CombNE: A Combined Network Emulator based on Programmable Switch

Xinhang Wang[1,2,3], Lizhuang Tan[1,2], Huiling Shi[1,2], Wei Zhang[1,2*]
[1]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
[2]Shandong Provincial Key Laboratory of Computing Power Internet and Service Computing,
Shandong Fundamental Research Center for Computer Science, Jinan, China
Email: 10431220360@stu.qlu.edu.cn, {tanlzh, shihl, wzhang}@sdas.org

***Abstract*—Network emulator is an equipment used in the field of computer networking to replicate and simulate real-world network conditions, especially poor-quality network conditions accompanied by various damages, in a controlled environment. It plays a crucial role in the development, testing, and validation of various new network-related technologies, protocols, and applications. Compared with simulation and test-bed methods, network emulation possesses the advantages of accuracy and cost-efficiency. However, legacy network emulation methods are implemented serially, which are typically restricted in efficiency and waste computing resources. In this paper, we propose a combined network emulator, CombNE. To implement this emulator, we consider P4 programmable switches as a desirable option. CombNE consists of three logical components. First, CombNE provides a policy specification scheme to intuitively describe operator's intents. Secondly, the CombNE parallelizer intelligently identifies the dependencies between network damages, automatically determines parallelization and generates a combination strategy. Third, CombNE will generate optimized P4 files and flow table information based on the combination strategy and deploy them to P4 programmable switches. Finally, we evaluated the performance and resources of CombNE.**

***Index Terms*—network emulation, programmable switch, parallelization**

I. Introduction

Since the birth of the Internet, human or accidental conditions such as network throughput limitation, data packet loss, data transmission delay and delay jitter have closely accompanied the development of the Internet. Today's Internet is a huge system that is crowded, busy and complex [1]. Different network services and applications share the same network infrastructure to send and receive traffic in a competitive manner. In the realm of network infrastructure, data traverses diverse network nodes, devices and protocols, each exhibiting distinct forwarding capabilities and transmission delays [2]. To emulate and characterize real-world networks [3], network evaluators have distilled a range of parameters from actual networks to describe the extent of network link characteristic. These parameters include packet loss, latency, bandwidth, jitter, reordering, and other pertinent factors. Due to their adverse impact on regular links, we refer to these parameters as network damages. Therefore, how to faithfully mimic networks, especially the regular state and sudden damage existing in the actual network, becomes an significant issue [4].

Existing network emulation methods can be categorized into two types: software and hardware. Software-based emulation methods leverage software tools or emulators that run on general-purpose hardware to emulate network behavior. Examples include OMNeT++ [5] and GNS3 [6]. While these methods offer flexibility and scalability for larger networks, their models often lack the full representation of real networks in terms of both functionality and performance. On the other hand, hardware-based methods utilize specialized hardware to emulate network damages, these methods offer higher performance and accuracy. Emulators like Cisco's VIRL/CML [7] or various FPGA-based solutions [8] fall into this category. They are customizable like simulators and show great functional fidelity. However, the size of experiments is bounded by the available CPU or FPGA cycles and memory, making it

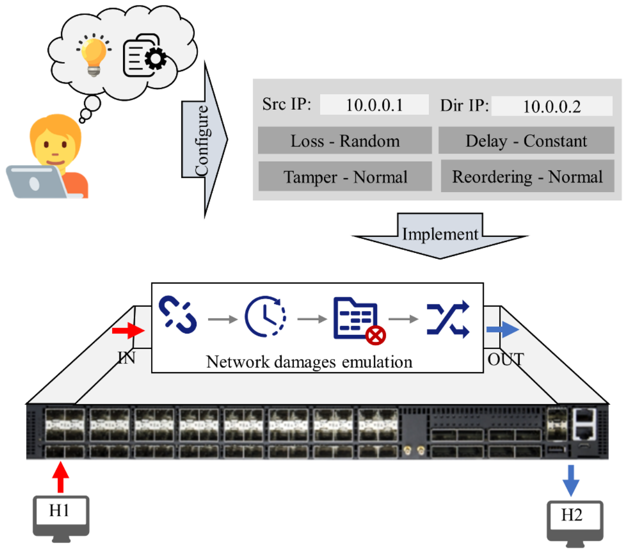
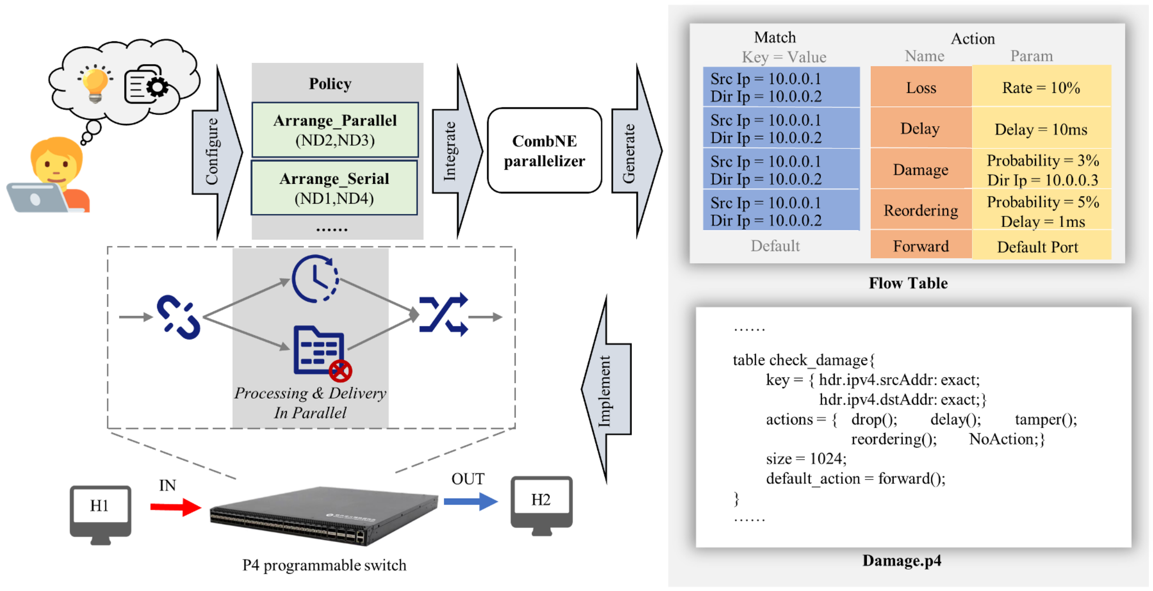


Fig. 1: The process of traditional serial network damage implements.



Match	Action	
Key = Value	Name	Param
Src Ip = 10.0.0.1 Dir Ip = 10.0.0.2	Loss	Rate = 10%
Src Ip = 10.0.0.1 Dir Ip = 10.0.0.2	Delay	Delay = 10ms
Src Ip = 10.0.0.1 Dir Ip = 10.0.0.2	Damage	Probability = 3% Dir Ip = 10.0.0.3
Src Ip = 10.0.0.1 Dir Ip = 10.0.0.2	Reordering	Probability = 5% Delay = 1ms
Default	Forward	Default Port

```
......

table check_damage{
      key = { hdr.ipv4.srcAddr: exact;
                hdr.ipv4.dstAddr: exact;}
      actions = {   drop();       delay();       tamper();
                    reordering();      NoAction;}
      size = 1024;
      default_action = forward();
}
......
```

Fig. 2: CombNE framework supporting combined network damages.

challenging to provide precise performance results, especially when emulating networks operating at Gbps levels.

Some research efforts try to emulate large networks within one real switch. BNV [9] leverages multiple OpenFlow [10] switches to emulate networks.However, due to the inflexibility of OpenFlow switches, BNV is fundamentally limited and fails in emulating large topologies and network performance metrics, such as link delay. TurboNet [4] [11] is a network emulator which leverages the power of programmable switches [12] [13] to faithfully mimic functionality, scale, and performance of production networks. And P7 [14] emulate certain network link characteristics and instantiate a network topology to run line-rate traffic using a single physical P4 switch. These solutions provide new ideas for implementing network damages on a switch, but they all continue the traditional network emulator's implementation of serial execution of different network damages, lacking further optimization of performance. So, it is possible to realize emulating certain network damages in a parallel manner.As we just mentioned, whether these solutions are hardware-based or software-based their implementation is usually executed serially, which limits the efficiency and accuracy of network damages.

A closer look into the network damages implements shows that some damages share no dependency and could work in parallel. For example, in the serial network damage implements shown in Fig. 1, the Tamper only needs to change certain fields of the data packet, such as source IP, destination IP, etc., while Delay only needs to add a specific timestamp to the data packet, causing it to be sent after the qualifying time is met. Therefore, as shown in the service graph in Fig. 2, we could send traffic into the Tamper and the Delay simultaneously, and merge them at the output to make them into packets that contain timestamps and that specific fields have been tampered with, achieving the same result as sequential execution. Assuming that the execution time required for all damages is the same (except for Delay), this way could bring a theoretical latency reduction by 25%. In addition, the existing network damage emulation have the following limitations:

As we just mentioned, whether these solutions are hardware-based or software-based their implementation is usually executed serially, which limits the efficiency and accuracy of network damages. A closer look into the network damages implements shows that some damages share no dependency and could work in parallel. For example, in the serial network damage implements shown in Fig. 1, the Tamper only needs to change certain fields of the data packet, such as source IP, destination IP, etc., while Delay only needs to add a specific timestamp to the data packet, causing it to be sent after the qualifying time is met. Therefore, as shown in the service graph in Fig. 2, we could send traffic into the Tamper and the Delay simultaneously, and merge them at the output to make them into packets that contain timestamps and that specific fields have been tampered with, achieving the same result as sequential execution. This way could bring a theoretical latency reduction by 25%. In addition, the existing network damage emulation have the following limitations:

1) There are logical conflicts between different network damages. For example, the packet loss will change the number of data packets, and if it precedes the packet tampering module, it will affect the damage accuracy of the latter.
2) Wrong serial execution network damage leads to a waste of device hardware resources. For example, the implementation of packet loss logic means that other

damages do not need to occupy additional computing resources.

3) Network damages usually have time constraints. Serial execution of the network damages cannot meet the delay limit of packet-by-packet forwarding under high throughput.

Therefore, we propose CombNE, a combined network emulator, that innovatively embraces network damages parallelism to reduce network emulation latency. And we implement it into Tofino programmable switch. As shown in Fig. 2, CombNE consists of three logical components including a policy specification scheme, CombNE parallelizer, and P4 programmable switch. Our main contributions are:

- We present the motivation and design challenges of introducing network damages parallelism into emulation, and propose the CombNE framework that exploits network damages parallelism to improve emulation performance and save hardware resource usage.
- We provide a policy specification scheme for intuitively representing sequential or parallel network damage requirement of network operators to improve the combined emulation effect.
- We design CombNE parallelizer that can intelligently identify the dependencies between network damages, automatically determines parallelization and generate P4 files and flow table information containing the combination strategy, and deliver them to P4 programmable switch.
- We optimize the P4 code so that the P4 compiler can perform network damages according to the ideal serial-parallel combination.
- We implement CombNE based on Tofino programmable switch that support P4 programming. And in the evaluation section, we compared CombNE with other emulators such as Turbonet and P7 and compared different serial-parallel combinations.

II. Background and Motivation

This section first describes the background and motivation for adopting combined network emulator. We then introduce design challenges of CombNE.

Network emulator and network damages: It helps testers and network engineers understand how the network performs and copes with different environments by introducing various network damages. Table 1 shows Different types of network damages include packet loss, delay, bandwidth limitations, jitter, out-of-order, etc. Loss can simulate the loss of data packets in the network, and Delay is used to simulate the delay of data packets during transmission. Bandwidth allows the evaluation of performance under network congestion or limited bandwidth, Jitter simulates latency fluctuations in the network, and Reordering is used to simulate changes in the order in which packets arrive at their destination. [15] [16] The network emulator enables a comprehensive assessment of the robustness and performance of network devices and applications under different adverse conditions. But the main problem with network emulator is that it is difficult to perfectly balance functions and performance. Traditional hardware network emulators have developed from the needs of adapting to operators and communication equipment manufacturers, and are more focused on ensuring test throughput and simulation accuracy. There is a lack of exploration of the diversity of network types and network scenarios. There are also some software network emulators on the market. During the development process of some simulation software, some network scenarios and network type templates are built in using network detection results, which are more in line with the usage scenarios of Internet companies. And we compared three different commercial network emulators as shown in Table 2.

TABLE I: Network Damage Types Table

Type	Description
Loss	Simulates loss of data packets
Delay	Simulates delay in packet transmission
Bandwidth	Restricts network bandwidth
Jitter	Simulates fluctuation in delay
Reordering	Simulates changes in packet arrival order

In addition, current network emulators typically execute serially, meaning they perform network damages one at a time in a certain order. This serial execution method may result in inefficient emulation and waste computing resources. Because some network damages may be independent of each other, in the current context they fail to take full advantage of parallel processing.

Network Damages Emulation in Programmable Switches: Programmable switches conduct re-configurable pac-ket processing on data planes with high bandwidth. The flexibility of programmable switches enables agile customization of emulated networks, allowing users to validate their design in various network environments. With the programmability that P4 brings to networking researchers and the capabilities of new generation P4 hardware supporting the PSA (Portable Switch Architecture) [17] and TNA (Tofino Native Architecture) [18], it is possible to emulate certain network damages using a single physical P4 switch (e.g., Tofino). TurboNet and P7 are good examples.

III. Policy Design

The traditional network emulator will execute in a fixed order according to the damages configured by the network operators, which cannot well reflect the true intention of the network operators, nor can it support combination well. Therefore, CombNE provides a policy specification scheme that allows operators to express their intentions more clearly and contributes to the subsequent generation of combined policies. The main policy rules include the following:

Arrange_Serial (damage_1, damage_2, damage_3, . . .): This rule means that the two network damages are implemented in the order of damage_1 first, then damage_2, and

TABLE II: Network Emulators Feature Comparison Table

Emulators		Spirent	Apple Network Link Conditioner	Linux Traffic Control
Type		Hardware	Software	Software
Supported Platforms		Unlimited	MacOS, iOS	Unlimited
Bandwidth	Bit rate control	Supported	Supported	Supported
	Queue depth	Not supported	Not supported	Supported
	Burst traffic	Not supported	Not supported	Supported
Loss Packet	Supported or Not	Supported	Supported	Supported
	Type	Random Cycle	Random	Random 4-state Markov Gilbert-Elliot
Constant Delay		Supported	Supported	Supported
Jitter	Supported or Not	Supported(With reordering)	Not supported	Supported(Configurable)
	Distribution	Linear Range Gaussian	Not supported	Evenly Normal Pareto
Burst		Supported	Not supported	Can be achieved through automation
Preset scenes		No	Yes	No
Automation Integration		Support Restful API	Controllable via AppleScript	Linux command line, scripting languages
Command control accuracy		10ms	1000ms	Depends on system performance

finally damage_3. For example, in the network damage implements shown in Fig. 1, the network operator can first send the traffic to Loss, then to Delay, third to Tamper, and finally to Reordering by specifying *Arrange_Serial (Loss, Delay, Tamper, Reordering)*. This rule type can be used to describe a sequential network damages composition intent. Therefore, this provides network operators with a richer combination of damage intent while ensuring the compatibility of CombNE to support sequential network damages. It is worth noting that this policy will directly be used in the construction of the final combined strategy and CombNE parallelizer could not explore parallelism opportunities for the network damages in Sequence rules.

Arrange_Parallel (damage_1, damage_2): This rule describes the parallel execution of two network damages. Network operators can execute corresponding network damages in parallel by specifying *Arrange_Parallel (damage_1, damage_2)*. If operators do not express parallel intention by this rule, CombNE will automatically collect all other network damages except serial arrange and create this parallel rule for them. Before generating the final combined strategy, the CombNE parallelizer will collect these rules and make parallelization decisions based on whether there is a dependency between the two network damages. If the two damages are non-parallelizable, they are executed in the order of their respective priorities.

Prior (damage_1, damage_2): Conflict may arise between two network damages when network operators want to describe the intent of executing them in parallel. For instance, when there is Loss, other damage operations will conflict with the issue of whether to lose packets. Therefore, network operators can specify the priority of two network damages by *Prior (damage_1, damage_2)* rule. If operators do not specify any priority rules, CombNE assigns default priorities to non-parallelizable network damages, where the default priority order is: Loss $>$ Delay = Shaping $>$ Reordering $>$ Tamper $>$ Other.

First/Last (damage_1): Since Loss marks the drop field of the data packet, the marked data packet will be dropped at the egress. Therefore, damage Loss should be set as the first position to avoid other damages from operating on the marked data packets, thereby reducing resource waste. To achieve this intention, CombNE provides *First/Last (damage_1)* rules for operators to set the first/last damage. It should be noted that this rule is only allowed to be set once. Multiple *First/Last (damage_1)* rules will cause conflicts when CombNE generates the final combination strategy.

With above rules, network operators can define network damage intents by composing multiple rules into a policy to describe a network damages combination. (Fig. 2). Finally, the rules manually written by operators could possibly conflict with each other. For example, an operator could assign a network damage at different positions, i.e. *First (damage_1)* and *Last (damage_1)*. The challenges of policy conflict detection and resolution have been recognized and studied in [19]. We will refer to prior wisdom and leave them to our future work.

IV. Parallelizer Design

CombNE parallelizer takes the CombNE policies as input, identifies network damage dependencies, and automatically compiles policies into high performance combination strategy with parallel damages. This section will introduce each step in the combination strategy construction process in detail.

A. Network Damages Parallelism Analysis

As introduced above, for CombNE policies, we provide a *Arrange_Parallel* rule to make two network damages execute in parallel. For network damages that are not specified by the operator through policy, CombNE parallelizer will retrieve them and assign *Arrange_Parallel* rules to them. So, we can further explore their parallelism possibility.

When the CombNE parallelizer receives policies, it will first create a mapping table named Network Damage Action Map (Table 3) based on the current network damages, which is used to store the operation fields and operation information of each damage. The operation fields are header fields, and the operation information values include reading and writing. Through this table we can observe that the actions of different network damages may conflict with each other. For instance, the Delay and the Shape both add a timestamp field. If the operator inputs an *Arrange_Parallel (Delay, Shape)*, the parallelizer is challenged to identify the parallelism possibility of the two network damages for high performance.

To analyze whether two network damages are parallelizable, the most important point is whether the damages execution result after parallel execution is the same as that of sequential execution. By this principle, we give Network Damage Dependency Table (Table 4). This table includes the respective operation information of the two network damages and the dependency relationship between the operation information of the two damages. The T1 and T2 represent parallelizable situations, and the difference lies in whether the data packet needs to be copied. For example, suppose damage_1 reads the packet header, and damage_2 later modifies the same header field. To ensure that damage_1 reads the original header that has not been changed by damage_2, we could copy the packets and send two copies into damage_1 and damage_2 in parallel. This method can further optimize performance and reduce resource waste. F denote unparallelizable situations. For example, if damage_1 first writes a packet header and later damage_2 reads this header, the operator intends to transmit the modification of damage_1 to damage_2. Therefore, the two damages should work in sequence.

B. Network Damages Parallelism Identification

Based on above network damages parallelism analysis, we propose a network damages parallelism identification algorithm and the CombNE parallelizer will execute according to this algorithm.

TABLE III: Network Damage Action Map. (R for Read, W for Write, T for True, F for False,and ADD/RM for Add headers to or Remove headers from packets)

Opt Field / NDs	SIP	DIP	SPORT	DPORT	DROP	ADD/RM
Loss	R	R	R	R	T	F
Tamper	R/W	R/W	R/W	R/W	F	F
Reordering	R	R	R	R	F	F
Delay	R	R	R	R	F	T
Shape	R	R	R	R	F	T
...	...	...	...	...	...	...

TABLE IV: Network Damage Dependency Table. (T1 means that it can be parallelized and does not need to copy data packets, T2 means that it can be parallelized but needs to copy data packets, and F means that it cannot be parallelized.)

Damage2 / Damage1	Read	Write	Add/Rm	Drop
Read	T1	T2	T2	T1
Write	F	T2	T2	T1
Add/Rm	F	F	T2	T1
Drop	F	F	T2	T1

Network damages parallelism identification algorithm: The CombNE parallelizer maintains a Network Damage Action Map (NDAM, i.e., Table 3) and a Network Damage Dependency Table (NDDT, i.e., Table 4), and takes the *Arrange_Parallel* rule as input. The algorithm can determine whether two NDs can be parallelized without packet replication or with packet replication, or cannot be parallelized. After receiving a *Arrange_Parallel* rule as input, the CombNE parallelizer obtains all operations of the two network damages from the network damage action map (Table 3), and then obtains all action pairs in the two network damages (such as the read operation of the first damage and the write operation of the second damage together constitute an action pair). Secondly, the CombNE parallelizer determines whether the two network damages can be parallelized according to the network damage dependency table (Table 4). For the read-write or write-write case, we need to further determine whether the two actions operate on the same field. If the two NDs can be parallelized by packet replication, the conflicting operations need to be recorded. Finally, the algorithm generates outputs of whether the two NDs are parallelizable (p) and the possible conflicting operations (ca), and inserts p into the *Arrange_Parallel* rule. The existence of conflicting operations indicates that packet duplication is required. Through this algorithm, the CombNE parallelizer adjusts the policy according to the p field in the

Algorithm 1 ND Parallelism Identification

```
1: Input: $Arrange\_Parallel$(ND1, ND2)
2: Output: $Arrange\_Parallel$(ND1, ND2) with Parallelizable $p$, Conflicting actions $ca$
3: $actions1 \leftarrow$ gatAction(NDAM, ND1)
4: $actions2 \leftarrow$ gatAction(NDAM, ND2)
5: $p =$ TRUE
6: $ca =$ NULL
7: for $(a1, a2) \in (actions1, action2)$ do
8:     if $(a1, a2) = (\text{read}, \text{write})$ or $(\text{write}, \text{write})$ then
9:         if $(a1, a2)$ operate on the same field then
10:            $ca$.extend$(a1, a2)$
11:            continue
12:        end if
13:    end if
14:    switch (fetchParallelism(NDDT, (a1, a2)))
15:        case NOT_PARALLELIZABLE:
16:            $p =$ FALSE
17:            return
18:        case PARALLELIZABLE_NO_COPY:
19:            continue
20:        case PARALLELIZABLE_WITH_COPY:
21:            $ca$.extend$(a1, a2)$
22:    end switch
23: end for
24: $Arrange\_Parallel$(ND1, ND2) $\leftarrow p$
```

Arrange_Parallel rule. Those rules that are judged as true will be regarded as policies and added to the generation of the final combined policy, while those rules that are false will be converted to *Arrange_Serial* policies according to their damage priority.

C. Combination Strategy Construction

The CombNE parallelizer will go through three steps to construct combination strategy based on policies. First, the policies will be transformed into storage states. Then, the parallelizer will compile the storage states into independent separation strategies. Finally, these separate strategies will be constructed a final combination strategy.

Transforming Policies into Storage States. We design three types of storage states to store policies, as shown in Fig. 3. For *Arrange_Parallel* and *Prior* rules, we implement the parallelism identification algorithm to check whether the NDs (Network Damages) can be parallelized and attach priority to them. The *Arrange_Parallel* and *Prior* rules are finally transformed into the representation shown on the left, which reveals the relationship between two NDs. For *First/Last* rules, we maintain the ND type and its first/last position in the middle representation block, which records the placement of a single ND. For *Arrange_Serial* rules, as shown on the left, we use the Map type to store multiple NDs executed sequentially, and their positions and damage types correspond to the keys and values in the map. **Compiling Storage States into Separate Strategies.** After transforming policies into storage states, we will construct separate strategies based on them. First the CombNE parallelizer will compile the storage states transformed by *Arrange_Parallel* and *Prior* rules. This will result in two different outcomes as shown in ND3 and ND4 in Table 3. If the parameter is_parallelizable is true, ND3 and ND4 will be combined in parallel. Otherwise, they will be combined in series according to their priority. Second, the storage states of *First/Last* rules will be compiled into the separate strategies directly connected to the output. Finally, the storage states of *Arrange_Serial* rules will be compiled into the separate strategies for internal network damages to be connected in the original order.

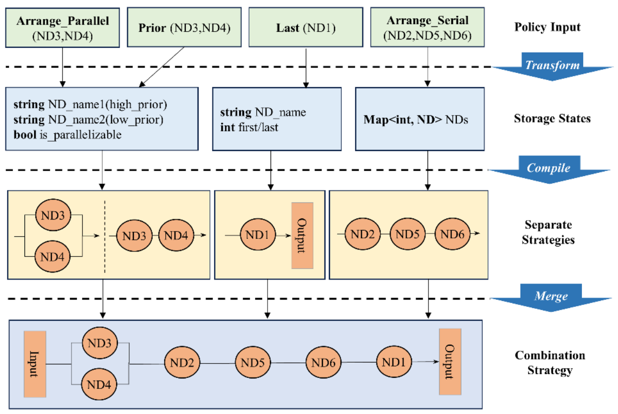


Fig. 3: Combination Strategy Construction Workflow

Merging Separate Strategies into Combination Strategy. The principle of merging separate strategies is based on their priorities. The separate strategies assigned by *First/Last* rules are first placed in the head/tail of the final strategy. For parallelizable network damages, we set the priority of its separate strategy to the priority of the higher-priority network damage. Then we will compare the priority of the parallel separate strategy with the first and last network damages in the serial separate strategy and merge based on the comparison results. Finally, the separate strategies at the input and output are merged to generate a final combination strategy.

V. Implementation in P4

While P4 code appears to execute in a sequential manner, the implementation of P4 in Tofino supports parallelism. This means that multiple operations can be executed simultaneously, with conditional statements being able to execute in parallel. Although the P4 compiler will automatically generate corresponding parallel execution code based on the structure and requirements of the program to improve processing efficiency. However, it cannot well support the serial and parallel combination of specific impairments. This section proposes the damage control block programming interface and an optimization method for P4 code.

A. Damage Control Block Programming Interface

CombNE parallelizer will generate P4 code and flow table. We provide a damage control block programming interface in the parallelizer to convert user damage intention into damage function in P4, and it hides the platform-specific metadata by providing a simple API with only one argument.

```
control XX_YY_damage (inout all_headers_t hd);
```

The "XX" in the function name corresponds to the type of network damage, and "YY" represents the different classifications of the damage. The argument hd is instantiated by the generic parser and provides all required protocol header fields, platform metadata and context metadata for the network damages. Code 1 shows an example implementation of a simple random packet loss damage. It can achieve random packet loss operation with an accuracy of 0.1%. The random number interval generated by the random method in P4 can only be within the nth power of 0 to 2^n, so the packet loss parameter needs to be converted first. After the random number is generated, it needs to be compared with the packet loss parameter. If it is smaller than the packet loss parameter, the drop operation will be performed. Otherwise, it will be forwarded normally.

```
control Loss_Random_damage(inout all_header_t hd){
  bit<10> loss_parameter =
      hd.meta.loss_parameter/1000*2^10;
  Random<bit<10>> random;
  action drop(){
    hd.meta.dropFlag = true;
  }
  action hit(Port_t port){
    hd.meta.egress_port = port;
  }
  table forward{
    key = {hd.ethernet.dst_addr:exact;}
    actions = {hit;}
    const default_action = hit();
  }
  apply{
    if(random < loss_parameter){
      drop();
    }
    forward.apply();
  }
}
```

Code 1: An example of a simple random packet loss damage using CombNE's programming interface.

B. Serial-Parallel Combination in P4

CombNE employs various methodologies to orchestrate Network Damages and formulate a unified, multi-pipeline P4 program suitable for compilation and deployment onto physical pipelines. In an effort to optimize resource utilization, CombNE consolidates multiple damages onto a single pipelet whenever hardware constraints permit. The determination of whether two damages can share a common pipelet necessitates a comprehensive understanding of the hardware resource consumption associated with each damage. This pertinent information is typically provided by the P4 compiler, which furnishes detailed resource consumption metrics such as MAU stages, SRAMs, and TCAMs for a given P4 program [20].

Network damages can be composed either sequentially or parallelly to share the pipelet in different ways. Code 2 shows the implementations of these two composition approaches wrapped in an ingress processing block. The first implements parallel calls with either packet loss or constant latency, but not both at the same time. After the packet passes a damage, CombNE will check the header to determine the type of damage to be performed next. In the second implementation, CombNE is called sequentially with random packet loss and constant delay, and then performs similar checks.

The two composition approaches offer an efficiency and feasibility trade-off for network damages placements. Parallel composition allows multiple damages to share the same MAUs, and thus can pack more damages on a pipelet. However, since packets can only traverse one branch on a pipelet, transitions from one branch to another require at least one resubmission or one recirculation.

On the contrary, sequential composition places multiple damages back-to-back on a pipelet and has no extra transition cost among those damages. However, in sequential composition, multiple network damages may access the same data fields in argument hd and thus incur different types of dependencies, e.g., match, action, or successor dependencies. And sequential composition enforces the P4 compiler to place the network damages in separate MAU stages, which may fail if the pipeline does not have enough stages.

```
//parallel operator
control MyIngress(inout all_header_t hd, inout
    common_metadata_t md, inout standard_metadata_t
    standard_metadata){
  apply{
    if(check_damage.apply().Loss_Random){
      Loss_Random_damage.apply(hd);
    }else if(check_damage.apply().Dalay_Constant){
      Dalay_Constant_damage.apply(hd);
    }
    check_nextND(hd);
  }
}
//sequential operator
control MyIngress(inout all_header_t hd, inout
    common_metadata_t md, inout standard_metadata_t
    standard_metadata){
  apply{
    if(check_Loss_Random_damage.apply()){
      Loss_Random_damage.apply(hd);
      check_nextND(hd);
    }
    if(check_Forward.apply()){
      Dalay_Constant_damage.apply(hd);
      check_nextND(hd);
    }
  }
}
```

Code 2: An example of the sequential and parallel compositions. Both invoke Loss of Random and Forward on the same pipelet.

VI. Evaluation

A. *Overview*

We deployed CombNE on a Barefoot Tofino switch.The switch is equipped with an Intel J1900 4-core 2.0GHz CPU and 8GiB memory, and has a total of 54 data interfaces and 2 management interfaces. The data interface includes 48 10GbE (SFP+) or 25GbE (SFP28) interfaces and 6 40GbE (QSFP+) or 100GbE (QSFP28) interfaces. The management interface includes a 10/100/1000M management interface. Network port and 1 standard UART console serial port.We will put two servers through the link to this switch for testing.The servers are equipped with two Intel(R) Xeon(R) E5-2690 v2 CPUs (3.00GHz,10 physical cores), 256G RAM and two 10G NICs. The servers run Linux kernel 4.4.0-31.

Next, We will evaluate CombNE from two aspects. First, we tested the latency of deploying multiple damages of the same type in different serial and parallel combinations in CombNE and compared them with TurboNet and P7. Then we combine multiple types of network impairments in different ways and evaluate them in three emulators.In order to facilitate unified evaluation results, the network damage standards we implement through P4 data plane programming are as follows:

Loss: Set a fixed probability of 2% for packet loss for a specific data flow.

Delay: Add a fixed delay of 100ns to packets that meet the conditions, implemented through recirculate in the PSA architecture.

Tamper: Modify a field of a data packet.

Reordering: Each data packet changes position with the next data packet with a 10% probability.

B. *Network Damage Number on Performance*

To avoid read and write conflicts between multiple identical network damages, we selected the Tamper damage for testing. We set 1 to 5 Tamper in hybrid(serial-parallel) combinations in three Emulators and tested the latency of packet forwarding.The results are shown in Fig. 4. Under the execution of serial and parallel combination, we can find that as the number of damages increases, the latency required by CombNE to process data packets is lower than TurboNet and P7. When the number of network damages is 5, CombNE's single packet forwarding delay is reduced by about 20ns compared to other emulators.

C. *Evaluation of Multiple Combinations*

In this part, we simulated and evaluated a set of series-parallel combination network damages, and tested its performance and resource usage.We selected four typical network damages: Loss, Delay, Tamper, and Reordering, and executed them in combination with serial and parallel combinations(Fig. 1) in CombNE and the other two Emulators.

Single Packet Forwarding Latency First, we measured the single packet forwarding latency of three different emulators executed in serial, parallel, and serial-parallel combinations. As shown in Fig. 5(a), we can see that the packet forwarding latency of the three emulators are similar under serial and

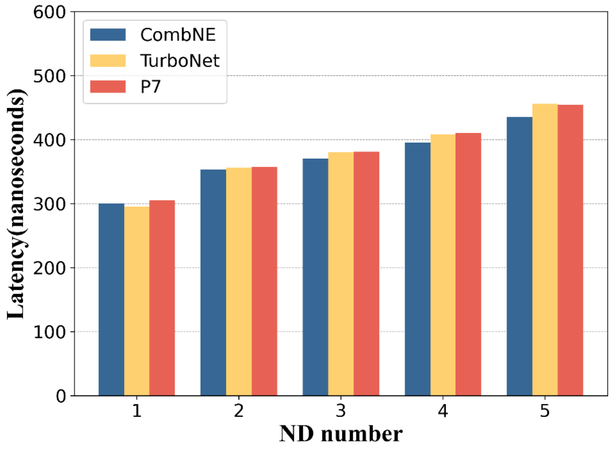


Fig. 4: Latency comparison of three emulators with multiple Tamper damages in hybrid combinations

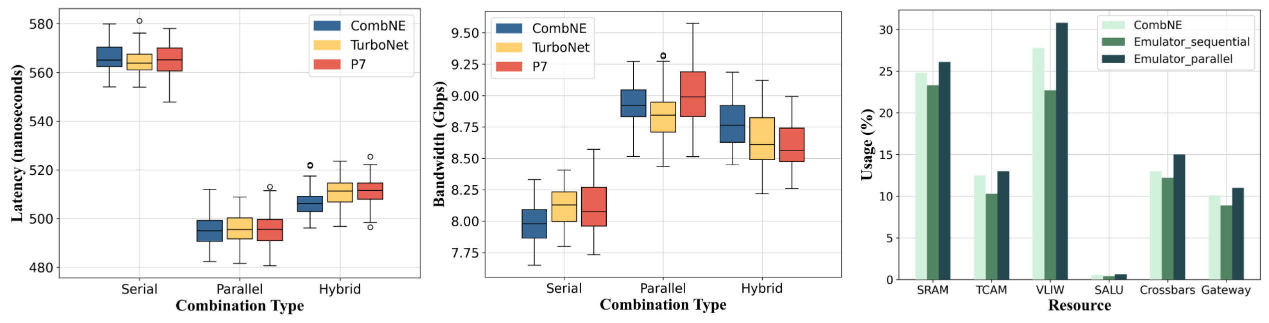
parallel execution. However, CombNE's latency under hybrid execution is significantly lower than TurboNet and P7.

Network Bandwidth Then we measured the bandwidth, connected the server to the switch, and used a 10G network. As shown in Fig. 5(b), since parallelization can use more computing and transmission resources, the bandwidth of parallelized execution is the highest. Serial execution limits the use of transmission resources, so it has the lowest bandwidth. When the three emulators are all executed in the Hybrid mode, the average bandwidth of CombNE is higher than the other two emulators.

ASIC Resource Usage We evaluated the hardware resources required by Comb-NE under hybrid execution, including memory resources (SRAM, TCAM) and computing resources (VLIM, SALU, Crossbars, Gateway), and compared with other emulators in serial and parallel situations. Since the network loss combination is similar in terms of serial and parallel resource usage in various emulators, we group it into one item here for comparison with the hybrid execution in CombNE. Fig. 5(c) shows the resource usage of CombNE and other emulators.We can see that the hardware resource usage of CombNE executed in hybrid mode is generally lower than that executed in parallel mode, but higher than that executed in serial mode.

VII. Conclusion

This paper presents CombNE, a combined network emulator that leverages a programmable switch to simulate various network damages. Meanwhile, CombNE can also combine its serial and parallel execution methods according to the optimal solution to improve performance and reduce resource overhead. We implemented a prototype in a P4 programmable switch and demonstrated its performance and overhead. As our future work, we will gradually implement more and more complex network damages, such as Markov four-state packet loss, normal distribution delay, etc., and conduct experimental



(a) Latency **(b) Bandwidth** **(c) Resource**

Fig. 5: Performance and resource comparison of three emulators with multiple damages in a serial-parallel combination

verification of more serial and parallel combinations of those damages.

Acknowledgment

This work was supported in part by the China University Research Innovation Fund under Grant No.2023IT207, the Shandong Provincial Natural Science Foundation under Grant No.ZR2023LZH011, No.ZR2023QF025 and No.ZR2022LZH015, the Taishan Scholar Program of Shandong Province in China under Grant No.TSQN202306258 and No.TSQN202312230.

References

[1] A. K. Tyagi, G. Rekha, and N. Sreenath, "Beyond the hype: Internet of things concepts, security and privacy concerns," in *Advances in Decision Sciences, Image Processing, Security and Computer Vision: International Conference on Emerging Trends in Engineering (ICETE), Vol. 1*. Springer, 2020, pp. 393–407.

[2] K. Alwasel, D. N. Jha, E. Hernandez, D. Puthal, M. Barika, B. Varghese, S. K. Garg, P. James, A. Zomaya, G. Morgan *et al.*, "Iotsim-sdwan: A simulation framework for interconnecting distributed datacenters over software-defined wide area network (sd-wan)," *Journal of Parallel and Distributed Computing*, vol. 143, pp. 17–35, 2020.

[3] 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 *2023 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, Oct. 2023.

[4] J. Cao, Y. Zhou, Y. Liu, M. Xu, and Y. Zhou, "Turbonet: Faithfully emulating networks with programmable switches," in *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, 2020, pp. 1–11.

[5] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, 2010.

[6] J. C. Neumann, *The book of GNS3: build virtual network labs using Cisco, Juniper, and more*. No Starch Press, 2015.

[7] B. Choi, *Python Network Automation Labs: Combining and Completing the Cisco IOS Upgrade Application*. Springer, 2021.

[8] M. E. Iranian, M. Mohseni, S. Aghili, A. Parizad, H. R. Baghaee, and J. M. Guerrero, "Real-time fpga-based hil emulator of power electronics controllers using ni pxi for dfig studies," *IEEE Journal of Emerging and Selected Topics in Power Electronics*, vol. 10, no. 2, pp. 2005–2019, 2020.

[9] P. G. Kannan, A. Soltani, M. C. Chan, and E.-C. Chang, "Bnv: Enabling scalable network experimentation through bare-metal network virtualization," in *11th USENIX Workshop on Cyber Security Experimentation and Test (CSET 18)*, 2018.

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[11] 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.

[12] S. Chole, A. Fingerhut, S. Ma, A. Sivaraman, S. Vargaftik, A. Berger, G. Mendelson, M. Alizadeh, S.-T. Chuang, I. Keslassy *et al.*, "drmt: Disaggregated programmable switching," in *SIGCOMM'17*, 2017, pp. 1–14.

[13] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 99–110, 2013.

[14] F. Rodriguez, F. G. Vogt, A. G. De Castro, M. F. Schwarz, and C. Rothenberg, "P4 programmable patch panel (p7) an instant 100g emulated network on your tofino-based pizza box," in *Proceedings of the SIGCOMM'22 Poster and Demo Sessions*, 2022, pp. 4–6.

[15] L. Nussbaum and O. Richard, "A comparative study of network link emulators," in *Communications and Networking Simulation Symposium (CNS'09)*, 2009.

[16] A. B. Downey, "Using pathchar to estimate internet link characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4, pp. 241–250, 1999.

[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] 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*, 2024.

[19] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang, "Pga: Using graphs to express and automatically reconcile network policies," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 29–42, 2015.

[20] L. Jose, L. Yan, G. Varghese, and N. McKeown, "Compiling packet programs to reconfigurable switches," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 103–115.