SDNIE: A Software-Defined Approach to High-Performance Network Impairment Emulation using Programmable Switches

Lizhuang Tan, Nguyen Van Tu, Xinhang Wang, Peiying Zhang, James Won-Ki Hong

***Abstract*—Network testing is critical for evaluating the performance, reliability, and security of modern computer networks. A key challenge is creating an accurate, cost-effective, and high-performance network emulation environment. Network Impairment Emulators (NIEs) emulate real-world network conditions such as bandwidth constraints, latency, and packet loss, but existing CPU- and FPGA-based solutions suffer from limited performance, high costs, and poor flexibility. This paper proposes Software-Defined Network Impairment Emulation (SDNIE), a novel framework that leverages programmable switches for scalable, cost-efficient network impairment emulation. SDNIE introduces three key techniques: (1) intent-driven network impairment configuration, automating impairment modeling; (2) serial-parallel combined execution, optimizing performance; and (3) CPU-Tofino collaborative deployment, offloading complex computations. Experimental results show that SDNIE matches commercial emulators in performance while significantly reducing costs. This work demonstrates the potential of programmable switches in network testing, offering a scalable, cost-effective, and high-performance alternative for next-generation network impairment emulation.**

***Index Terms*—Software-Defined Networking, Programmable Data Plane, Network Testing, Network Impairment Emulation, Network Management**

I. Introduction

Network testing [1] plays a crucial role in promoting the continuous innovation and development of computer network architectures and technologies. For researchers and engineers, building a realistic or high-fidelity network testing environment is a prerequisite for conducting experiments, certification, operation, administration, and maintenance.

Currently, there are three approaches to building a network testing environment: simulation [2], emulation [3], and testbed [4]. As shown in Table I,

1) **Simulation** is used to quickly verify the approximate performance of network protocols under large-scale topologies, offering low cost but poor accuracy. For example, the most widely used tools currently include NS-3 [5], OMNeT++ [6], Mininet [7], and DONS [8].
2) **Emulation** typically relies on hardware to closely imitate real network transmission characteristics, providing high precision but requiring expensive equipment. Linux provides the tc-netem [9] module for emulating network performance. CORE [10] has open-sourced more comprehensive emulation capabilities. Sprient Network Emulator [11] is the most well-known commercial product.
3) **Testbed** creates a real network environment to conduct large-scale testing tasks, making them the most expensive and scarce option. ARPANET, the precursor to the modern internet, was one of the earliest network technology testbeds. New-generation testbeds, represented by GENI [12], CERNET2 [13] and CENI [14], have led the development of future network technologies such as SDN and IPv6.

To a certain extent, emulation can be regarded as a compromise between simulation and testbed [15].

The network impairment emulator (NIE) is a hardware device designed to emulate network characteristics such as throughput, delay, loss, packet reordering, and other transmission impairments. Well-known manufacturers of such NIEs include Keysight [16] and Spirent [11]. Currently, the commercial NIE are generally implemented using CPU or FPGA, claiming that their highest performance can reach the emulation of end-to-end network with 100Gbps bandwidth and 30ms delay.

However, several issues persist in the available network impairment emulators, including:

1) **Limited Performance of CPU-based Solutions**: CPU-based solutions typically exhibit low performance, though they benefit from lower development costs. They can generally support network impairments only up to 10Gbps, which may not suffice for high-throughput network testing.
2) **Challenges with FPGA-based Solutions**: FPGA-based solutions offer high performance but come with in-

This work was supported by the National Key R&D Program of China under Grant No.2024YFB2907000, the Shandong Provincial Natural Science Foundation under Grant No.ZR2024LZH006, the National Natural Science Foundation of China under Grant No.62402257, the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2024-00392332), the Pilot Project for Integrated Innovation of Science, Education and Industry of Qilu University of Technology (Shandong Academy of Sciences) under Grant No.2025ZDZX01. (Corresponding author: James Won-Ki Hong)

Lizhuang Tan and Xinhang Wang are with the Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Shandong Computer Science Center (National Supercomputer Center in Jinan), Qilu University of Technology (Shandong Academy of Sciences), Jinan, China. And they are also with Shandong Provincial Key Laboratory of Computing Power Internet and Service Computing, Shandong Fundamental Research Center for Computer Science, Jinan, China (e-mail: tanlzh@sdas.org; 10431220360@stu.qlu.edu.cn).

Nguyen Van Tu is with MangoBoost Inc., Seoul, Korea (e-mail: nguyen.tu@mangoboost.io).

Peiying Zhang is with the Qingdao Institute of Software, College of Computer Science and Technology, China University of Petroleum (East China), Qingdao 266580, China and with the Shandong Key Laboratory of Intelligent Oil & Gas Industrial Software, Qingdao, China (e-mail: zhangpeiying@upc.edu.cn).

James Won-Ki Hong is with Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang, Korea (e-mail: jwkhong@postech.ac.kr).

TABLE I
COMPARISON OF THREE NETWORK TESTING ENVIRONMENT CONSTRUCTION METHODS

Type	Feature	Example
Simulation	The hierarchical characteristics of the network are abstracted conceptually, and a fully virtual and controllable abstraction environment is constructed through software simulation.	NS-3 [5], OMNeT++ [6], Mininet [7], DONS [8], etc.
Emulation	Treat the network as a black box and emulate its characteristics through the combination of software and hardware.	Linux tc-netem [9], CORE [10], Sprient Network Emulator [11], etc.
Testbed	Build the network test environment with close to real functions and performance by deploying real hardware and software.	ARPANET, GENI [12], CERNET2 [13], CENI [14], etc.

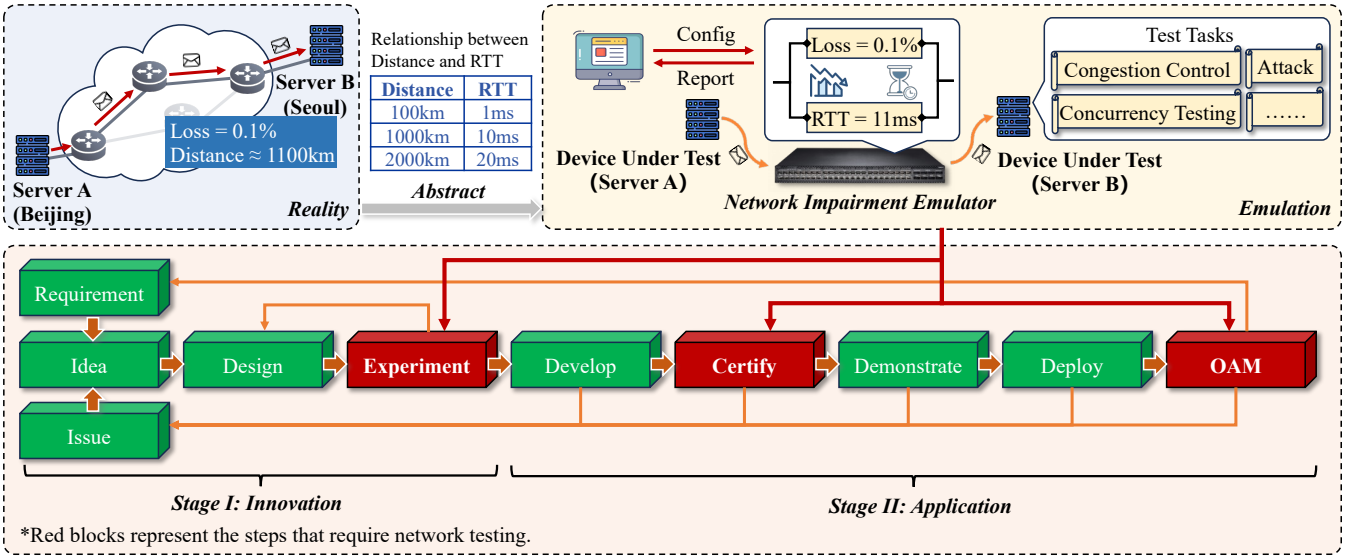


Fig. 1. Network Impairment Emulation and Its Use in the Process of Innovation and Application of Computer Networks.

creased costs and power consumption. They require custom designs to support specific network protocols, leading to greater development and maintenance complexities.

3) **Lack of Interoperability Standards**: There is no established standard for NIEs, and implementations of identical impairment functions can vary significantly. This inconsistency results in poor protocol compatibility and interoperability issues across different manufacturers.
4) **Suboptimal User Interaction**: Web-based user interfaces often lead to complex configuration processes, while RESTful APIs typically support only basic remote control functionalities. Consequently, user interaction mechanisms of NIEs require optimization to enhance usability and efficiency.

To address the future demands for network impairment emulation with more complex functions and higher performance, this paper proposes the **S**oftware-**D**efined **N**etwork **I**mpairment **E**mulation (SDNIE), which leverages software-defined networks and programmable data plane, offering a technical approach distinct from CPU/FPGA-based solutions. SDNIE repurposes readily available programmable switches to implement network impairment functionalities and optimizes their execution through three key technologies: (1) intent-driven network impairment configuration, (2) serial-parallel combined impairment execution, and (3) CPU-Tofino[1] collaborative impairment function deployment. We implement the SDNIE prototype based on common P4 switch [17], and the experimental results show that the performance of the SDNIE prototype is on par with that of the most advanced commercial network impairment emulators.

II. BACKGROUND AND RELATED WORK

Network testing runs through the entire process of the innovation and development of computer networks, as shown in Figure 1. The NIE is a niche device that is employed in scenarios such as mobile communications [18], wide area networks [19], and data center networks [20] to build a near-realistic testing environment. It treats the network as a black box and emulates its characteristics through either software-based methods (e.g., Linux tc-netem [21]), hardware solutions (e.g., Network Impairment Emulator), or a combination of both. If researchers and engineers want to test the performance of a 1,100-kilometer end-to-end communication application from Beijing, China to Seoul, Korea, they do not need to actually deploy servers in two remote locations. Instead, they

[1]Tofino, developed by Barefoot Networks (now part of Intel), is a fully programmable network switch chip widely used in high-performance data center networks and cloud computing environments. In this paper, we use "Tofino" as a general term to refer to the programmable data plane found in Tofino or similar P4-based hardware switches.

TABLE II
BASIC REQUIREMENTS OF NETWORK IMPAIRMENT EMULATION

Function	Description
Interface Specification	Supporting 10/100Mbps, 1/10/100/200Gbps and higher electrical or optical interfaces, compliant with IEEE 802.3.
Line-speed Delay	1Gbps: $\leq 100\mu s$, 10Gbps: $\leq 10\mu s$, 100Gbps: $\leq 1\mu s$.
Basic Function	Supporting multiple packet matching methods including exact matching, wildcard matching, regular expression matching, content matching, etc.
Functional Impairment	The function of network system is affected, and the expected operation or service can not be performed normally.
Packet Tampering	Including but not limited to bit tampering, byte tampering, field tampering or data tampering of packets with a specified probability or specified period.
Packet Duplication	Including but not limited to copying and forwarding packets with a specified probability or a specified period(interval) or specified matching rules.
Packet Loss	Including but not limited to discarding of packets with a specified probability or a specified period (interval) or a specific burst mode.
Packet Reordering	Including but not limited to reordering and forwarding data packets with a specified probability using time or packet interval as the measurement unit.
Packet Fragmentation	Including but not limited to fragmenting large packets with a specify fragmentation threshold.
Performance Impairment	The performance of the network decreases or changes, but the network function is still available.
Bandwidth Impairment	Including but not limited to limiting network bandwidth with fixed or fluctuating limits.
Delay/Jitter Impairment	Including but not limited to increasing the packet forwarding delay with the fixed/uniform/normal/long-tail distribution probability or a custom pattern.
Other Impairment	Other network impairments that are inconvenient to classify as functional impairment or performance impairment.
Compound Impairment	Supporting the combination of multiple basic functional impairments or performance impairments.
Impairment Function Virtualization	Supporting logical division of multiple virtual data transmission links on a single physical link and loading of impairments.
Specific Protocol Impairment	Supporting the emulation of typical network protocol behaviors, e.g., Differentiated Services (DS), Random Early Drop (RED), Explicit Congestion Notification (ECN), and Priority-based Flow Control (PFC).
User Interaction	Providing the Graphical User Interface (GUI), RESTful API and more efficient interaction methods.
Security Feature	Including but not limited to network storm suppression, access control lists, password security, and permission management.
Management Feature	Including but not limited to log management, configuration management, and alarm notifications, compatible with multiple management access methods.

only need to build a small testing environment and emulate a 1,100-kilometer network through a NIE.

Table II summarizes the common requirements of NIE, detailing their capabilities in creating near-realistic testing environments.

Currently, CPU-based or FPGA-based NIEs are the most popular commercial solutions, such as zMonkey [22] and HoloWAN [23]. zMonkey uses Data Plane Development Kit (DPDK) and supports Intel E810 and Mellanox ConnectX-5. HoloWAN uses a hybrid architecture of DPDK and FPGA, making it one of the highest-performing commercial products.

The advent of the Programmable Protocol-Independent Packet Processor (P4) [24], designed for high-speed and flexible packet processing, introduces a novel approach to implementing network impairments. On a related note, by combining programmable chips (e.g., Tofino [25]) and the P4 programming language [26], P4 has the potential to transform traditional switches into high-performance traffic classifiers [27], generators [28] and traffic replayers [29], showcasing the extensive capabilities of P4 for network testing [30].

Overall, it is technically feasible to repurpose a programmable switch as a network impairment emulator. TurboNet [31] argues that the existing network experiment platforms either fail to accurately emulate the functionality and performance of production networks or struggle with scalability due to cost constraints. To overcome these challenges, TurboNet employs one or more programmable switches to achieve faithful emulation of both the network data plane and control plane. For data plane emulation, TurboNet proposes several key components, including the port mapper, queue mapper, and delayed queue, to emulate network topologies and performance metrics with high flexibility and accuracy. For control plane emulation, TurboNet supports static routing configurations, distributed routing agents, and centralized routing controllers. However, TurboNet focuses on the construction of large-scale network test topology, neglecting how to achieve single-machine impairment at rates of 100Gbps or higher.

P7 [32] is a network emulator designed for P4-enabled devices. It allows for the emulation of network topologies using mechanisms like recirculations, port configurations, match-action tables, and Direct Attach Copper (DAC) cables. Network link characteristics are emulated through both simple metrics (e.g., connectivity, latency, bandwidth) and more advanced metrics (e.g., packet loss [%], jitter [ms], reordering [%]). However, P7 relies heavily on the recirculation mechanism, resulting in an obvious forwarding performance bottleneck. In addition, the network impairment functionality of P7 is overly simplistic and cannot accommodate the varied

TABLE III
COMPARISON OF COST-PERFORMANCE OF NIES BASED ON DIFFERENT ARCHITECTURES

NIE	CPU-based	FPGA-based	P4-based
Representative Product	zMonkey [22]	HoloWAN [23]	P7 [32]/TurboNet [31]/SDNIE
Capacity, Line-Rate	10Gbps*4/50Gbps*1/100Gbps*1, No	100Gbps*4, Yes	**100Gbps*32+400Gbps*16, Yes**
Cost-effectiveness	**$7500** ($75/Gbps)	$54000 ($135/Gbps)	$11500 **($1.28/Gbps)**
Memory	64GB DDR	8GB HBM+32GB DDR+1MB BRAM	8GB DDR+16MB Buffer
Power	750W (7.5W/Gbps)	750W (1.87W/Gbps)	**400W (0.06W/Gbps)**

requirements for impairment emulation. For example, it only supports fixed-probability packet loss impairments and coarse-grained delay impairments. Moreover, it is only suitable for software switches such as BMv2 [33] and cannot fully leverage the line-rate performance of P4 switches.

In addition, the cost-performance comparison of common NIEs based on CPU, FPGA, and P4 is shown in Table III, and P4 demonstrates excellent cost-effectiveness[2].

III. CHALLENGES AND KEY REQUIREMENTS

The challenges of software-defined network impairment emulation can be summarized below as the contradiction between the diversity of network impairment function requirements and the capability limitations of P4 hardware/software.

1. **Limited Hardware Resource of the Tofino Chip**. The restricted capacity of Ternary Content-Addressable Memory (TCAM) and Static Random Access Memory (SRAM) limits support for large-scale rule tables and their flexibility [34]. Additionally, the buffer size (typically 16MB or 32MB) becomes a bottleneck in high-precision delay emulation.

2. **Limited Expressiveness of the P4 Language**. P4 is designed to describe packet forwarding behavior and lacks direct support for complex state logic and advanced processing functions. Therefore, when emulating complex network impairment such as network delay/jitter and packet loss, it requires sophisticated programming techniques to overcome language limitations.

To overcome these challenges, SDNIE must optimize hardware resource utilization and overcome the expressiveness limitations of the P4 language. For the former, rule merging can optimize the use of TCAM and SRAM, while certain impairment functions can be offloaded to CPU hardware resources [35]. For the latter, complex impairment functionalities can be implemented by designing finite state logic as multi-level processing flows, utilizing registers and counters to manage packet states where necessary.

IV. SOFTWARE-DEFINED NETWORK IMPAIRMENT EMULATION

Unlike the simulation, the purpose of a network impairment emulation is usually to achieve high-performance emulation of end-to-end network characteristics using a single device. The basic workflow of NIE can be summarized as follows:

[2]Here, the comparison of cost-effectiveness relies on the following typical platforms: CPU-based is the Inspur 5280M5 server, FPGA-based is the HoloWAN HPP 100G emulator, and P4-based is the OpenMesh BF-32D16C switch.

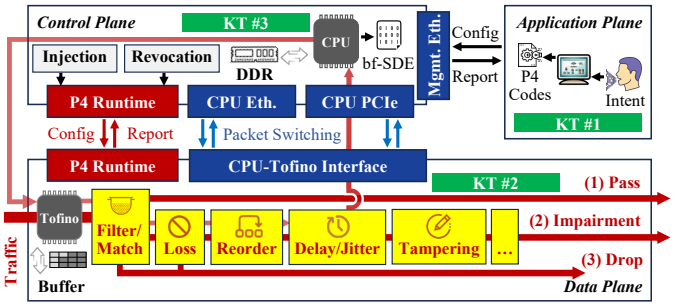


Fig. 2. The Architecture and Workflow of SDNIE.

(1) Task Reception, which is usually a description of link characteristics or network functions, such as achieving 1% random packet loss for packets with $sourceIP == 102.168.0.1$;

(2) Impairment Emulation, which is to implement the impairment task with high performance and cost-efficiency [36]. This often requires a combination of both software and hardware components, including high-speed packet processing and effective buffer management.

(3) Result Feedback, which is to provide feedback on the impairment process and its results, presented through visual charts and reports for easy analysis and interpretation.

This workflow ensures precise, efficient, and cost-effective network impairment emulation, allowing for effective testing and performance evaluation of network systems.

Corresponding to the workflow, the SDNIE can be generally divided into three planes: Application, Control, and Data, as shown in Figure 2.

Application Plane: which provides the interactive interface of NIE, receives impairment tasks and automatically generates P4 programs.

Control Plane: which is responsible for compiling the code and executing the impairment tasks.

Data Plane: which decomposes the impairment into specific stages (yellow block). If a packet exceeds the capacity of the Tofino, it is delivered to the CPU for processing (red process). According to the flow-table rules, all packets will eventually have one of three outcomes: being passed directly, impaired, or dropped.

For the three planes, the SDNIE has further improved three key technologies to support high-performance network impairment emulation, namely

Key Technology #1: Intent-driven Network Impairment Configuration,

Key Technology #2: Serial-parallel Combined Impairment Execution,

Key Technology #3: CPU-Tofino Collaborative Impairment Deployment.

Among them, the first one can improve the input efficiency of impairment intent, and the latter two solve the impairment function implementation for P4 switches. These three are not involved in TurboNet [31] and P7 [32].

A. Intent-driven Network Impairment Configuration

Existing methods for configuring network impairments face several key challenges:

(1) Inaccuracy in Describing Network Impairment Emulation Requirements: Natural language descriptions of network impairment emulation needs can often be imprecise. Different engineers may interpret the same impairment intent in varied ways, leading to miscommunications or inconsistencies in configuration and deployment. This discrepancy can result in network impairments that do not align with the original intents.

(2) Inefficiency and Complexity: Current processes for configuring network impairments are often cumbersome and inefficient, requiring significant time and effort.

(3) Lack of Compatibility Across Manufacturers: Impairment configurations from different manufacturers are typically not interchangeable, limiting flexibility and interoperability.

Therefore, SDNIE uses an intent-driven network impairment configuration method for programmable switches, which automatically converts the user high-level intents into low-level P4 program configurations to implement various network impairment emulations. To this end, we propose a method for generating network impairment configurations enhanced by domain knowledge. The method significantly improves the accuracy and reliability of P4 program generation through the construction of a domain ontology for network impairment, the design of a multi-level semantic parsing algorithm, and the implementation of a code correctness verification mechanism.

1) Problem Formalization: Given a user intent description $I \in \mathcal{L}_{NL}$ (natural language space), the network impairment configuration generation problem is defined as finding the mapping function

$$\mathcal{F} : \mathcal{L}_{NL} \times \mathcal{O} \times \mathcal{R} \to \mathcal{S} \times \mathcal{Q}, \tag{1}$$

where $\mathcal{O}$ is the domain ontology knowledge space, $\mathcal{R}$ is the hardware resource constraint space, $\mathcal{S}$ is the structured configuration space, and $\mathcal{Q}$ is the configuration quality assessment space. The structured configuration $\mathcal{S} = \langle F, P, C, \Phi \rangle$ consists of four components defined as

$$\begin{aligned} F &= \{f_i \mid f_i \in \mathcal{F}_{imp}, 1 \le i \le n\}, \\ P &= \{p_j \mid p_j \in \bigcup_{f \in \mathcal{F}_{imp}} \mathcal{D}_{param}(t), 1 \le j \le m\}, \\ C &= \{c_k \mid c_k \in \mathcal{M}_{match}, 1 \le k \le l\}, \\ \Phi &\in \mathcal{L}_{P4}, \end{aligned} \tag{2}$$

where $\mathcal{F}_{imp}$ is the impairment type domain, $\mathcal{D}_{param}(f)$ is the parameter domain for impairment funciton f, $\mathcal{M}_{match}$ is the matching condition space, and $\mathcal{L}_{P4}$ is the P4 language space.

The configuration constraint system is defined as a 4-tuple

$$\mathcal{C}_{sys} = \langle \mathcal{V}, \mathcal{D}, \mathcal{K}, \mathcal{O}_{obj} \rangle, \tag{3}$$

where $\mathcal{V} = \mathcal{V}_{hw} \cup \mathcal{V}_{perf} \cup \mathcal{V}_{sem}$ represents the set of constraint variables comprising hardware-related, performance-related, and semantics-related components. $\mathcal{D} = \{D_v \mid v \in \mathcal{V}\}$ is the variable domain set, $\mathcal{K} = \mathcal{K}_{hard} \cup \mathcal{K}_{soft}$ is the constraint set, $\mathcal{O}_{obj}$ is the optimization objective function.

The constraint set $\mathcal{K}$ can be classified into two categories:

(1) Hard Constraints $\mathcal{K}_{hard}$:

(a) Resource constraints: $\forall r \in \mathcal{R}_{hw} : \sum_i usage_i(r) \le capacity(r)$, ensuring that resource utilization does not exceed hardware capacity limits.

(b) Syntax constraints: $\Phi \in \mathcal{L}_{P4}^{valid}$, where $\mathcal{L}_{P4}^{valid}$ represents the set of syntactically correct P4 programs.

(c) Semantic constraints: $\models_{sem} (\Phi, I)$, ensuring that the P4 program Φ semantically entails the user intent I.

(2) Soft Constraints $\mathcal{K}_{soft}$:

(a) Performance preference: $performance(\Phi) \ge threshold_{perf}$, specifying desired performance levels for the generated configuration.

(b) Resource efficiency: $efficiency(\Phi) \ge threshold_{eff}$, promoting efficient utilization of available resources.

(c) Maintainability: $maintainability(\Phi) \ge threshold_{main}$, ensuring the generated program is maintainable and readable.

To quantify the semantic alignment between intent and generated program, we define the semantic distance as follows.

The semantic distance between user intent I and P4 program Φ is defined as

$$d_{sem}(I, \Phi) = \|embed_{NL}(I) - embed_{P4}(\Phi)\|_2, \tag{4}$$

where $embed_{NL}(\cdot)$ and $embed_{P4}(\cdot)$ are embedding functions for natural language and P4 program respectively. Therefore, the objective function of the network impairment configuration can be expressed as finding the P4 program with the smallest semantic distance to the input intent in the valid P4 program set, that is,

$$\min_{\Phi} d_{sem}(I, \Phi) \quad \text{s.t.} \quad \Phi \in \mathcal{L}_{P4}^{valid}. \tag{5}$$

Another understanding of Equation 5 is to achieve reliable inference under limited knowledge. We define the coverage of domain knowledge base $\mathcal{K}$ over intent space $\mathcal{I}$ as

$$coverage(\mathcal{K}, \mathcal{I}) = \frac{|\{I \in \mathcal{I} \mid \exists k \in \mathcal{K} : covers(k, I)\}|}{|\mathcal{I}|}, \tag{6}$$

where $\mathcal{I}_{covered} = \{I \in \mathcal{I} \mid \exists k \in \mathcal{K} : covers(k, I)\}$ and $\mathcal{I}_{uncovered} = \mathcal{I} \setminus \mathcal{I}_{covered}$. The reliability of inference can be expressed as

$$Reliability = \begin{cases} P(correct \mid I, \mathcal{K}) \ge \theta_{high}, & \text{if } I \in \mathcal{I}_{covered}; \\ P(correct \mid I, \mathcal{K}) \ge \theta_{low}, & \text{if } I \in \mathcal{I}_{uncovered}. \end{cases} \tag{7}$$

2) Impairment Configuration Generation based on Domain Knowledge Enhancement: We construct a hierarchical network impairment domain ontology $\mathcal{O} = \langle \mathcal{C}, \mathcal{R}, \mathcal{A} \rangle$ containing concepts, relationships, and axioms. Let $\mathcal{F}_{imp}$ denote the domain of all impairment functions as shown in Table II.

For any two impairment functions $f_i, f_j \in \mathcal{F}_{imp}$, we define three key relationships:

(1) Dependency: depend(f_i, f_j) indicates that f_i depends on f_j.

(2) Conflict: conflict(f_i, f_j) indicates mutual conflict.

(3) Compatibility: compatible(f_i, f_j) indicates the feasibility of parallel execution, which is discussed in Section IV-B.

Definition 1 (Resource Compatibility): Two impairment functions f_i, f_j are resource compatible if

$$\text{ResCom}(f_i, f_j) \Leftrightarrow \text{ResSet}(f_i) \cap \text{ResSet}(f_j) = \emptyset, \quad (8)$$

where ResSet(f) denotes the set of hardware resources required by impairment function f.

Definition 2 (Impairment Compatibility): Two impairment functions $f_i, f_j \in \mathcal{F}_{imp}$ are compatible if and only if

$$\text{compatible}(f_i, f_j) \Leftrightarrow \neg\text{conflict}(f_i, f_j) \wedge \text{ResCom}(f_i, f_j). \quad (9)$$

Theorem 1 (Parallelization Condition): Given an impairment functions set $\mathcal{F}_{imp} = \{f_1, f_2, \ldots, f_n\}$ and its subset $F' \subseteq \mathcal{F}_{imp}$, the subset F' can execute in parallel if and only if

$$\forall f_i, f_j \in F', i \neq j : \text{compatible}(f_i, f_j). \quad (10)$$

Proof: The proof follows directly from Definition 2. Parallel execution requires absence of both semantic conflicts and resource conflicts among all impairment function pairs in F'.

We employ LLaMA 3-8B as the core language model to implement the configuration framework. The theoretical embedding functions $embed_{NL}(\cdot)$ and $embed_{P4}(\cdot)$ are realized through LLaMA hidden state representations:

$$embed_{NL}(I) = \text{LLaMA}_{encoder}(I)[-1] \in \mathbb{R}^{d_{model}}, \quad (11)$$

$$embed_{P4}(\Phi) = \text{LLaMA}_{encoder}(\Phi)[-1] \in \mathbb{R}^{d_{model}}, \quad (12)$$

where d_{model} is the hidden dimension of LLaMA 3-8B.

Directly computing the semantic distance between a user intent I and a candidate P4 program Φ in a high-dimensional embedding space is computationally expensive and often impractical for large-scale systems. To address this, we propose an efficient approximation strategy that leverages the generative probability of the LLM and explicit constraint violation penalties.

Generative Probability: $P_{LLaMA}(\Phi|I)$ denotes the conditional probability that LLaMA 3-8B generates the code Φ given the intent I. A higher probability indicates better semantic alignment.

Constraint Violation Penalty: ConstraintViolation(Φ) quantifies the degree to which Φ violates hard or soft constraints (e.g., constraint set $\mathcal{K}$). The hyperparameter λ balances the trade-off between semantic alignment and constraint satisfaction.

Thus, the overall semantic distance of Equation 4 is approximated as:

$$\hat{d}_{sem}(I, \Phi) = -\log P_{LLaMA}(\Phi|I) + \lambda \cdot \text{ConstraintViolation}(\Phi) \quad (13)$$

where a lower value indicates a better candidate. In practice, $P_{LLaMA}(\Phi|I)$ can be estimated by the product of token probabilities output by the LLM, and ConstraintViolation(Φ) can be a weighted sum of binary or continuous constraint checks.

Furthermore, to overcome the limitations of LLaMA in inference of P4, we employ a Retrieval-Augmented Generation (RAG) framework. The domain knowledge base $\mathcal{K}_{kb}$[3] consists of structured representations of network impairment patterns, configuration templates, and best practices, each encoded as a vector in a semantic space. The user intent I is embedded into the same vector space using $embed_{NL}(I)$. We compute the similarity between $embed_{NL}(I)$ and all knowledge vectors in $\mathcal{K}_{kb}$, retrieving the top-k most relevant knowledge items TopK (Sim$(embed_{NL}(I), \mathcal{K}_{kb})$), where Sim$(\cdot, \cdot)$ is Euclidean distance. And the top k pieces of knowledge with the highest similarity selected by the TopK operation can be set to $k = 5$.

The process of generating an optimal P4 configuration from user intent is formulated as a constrained optimization problem. As shown in Figure 3, the practical solution involves the following steps:

(1) Knowledge Retrieval: Retrieve relevant domain knowledge using the RAG mechanism described above.

(2) Prompt Construction: Construct an enhanced prompt that includes the user intent, retrieved knowledge, and any relevant hardware/resource constraints.

(3) Candidate Generation: Use LLaMA 3-8B to generate multiple candidate P4 programs $(\Phi_1, \Phi_2, ..., \Phi_k)$ conditioned on the enhanced prompt.

(4) Constraint Validation: For each candidate, check for compliance with constraints $\mathcal{K}$ and compute the constraint violation score.

(5) Scoring: Compute the semantic distance $\hat{d}_{sem}(I, \Phi_i)$ for each valid candidate.

(6) Selection: Select the candidate with the lowest semantic distance as the final configuration:

$$\Phi^* = \arg\min_{\Phi_i} \hat{d}_{sem}(I, \Phi_i). \quad (14)$$

If no valid candidate is found, the template-based generation is triggered or the generation is exited directly, which is the default operation of SDNIE.

B. Serial-Parallel Combined Impairment Execution

To achieve efficient and correct execution, we propose a serial-parallel combined execution mechanism that carefully manages the dependencies and conflicts between different impairment functions. First, an impairment task may involve multiple impairment functions, which could conflict with or depend on each other, such as in cases of priority. To

[3]The example knowledge base can be obtained from https://tanlizhuang.cn/data_cn.html. For commercial considerations, we only open example of the knowledge base.

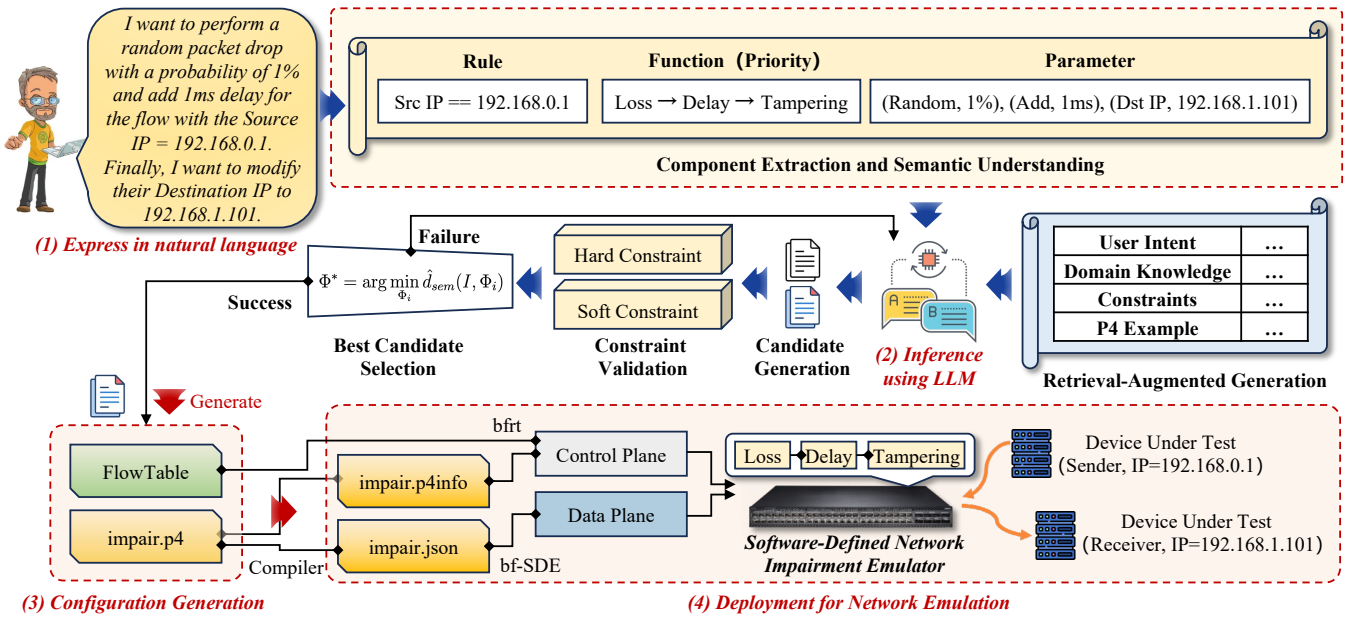


Fig. 3. Intent-driven Network Impairment Configuration in SDNIE.

systematically describe the operations involved, we identify four basic actions that an impairment function can perform on packets.

Definition 3 (Basic Impairment Actions): For any impairment function $f \in \mathcal{F}_{imp}$, its operations can be decomposed into four basic actions:

$$\mathcal{A}(f) = read, write, add/remove, drop. \tag{15}$$

Furthermore, we define the compatibility of the impairment function.

Definition 4 (Function Compatibility): Two impairment functions $f_i, f_j \in \mathcal{F}_{imp}$ are compatible if they satisfy all of the following conditions:

(1) Their actions do not conflict on shared resources.

(2) They do not have order-dependent effects on packet processing.

(3) Neither function includes a drop action.

Formally, we define compatibility as

$$\text{compatible}(f_i, f_j) = \begin{cases} \text{true}, & \text{if } \mathcal{A}(f_i) \cap \mathcal{A}(f_j) \subseteq \{read\} \\ \text{true}, & \text{if ResSet}(f_i) \cap \text{ResSet}(f_j) = \emptyset \\ \text{false}, & \text{if } drop \in \mathcal{A}(f_i) \cup \mathcal{A}(f_j) \\ \text{false}, & \text{otherwise} \end{cases} \tag{16}$$

Theorem 2 (Execution Correctness): Given a set of impairment functions $\mathcal{F}_{imp} = \{f_1, ..., f_n\}$, the serial-parallel execution maintains correctness if and only if for any two functions $f_i, f_j \in \mathcal{F}_{imp}$:

$$\text{Result}(f_i \text{ followed by } f_j) = \text{Result}(f_j \text{ followed by } f_i), \tag{17}$$

when $\text{compatible}(f_i, f_j) = \text{true}$. Here, $\text{Result}(\cdot)$ represents the final packet state after applying the impairment functions in the specified order.

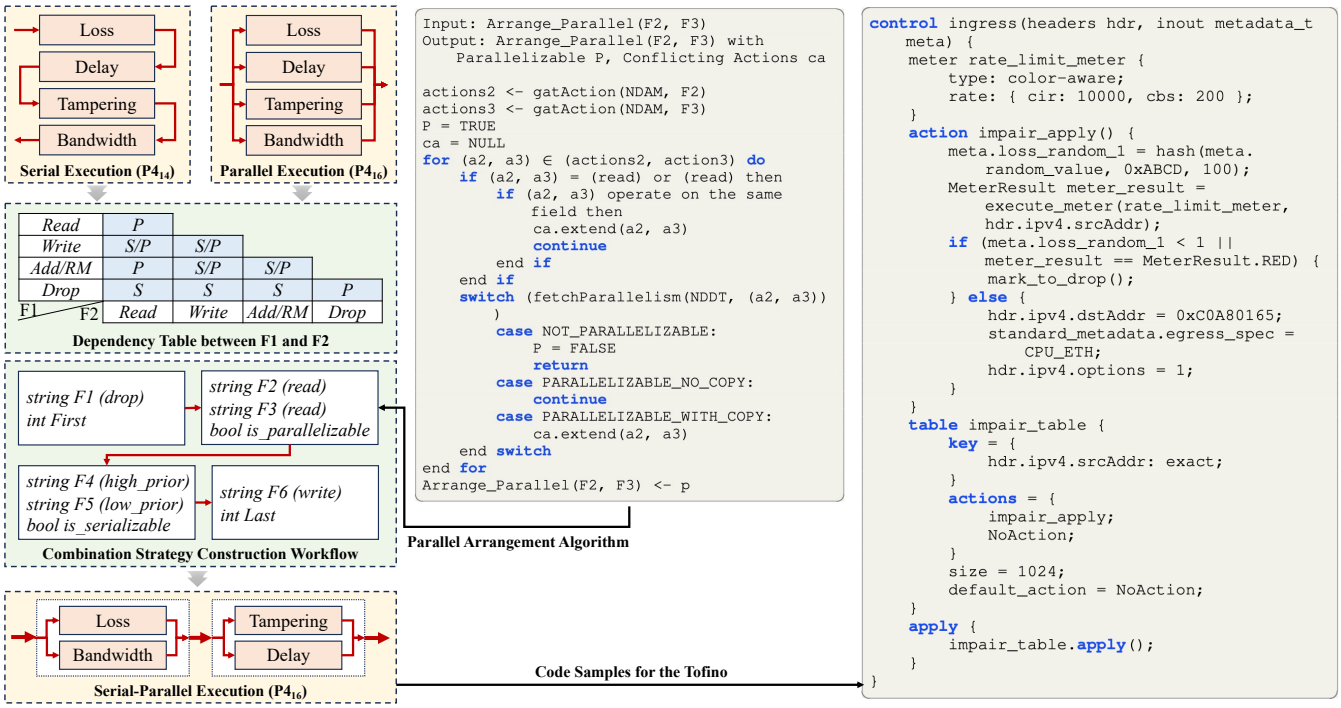
Proof: Let p be any input packet:

(1) Necessity: If f_i and f_j are compatible but $f_i(f_j(p)) \neq f_j(f_i(p))$, then the execution order affects the result, violating determinism.

(2) Sufficiency: When $f_i(f_j(p)) = f_j(f_i(p))$ for compatible functions, parallel execution is equivalent to any sequential ordering, ensuring correctness.

The dependency table in Figure 4 provides a comprehensive analysis of action relationships between two impairment functions (f_i and f_j). Among them, S represents only serialization, P represents parallelization, and S/P represents free combination according to specific circumstances. According to the priority between functions, SDNIE uses the combination strategy construction workflow to describe the sequential relationship and combination between different impairment functions, and determine whether two impairment functions can be parallelized through the Parallel Arrangement Algorithm. As shown in the lower left corner of Figure 4, with this approach, SDNIE first combines the loss impairment and bandwidth impairment since both may drop packets. It then merges tampering impairment and delay impairment, and finally executes the two aggregated impairments sequentially.

Finally, as shown in the right of Figure 4, we present a new serial-parallel execution solution and a P4 program example. Different hardware platforms and compilation tools have different process control for the different version of P4 [37]. In $P4_{14}$, an early version of the P4 programming language, the compiled logic is executed strictly in the order in which the code is written [38]. This sequential execution model is relatively simple but lacks flexibility [39]. Conversely, in $P4_{16}$ [40], the execution is based on a table-driven model that allows for dynamic branching and conditional execution. When they implement the same impairment functions, the processing delay and hardware resource usage are different [41].




```
Input: Arrange_Parallel(F2, F3)
Output: Arrange_Parallel(F2, F3) with
    Parallelizable P, Conflicting Actions ca

actions2 <- gatAction(NDAM, F2)
actions3 <- gatAction(NDAM, F3)
P = TRUE
ca = NULL
for (a2, a3) ∈ (actions2, action3) do
    if (a2, a3) = (read) or (read) then
        if (a2, a3) operate on the same
            field then
            ca.extend(a2, a3)
            continue
        end if
    end if
    switch (fetchParallelism(NDDT, (a2, a3))
        )
        case NOT_PARALLELIZABLE:
            P = FALSE
            return
        case PARALLELIZABLE_NO_COPY:
            continue
        case PARALLELIZABLE_WITH_COPY:
            ca.extend(a2, a3)
    end switch
end for
Arrange_Parallel(F2, F3) <- p
```

```
control ingress(headers hdr, inout metadata_t
    meta) {
    meter rate_limit_meter {
        type: color-aware;
        rate: { cir: 10000, cbs: 200 };
    }
    action impair_apply() {
        meta.loss_random_1 = hash(meta.
            random_value, 0xABCD, 100);
        MeterResult meter_result =
            execute_meter(rate_limit_meter,
            hdr.ipv4.srcAddr);
        if (meta.loss_random_1 < 1 ||
            meter_result == MeterResult.RED) {
            mark_to_drop();
        } else {
            hdr.ipv4.dstAddr = 0xC0A80165;
            standard_metadata.egress_spec =
                CPU_ETH;
            hdr.ipv4.options = 1;
        }
    }
    table impair_table {
        key = {
            hdr.ipv4.srcAddr: exact;
        }
        actions = {
            impair_apply;
            NoAction;
        }
        size = 1024;
        default_action = NoAction;
    }
    apply {
        impair_table.apply();
    }
}
```

Fig. 4. Serial-Parallel Combined Impairment Execution in SDNIE.

C. CPU-Tofino Collaborative Impairment Deployment

Currently, P4 is insufficient in implementing complex logic functions and cannot support the growth of SDNIE functional requirements in the future. This is reflected in two aspects: (1) The programming ability of the P4 language is weak and cannot describe complex impairment intents. For example, the random number generated by P4 has a fixed bit width (usually 10 bits, i.e., 0-1023), which cannot generate high-precision random numbers, thus limiting the flexibility of probability-based packet loss impairment. (2) Tofino buffer resources are precious and should not be wasted by Recirculation or Clone [42]. Consider that the existing P4 switch has only two processing engines, CPU and Tofino, which are configured with larger RAM and smaller buffer respectively. These language and resource issues can be mitigated via CPU-Tofino collaborative deployment.

First, we define the implementation capabilities of Tofino.

Definition 5 (Implementation Capability): For any impairment function $f_i \in \mathcal{F}_{imp}$, its implementation capability on Processor Pr is defined as

$$\text{Cap}(f_i, Pr) = \begin{cases} \text{Full}, & \text{if } Pr \text{ can implement } f_i \text{ independently;} \\ \text{Partial}, & \text{if } Pr \text{ can implement part of } f_i; \\ \text{None}, & \text{otherwise.} \end{cases} \tag{18}$$

Let f_i be an arbitrary network impairment function,and $Pr \in \{\text{Tofino}, \text{CPU}\}$ be candidate processors.The processor assignment for f_i is given by

$$Pr = \begin{cases} \text{Tofino}, & \text{if Cap}(f_i, \text{Tofino}) = \text{Full;} \\ \text{Tofino+CPU}, & \text{if Cap}(f_i, \text{Tofino}) = \text{Partial;} \\ \text{CPU}, & \text{if Cap}(f_i, \text{Tofino}) = \text{None.} \end{cases} \tag{19}$$

Here, we take delay impairment as an example. As shown in Figure 2, for complex delay impairments, such as emulating the long-tail delay in wide area networks, it is challenging for Tofino to calculate an accurate qualified time, which can be easily obtained on the CPU. In addition, increasing the retention time of packets on the switch using the Recirculation mechanism is inefficient because the packets continuously occupy buffer space. Theoretically, for a fixed delay of 10ms and 16MB of available buffer resources, the maximum throughput that the switch can achieve is only 12.8Gbps.

The most effective solution to the problem of insufficient Tofino buffer space is to add DDR5 RAM, or to indirectly integrate high-bandwidth memory HBM3 [43]. Both can easily accommodate 100Gbps network flow. However, the former has a slightly lower IO bandwidth, while the latter can reach up to 1.2TB/s. However, on existing programmable switches, SDNIE overcomes this problem through CPU-Tofino collaboration. SDNIE expects to use the CPU to store and process packets, which relies on the traffic shaping mechanisms, and should use Tofino to preliminarily calculate the qualified time and determine whether the packet has met it. This can be transmitted through the CPU Eth., which is a virtual interface that the CPU communicates with Tofino on the programmable switch. Using multi-core CPU and DPDK [44], SDNIE can achieve line-rate close to 100Gbps. In addition to CPU Eth. path, the PCIe path can also enable data exchange between the CPU and Tofino, but this is not currently available.

For a delay impairment task, when the Tofino buffer cannot be cached and the recirculation mechanism has the performance bottleneck, as shown in Figure 5, its processing flow is Ingress $\rightarrow$ Tofino $\rightarrow$ CPU $\rightarrow$ Tofino $\rightarrow$ Egress. The division of labor between the two chips is as follows:

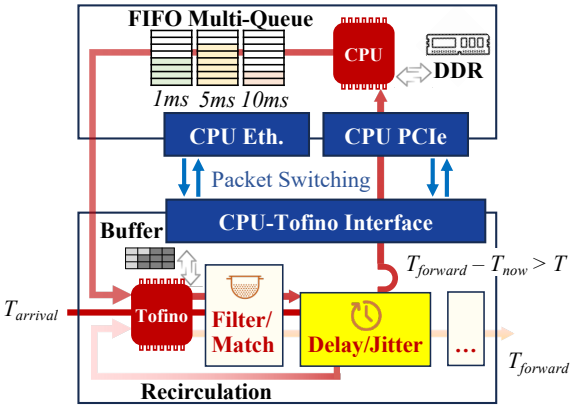


Fig. 5. The Implementation Example of Delay Impairment.

(1) The Tofino is responsible for packet forwarding and calculates the qualified time of the packet $T_{forward} = T_{arrival} + D$. If $T_{forward} - T_{Now} > T$, then this packet should be sent to the CPU. Among them, $T_{arrival}$ is the packet arrival time, T_{now} is the current time. T is a threshold, which can be obtained by $T = Q_{if}/R + RTT_{T-C}$, where Q_{if} and R are the real-time queue length and throughput of the interface, RTT_{T-C} is the RTT between Tofino and CPU. T is usually small.

(2) The CPU processes the packets received by Eth. through multiple-core and DPDK, maintains multiple queues with different residence times (e.g., 1ms, 5ms, 10ms, etc.), and implements FIFO in the queues to reduce the overhead of delay impairment.

In general, through the above three mechanisms, SDNIE further enriches the impairment function of the switch as a impairment emulator and improves its processing capability based on TurboNet [31] and P7 [32]. [4]

V. Experimental Evaluation

Before quantitatively evaluating SDNIE, we qualitatively compare the functionality and nominal performance of SDNIE with the emulators we were able to obtain, including:

CPU-based: zMonkey [22],
FPGA-based: HoloWAN [23],
P4-based: TurboNet [31] and P7 [32].

In general, as shown in Table IV, zMonkey does not support large-scale topology emulation and relies on the performance of CPU/NIC and PCIe, resulting in large jitter errors. TurboNet and P7 also rely on Tofino hardware, similar to SDNIE, but are inferior to SDNIE in terms of delay and loss impairment. The functionality and performance of SDNIE are comparable to HoloWAN, a high-performance commercial impairment emulator.

A. Setup

As shown in Figure 6, we have developed a SDNIE prototype based on a commercial Barefoot Tofino switch (OpenMesh BF-32X) equipped with an Intel J1900 4-core 2.0GHz CPU and 8GB RAM, with 32*100GE(QSFP28) interfaces. The Tofino-1 chip provides 3.2 Tbps switching bandwidth and 16MB buffer. The two devices under test are Inspur 5280M5 servers equipped with NVIDIA BlueField-3 SmartNIC (100G). The LLM model runs on the cloud GPU server [45] with 1 RTX 4090 (24GB) GPU.

[4]It should be noted that SDNIE focuses on improving the impairment emulation performance of a single switch rather than building large-scale simulation topologies, which is the focus of TurboNet and P7.

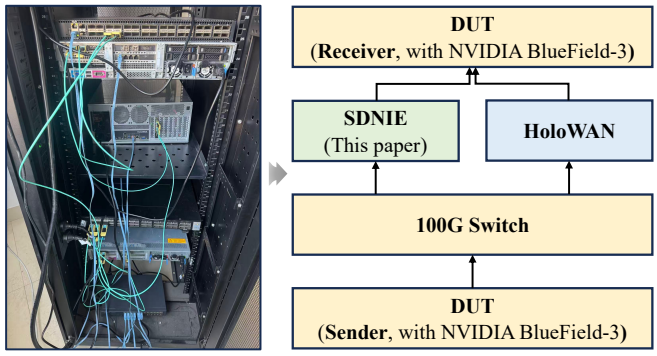


Fig. 6. Aerial View of the Prototype and Experimental Topology.

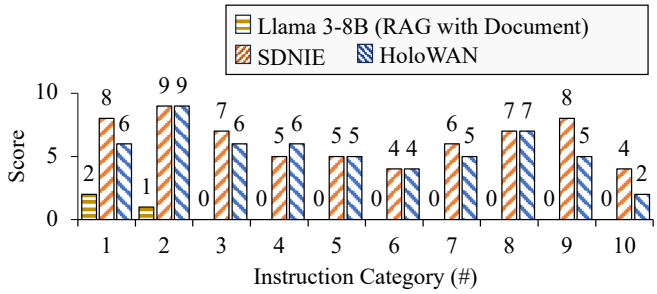


Fig. 7. Configuration Performance.

B. Result

1) Configuration Performance: We selected 10 types of network impairment configuration instructions to form a test set[5], including 1 basic impairment and 9 complex impairments, as shown in Table V. Each type of task has 10 tasks, totaling 100 tasks. We verified the performance of Llama 3-8B (Retrieval-Augmented Generation with Document), SDNIE and HoloWAN [23], which is a commercial NIE that supports intent configuration. The experiment results are shown in Figure 7. The Llama 3-8B augmented with P4 development documents performed poorly and was only effective for limited basic impairments, with no improvement from adjusting training parameters. SDNIE achieves satisfactory results, surpassing HoloWAN by 8 points. It should be noted that HoloWAN is an FPGA-based emulator, which has made approximate replacements for some of the P4 instructions of test set.

2) Execution Performance: We impose the following four network impairments on UDP traffic at 10Gbps.

Loss: Set a fixed loss with the probability of 0.2%.
Bandwidth: Limit UDP traffic to 1Gbps.
Delay: Add a fixed delay with 100us for all packets, implemented through P4 recirculate mechanism.

[5]The test set can be obtained from https://tanlizhuang.cn/data.html.

TABLE IV
PERFORMANCE COMPARISON OF THE FOUR FUNCTIONS FOR FIVE EMULATORS.

Function	Topology	Bandwidth/Pkt Fwd. Rate/Acc.	Delay/Acc.	Loss/Acc.
zMonkey	○	200Gbps/90Mpps/⋆	100ms/Unknown	●/0.1%
HoloWAN	○	100Gbps/148.81Mpps/1bps	100ms/±4ns	●/0.000001%
TurboNet	●	●/⋆/⋆	1ms/±9us	●/1%
P7	●	100Gbps/148.81Mpps/100Kbps	ms-level/⋆	●/⋆
SDNIE	●	100Gbps/148.81Mpps/100Kbps	100ms/±1us	●/0.05%

○ means no support. ● means support. ⋆ means hardware dependency, it is hard to get a conclusion. "Acc." stands for "Accuracy".

TABLE V
IMPAIRMENT TEST SET

No.	Description
1	Basic configuration
2	Packet loss
3	Delay
4	Packet tampering
5	Traffic redirection/Mirroring
6	Traffic shaping/Rate limiting
7	Packet reordering
8	Explicit congestion notification
9	Multicast
10	Traffic monitoring/statistics

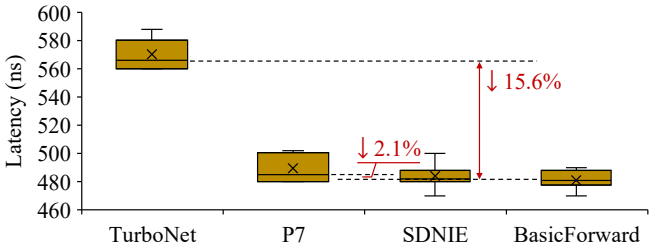


Fig. 8. Single Packet Forwarding Latency.

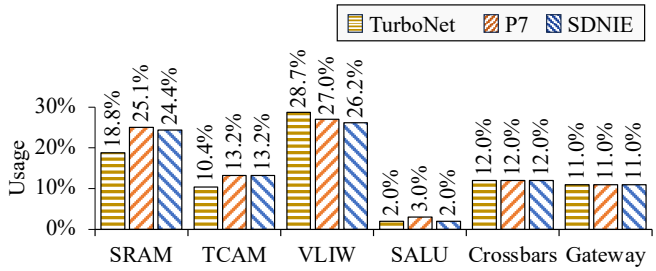


Fig. 9. Resource Usage.

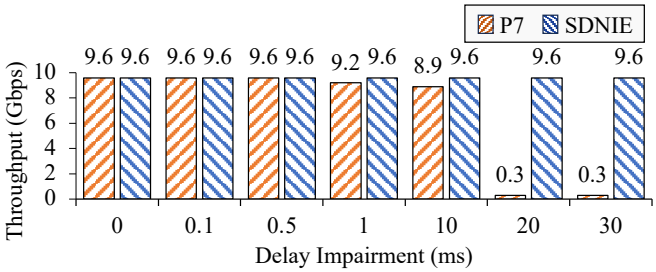


Fig. 10. TCP Throughput Collapse under Delay Impairment.

Tampering: Modify the source IP to 192.168.1.1. Native TurboNet and P7 do not support this impairment, we extended their codebases to support this impairment for a fair comparison.

TurboNet uses a serial execution strategy, P7 uses a parallel execution strategy, and SDNIE uses the serial-parallel execution strategy described in Section IV-B. First, we measure the single packet forwarding latency after loss impairment without enabling delay impairment. The results are shown in Figure 8. Compared with the other two, the average packet forwarding latency of SDNIE decreased by 15.6% and 2.1% respectively. Lower latency helps SDNIE deploy more impairment functions under sensitive packet forwarding constraints. Furthermore, compared to basic forwarding, SDNIE shows almost no increase in average forwarding latency.

Then, we evaluate the hardware ASIC resources usage, including memory resources (SRAM, TCAM) and computing resources (VLIW, SALU, Crossbars, Gateway), required by three solutions. As shown in Figure 9, the hardware resource usage of SDNIE is lower than P7 that executed in parallel mode, but higher than TurboNet. This moderate increase in resource utilization is a reasonable trade-off for the enhanced functional flexibility.

3) CPU-Tofino Collaboration Performance: We construct 10 TCP flows to fully utilize the 10G CPU-Tofino Eth. NIC, and verify the impact of fixed delay impairment with different parameters from 100us to 30ms on TCP throughput, as shown in Figure 10. Since P7 only relies on the Tofino recirculation mechanism to implement delay impairment, the theoretical maximum delay is 12.8ms under a 16MB buffer, but in fact, throughput has already dropped when 10ms. And throughput collapse occurs with the packet loss rate of up to 97% when the impairment is 20ms or above. This may be due to buffer overflow caused by multi-flow competition, which makes TCP unable to recover. We set the collaboration policy for SDNIE to forward packets to the CPU/RAM, ensuring 0 packet loss in any case.

4) Processing Performance: Figure 11 compares the processing performance of SDNIE and zMonkey, where we limit ourselves to a single-core CPU and 10GE. Since the packet forwarding occurs in the Tofino plane, SDNIE can always achieve line rate for all packet sizes, while zMonkey cannot process small packets at line rate, which means that the impairment of small packets may not achieve the desired effect. For 64-byte packets, packet throughput of zMonkey is 8.78Mpps, which is 1.22X smaller than SDNIE.

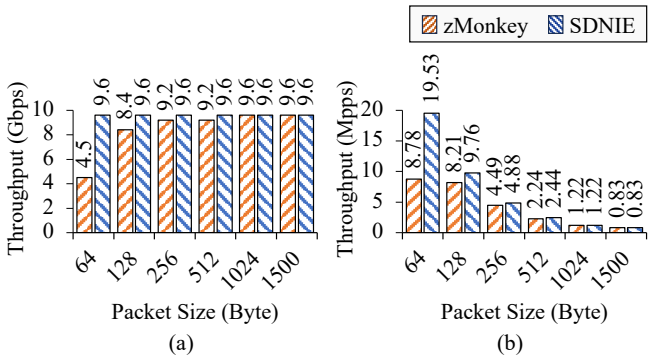


Fig. 11. TCP Throughput and Packet Rate vs Packet Size under Delay Impairment.

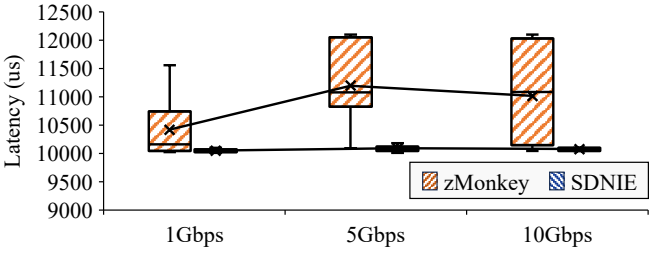


Fig. 12. Deviation of Delay Impairment.

We construct UDP traffic and added 10ms delay to all traffic. Due to processing performance limitations, Figure 12 shows that the zMonkey delay offset is significantly inaccurate. At the same time, the delay of SDNIE never exceeded 10180us, and the maximum error was only 1.8%.

VI. Conclusion and Future Work

This paper addresses how to repurpose programmable switches to create a network impairment emulator, thereby assigning it a novel role in network testing.

A. Conclusion

In this paper, we thoroughly analyzed the technical limitations of current commercial network impairment emulators and proposed the Software-Defined Network Impairment Emulation (SDNIE), a new scheme based on programmable switches, distinct from CPU- and FPGA-based solutions. This approach simplifies and refines the impairment configuration process through the intent-driven method, optimizes the deployment of impairment functions with the serial-parallel combination strategy, and supports complex impairment logic using the CPU-Tofino collaboration mechanism.

B. Future Work

Continuously meeting the needs of large-scale, high-precision network impairment emulation will be an important goal driving the development of SDNIE. Looking ahead, three future works for SDNIE research can be identified:

(1) Enhancing scalability and performance to handle even larger and more complex network topologies.

Wide area networks (WAN) and data center networks (DCN) are the two most widely used areas for network impairment emulators. With the development of WAN from land-based networks to space-air-ground integrated networks [46], and the continuous increase in the hierarchy and node scale of DCN, further improving the network impairment emulation capabilities of SDNIE under large-scale topologies [47] is an important research direction. This requires enhancing its scalability and performance, which can be solved by customizing buffer resources [48], optimizing compilation strategies and improving chip programmability [49]. More importantly, as Intel announces the cessation of Tofino development and open-sources P4 Software [50], the selection of hardware and software for SDNIE will be the first issue to be addressed. Perhaps Broadcom Jericho and NVIDIA Spectrum-X are potential options.

(2) Improving the integration of artificial intelligence to improve the functions of network impairment emulation.

In addition to using LLM [51] to simplify user interactions, AI can play a big role in the system design, function development, and use of NIEs. For example, (a) by learning human modifications to impairment intent configuration, the accuracy of intent-driven configuration can be continuously improved. (b) Using neural networks to capture network characteristics and optimize the expression of impairment functions and parameters, which is also a prerequisite for network impairment emulation [52]. (c) Using AI to assist in analyzing the relationship between different impairment functions and further optimize the computational efficiency of serial and parallel combinations.

(3) Expanding the interoperability capabilities to support more diverse hardware and network environments.

Interoperability is an important feature to ensure that SDNIE is compatible with different types of programmable switches [53], including newer models and models from different manufacturers, as well as with various network infrastructures such as 5G/6G, satellite communications, and IoT networks. This requires the development of a more flexible and adaptable software framework that can integrate with heterogeneous hardware through standardized interfaces and protocols [54]. In addition, this involves supporting multiple network protocol stacks. By extending these capabilities, SDNIE can become a more versatile tool that can adapt to the evolving technology environment and promote more comprehensive and realistic impairment emulation.

References

[1] L. Angrisani and C. Narduzzi, "Testing Communication and Computer Networks: An Overview," *IEEE Instrumentation & Measurement Magazine*, vol. 11, no. 5, pp. 12–24, 2008.

[2] K. Wehrle, M. Güneş, and J. Gross, *Modeling and Tools for Network Simulation*. Springer, 2010.

[3] Z. Xiang, S. Pandi, J. Cabrera, F. Granelli, P. Seeling, and F. H. P. Fitzek, "An Open Source Testbed for Virtualized Communication Networks," *IEEE Communications Magazine*, vol. 59, no. 2, pp. 77–83, 2021.

[4] T. Huang, F. R. Yu, C. Zhang, J. Liu, J. Zhang, and Y. Liu, "A Survey on Large-scale Software Defined Networking (SDN) Testbeds: Approaches and Challenges," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 891–917, 2016.

[5] ns 3 Project, "ns-3: Network Simulator 3," 2025, [Online; visited May-20-2025]. [Online]. Available: https://www.nsnam.org/

[6] O. Project, "OMNeT++: Discrete Event Simulator," 2025, [Online; visited May-20-2025]. [Online]. Available: https://omnetpp.org/

[7] M. Project, "Mininet: An Instant Virtual Network on your Laptop (or other PC)," 2025, [Online; visited May-20-2025]. [Online]. Available: https://mininet.org/

[8] K. Gao, L. Chen, D. Li, V. Liu, X. Wang, R. Zhang, and L. Lu, "DONS: Fast and Affordable Discrete Event Network Simulation with Automatic Parallelization," in *Proceedings of SIGCOMM'23*, New York, NY, USA, 2023, pp. 167–181.

[9] L. Foundation, "netem," 2025, [Online; visited May-20-2025]. [Online]. Available: https://wiki.linuxfoundation.org/networking/netem

[10] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, "CORE: A real-time network emulator," in *Proceedings of MILCOM'08*. San Diego, CA, USA: IEEE, 2008, pp. 1–7.

[11] Spirent, "Network Impairment and Emulation Testing," 2025, [Online; visited May-20-2025]. [Online]. Available: https://www.spirent.com/products/network-impairment-emulation-testing

[12] GENI, "Global Environment for Network Innovations," 2025, [Online; visited May-20-2025]. [Online]. Available: https://www.geni.net/

[13] CERNET, "The Second Generation China Education and Research Network," 2025, [Online; visited May-20-2025]. [Online]. Available: https://www.cernet.com/ipv6ycernet2/index.html

[14] CENI, "China Future Internet Infrastructure," 2025, [Online; visited May-20-2025]. [Online]. Available: https://ceni.org.cn/

[15] Y. Fu, H. Zhao, X. Wang, H. Liu, and L. An, "State-of-the-art Survey on Network Behavior Emulation," *Journal of Software*, vol. 33, no. 1, pp. 274–296, 2022.

[16] Keysight, "Network Emulator," 2025, [Online; visited May-20-2025]. [Online]. Available: https://www.keysight.com/us/en/products/network-test/network-test-hardware/network-emulator-ii.html

[17] S. Kaur, K. Kumar, and N. Aggarwal, "A Review on P4-Programmable Data Planes: Architecture, Research Efforts, and Future Directions," *Computer Communications*, vol. 170, pp. 109–129, 2021.

[18] G. Nardini, G. Stea, A. Virdis, D. Sabella, and P. Thakkar, "Using Simu5G as A Realtime Network Emulator to Test MEC APPs in An End-to-end 5G Testbed," in *Proceedings of PIMRC'20*, London, UK, 2020, pp. 1–7.

[19] P.-W. Tsai, F. Piccialli, C.-W. Tsai, M.-Y. Luo, and C.-S. Yang, "Control Frameworks in Network Emulation Testbeds: A Survey," *Journal of Computational Science*, vol. 22, pp. 148–161, 2017.

[20] S. Amaro, M. Matos, and V. Schiavoni, "KOLLAPS: Decentralized and Efficient Network Emulation for Large-Scale Systems," *IEEE/ACM Transactions on Networking*, vol. 32, no. 1, pp. 1–16, 2024.

[21] J. H. Salim, D. Chatterjee, V. Nogueira, P. Tammela, T. Osinski, E. Haleplidis, B. Sambasivam, U. Gupta, K. Jain, and S. Sethuramapandian, "Introducing P4TC-A P4 implementation on Linux Kernel using Traffic Control," in *Proceedings of EuroP4'23*, Paris, France, 2023, pp. 25–32.

[22] zartbot, "zMonkey," 2025, [Online; visited May-20-2025]. [Online]. Available: https://github.com/zartbot/zmonkey

[23] HoloWAN, "HoloWAN," 2025, [Online; visited May-20-2025]. [Online]. Available: http://www.msytest.cn/en/index.html

[24] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming Protocol-independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[25] D. Scholz, H. Stubbe, S. Gallenmüller, and G. Carle, "Key Properties of Programmable Data Plane Targets," in *Proceedings of ITC'20*, Osaka, Japan, 2020, pp. 114–122.

[26] N. Anerousis, P. Chemouil, A. A. Lazar, N. Mihai, and S. B. Weinstein, "The Origin and Evolution of Open Programmable Networks and SDN," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1956–1971, 2021.

[27] A. T.-J. Akem, G. Fraysse, M. Fiore *et al.*, "Encrypted Traffic Classification at Line Rate in Programmable Switches with Machine Learning," in *Proceedings of NOMS'24*, Seoul, Korea, 2024, pp. 1–9.

[28] Y. Zhou, Z. Xi, D. Zhang, Y. Wang, J. Wang, M. Xu, and J. Wu, "HyperTester: High-performance network testing driven by programmable switches," in *Proceedings of CoNEXT'19*, Orlando, USA, 2019, pp. 30–43.

[29] F. G. Vogt, F. Rodriguez, F. G. Costa, M. C. Luielli, C. E. Rotenberg, G. Patra, and G. Pongracz, "P4 Replay (P4R): Reproducing Packet Traces and Stateful Connections at Line-Rate on Your P4-capable Hardware," in *Proceedings of SIGCOMM'24*, Sydney, Australia, 2024, pp. 122–124.

[30] L. Waind, G. Chen, Z. Hu, and D. Jin, "Comparative Analysis and Evaluation of P4-Based Network Emulation Testing Environments," in *Proceedings of SIGSIM-PADS'24*, Atlanta, USA, 2024, pp. 77–78.

[31] J. Cao, Y. Liu, Y. Zhou, L. He, and M. Xu, "Turbonet: Faithfully Emulating Networks with Programmable Switches," *IEEE/ACM Transactions on Networking*, vol. 30, no. 3, pp. 1395–1409, 2022.

[32] F. E. R. Cesen, F. G. Vogt, A. G. D. Castro, and C. E. Rothenberg, "Towards Multiple Pipelines Network Emulation with P7," in *Proceedings of NetSoft'23*, Madrid, Spain, 2023, pp. 290–292.

[33] M. Elbediwy, B. Pontikakis, J.-P. David, and Y. Savaria, "Enabling Rank-Based P4 Programmable Schedulers: Requirements, Implementation, and Evaluation on BMv2 Switches," *IEEE/ACM Transactions on Networking*, vol. 33, no. 1, pp. 1–12, 2024.

[34] D. Franco, E. O. Zaballa, M. Zang, A. Atutxa, J. Sasiain, A. Pruski, E. Rojas, M. Higuero, and E. Jacob, "A Comprehensive Latency Profiling Study of the Tofino P4 Programmable ASIC-based Hardware," *Computer Communications*, vol. 218, pp. 14–30, 2024.

[35] G. Lu, R. Miao, Y. Xiong, and C. Guo, "Using CPU as A Traffic Coprocessing Unit in Commodity Switches," in *Proceedings of HotSDN'12*, Helsinki, Finland, 2012, pp. 31–36.

[36] Z. Cui, S. Hou, L. Tian, Y. Wang, X. Yi, Y. Wang, P. Yi, and H. Chen, "P4-Ace: Resource-Efficient Optimization and Verification for Programmable Switches," in *Proceedings of SIGCOMM FMANO'24*, Sydney, Australia, 2024, pp. 8–13.

[37] A. G. Alcoz, C. Busse-Grawitz, E. Marty, and L. Vanbever, "Reducing P4 Language's Voluminosity using Higher-level Constructs," in *Proceedings of EuroP4'22*, Rome, Italy, 2022, pp. 19–25.

[38] D. D. Robin and J. I. Khan, "An Open-source P416 Compiler Backend for Reconfigurable Match-action Table Switches: Making Networking Innovation Accessible," *Computer Networks*, vol. 242, p. 110246, 2024.

[39] L. Tan, W. Su, W. Zhang, J. Lv, Z. Zhang, J. Miao, X. Liu, and N. Li, "In-band network telemetry: A survey," *Computer Networks*, vol. 186, p. 107763, 2021.

[40] M. Budiu and C. Dodd, "The P416 Programming Language," *SIGOPS Operating Systems Review*, vol. 51, no. 1, pp. 5–14, 2017.

[41] H. Harkous, M. Jarschel, M. He, R. Priest, and W. Kellerer, "Towards Understanding the Performance of P4 Programmable Hardware," in *Proceedings of ANCS'19*, 2019, pp. 1–6.

[42] S. Kodeswaran, M. T. Arashloo, P. Tammana, and J. Rexford, "Tracking P4 Program Execution in the Data Plane," in *Proceedings of SOSR'20*, San Jose, CA, USA, 2020, pp. 117–122.

[43] Micron, "HBM3E," 2025, [Online; visited May-20-2025]. [Online]. Available: https://www.micron.com/products/memory/hbm/hbm3e

[44] DPDK, "Data Plane Development Kit," 2025, [Online; visited May-20-2025]. [Online]. Available: https://core.dpdk.org/perf-reports

[45] AutoDL, "AutoDL," 2025, [Online; visited May-20-2025]. [Online]. Available: https://www.autodl.com/home

[46] Z. Lai, H. Li, and J. Li, "Starperf: Characterizing Network Performance for Emerging Mega-constellations," in *Proceedings of ICNP'20*, Madrid, Spain, 2020, pp. 1–11.

[47] Z. Chen, Z. Zhao, Z. Li, J. Shao, S. Liu, and Y. Xu, "SDT: A Low-cost and Topology-reconfigurable Testbed for Network Research," in *Proceedings of CLUSTER'23*, Santa Fe, NM, USA, 2023, pp. 343–353.

[48] M. Hogan, S. Landau-Feibish, M. T. Arashloo, J. Rexford, and D. Walker, "Modular Switch Programming under Resource Constraints," in *Proceedings of NSDI'22*, Renton, WA, USA, 2022, pp. 193–207.

[49] J. Liu, G. Zhao, H. Xu, B. Wang, P. Yang, C.-J. Chung, M. Chen, and X. Yang, "HifiCNet: High-Fidelity Cloud Network Validation Platform at Scale by Hybrid Architecture," in *Proceedings of ICNP'24*, Charleroi, Belgium, 2024, pp. 1–12.

[50] Intel, "P4," 2025, [Online; visited May-20-2025]. [Online]. Available: https://github.com/p4lang/open-p4studio

[51] D. Wu, X. Wang, Y. Qiao, Z. Wang, J. Jiang, S. Cui, and F. Wang, "Netllm: Adapting Large Language Models for Networking," in *Proceedings of SIGCOMM'24*, Sydney, Australia, 2024, pp. 661–678.

[52] F. Wilhelmi, M. Carrascosa, C. Cano, A. Jonsson, V. Ram, and B. Bellalta, "Usage of Network Simulators in Machine-learning-assisted 5G/6G Networks," *IEEE Wireless Communications*, vol. 28, no. 1, pp. 160–166, 2021.

[53] J. Gao, E. Zhai, H. H. Liu, R. Miao, Y. Zhou, B. Tian, C. Sun, D. Cai, M. Zhang, and M. Yu, "Lyra: A Cross-platform Language and Compiler for Data Plane Programming on Heterogeneous ASICs," in *Proceedings of SIGCOMM'20*, Virtual Event, USA, 2020, pp. 435–450.

[54] L. Tan, W. Su, W. Zhang, H. Shi, J. Miao, and P. Manzanares-Lopez, "A packet loss monitoring system for in-band network telemetry: Detection, localization, diagnosis and recovery," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4151–4168, 2021.

Lizhuang Tan is currently an Associate Professor with Shandong Provincial Key Laboratory of Computer Networks, Shandong Computer Science Center (National Supercomputer Center in Jinan), Qilu University of Technology (Shandong Academy of Sciences). He received his Ph.D. degree from School of Electronic and Information Engineering, Beijing Jiaotong University in 2022. He has published more than 20 journal or conference papers, such as IEEE TCC, IEEE TNSM, ELSEVIER CN, APNOMS, etc. His research interests include network measurement, management and optimization.

Nguyen Van Tu received the B.Sc. degree in electronics and communication from the Hanoi University of Science and Technology, Vietnam, in 2015, the M.Sc. and Ph.D. degree in computer science from Pohang University of Science and Technology, South Korea, in 2018 and 2025. His research is focused on high-performance networks, and programmable networks.

Xinhang Wang received his M.Sc. degree and B.Sc. degree from Shandong Computer Science Center (National Supercomputer Center in Ji'nan), Qilu University of Technology (Shandong Academy of Sciences), Jinan, China in 2022 and 2025. His research interests include Software Defined Networking and Programmable Networks.

Peiying Zhang is currently an Professor with the College of Computer Science and Technology, China University of Petroleum (East China). He received his Ph.D. in the School of Information and Communication Engineering at University of Beijing University of Posts and Telecommunications in 2019. He has published 100 IEEE/ACM Trans./Journal/Magazine papers, such as IEEE TII, IEEE T-ITS, IEEE TVT, IEEE TNSE, IEEE TNSM, IEEE TETC, IEEE Network and etc. His research interests include semantic computing, future internet architecture, network virtualization, and artificial intelligence for networking.

James Won-Ki Hong received his HBSc and MSc degrees in Computer Science from the University of Western Ontario, Canada, in 1983 and 1985, respectively, and the PhD degree in Computer Science from the University of Waterloo, Canada, in 1991. He is Professor in the Department of Computer Science and Engineering at Pohang University of Science and Technology (POSTECH), Pohang, Korea. He had worked as the Chief Technology Officer and Senior Executive Vice President for KT (Korea Telecom), the largest telecommunications company in Korea from March 2012 to February 2014, where he was responsible for leading the R&D effort of KT and its subsidiary companies. He was Chairman of National Intelligence Communication Enterprise Association and Chairman of ICT Standardization Committee in Korea. He cofounded and is currently served as Executive Director of SDN/NFV Forum in Korea. He had served as the Head of Department of Computer Science and Engineering, Dean of Graduate School of Information Technology, Director of POSTECH Information Research Labs, and Head of the Division of IT Convergence Engineering at POSTECH. He has served as Chairman of the IEEE Communications Society, Committee on Network Operations and Management. He has also served IEEE ComSoc Director of Online Content (2004–2005, 2010–2011). He is the Editor-in-Chief of International Journal on Network Management (IJNM), IEEE ComSoc Technology News, and KNOM Review Journal. He is the General Chair of NOMS'24, ICBC'19, NetSoft'16 and APNOMS'06. He is an editorial board member of IEEE Transactions on Network and Service Management, Journal of Network and Systems Management and Journal of Communications and Networks. His research interests include network innovation, such as software-defined networking and network function virtualization, cloud computing, mobile services, IPTV, ICT convergence technologies (e.g., Smart Home, Smart Energy, and Health care), and Internet of Things.