electronics

MDPI

Article

FPGA-Based High-Performance Network Impairment Emulator

Dexuan Duan [1,2], Xinshuo Wang [1,2], Lin Li [1,2] and Lei Liu [1,2,*]

[1] National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, No. 21, North Fourth Ring Road, Haidian District, Beijing 100190, China; duandx@dsp.ac.cn (D.D.); wangxs@dsp.ac.cn (X.W.); lil@dsp.ac.cn (L.L.)

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

* Correspondence: liul@dsp.ac.cn

Abstract: With the rapid growth of Wide Area Networks (WANs) and advancements in 5G, cloud computing, and IoT, networks face higher demands for low cost, high capacity, reliability, and security. To ensure these requirements, network impairment emulators have become essential tools for testing and optimizing network performance under various conditions. This paper presents an FPGA-based Network Impairment Emulator (FNIE) that accurately emulates packet loss, reordering, and delay with high throughput and low cost. FNIE can achieve up to 100 Gbps throughput, with a configurable packet loss rate ranging from 0.001% to 100%, with a resolution as fine as 0.001%. It also supports up to 64 descriptor queues for reordering, a maximum reordering extent of 511, and emulates delays from 1 µs to 1 s.

Keywords: FPGA; impairment; network; WAN emulation

check for updates

Citation: Duan, D.; Wang, X.; Li, L.; Liu, L. FPGA-Based High-Performance Network Impairment Emulator. *Electronics* **2024**, *13*, 4998. https://doi.org/10.3390/electronics13244998

Academic Editors: Alexander Barkalov, Ke Xie, Shoue Chen and Xiaolong Wang

Received: 24 October 2024
Revised: 2 December 2024
Accepted: 18 December 2024
Published: 19 December 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Researchers and developers often encounter significant challenges when testing network applications and protocols in Wide Area Networks (WANs). Due to the widespread distribution of real WAN environments, testers need to deploy equipment across multiple geographical regions, which is not only costly but also difficult to implement. Furthermore, large-scale experiments are often impractical on production networks, making it challenging to establish realistic test platforms. As a result, WAN emulation technology has emerged as a viable alternative. Researchers can test network applications in Local Area Networks (LANs) by emulating typical WAN characteristics (such as delay, packet reordering, jitter, packet loss, and bandwidth limitation) [1], which significantly reduce testing costs. The location where the WAN emulator is connected to the LAN is shown in Figure 1.

Existing WAN emulators include software-based and hardware-based implementations. Software-based WAN emulators, such as Dummynet, NISTNet, and NetEm, are widely used in research due to their low cost and ease of deployment. These tools are capable of emulating various network conditions, including latency, packet loss, and bandwidth limitations. However, their accuracy is constrained by their reliance on the system's CPU and memory resources, which are typically not designed for precise timing control. Studies have highlighted significant limitations in high-latency and high-bandwidth scenarios. For instance, Velásquez and Gamess [1] observed stability and accuracy issues, while Nussbaum [2] identified discrepancies in timing precision caused by system clock constraints. Furthermore, Angrisani et al. [3] emphasized that software emulators often struggle to replicate low-latency or highly dynamic network conditions accurately. Kathará [4] leverages container-based virtualization for scalability and ease of use but is less suitable for scenarios requiring precise timing control. Similarly, Fogify [5] supports complex topology modeling and real-time configuration but faces challenges in high-throughput scenarios

due to container-based overhead and hardware limitations. These performance bottlenecks become particularly apparent in high-throughput scenarios, where the emulation may fail to maintain line-rate processing or introduce deviations from intended network conditions.

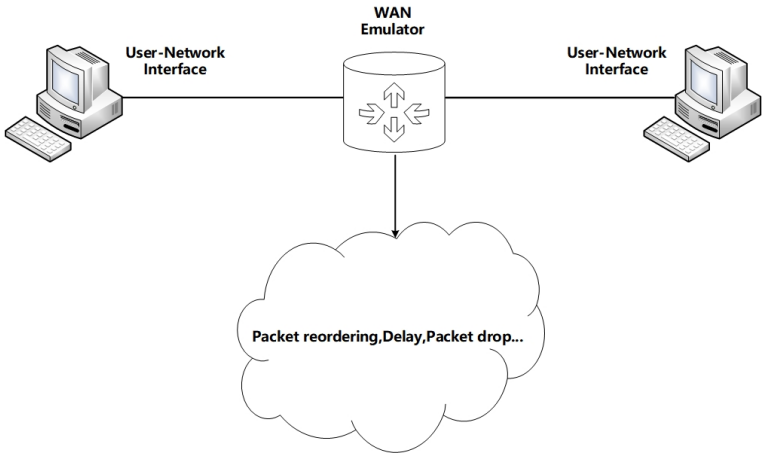


Figure 1. The location of the WAN emulator within the LAN.

In contrast, hardware-based WAN emulators, such as Spirent SNE-X, iTrinegy Network Emulators, and Keysight solutions, provide superior precision and performance by leveraging dedicated hardware components like ASICs or FPGAs. These emulators excel in maintaining accuracy for latency, packet loss, and bandwidth emulation even under high-throughput conditions. However, their high acquisition and maintenance costs, coupled with complex deployment requirements, limit their accessibility. Colosseum is a large scale wireless network emulator consisting 256 software-defined radios (SDR), 24 racks of powerful servers, and a Massive Channel Emulator [6,7]. However, it should be noted that its deployment environment is quite complex. Small to medium-sized laboratories, especially those with constrained budgets, often find it challenging to adopt these solutions. Additionally, while hardware emulators are highly effective, their flexibility is limited when adapting to rapidly changing test scenarios or environments that require frequent reconfiguration. Such limitations restrict their practical applicability in many real-world scenarios, despite their technical advantages.

In order to solve the problems above, this paper proposes an FPGA-based Network Impairment Emulator (FNIE). FNIE is specifically designed for network impairment testing in WAN environments, accurately emulating common impairments such as packet loss, reordering, and delay at a line rate of 100 Gbps. It can be integrated into any project. The main contributions of this work are as follows:

- This study proposes a modular network impairment emulation framework. Based on the AXIS bus, FNIE can be integrated into existing user projects, supporting a wider range of applications and expanding the scenarios of hardware emulators.
- Designed on the Xilinx Ultrascale+ FPGA platform, FNIE can achieve up to 100 Gbps line-rate throughput. This high performance enables the emulator to handle large-scale packet flows while maintaining precise control over network impairments, making it suitable for high-throughput and real-time testing scenarios.
- This study uses specialized configure systems that allow users to adjust network impairment parameters in real time without requiring FPGA development. This flexibility supports a wide range of testing scenarios and improves testing efficiency by dynamically adapting to different application requirements.

2. Related Work

Carson et al. [8] introduce NIST Net, a Linux-based network emulation tool developed by the National Institute of Standards and Technology (NIST). NIST Net enables testing of network protocols by emulating various network conditions, such as delay, jitter, packet

loss, and bandwidth limitations on live IP traffic passing through a Linux-based router. The tool offers flexibility by allowing real-time adjustment of emulation parameters via a command-line interface. While NIST Net is capable of emulating high network data rates up to 1 Gbps, it relies on an older Linux kernel version, which limits compatibility with modern systems. Additionally, the tool's performance degrades in high-bandwidth, low-latency scenarios due to increased processing overhead.

Dummynet [9] is a network emulation tool originally developed by Rizzo and later enhanced by Carbone and Rizzo. The tool allows users to configure various network parameters, such as bandwidth, delay, and packet loss, to emulate different network conditions. Dummynet supports multiple queue management mechanisms, such as FIFO, RED, and WF2Q+, and is capable of handling complex network topologies. Although widely used in network emulation, Dummynet has limitations in terms of timing accuracy, particularly in high data rate scenarios.

Hemminger et al. [10] introduce NetEm, a network emulation tool for Linux that can emulate delays, packet loss, and other network conditions to replicate Wide Area Network (WAN) characteristics in a lab environment. NetEm is built upon the Quality of Service (QoS) and Differentiated Services (DiffServ) facilities within the Linux kernel and is configured through simple command-line utilities. Its main features include the emulation of constant or random delays, packet loss, duplication, and reordering. Although Linux can be configured to operate as a real-time operating system (RTOS), achieving a truly real-time solution requires additional configurations and considerations, making it challenging to ensure precise timing control in certain scenarios.

Kalita and Nambiar [11] introduce WANem, an open-source WAN emulation tool based on Linux. WANem can emulate WAN characteristics in a LAN environment, such as bandwidth limitations, delay, packet loss, reordering, duplication, corruption, and disconnections. The tool utilizes Linux kernel modules like NetEm, HTB, and Netfilter, and provides an easy-to-use web-based Graphical User Interface (GUI) for configuring emulation parameters. Although WANem is widely applicable for various application testing and performance debugging, the limitations in timing precision during high-bandwidth emulations stem not only from its reliance on Linux kernel timers but also from the combined effects of the software implementation and the underlying hardware, which may lack the deterministic performance required for precise emulation.

Kodama et al. [12] introduce GNET-1, a gigabit Ethernet testbed for network emulation. GNET-1 is a programmable network tool based on FPGA, supporting functions such as WAN emulation, network measurement, traffic shaping, and generation. It aims at facilitating the development of network-related grid software. This tool can emulate network conditions, such as delay, bandwidth, and bit errors. It can accurately measure network statistics (e.g., bandwidth and frame count) with a resolution as low as 1 millisecond. Compared to software-based tools like NIST Net, GNET-1 achieves network emulation through a hardware pipeline and provides more stable performance under high-load conditions. Although GNET-1 supports gigabit Ethernet, its performance may be limited when meeting higher bandwidth requirements, such as 10 G or above. While the paper mentions the development of a 10 G verison, this will require further hardware investment and design.

Song et al. [13] introduce MicroGrid, a scientific tool for computational grid emulation. MicroGrid supports running Globus applications on virtual grid resources by virtualizing the execution environment, allowing researchers to experiment with a variety of virtual configurations. The tool can emulate resources such as computing, memory, and networking, and employs a global virtual time model to ensure emulation accuracy. Compared to simple test platforms, MicroGrid provides dynamic behavior emulation for complex grid systems, with an error margin of 2% to 4%. However, this approach is typically quite complex; it is complex to deploy beyond the laboratory and is often limited due to the intricate setup phase involved.

3. System Design

The FNIE system design is shown in Figure 2. Packets are transmitted to the FNIE system via the AXIS bus, while the AXI Lite bus is used to configure subsystems, allowing dynamic adjustment of various impairment parameters. The system prioritizes network impairments in the order of packet loss, reordering, and delay, ensuring processing accuracy and efficiency under high load conditions. Specifically, packet loss impairment has the top priority, minimizing overall processing latency. After packet loss is addressed, packet reordering impairment arranges the packet sequence. Finally, delay impairment temporarily stores packets in off-chip memory (e.g., DDR) and retrieves them after the configured delay. Since delay processing requires frequent access to off-chip storage, a deep pipeline is employed to allow the system to process one packet per cycle, optimizing bus bandwidth utilization. This priority arrangement is based on a comprehensive consideration of network behavior, ensuring FNIE precisely emulating various network impairment scenarios while minimizing the need for off-chip storage access, thereby enhancing overall system performance.

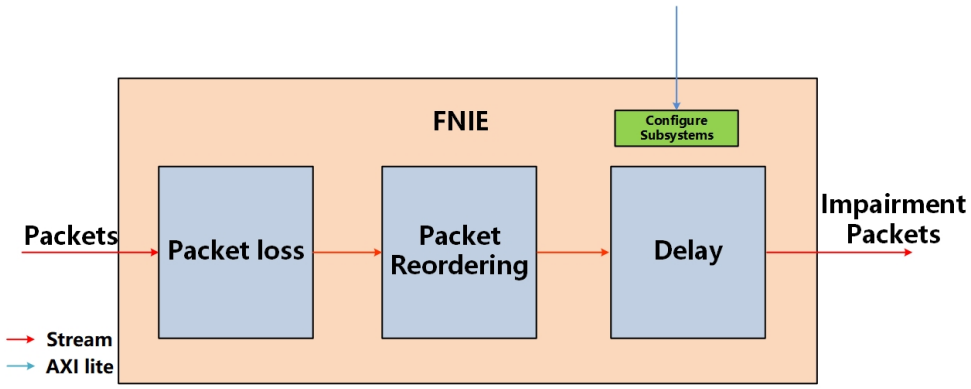


Figure 2. System architecture.

3.1. Pseudo-Random Number Generator (PRNG)

While many types of RNGs are available [14–22], the feedback shift register method remains a popular choice [14]. Although traditional LFSR-based pseudo-random number generators consume minimal hardware resources, their periodicity can impact the accurate emulation of random packet loss due to link congestion on the WAN. We enhance the entropy of the random number generator by adopting a hybrid approach, which increases the unpredictability of the harsh environment of WANs. This improvement provides a more accurate emulation of network randomness and ensures that the emulated network conditions better reflect real-world variations. The pseudocode for the proposed PRNG is presented as follows (Algorithm 1):

Algorithm 1 Enhanced Pseudo-random Numbers Generator

function ENHANCED_HASH_TOEP(*timestamp*, *seed*, *hash_key*)
  $seed \leftarrow LFSR(seed)$ ▷ Update the seed with LFSR
  $hash_input \leftarrow hash(timestamp \parallel seed)$ ▷ Hash function applied to timestamp and seed
  $hash_toep \leftarrow 0$ ▷ Initialize hash_toep
  for $i \leftarrow 0$ to $RANDOM_NUM_WIDTH - 1$ **do**
    if $hash_input[i]$ is true **then**
      $h_toep \leftarrow h_toep \oplus h_key[i \times H_WIDTH : (i+1) \times H_WIDTH - 1]$
    end if
  end for
  return *hash_toep*
end function

3.2. Packet Loss Impairment Design

The design of packet loss impairment is shown in Figure 3. The packet loss impairment module implemented by FNIE is designed to offer highly configurable and flexible random packet loss capabilities, making it suitable for emulating various network impairments during network protocol design and optimization. This module supports the following packet loss modes:

- Single packet loss: Select the discarded packets according to the packet loss location configured by the user to emulate the occasional packet loss in the WAN.
- Random packet loss: According to the packet loss rate set by the user, the PRNG generates a random number each cycle and decides whether to drop the current packet based on a comparison with the set packet loss rate. This emulates irregular packet loss caused by network congestion or interference.
- Burst packet loss: Multiple packets are dropped continuously. The burst length and packet loss rate are adjustable through user-configured parameters to emulate short-term network failures or severe congestion.

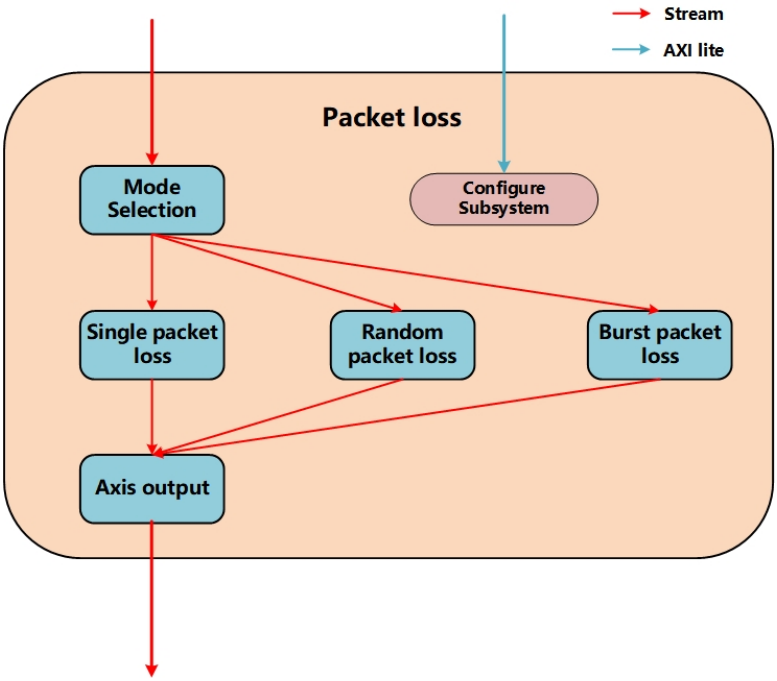


Figure 3. The design of packet loss impairment.

The core feature of this design is the burst window-based packet loss mechanism. Users can control the packet loss behavior by configuring the window size and packet loss rate. In burst packet loss mode, the system drops all packets within a predetermined window to emulate the burst packet loss phenomenon with more realistic conditions, such as network congestion or severe interference. This mechanism uses a hash-based pseudo-random number generator to determine the location and frequency of packet loss in each burst window, effectively avoiding the periodicity problem of traditional generators. The high entropy and unpredictability characteristics of the hash function further enhance the randomness of the packet loss pattern, enabling the scheme to more accurately reproduce the effects of burst impairments in real networks with more accuracy.

3.3. Packet Reordering Impairment Design

According to the packet reordering standard defined in [23], one of the key indicators for measuring the degree of reordering is Type-P-Packet-Reordering-Extent-Stream, which represents the maximum difference between the sequence numbers of two consecutive arriving packets. To improve resource utilization and effectively schedule packet transmis-

sion, we design a packet reordering management scheme based on shared buffers and FIFO queues, as shown in Figure 4. This scheme uses BRAM (Block RAM) as a shared buffer to store and manage packets, and records the specific buffer address of each packet through a packet descriptor. Each packet descriptor is randomly assigned to one of the multiple FIFO queues, and the system polls these queues to determine the transmission order of the packets.

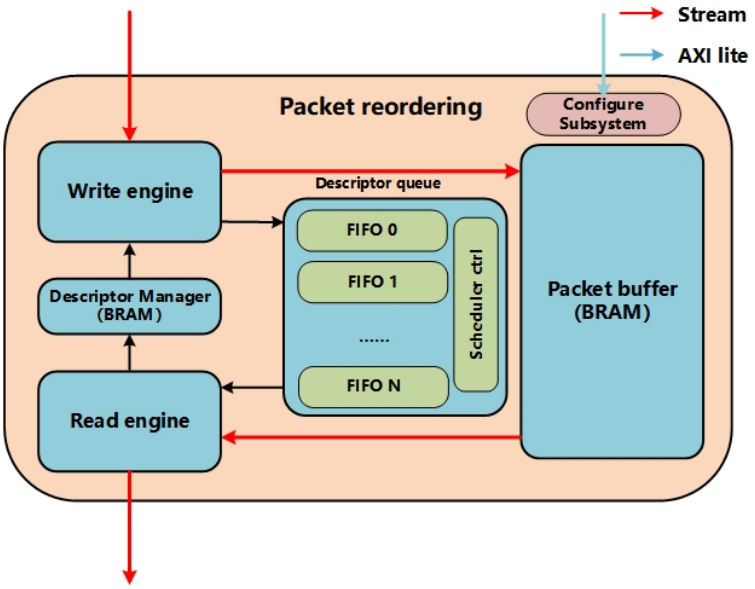


Figure 4. The design of packet reordering impairment.

When a packet arrives, it is first written to the shared BRAM buffer. The descriptor of each packet is randomly allocated to one of the N FIFO queues to ensure that the descriptors are arranged in disorder, achieving the intended out-of-order effect of the packet. The system reads the descriptors from each FIFO queue in sequence and retrieves the corresponding packet from the buffer for transmission according to the address indicated by the descriptor. To ensure the out-of-order output, the system starts reading from the first queue. If the queue is empty, it retries the read in the next cycle and retries up to 16 cycles. If the descriptor cannot be read from the current queue within 16 cycles, the system switches to the next queue to continue reading. The reordering extent can be increased by adjusting the number of FIFO queues. The combination of random descriptor allocation and sequential polling ensures that packets are output in a random and disordered manner, effectively emulating the out-of-order phenomenon of packets that may occur in Wide Area Networks.

As the number of queues increases, the polling period of packets in the queues will also increase, thereby enhancing the reordering extent, at the same time introducing additional latency. For example, when the number of queues is 16, the reordering extent is significantly increased, and the latency remains in an acceptable range; when the number of queues increases to 64, the system latency increases significantly, affecting the overall performance. Thus, it is essential to balance reordering extent and latency to achieve a sufficient out-of-order effect while avoiding excessive latency that could degrade system performance.

3.4. Delay Impairment Design

Delay impairment requires caching a large number of packets and achieving accurate time control. Due to the limited resources on the FPGA chip, these packets often need to be cached externally. Under high-throughput conditions, frequent DDR read and write operations may reduce overall throughput. Backpressure from these operations can even result in packet loss.

To achieve higher throughput, we divide DDR into descriptor and data spaces, utilizing two descriptors to manage normal and jumbo packets. The first descriptor points to continuous off-chip storage space for caching multiple normal packets. While the second descriptor is dedicated to jumbo packets, jumbo packets are typically larger than 1518 bytes. The flowchart for packet caching is shown in Figure 5. Upon receiving a packet, the Cal Pktlen module calculates its length and checks available storage space based on the descriptor pointer. If no storage is available, it will allocate the first descriptor; if the packet is a jumbo packet, it will use the second descriptor and divides the packet into two segments for caching. If storage is available and the packet is normal, it is directly cached according to the first descriptor. Since normal packets are the majority, this approach reduces DDR read and write operations by approximately 40–50%.

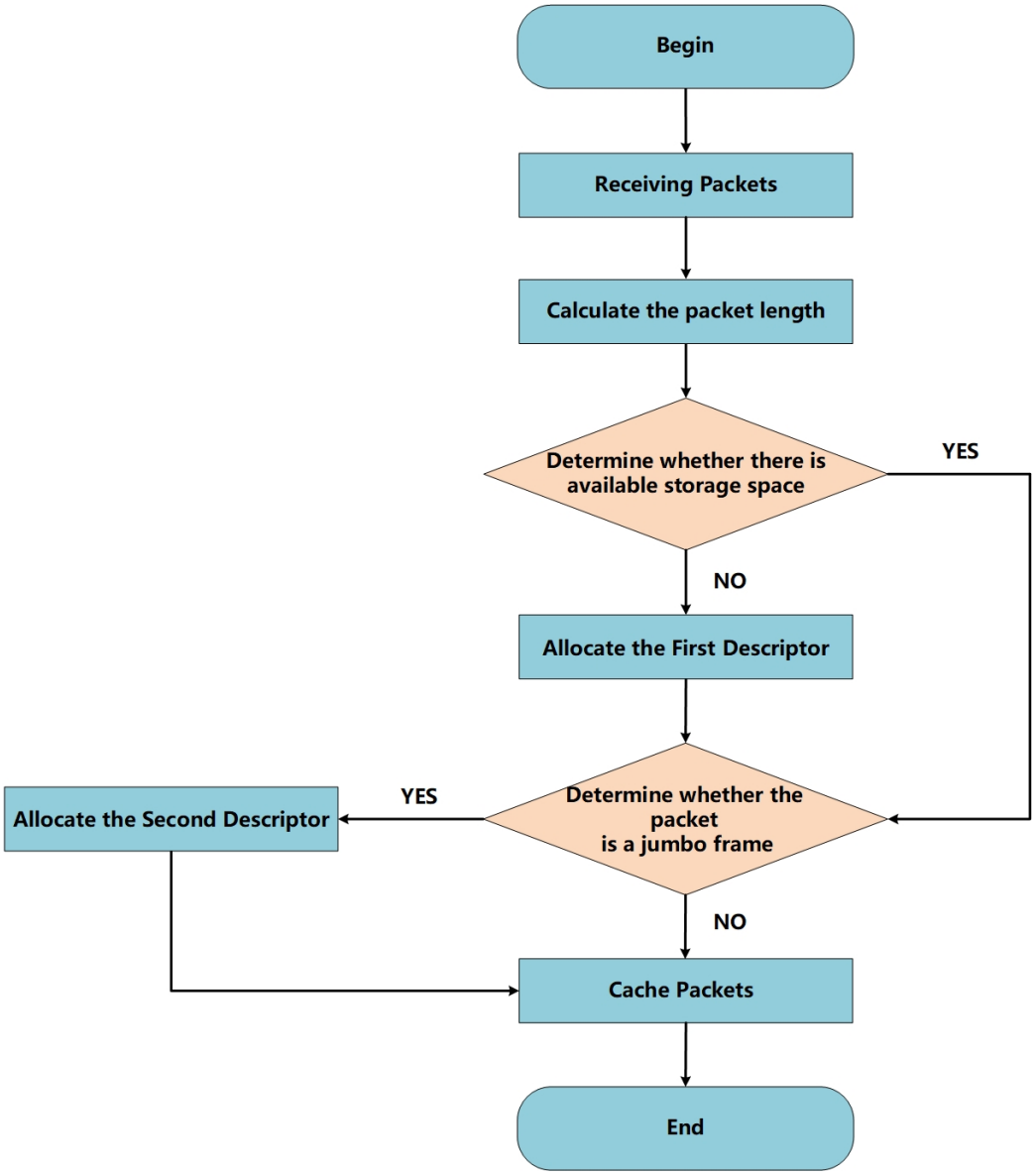


Figure 5. The flowchart of packet caching.

The design of delay impairment is shown in Figure 6. When a packet has identified its designated buffer location, the timer initiates, utilizing a 250 MHz clock as the timing reference to ensure the accuracy of delay emulation. This design provides a time resolution of 4 ns, allowing for accurate control of delay intervals at a granularity of 4 ns, which is sufficient for emulating high-latency scenarios, such as WAN or satellite communications.

Once the configured delay time elapses, the Cache module reads the packet from DDR. Before the Timer initiates, the additional delay induced by timing adjustments remains under five cycles.

This design leverages pre-allocated memory space, reducing the overhead associated with repeatedly allocating and deallocating memory for each packet, thereby reducing DDR access frequency and improving overall system performance. For processing jumbo packets, segmentation enables more efficient utilization of storage resources, preventing a single large packet from occupying excessively large continuous storage blocks, thereby minimizing space wastage. The design also offers flexible configuration capabilities to accommodate various network environments and latency requirements.

By utilizing pre-allocated storage space, the design avoids the overhead associated with repeatedly requesting and releasing storage space. Each time a packet is cached, thereby reducing DDR access frequency and enhancing overall system performance. For handling jumbo packets, segments allows more efficient use of storage resources, preventing a single jumbo packet from occupying excessively large continuous storage blocks, thus minimizing wasted space. This design also offers flexible configuration capabilities to accommodate various network environments and latency requirements.

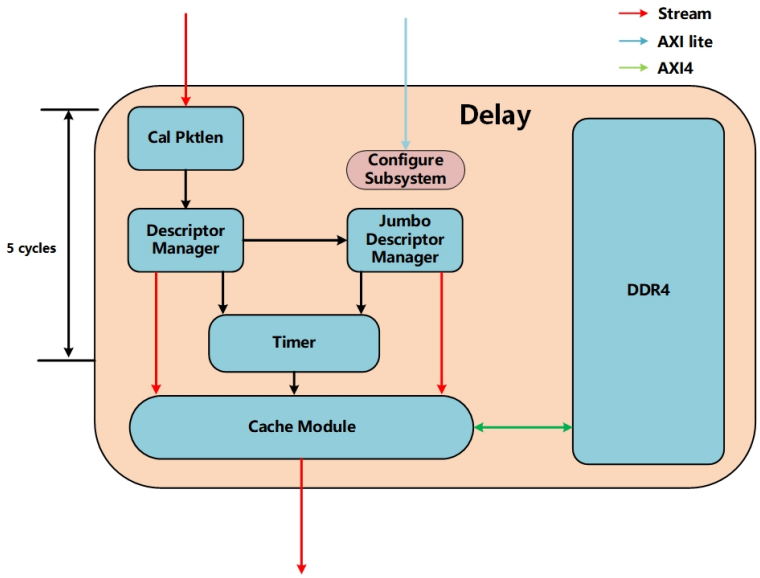


Figure 6. The design of delay impairment.

However, due to its reliance on off-chip storage, system performance may be limited by DDR memory access latency and bandwidth bottlenecks. Under high-load or extreme latency conditions, the speed of off-chip storage may impact the overall latency accuracy of the system. Therefore, future optimization directions include considering the use of on-chip memory or hybrid storage solutions to further improve system response speed and accuracy.

4. Experimental Result

4.1. Implementation

We deploy FNIE on the Alevo U200 platform. To verify the performance of FNIE, we use the IXIA high-speed network traffic generation tool to generate a 100 Gbps high-speed data stream. The data stream passes through FNIE and returns to IXIA. The impairment packets can be observed on the IXIA statistics interface. The experimental environment deployment is shown in Figure 7.

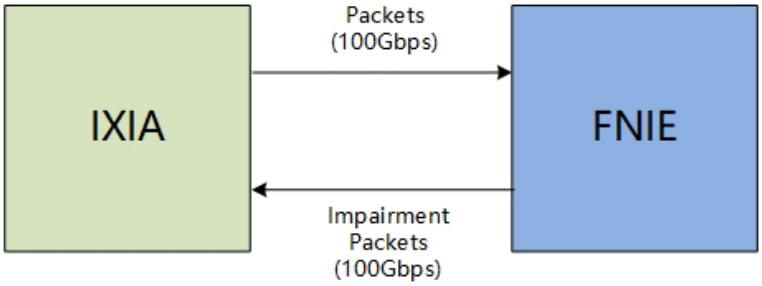


Figure 7. Experimental environment.

4.2. Evaluation

We test the performance of FNIE under different packet sizes, packet loss rates, reordering depths, and delays. After the packets are impaired by FNIE, they are still output at a rate of 100 Gbps. The resource utilization of the three impairment modules supported by FNIE is shown in Table 1.

Table 1. FPGA resource utilization.

	LUTs	**Registers**	**BRAMs**
Packet loss	250 (0.02%)	392 (0.02%)	0 (0%)
Packet reordering	4239 (0.26%)	2576 (0.13%)	264.5 (12.5%)
Delay	11,509 (0.71%)	8116 (0.41%)	11 (0.52%)

We compare the resource utilization of a 16-bit LFSR and the PRNG proposed in Section 3.1 on the Xilinx Ultrascale+ FPGA platform. Experimental results show that the PRNG uses only 20 additional LUTs compared to the LFSR, making the extra resource consumption negligible. Furthermore, we analyze the output of the LFSR and PRNG over 65,536 iterations. The statistical results indicate that within the range of [0–65,535], the LFSR failes to cover 313 numbers, whereas the PRNG misses only 34 numbers. This experiment is repeated 1000 times, with no significant conflicts observed, demonstrating that the proposed PRNG exhibits higher entropy than the typical LFSR. Subsequent packet loss experiments further validated this conclusion.

According to the experimental results, FNIE can stably achieve a line-rate throughput of 100 Gbps, with a configurable packet loss rate ranging from 0.001% to 100%. Table 2 shows the experimental results of packet loss impairment. From the results, it can be seen that whether small packets (128 B) or large packets (8192 B), FNIE can achieve high accuracy within a packet loss rate above 0.001%. This is because the packet loss module in FNIE uses a pseudo-random number generation mechanism, and a random number is generated every cycle. This ensures accurate control of packet loss (regardless of the packet size) while avoiding any impact on the overall system throughput. This design not only guarantees randomness, but also avoids resource bottlenecks when processing large-scale data streams. Therefore, even under extreme packet loss conditions, FNIE can achieve near-line-rate performance.

In the packet reordering impairment experiment, increasing the number (k) of FIFO queues (N) directly affects the Reordering Extent (RE) by increasing the probability of packets being randomly distributed across different queues. This results in greater variance in packet sequence numbers during retrieval, thereby emulating real-world reordering scenarios. The relationship between Reordering Extent RE, the number of FIFO queues, and polling time T is illustrated below:

$$RE = k * N * T \tag{1}$$

Table 2. Experimental results of packet loss.

Packet Size	Packet Loss Rate	Actual Packet Loss Rate
128 B	0.001%	0.0012%
1024 B	0.001%	0.0013%
1500 B	0.001%	0.0011%
8192 B	0.001%	0.0011%
128 B	0.1%	0.1008%
1024 B	0.1%	0.1009%
1500 B	0.1%	0.1001%
8192 B	0.1%	0.1002%
128 B	1%	0.9998%
1024 B	1%	0.9999%
1500 B	1%	0.9998%
8192 B	1%	0.9999%
128 B	10%	10.0004%
1024 B	10%	10.0003%
1500 B	10%	9.9998%
8192 B	10%	9.9999%
128 B	99%	99.0001%
1024 B	99%	98.9998%
1500 B	99%	99.0004%
8192 B	99%	99.0001%

An increase in N will result in higher latency due to the longer polling period. Therefore, optimizing the balance between FIFO queue count and polling interval can achieve effective out-of-order emulation while achieving acceptable system performance. Table 3 shows the impact of the number of FIFO queues on the system's average latency for a packet size of 1024 B.

In our experiment, we used three packet sizes to evaluate system deviation under various delay settings. Figure 8 shows that deviation is minimal for short delays, as packets are not impacted yet by DDR access bottlenecks; the primary deviations are clock precision and buffering delay.

Table 3. Experimental results of packet reordering.

N	Latency (ns)	MAX RE
4	156	31
8	282	63
16	538	127
32	1044	255
64	2064	511

As the delay increases, frequent DDR access results in bandwidth limitations and access delays, especially for small packets, resulting in more significant deviations. Overall, system deviation trends upward with longer delays but remains manageable in most cases. To reduce deviation and enhance performance, a two-tier caching architecture with BRAM and DDR can decrease DDR access frequency. Additionally, optimizing storage allocation and buffering mechanisms can improve responsiveness and delay accuracy for short-delay applications.

To further evaluate the performance of the system under different impairment conditions, we configure various combinations of packet loss, packet reordering, and delay. The packet size used in all experiments is 1024 bytes. Importantly, the system maintains a throughput of 100 Gbps across all test scenarios, demonstrating its ability to handle high-speed traffic even under complex impairment patterns. The additional system latency

introduced by each impairment configuration is summarized in Figure 9. Due to space limitations, in Figure 9, PL represents the packet loss rate, PR represents the maximum reordering extent, and DE represents the delay time.

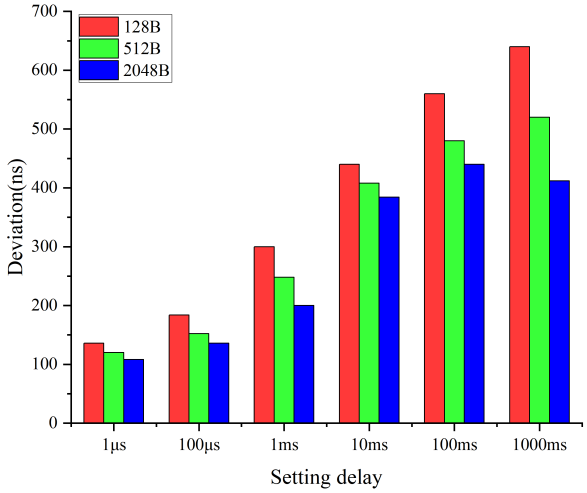


Figure 8. Experimental results of delay.

When impairments are applied individually, such as 10% packet loss with a 10 ms delay or 63 maximum reordering extent, the additional system latency remained below 700 ns. This demonstrates the system's efficiency in isolating and applying single impairments without significant performance degradation. In scenarios involving combined impairments, such as 10% packet loss, the maximum reordering extent is 63, and for a 10 ms delay, the additional system latency increased slightly, reaching a maximum of 720 ns. Even for more complex configurations, such as a 127 maximum reordering extent, 100 ms delay, and 10% packet loss, the system maintained an additional latency of 1172 ns, which remains within acceptable limits for high-performance network testing. Experiments involving higher delays (e.g., 100 ms) contributed the most to system latency. However, the results confirm that the delay impairment function effectively handles high-delay scenarios while maintaining high throughput.

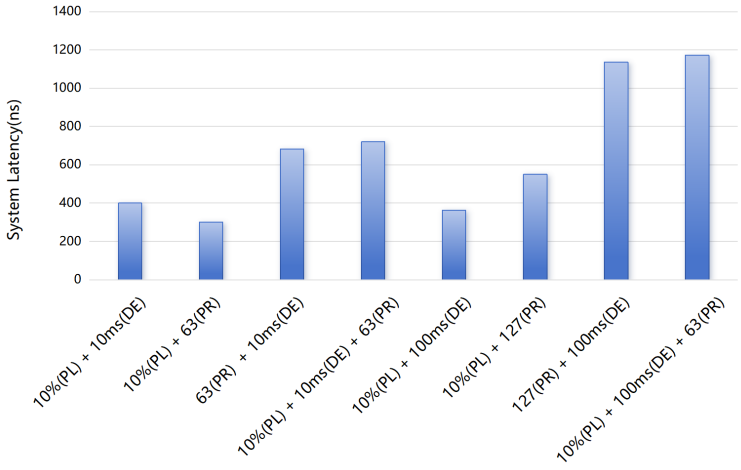


Figure 9. Experimental results of multiple impairments.

Overall, these results validate the proposed system's ability to simulate diverse and complex WAN impairments with minimal impact on throughput and acceptable additional latency, making it suitable for high-performance network testing scenarios.

4.3. Analysis

To better demonstrate the performance of different types of WAN emulators in various test scenarios, we compared the throughput performance of different WAN emulators and FNIE, as shown in Table 4. Traditional WAN emulators are cost-effective and easy to deploy, and suitable for low to medium bandwidth testing. However, they often struggle to meet the needs of high-precision and high-throughput environments.

FNIE combines the high performance of hardware solutions with relatively low costs. By leveraging FPGA hardware acceleration, FNIE not only provides low-latency, high-precision network impairment emulation at 100 Gbps, but also provides a degree of flexibility that makes it suitable for a wide range of network test scenarios. Compared with traditional hardware solutions, FNIE provides a more cost-effective option, especially in cases that require dynamic configuration and real-time testing.

Table 4. Performance comparison.

	Our Works	**NIST Net**	**Dummynet**	**WANem**	**GNET-1**
Throughput	100 Gbps	1 Gbps	1 Gbps	unknown	Below 10 Gbps

4.4. Limitations

Although the FPGA-based Network Impairment Emulator (FNIE) proposed in this study demonstrates superior accuracy and performance compared to existing solutions, there are several limitations that need to be addressed in future work:

Precision of Packet Loss Simulation: While FNIE achieves a packet loss precision of 0.001%, minor deviations may still occur under certain network conditions, potentially affecting high-precision network simulations. Further refinements are needed to ensure consistent accuracy, especially in scenarios requiring extremely low packet loss rates.

Configuration Constraints for Packet Reordering: Although FNIE supports adjustable packet reordering via the number of FIFO queues, it currently lacks the ability to precisely configure the degree of reordering. For instance, scenarios requiring specific reordering probability distributions cannot be accurately simulated with the existing implementation.

Throughput Limitations in Delay Impairment: In scenarios involving small packets, such as 64-byte packets, the frequent DDR read and write operations required for Delay Impairment result in reduced throughput, making it challenging to sustain a line-rate performance of 100 Gbps. This limitation arises primarily from off-chip memory access latency and bandwidth bottlenecks, impacting delay accuracy under high-load conditions.

5. Conclusions

This paper presents an FPGA-based WAN emulator capable of emulating packet loss, reordering, and delay scenarios. By employing configurable systems, the emulator allows dynamic adjustment of network impairment parameters, enabling the emulation of realistic WAN conditions within an LAN environment. Implemented on Xilinx Ultrascale+ FPGA, the architecture supports 100 Gbps throughput, with higher performance and flexibility compared to existing solutions. FNIE serves as a cost-effective tool for testing network applications and protocols. Future work will focus on optimizing resource efficiency and exploring more complex network scenarios. The performance and applicability of FNIE can be further improved by using hybrid storage solutions or higher-performance chips and optimizing the algorithm design.

Author Contributions: Conceptualization, L.L. (Lei Liu); methodology, implementation, and validation, D.D.; writing—original draft preparation: D.D., X.W. and L.L. (Lin Li); writing—review and editing, L.L. (Lei Liu) and D.D.; supervision, L.L. (Lei Liu). All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Oriented Project Independently Deployed by the Institute of Acoustics, Chinese Academy of Sciences: Research and Development of Key Technologies and Equipment for Low-Latency Interconnection Network in Intelligent Computing Center Cluster (Project No. MBDK202401).

Data Availability Statement: All the necessary data are included in the article.

Acknowledgments: The authors would like to thank Lei Song, Yiwei Chang and Zhixiang Zhao for insightful comments. The authors would like to sincerely thank the anonymous reviewers for their feedback on earlier versions of this manuscript.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Velásquez, K.; Gamess, E. A comparative analysis of WAN emulators. In Proceedings of the 7th Latin American Networking Conference, Medellín, Columbia, 4–5 October 2012; Association for Computing Machinery: New York, NY, USA, 2012; pp. 44–51.
2. Nussbaum, L.; Richard, O. A comparative study of network link emulators. *ACM SIGCOMM Comput. Commun. Rev.* **2009**, *44*, 85.
3. Angrisani, L.; Capriglione, D.; Cerro, G.; Ferrigno, L.; Miele, G. Experimental analysis of software network emulators in packet delay emulation. In Proceedings of the 2017 IEEE International Workshop on Measurement and Networking (M&N), Naples, Italy, 27–29 September 2017; pp. 1–6.
4. Scazzariello, M.; Ariemma, L.; Caiazzi, T. Kathará: A Lightweight Network Emulation System. In Proceedings of the NOMS 2020—2020 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 20–24 April 2020; pp. 1–2.
5. Symeonides, M.; Georgiou, Z.; Trihinas, D.; Pallis, G.; Dikaiakos, M.D. Fogify: A Fog Computing Emulation Framework. In Proceedings of the 2020 IEEE/ACM Symposium on Edge Computing (SEC), San Jose, CA, USA, 12–14 November 2020; pp. 42–54.
6. Melodia, T.; Basagni, S.; Chowdhury, K.R.; Gosain, A.; Polese, M.; Johari, P.; Bonati, L. Colosseum, the world's largest wireless network emulator. In Proceedings of the 27th Annual International Conference on Mobile Computing and Networking, New Orleans, LA, USA, 25–29 October 2021; Ser. MobiCom '21; Association for Computing Machinery: New York, NY, USA, 2021; pp. 860–861.
7. Seppänen, K. Real-time Millimeter Wave Wireless Mesh Network Emulation. In Proceedings of the 2024 International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, Croatia, 26–28 September 2024; pp. 1–7.
8. Carson, M.; Santay, D. NIST Net: A Linux-based network emulation tool. *SIGCOMM Comput. Commun. Rev.* **2003**, *33*, 111–126. [CrossRef]
9. Carbone, M.; Rizzo, L. Dummynet revisited. *ACM SIGCOMM Comput. Commun. Rev.* **2010**, *40*, 12–20. [CrossRef]
10. Hemminger, S. Network Emulation with NetEm. Available online: https://www.rationali.st/blog/files/20151126-jittertrap/netem-shemminger.pdf (accessed on 17 December 2024).
11. Kalitay, H.K.; Nambiarz, M.K. Designing WANem: A Wide Area Network emulator tool. In Proceedings of the 2011 Third International Conference on Communication Systems and Networks (COMSNETS 2011), Bangalore, India, 4–8 January 2011; pp. 1–4.
12. Kodama, Y.; Kudoh, T.; Takano, R.; Sato, H.; Tatebe, O.; Sekiguchi, S. GNET-1: Gigabit Ethernet network testbed. In Proceedings of the 2004 IEEE International Conference on Cluster Computing (IEEE Cat. No.04EX935), San Diego, CA, USA, 20–23 September 2004; pp. 185–192.
13. Song, H.J.; Liu, X.; Jakobsen, D.; Bhagwan, R.; Zhang, X.; Taura, K.; Chien, A. The MicroGrid: A Scientific Tool for Modeling Computational Grids. In Proceedings of the SC '00: 2000 ACM/IEEE Conference on Supercomputing, Dallas, TX, USA, 4–10 November 2000; p. 53.
14. MahendraBabu, G.R.; Sridhar, K.P.; Baskar. FPGA based hybrid random number generators. In Proceedings of the 2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 5–7 November 2020; pp. 404–406.
15. Dahmani, S.; Maamoun, M.; Zerari, G.; Chabini, N.; Beguenane, R. An efficient FPGA-based Gaussian random number generator using an accurate segmented Box—Muller method. *IEEE Access* **2023**, *11*, 64745–64757. [CrossRef]
16. Di Patrizio Stanchieri, G.; De Marcellis, A.; Faccio, M.; Palange, E. An FPGA-based architecture of true random number generator for network security applications. In Proceedings of the 2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, 27–30 May 2018; pp. 1–4.
17. Târşa, I.G.; Budariu, G.-D.; Grozea, C. Study on a true random number generator design for FPGA. In Proceedings of the 2010 8th International Conference on Communications, Bucharest, Romania, 10–12 June 2010; pp. 461–464.
18. Tian, X.; Benkrid, K. Mersenne Twister Random Number Generation on FPGA, CPU and GPU. In Proceedings of the 2009 NASA/ESA Conference on Adaptive Hardware and Systems, San Francisco, CA, USA, 29 July–1 August 2009; pp. 460–464.
19. Sileshi, B.G.; Ferrer, C.; Oliver, J. Accelerating hardware Gaussian random number generation using Ziggurat and CORDIC algorithms. In Proceedings of the SENSORS, 2014 IEEE, Valencia, Spain, 2–5 November 2014; pp. 2122–2125.
20. Li, C.; Wang, Q.; Jiang, J.; Guan, N. A metastability-based true random number generator on FPGA. In Proceedings of the 2017 IEEE 12th International Conference on ASIC (ASICON), Guiyang, China, 25–28 October 2017; pp. 738–741.

21. Maamoun, M.; Saadi, H.A.; Dahmani, S.; Zerari, G.; Chabini, N.; Beguenane, R. An Optimized FPGA Based Box-Muller Gaussian Random Number Generator Architecture for Communication Applications. In Proceedings of the 2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 27–30 October 2021; pp. 0772–0777.
22. Wold, K.; Petrovic, S. Optimizing Speed of a True Random Number Generator in FPGA by Spectral Analysis. In Proceedings of the 2009 Fourth International Conference on Computer Sciences and Convergence Information Technology, Seoul, Republic of Korea, 24–26 November 2009; pp. 1105–1110.
23. Jayasumana, A.; Dovrolis, C.; Morrell, J. Packet Reordering Metric. RFC 4737. 2006. Available online: https://datatracker.ietf.org/doc/rfc4737/ (accessed on 17 December 2024).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.