Available online at www.sciencedirect.com

ScienceDirect

Optical Switching and Networking 00 (2018) 1–9

High-speed optical networks latency measurements in the microsecond timescale with software-based traffic injection

Rafael Leira[a,*], Javier Aracil[a,b], Jorge E. López de Vergara[a,b], Paula Roquero[a,b], Iván González[a,b]

[a] *Universidad Autónoma de Madrid, Spain.*
[b] *Naudit High Performance Computing and Networking, S.L., Spain*

Abstract

In many optical network scenarios, such as Storage Array Network (SAN) replication, keeping latency under control is cornerstone to provide a proper Quality of Service (QoS). Hence, measuring latencies in such optical networks becomes fundamental. However, for low distances, microseconds resolution is required, which, in turn, demands ad-hoc hardware implementation for the measurement device. Alternatively, a more cost-effective solution is that of software-based methods, but up to date they were not precise enough at 10 Gbit/s or above. In this paper, we analyze current high-performance packet engines, such as DPDK, and pinpoint the issues involved when it comes to measure latencies in high-speed optical networks. Based on these findings, we propose the use of a software-based solution to measure latency. Furthermore, we also propose an extension that serves to measure bandwidth as well, with the novel concept of convoy of packet trains.

Keywords: Latency measurements, network measurements, high-speed optical network measurements, software-based

1. Introduction

Precise measurements of end-to-end latencies in optical networks are of fundamental importance for performance evaluation of data-center systems such as Storage Array Network (SANs) with synchronous replication. Figure 1 shows a SAN replication scenario, which is common in many data-centers. In order to perform a *write* operation in the SAN, the data must be saved in *both* primary and backup SANs, which must provide explicit confirmation that the data has been saved. As a result, the *write* latency is constrained by the round-trip time between the primary and backup SAN. To speed up *write* operations, a dark fiber or wavelength is setup between primary and backup SAN, just to carry the input/output traffic to/from the SAN as fast as possible and with no interference with the rest of the traffic. This is a common scenario in critical transactional systems such as credit card authorization systems, which demand a very tight response time and synchronous copies of all transactions.

Being RTT the round-trip time between the primary and secondary SAN and N the number of write operations that can be performed in parallel in the primary database, we note that the throughput ρ, in write operations per second, is upper bounded as follows

$$\rho < \frac{N}{RTT} \quad (1)$$

Usually, synchronously replicated SANs are not located very far apart, to avoid large RTTs. Actually, primary and backup SANs may be placed at 10 kilometers distance from each other, in the same city, namely an approximate RTT of 100 microseconds. If $N = 10$ write operations can be performed in parallel, it turns out that increasing the RTT in just 10 microseconds makes the overall throughput decrease in roughly 10,000 operations per second, which is very significant. Such decrease is due to the fact that Eq. 1 is hyperbolic and small increases in the denominator have a large impact on the result.

The previous motivation shows that an accurate estimation of the RTT becomes of fundamental importance to assess the SAN performance. Chances are that the network operator switches the optical fiber to a longer path, and the resulting increase of latency in the microseconds timescale has a large impact on the performance. Hence, continuous monitoring of the latency at the microsecond resolution is in order.

In order to measure latency with such a fine-grain granularity, reflectometers can be used in an optical segment. However, such reflectometry does not take into account the interme-

☆Manuscript received 19 December 2017. Revised 19 March 2018. Accepted 27 March 2018

☆☆The final publication is available at Elsevier via `https://doi.org/10.1016/j.osn.2018.03.004`

*Corresponding Author

Email address: `rafael.leira@uam.es` (Rafael Leira)

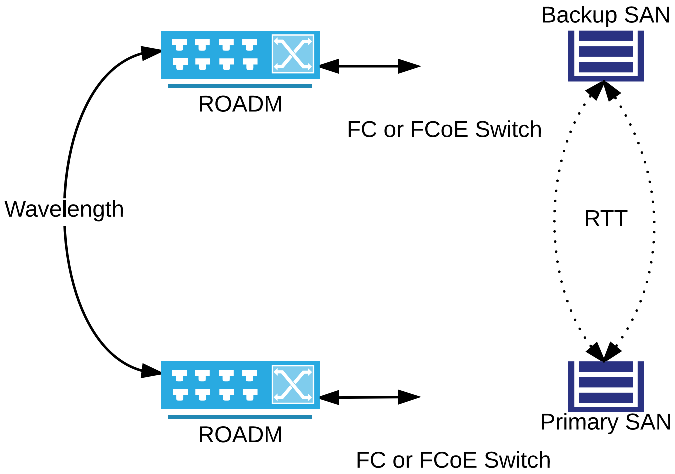


Figure 1. SAN replication scenario

diate active equipment such as SAN switches (Fiber Channel -FC-, Fiber Channel over Ethernet -FCoE- or Internet SCSI -iSCSI-) which make an integral part of the path between endpoints. To account for such intermediate hops, a packet or packet train [1, 2] can be sent from one endpoint, which, upon reflection in the other endpoint, comes back to the transmitting endpoint for RTT calculation. Such packet probing technique can be implemented either in hardware or in software.

We note that a hardware-based implementation, at multi-gigabit-per-second speeds, is very expensive. Actually, software-based solutions, running on general-purpose hardware, are more cost-effective. In this article, we explore the limitations of such software-based solutions, both in terms of precision and fine-tuning of measurement parameters. Our finding shows that, if the measurement parameters are carefully selected, the resulting measurement is precise enough to measure optical network latencies in the microseconds timescale.

Furthermore, going back to our SAN case study, we note that the SAN controller workstation, which is usually a Unix/Linux solution, is an ideal choice for running the software-based measurement solution. Hence, continuous monitoring of the optical network can be performed. This has the benefit of promptly alerting the SAN manager in case an increase of latencies is observed. More specifically, we show that measuring optical network latencies with very high precision (hundreds or even tens of nanoseconds) is feasible at software level if the following issues are taken into account: i) the effect of packet queuing and the difficulties in packet timestamping due to the non-deterministic nature of software; ii) the lack of granularity in the system clock, which complicates matters for obtaining a precise time measurement.

In this article, we report on the lessons learned in software-based latency estimation at multi-gigabit-per-second speeds, which are useful for practitioners in the field. We also release a code that can be used to carry out such high-precision measurements in the optical network. In the next section, we explore how hardware and software have been used to date in order to measure networks and networking devices. Then, we survey the Intel Data Plane Development Kit (DPDK)[1], as the current *de facto* standard in packet processing engines. We explain how packet-handling process works in DPDK and its strengths and drawbacks for the design of network measurement tools. The article follows with the experimental results in a real dark fiber setup. Finally, we present a simple methodology to measure high-speed networks using DPDK, followed by the conclusions that can be drawn from our research.

2. Related work

The state of the art features several software measurement tools, such as iPerf[2], Aria2[3], or JMeter[4], each of them devoted to measure different network or application layer parameters. The performance of these applications depends on the Operating System (OS) and kernel network stack where they run. We note that commonly used OS, such as Linux or Windows, are not deterministic. Consequently, the response time of any system call is not bounded, which applies to system calls to ask for the system time as well. On the contrary, Real-Time Operating Systems (like Linux RT[5]) offer better response time guarantees, albeit they are not totally deterministic as well.

As it turns out, the fact that response time is not guaranteed by the OS has an impact on software-based traffic injection measurements, and especially in latency measurements. That is the reason why there is a strong criticism against software-based traffic injection, and in favor of hardware devices, such as FPGAs [3, 4, 5, 6]

The answer from the software community, in order to address this call for better precision and accuracy, was to create *pktgen* [7]. The *pktgen* software runs in kernel space and is allowed to have direct access to network interface cards (NIC) queues. As a result, most overheads due to kernel context switching and packet queuing are removed. Consequently, *pktgen* performs better than the corresponding userspace counterpart does, which employs a standard socket programming interface. However, the use of non-deterministic kernel functions (a simple `printk` already has an unbounded response time), as well as the kernel network stack, hampered *pktgen* suitability above 1 Gbit/s. After much work, *pktgen* has significantly improved performance, even though it is well below 10 Gbit/s [8].

In order to scale to higher speeds, a radical departure from the traditional socket-based paradigm (even at kernel level) was in order and *packet engines* appeared in the scene. For example, Luca Deri provided the first packet engine (PF_RING [9]) whereas Sangjin Hal *et al.* provided the first one that uses coprocessors (PacketShader) [10]. In their work, they mapped some network functions to userspace, skipping the kernel network stack, which drastically reduced the number of system calls needed to read or write network packets. Such a novel design approach allowed them to fully exploit the capabilities

[1] http://dpdk.org/
[2] https://github.com/esnet/iperf
[3] https://aria2.github.io/
[4] http://jmeter.apache.org/
[5] https://rt.wiki.kernel.org

of commercial high-speed (10 Gbit/s) network cards. Sometime later, different researchers adopted this idea to create their own userspace drivers. Among others, Netmap, HPCAP, or the popular DPDK engine [11]. The combination of this innovative approach with the latest developments in virtualization technology has made possible to implement different network devices [12, 13] with general-purpose hardware.

As it turns out, DPDK technology has overwhelmed the rest of packet engines currently available. Such an engine, which was primarily developed to build software-based switches and routers, has indeed set a milestone in the state of the art. The cornerstone of DPDK is the ability to work at userspace, skipping almost all kernel calls, and the availability of an abstraction layer that allows working with several network cards and even switches such as FM10k[6]. Such DPDK developments have been used to implement low latency tools [12, 14, 15], which represent fundamental contributions for the state of the art in software-based traffic injection.

In this regard, Moongen [16], as a packet injector alternative, promises packet generation based on LUA scripts with latency measurements at nanoseconds level. However, this is limited to NICs that support hardware-based timestamping, which is typically provided for specific packet types, such as PTP packets in Intel cards. However, security policies are becoming more and more stringent and chances are that PTP packets are filtered out in their way from source to destination being measured.

Furthermore, we note that the measurement traffic must adapt to the network setup and not the other way around. For example, in case Fiber Channel is encapsulated over Ethernet (FCoE) neither UDP nor PTP can be used to send the measurement packets, as FCoE is a level 3 protocol, which renders NIC-hardware timestamping unfeasible. Fortunately, a software-based solution is adaptable to any network. In the former example, we can actually use reserved fields on the FCoE packets to place the timestamp.

Our experimental results show that further research is necessary to understand the strengths and drawbacks of DPDK as a generic framework to measure latency in optical networks. As stated in the previous section, there are typical optical network scenarios that call for latency measurements in the microseconds timescale. Our findings show that DPDK is good enough to measure such latencies at 10 Gbit/s if and only if packet sizes and inter-packet times are carefully tuned. Such sensitivity analysis, which is fundamental in order not to bias the measurements severely, was missing in the literature.

3. DPDK packet handling

In this section, we provide a brief overview of the DPDK packet handling subsystem, depicted in Figure 2. Within DPDK environment, packets are handled as *mbuf*s, which are the *packet-descriptor* used in kernel space. An *mbuf* contains information related to a particular packet, namely the memory address where the packet is stored, its length and the physical port where it was received. Typically, and in order to speed-up processing and avoid system calls, an *mbufpool* is instantiated at the beginning. Such pool is a memory region of a specific NUMA node where packet descriptors can be requested with low latency and at a high rate. Before a packet is ready to be enqueued, the *mbuf* metadata fields must be filled up, namely the packet length or the parameters that indicate that the packet belongs to more than one *mbuf*.

Most importantly, DPDK works with packet batches, namely a batch of *mbufs* (i.e. an *mbuf array*) serves as an input for the transmission queue, no matter the number of packets we wish to enqueue at a time, and even in case a single packet is enqueued. However, performance depends on the number of packets in the batch and, usually, 144 *mbufs* is the recommended number.

Furthermore, whenever a physical port in a NIC is initialized, it is also mandatory to instantiate at least two software queues, for packet transmission and reception respectively. A software queue is a ring buffer with a finite number of *mbuf* pointers. Such pointers are handled by the NIC, which reads or writes in the available *mbufs* in the transmission or reception queues.

In order to mitigate possible delays and jitter in PCIe transactions, the NIC has a small internal queue that bridges the host software queue and the NIC processor. As it turns out, the NIC chipset performs some pre-processing of the packets being placed in such internal NIC memory. For instance, Receive Side Scaling (RSS) is a technique that serves to distribute the incoming packets in the CPU cores, such that higher ingestion rates can be achieved. In this regard, an RSS hash can be obtained per packet, which is used to demultiplex the packet stream into the former queues. Other hardware-level features include PTP timestamping, for instance.

Once the card has processed the packet, it is ready to be sent to the physical interface. We note that depending on the transmission technology, the packet can be queued again until it is finally sent to the physical medium. For example, let us consider a 10 Gigabit Ethernet (GE) NIC with an optical SFP+ transceiver, which uses the 10 Gigabit Media Independent Interface (XGMII) protocol to interplay with the card chipset. Such protocol does not allow sending a frame arbitrarily at any time because a certain bit alignment is in order. This alignment implies that the inter-frame gap length is not necessarily constant and equal to 12 Bytes. In fact, the 10 GE standard [17] specifies that it can be reduced up to 5 Bytes as long as the 12 Bytes are observed as an average. Such XGMII effect causes small variances in the latency between packets, even in dedicated hardware devices, in the order of a few nanoseconds.

3.1. DPDK limitations for traffic injection

As noted before, either packet pairs or trains can be used as measurement traffic. We may wish to send a packet train to measure the behavior of optical or Fiber Channel switches in the path with different traffic burstiness. However, we note that DPDK has limitations when working with packet batches of arbitrary size, which must be taken into account. More specifically, there is a minimum number of packets in a batch, mainly because the NIC works with a minimum number of packets in the PCIe transactions. Fortunately, such minimum number of

[6] `http://dpdk.org/doc/nics`

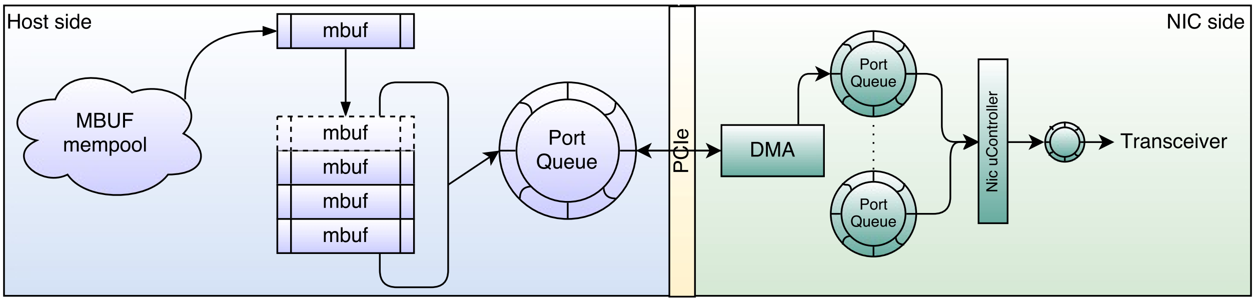


Figure 2. DPDK workflow

packets is small enough, for example, 4 packets in the 40 GE Intel XL710. Besides, a latency is introduced in the transmission queue. Indeed, when DPDK is requested to send a small batch of packets, it puts them in the transmission queue and informs the NIC. Then, some nanoseconds later, the NIC polls the memory where the packet descriptors are placed and it starts transmitting the frames. If another packet batch is enqueued before the previous batch is completely sent, the network card groups both batches without any additional interruption. The time interval between the NIC notification and NIC transmission is not constant, resulting in a severe bias in latency estimation, even at the microsecond timescales.

The second DPDK limitation is the fact that the queues introduce non-deterministic delays. When the network bandwidth increases, the inter-packet times get smaller and small random delays have an impact on the instantaneous frame rate, which, in turn, adds noise to the measurement. To overcome this limitation, several packet trains should be transmitted and the corresponding measurement samples should be averaged to reduce the noise.

The third DPDK limitation is related to clock precision and synchronization, which affects, in general, to any software-based traffic injection system. A basic functionality such as obtaining the current time can vary in precision, accuracy and computational cost across different CPUs or kernel versions. In order to obtain a precise timestamp, in the order of hundreds of nanoseconds, we require that the procedure takes the minimum CPU cycles as possible. The best way to go is to use the CPU cycle counter (Time Stamp Counter, TSC), which allows obtaining tens of nanoseconds resolution with a single instruction in assembler. However, this method is only useful if the CPU supports invariant TSC, namely if the cycle counter is common to all CPU cores, no matter their current frequency or their energy saving state (P-state).

3.2. DPDK experimental setup

In order to evaluate the above DPDK limitations and assess the suitability of DPDK for measuring optical network segments, we set up an experimental scenario consisting of a loopback fiber with a variable length, such that the propagation delay is known beforehand (see Figure 3). Thus, the delay measurements can be compared against a consistent ground truth.

Our testing environment consists of a single server with two Intel Xeon E5-2630 processors at 2.30 GHz and 16 GB RAM at 1333 MHz per NUMA node in quad-channel (32 GB in sum). The NIC is an Intel 10 Gigabit Ethernet SFP+ 82599ES. The OS is a Linux CentOS 7.3, with DPDK 17.05 as packet transmission and reception engine.

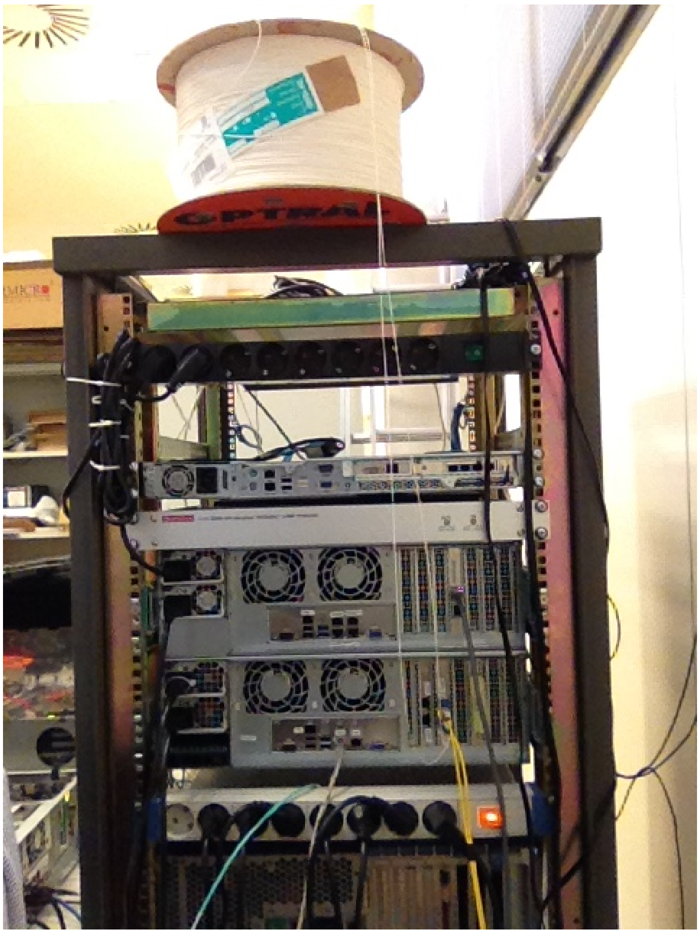


Figure 3. Fiber loop used in the testbed.

It is important to take into account that, unlike pure hardware solutions, software must timestamp the packet before the first bit has been sent and after all the bits have been received, which implies that the latency is dependent of the frame size used to test. Thus, given the refractive index of the fiber (n), the fiber length in m (l), the speed of light in vacuum in m/s (c_0), the link data rate in bit/s (r) and the data frame length in bits (b), the theoretic total delay (d) can be easily calculated as follows:

$$d = d_{prop} + d_{tx} = \frac{n \cdot l}{c_0} + \frac{b}{r} \quad (2)$$

where the first term refers to the propagation delay (d_{prop}), and the second term to the transmission delay (d_{tx}).

For example, based on Eq. 2, in a 2 m long fiber with a typical refractive index of 1.4444 transmitting at 10 Gbit/s, the resulting delay should be between 96 ns for the shortest frame (60 bytes at Ethernet layer excluding CRC, preamble and inter-frame gap) and 1262 ns for the longest frame (1518 bytes at Ethernet layer with VLAN). We further note that Jumbo-frames will lead to longer delays.

In the next section, we use the experimental setup described above in order to evaluate the strengths and drawbacks of DPDK for high-precision latency measurements in optical networks, which will provide guidance into the design principles of a software-based measurement system.

4. Design principles for high-resolution latency measurement

Due to the DPDK limitations stated above, current DPDK software for packet injection (such as DPDK-pktgen[7]) cannot be used directly without proper tuning.

This is the reason why we have contributed to the state of the art with our own tool DPDK-LatencyMetter, which is available to measure optical networks latency with high-resolution [8] and no special hardware requirements like timestamping capabilities at the NIC level [16]. In what follows, we provide insights into the design principles of DPDK-LatencyMetter.

4.1. CPU isolation

To minimize random effects, all measurement processes have to be associated to CPU cores that are isolated from any other process. This type of isolation can easily be achieved by using the `isolcpus` Linux kernel parameter. In any case, it is advisable that the Linux kernel is compiled with the `tickless` option. Otherwise, the process will suffer from periodic interruptions (a tick happens even if a process is not a candidate to be replaced by another one), producing context changes, which artificially introduce peaks in the delay measurements.

This effect can be observed by sending 1000 trains of consecutive packets and plotting the latency measurement as they were received. The Figure 4a shows this scenario with a train length of 256 packets, without any kind of isolation (kernel or other processes). The disturbing spike frequency is almost constant for a certain frame-size, but it changes with different frame sizes. This is due to the fact that a train with larger packets takes more time to be sent, or in other words, the time between measurements is longer. That is the reason why trains with larger packets suffer more frequent spikes over time than trains with smaller ones. The spikes vanish as soon as the `tickless` and `isolcpus` kernel options are activated (as shown in Figure 4b).

4.2. Power saving

Power saving policies directly affect the CPU clock performance and, consequently, the number of MIPS (*Million Instruction Per Second*) that can be achieved. Interestingly, we note that such number of MIPS does not have a noticeable impact on the latency measurements, but it does have in the bandwidth measurements. In fact, pure latency measurements do not require to produce packets at a certain rate, whereas bandwidth measurements are influenced by packet generation speed. In turn, the number of packets that can be generated and enqueued per second directly depends on the CPU frequency and its MIPS, as well as other factors, such as NIC-driver implementation or memory bandwidth.

When the CPU has enough MIPS to reach the maximum link capacity, both measurements become stable at some point, independently of the actual CPU frequency. Nevertheless, the time needed to reach such steady state directly depends on the MIPS and the number of packets per second required to achieve it, which also depends on the frame size (see the ramps/transient state shown at Figure 4).

4.3. Effect of frame size

Using the simplest packet train method, we have observed a correlation with the frame size and the delay metrics. In order to understand this effect, we have sent each train a thousand times for each frame size. For visibility reasons, we have chosen a frame size interval between 60 and 6000 Bytes at Ethernet level (excluding CRC, preamble, inter-frame gap, etc.) and we represent the results in terms of Interval Dispersion Index (IDI, defined as the variance of each test divided by its average, as shown in Eq. 3).

$$IDI_{delay} = \frac{\sigma^2_{delay}}{\mu_{delay}} \quad (3)$$

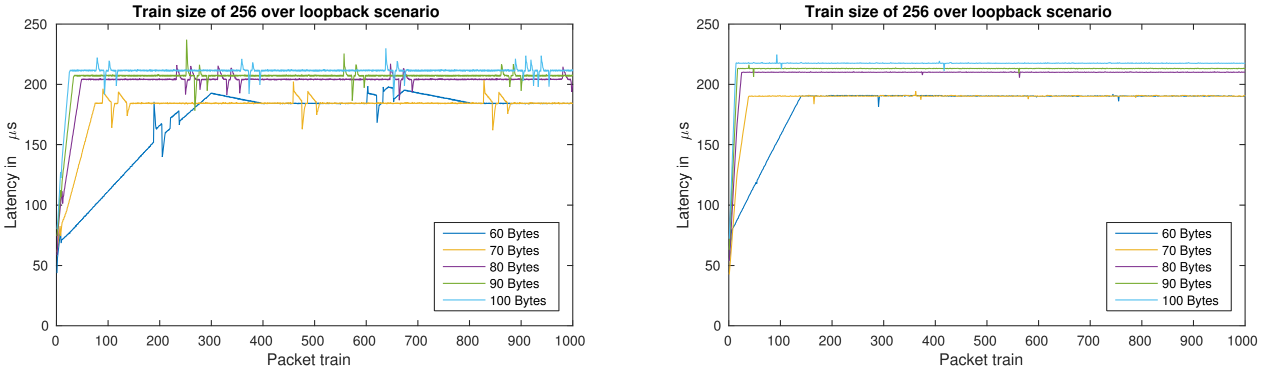
In order to improve visibility, values above 1.5 times the interquartile range from the mean have been considered as outliers and consequently removed.

When the frame length is close to certain sizes, there is a large variance peak (indeed the packet sizes in the vicinity also produce high variances), which are due to small misalignments in PCIe. PCIe works as a packet switched network (which is called *Transaction Layer Packet* (TLP)) with fixed packet size negotiated at boot-up. The TLP size varies from NIC to NIC because it depends on the manufacturer and link speed. When PCIe transfer is slightly larger than a TLP size multiple, it produces a PCIe bandwidth waste. For example, if the TLP size is fixed to 64 Bytes, a packet with size 65 Bytes requires 2 TLP packets of size 64 and a PCIe bandwidth waste of about a 50%.

In Figure 5 we observe spikes produced when packet size in the train is larger than 65 Bytes. Then, starting at 280 Bytes, and for each 288 Bytes, there is an increment in the IDI, which happens in bursts. This effect changes in intensity and periodicity over different machines and network cards, but it is present in every environment we have tried, and always repeatable for the same hardware and software combination. In this test, we have also noticed that the mean and median latency slightly varies

[7] `https://github.com/pktgen/Pktgen-DPDK`

[8] `https://github.com/hpcn-uam/iDPDK-LatencyMetter`



(a) Kernel with ticks enabled, without *isolcpus* and CPU in *powersave* (default) mode.

(b) Test executed with kernel and process isolation. The CPU is tuned in *performance* mode.

Figure 4. Latency measurements in 1000 consecutive trains, each one of 256 packets.

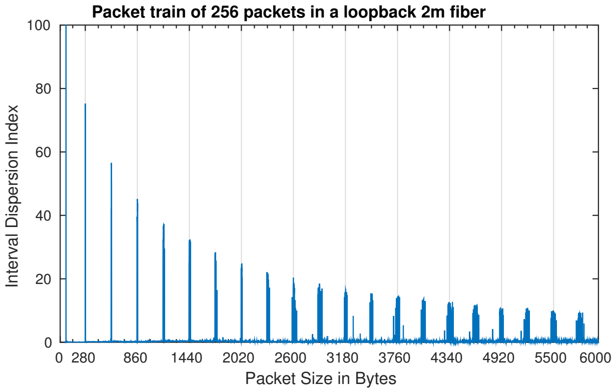


Figure 5. Index of dispersion of intervals (IDI) over 2 meters fiber. The test was carried out using different frame sizes and always 256 packets as train length.

for each 32-Byte increase, creating a saw-tooth shape in the measurements. We think both effects are related since the high variability peak is produced near 288 Bytes, which is 9 times 32. Furthermore, experiments in a different environment (CPU and NIC card) had a periodicity of 320 Bytes, which is exactly 10 times 32.

4.4. Queueing effects

Figure 6 shows the measured latency against the frame size and the inter-packet time (sleep time between packets) in two scenarios: 2 m and 2 Km fiber loops. The only noticeable difference between both scenarios is the minimal value: $4\mu s$ in Figure 6a and $13\mu s$ in Figure 6b. We clearly observe that severe measurement errors happen if either the frame size increases or the inter-packet time decreases, as the internal queues build up. In the case of small packets, the processing time for building and sending just one packet is large enough in our CPU to keep the queue empty. Thus, we advocate for the use of small packets, which provide a reliable measurement for all the targeted inter-packet times. We note that the delay overestimation is caused by the queueing effect of both the DPDK and the underlying hardware overhead, which makes the packet departure rate variable instead of constant. Since the packet arrival rate to the transmission subsystem is constant (sleep between packets) chances are that the outgoing packets already find packets in the transmission queue, which adds queueing delay to the measurement.

Despite the results with 60-Byte frames seem to provide an apparently stable measurement, a small variance can be observed, which is responsible for a small random noise. In order to actually verify that such variance is due to random operating system glitches and not to a hypothetical structural cause, we analyze the variance decay as the number of samples increases in a sample mean.

To this end, we consider a batch of N measurements, $X_1, \ldots, X_N$, which are deemed independent and identically distributed as the null hypothesis, with variance σ^2 and mean μ. As it turns out, the sample mean

$$S = \frac{1}{N}\sum_{i=1}^{N} X_i \tag{4}$$

is an unbiased estimator of μ and

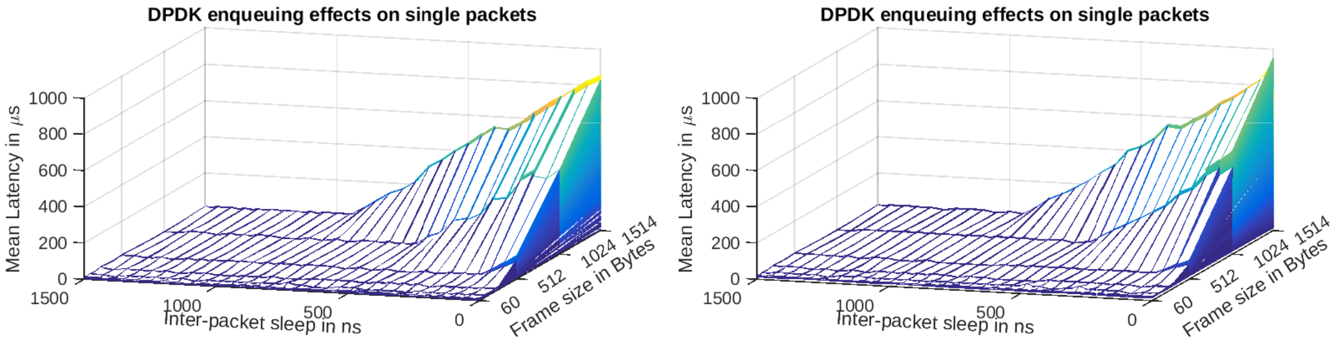
$$Var(S) = \frac{1}{N}\sigma^2 \tag{5}$$

We perform a thousand tests sending $N = 100$ packets very far apart in time ($15\mu s$) and calculate the corresponding sample mean S. From Eq. 5 the variance decay in logarithmic scale versus the number of samples N is equal to

$$Log\,(Var(S)) = Log\left(\sigma^2\right) - Log\,(N) \tag{6}$$

namely, a linear decay with $Log(N)$. The former equation is depicted in Figure 7, for a 2 m (left) and a 2 Km (right) long fiber respectively, which verifies our null hypothesis of independence. Thus, we hypothesize that the measurement variance is due to random operating system glitches.

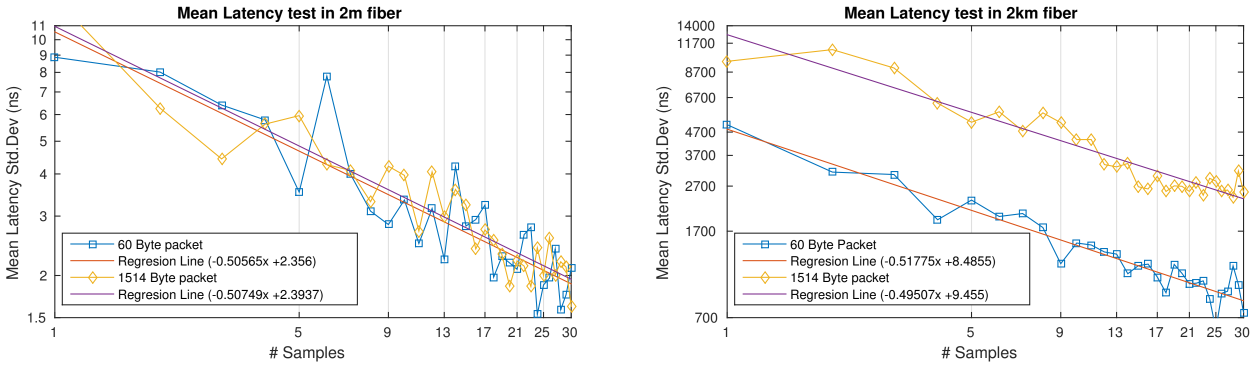
The same figure can be used to find out the number of times a 100-packets long train should be repeated in order to reduce the variance of the sample mean.



(a) 2 m long fiber loop.

(b) 2 Km long fiber loop.

Figure 6. Queuing effects with different frame size and different sleep times between packets.



(a) Test over 2 meters.

(b) Test over 2 kilometers.

Figure 7. Mean standard deviation convergence over a fiber using single packets.

At this point, we conclude that the usage of packet trains without control (saturating the queue) leads to severe measurement bias in DPDK.

4.5. Calibration

The former sections have shed considerable light into the dynamics of DPDK for high-precision latency measurement in optical networks. As it turns out, software-based optical network latency measurements are feasible provided that hardware features such as memory speed, CPU speed, PCIe version and number of lanes, and other low-level parameters are taken into account.

For example, following the memory specifications[9] it turns out that the time it takes to read 64 contiguous bytes in a DDR4 memory can vary from 15 to 20 ns depending on the memory module. Newer processors can handle up to 6 DDR4 channels, which implies that reading 64 consecutive bytes in a single channel architecture is equivalent in terms of latency to reading 384 consecutive Bytes in a six-channel architecture. Additionally, memory must have a refreshing progress, which is programmed by the memory controller. If the memory controller is not smart enough, it can try to read a memory address, which is in a refreshing state, yielding an undetermined waiting time until the data is finally read. The latter effect should happen very seldom, but in that case, it does introduce some delay variability.

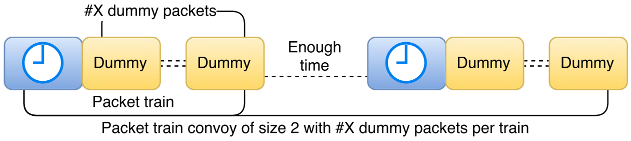

#X dummy packets
Dummy
Dummy
Enough time
Dummy
Dummy
Packet train
Packet train convoy of size 2 with #X dummy packets per train

Figure 8. Convoy of packet trains.

In the light of the above, we advise to perform calibration of the measurement device using a loopback fiber in the NIC. Estimation of the ground truth latency can be performed beforehand, with the fiber length and refractive index. By comparing the ground truth with the obtained measurements the low-level measurement noise, such as transit time through the NIC queues can be removed, even though such queueing effects are under the microsecond timescale.

5. Extensions to bandwidth measurement

Even though our focus is to measure optical network latency, we can easily extend our methodology to measure bandwidth as well. In this case, packets are sent at full speed in batches

[9] https://www.jedec.org/document_search?search_api_views_fulltext=jesd79-4%20ddr4

(packet trains) and the bandwidth estimation, in the receiving endpoint, is given by the length of the train (in bits) divided by its corresponding duration.

However, the fact that packets are sent at full speed entails, necessarily, that the *mbufs* queue should be saturated and, therefore, the latency measurement is severely distorted, as shown in Figure 6. In a first approximation, we have considered using a single packet pair to estimate the bandwidth [18]. Nonetheless, this solution is not suitable for many network scenarios. For example, for an aggregated LACP link of *n* physical fibers, a single packet approach would actually produce exactly *n* times less bandwidth that available. In addition, the transmission pattern of the packet pair depends on the LACP policy used, which, in turn, may distort the measurement. In order to overcome this issue, we propose a novel methodology to measure both bandwidth and latency, which we denote by *convoy of packet trains*.

Briefly, we send several packet trains at full speed, which saturate the transmission queue, but only the first packet in the train is used to measure latency. We note that the first packet is the head-of-line in the queue and does not incur into the waiting time that distorts the latency estimation. Therefore, only the first packet in the train is timestamped, being the rest dummy packets that serve to measure the link capacity (see Figure 8). We stress that such timestamp should be written right before the whole `mbufarray` is sent to the queue to minimize the waiting time for this head-of-line packet. We also note that the convoy of packet trains allows estimating losses in the network, as the ratio of dummy packets lost.

In Table 1 we show the results obtained with our convoy of packet trains methodology with the 2 m and 2 Km long fibers, respectively. The test was performed with a convoy of 100 trains, with 100 packets each.

The expected results for bandwidth measurements should be equal in both scenarios, namely 9.844 Gb/s for 1518-Byte long frame and 7.143 Gb/s for a 60-Byte long frame, respectively. Thus, the results shown in Table 1 are very accurate. The difference in bandwidth obtained between large and small packets is caused by the Ethernet frame overloading (CRC, prelude, and inter-frame gap). In terms of latency, in the 2 meters fiber, the results should be 144 ns for 60-Byte frames and 2.470 ns for 1518-Byte frames. At this point, we must notice that the error seen with 60-Byte frames and 1518-Byte frames are not the same. As explained before, this error is constant for a fixed frame size, but it does not have a linear dependency with the frame length. The results expected for the 2 Km fiber are 10,057 ns and 12,383 ns for 60-Byte and 1518-Byte, respectively. As expected, the difference between 2 m latency and 2 Km are constant between packet sizes.

As a downside, the proposed methodology demands a larger number of packets and consumes a larger bandwidth from the measured optical network link. It can also interfere with production traffic. Thus, it may be used in combination with latency measurements with a single packet (or small packet trains). For example, measuring latency during peak hours and bandwidth off-peak seems appropriate as a measurement policy that allows accurately portraying the overall performance of the optical network link.

Table 1: Bandwidth estimation results obtained in a loopback scenario using convoy of packet trains

Fiber Length	Frame Size	Lat. estimated	BW. estimated
2 m	60 Bytes	582 ns	7.1 Gb/s
2 m	1518 Bytes	6,685 ns	9.8 Gb/s
2 Km	60 Bytes	10,510 ns	7.1 Gb/s
2 Km	1518 Bytes	16,777 ns	9.8 Gb/s

Conclusions

In this paper, we have proved the feasibility of DPDK-based measurement engines to measure delay in a high-speed optical network at the microseconds timescale, provided that a careful tuning of the different DPDK and system parameters is performed. We have also contributed with an open-source measurement software that serves to this purpose. We believe that the proposed methodology, along with the results presented in this article, can help researchers and practitioners in the field to accurately and cost-effectively measure latencies in high-speed optical networks.

Acknowledgment

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund under the project TRÁFICA (MINECO/FEDER TEC2015-69417-C2-1-R), and by the European Commission under the project H2020 Metro Haul (Project ID: 761727).

References

[1] A. Johnsson, On the comparison of packet-pair and packet-train measurements, in: Proc. Swedish National Computer Networking Workshop, 2003, pp. 241–250.

[2] J. Ramos, P. Santiago del Río, J. Aracil, J. E. López de Vergara, On the effect of concurrent applications in bandwidth measurement speedometers, Computer Networks 55 (6) (2011) 1435 – 1453. doi:http://dx.doi.org/10.1016/j.comnet.2010.10.022.

[3] M. Ruiz, J. Ramos, G. Sutter, J. E. López de Vergara, S. López-Buedo, J. Aracil, Accurate and affordable packet-train testing systems for multigigabit-per-second networks, IEEE Communications Magazine 54 (3) (2016) 80–87. doi:10.1109/MCOM.2016.7432152.

[4] A. Tockhorn, P. Danielis, D. Timmermann, A configurable FPGA-based traffic generator for high-performance tests of packet processing systems, in: 6th International Conference on Internet Monitoring and Protection (ICIMP), 2011, pp. 14–19.

[5] Y. Wang, Y. Liu, X. Tao, Q. He, An FPGA-based high-speed network performance measurement for RFC 2544, EURASIP Journal on Wireless Communications and Networking 2015 (1) (2015) 2.

[6] J. W. Lockwood, M. Monga, Implementing ultra low latency data center services with programmable logic, in: 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects, 2015, pp. 68–77. doi:10.1109/HOTI.2015.20.

[7] R. Olsson, pktgen the linux packet generator, in: Proceedings of the Linux Symposium, Ottawa, Canada, Vol. 2, 2005, pp. 11–24.

[8] D. Turull, P. Sjödin, R. Olsson, Pktgen: Measuring performance on high speed networks, Computer Communications 82 (2016) 39 – 48. doi:https://doi.org/10.1016/j.comcom.2016.03.003.

[9] 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.

[10] S. Han, K. Jang, K. Park, S. Moon, PacketShader: a GPU-accelerated software router, in: ACM SIGCOMM Computer Communication Review, Vol. 40, ACM, 2010, pp. 195–206.

[11] V. Moreno, J. Ramos, P. M. Santiago del Río, J. L. García-Dorado, F. J. Gomez-Arribas, J. Aracil, Commodity packet capture engines: Tutorial, cookbook and applicability, IEEE Communications Surveys Tutorials 17 (3) (2015) 1364–1390. `doi:10.1109/COMST.2015.2424887`.

[12] A. Beifuß, T. M. Runge, D. Raumer, P. Emmerich, B. E. Wolfinger, G. Carle, Building a low latency linux software router, in: 2016 28th International Teletraffic Congress (ITC 28), Vol. 01, 2016, pp. 35–43. `doi:10.1109/ITC-28.2016.114`.

[13] S. Han, K. Jang, K. Park, S. Moon, Building a single-box 100 gbps software router, in: 2010 17th IEEE Workshop on Local Metropolitan Area Networks (LANMAN), 2010, pp. 1–4. `doi:10.1109/LANMAN.2010.5507157`.

[14] S. Ma, J. Kim, S. Moon, Exploring low-latency interconnect for scaling out software routers, in: 2016 2nd IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB), 2016, pp. 9–15. `doi:10.1109/HIPINEB.2016.12`.

[15] C. Wang, O. Spatscheck, V. Gopalakrishnan, Y. Xu, D. Applegate, Toward high-performance and scalable network functions virtualization, IEEE Internet Computing 20 (6) (2016) 10–20. `doi:10.1109/MIC.2016.111`.

[16] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, G. Carle, Moongen: A scriptable high-speed packet generator, in: Proceedings of the 2015 ACM Conference on Internet Measurement Conference, ACM, 2015, pp. 275–287.

[17] IEEE Standard for Ethernet, IEEE Std 802.3-2015 (2016) p. 151`doi:10.1109/IEEESTD.2016.7428776`.

[18] J. Curtis, T. Mcgregor, Review of bandwidth estimation techniques, in: In New Zealand Computer Science Research Students' Conference, 2001.