On-the-Fly Capture and Replay Mechanisms for Multi-Port Network Devices in Operational Networks

Ying-Dar Lin, *Fellow, IEEE*, Po-Ching Lin, Yu-An Lin, and Yuan-Cheng Lai

***Abstract*—Testing network devices in a live environment is desirable due to its reality. However, the defects are not reproducible, and the network connectivity will be broken if the device is down. For effective defect reproduction from real traffic, we design a new mechanism, which allows the device under test (DUT) to be automatically online/offline, and supports multi-port replay for multi-port network devices with an OpenFlow switch. The defect traces are captured when the DUT is online. When a DUT failure is detected, the DUT will be offline, and the defect-triggering traces will be replayed to identify the defect. For efficient replay, we keep only partial payloads in a reduced number of packets in the defect traces that are sufficient to trigger the defects. For defect identification, reduction based on a binary search algorithm is presented to deal with the defects caused by payload anomalies and by overloading. The downsizing ratios in the cases of payload anomalies and overloading are up to 98.8% and 96%, respectively. The minimum outage time of the failover during the DUT failure is obtained when the check interval is 1 second and the number of tolerable consecutive failures is 2.**

***Index Terms*—Network devices, failover, OpenFlow switch, multi-port replay, downsizing.**

I. Introduction

NETWORK device testing, which is intended to find out the defects of network devices and fix them before marketing, can improve the correctness and robustness of network devices. The traffic to trigger the defects of the network devices in a testbed can be classified into *artificial traffic* and *real traffic*. The former is generated with protocol modeling [1]–[3]. It is easy to produce test cases for specific protocols. Although the traffic can be generated from a realistic model, it still lacks the sufficient diversity and complexity in a real network. The latter is captured from an operational network [4]–[6]. It contains diverse network scenarios such as peer-to-peer (P2P), on-line games, and probably zero-day attacks. The scenarios are hard to be emulated by known modeling. Thus, real traffic is more effective to discover unexpected defects than artificial traffic.

Capturing the bulk traffic in an operational network and then replaying it to identify potential defects of network devices are usually impractical due to the huge volume of traffic. The cost of storage and the time of replaying the bulk traffic are also prohibitively high. Moreover, if a device under test (DUT) is deployed in an operational network for live testing, triggering its defects may disrupt the network connectivity. This problem is unacceptable to the users in the network. The defects are also not reproducible if the right packet traces causing the network disruption are not captured and replayed. Thus, an *on-the-fly mechanism to capture and replay the piece of problematic traffic triggering the defects of the DUT* is required to help reduce the storage requirement and debug with the traffic. This mechanism should also *minimize the outage time of the operational network due to the DUT failure*.

Replaying traffic to a DUT can be either *two-port* or *multi-port*, depending on the number of ports of the DUT. The two-port replay sends traffic from one port to the other through the DUT to reproduce defects. To the best of our knowledge, existing replay tools all support replay with only two ports. However, replaying with multiple ports on a DUT faces a particular problem: Some defects are triggered by the interaction of traffic from multiple ports. Replay tools working with only two ports are unable to reproduce the scenario. If a multi-port DUT is tested in the replay, *splitting the traffic to each port and synchronizing the ports during the replay* are a challenge.

We focus on *network failover*, *replay accuracy* and *debugging efficiency* to address the above issues. When the DUT is broken down due to a defect, the operational network will be disconnected. A mechanism is needed to allow the traffic to bypass the DUT as soon as possible to reduce the outage time of network disconnection. For the replay accuracy, since most replay tools are not supposed to work well for multi-port DUTs, a new replay mechanism is needed to split the traffic to the ports on such DUTs in a live environment. For debugging efficiency, the raw captured packet traces are usually huge. A method is needed to identify the minimum subset of the defect-triggering traffic, so that replaying it can reconstruct the status in which the defects occurred.

In this work, we present a mechanism, namely OFCR (i.e., On-The-Fly Capture and Replay), *to realize automatic multi-*

Manuscript received July 31, 2013; revised January 31, 2014. The associate editors coordinating the review of this manuscript and approving it for publication was L. Deri.

Y.-D. Lin is with the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, 300 (e-mail: ydlin@cs.nctu.edu.tw).

P.-C. Lin is with the Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan, 621 (e-mail: pclin@cs.ccu.edu.tw).

Y.-A. Lin was with the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, 300. He is now with Chunghwa Telecom Co., Taoyuan, Taiwan, 326 (e-mail: shower604@gmail.com).

Y.-C. Lai is with the Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan, 106 (e-mail: laiyc@cs.ntust.edu.tw).

Digital Object Identifier 10.1109/TNSM.2014.021714.130528

port network device testing in an operational network. When a DUT is in a normal state, we perform live testing for the DUT and buffer ongoing traffic simultaneously. While the DUT is broken down, we bypass the live traffic and shift the DUT to a replay testbed automatically. When the DUT is in the replay testbed, we replay the captured traffic before the breakdown to reproduce and identify the defect. These operations are implemented on an OF switch (short for OpenFlow switch; www.openflow.org/wp/documents) instead of on an expensive device such as a bypass switch and an aggregator switch. The contributions of this work are summarized as follows:

- We realize testing a multi-port network device in an operational network, the connectivity of which can be restored automatically as soon as possible to avoid disrupting normal network usage. The process is achieved by cleverly adopting an openflow switch.
- The captured packet traces can be replayed to multiple ports on the DUT like the scenario in the live environment, also by the use of an openflow switch.
- The minimum subset of the defect-triggering traffic can be identified and replayed to reconstruct the status in which the defects occurred.

The remainder of this paper is organized as follows. Section II presents the background and related work. Section III describes the terminology and assumption, and Section IV describes the architecture and implementation of OFCR. The experimental results and case study are presented in Section V. Finally, we conclude this work and discuss the future work in Section VI.

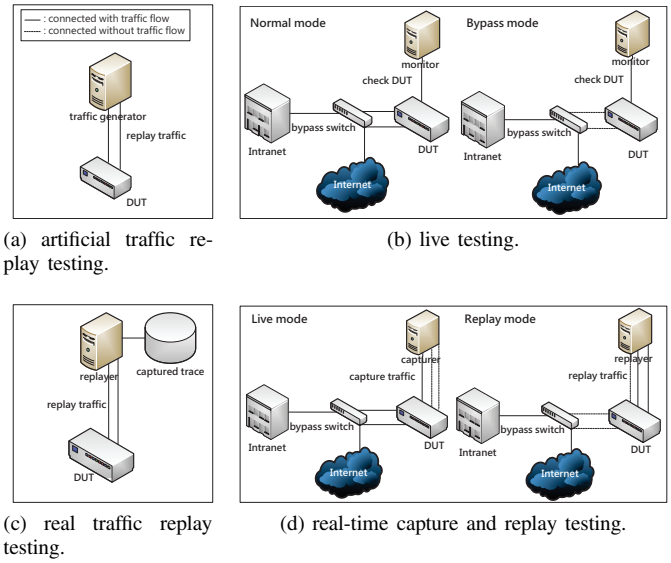
II. Background and Related Work

This section underlines the overview of network device testing, as well as the architecture of the OpenFlow network. Finally, the related work is discussed.

A. Network Device Testing

We categorize network device testing with network traffic into four types: (1) *artificial traffic replay testing*, (2) *live testing*, (3) *real traffic replay testing* and (4) *real-time capture and replay testing*. Figure 1 illustrates the testbeds of the four types of testing.

Figure 1(a) illustrates the artificial traffic replay testing. The DUT and the traffic generator are deployed in a closed testbed, in which the traffic generator sends the testing traffic to the DUT. The traffic is produced by traffic generators such as SmartBits (www.spirent.com), Codenomicon (www.codenomicon.com). The traffic generators can generate artificial traffic at very high speed, and thus have an indispensable advantage for stress testing to benchmark the performance of network devices. Some studies presented the methods to model realistic traffic for high-performance traffic generation [2], [3]. Malicious traffic from some known attack vectors can be also modeled and generated in [7]. Nonetheless, a fundamental limitation of this approach is that it is difficult for the generated traffic to fully reflect the great heterogeneity and rapid change of real traffic [8]. Thus, real traffic from an operational network is desired to complement this type



(a) artificial traffic replay testing.

(b) live testing.

(c) real traffic replay testing.

(d) real-time capture and replay testing.

Fig. 1. Testbeds for the four types of testing.

of testing, particularly for identifying unexpected defects of network devices.

Figure 1(b) illustrates the live testing, in which the DUT is deployed in a live environment. The monitor keeps checking the DUT, and the bypass switch allows the traffic to bypass the DUT to avoid network disruption when the DUT fails. The live testing is in the normal mode when the DUT works correctly. In this mode, the traffic passes through the bypass switch as if the DUT were connected to live network directly. When the monitor detects a failure, the testing will be switched to the bypass mode. In this mode, the ports connected to the DUT will be closed and the traffic will bypass the DUT.

Figure 1(c) illustrates the real traffic replay testing, which is also in a closed testbed. This testing replays the traffic captured from a live environment by a replay tool such as TCPreplay (tcpreplay.synfin.net). Figure 1(d) illustrates the real-time capture and replay testing. This testing has two modes: live mode and replay mode. The live mode is similar to that in the live testing, but the difference is that the capturer not only checks the DUT but also buffers the live traffic. When a failure is detected, the bypass switch will allow the traffic to bypass the DUT. The testing will be switched to the replay mode, and the replayer can replay the traces buffered earlier to reproduce the defects for debugging.

The four types of testing are compared in Table I. Despite the simplicity of traffic, the advantages of the artificial traffic replay testing are the ability of reproducing the defects and good customization of test cases. The latter is feasible because the characteristics of the traffic from most traffic generators is configurable. On the contrary, the live testing keeps the reality of traffic, but it may sacrifice the service quality for unexpected network disruption due to the DUT. Moreover, it cannot reproduce the defects and has poor customization of test cases.

The real traffic replay testing is a compromise between the first two types of testing. It uses captured real traffic to improve the diversity of traffic, and keeps the ability of defect reproduction, but the ability is worse than that in the live

TABLE I
COMPARISONS OF FOUR TYPES OF TESTING.

	Artificial traffic replay testing	Live testing	Real traffic replay testing	Real-time capture and replay testing
Traffic source	artificial traffic	live traffic	captured traffic	live traffic/ captured traffic
Network service quality	yes	no	yes	no
Traffic diversity	low	high	middle	middle-high
Defect reproduction	fully reproducible	no	partially reproducible	mostly reproducible
Test case customization	high	low	middle	low

testing because of the limitation of replay tools and replay scenarios. It does not affect the service quality because of a closed testbed. The customization is also better than that in the live testing because the collected traces can be categorized into several groups for testing. The real-time capture and replay testing has higher traffic diversity than the real traffic replay testing because it deploys the DUT in a live environment, and it has better defect reproduction because the replay scenario is closer to that in the real deployment. The tradeoff is the network disconnection during switching the modes. Because the test cases depend on the live traffic, its customization is as low as the live testing.

B. OpenFlow Network

OpenFlow is a protocol which provides access to the data plane of network devices. It separates the control plane and the data plane, so that a remote controller can decide the forwarding path of network devices. Administrators can change the network topology from a software controller. Thus, the flexibility of network traffic management can be significantly enhanced.

An OpenFlow network consists of two components: an OF switch and an OF controller. The OF switch transmits data packets according to a flow table and interacts with the OF controller through a secure channel. When a packet arrives, the OF switch will check its flow table first. If the packet does not match any rule in the flow table, it will be sent to the controller through the secure channel. The OF controller will make a decision for this packet and add a rule into the flow table. When the next packet of the same flow arrives, the OF switch can handle it according to this rule. There are many powerful OF controller implementations such as NOX/POX (www.noxrepo.org), Beacon (openflow.stanford.edu/display/Beacon/Home) and Floodlight (floodlight.openflowhub.org). We can control the OF switch by programming the controller for replaying the traffic to the multi-port DUT in this work.

C. Related Work

We review existing studies and tools on capturing, generating and replaying network traffic. The studies of reducing traffic traces are also covered in this subsection.

High-performance packet capturing is critical to avoid dropping packets during the capturing process. Commercial products such as DAG Packet Capture Cards [9] are designed for zero-loss packet capture. Besides commercial solutions, the nCap architecture is presented to create a straight path from the network adapter to the user-space by means of memory mapping for wire-speed packet capture [10]. Although high-performance packet capture is not the purpose of this work, this work can adopt these solutions if the live environment is a high-speed network.

Several popular replay tools are compared in Table II. Replay tools can be either stateless or stateful. Stateless replay tools send the packets in the order according to the timestamps of packets. TCPreplay [11] can split the packet traces to simulate the behavior between the server and the client through two interfaces. Tomahawk [12] is similar to TCPreplay, but it can retransmit packets when the packets are dropped. TCPivo [13] was designed for high-performance packet replay by novel mechanisms of managing trace files, low-overhead timers, low-latency kernel patches, and priority scheduling.

Stateful replay tools can keep the states of connections (usually TCP) during replay. SocketReplay [14] mimics the TCP/IP stack and replays the payloads to maintain the TCP semantics with respect to the connection states. It also supports loss discovery to recover incomplete connections due to capture loss. Monkey [15] focuses on TCP replay. It uses the socket interface to keep the connection state and simulate the delays in the connections. TCPopera [16] emulates the state for each TCP connection, and replays the traces interactively based on the TCP connection-level and the IP flow-level parameters. Volume control replay in [17] focuses on the effectiveness of stateful replay. It can dynamically change the volume of generated traffic during replay. However, it still does not keep the states during the replay.

To improve the effectiveness of defect reproduction, we need to reconstruct a replay environment as similar as possible to the defect-triggering environment. The above replay tools are unable to reconstruct the replay environment for multi-port network devices. Since these tools cannot split traffic during replay, they cannot reproduce the defects caused by the interactions in the multi-port traffic, such as that overloading two different VLANs on a switch.

Several studies aim to reduce the huge volume of traffic in an operational network in the packet capture [4], [14], [18]–[23]. The primary purpose is to reduce the storage space for the packet traces while the important characteristics of the packet traces are kept. Lin et al. developed a tool named SocketReplay [14] to extract the subset of packet traces with events such as attacks or viruses, besides its support of stateful replay described earlier in this subsection. However, it does not deal with testing a DUT for its defects in a live environment, the support of multi-port replay, or the mechanism to sustain the operational network due to a DUT failure. Kornexl et al. [4] presented a *Time Machine* to buffer and save up to the first K bytes in the network connections (e.g., $K = 20,000$). The mechanism can save a large amount of storage space, while retaining the traces of high interest for later forensics due to the heavy-tailed nature of network traffic. Kyriakopoulos et

TABLE II
COMPARISONS OF POPULAR REPLAY TOOLS.

	Capture	Replay	Feature
TCPreplay [11]	N/A	stateless, 2 ports	divide traffic to server and client
Tomahawk [12]	N/A	stateless, 2 ports	traffic retransmission
TCPivo [13]	high volume traffic	stateless, 2 ports	high-performance packet replay
SocketReplay [14]	high volume traffic	stateful, 2 ports	long connection cutoff, payload cutoff, socket connection
Monkey [15]	low volume traffic	stateful, 2 ports	delay simulation
TCPopera [16]	N/A	stateful, 2 ports	TCP state emulation
Volume control replay [17]	N/A	stateful, 2 ports	replay traffic volume control

al. in [18] and Aceto et al. in [19] presented the compression schemes to reduce the storage requirement for packet traces, whereas this work reduces the packet traces to exclude those unrelated to triggering the defects of the DUT. The purpose and the method in this work are virtually different from theirs. Liu et al. [20] took an information-theoretic approach to the study of compressibility of packet traces and the information in the traces of different paradigms. Pescapè [21] reduced packet traces so that they are as informative as the original ones in terms of characteristics such as inter-arrival time. Amer et al. [22] and Tammaro et al. [23] studied the selections of statistical sampling methods for reducing packet traces and how they impact on the later analysis. Compared with these studies, this work aims to keep the minimum subset of the defect-triggering traffic for reconstructing the status in which the defects occurred, *instead of reducing packet traces for later traffic analysis*. Reducing packet traces for replaying defect-triggering traffic is not addressed in prior studies at all.

III. TERMINOLOGY AND NOTATIONS

In this work, a defect trace means a trace recording the traffic that causes failures of the DUT. Not all defects can be reproduced, such as those due to race conditions. For a defect trace with reproducible defects, we call it a defect-triggering trace. We classify reproducible defects into two types: *overload defect* and *protocol defect*. The former is caused by a busy condition in the DUT such as overloading the capacity of hardware and overflowing a table, while the latter is triggered by anomalous packets, such as too short (or too long) payloads and content anomaly.

The procedure in OFCR can be divided into three modes: *live mode*, *live-to-replay failover mode* and *replay mode*. OFCR spends most of the time in the live mode. It records the defect traces when the DUT fails from a normal state. To extract the right defect traces, the number of captured packets and the maximum packet lengths are both limited. The live-to-replay failover mode is a transition from the live mode to the replay mode when the network breaks down due to the DUT failure. The flow table on the OF switch is modified to keep the network alive and to deploy a multi-port replay circuit. Similarly, the failover can switch from the replay mode to the live mode when the DUT is recovered from a failure. In the replay mode, OFCR replays the defect trace to the multiple ports on the DUT. If the DUT fails again after replay, then the trace is a defect-triggering trace.

TABLE III
DESCRIPTIONS OF THE NOTATIONS.

Categories	Notation	Descriptions
DUT	N	The number of ports of the DUT
	U_i	Port i of the DUT
OF switch	D_i	Port i of the OF switch to the DUT
	P_i	Port i of the OF switch to the live network
	R_1, R_2	Ports to the replayer in the OF switch
Trace	$T = \{t_{i,j} \mid i \leq i \leq N, i \leq j \leq uc_j\}$	Defect trace in the captured traffic, where $t_{i,j}$ is the j-th packet (connection) replayed to U_i
	uc_i	Packet (connection) count to port U_i in T
	c	Packet (connection) count in T
	l	Maximum packet length in T
	$T_r = \{r_{i,j} \mid i \leq i \leq N, i \leq j \leq ucr_i\}$	Defect trace derived from T with recalculated checksums for replaying the defects, where $r_{i,j}$ is last j-th packet (connection) replayed to U_i
	ucr_i	Packet (connection) count to port U_i in T_r
	c_r	Packet (connection) count in T_r
	l_r	Maximum packet length in T_r
	T_o	Reduced trace by overload defect reduction
	T_p	Reduced trace by protocol defect reduction
	T_{min}	Minimum trace after reduction
Reduction	T_{in}	Input trace of packet/payload reduction
	T_{out}	Output trace of packet/payload reduction
	cin_i	Packet (connection) to port U_i in T_{in}
	$cout_i$	Packet (connection) to port U_i in T_{out}
	l_{in}	Maximum packet length in T_{in}
	l_{out}	Maximum packet length in T_{out}
	$head_i$	Index of the first packet (connection) to U_i in T_r
	$tail_i$	Index of the last packet (connection) to U_i in T_r
	cut/p	Cut unit in packet/payload reduction
	t_c/t_p	Cut unit in packet/payload reduction

Table III summarizes the notations used in this work. There are three types of ports on the OF switch. D_i represents port i of the OF switch to the DUT port U_i, P_i represents port i to the live network. R_1 and R_2 denote the ports to the replayer. T denotes the defect trace in the captured traffic with the total count of packets (connections) c and maximum packet length l. Inside T, $t_{i,j}$ represents the last j-th packet (connection) to U_i. If the DUT is a layer-2 or layer-3 device, the unit is packet; otherwise, the unit is connection. uc_i is the count of packets (connections) to U_i. Some packets in T are incomplete because their original length is longer than l. The incomplete packets will be dropped by a network interface, so we recalculate the checksums of the incomplete packets in T by two packet modification tools, [24] and [25] to derive T_r for replaying the defects. When OFCR operates the hybrid defect reduction, the reduced traces, T_o, T_p and T_{min} will be generated.

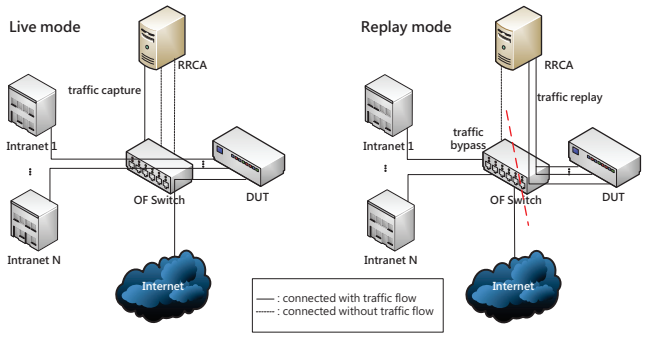


Fig. 2. The architecture of OFCR.

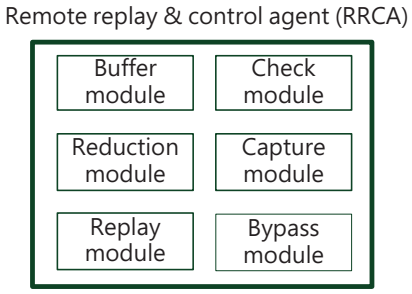


Fig. 3. The architecture of RRCA.

IV. On-the-fly Capture and Replay (OFCR) Mechanism

The trace T captured before the DUT failure may face several problems for debugging the DUT. First, the packet count c and the maximum packet length l of T may be insufficient to trigger the defects. Second, the defects may not be triggered because the replayer cannot forward traffic to multiple ports U_i like the original defect-triggering scenario. Finally, even though the defect trace T_r is small, it is not easy to identify the defect-triggering packets in T_r.

In this section, we state the OFCR mechanism and the details of each module in it. The implementation issues are also discussed. The objectives of OFCR is as follows. Given a DUT with N ports connecting to the network by an OF switch and a trace T captured when the DUT fails, the objectives are (1) to find out the minimum c and l in T that can trigger defects, (2) to replay packet $r_{i,j}$ in T_r to multiple ports U_i on the DUT, and (3) to derive the minimum defect-identifying trace T_{min} from T_r.

A. Overview of OFCR

As illustrated in Figure 2, the architecture of OFCR is composed of two modes: (1) the live mode for the DUT normal state and (2) the replay mode for the DUT failure state. In the live mode, as illustrated in the left part of Figure 2, the OF switch not only forwards bidirectional traffic between live network (the intranets and the Internet) and the DUT, but also mirrors traffic to the RRCA (Remote Replay and Control Agent). The RRCA checks the DUT states and buffers the mirrored traces. In the right part of Figure 2 is the replay mode, in which the OF switch separates the network into two parts. The left part is the live network, and the other is the multi-port replay network. The RRCA extracts the defect traces from the buffered traces, and then replays them to the DUT. OFCR uses existing two-port replay tools and splits replayed traffic on the OF switch for multi-port replay. If the defects can be triggered by multi-port replay, OFCR will perform hybrid defect identification to find the minimum defect-triggering trace. The identification involves overload defect reduction and protocol defect reduction sequentially to identify defects. The former assumes the defect is an overload defect, and the latter assumes the defect is a protocol defect. Finally, the minimum defect-triggering traces will be derived.

The mechanism switching between the two modes is called live-to-replay failover. Its major objective is to recover from network disconnection caused by the DUT failure in the live mode as soon as possible. The details will be discussed later.

All the modules in RRCA are shown in Figure 3. Which modules are active depends on the mode in which OFCR is. In the live mode, two modules are active: the buffer module and the check module. The former is used to buffer the mirrored traces, and the latter checks the DUT state. In the replay mode, there are four operating modules: the check module, the capture module, the replay module and the reduction module. The check module is the only module used in both modes. The capture module is used to store the defect traces. The replay module performs multi-port replay, and the reduction module identifies the defect in the defect trace.

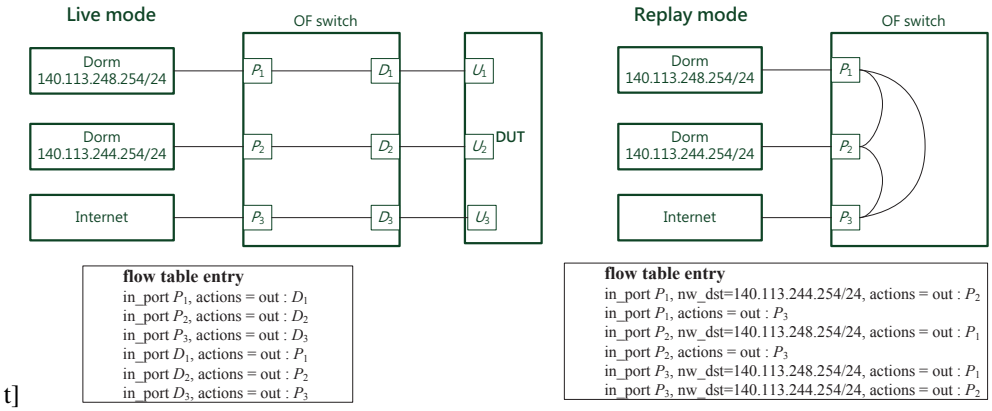
The bypass module belongs to neither the live mode nor the replay mode. It performs live-to-replay failover in OFCR. This module changes the flow table in the OF switch according to the mode of OFCR. In the live mode, the OF switch forwards incoming packets from port P_i to D_i on the OF switch, where P_i is port i to the live network and D_i is port i to the DUT (see Figure 4). Thus, packets pass through the DUT for testing. When OFCR switches to the replay mode due to the DUT failure, this module needs to cut off the connections between P_i and D_i and builds a backup circuit. Therefore, a pre-defined configuration is essential for this module to modify rules in the flow table to let packets pass through P_i.

Figure 4 is an example of live-to-replay failover. The DUT connects to two dorms (i.e., the intranets) and the Internet through an OF switch. The entries in the flow table are one-to-one mapping between P_i and D_i in the live mode. There is no entry between P_i and D_i in the replay mode. The entries about D_i will be excluded, and the bypass module will build the rules of relationship between P_i by a pre-defined configuration.

The DUT to be tested could be any network devices as long as a backup device is available for failover. The OF switch can serve as the role of the backup device (if a dedicated one is unavailable), if it is implemented or equipped with the functions of the DUT. With a dedicated backup device, an example of live-to-replay failover is illustrated in Figure 5; otherwise, the example had been illustrated in Figure 4.

B. The Live Mode of OFCR

Figure 6 presents the behavior of the modules in the live mode. The check module monitors and records the DUT states, and the buffer module records the mirrored traffic T from P_i of the OF switch. The trace T will be reserved for a while. If



t]

Fig. 4. An example of live-to-replay failover.

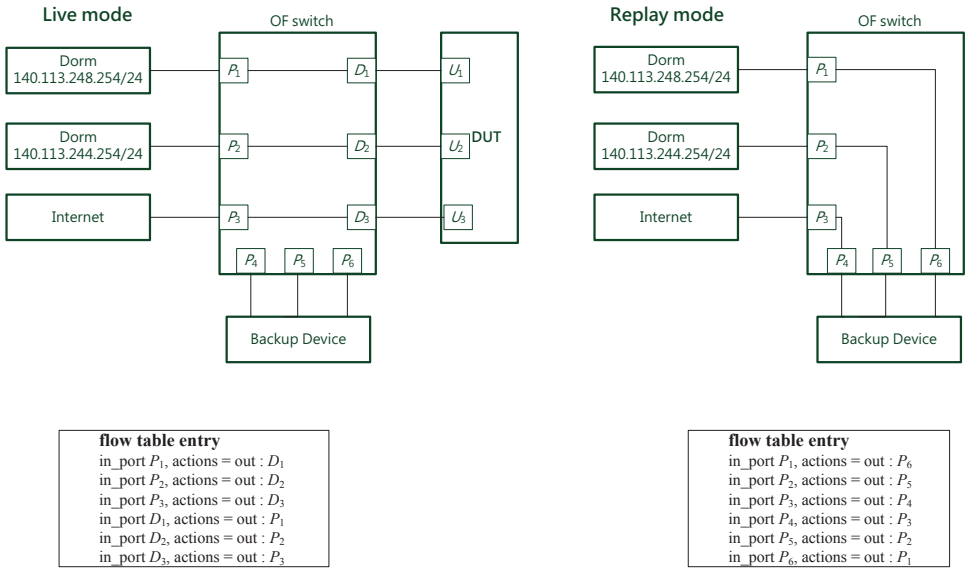


Fig. 5. An example of live-to-replay failover with a backup device.

the DUT keeps normal, this trace will be removed because it is irrelevant to triggering the defects.

1) Check module: The check module is responsible to check the DUT states by probing the DUT (to be explained in Section IV-D). To collect defect information, this module accesses and records the DUT states through SNMP or the console port. The states can be categorized into common states and specific ones. The former states are CPU usage, memory usage, bandwidth usage and error logs. The latter states can be those in the MAC table and the ARP table on a switch, or in the ARP table and the routing table on a router, depending on the layer of the DUT.

2) Buffer module: The buffer module captures mirrored traffic T continually with c packets (connections) and the maximum packet length, l bytes. The capture size has a tradeoff between memory storage and defect effectiveness. Because common network devices usually have 1∼4 ports for traffic mirroring, it is likely that the mirroring ports encounter bandwidth overloading. The buffer module can apply many-to-many mirroring by setting the flow table rules on the OF switch. This approach can reduce packet losses by relieving the bandwidth overloading, and can also allocate diverse mirroring groups for the intranet and the Internet to reduce the overloading in the replay pre-processing.

C. The Replay Mode of OFCR

Figure 7 presents the modules in the replay mode. The check module does the same job as it is in the live mode. It is used to determine the effectiveness of replaying the defect trace. The capture module extracts the defect trace T_r from the buffer module in the live mode. The replay module forwards packet $r_{i,j}$ to different ports D_i, in order to transmit to the proper ports U_i in the DUT, where $r_{i,j}$ is the last j-th packet (connection) to the DUT port U_i in T_r. The reduction module downsizes the defect-triggering trace T_r to derive the minimum trace T_{min}. The reduction module can use different reduction approaches according to the detection type.

1) Capture module: The capture module extracts the defect trace from the buffered trace T. The defect trace T_r has the same capture size $c_r = c, l_r = l$. The packet (connection) count to DUT port U_i, ucr_i, may not be equal to uc_i because the DUT ports for replay can be different from those in the live mode during the multi-port replay. The maximum packet length l may lead to incomplete packets during capturing, so the module needs to recalculate the checksum of each packet in T_r to derive packet $r_{i,j}$.

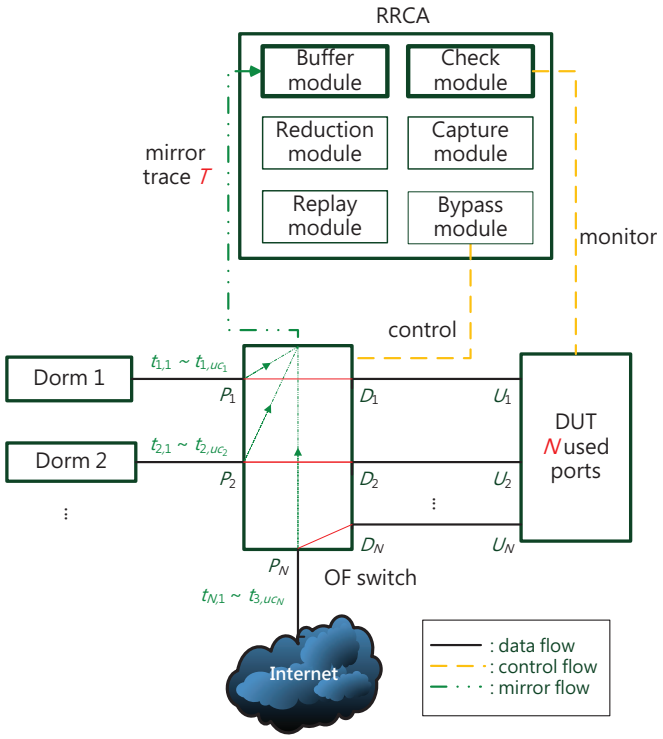


Fig. 6. The live mode of OFCR.

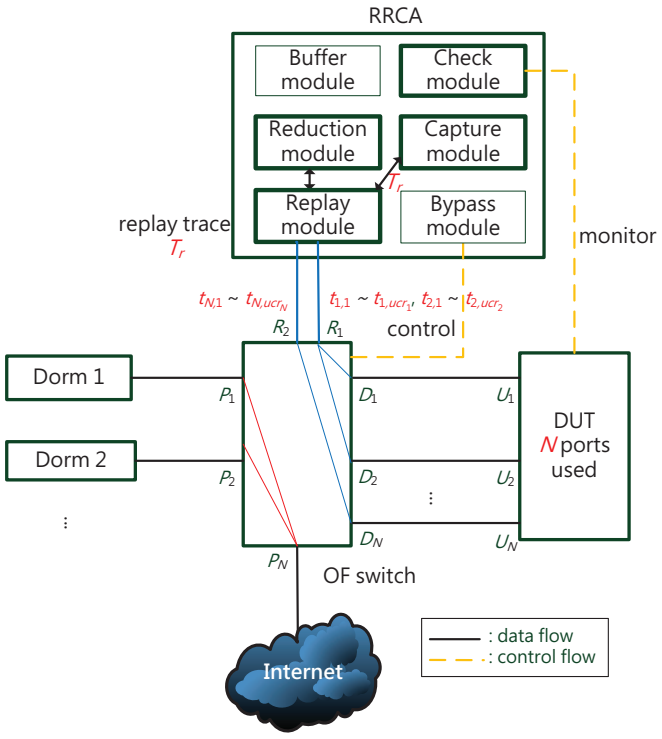


Fig. 7. The replay mode of OFCR.

2) Replay module: The replay module replays T_r to trigger defects. To reconstruct the scenario similar to that in which the DUT is in the live mode, we replay each packet in T_r to the original DUT port in the live mode. This module splits T_r into the intranet side and the Internet side, and then replays T_r from ports R_1 and R_2, depending on which side the packet (connection) belongs to. To forward packets $r_{i,j}$ to the corresponding port D_i, the OF switch splits incoming packets by their source IP addresses. The relations between the source IP addresses and the ports should be also specified in the pre-defined configuration to configure the flow table on the OF switch. Figure 8 presents an example of the replay module. There are two dorms in the live mode, so the OF switch splits the packets from R_1 to D_1 and D_2 according to their subnets. Thus, we need the pre-defined configuration to configure the flow table to implement the forwarding paths on the right side of Figure 8 accordingly, so that OFCR can quickly switch to the new forwarding paths for the replay.

3) Reduction module: The reduction module identifies defect-triggering traces by hybrid defect reduction. It assumes the defect may be an overload defect or a protocol defect, and then applies both the overload defect reduction and the protocol defect reduction. As illustrated in the left part of Figure 9, the hybrid defect reduction performs two reductions sequentially, and generates two reduced traces T_o and T_p. We can determine T_{min} by comparing T_o, T_p and the original defect trace T_r.

There are two conditions to derive the minimum trace T_{min}. (1) If $T_o = T_r = T_p$, it means that T_r is not a defect-triggering trace or that T_r is the minimum defect-triggering trace with no redundant packets. We set T_{min} to T_r. (2) Otherwise, we keep both reduced traces $T_{min} = T_o \cup T_p$. Because T_o and T_p may be different in packet (connection) count and maximum packet length, we preserve both traces to keep the information for debugging.

In the right part of Figuree 9 are the procedures of both reductions. Because the overload defects are usually caused by flooding packets and the number of defect-triggering packets is unclear. When the packets are reduced, the results may be quite different each time. Therefore, to minimize the reduced trace size, the overload defect reduction removes redundant payloads first, and then concentrates on reducing the packet count. In contrast, protocol defects are caused by one or a few packets. To save the processing time of replay in the reduction, the protocol defect reduction downsizes the number of packets first, and then finds the critical parts of payloads.

Figure 10 presents the flow chart of packet reduction and payload reduction with binary search. The input trace is T_{in}, which is a subset of T_r, and has two parameters cin_i and l_{in}, where cin_i is the packet (connection) count through DUT port U_i and l_{in} is the maximum packet length. The output trace of the reduction is T_{out}. Similarly, T_{out} has $cout_i$ and l_{out}. The two reductions may increase or decrease the packet (connection) count by cut or p (i.e., the cut unit), and they set the thresholds t_c and t_p to stop the reduction. When cut or p meets the thresholds, the reduction stops and generates T_{out}. Packet reduction removes redundant packets before and after the defect-triggering part. We use $head_i$ and $tail_i$ to represent the indexes of the first and the last packet (connection) to port i in T_{in}. The packets with an index between $head_i$ and $tail_i$ will be kept in T_{out}. The left and right parts of this figure present the packet reduction before and after the defect-triggering part, respectively. If the device is a layer-4 or layer-7 one, the packet reduction will cut the trace in the unit of connection because

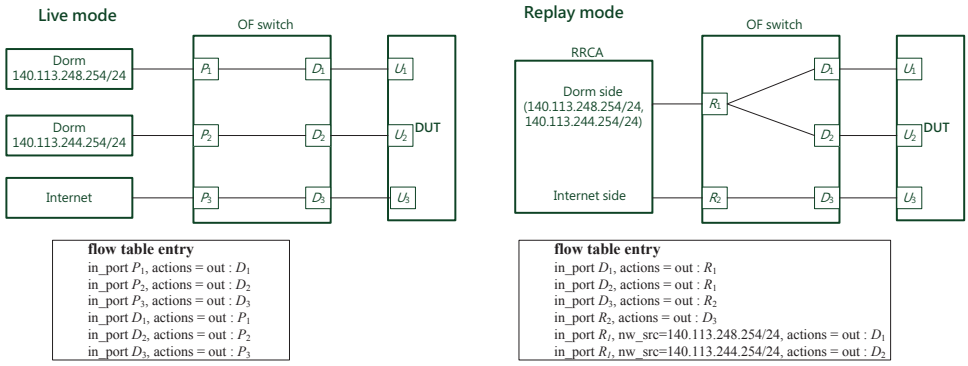


Fig. 8. Example of the replay module.

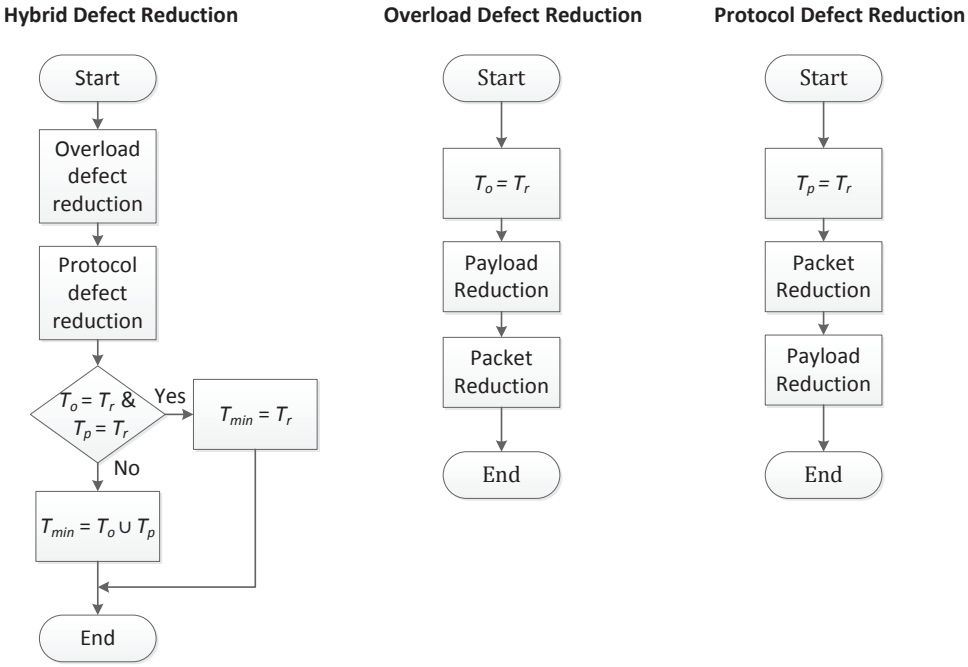


Fig. 9. The procedure in the reduction module.

an incomplete connection cannot reproduce the connection state. The payload reduction is simpler than packet reduction because it only reduces the maximum packet length of T_{in}.

D. Implementation Issues

In this subsection, we discuss the implementation issues of traffic capture and failure detection. The captured traffic may lose packets in the mirroring module on the network devices and the capture interfaces in the RRCA. To reduce packet losses in the traffic mirroring, we use the OF switch to deploy several mirroring ports. For packet losses on the network interfaces in the RRCA, we add memory (up to 8GB) and use an enhanced traffic capture tool, Gulp (staff.washington.edu/corey/gulp), which uses a ring buffer, and allocates the packet reader and writer in different CPUs to reduce packet losses. Furthermore, the buffer module records traces by appending a number which is in a loop to the end of trace file name. It is used to prevent the situation in which the defect-triggering packets are recorded in the end of the first trace and the beginning of the second trace, but because we buffer a single trace at a time, we only get the second trace finally.

Failure detection is important because it determines mode switching in the RRCA. The tool used in the check module, namely CheckDev, was developed by NBL (www.nbl.org.tw). It sends ARP, ICMP and HTTP requests to the DUT so as to probe the DUT status. Moreover, it retrieves the DUT states by the SNMP and console ports. Because of the diverse commands in the console port for different DUTs, we write specific scripts by Expect (www.nist.gov/el/msid/expect.cfm) to dump the state information from the DUT. Another implementation issue in the failure detection is the failure criteria. If the failure criteria are too loose, some defect traces will be lost; otherwise, we will get more defect traces and process more live-to-replay failover. The defect traces may contain normal traces. It is hard to distinguish normal traces and non-reproducible defect traces from defect traces because they both do not trigger any failure during replay. The failure criteria involves three parameters: *check interval*, *check timeout* and *tolerant consecutive failure time*. They determine live-to-replay failover and keep the effectiveness of defect traces.

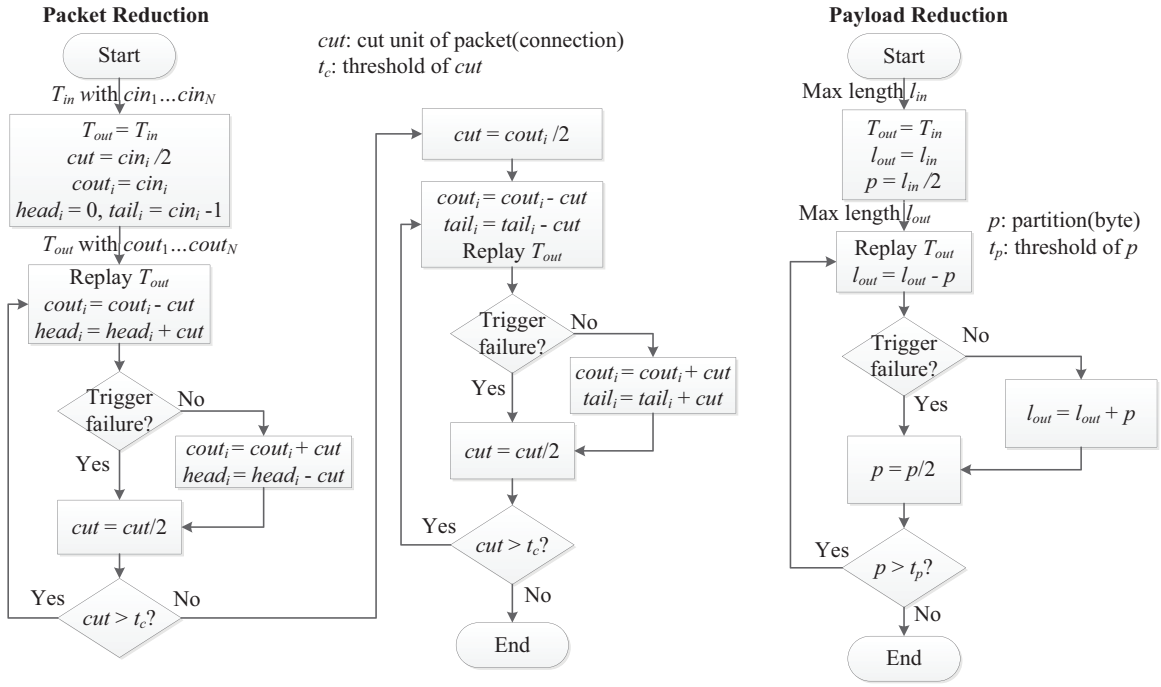


Fig. 10. Packet reduction and payload reduction.

V. Experiments and Analysis

The experimental environment and results will be discussed in this section. First, we introduce the testbed in this work, and then discuss the experimental results from the perspectives of the vendors and users. Finally, we analyze the cases of the defect traces.

A. Experiment Testbed

The real traffic *in an operational network* cannot be customized. In other words, it cannot be expected that certain defects of a specific protocol will happen on a specific DUT and their packet traces can be collected in a live environment. We use network debugging tools, TestCenter (www.spirent.com/Ethernet_Testing/Software/TestCenter) and Codenomicon, to emulate the appearance of certain defect-triggering traffic. Therefore, we can evaluate the performance of OFCR for debugging certain defects of the DUT in the experiments; otherwise, the defects may not appear in a live environment. If the generated traffic triggers a defect on the DUT and it can trigger the defect by replaying, we capture it as a defect-triggering trace for the experiments. For reproducing the experiments, we also capture the dorm traffic over a period as the normal trace used in the experiments.

The testbed presented in Figure 11 is composed of two steps. The first step is collecting normal and defect-triggering traffic, as illustrated on both sides of the figure. On the left side is the collection of normal traffic from the dorms of our campus, and on the right side is the collection of defect-triggering traffic. The multi-port DUTs are ZyXEL GS-2750 (a layer-2 switch) and SuperMicro SSE-G24-TG4 (a router) in the experiments. The RRCA is a PC equipped with an Intel i3-2130 processor, 8GB memory and 9 network interfaces.

After collecting normal and defect-triggering traffic, we conduct the second step, as illustrated in the middle part of Figure 11. Because the OF switch TL-WR1043ND is a SOHO AP (controlled by FloodLight, www.projectfloodlight.org/floodlight), it is not capable of handling the amount of captured dorm traffic. We use a subset of the traffic to perform multi-port replay and live-to-replay failover with the OF switch. It is noted that we could conduct the experiments with a high-end OF switch, but another higher-end OF switch we have does not have full support of the functions we need. Specifically, we are unable to change the flow table in it due to its immature design (still a prototype so far). Thus, we still use the SOHO AP as the OF switch in the experiments. Fortunately, the observations we get depend on the *functionality* of the OpenFlow switch, instead of the *capacity* of it. Using a high-end OF switch will not change the main observations in the experiments.

The multi-port replay sends traffic from the RRCA to the OF switch, which then passes traffic to the DUT. The connectivity tester is used to measure the effectiveness of live-to-replay failover. We conduct the live-to-replay failover experiment by replaying defect-triggering traces from the RRCA to the DUT directly and at the same time probing the connectivity tester though the OF switch, which transmits the probe messages to the connectivity tester directly or through the DUT according to the DUT state. We use the RRCA and the DUT without the OF switch in the experiment of traffic capture and reduction because the traffic can overload the OF switch. Once the DUT fails, RRCA will start to replay after reducing the packet traces via multiple links, as illustrated in the middle of Figure 11 (only two links for replaying traffic are shown in the figure).

The compositions of the normal traces and defect-triggering traces are presented in Table IV. The normal traces are collected from one dorm and two dorms in the campus. Around 65% of them are TCP traffic, and the others are UDP traffic.

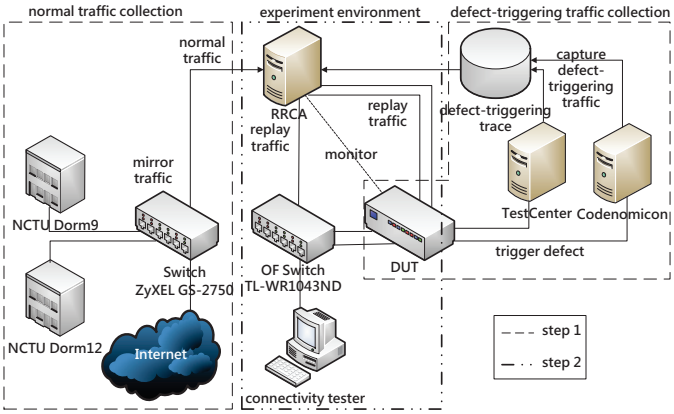


Fig. 11. Experimental environment.

TABLE IV
TEST TRACES.

Trace	Type	Trace count	Average size	Packet count
two-dorm traffic	normal	1	918.5MB	818,970
one-dorm traffic	normal	1	446.3MB	368,238
IP payload anomaly	protocol	10	56.7KB	873
Dense ARP requests	overload	1	72.1MB	540,012
Dense ICMP requests	overload	1	864KB	6,331
OSPF payload anomaly	protocol	3	20.3KB	243

There are dozens of application protocols in them, including BitTorrent, STUN, SSH, etc., and none of them are dominant ones. Codenomicon produces the traces with protocol defects, including the IP payload anomalies on the ZyXEL switch and the OSPF payload anomalies on the SuperMicro router. The former anomalies are packets with anomalous fields in the IP header (90% of the packets are ICMP packets, and the others are fragmented IP packets), and the latter anomalies are OSPF packets with anomalous fields. We select the above anomalies because they are the only ones from Codenomicon that can break down the DUTs in the experiments. TestCenter generates the traces with the overloading defect on the ZyXEL switch. All the packets in them are two types of requests for overloading: (1) dense ARP requests to overflow the MAC table and (2) dense ICMP (from ping) requests to overflow the ARP table. They can overflow the MAC table and the ARP table on the ZyXEL switch.

We mix a normal trace and a defect-triggering trace to emulate the defect-triggering traffic in the live network. The procedure of traffic mixing is replaying a normal trace and a defect-triggering trace from two different network interfaces to the DUT simultaneously, and capturing them as a defect trace from the DUT. As a result, the mixed packet traces will contain the packets from normal traffic among those from the defect-triggering trace.

If the number of normal packets into the mixed traces is changed, the downsizing ratio, the packet count and the processing time will be certainly affected, but not the maximum packet length (see Section V-B). We use the normal packet traces from one dorm and two dorms to study the impact of this change on the results in Section V-B. It is also noted that the order of the packets in the defect-triggering trace should be preserved for OSPF payload anomaly because the defect occurs after a certain state transitions in the OSPF operation. The order is not so critical for the other anomalies because the DUT will be still busy or unable to properly handle the fragments. The position of the defect-triggering trace in the mixed traffic may affect the processing time of reductions, but not the other results because amount of the defect-triggering trace is the same.

For L2 devices, we have 12 mixed defect traces for the one-dorm environment and 12 for the two-dorm environment. In both environments, the mixed defect traces combine the normal traffic with 10 IP payload anomaly traces, a dense ARP requests trace and a dense ICMP requests trace, so there are 10 protocol defect-triggering traces and 2 overload defect-triggering traces in these 12 mixed defect traces. For L3 devices, we only have 3 OSPF payload anomaly traces. After traffic mixing of a two-dorm normal trace and an OSPF anomaly trace, we derive 3 protocol defect-triggering traces for L3 devices.

B. Experimental Results

The experimental results can be viewed from the perspectives of the vendors and the users. Vendors care about the effective capture size in the OFCR live mode, the diversity of reduction for various types of defects, and the efficiency of reduction thresholds in the OFCR replay mode. The only thing users are concerned about is the outage time during the live-to-replay failover, so the relationship between the failure criteria and the outage time is also discussed.

1) Capture size: The capture size is the first we need to decide when starting OFCR. The parameters c and l influence the effectiveness and the size of the captured defect traces, and their optimal values should be found to balance the above two things for DUTs of different layers. The results of the ZyXEL switch in the one-dorm and two-dorm environments are presented in Figure 12 and 13, and those of the SuperMicro router are presented in Figure 14.

Figure 12 presents the effectiveness of defect reproduction for different packet count c and maximum packet length l in the two-dorm testbed. The results show that the first 46 bytes of the payload (only ARP/ICMP headers) are sufficient to trigger all the defects. Moreover, capturing 10,000 packets ($c/c_r = 1.2\%$) in the trace is sufficient to keep the protocol defect-triggering packets, and $c = 50,000$ packets ($c/c_r = 6.1\%$) in the trace are required to keep all defect-triggering packets in the 12 mixed defect traces because the overload defect traces need a large number of packets to break down the DUT. When l is less than 46 bytes, only one overload defect trace can be reproduced with $c/c_r \geq 6.1\%$. This is because this overload defect is triggered by too many broadcast packets. Even though these captured broadcast packets only include their IP headers, they still can paralyze the DUT.

The L3 SuperMicro router is also tested with the 12 mixed defect traces in the two-dorm testbed. We find out that 10 protocol defect traces are ineffective because they are defects

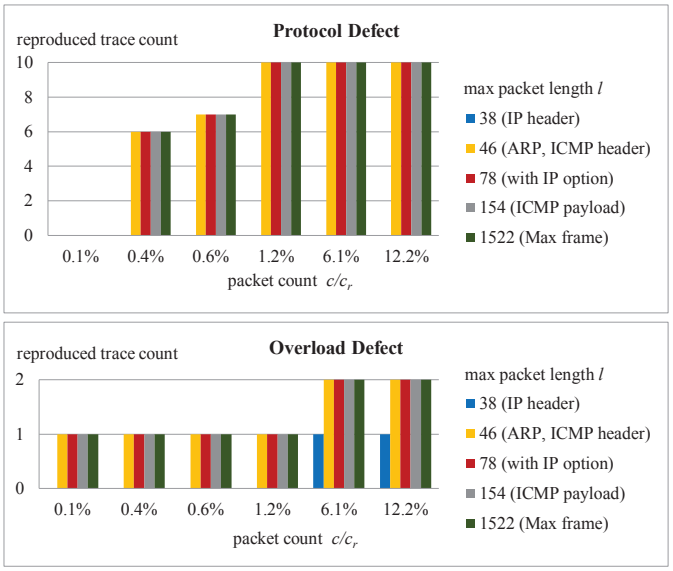


Fig. 12. Capture size of L2 switch in the two-dorm testbed.

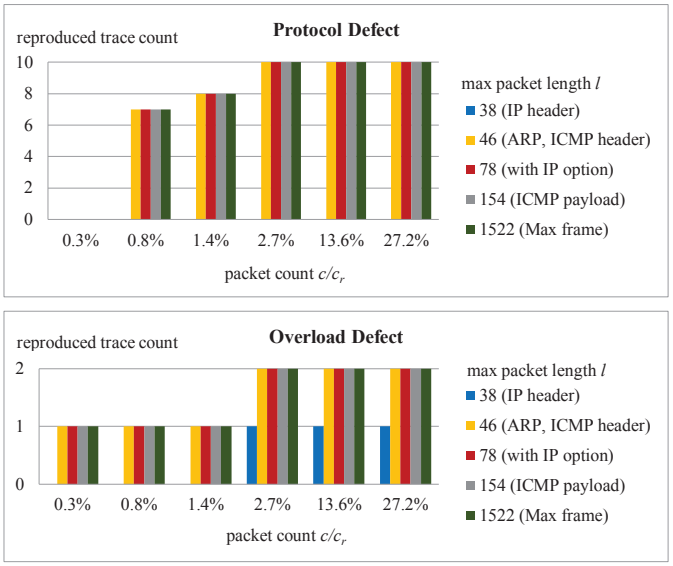


Fig. 13. Capture size for the L2 switch in the one-dorm testbed.

specific to the ZyXEL switch, but the two overload defect traces still can trigger the defects when l is 46 bytes and c is $50,000$ ($c/c_r = 6.1\%$). Thus, the captured traffic with a maximum packet length $l = 46$ bytes is sufficient to keep the overload defect for the router.

Figure 12 presents that the capture sizes of the L2 switch in the one-dorm environment. The difference between Figure 12 and Figure 13 is the traffic scale of the environment; the other factors in both experiments are the same. Because the traffic volume in the one-dorm environment is much smaller than that in the two-dorm environment and thus the overload defect traces occupy a larger proportion of the total trace, we can reproduce the overload defect traces with $c = 10,000$ ($c/c_r = 2.7\%$) rather than $50,000$ ($c/c_r = 6.1\%$) in the two-dorm environment, and the reproduction effectiveness of the protocol defect trace is improved slightly with $c = 3,000$ ($c/c_r = 0.8\%$) and $5,000$ ($c/c_r = 1.4\%$). To reproduce all

t]

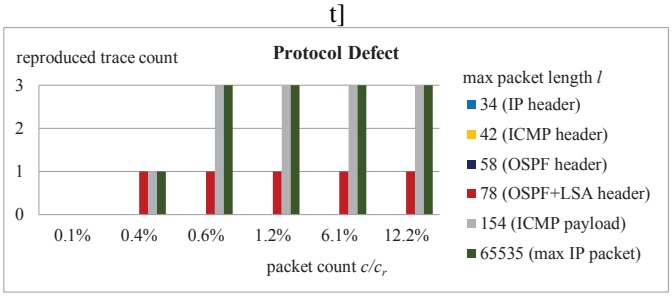


Fig. 14. Capture size for the L3 router in a two-dorm testbed.

protocol defect traces, the capture size of packet count c is $10,000$ ($c/c_r = 2.7\%$), and the maximum packet length l is 46 bytes. The capture packet count $c = 10,000$ with the maximum packet length $l = 46$ bytes can also reproduce all overload defect traces. Therefore, capturing $10,000$ (2.7%) packets in a trace and the maximum packet length 46 bytes is sufficient to reproduce all 12 defect-triggering traces in the one-dorm environment.

Figure 14 presents the results of the L3 router in the two-dorm environment. There are only three protocol defect traces in this experiment. We need to capture the first 154 bytes of the payloads to trigger the protocol defects. These protocol defects are triggered by LS (Link State) update packets in the OSPF. The OSPF component of the DUT is unable to handle the inconsistency in the fields of OSPF packets when it processes the LSA (Link State Advertisement) information following the LSA header in the LS update packet. Because these protocol defects need the OSPF payloads to trigger the defects, the maximum packet length l depends on the maximum LSA of the router. In our experiment, the SuperMicro router needs the first 154 bytes to trigger defects.

2) Diversity of reductions: Because of the different properties of the protocol and overload defects, a single reduction will not be suitable for both types of defects. The effectiveness of different reductions is evaluated by the processing time and the downsizing ratio. The former is the time from receiving the trace to finishing the test, and the latter is the relationship between the trace size after reduction and the original trace size. The formula is expressed as

$$downsize\,ratio = (1 - \frac{trace\,size\,after\,reduction}{original\,defect\,trace\,size}) * 100 \quad (1)$$

We compare five reductions in the OFCR with the protocol defects and overload defects in Figure 14 and 15. The original defect traces have 50,000 packets and the maximum packet length is 1,522 bytes. The reduction thresholds t_c and t_p are 1,000 and 10 bytes.

In Figure 15, either the packet reduction and payload reduction alone is not good on trace downsizing. The protocol defect reduction and overload defect reduction have the same downsizing ratio (98.8%) on the protocol defects, but the latter has better downsizing ratio on the overload defects (up to 96%). The hybrid defect reduction chooses the best result from the protocol defect reduction and overload defect reduction, so its efficiency is as high as expected.

Figure 16 shows the results of processing time of different

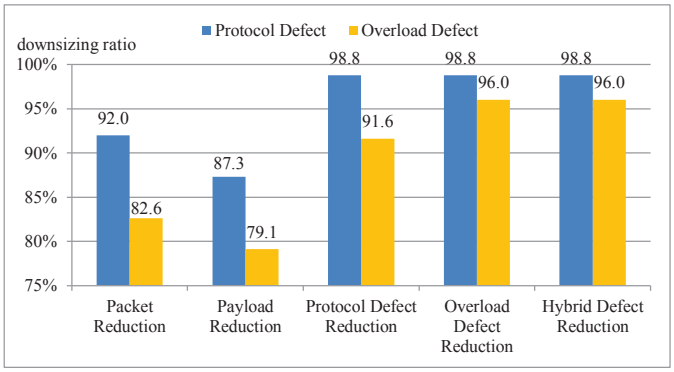


Fig. 15. Downsizing ratio of reductions.

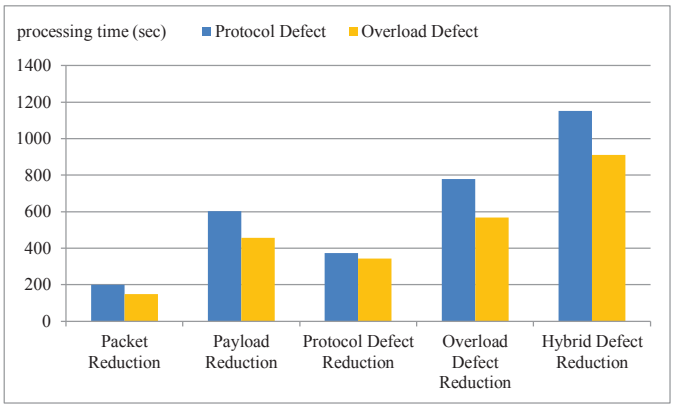


Fig. 16. Processing time of reductions.

reductions. The processing time of the protocol defects is longer than that of the overload defects in each reduction. The reason is that there are few defect-triggering packets in the protocol defect traces, so protocol defect traces need more rounds to replay the reduced trace during the reduction. Because the packet count influences the replay time significantly, the packet reduction and protocol defect reduction which reduce packet counts first will spend shorter processing time than other reductions. However, considering the downsizing ratio, protocol defect reduction is more efficient than packet reduction because it spends additional 200 seconds to reduce more 6% of the size (up to 98% for protocol defect). The downsizing ratio of the protocol defect reduction is insufficient for overload defect (91.6%). The overload defect reduction spends 230 more seconds to remove additional 4% of the size (up to 96%). The hybrid defect reduction has the longest processing time because it performs both protocol defect reduction and overload defect reduction.

3) Efficiency of reduction thresholds: The efficiency of the reduction is controlled by t_c and t_p, which are the thresholds of the *cut* unit for packet count and the maximum packet length. The results of choosing diverse thresholds in the hybrid defect reduction are presented in Figure 17. The defect traces also have 50,000 packets and maximum packet length 1,522 bytes. Large thresholds t_c and t_p will save the processing time but lower the downsizing ratio. However, when the thresholds t_c and t_p are too small, the processing time will increase significantly but the downsizing ratio will rise limitedly. Given

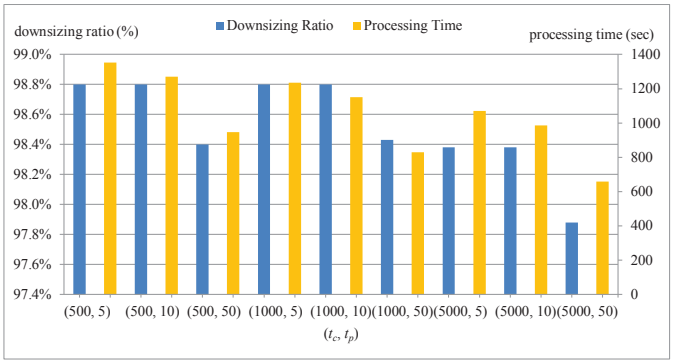


Fig. 17. The efficiency of reduction thresholds.

the thresholds t_c = 5,000, 1,000 and 500, and $t_p = 50, 10$ and 5, we can derive the best downsizing ratio 98.8% when $t_c \leq 1,000$ (2% of the packet count) and $t_p \leq 10$. Considering the processing time, the reduction is the most efficient when $t_c = 1,000$ and $t_p = 10$.

4) Outage time vs. failure criteria: The defects affect different parts of the DUT. Some defects make the DUT fail slightly, and the DUT can recover from the failure by itself in a short period of time. Some break down the DUT, which cannot be recovered until the administrators reboot it. The outage time is the time when the OFCR spends in switching from the live to the replay mode during the DUT failure and from the replay to the live mode when DUT is recovered. The users will experience disconnection in this period.

To reduce the outage time, we detect a failure with different check intervals and numbers of tolerable consecutive failures. The live-to-replay failover will be executed when the number of consecutive DUT failures exceeds a threshold. The response timeout is the waiting time for the reply of the probing request. Figure 18 presents the outage time resulting from different check intervals and numbers of tolerant consecutive failures with 1-second response timeout for the ZyXEL switch. Either a defect or no defect will be triggered in the experiment. The leftmost bar presents the outage time without failover (19 seconds), which is the real failure time of the DUT. We can find out that most failure criteria can reduce the outage time except (7, 3). Given the same tolerant consecutive failure time, the smaller the check interval is, the shorter the outage time is. A small number of consecutive failures can also reduce the outage time, but it is better to be larger than one. If the number is one, we may presume a DUT failure just because of incomplete reply packets or packet losses, and then proceed to meaningless live-to-replay failover.

C. Case Studies

We also interpret the protocol defect traces in the test traces. When Codenomicon is used to test the ZyXEL switch with IP anomaly traces, the ICMP module of the switch crashes, but the switch itself does not detect it. We try to analyze the packets in the IP anomaly defect trace without any error logs, and find out that the anomaly is in the `flags` and `fragment offset` fields of the IP header. Figure 19 is the details about the IP anomalies.

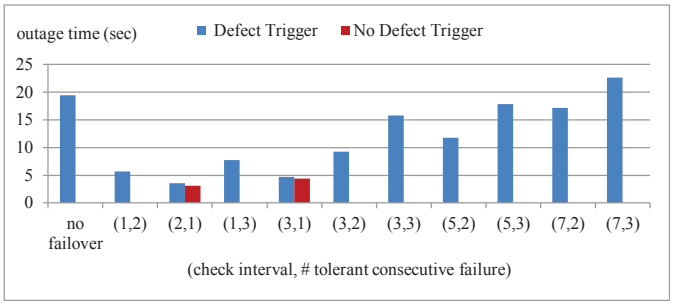
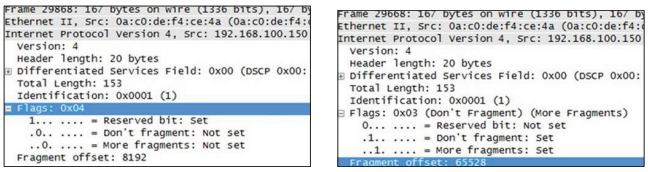


Fig. 18. Outage time under different failure criteria.



(a) artificial traffic replay testing. (b) live testing.

Fig. 19. The example of IP anomaly.

Figure 19(a) presents the anomaly in the `flags` field. The first bit in this field is a reserved bit, and it should be 0, but here it is 1. Figure 19(b) presents the anomalies in both the `flags` and `fragment offset` fields. The `more` bit in the `flags` field is set to 1, meaning there is another fragment, but the fragment offset value is the maximum value. It is impossible to have another fragment after this fragment. This anomaly is hard to be detected because if these two fields are checked separately, either is correct, but they become an anomaly when appearing together.

Although the logs in the DUT have no error messages, we still can analyze the defects by accessing other DUT states. We find out that there is no overflow in any tables, and the CPU and memory are not in the busy states. Combining with the information and the behavior of the DUT failure, we can conjecture that a single anomalous packet may generate an exception, but it does not break down the relevant component in the DUT. However, consecutive IP fragment anomalies could break down the component because of the queues for the fragment buffer. The protocol defects may not be caused by just a single anomalous packet. They could be generated by a sequence of anomalous packets, which are much fewer than the packets triggering overload defects.

VI. Conclusion

The OFCR mechanism can extract the defect-triggering traffic in the live network on the fly, and implement multi-port replay to emulate the situation in which a defect happens in a multi-port network device under test. In the live mode, the OF switch passes the live network traffic to the DUT and mirrors the traffic to the RRCA. OFCR switches to the replay mode when detecting a DUT failure. The OF switch lets the live traffic bypass the DUT and connects the DUT to the RRCA to replay the captured defect trace to the proper DUT ports. If the defect can be reproduced, the OFCR performs the defect identification by hybrid defect reduction. The framework is at National Chiao Tung University and is available to vendors who need to have their products tested. Thus, the framework is available to the industry. It is not in the form of an open source tool.

We demonstrate the efficiency of defect reproduction and ability of failover in OFCR, even though the numerical results will certainly vary with a different DUT (thus different defects) and the real traffic in the operational network. (1) Only a small portion of the captured traffic is required for the replay because the downsizing ratio can be up to 98.8% for the protocol defects, and 96% for the overload defects. (2) The outage time of the operational network is only a few seconds. If the system were deployed in a larger environment and suppose there were more normal traffic (like the discussion of one-dorm vs. two-dorm environment), the proportion of defect-triggering traffic would be decreased, and more overall packets would be required for the replay. If the anomalies are in the packet payloads instead of the packet headers, longer packet lengths should be kept. The bottom line is that the anomalies should be preserved in the reduced packet traces.

We plan to test more upper-layer DUTs to find suitable capture sizes for them in the future. Improving the accuracy of failure detection is also a direction of our future work. So far, CheckDev only sends ARP, ICMP and HTTP requests to probe the DUT, but sometimes a specific component in the DUT breaks down, and the DUT can still serve those probe messages. In this case, CheckDev cannot detect such a failure, so its functionality should be expanded to support more types of service probing like IGMP or RIP. Moreover, some failures are caused by the medium instead of the DUT, and it is impossible to reproduce this kind of failure by replaying traffic to the DUT. We hope to develop a checking mechanism on the OF switch to detect the failure of medium between the DUT and RRCA.

Acknowledgment

This work was supported in part by National Science Council and Industrial Technology Research Institute, Taiwan, and also in part by ZyXEL Inc., D-Link Corp., and Cisco Systems Inc. The authors would like to thank the staff engineers at Network Benchmarking Lab and Information Technology Service Center of National Chiao Tung Univeristy, Taiwan, for their continuous support on the beta site infrastructure.

References

[1] The Spirent TestCenter homepage, http://www.spirent.com/Products/Spirent-TestCenter/EnterpriseAndDataCenterNetworks.
[2] A. Botta, A. Dainotti, and A. Pescapè, "A tool for the generation of realistic network workload for emerging networking scenarios," *Computer Networks*, vol. 56, no. 15, pp. 3531–3547, Oct. 2012.
[3] N. Bonelli, A. D. Pietro, S. Giordano, and G. Procissi, "Flexible high performance traffic generation on commodity multi-core platforms," *2012 International Conference on Traffic Monitoring and Analysis.*
[4] S. Kornexl, V. Paxson, H. Dreger, A. Feldmann, and Robin Sommer, "Building a time machine for efficient recording and retrieval of high-volume network traffic," *2005 ACM Internet Measurement Conf.*
[5] Y.-D. Lin, P.-C. Lin, S.-H. Wang, I-W. Chen, and Y.-C. Lai, "PCAPLib: a system of extracting, classifying, and anonymizing real packet traces," *IEEE Syst. J.*, to appear.
[6] Y.-D. Lin, I-W. Chen, P.-C. Lin, C.-S. Chen, and C.-H. Hsu, "On campus beta site: architecture designs, operational experience, and top product defects," *IEEE Commun. Mag.*, vol. 48, no. 12, pp. 83–91, Dec. 2010.
[7] J. Sommers, V. Yegneswaran, and P. Barford, "A framework for malicious workload generation," *2004 ACM Internet Measurement Conference.*

[8] S. Floyd and V. Paxson, "Difficulties in simulating the Internet," *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, pp. 392–403, Aug. 2001.
[9] DAG Packet Capture Cards, http://www.emulex.com/products/network-visibility-products/dag-packet-capture-cards.
[10] L. Deri, "nCap: wire-speed packet capture and transmission," *2005 IEEE Workshop on End-to-End Monitoring Techniques and Services.*
[11] The TCPreplay homepage, http://tcpreplay.synfin.net.
[12] The Tomahawk homepage, http://tomahawk.sourceforge.net.
[13] W. Feng, A. Goel, A. Bezzaz, W. Feng, and J. Walpole, "TCPivo: a high-performance packet replay engine," *2003 ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research.*
[14] Y. D. Lin, P. C. Lin, T. H. Cheng, I. W. Chen, and Y. C. Lai, "Low-storage capture and loss-recovery selective replay of real flows," *IEEE Commun. Mag.*, vol. 50, no. 4, pp. 114–121, Apr. 2012.
[15] Y. C. Cheng, U. Hölzle, N. Cardwell, S. Savage, and G. M. Voelker, "Monkey see, monkey do: a tool for TCP tracing and replaying," *2004 USENIX Technical Conference*, General Track.
[16] S. S. Hong and S. F. Wu, "On interactive Internet traffic replay," *2005 Symp. Recent Advanced Intrusion Detection.*
[17] W. Chu, X. Guan, Z. Cai, and L. Gao, "Real-time volume control for interactive network traffic replay," *Computer Networks*, vol. 57, no. 7, pp. 1611–1629, May 2013.
[18] K. G. Kyriakopoulos and D. J. Parish, "Applying wavelets for the controlled compression of communication network measurements," *IET Commun.*, vol. 4, no. 5, pp. 507-520, 2010.
[19] G. Aceto, A. Botta, A. Pescapè, and C. Westphal, "Efficient storage and processing of high-volume network monitoring data," *IEEE Trans. Network and Service Management*, vol. 10, no. 2, pp. 162-175, 2013.
[20] Y. Liu, D. Towsley, T. Ye, and J. C. Bolot, "An information-theoretic approach to network monitoring and measurement," *2005 ACM SIGCOMM Conference on Internet Measurement.*
[21] A. Pescapè, "Entropy-based reduction of traffic data," *IEEE Commun. Lett.*, vol. 11, no. 2, pp. 191-193, Feb. 2007.
[22] P. D. Amer and L. N. Cassel, "Management of sampled real-time network measurements," in *1989 Conference of Local Computer Networks.*
[23] D. Tammaro, S. Valenti, D. Rossi, and A. Pescape, "Exploiting packet-sampling measurements for traffic characterization and classification," *International J. Network Management*, vol. 22, no. 6, pp. 451-476, 2012.
[24] C. Y. Ku, Y. D. Lin, Y. C. Lai, P. H. Li, and K. C. J. Lin, "Real traffic replay over WLAN with environment emulation," *2012 IEEE Wireless Communications and Networking Conference.*
[25] The Colasoft packet builder homepage, http://www.colasoft.com/packet_builder.

Ying-Dar Lin (F'13) is Professor of Computer Science at National Chiao Tung University (NCTU) in Taiwan. He received his Ph.D. in Computer Science from UCLA in 1993. He served as the CEO of Telecom Technology Center in Taipei during 2010-2011 and a visiting scholar at Cisco Systems in San Jose during 2007-2008. Since 2002, he has been the founder and director of Network Benchmarking Lab (NBL, www.nbl.org.tw), which reviews network products with real traffic. He also cofounded L7 Networks Inc. in 2002, which was later acquired by D-Link Corp. His research interests include design, analysis, implementation, and benchmarking of network protocols and algorithms, quality of services, network security, deep packet inspection, and embedded hardware/software co-design. His work on "multi-hop cellular" was the first along this line, and has been cited over 600 times and standardized into IEEE 802.11s, IEEE 802.15.5, IEEE 802.16j, and 3GPP LTE-Advanced. He is an IEEE Fellow (class of 2013) for his contributions to multi-hop cellular communications and deep packet inspection, and also an IEEE Distinguished Lecturer for 2014/2015. He has served on the editorial boards of several journals and program committees of many conferences. He published a textbook *Computer Networks: An Open Source Approach* (www.mhhe.com/lin), with Ren-Hung Hwang and Fred Baker (McGraw-Hill, 2011).

Po-Ching Lin received the B.S. degree in computer and information education from National Taiwan Normal University, Taipei, Taiwan, in 1995, and the M.S. and Ph.D. degrees in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2001 and 2008, respectively. He joined the faculty of the Department of Computer and Information Science, National Chung Cheng University (CCU), Chiayi, Taiwan, in August 2009. He is currently an Assistant Professor. His research interests include network security, network traffic analysis, and performance evaluation of network systems.

Yu-An Lin received the master degree in computer science from National Chiao Tung University in 2013. He serves as the research & development software engineer at Telecommunication Laboratories, Chunghwa Telecom Co. in Taiwan. His research interests include network traffic behavior, network devices testing and cloud computing.

Yuang-Cheng Lai received the Ph.D. degree in computer science from National Chiao Tung University in 1997. He joined the faculty of the Department of Information Management at National Taiwan University of Science and Technology in 2001 and has been a professor since 2008. His research interests include wireless networks, network performance evaluation, network security, and social networks.