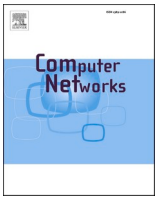
Computer Networks 185 (2021) 107689

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Formullar: An FPGA-based network testing tool for flexible and precise measurement of ultra-low latency networking systems

Taejune Park [a], Seungwon Shin [a], Insik Shin [a], Kilho Lee [b,*]

[a] KAIST, Republic of Korea
[b] Soongsil University, Republic of Korea

ARTICLE INFO

Keywords:
Network
Performance measurement
Hardware
Ultra-low latency

ABSTRACT

As network systems become widespread in emerging time-sensitive domains, latency becomes an important performance factor. Thus, there is a new demand for ultra-low latency (ULL), which requires an extra-ordinarily low latency even in a deterministic manner. To support the ULL requirement, it is mandatory that each network device strictly guarantees this latency. As a result, there is an increasing demand for a precise latency measurement tool that achieves two conflicting goals: high time precision and flexible traffic control. To support these goals, this paper proposes a novel measurement tool, named Formullar, which has a hybrid architecture composed of hardware (Formullar FPGA) and software (Formullar controller) layers. Formullar generates packets in an accurate timing, according to the various traffic patterns. The packet generation is controlled in a fully programmable manner that makes the testing fully automated and flexible. Then, latency is measured based on packet generation and reception timing; the measurement is done by the hardware layer, providing highly precise measurement results. Using this carefully constructed system, a Formullar prototype was implemented on top of the commodity FPGA board NetFPGA-SUME. The evaluation indicates that Formullar provides highly precise measurements with nanoseconds precision at a bandwidth of 10 Gbps.

1. Introduction

As network systems have become widespread in emerging time-sensitive domains, *latency* has become a key factor in terms of determining and specifying system performances directly. A network must guarantee an end-to-end latency that falls within system requirements, even when packets must go through various network elements (e.g., switches and middle boxes) introducing processing delays [1].

Hence, there is new demand for extraordinarily low (even deterministic) communication delays, i.e., *ultra-low latency* (ULL) [2–5]. For instance, an autonomous car requires less than a 10 μs communication delay in order to maintain control of its powertrain/chassis [6], and the Tactile Internet [4] offered by fifth-generation (5G) mobile networking demands less than 1 ms delays. Such requirements should be strictly enforced, as late delivery of a *single* packet could result in catastrophic results, such as a car crash caused by a delayed reaction on the part of the braking system or significant QoE (Quality of Experience) degradations. Therefore, to satisfy such latency requirements, the next-generation network associations have even standardized ULL communications; IEEE TSN (Time-Sensitive Networking), a deterministic Ethernet standard for vehicular/industrial networks, offers ULL communication with less than a 100 μs delay [7]. The 5G mobile networking standard also attempts to support a maximum end-to-end delay of 1 ms [2].

In this respect, in order to realize *reliable* ULL-based time-critical systems, it is essential to confirm whether or not the latency in a system is guaranteed; thus, an accurate latency measurement is mandatory for the system testing process. To ensure that a system can guarantee a required latency in various operating environments, it is necessary to go beyond simulation or analysis-based verification and institute measurement-based validation.

To get reliable data, the test must be conducted while (1) collecting very precise timing measurements and (2) generating various traffic patterns. Thereby, the goal of our work is to develop a novel measurement system to achieve those requirements. However, it is not trivial to measure ULL accurately since any attempt to do so must support two conflicting requirements. The first requirement is that packets must be handled with extremely high precision since latency should be measured in very small time units, such as μs (even ns). Traditional latency measurement techniques (e.g., ping) cannot support this requirement

* Corresponding author.
E-mail address: khlee.cs@ssu.ac.kr (K. Lee).

https://doi.org/10.1016/j.comnet.2020.107689
Received 10 August 2020; Received in revised form 28 October 2020; Accepted 16 November 2020
Available online 23 November 2020

because they typically consist of software-based approaches. With μs and ns precision, a small software overhead (e.g., scheduler, interrupt handler, and network protocol suite) accumulates in the packet timestamps and becomes a huge noise in latency measurements. One could claim that a hardware-based measurement tool can support high precision; however, it cannot support another important requirement, i.e., flexible traffic generation. Another requirement is flexible control of traffic (and each packet) generating patterns. The measurement tool should be able to generate packets with various headers and payloads, according to the functionality of each network device. For instance, to test the performance of a layer-3 router, each packet should contain a proper IP-header field. In addition, the tool should be able to control traffic generating patterns in order to validate a system in various operating environments. The traffic pattern can be represented by *pps* (packets per second), throughput, burst, and even periodic messages. Software-based approaches can easily provide such flexible control; however, as mentioned above, they cannot support precision to the degree it needs to be supported. In contrast, hardware-based approaches are not suitable for flexible control due to their static nature.

Thus, unfortunately, traditional measurement techniques are not able to satisfy these requirements, particularly in terms of measuring ULL. Representative measurement tools [8,9] focus on throughput because throughput often becomes a dominant factor when determining performance, e.g., QoE (quality of experience) for cloud-based services. Some studies consider the latency measurement [10–12]; however, these techniques cannot support the highly precise (i.e., μs precision) latency measurements required for ULL systems.

The goal of this paper is to develop a novel latency measurement tool tailored to ULL systems that satisfies two conflicting requirements: high precision and flexible control. It entails three system challenges, as follows:

C1. It should provide a reconfigurable/programmable packet generator, including packet headers, payloads, and even traffic patterns.
C3. It should provide high measurement precision.
C2. It should generate various traffic patterns based on well-defined traffic models.

To this end, we combine the software- and hardware-based approaches and propose a hybrid system, named Formullar. To address the challenges listed above, Formullar employs: (1) a hybrid architecture composed of a software-based controller and a hardware-based (i.e., Field Programmable Gate Array (FPGA)) module, (2) a hardware-based deterministic packet generator/receiver, and (3) a general traffic generating model. The operating principle of Formullar involves decoupling the director and executer, i.e., measurements are performed on the hardware module under configurations from the host software. The hardware module is designed to exclude as many factors that could potentially affect the measurement results as possible; thus, it can generate/receive measurement packets at specified times and record their timestamps without external control. This benefit allows it to manage the test traffic based on the software controller's configurations, enabling a network administrator to program test benchmarks for the automation of measurements under various traffic models designed for changing conditions. As a result, Formullar can produce measurements of high purity and also provide the flexibility and convenience of software in terms of network measurements.

We implemented the prototype of Formullar using NetFPGA-SUME [13,14]. Our prototype supports a precision of 6.4 ns with 10 Gbps of speed. In our evaluation, our prototype shows that its internal processing delay is very low, i.e., approximately 0.66 μs with a standard deviation of just 0.004 μs, meaning that the time variation is very small. In addition, the time precision for packet generation/receipt is very precise (almost zero variance), while the software-based approach has time deviations of hundreds of μs. As a result, Formullar can achieve the ns level of precision required to measure the target device's packet processing overheads, while the software-based approach cannot achieve this same feat due to the internal noises caused by complicated software layers.

This paper's contributions can be summarized as follows:

- We propose a highly precise and flexible measurement tool for ULL systems; in addition, to the best of our knowledge, this is the first work to consider ULL measurements with high precision.
- We propose an effective system design based on a hybrid architecture, which consists of a software-based controller and a hardware-based packet generator.
- We implement the proposed system on a commodity FPGA network device.
- We present an extensive evaluation on top of the real-world testbed.

This paper is organized as follows. Section 2 defines the problem, and Section 3 presents the system design. Section 4 describes the implementation. Next, Section 5 evaluates Formullar's performance and compares it to other approaches. We then present related works in Section 6. Finally, we conclude the paper in Section 7.

2. Challenge and research goal

2.1. Challenge

In this paper, we aim to develop a network latency measurement tool that generates packets corresponding to networking devices, sends/receives the generated packets, and measures the time spans involved in the packets' round trips. The measurement tool faces a number of challenges in terms of testing various ULL-based systems with high measurement precision, which are detailed below.

Programmability. Since a ULL-based system consists of various networking devices, the tool should generate packets with various headers and payloads, according to the functionality of each network device. In addition, each device should be tested with varying traffic patterns. Therefore, in order to be categorized as a general-purpose tool, the measurement tool must be reconfigurable; it should provide as many as possible control knobs, including packet content, transmission timing, and traffic pattern knobs, as possible. Furthermore, to maximize usability, such control knobs should be easily programmable with well-defined interfaces.

High precision. As ULL-based systems require extremely low latency, each device within such a system should guarantee such ultra-low latency. To test for ULL in each device, the tool should be able to capture very small-scale time units (i.e., μs/ ns precision) and support strict time management within those units [2]. For instance, each packet to be measured must be generated/sent at the exact time specified, and the arrival time of each incoming packet must be recorded with a highly precise timestamp. The tool should manage the timing with ns precision; to this end, it should eliminate any unexpected overhead that may affect the measurement result. If not, a small overhead may result in huge noise in the measurement. For instance, an overhead taking up to 1000 clock cycles on a 100 MHz machine may result in a noise of up to 10 ms, which may be unacceptable for ULL systems.

General traffic model. To ensure that an ULL system can guarantee the latency requirement even in the worst possible scenario, the measurement tool should control the traffic patterns precisely. Additionally, to generate various traffic patterns (e.g., constant bit rate, variable bit rate, or periodic messages), a well-defined general traffic model is required. In addition, it is important to test the system with varying workloads; the general model is also required to change with the workload. Therefore, various traffic-generation patterns should be modeled in a reproducible form [15–17].

2.2. Existing solutions

Here, we present existing approaches and their limitations in terms of ULL latency measurements [18,19].

Software-based approach. A common method for making latency measurements is to use software tools such as hping [20], Netperf [9] or Iperf [8]. They are easy to use and provide a variety of options for generating traffic, thereby enabling the testing of multiple cases. However, as they rely on multiple software layers, including operating systems, running many processes, they are typically influenced by the behavior of that software, such as I/O interrupts and even OS threads, and their software timer cannot meet the precision requirements (i.e., µs level) [21,22]. In addition, since it runs on top of an operating system, it uses many steps (i.e., call stacks) to generate and receive packets for measurement. Furthermore, small amounts of noise from the steps accumulate and increase the packet timestamps, eventually resulting in a huge error in the measurement result [23]. In summary, although the software-based approaches can provide reconfigurable packet generation, they cannot provide enough precision for ULL measurement.

Hardware-based approach. Another way to measure the latency is to use a special instrument, such as an oscilloscope, or custom measurement hardware [24,25]. Hardware-based approaches include highly accurate timers and deterministic behavior based on the number of clock cycles; for instance, they take a fixed number of cycles to generate and receive packets. Therefore, hardware-based approaches can easily support highly precise latency measurements. However, in marked contrast to software, hardware requires the manual configuration of every bitstream in every packet; hence, the hardware circuit must be redesigned whenever other packet/traffic patterns need to be generated. A few studies present FPGA (Field Programmable Gate Array)-based approaches [26–30]. Although they provide precise network performance measurements, they suffer from lack of programmability due to replay-based traffic generation [26–28], limited configurability (i.e., packet size and inter-packet interval) and performance of 1 Gbps [29], or requiring hardware-level programming [30]. Consequently, although hardware-based approaches are good for high precision, they have difficulties with reconfigurable measurements.

2.3. Research goal

When it comes to ULL latency, we feel that there is a huge demand for a new measurement tool that can provide reconfigurable traffic generation and highly precise latency measurements. Since both hardware- and software-based approaches have shortcomings, the goal of this paper is to design and implement a new system that addresses the aforementioned challenges effectively.

To achieve our goal, we propose a hybrid measurement system, named Formullar, based on Field-Programmable Gate Arrays (FPGA). It generates packets for measurement within a hardware layer that is able to support precise timing and eliminate any unintended interference, thus enabling high precision. Here, we generalize traffic patterns and structurize their creation, thus Formullar can generate various traffic models required for latency measurement in a fully programmable manner. Therefore, it also provides programmable control knobs for, among other features, packet contents, and traffic patterns through a software-layer controller that communicates with the FPGA.

3. System design

3.1. System overview

As shown in Fig. 1, Formullar employs a hybrid architecture that consists of two parts, Formullar FPGA (hardware) and controller (software). This new architecture plays a major role in enabling a reconfigurable latency measurement tool (C1) while providing high precision (C2) and a general traffic model (C3). After connecting Formullar to the target device, a user can easily (re-)configure Formullar through executing a simple application (or script) running on top of the Formullar controller. In this application, Formullar FPGA generates packets and sends them to the target device while utilizing strict timing control; next, it receives packets returning from the device and captures receipt times without any external noise (e.g., interference generated by OS routines). Formullar FPGA eliminates internal/external measurement noises, and utilizes deterministic packet generation/reception timing. In addition, Formullar controller provides a general traffic mode, i.e., by relying on the well-defined APIs provided by this controller, a user can easily generate various traffic patterns with a short application.

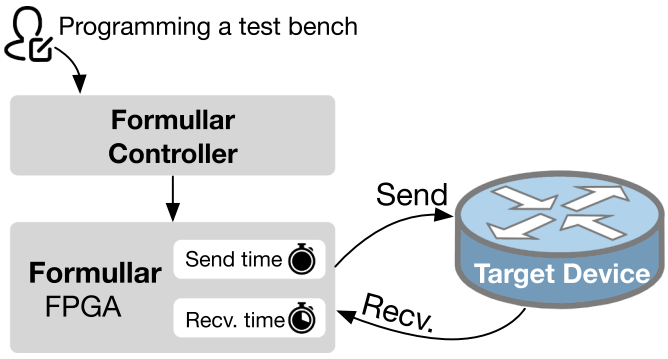


Fig. 1. Formullar overview.

3.2. Traffic modeling

Since latency depends highly on traffic patterns, including packet size, bitrate, burst, and so on, it is essential to test networking devices with varying traffic patterns. In order to do so, we establish a general traffic model that can represent various traffic patterns, including CBR (constant bit rate), VBR (variable bit rate), and even periodic message patterns. Fig. 2 depicts the model; we define each *flow* as an (unbounded) series of *packets* grouped by *messages*. Note that each flow corresponds to a network application session, which is typically determined by a tuple (e.g., a 5-tuple consisting of protocol, source/destination addresses, and source/destination ports). Each flow generates a set of messages $\mathscr{M}$, and each message $m_i \in \mathscr{M}$ is generated until i reaches the total number of messages NM. Note that for ease of presentation, the flows are not indexed; however, each flow has its own message set $\mathscr{M}$. Each message m_i is determined by the following set of attributes:

$\langle p_{i,j},\ len(m_i),\ len(p_{i,j}),\ intv(m_i),\ intv(p_{i,j}),\ N_i \rangle$, where $p_{i,j}$ represents the j-th packet generated by the message m_i; $len(m_i)$ and $len(p_{i,j})$ are the lengths of m_i and $p_{i,j}$, respectively; $intv(m_i)$ and $intv(p_{i,j})$ present the time intervals between the two messages (m_i and m_{i+1}) and the two packets ($p_{i,j}$ and $p_{i,j+1}$), respectively; and, N_i is the number of packets generated by each m_i. In addition, please note that an attribute without an index indicates all elements specified by that omitted index; for instance, p_i (omitting the index k) represents all packets generated by m_i (i.e., $\forall_k p_{i,k}$).

With these parameters, traffic patterns can easily be controlled in various ways. For instance, we can increase the bandwidth by increasing $len(m_i)$ and $len(p_{i,j})$; we can also easily generate a burst by decreasing $intv(m_i)$ and $intv(p_{i,j})$. Each request made of a network system is typically divided into multiple packets due to physical limitations (i.e., the maximum transmission unit (MTU) of links). We can simulate this situation by controlling the set of packets ($p_{i,j}$ and $len(p_{i,j})$) in each message. Furthermore, it is worthwhile to note that periodic traffic can be generated by controlling $intv(m_i)$ and $intv(p_{i,j})$. When $intv(m_i)$ is set to 0, the flow will periodically generate packets with inter-packet intervals $intv(p_{i,j})$. If $intv(m_i)$ is set to be non-zero, then the flow can generate periodic messages (bursts) with inter-message intervals $intv(m_i)$. With this model, it is also possible to generate traffic patterns using other representations. In particular, the model can control PPS (packets per

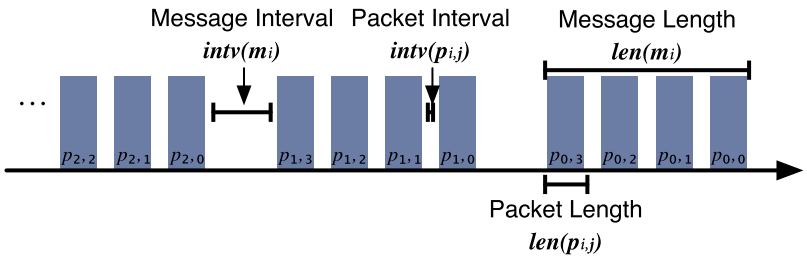


Fig. 2. Traffic model.

second), bps (bits per second), and bursts and can generate CBR/VBR patterns; in addition, it can also emulate heart-beat signal patterns when $len(m_i)$ is set as small as possible and $intv(m_i)$ represents a heartbeat interval. This wide range of traffic generation capabilities will be very helpful in terms of testing various ULL-based systems, from video-streaming systems to autonomous driving vehicle systems.

3.3. Formullar *FPGA*

As depicted in Fig. 3, the main idea for achieving high precision in the Formullar system is to employ an FPGA-based packet handler utilizing deterministic timing. Based on its deterministic behavior, Formullar FPGA eliminates any internal overhead and interference for the core modules *shared timer, packet generator, packet listener*, and *control interface*.

The shared timer is a cycle counter driven directly by the FPGA oscillator hardware; it increases the counter by every clock cycle. All Formullar modules are time-synchronized since they share this timer. In addition, the timer provides maximum precision along with physical limitations. For instance, this implementation can provide 6.4 ns precision with a 160 MHz FPGA board (see Section 4 for further details). In addition to eliminating any internal overhead, Formullar FPGA dedicates a set of packet generators/listeners to each network interface. All packet generators/listeners operate in parallel, which means that, at a specific instant, a packet can be handled on each interface. This architecture can eliminate additional internal scheduling overhead (e.g., round-robin scheduling) involved in switching network interfaces. Based on the shared timer and dedicated packet generators/listeners, Formullar FPGA strictly manages packet generation timings and precisely measures packet sending/receiving timings.

For each measurement, the packet generator generates each packet whose size and timing are configured by the controller. Fig. 4 details how the packet generator works. The configuration attributes constraint the interval or the number of repetitions during the traffic generation process. When starting a measurement, the packet assembler builds a bitstream for a defined packet (e.g., header and payload) over several clocks to match the packet length ($len(p_{i,j})$), and the generated packet is directly transferred to a target device through an interface. The packet assembly repeatedly makes packets with the time interval $intv(p_{i,j})$, until the number of generated packets reaches to *NM*. If the message is finished, it waits for the message interval time $intv(m_i)$ and starts to generate new packets for the next message. The configurations can be

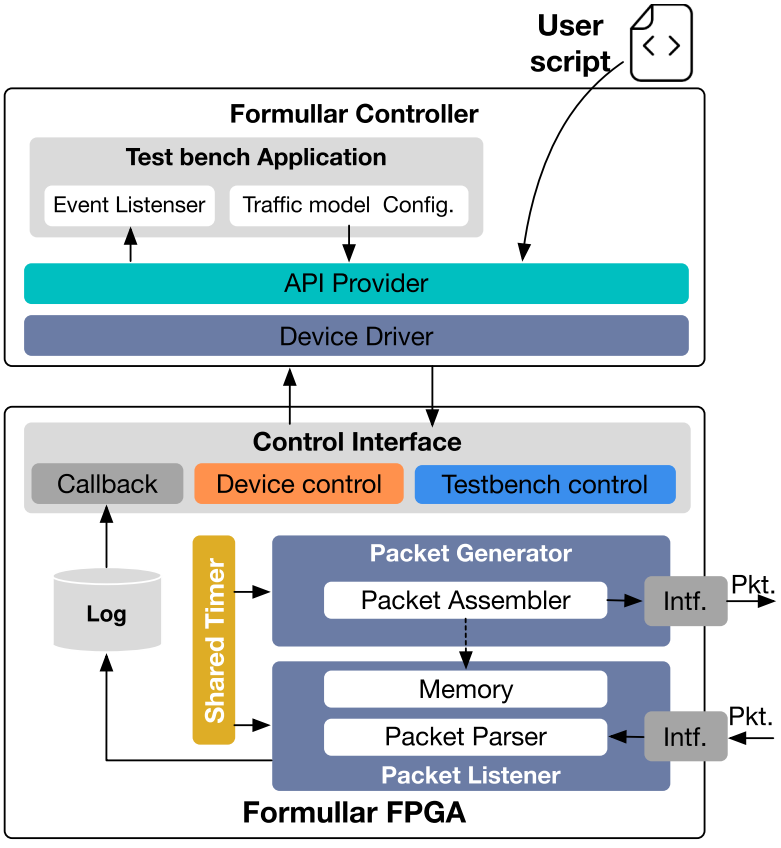


Fig. 3. Formullar architecture.

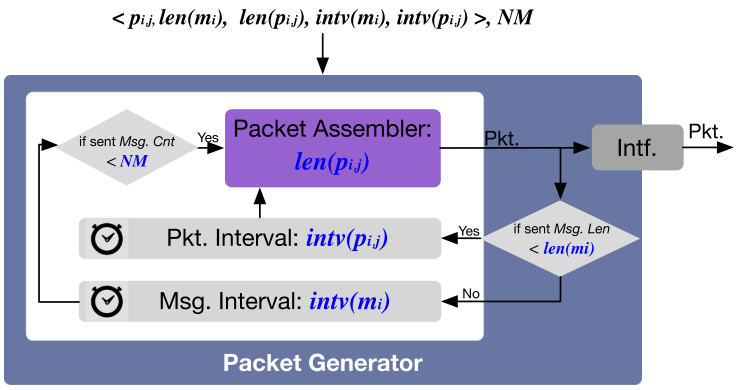


Fig. 4. Packet generator.

programmable even at runtime, and so that measurement scenario can be controllable depending on the runtime context. It is worthwhile to note that the packet generator accepts multiple traffic attributes in series. For instance, when a user wants to apply different time intervals for two distinct packets $p_{i,j}$ and $p_{i,j+1}$, then the user can configure the attributes $intv(p_{i,j})$ and $intv(p_{i,j+1})$ with different values and feed them into the generator in turn. It helps the packet generator to be more flexible.

And, upon receiving packets from the target device, the packet listener captures and parses the response time. The Formullar FPGA stores all timing associated with generating/sending/receiving each packet and shares this timing information with the controller through the log memory and control interface.

3.4. Formullar *controller and interface*

As mentioned previously, the key to rendering Formullar reconfigurable is its hybrid architecture. In order to realize this architecture, it is necessary to design the controller and interface between the hardware and software layers carefully. Fig. 3 depicts the architectures of Formullar controller and interfaces. The main ideas behind the interface are: (1) using a set of registers to control/configure the internal components of Formullar FPGA, such as the packet generator (see *Control Interface*); (2) constructing a *device driver* that can access those registers through I/O operations; and (3) providing a set of APIs (see *API Provider*) that allow Formullar applications to control the device easily while hiding the low-level details of primitive device driver operations.

The control interface on Formullar FPGA has three types of registers: (1) testbench control, (2) device control, and (3) callback. The testbench control registers are used to configure the packet generator. The packet generator should generate packets according to the given traffic model; in order to do so, the packet generator refers to the testbench control registers, which contain the values of the traffic model parameters given by Formullar application. The device control registers contain the control signal and device status; for instance, Formullar application can start/stop/pause/reset the FPGA through these registers. The callback registers are used to execute callback functions in the device driver; they are used for copying the measurement results updated in the log memory to the controller. When the device driver invokes a callback, it then executes the *event listener* (pre-registered as a callback) of the application in order to collect the measurement results.

Algorithm 1 provides an example of a simple Formullar application. After opening a Formullar case, it first sets the parameters for the traffic model. In this example, traffic is set to 1000 messages (set *NM*) with four packets per message (set *N*) and an inter-message generating interval of 100 ms (set $intv(m)$). In addition, each packet has a length of 128 bytes and an inter-packet generating interval of 0 (set $len(p)$ and $intv(p)$). Note again that notations omitting the subscript represent all elements corresponding to that omitted subscript; for instance, m and p represent all messages ($\forall_i m_i$) and all packets ($\forall_i \forall_k p_{i,k}$), respectively. After configuring the traffic model, the algorithm begins measuring by executing `formullar_start()`. The measurement results can be monitored easily through the listener callback API. The callback provides the latency values with metadata, such as the message/packet IDs corresponding to m_i and $p_{i,k}$ (see RESULT.MID and RESULT.PID, respectively). The metadata also contains timing information so that the controller can access the latency. The controller can monitor the latency continuously and control Formullar in a programmable manner. As seen in the *if* statement in Algorithm 1, if the latency is less than 100 μs after the 50th message has been received for Case ID 10, the size of each packet and the number of packets in each message will be doubled to change the testing condition and monitor the changes in latency results. After defining the callback, the application waits until the measurement is done, then it closes the Formullar case.

Note that the time complexity of a Formullar application highly depends on implementation. However, an application typically consists of two parts: traffic attribute setup and a callback function. In the traffic attribute setup part, it may require num_{attr} times of memory accesses, where num_{attr} represents the number of attributes to be configured. And, since a callback function is typically used to change the value of traffic attributes, it also requires num_{attr} times of memory accesses. After memory accesses, an algorithm may use Formullar APIs to communicate with Formullar-FPGA. Since each API is for communication, its time complexity is proportional to num_{attr}. Thereby, as long as the algorithm does not contain other complicated computation logic, it may have $O(num_{attr})$ as its time complexity. For instance, Algorithm 1 presents a typical Formullar use case, which makes the time complexity of $O(num_{attr})$.

After applying our programmable traffic model, Formullar can generate various traffic patterns to emulate distinct situations, by carefully configuring the value of each attribute of the model.

For example, a periodic message or stream traffic can be easily generated without any complicated configuration. (1) The periodic message (Fig. 5a) is widely used in network applications for a heartbeat or data polling, and Formullar can model this pattern by setting the intervals ($intv(p)$ and $intv(m)$). It is also possible to perform priority-based latency comparisons by assigning different intervals for multiple periodic messages [31]. (2) The stream patterns (Fig. 5b) can emulate video, audio, VR (Virtual-Reality), and other multimedia traffic patterns. Moreover, it can emulate traffic bursts for stress testing such as flash crowds or Denial of Service (DoS) attacks. It is achieved by setting the intervals ($intv(p)$ and $intv(m)$) to zero, and the bandwidth is also configurable by adjusting the packet size ($len(p)$). It can also represent a sudden increase in bandwidth by updating the packet size at a specific time t'. In addition to this, irregular traffic, traffic with certain conditions can be reproduced in a programmable manner, and the latency can be evaluated accordingly with those circumstances.

In addition to those examples, Formullar can generate more complicated patterns by configuring traffic attributes packet-by-packet. Note

```
f = formullar_open()
c1 = formullar_case(10) /* Case ID (CID) */ c1.NM = 1000 /* messages */
c1.N = 4 /* packets in a message */
c1.intv_m = 100 /* ms gaps between messages */
c1.len_p = 128 /* bytes per packet */
c1.intv_p = 0 /* ms between packets */
formullar_start(f, c1)
Procedure formullar_listener_callback(RESULT)
    print(RESULT.CID, RESULT.MID, RESULT.PID, RESULT.LATENCY)
    if RESULT.MID ≥ 50 and RESULT.LATENCY ≤ 100 μs then
        /* Traffic pattern changes to */
        f.len_p = 256 /* bytes per packet */
        f.N = 8 /* packets in a message */
        formullar_update(f)
    end
formullar_wait(f)
formullar_close(f)
```

Algorithm 1. Example of a Formullar application.

that Formullar is capable to configure each packet's attribute (e.g., $len(p_{i,j})$). Besides, beyond such deterministic traffic patterns, Formullar is also possible to generate some statistical patterns, e.g., Poisson distribution traffic model [32]. To do this, we only need to put appropriate traffic attribute values (e.g., $intv(m_i)$) which follow the statistical model. Note that we can easily calculate such statistical values through a random number generator.

Despite the high programmability, it is hard for Formullar to generate some patterns. For instance, to generate a rapidly changing traffic pattern is not easy, because it may require a lot of packet-by-packet traffic attribute values. And, when traffic is sensitively changed in response to network conditions, it is not easy to generate a pattern since it may require complicated callback functions. We expect that such limitations can be addressed if we develop more general traffic model and programming model. However, since optimizing the model is another open research issue, we leave it as future work.

4. Implementation

In order to validate the feasibility of Formullar's design, a prototype of Formullar FPGA was implemented using NetFPGA-SUME (Fig. 6), which is an FPGA-based PCI Express board with four SFP+ 10 Gbps interfaces and a Xilinx Virtex-7 XC7V690T [13,14]. The codes are written in Verilog and synthesized with Xilinx Vivado. Fig. 7 provides the implementation details; this prototype runs at a 156.4 MHz clock rate, meaning that it has an accuracy of 6.4 ns. The packet generator/listener modules share the timer and are connected to network interfaces. The control interface provides input registers to configure the device and traffic parameters and is connected to the control channel so that it can communicate with the Formullar controller. The log memory is implemented using block-ram. The prototype consumed about 28,000 Look-Up Tables (LUTs), indicating the FPGA resource usage.

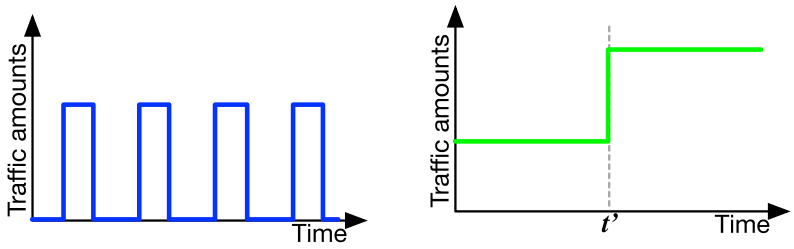
Fig. 8 shows the implementations of the control channel and controller. The control channel provides two ways to communicate with the controller, i.e., Direct Memory Access (DMA) and Ethernet network interface. (1) The DMA corresponds to the PCI express with the AXI protocol [33], and the hardware is operated by directly communicating with a device driver. The device driver is implemented via a reference device driver in NetFPGA-SUME [34] and extended to support two engines corresponding to the DMA and network channels. The communication messages are stored in the proxy memory and delivered to Formullar applications through the API. The API provider is implemented as a set of C libraries, and it communicates with the device driver through the *ioctl* and *netlink* protocols, thereby allowing the host API layer to enforce the control parameters and read the memory logs. It is worthwhile to note that although our prototype was implemented on top of Ubuntu 14.04 (i.e., the recommended version of NetFPGA-SUME [13]), it is also compatible with the latest Linux version; we confirm that Formullar works on Ubuntu 20.04. In addition, since Formullar utilizes the standard PCI-E interface, it is possible to port Formullar to other operating systems including Windows as long as we have a proper PCI-E device driver.

5. Evaluation

In this section, Formullar prototype implementation is evaluated. Note that the evaluation focuses mainly on the precision of the latency measurement since other requirements (i.e., reconfigurability and the general traffic model) can be achieved via the hybrid architecture and software-based controller.

5.1. Evaluation setup

To evaluate Formullar, we show how precisely Formullar measures the target device's latency. Recalling that ULL-based systems are of primary interest, the target device is set to have very low latency (i.e., less than 1



(a) Periodic messages (b) Stream

Fig. 5. Example of traffic patterns.

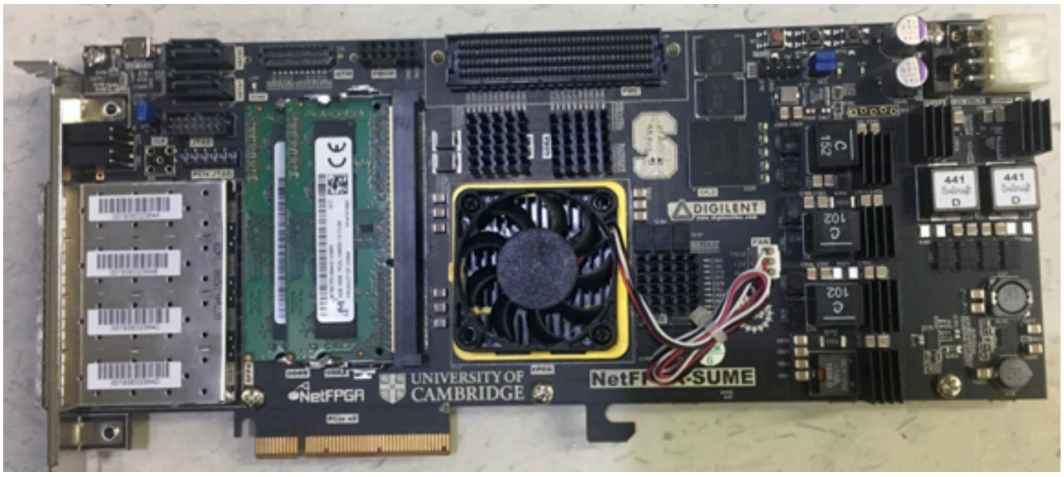


Fig. 6. NetFPGA-SUME.

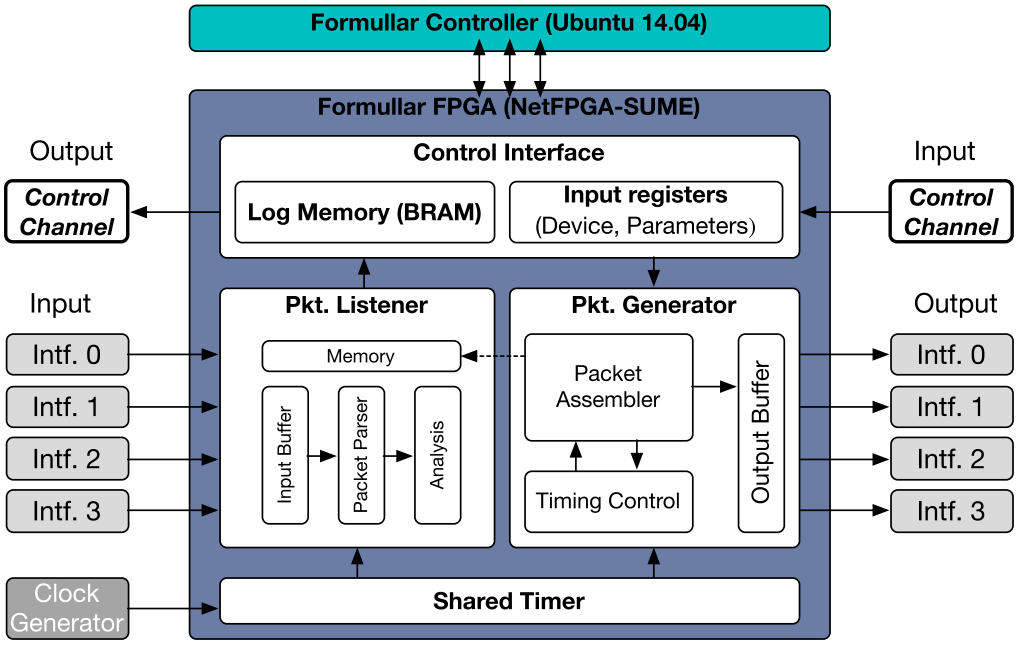


Fig. 7. FPGA implementation details.

μs). Fig. 9 shows our testbed setup. It consists of two desktops equipped with NetFPGA-SUME, Intel Xeon E5-2630, 64 GB RAM, and Intel X520 10 GbE NICs. Furthermore, it runs Ubuntu 14.04 with one machine running Formullar, and the other machine simulating the target device via control of the internal delay. Note that the FPGA is also employed to make the internal delay constant.

We evaluate Formullar with three different setups, as shown in Fig. 10. In the loopback case (Fig. 10(a)), Formullar's sending and receiving interfaces are connected directly to measure the delay made by the measurement tool itself. In the no-delay case (Fig. 10(b)), the

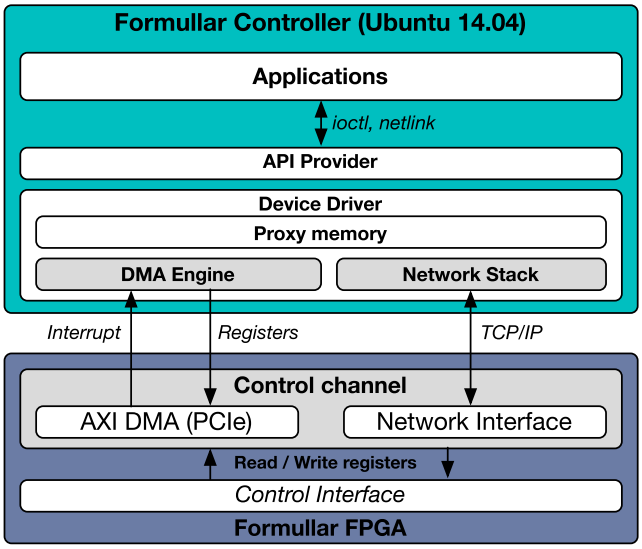


Fig. 8. Controller implementation details.

measurement tool and target device are connected. The target device's delay is set to zero to test how well the measurement tool captures the very low (zero) delay. In the constant delay case (Fig. 10(c)), the target device's delay is varied from 150 ns to 450 ns, to see how accurately the tool captures these delays. To produce each case, the target device is implemented using NetFPGA-SUME, which simulates the delay by deploying internal ring buffers that hold packets for specified time intervals. In each case, the measurement tool generates a series of 100 packets with $len(p)$ and $intv(p)$ at 128 bytes and 0.1 s (10 PPS), respectively. Note that the results show identical trends when we use the other packet lengths and intervals.

In addition, to compare the result with another approach, we perform the same evaluation with a software-based latency measurement tool as a baseline. There are various network testing tools such as NetPerf [35], iperf [36], or hping [20]. Among them, hping is specialized to measure network ping tests inspired by the *ping* UNIX command. It supports various protocols such as TCP, UDP, and RAW-IP protocols over ICMP so that we employ hping to generate packets in this evaluation. When it generates packets, it assembles them on an application layer and sends them through a kernel layer. When it receives packets, packets go through the kernel and arrive at the application layer. Therefore, hping deserves to be a baseline since it utilizes fully software-based packet generation/reception, while Formullar utilizes totally hardware-based packet handling.

5.2. Loopback test

In order to discover unintended delays produced by the measuring device itself, we first conduct the loopback test. Fig. 11 shows the results via candlestick plots. The top/bottom of each rectangle represents the average plus/minus the standard deviation; the middle of the rectangle shows the average; and the top and bottom points of the bar show the maximum and minimum values, respectively.

In the graph, the measured delay of each packet $p_{i,j}$ is calculated as

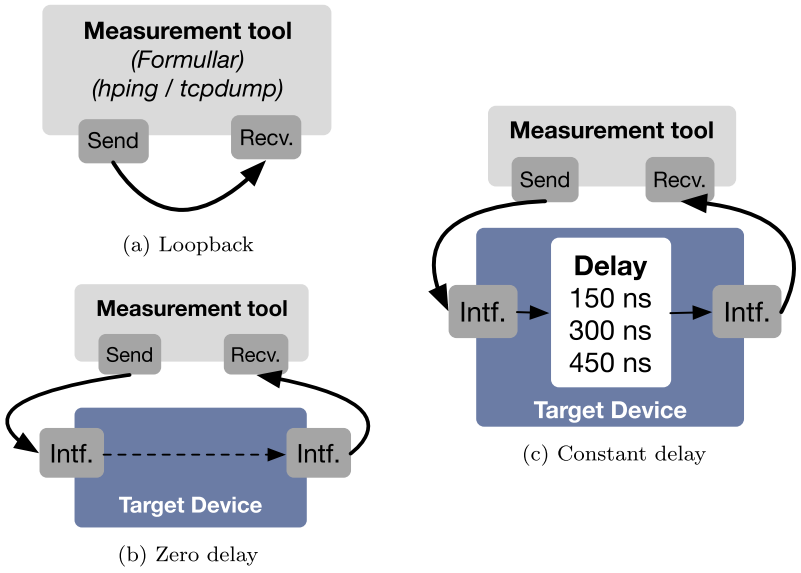
$$\delta_{i,j} = t_{i,j}^{rcv} - t_{i,j}^{gen} \tag{1}$$

where $t_{i,j}^{rcv}$ and $t_{i,j}^{gen}$ represent the reception and generation time of the packet $p_{i,j}$, respectively. The average and standard deviation are simply calculated as $\bar{\delta} = \sum \delta_{i,j} / \sum N_i$ and $\sqrt{\sum (\delta_{i,j} - \bar{\delta})^2 / \sum N_i}$, respectively.

Fig. 11 shows the results of three repeated trials marked as Sets 1–3. In this figure, Formullar has a very low self-delay, which averages about 0.65 µs, and a standard deviation of 0.004 µs. In contrast, the software-based approach results in a much greater self-delay of up to 800 µs and a high standard deviation, which means that the latency measurement would contain a huge measurement error on the order of hundreds of µs, regardless of the target device. One might be tempted to consider this much error minor. However, in ULL-based systems, this size error could account for about 80% of the measured latency when attempting to measure a delay of 1 ms.

In detail, Formullar's self-delay of 0.65 µs is inevitable; it is a kind of the propagation delay, and an internal queuing delay to match the transmission rate and timing of the interface-to-interface communications in the physical layer. Here, the queuing may contain a small deviation up to 0.03 µs to manage the incoming/outgoing packets in each interface. Thus, Formullar's self-delay is about 0.63–0.66 µs, meaning that Formullar's error range is only up to 30 ns. In contrast, the software-based approach exhibits a wide range of errors caused by scheduling and

Fig. 9. Testbed setup.

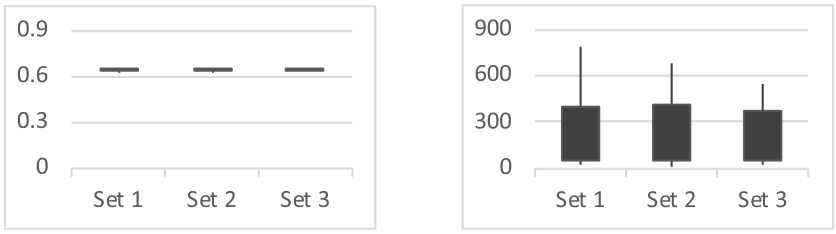


(a) Loopback

(b) Zero delay

(c) Constant delay

Fig. 10. Evaluation setup.



(a) Formullar (μs)

(b) Software-based approach (μs)

Fig. 11. Results for the loopback case.

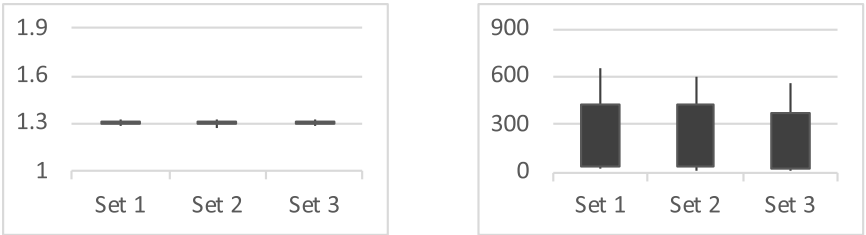
interruptions from the host OS, which cannot be predicted exactly on a ULL scale, resulting in unreliable measurement results.

5.3. The no-delay target

Next, we evaluate the system with a target device with zero delays. As shown in Fig. 12, Formullar provides a consistent result of about 1.3 µs with a low standard deviation. By subtracting the self-delay (i.e., 0.65 µs, as measured by the loopback test) from the measured value, we can derive the exact delay of the target device, which is about 0.65 µs in this evaluation. Considering that the target device is also implemented by NetFPGA-SUME (i.e., the same device used in the prototype of Formullar), it has a self-delay identical to that of Formullar. Thus, the (processing) latency of the target device can be obtained by subtracting the self-delays of Formullar and the target-device (0.65 and 0.65 µs) from the measured value. In this case, the latency is zero ($1.3 - 0.65 - 0.65 = 0.0$), showing that Formullar can measure the latency precisely. In contrast, the result of the software-based approach is similar to its result in the loopback case, i.e., it has a large error range, because this approach always results in non-deterministic errors made by various software layers, including the operating system.

5.4. The target with a delay

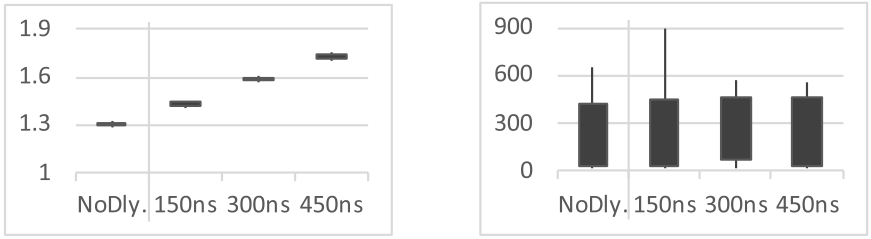
To evaluate how precisely Formullar measures the latency of the target device, we then conduct measurement tests with varying the internal processing delay of the target device. The target device is set to various constant delays (i.e., 150, 300, and 450 ns). Fig. 13 depicts the results of each case. As shown in the figure, the latency is measured as 1.45, 1.60, and 1.75 µs, respectively, on average. Note that `NoDly.` represents the measured delay with the target device having no internal delays. After subtracting the self-delays of two devices (0.65*2), the latency results are 150, 300, and 450 ns, which are identical to the evaluation setup values. This evaluation implies that Formullar can



(a) Formullar (μs)

(b) Software-based approach(μs)

Fig. 12. Results for the no-delay case.



(a) Formullar (μs) (b) Software-based (μs)

Fig. 13. Results for the constant delay cases.

measure the delay of the target device on the ns scale precisely. In contrast, the software-based approach cannot measure the latency precisely. Moreover, the impressive thing is the magnitude of the error (up to 900 μs), even in the 150 ns case. Due to this huge error range, the software-based approach produces nearly the same measurement results for the 300 ns and 450 ns cases. Again, the huge error range mainly comes from the inevitable delay factors of software-based systems, which are packet processing delay on each network layer, interference by schedulers, and interrupt handlers. The result implies that the hardware-based approach is suitable for ULL systems than the software-based one since it can achieve highly precise latency measurements.

5.5. Traffic time precision

To evaluate how precisely the measurement tool generates packets with correct timing (i.e., deterministic packet generation), packets are generated periodically and the deviation in generation/receipt time measured. As shown in Fig. 14, the device is configured to create packets every 0.1 s and capture the error between each measured time interval I_k and the reference interval (i.e., 0.1 s).

For the measurement, the packet generation and reception interval errors $e^{gen}_{i,j}$ and $e^{rcv}_{i,j}$ are respectively calculated as

$$e^{gen}_{i,j} = intv(p) - \left(t^{gen}_{i,j} - t^{gen}_{i,j-1}\right) \tag{2}$$

$$e^{rcv}_{i,j} = intv(p) - \left(t^{rcv}_{i,j} - t^{rcv}_{i,j-1}\right) \tag{3}$$

where $t^{gen}_{i,j}$ and $t^{rcv}_{i,j}$ represent the measured generation and reception time of the packet $p_{i,j}$, respectively. Thereby, an error close to zero implies that the packet was handled with exact timing.

Fig. 15 shows the errors of 100 packets in the loopback case. As shown in the figure, Formullar produces errors close to zero, implying that Formullar can handle the packets with exact timing, deterministic and precise traffic generation, and timer management. In contrast, the software-based approach results in significant errors of about ± 600 μs. Besides, the error range fluctuates widely, creating huge noises in the latency measurement results.

6. Related works

Ultra-low latency. Several networking standards are closely related to ULL technology. Real-time networking systems are the representative cases for the application of ULL. The IEEE 802.1 Time-Sensitive Network (TSN) [37] working group was organized to develop the next-generation standard networking architecture for real-time networks [38]. Deterministic networking systems are also related to ULL technology via bounds on latency and packet delay variations with high reliability. The IETF Deterministic Networking (DetNet) working group [39] is working to define the new standard for deterministic communication. The 5G mobile network, standardized by The 3rd Generation Partnership Project (3GPP) [40], is attempting to unite seven telecommunication standard organizations (ARIB, ATIS, CCSA, ETSI, TSDSI, TTA, and TTC) around ultra-reliable and low latency communication (URLLC).

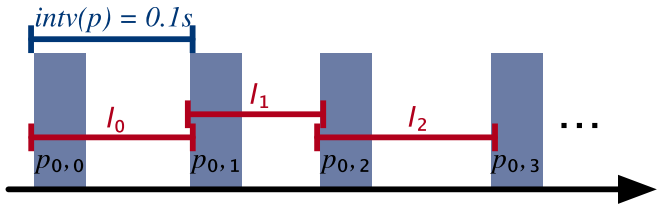
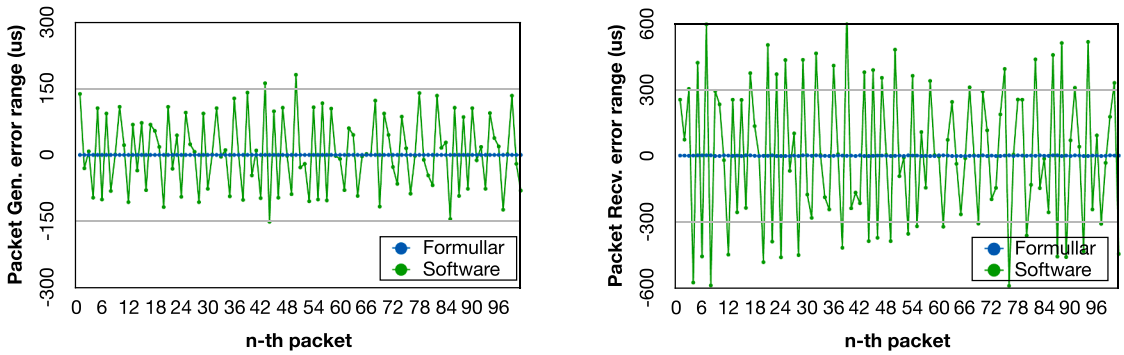


Fig. 14. Definitions for measuring time precision.

Ultra-low latency applications. There are some studies on leveraging ULL networks. Ford et al. [5] proposes a low-latency architecture with flexible data link layers for 5G networks. Pilz et al. [4] suggests a variety of ULL-based services and demonstrates the Tactile Internet. Chen et al. [41] introduces emerging applications, design challenges, and potential approaches in 5G design for URLLC. Lockwood et al. [3] demonstrates an ULL data center network. In terms of practical cases that have been (almost) released to the public, Google Stadia [42] and the Microsoft xCloud [43] Project both require ULL networks. Google's autonomous car [44] and Tesla's autopilot system [45] are notable examples involving cars. Autonomous cars require ULL techniques for not only V2V (vehicle-to-vehicle) and V2I (vehicle-to-infrastructure) communications but also for in-vehicle networking to control the internal powertrain/chassis (e.g., brakes and accelerator). In particular, in-vehicle networks have a direct link to safety; thus, network latency becomes the first priority. Countless other researchers have also conducted ULL studies. For instance, Nasrallah et al. [2] investigates various ULL technologies based on the new standards.

Latency measurement approaches. IXIA [6] is one of the most famous network vendors in the networking testing equipment area. However, their products are priced at tens of thousands of USDs, presenting a significant cost burden. On the other hand, Formullar is available for hundreds to thousands of USDs [13,46,47]. There are some studies that have measured latency in data center networks, in particular, by taking advantage of Software-Defined Networking (SDN). OpenSketch [48] provides a southbound API in a control plane with which to configure measurements. OpenSample [49] uses sFlow, an industry standard for mandatory sampling, and TCP sequence numbers to achieve low latency. Planck [10] achieves ms-scale latency measurements in data center networks by utilizing port mirroring supported by commodity switches. Pingmesh [11] proposes a latency measurement for large-scale data center networks that leverages the TCP or HTTP pings made by all servers. Kown et al. [12] studies network latency profiling for cloud server selection using network measurements for distributed and cloud computing. Narayana et al. [50] suggests that co-designing declarative performance queries along with their associated hardware primitives can bring the benefits of programmability to high-speed network performance measurements. Although there are many studies on latency measurements, no study to date has focused completely on measuring networks at the ns scale, meaning the precision needed for ULL systems has yet to be achieved.

FPGA-based network testing tools. Measuring network at high-



(a) Packet generation time deviation (b) Packet receipt time deviation

Fig. 15. Time deviation for packet generation and receipt.

precision has been in demand and studied for years. In particular, the limitation of precision in software and the high-price of commercial products are well-known issues, thus there are several network measurement studies using FPGA that look similar to our approach. Here, we provide literature reviews for several state-of-the-art studies and what makes Formullar different from them.

A popular approach is to measure network performance from recording and replaying traffic. iTester [51], Siyi Qiao et al. and TNT10G [26] are high-speed network trace players and recorders, and their timestamps between transmission and reception can calculate latency at high-precision. However, they are focused on hardware implementation so that it does not support programming interfaces or automation in measurements. OSNT [27,52] and FlueNT10G [28] are an improved approach. They can perform measurements in a programmable way with pre-recorded traffic (i.e., pcap). However, as they replay the pre-defined traffic, they do not change traffic generation in real-time so that measurements should run only under pre-defined conditions and environments. Also, it is challenging to deal with huge traffic with pcap since the memory in a device is finite. To summarize, there is no fully programmable traffic generation architecture with hardware.

Yong Wang et al. [29] introduced various traffic models that meet a network benchmark standard (i.e., RFC 2544). It proposes an FPGA-based measurement system with a configurable traffic generator. However, it only supports limited configurability (i.e., packet size and inter-packet interval) and throughput (i.e., 1Gbps). FPGEN [30] proposes a precise measurement system equipped with a programmable FPGA-based traffic generator. Although it provides programmability, a fundamental difference from Formullar is that it requires hardware-level programming, while Formullar requires software-level one. To configure FPGEN, a user has to implement a new FPGA logic; it requires background knowledge of circuit-level hardware programming and hardware description language (i.e., HDL), and takes a long development and compilation time at least hours to apply new traffic patterns. In contrast, Formullar allows to easily re-configure the traffic generator with a high-level software script without any recompilation of FPGA circuits.

In summary, instead of replaying pre-recorded traffic, Formullar can control and program the FPGA-based traffic generator with a generalized traffic model; therefore, Formullar can generate very long (even infinite) traffic patterns. In addition, Formullar can dynamically manage network testing for various cases at runtime according to its flexible programmability. And, Formullar supports very high throughput, up to 10 Gbps. To the best of our knowledge, Formullar may be the first study for the fully programmable FPGA traffic generator to measure precise latency at the high bandwidth networking systems.

7. Conclusion

This paper proposed a novel latency measurement tool, named Formullar, to support highly precise latency measurements for ULL network systems while providing reconfigurable and general traffic pattern generation. To this end, we presented a new hybrid architecture composed of FPGA (hardware) and a controller (software). This new architecture can easily (re-)configure the traffic patterns needed to test various ULL networking devices and provides ns-level precision. We presented a full implementation of the Formullar prototype with a commodity FPGA board (NetFPGA-SUME), evaluated the proposed system while comparing it to the software-based approaches, and showed that Formullar provides highly precise ULL measurement results.

Although this implementation provides 6.4 ns precision with its operating frequency limited to 160 MHz, the precision can be improved easily by employing a faster FPGA board. Fortunately, most of the state-of-the-art FPGA boards support frequencies over 100 MHz at a reasonable price (around USD 100–200, including network interfaces) [46,47]. Since Formullar has a general design principle of Formullar, it can adapt to other FPGA boards easily. With a high-speed FPGA, we can easily improve the precision of the measurement tool.

CRediT authorship contribution statement

Taejune Park: Conceptualization, Methodology, Software, Writing - original draft. **Seungwon Shin:** Validation, Project administration, Supervision, Funding acquisition. **Insik Shin:** Validation, Project administration, Supervision. **Kilho Lee:** Conceptualization, Methodology, Writing - original draft.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by the National Research Foundation of Korea (No. NRF-2020R1F1A1076425 and NRF-2020R1A2C2005479) and ERC (NRF-2018R1A5A1059921); and by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2018-0-00254, SDN security technology development).

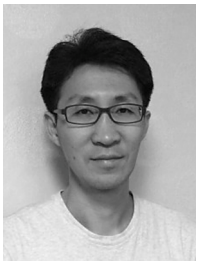
References

[1] K. Qian, F. Ren, D. Shan, W. Cheng, B. Wang, Xpresseth: concise and efficient converged real-time ethernet. 2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS), IEEE, 2017, pp. 1–6.

[2] A. Nasrallah, A.S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, H. ElBakoury, Ultra-low latency (ULL) networks: the IEEE TSN and IETF DetNet standards and related 5G ULL research, IEEE Commun. Surv. Tutor. 21 (1) (2018) 88–145.

[3] J.W. Lockwood, M. Monga, Implementing ultra low latency data center services with programmable logic. 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects, IEEE, 2015, pp. 68–77.
[4] J. Pilz, M. Mehlhose, T. Wirth, D. Wieruch, B. Holfeld, T. Haustein, A tactile internet demonstration: 1ms ultra low delay for wireless communications towards 5G. 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), IEEE, 2016, pp. 862–863.
[5] R. Ford, M. Zhang, M. Mezzavilla, S. Dutta, S. Rangan, M. Zorzi, Achieving ultra-low latency in 5G millimeter wave cellular networks, IEEE Commun. Mag. 55 (3) (2017) 196–203.
[6] ixia, Automotive Ethernet: An Overview. https://support.ixiacom.com/sites/default/files/resources/whitepaper/ixia-automotive-ethernet-primer-whitepaper_1.pdf.
[7] IEEE, IEEE standard for local and metropolitan area networks – bridges and bridged networks - amendment 25: enhancements for scheduled traffic. IEEE Std 802.1Qbv, 2016, pp. 1–57.
[8] iperf, The ultimate speed test tool for TCP, UDP and SCTP. https://iperf.fr/.
[9] netperf https://github.com/HewlettPackard/netperf.
[10] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, R. Fonseca, Planck: millisecond-scale monitoring and control for commodity networks. ACM SIGCOMM Computer Communication Review 44, ACM, 2014, pp. 407–418.
[11] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, et al., Pingmesh: a large-scale system for data center network latency measurement and analysis. ACM SIGCOMM Computer Communication Review 45, ACM, 2015, pp. 139–152.
[12] M. Kwon, Z. Dou, W. Heinzelman, T. Soyata, H. Ba, J. Shi, Use of network latency profiling and redundancy for cloud server selection. 2014 IEEE 7th International Conference on Cloud Computing, IEEE, 2014, pp. 826–832.
[13] NetFPGA, NetFPGA-SUME board. https://netfpga.org/site/#/systems/1netfpga-sume/details/.
[14] N. Zilberman, Y. Audzevich, G.A. Covington, A.W. Moore, NetFPGA SUME: toward 100 Gbps as research commodity, IEEE Micro 34 (5) (2014) 32–41.
[15] MoMeTools '03: Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research, Association for Computing Machinery, New York, NY, USA, 2003.
[16] S. Floyd, E. Kohler, Internet research needs better models, ACM SIGCOMM Comput. Commun. Rev. 33 (1) (2003) 29–34.
[17] J. Sommers, P. Barford, Self-configuring network traffic generation. Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, 2004, pp. 68–81.
[18] C. Williamson, Internet traffic measurement, IEEE Internet Comput. 5 (6) (2001) 70–74.
[19] A. Botta, A. Dainotti, A. Pescapé, Do you trust your software-based traffic generator? IEEE Commun. Mag. 48 (9) (2010) 158–165.
[20] hping, A command-line oriented TCP/IP packet assembler/analyzer. http://www.hping.org/.
[21] Y. Dun-fan, Z. Fei-fan, M. Liang-liang, Design and implementation of high-precision timer in Linux. 2009 WRI World Congress on Computer Science and Information Engineering 7, IEEE, 2009, pp. 341–345.
[22] I. Fedotova, E. Siemens, H. Hu, A high-precision time handling library, J. Commun. Comput. 10 (2013) 1076–1086.
[23] F. Schneider, J. Wallerich, A. Feldmann, Packet capture in 10-gigabit ethernet environments using contemporary commodity hardware. International Conference on Passive and Active Network Measurement, Springer, 2007, pp. 207–217.
[24] Spirent Communications, Spirent test modules and chassis. https://www.spirent.com/Products/TestCenter/Platforms/Modules.
[25] Endace, Endace Measurement Systems. http://www.endace.com.
[26] J.F. Zazo, M. Forconesi, S. Lopez-Buedo, G. Sutter, J. Aracil, Tnt10g: a high-accuracy 10 Gbe traffic player and recorder for multi-terabyte traces. 2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14), IEEE, 2014, pp. 1–6.
[27] G. Antichi, M. Shahbaz, Y. Geng, N. Zilberman, A. Covington, M. Bruyere, N. McKeown, N. Feamster, B. Felderman, M. Blott, et al., Osnt: open source network tester, IEEE Netw. 28 (5) (2014) 6–12.
[28] A. Oeldemann, T. Wild, A. Herkersdorf, Fluent10g: a programmable FPGA-based network tester for multi-10-gigabit ethernet. 2018 28th International Conference on Field Programmable Logic and Applications (FPL), IEEE, 2018, pp. 178–1787.
[29] Y. Wang, Y. Liu, X. Tao, Q. He, An FPGA-based high-speed network performance measurement for RFC 2544, EURASIP J. Wirel. Commun. Netw. 2015 (1) (2015) 2.
[30] M. Sanlı, E.G. Schmidt, H.C. Güran, Fpgen: a fast, scalable and programmable traffic generator for the performance evaluation of high-speed computer networks, Perform. Eval. 68 (12) (2011) 1276–1290.
[31] K. Lee, T. Park, M. Kim, H.S. Chwa, J. Lee, S. Shin, I. Shin, MC-SDN: supporting mixed-criticality scheduling on switched-ethernet using software-defined networking. 2018 IEEE Real-Time Systems Symposium (RTSS), IEEE, 2018, pp. 288–299.
[32] V.S. Frost, B. Melamed, Traffic modeling for telecommunications networks, IEEE Commun. Mag. 32 (3) (1994) 70–81, https://doi.org/10.1109/35.267444.
[33] ARM, AMBA®AXI and ACE Protocol Specification. https://static.docs.arm.com/ihi0022/g/IHI0022G_amba_axi_protocol_spec.pdf.
[34] NetFPGA-SUME, NetFPGA Reference NIC. https://github.com/NetFPGA/NetFPGA-SUME-public/wiki/NetFPGA-SUME-Reference-NIC.
[35] R. Jones, Netperf benchmark, http://www.netperf.org/(2012).
[36] C.-H. Hsu, U. Kremer, IPERF: a framework for automatic construction of performance prediction models. Workshop on Profile and Feedback-Directed Compilation (PFDC), Paris, France, Citeseer, 1998.
[37] IEEE 802.1, Time-Sensitive Networking (TSN) Task Group. https://1.ieee802.org/tsn/.
[38] IEEE standard for local and metropolitan area network–bridges and bridged networks. IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014), 2018, pp. 1–1993, https://doi.org/10.1109/IEEESTD.2018.8403927.
[39] IETF, DetNet Working Group. https://datatracker.ietf.org/wg/detnet/about/.
[40] 3GPP, The 3rd Generation Partnership Project. https://www.3gpp.org/.
[41] H. Chen, R. Abbas, P. Cheng, M. Shirvanimoghaddam, W. Hardjawana, W. Bao, Y. Li, B. Vucetic, Ultra-reliable low latency cellular networks: use cases, challenges and approaches, IEEE Commun. Mag. 56 (12) (2018) 119–125.
[42] Google Stadia. https://stadia.dev/.
[43] Microsoft Project xCloud. https://www.xbox.com/xbox-game-streaming/project-xcloud.
[44] A.S. Brown, Google's autonomous car applies lessons learned from driverless races, Mech. Eng. 133 (2) (2011) 31.
[45] F. Lambert, Tesla's software timeline for 'enhanced autopilot' transition means 'full self-driving capability' as early as next year, Electrek 20 (October 2016).
[46] Digilent FPGA, Arty A7: Artix-7 FPGA Development Board for Makers and Hobbyists. https://store.digilentinc.com/arty-a7-artix-7-fpga-development-board-for-makers-and-hobbyists/.
[47] Digilent FPGA, Zybo Z7: Zynq-7000 ARM/FPGA SoC Development Board. https://store.digilentinc.com/zybo-z7-zynq-7000-arm-fpga-soc-development-board/.
[48] M. Yu, L. Jose, R. Miao, Software defined traffic measurement with opensketch. Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), 2013, pp. 29–42.
[49] J. Suh, T.T. Kwon, C. Dixon, W. Felter, J. Carter, Opensample: a low-latency, sampling-based measurement platform for commodity SDN. 2014 IEEE 34th International Conference on Distributed Computing Systems, IEEE, 2014, pp. 228–237.
[50] S. Narayana, A. Sivaraman, V. Nathan, M. Alizadeh, D. Walker, J. Rexford, V. Jeyakumar, C. Kim, Hardware-software co-design for network performance measurement. Proceedings of the 15th ACM Workshop on Hot Topics in Networks, ACM, 2016, pp. 190–196.
[51] F. Zhang, Y. Xie, J. Liu, L. Luo, Q. Ning, X. Wu, Itester: a FPGA based high performance traffic replay tool. 22nd International Conference on Field Programmable Logic and Applications (FPL), IEEE, 2012, pp. 699–702.
[52] S. Qiao, C. Xu, L. Xie, J. Yang, C. Hu, X. Guan, J. Zou, Network recorder and player: FPGA-based network traffic capture and replay. 2014 International Conference on Field-Programmable Technology (FPT), IEEE, 2014, pp. 342–345.

Taejune Park is currently pursuing his Ph.D. degree in School of Computing at KAIST, Republic of Korea, from September 2015. He received his B.S. degree in Computer Engineering at Korea Maritime and Ocean University, Republic of Korea, in August 2013, and his M.S. degree in Information Security at KAIST, Republic of Korea, in August 2015. His research interests focus on the security issues on SDN/NFV environments and data-planes.



Seungwon Shin is an associate professor in the School of Electrical Engineering at KAIST. He received his Ph.D. degree in Computer Engineering from the Electrical and Computer Engineering Department, Texas A&M University, and his M.S degree and B.S degree from KAIST, both in Electrical and Computer Engineering. He is currently a Research Associate of Open Networking Foundation (ONF), and a member of security working group at ONF. His research interests span the areas of Software Defined Networking (SDN) security, IoT security, and Botnet analysis/detection.

Insik Shin is a professor in the School of Computing and a Chief Professor of Graduate School of Information Security at KAIST, Korea. He received a Ph.D. degree from the University of Pennsylvania, USA, an MS degree from Stanford University, USA, and a BS degree from Korea University, Korea, all in Computer (& Information) Science. Prior to joining KAIST in 2008, he has been a post-doctoral research fellow at Malardalen University, Sweden, and a visiting scholar at University of Illinois at Urbana-Champaign, USA. His research interests include real-time embedded systems, systems security, mobile computing, and cyber-physical systems.

Kilho Lee is currently an assistant professor at Dept. of Smart Systems Software, Soongsil University. Before joining Soongsil in Mar. 2020, He worked as a Post-doc researcher at School of Computing, KAIST, from Mar. 2019 to Feb. 2020. He received my Ph.D. degree in Computer Science from KAIST in 2019. His research interests include system design/implementation for cyber-physical systems and real-time embedded systems.