter-batch time among the packets belonging to the same batch [14]. However, this method depends on the link load and when the link load is lower, poorer results will get since they distribute real inter-packets gaps among all packets in the batch. And a driver modification to poll NIC buffer method is also presented to avoid batch processing; however, it still has some problems as explained in the next section.

The above literature either does not care about the timestamp of packets or depends on some specialized hardware support to timestamp packets. At the same time, little work is done to reduce the timestamp inaccuracy incurred by batch processing. It is inevitable to degrade the performance of packet capture if the batching processing is forbidden for DPDK. Without specialized hardware support, to provide a relatively accurate timestamp of packets, a new timestamping mechanism is a must for DPDK burst-oriented packet processing, which motivates us to design a mechanism to reduce the influence on the timestamp accuracy imposed by batch processing.

3. **DPDK Introduction and Problem Statement.** DPDK is the Data Plane Development Kit that consists of libraries to accelerate packet processing workloads running on a wide variety of CPU architectures [17]. To handle the bottleneck of packet capture at 10 GbE links caused by traditional Linux Network Stack [18], DPDK adopts the following features.

- Kernel Bypassing. DPDK allows user applications to circumvent the Linux kernel network stack and access the DMA (Direct Memory Access) buffer directly.
- Zero-Copy. By accessing the DMA buffer directly, the data does not need to copy from kernel space memory to user space memory.
- Huge Page. DPDK utilizes Huge Pages to pre-allocate packet memory buffer when the application is initialized to save the cost of further allocation or de-allocation of memory.
- Polling-based Packet receiving. Polling-based instead of interrupt-based packets receiving avoids extra overhead imposed by the interrupt.
- Batch Processing. Burst-oriented functions are used to amortize the cost of packet processing.

In our paper, our experiments are based on the DPDK load balance example [22] and some modifications are done to meet our requirements. The packet processing procedure in the modified load-balance example is shown in Figure 1. A special core (thread) is responsible for retrieving packets from the NIC in a burst, called management core. When a burst of packets is received, the management core timestamps the packet, and then distributes it to a processing core called worker, using some load balance algorithms such as HRW [23,24], and RSS [25,26]. On each worker, one can do some processes on each packet, such as packet classification [29] and packet parsing. In this paper, our focus is the packet timestamping mechanism on the management core instead of packet processing on the process cores (worker). Therefore, in our experiments, no more attention will be paid to the processing cores.

For DPDK, to achieve optimal performance, burst-oriented functions are employed in receiving or transmitting packets. The burst size can be configured by programmers on demand. Due to the polling nature, a particular core (or thread) is continuously polling

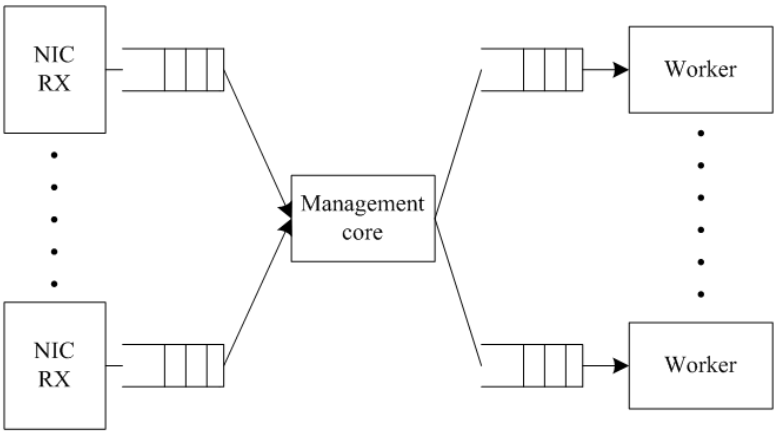

FIGURE 1. Packet processing procedure in modified load balance example

TABLE 1. Theoretical polls per second for different packet size with varying burst sizes

Packet size (B)	Burst size	Poll frequency
64	1	14880952
1518	1	812744
64	16	930059
1518	16	50796
64	32	465029
1518	32	25398
64	64	232514
1518	64	12699

the NIC ports for packets regardless of packet availability resulting in a 100 percent utilization of the poll core [18]. To deal with 10 Gbps incoming traffic, the theoretical maximum polls per second for different size packets of varying burst sizes are presented in Table 1.

DPDK sets burst size 64 as the default configuration in our used version, which can be changed by users on demand. However, the number of packets in each poll is always the minimum between burst size and the number of available packets [14]. From Table 1, we can get at 10 GbE line rate network, the theoretical polls per second are in the range of $[12699, 232514]$ as the burst size is 64.

To approach the arrival time of packets, timestamping packets should be as close as possible to the packets observed. It is not an easy task due to the high traffic rate in a software way.

Moreno et al. propose a kernel poll mode to fetch packets from NIC's receive queue regardless the link load just as DPDK does [14]. If one or more packets are available, they will be copied in a 1-by-1 basic to the corresponding circular buffer. Each packet will be timestamped before the packet copy is made utilizing the Linux kernel getnatimeofday() function. There is doubt about timestamping packets since each packet will lead to a getnatimeofday() call. Similar to Moreno, a software-based packet capturing with libpcap [27] proposed by Orosz and Skopko also timestamps each packet by acquiring system time when it is enqueued to the kernel's input packet queue. Is it suitable for DPDK burst-oriented packets receiving to timestamp packets? Now we will give our analysis in theory.

When the burst size is set 64, for a packet whose length is 64 bytes, the theoretical poll frequency is 232514, while the value is 12699 for a packet whose length is 1518 bytes. Accordingly, the time upper bound of each poll is 4.3 microseconds (μs) and 78.7 μs, respectively. That is, the cumulative time gaps of packets in the same burst should be no more than 4.3 μs when the packet length is all 64 bytes, and for packets whose length is 1518 bytes, the cumulative time gaps should be no more than 78.7 μs. The cumulative timestamp gap upper bound is varying from different burst sizes, but it is very easy to get by simple calculations.

Let the cumulative timestamp gaps of packets in the same burst be Δt_{cumul} and its upper bound be U. Supposed that n is the number of packets in the burst and t_i is the timestamp of the ith packet in the burst. We have Formula (1)

$$\Delta t_{cumul} = \sum_{i=2}^{n} (t_i - t_{i-1}) \leq U. \tag{1}$$

Assumed that the total time spending on processing a packet is t_{total}. For instance, in our load balance example t_{total} includes the time to acquire system time to timestamp packets and the time spent by load balancing. Although we receive packets in a burst, we still need to distribute packets in a one by one fashion for our balance goal. Therefore, the timestamp of the ith packet in the burst can be expressed as Formula (2)

$$t_{i+1} = t_i + t_{total}, \quad 1 \leq i \leq (n-1), \tag{2}$$

where n is the number of packets in the burst.

Substituting Formula (2) into Formula (1), we have the following result

$$(n-1) * t_{total} \leq U. \tag{3}$$

Formula (3) shows that not only the packet number in the burst but also t_{total} which is dependent on the subsequent operations on each packet has an effect on the cumulative time gaps of the packets in the same burst. It is obvious that no one can guarantee that Formula (3) is always true. Therefore, it is not a wise choice to get system time for each packet to timestamp.

The above deduction is straightforward. However, it inspires us to design a new timestamping mechanism for DPDK burst-oriented packet receiving. In the new method, the timestamp of the packets in the burst should be approaching the time the packets arrive on the condition of satisfying Formula (3).

4. **Packet Length Aware Timestamping Mechanism.**

To resolve the above problem, in this paper, a Packet Length Aware timestamping mechanism (PLA) for DPDK burst-oriented packet processing is presented. Before introducing our PLA, we briefly describe a very simple but practical method in our project as the timestamping mechanism in the Alpha version, which is called Plus a Fixed Value (PAFV). In PAFV, we only need to get system time once for the first packet in the burst. For other packets in the same burst, PAFV only needs to use the timestamp of the previous packet plus a fixed value as the timestamp of the new packet. As for the fixed value, we can use the time interval between two consecutive frames at the 10 GbE links when the link is full. However, though PAFV is simple enough, the timestamp of the packet is less accurate, which motivates us to find a better mechanism to timestamp packets so as to get more accurate timestamp of packets. Inspired by PAFV, PLA comes to us. In PLA, for each burst, we also need to get system time only once; however, the timestamp of packets in the same burst is not depending on its previous packet timestamp plus a fixed value. On the contrary, we

change the fixed value to a variable one, which is relative with each packet in the burst. Our timestamping mechanism can be modeled as follows:

$$T_{j+1} = \begin{cases} t_{cur}, & j = 0 \\ T_j + \sigma, & j \in [1, (n-1)] \end{cases}, \tag{4}$$

where n is the number of packets in the burst and t_{cur} is the current time of the system. σ is a function of packet length x and its corresponding time interval y, which is expressed as follows:

$$\sigma : x \to y, \quad x \in [64, 1518]. \tag{5}$$

Here, we use the standard Internet frame length whose maximum is 1518 bytes and the minimum is 64 bytes, defined in RFC 894 [28].

For σ, given arbitrary packet length in the range of $[64, 1518]$, the corresponding time interval between two consecutive frames can be obtained by simple calculations at 10 GbE links. Table 2 lists popular packet length and its time interval, defined as Δt.

TABLE 2. Popular packet length and its time interval (B = bytes, ns = nanosecond)

	64 (B)	128 (B)	256 (B)	512 (B)	1024 (B)	1518 (B)
Δt (ns)	60.8	112	214.4	419.2	828.8	1224

When packets in a burst are retrieved, if we compute the time interval for each packet, the cost may be non-negligible. Fortunately, the function σ can be known in advance. Given the fact that the fast lookup speed of the array, when the program is initialized, we put it in a global array of the type double to alleviate the negative impact on the system performance caused by the proposed method. Finally, Formula (5) can be rearranged as follows:

$$T_{j+1} = \begin{cases} t_{cur}, & j = 0 \\ T_j + time[i], & j \in [1, (n-1)],\ i \in [1, 1518] \end{cases}, \tag{6}$$

where $time[i]$ is the global array of the packet length and its corresponding time interval.

Please note that since the minimum Internet frame length is 64 bytes, we set $time[i] = time[64]$ for $i \in [1, 63]$.

Now, we should demonstrate that the proposed timestamping mechanism can always enable (1) to be true. From Formula (6), when packets in a burst are retrieved, we only need to get the system time once, and subsequence packets in the same burst are timestamped based on its previous packet. In other words, the timestamp of the packet in the same burst has nothing to do with any operations performed on each packet later. According to our new mechanism, Formula (1) can be modeled as follows:

$$\Delta t_{cumul} = \sum_{j=1}^{n-1} time[p_j], \tag{7}$$

where p_j is the packet length of the jth packet in the burst and n is the number of packets in the burst.

Firstly, let us consider the case where all packets have the same length in the same burst. The maximum packet length we use in this paper is 1518 bytes while the minimum is 64 bytes which is the standard Ethernet frame length, defined in RFC 894 [28]. When the packet length is 1518 bytes, the time interval is 1.2 μs. Substituting $n = 64$ and $time[p_j] = 1.2$ μs into Formula (7), we have $\Delta t_{cumul} = 75.6$ μs. At the same time, the theoretical upper bound is 78.7 μs, which is larger than the value in our method. When the packet length is 64 bytes, the time interval we have is 60.8 ns. Then substituting $n = 64$ and $time[p_j] = 60.8$ ns into Formula (7), we have $\Delta t_{cumul} = 3.8$ μs, which is lower

than the theoretical upper bound whose value is 4.3 μs. As for other packet lengths in the range of [64, 1518], the same result can get.

Then, we should consider the case where packets in the burst have different packet lengths. At 10 Gb/s links, the maximum packet rate is computed as $\frac{10\ Gb/s}{(Pakcet\ length\ +\ Preamble\ (8\ bytes)\ +\ Inter\ packet\ gap\ (12\ bytes))\times 8}$. The theoretical poll frequency equals maximum packet rate as the burst size is 1. As for different burst sizes, the theoretical burst poll frequency is $\frac{Maximum\ Packet\ Rate}{burst\ size}$. Then, the theoretical timestamp upper bound of each poll is $\frac{1}{poll\ frequency}$. For our new timestamping mechanism, the time interval between two consecutive packets we use is similar to the one used in [14], computed as $\frac{(packet\ length\ +\ inter\ packet\ gap\ (12\ bytes))\times 8}{10\ Gb/s}$. Supposed the burst size is n, and pkt_len_i is the packet length of the ith packet in the burst, where $1 \le i \le n$. The cumulative time gaps of our PLA is $\sum_{i=1}^{n-1} \frac{(pkt\_len_i+12)\times 8}{10\times 10^9}$, written as t_{new}. From Table 1 and the above analysis, even though the burst size is fixed, the theoretical timestamp upper bound varies with different packet lengths. When the burst size is n and the packet lengths are different, it can be thought of consisting of N bursts with a size of 1, where $N = n$. Therefore, the theoretical timestamp upper bound of the burst with different packet lengths is $\sum_{i=1}^{N} \frac{(pkt\_len_i+20)\times 8}{10\times 10^9}$, written as t_{theor}. Obviously, we have $t_{new} < t_{theor}$.

Up to now, we have shown that the proposed method always conforms to the model we give, which demonstrates the rationality of the new timestamp mechanism. Instead of acquiring system time for each packet in a burst, our packet length aware timestamping mechanism only needs to acquire system time once for a burst, and it is not affected by the number of packets in the burst. What is more, any operations on packets in the same burst has nothing to do with packets' timestamp. By doing so, we decrease the impact of burst-oriented packet processing on the timestamp accuracy as much as possible. The new proposed and effective timestamping mechanism, on the one hand, provides a simple but rational method to timestamp packets without specialized hardware support, which can be used by these fast packet I/O frameworks to provide a relatively accurate packets' timestamp. On the other hand, considering the impact of batch processing, it can be used as a reference for how to provide a relatively accurate timestamp of packets in the case of satisfying the theoretical model.

5. **Experimental Results.** The test environment is set up including a packet generator, a switch, and a server. The test server with two quad-core Intel Xeon E5-2609 processors running at 2.40 GHz with 16 G of DDR3 RAM is equipped with a 10 GbE Intel NIC based on 82599 chip. We use IXIA as our packet generator and traffic is mirrored to our test server through a specific port on the switch by a 10 Gb/s fiber-based link. The DPDK version we use is 17.05.2.

As we know, the polling-based method is not sensitive to link load, and it is not a must to saturate the 10 Gb/s link. We use Ixia to generate two traffic traces, and the detailed information about them is shown in Table 3 and Table 4. Our test uses the default burst size, which is 64. Since the number of packets in a burst is the minimum value between the burst size and the available packets, it may have a varying number of packets in a burst under the same configuration. We do some comparisons among PLA, KPT, and PAFV so as to verify the rationality and effectiveness of our proposed method.

First of all, a test is performed to verify the timestamping mechanism used by Moreno et al. and Orosz and Skopko, which timestamps packets in a 1-by-1 fashion after acquiring system time, named KPT for simplicity.

Figure 2 shows the experimental results. The horizontal axis is the number of packets in the same burst while the vertical axis represents the cumulative time gaps. The red

TABLE 3. Percentage of different packet lengths in trace 1

Packet length (bytes)	Percent (%)
64	33.1
66	0.5
412	0.3
1356	0.3
1518	65.8

TABLE 4. Percentage of different packet lengths in trace 2

Packet length (bytes)	Percent (%)
64-128	51.1
129-512	0.4
513-1024	0.4
1025-1518	48.1

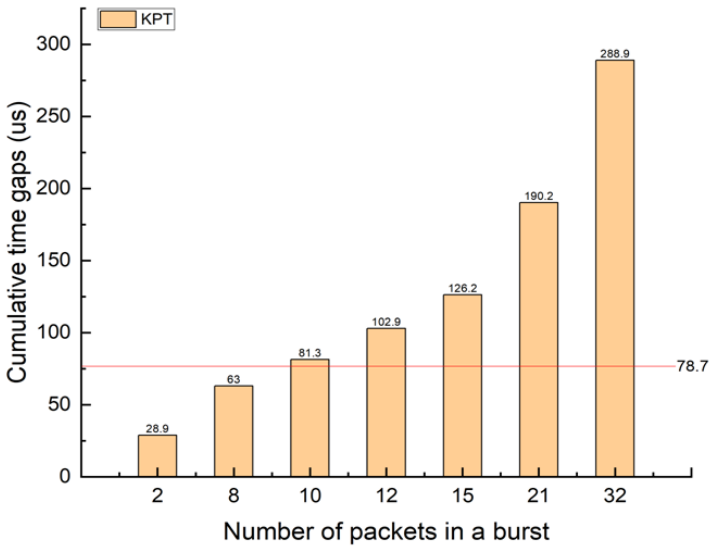

FIGURE 2. The cumulative time gaps of the packets in the same burst for KPT

line is the upper bound of the cumulative time gaps under the test environment. It is seen that when the available packets in a burst reach 10, the cumulative time difference exceeds 78.7 μs, which is the maximum theoretical upper bound while the burst size is 64. The experimental results are consistent with our theoretical analysis in Section 3. To timestamp packet, we should acquire the system for each packet at the same time any operations on the packet will have a negative effort on the next packet in the same burst. Finally, the cumulative time difference is uncontrollable which makes this method not rational in theory and timestamp inaccuracy. The more available packets in a burst or the more complex the subsequent operations of the packet, the less accurate the timestamp. Hence, it is not a wise choice to get system time for each packet to timestamp.

It is also concluded that when the burst size is set bigger than 10 (often 16), the timestamp inaccuracy occurs more and more frequently due to burst-oriented packet receiving. A similar result is also found by Moreno et al.

The packet length aware timestamping mechanism proposed by us is examined in another experiment. The results are depicted in Figure 3. The horizontal axis is the number

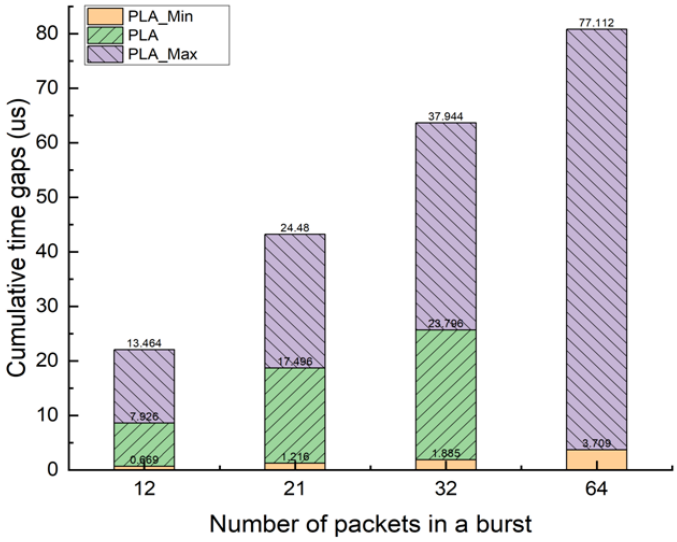

FIGURE 3. The cumulative time gaps of the packets in the same burst for PLA

of packets in the same burst while the vertical axis represents the cumulative time gaps. PLA maximum in Figure 3, recorded as PLA_Max, is the cumulative time gaps in the worst case in which all packets in the burst have maximum length while PLA minimum, recorded as PLA_Min, is minimum of the cumulative time gaps in case all packets' length is 64 bytes. The experimental results are indicated as PLA. When the available packets in a burst are 12, 21, 32, which have a higher frequency of occurrence in our test, the cumulative time gaps are all smaller the maximum and also lower than the theoretical upper bound. That is to say, no matter what the number of packets in the burst (no more than the burst size), the cumulative timestamp gaps of our proposed timestamping mechanism are always no more than the theoretical upper bound, which proves our new method is independent of the number of packets in the burst.

One finds that there is not an experiment value when the packet number equals the burst size. The reason is that we seldom see it even though the test is repeated many times. We still list the maximum since it is the worst case in our proposed timestamping mechanism. As the results show, even in the worst case, the maximum of the cumulative time gaps is about 77 μs, lower than the maximum theoretical upper bound, which provides a strong proof of the effectiveness of our timestamping mechanism. By the way, enlarging the traffic, packet number reaching 64 in a burst happens frequently.

Figure 4 describes the timestamp distribution of packets in the same burst when the burst size is 64. The horizontal axis is the packet number while the vertical axis represents the timestamp of packets. We set the timestamp of the first packet in the burst to 0 for convenience. Other packets timestamp just use their previous packet timestamp plus a corresponding time difference between the current one and its previous packet. Therefore, the timestamp of the packet is equal to the cumulative timestamp gaps. Compared with KPT, whose timestamp of the tenth packet exceeds the maximum theoretical upper bound, the timestamp of the packet for PLA is always confirming to the model we give. As the number of packets increases, the timestamp of packets is always no more than 78.7 μs, which is the maximum theoretical upper bound when the burst size is 64. Once again, the result shows that our timestamping mechanism has nothing to do with the number of packets in the burst. Compared with PAFV, in the condition of satisfying our given model, the timestamp distribution of packets in the same burst is more dispersed, which is more approaching to the arrival time of the packet.

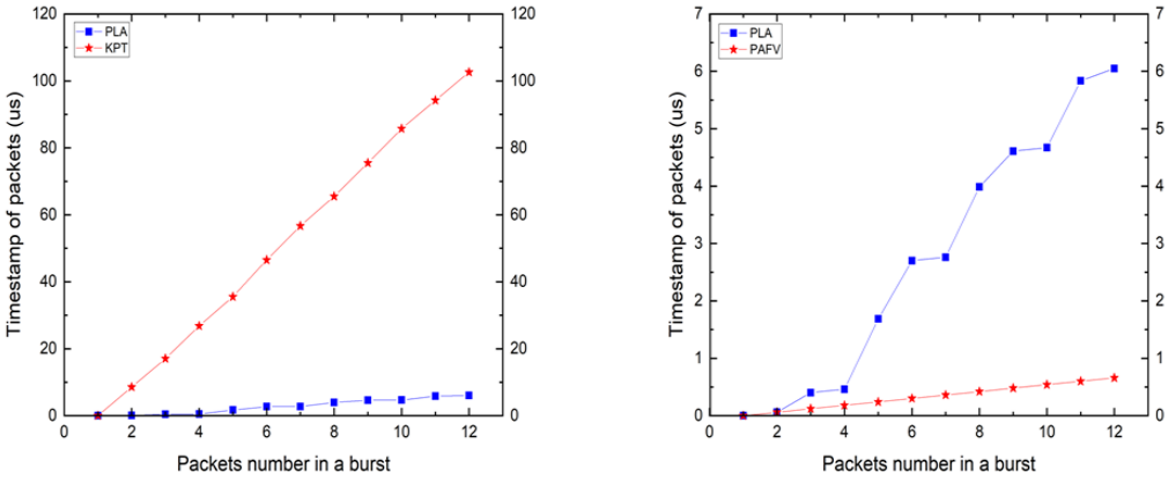

FIGURE 4. Timestamp distribution of packets in the same burst for PLA, KPT, PAFV

As Figure 1 shows, after receiving packets in a burst, we timestamp packets on the management core, then distribute them to the worker for further processing in one by one fashion. For KPT, we should acquire system time for each packet in the burst. When the first packet is timestamped, if we do packet header parsing before distributing it to the worker, the timestamp of the next packet would be larger than do nothing resulting in the cumulative gaps exceed the theoretical upper bound with fewer packets. If the operation is complex enough, the timestamp gaps of the first two packet will be larger than the theoretical upper bound. Since our timestamping mechanism only needs to acquire system time once for a burst, no matter how complicated the operation is, it has no effect on the timestamp of packets in the same burst. The cumulative timestamp gaps are always no more than the theoretical upper bound, just as Figure 3 shows, which indicates our method has nothing to do with the operations of packets.

The new timestamping mechanism for DPDK burst-oriented packet processing acquires system time to timestamp the first packet in a burst and the other packets in the same burst are timestamped based on their previous packet length. Figure 5 describes the relationship between packet length and timestamp gaps of the packets, which is a graphical representation of the function σ.

The last experiment is about a comparison between the cost of acquiring system time and looking-up in a given array for our proposed PLA method. We repeat our experiment ten times and compute the average time spending on looking-up operation and getting system time. Table 5 illustrates the result.

The lookup operation in the array requires 38 cycles while acquiring the system time costs 90 cycles on average. It is the result that provides a strong proof of the effectiveness of our proposed timestamping mechanism. As for PAFV, since it uses the timestamp of the previous packet plus a fixed value as the timestamp of the new packet, the calculating cost of time can be avoided.

6. **Conclusions and Discussions.** The DPDK burst-oriented packet receiving boosts up the performance of packet IO engines; however, it is also blind to the timestamping mechanism of the packets. Traditional methods timestamp packets after acquiring the system time in 1-by-1 fashion, like KPT, which results in timestamp inaccuracy due to its dependence on the packet number in the same burst and subsequent operations on packets. To mitigate the above problem, a packet length aware timestamping mechanism

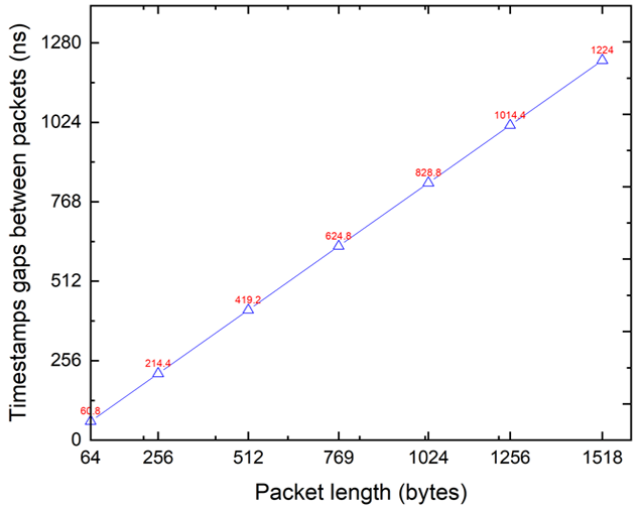

FIGURE 5. The relationship between packet length and timestamp gaps of the packets

TABLE 5. Average time spending on lookup and getting system time

Method	Time spending on average (cycles)
PLA	38
KPT	9

is proposed based on the PAFV method in our practical implements. When a burst of packets is received, we only need to acquire the system time once for the first packet, and the others are timestamped based on its previous packet length. Firstly, we build a model for the cumulative time gaps of packets in the same burst, which gives the upper bound of the cumulative time gaps. Then we demonstrate that timestamping packet in a 1-by-1 fashion after acquiring system time is not suitable for DPDK and our proposed method is always conforming to the theoretical model we give. Finally, experimental results show the effectiveness and rationality of our timestamping mechanism. At the same time, since the time interval between two consecutive frames can be obtained in advance at 10 GbE network, it does not incur any additional overhead. On the contrary, place the relationship between packet length and corresponding time gap in a global array initialized when the program is constructed and getting the timestamp by simple lookup operation cost 2/3 less time than acquiring the system time which also demonstrates our method is effective.

In the proposed timestamping mechanism, since the time interval we use is the value between two consecutive frames at 10 GbE links when the links are full, actually, it is far from the real value because almost the load of links is less than 10 Gbps. To approach the arrival time of packets, timestamping packets should be as close as possible to the packets observed. In our future work, our team would like to design an FPGA-based packet capture card, which can provide nanosecond timestamp of packets. To integrate with DPDK, it is a long way to go.

Acknowledgment. This work was supported by Strategic Leadership Project of Chinese Academy of Sciences: SEANET Technology Standardization Research System Development (Project No. XDC02070100). We also gratefully acknowledge the editor and all reviewers for their valuable suggestions.

REFERENCES

[1] L. Cheng, J. Wu, H. Gao and T. Han, Optimal performance of networked control systems with packet dropouts and bandwidth constraints, *ICIC Express Letters*, vol.11, no.4, pp.863-870, 2017.

[2] Cisco, *Cisco Carrier Routing System*, http://cisco.com/en/US/products/ps5763/index.html, Accessed in 2019.

[3] L. Rizzo, L. Deri and A. Cardigliano, *10 Gbit/s Line Rate Packet Processing Using Commodity Hardware: Survey and New Proposals*, 2012.

[4] N. Bonelli, A. Di Pietro, S. Giordano and G. Procissi, On multi-gigabit packet capturing with multi-core commodity hardware, *International Conference on Passive and Active Network Measurement*, pp.64-73, 2012.

[5] S. Bradner, *Benchmarking Terminology for Network Interconnection Devices*, RFC Editor, 1991.

[6] J. Micheel, S. Donnelly and I. Graham, Precision timestamping of network packets, *Proc. of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pp.273-277, 2001.

[7] T. Mizrahi, *Zen and the Art of Network Timestamping*, 2018.

[8] P. Emmerich, S. Gallenmüller, G. Antichi, A. W. Moore and G. Carle, Mind the gap – A comparison of software packet generators, *ACM/IEEE ANCS*, pp.191-203, 2017.

[9] *The DAG Project*, http://www.endace.com, Accessed in 2019.

[10] A. Pásztor and D. Veitch, PC based precision timing without GPS, *ACM SIGMETRICS Performance Evaluation Review*, vol.30, no.1, pp.1-10, 2002.

[11] *Cace TurboCap Network Interface Card*, http://www.cacetech.com/products/turbocap.html.

[12] N. Zilberman, Y. Audzevich, G. A. Covington and A. W. Moore, NetFPGA SUME: Toward 100 Gbps as research commodity, *IEEE Micro*, vol.34, no.5, pp.32-41, 2014.

[13] *https://www.napatech.com*, Accessed in 2019.

[14] V. Moreno, P. M. S. del Rio, J. Ramos, J. J. Garnica and J. L. Garcia-Dorado, Batch to the future: Analyzing timestamp accuracy of high-performance packet I/O engines, *IEEE Communications Letters*, vol.16, no.11, pp.1888-1891, 2012.

[15] M. Trevisan, A. Finamore, M. Mellia, M. Munafo and D. Rossi, Traffic analysis with off-the-shelf hardware: Challenges and lessons learned, *IEEE Communications Magazine*, vol.55, no.3, pp.163-169, 2017.

[16] P. Orosz and T. Skopko, Software-based packet capturing with high precision timestamping for linux, *The 5th International Conference on Systems and Networks Communications*, pp.381-386, 2010.

[17] *https://www.dpdk.org*, Accessed in 2019.

[18] H. G. Trifonov, Traffic-aware adaptive polling mechanism for high performance packet processing, *University of Limerick Institutional Repository*, 2017.

[19] M. Trevisan, A. Finamore, M. Mellia, M. Munafò and D. Rossi, *DPD-KStat: 40 Gbps Statistical Traffic Analysis with Off-the-Shelf Hardware*, Technical Report, 2016.

[20] L. Rizzo, Netmap: A novel framework for fast packet I/O, *The 21st USENIX Security Symposium (USENIX Security 12)*, pp.101-112, 2012.

[21] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart and G. Carle, MoonGen: A scriptable high-speed packet generator, *Proc. of the 2015 Internet Measurement Conference*, pp.275-287, 2015.

[22] *http://doc.dpdk.org/guides/sample_app_ug/load_balancer.html*, Accessed in 2019.

[23] K. W. Ross, Hash routing for collections of shared Web caches, *IEEE Network*, vol.11, no.6, pp.37-44, 1997.

[24] D. G. Thaler and C. V. Ravishankar, Using name-based mappings to increase hit rates, *IEEE/ACM Trans. Networking*, vol.6, no.1, pp.1-14, 1998.

[25] O. Shemesh, *System and Method for Symmetric Receive-Side Scaling (RSS)*, U.S. Patent No. 8,635,352, U.S. Patent and Trademark Office, Washington, D.C., 2014.

[26] K. Li, L. Ye and X. Z. Yu, Traffic dynamic load balancing method based on DPDK, *Intelligent Computer and Applications*, vol.7, no.4, pp.85-89, 2017.

[27] Libpcap, *A Common Open Source Packet Capture Library for Unix/Linux Systems*, http://www.tcpdump.org/, Accessed in 2019.

[28] C. Hornig, *A Standard for the Transmission of IP Datagrams over Ethernet Networks*, RFC 894, 1984.

[29] L. N. Jing, Z. X. Ye and X. Chen, Packet classification algorithms based on decision tree, *Journal of Network New Media*, vol.7, no.2, pp.1-11, 2018.