Bug Traces: Identifying and Downsizing Packet Traces with Failures Triggered in Networking Devices

Ying-Dar Lin, Chun-Nan Lu, Yuan-Cheng Lai, and Zongo Pawendtaore Eliezer

ABSTRACT

Testing networking devices before releasing them onto the market is a way of ensuring quality and robustness. Replaying artificial or real-world traffic is a method to test networking devices. Using real-world traffic is desirable as it uncovers more realistic properties. The challenges of testing with real-world traffic are mainly the *high volume* of the captured traces and the prolonged time required for replay testing. In order to efficiently *reproduce* the failures of networking devices and reduce the replay time, it is necessary to reduce the size of the traces that have triggered the failures. In this work, two algorithms used to *downsize* the traces but still retain the failures they triggered, Binary Downsizing (BD) and Linear Downsizing (LD), are proposed. Meanwhile, a metric called downsizing ratio (DR), the ratio between the size of the downsized traces and that of the original traces, is defined in order to evaluate the efficiency of the trace downsizing. Three kinds of probes following the basic RFC benchmarking requirements, ARP, ICMP, and HTTP requests, are regularly sent to diagnose the devices during the testing. ARP and ICMP probes test the reachability of a networking device hosted on the local network, and HTTP probes check if the device still responds to users' requests. The evaluation of failure distribution shows that 70 percent of failures happened because they failed to respond to one of the three probes, 23 percent failed to respond to two probes, and 7 percent failed to respond to all probes. From the downsizing experiments, LD was inferred to have a slightly higher DR than that of BD, but BD generally would require fewer iterations than LD.

Ying-Dar Lin, Chun-Nan Lu, and Zongo Pawendtaore Eliezer are with National Chiao Tung University. Yuan-Cheng Lai is with National Taiwan University of Science and Technology.

INTRODUCTION

Testing is one of the methods used to discover product defects in the development stage, and aims at reducing the number of after-sale failures found by customers [1]. A defect is usually a kind of inherent fault in the product. If the defect is triggered, in certain situations the product would produce the wrong results, which means failures. Even though they pass a series of tests during their development, networking devices may still reveal customer found defects (CFDs) when they are used under real-world conditions. This could be explained by the fact that laboratory testing often uses artificial traces that have less realistic properties, and less diversity and complexity in terms of applications and protocols [2]. For instance, applications such as peer-to-peer (P2P), video streaming services, and online games often have proprietary protocols or diverse behaviors, which are hard to mimic with artificial traffic. In order to reduce CFDs, networking devices should be tested using real-world as well as artificial traffic, because there is a higher chance of triggering device bugs or failures that would not easily be discovered when using artificial traffic because of the complicated context of the real world.

In order to *reproduce* failures to facilitate the debugging process, real-world traffic needs to be *captured* and later *replayed*. The major constraints of real-world traffic replay are twofold: high volume of captured traces and time consumption [3]. For example, during peak hour traffic, the captured network traffic volume at the Beta Site [1] could reach 60 Gbytes in 30 minutes, and even during regular hours around midnight, the traffic volume could still reach at least 20 Gbytes in 30 minutes. Reproducing a failure triggered by high volume traffic is time-consuming. The downsizing of such a failure-triggering packet trace, called a *bug trace* in this work, to reduce the time to trigger the failures turns out to be a very crucial and imperative operation.

To reduce the size of traces while maintaining authentic and reliable information capable of reproducing the very *same* failures triggered by the *original* traces is not trivial. *Failure-driven* trace downsizing is challenging since the downsized traces should be representative of the entire original traces and able to ensure failure reproduction while reducing testing time.

There has been some work on network traffic data reduction and compression. Two trace

0163-6804/14/$25.00 © 2014 IEEE

reduction approaches were based on the concept of entropy and were aimed at reducing traces with the objective of improving traffic trace analysis by removing data *redundancy* in a large trace [3, 4]. While those two investigations focused on traffic data reduction, in [5–7] the emphasis fell instead on the compression of packet *headers*. Their goal was to apply a trace compression technique to online packet headers, allowing communication over low-bandwidth channels. While the purpose of these investigations was to reduce trace collection storage, none of them resulted in failure-driven trace downsizing.

In our failure-driven trace downsizing approach, we differentiate the failures that occur with different types of devices based on the distinct observed features and then reduce the traces in different groups. The downsized failure-triggering traces were evaluated by measuring the downsizing ratio (DR), and the possible causes of the networking device failures can be addressed by doing a forensic investigation of the downsized traces.

The first to be identified among the raw large traces were those that triggered device failures. The testing technique used was based on that of in-laboratory real testing (ILRT) [8], where the devices under test (DUTs) are constantly monitored during replay testing. Once a failure is triggered, the identified traces are downsized using a linear or binary downsizing algorithm, named Linear Downsizing (LD) and Binary Downsizing (BD), respectively. LD is done through an incremental rollback and replay technique, while BD is based on a binary search technique, and is used as a complement to LD. The final step consists of splitting and classifying traces according to the failures triggered.

The rest of this article is organized as follows. In the next section, some important related literature is surveyed. We then describe the terminologies and discuss the related issues. Next, our proposed algorithms are presented, and their performance is evaluated. Finally, our conclusions are given.

Related Work

Traffic Capture

Testing networking devices with real-world traffic requires the capture of traces in a more *controlled* environment. The quality of the captured traces is a determining factor of the accuracy and efficiency of tests performed on products or experimental studies [9]. There is a beta site testing environment within the National Chiao Tung University campus where the network traffic generated by volunteers can be captured [1]. This environment, called Beta Site, pursues the goal of evaluating products' performance with real-world traffic. More than 800 students were invited as volunteers, and their daily traffic is captured in files with PCAP format. Beta Site provides a good way to evaluate DUTs in an online *live* mode or an offline *replay* mode.

Stability Testing

Networking devices that have passed laboratory tests still have a good chance of failing in a real-world environment. CFDs, the defects found after the release of the devices, are not trivial to clarify because of the unclear context experienced by customers, and are likely to damage a manufacturer's reputation. In order to assess as early as possible whether a networking device can *stably* work for an extended time, the Network Benchmarking Lab (NBL) [10] used *ILRT* and *NATreplay* [8] to test the stability of WLAN routers. NATreplay is a modified replayer based on *Tcpreplay* [11] and integrated with a Network Address Translation (NAT) functionality. Stability testing consisted of replaying network traffic to a DUT while monitoring it with a tool called *CheckDev*. Figure 1 shows the overview of ILRT. CheckDev is a probing tool that regularly sends Address Resolution Protocol (ARP), HTTP, and Internet Control Message Protocol (ICMP) queries to a DUT to check if the DUT is still alive. The PCAP Library stores the traffic traces fetched from Beta Site, and NATreplay separates the traffic into two directions, and sends them to the LAN and WAN interfaces of the DUT.

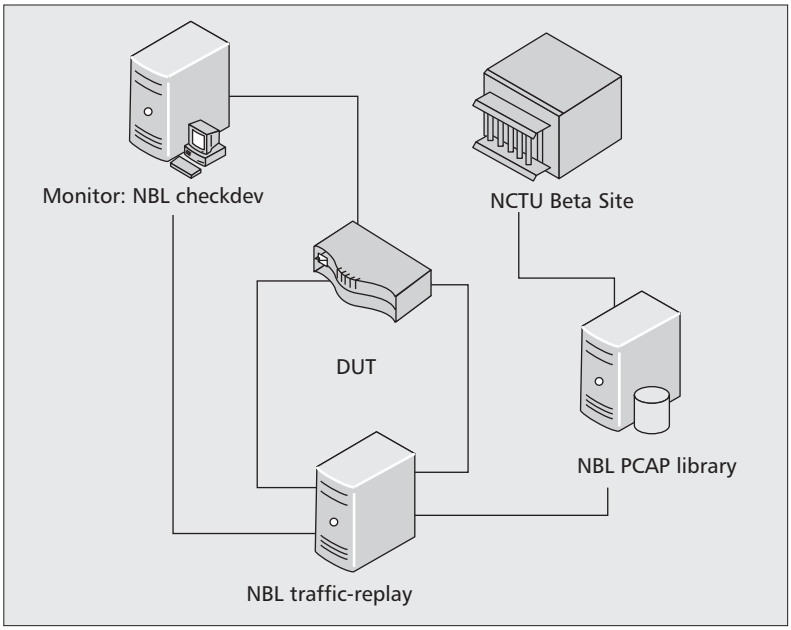

Figure 1. In-laboratory replay test.

The traffic used to test a DUT was double-checked after each round of replay test. If it triggered some failures, it was regarded as *valuable* and stored; otherwise, the traffic was discarded. Figure 2 shows the trace collection process. First, the traces were fetched from Beta Site, and then replayed to multiple DUTs using ILRT. If the traffic triggered failures on several DUTs simultaneously, the replayed traffic was marked as a *bug trace*, and collected to another repository. Figure 2 illustrates the process of using n DUTs as the threshold.

Trace Reduction

There are several studies related to network traffic trace reduction. They aim at different objectives, and their approaches are diverse. The main objective of Botta *et al.* [3] consists of improving data collection and analysis on one hand and reducing the size of the original data set with an acceptable loss of properties on the

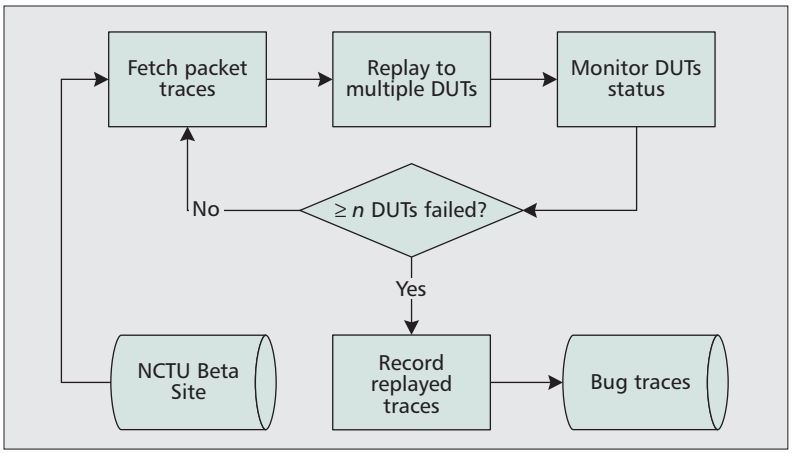

Figure 2. Bug trace collection.

other. It proposes an offline entropy-based approach for reducing large data sets applied to Counter-Strike traffic traces. It aims at producing network traffic statistics by completely characterizing and modeling network traffic without losing sensible information. The proposed offline technique to reduce traffic trace data sets has the advantage of correctly capturing mean, standard deviation, and marginal distributions of network traffic traces, without compromising time properties.

The study of Yong Liu *et al.* [5] proposes a compression algorithm that removes redundancy in order to produce substantially smaller traces. A method to compress and anonymize packet traces [7] intends to resolve two main problems in packet trace collection and analysis: volume of data and privacy issues. It is based on the fact that there are many fields in the IP, UDP, and TCP headers that do not change over the lifetime of a connection when observed at a single location.

It is obvious that these approaches all share the objective of either reducing or compressing traces in order to either improve data storage or reduce the complexity of traffic trace analysis. However, these do not meet one of the requirements of our study: the trace reduction methods are not failure-driven, as we propose. The purpose of our work consists first of identifying traces that have triggered failures of networking devices, then downsizing the traces in order to reduce their volume. Therefore, it is failure-driven downsizing of bug traces.

Trace Downsizing Issues

Lin *et al.* [1] suggest that one month and one year of beta site testing are minimum test durations for low-end and high-end products, respectively. In order to reproduce and analyze the triggered failures, real-world traffic needs to be captured and later replayed. The obstacles to replaying the captured real-world traffic are the enormous storage requirements for the captured traffic traces and the lengthy replay time required to trigger the failures. In order to save the storage and replay time required to reproduce the failures, the captured traces have to be downsized. However, trace downsizing is not trivial because the results and causality of the triggered failures should still be retained after the downsizing operation. The downsized traces can be used not only to analyze the reasons why the failures happened, but also to test other similar networking devices, whether they have the same failures or not. In this section, three related issues are addressed.

Failure Distribution

Beta site testing is good at triggering stability failures because of the inherent complicated user behaviors and diverse application contents of real-world traffic. During the stability testing process, three types of probes, ARP, ICMP, and HTTP, are sent by the CheckDev to DUTs [8]. ARP and ICMP probes are used to test the reachability of a networking device hosted on a local network. If too many such probes failed, the DUT is likely to be regarded as down, which affects the overall path routing and dynamic topology of the network and, even worse, degrades the overall performance. HTTP probes are used to check if a networking device still responds to a user's requests. If such probes fail, it means the DUT is functionally faulty. Based on the combinations of responses, seven kinds of failures were differentiated and classified into three groups: *single-failure*, *combined-failure*, and *hanging*, which means all three types of probes failed. The single-failure group is composed of a failure caused by any one of the ARP, ICMP, or HTTP probes failing to respond. The combined-failure group encompasses the failures caused by any two ARP, ICMP, or HTTP probes failing to respond, and hanging happens when all three probes fail to respond. Failure distribution is a means that could reveal the observable structure of stability failures of networking devices.

Downsizing Ratio

DR is the ratio between the size of reduced traces and that of the original traces. It is an expression of the number of times by which original traces have been reduced and is a metric to evaluate the effect of the downsizing process. However, DR is meaningful only in a situation where the original triggered failures are kept. Networking devices with similar functionalities differ in the way they are implemented. They might not function the same way under the same conditions, and thus are unlikely to trigger the same failures. Besides, DR would also be affected by the types of failures. For example, DR would be higher for failures caused by some critical packet than failures caused by a series of accumulated mistakes. Lastly, even for the same failures, different replay throughput settings may cause distinct replay results because they put different pressure on resource allocation. Higher replay throughput implies less time to adjust resource allocation and hence raises the possibility of failures, which indirectly affects DR.

Factors Causing Device Failures

Finally, a very important aspect worth investigating is an analysis of the causes of device failures. Packet trace properties [8, 9] would trigger faulty operations of networking devices. For instance, traffic such as P2P that is generated by

multiple concurrent connections often causes a sudden burden on DUTs in terms of resource allocation and garbage collection, thus affecting their stability. Digging out the factors causing failures from downsized traces would be an efficient way to improve the quality of DUTs in a short period of time.

Packet Trace Identification and Downsizing

The design of our solution is composed of two major components. The first component is *trace identification*, where individual triggered failures are correlated with the corresponding bug traces via the failure logs. Each of the failure logs includes the types of failures that occurred, the failure occurrence time, the number of built connections, and the total size of replayed traffic, which are all used to find and extract the suspicious traces from the original traces.

The other component is *trace downsizing*, where two downsizing algorithm alternatives, a Linear Downsizing (LD) algorithm and a Binary Downsizing (BD) algorithm, are invoked to downsize the volume of bug traces and still retain the triggered failures. The basic processing unit in both LD and BD is a packet. Without loss of generality, it was assumed that there was no a priori knowledge of what was contained in real-world traffic; otherwise, testers could quickly locate the critical packet(s) or flow(s) that triggered the failures by test reports or logs.

Linear Downsizing

The idea of LD is *rollback-and-replay*. Whenever a failure is triggered, the failure would be logged and the sequential traces triggering the failure are abstractly regarded as a whole and divided into equal-sized pieces of traces from the beginning based on a predefined size, the *rollback size*. Next, the traces are replayed from the last trace toward the first one. If the failure cannot be triggered again, LD repeatedly involves the rollback size of previous adjacent traces that were replayed right before the failure occurred, and replayed from the beginning of the new, larger, combined traces until the failure is reproduced. Then the replayed traces triggering the failure would be stored as the final downsized ones. LD is likely to be advantageous if the traffic traces triggering the failure belong to the last few replayed traces.

Binary Downsizing

BD locates the sequential traces triggering the failure by recursively splitting the traces in halves and replaying the smaller ones in turn until the failure is missed. Next, the traces located between the last two testing rounds are iteratively augmented in halves to replay until the original failure is reproduced. Then the replayed traces triggering the failure are stored as the final downsized ones.

Figure 3 is an illustration of how LD and BD actually work. In the illustration, the volume of the original bug traces triggering the failure is 8047 Mbytes, and the rollback size is set to 500 Mbytes for LD. ✓ means that the traces triggered the failure while × means that the traces did not. In Fig. 3a, each time the bug traces left are regarded as a whole, split into halves in turn, and replayed, while in Fig. 3b, the candidate output traces are augmented and replayed until the failure is reproduced. The final downsized traces are 1572 Mbytes and 1547 Mbytes for BD and LD, respectively.

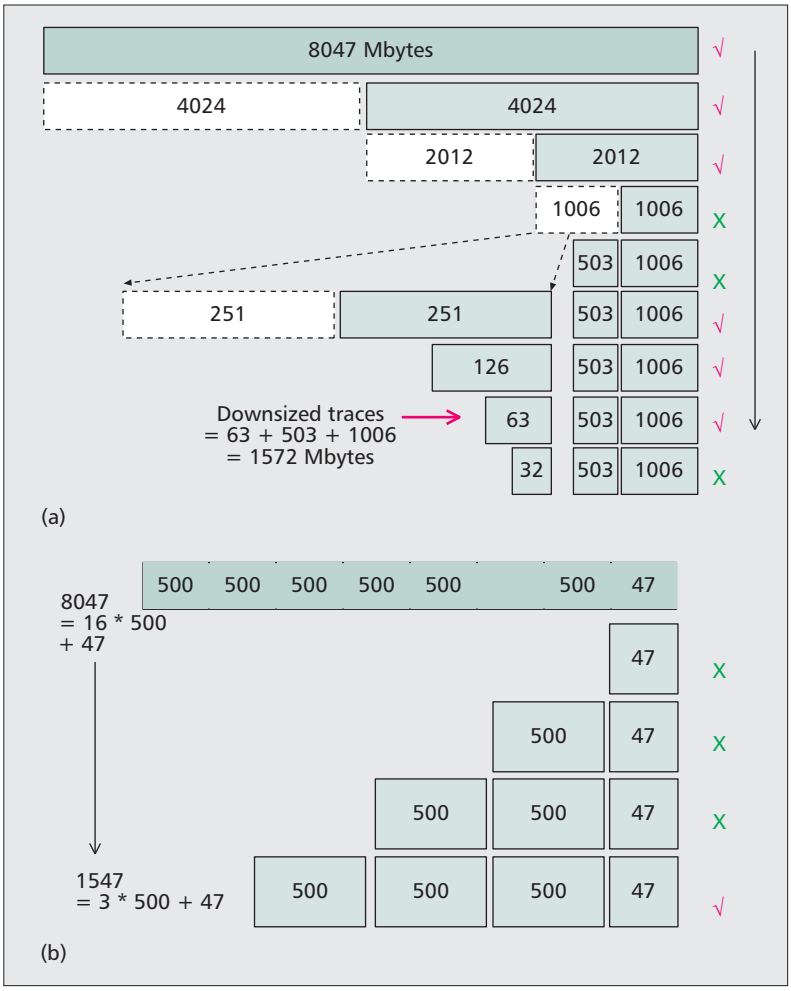

Figure 3. BD and LD illustrations: a) BD process; b) LD process.

Experiment Studies and Results Analysis

In this section, the experiments conducted to identify the traces that trigger the failures of networking devices are discussed. The outcomes of the two downsizing algorithms, LD and BD, are compared, and a case that triggered a DUT to crash is studied.

Trace Selection and Experimental Testbed

To conduct the experiments, we use the network traffic captured from the NCTU Beta Site and employ ILRT testing to retain 34 PCAP files causing the failure of four WLAN routers. The total size of the 34 files is 1.2 Tbytes, and the application profile of the traces is listed in Table 1.

There are hundreds of applications detected, and only the applications with volume ratio larg-

Name	Ratio (%)	Name	Ratio (%)	Name	Ratio (%)	Name	Ratio (%)	Name	Ratio (%)
HTTP	29.3	UDP	2.73	Garena	0.43	ku6speedup	0.12	webmail	0.04
PPStream	10.76	YouTuBe	1.9	SSH	0.3	Steam	0.07	YouKu	0.04
FTP	10.33	Flashcom	1.5	RTSP	0.23	FlashGet	0.06	DNS	0.04
TCP	9.84	MSN	1.4	tudou	0.21	VNC	0.06	Dropbox	0.03
PPLive	9.84	QQTV	1.33	Gmail	0.2	qingyule	0.06	Shoutcast	0.03
RDP	4.61	QVOD	0.94	Fs2you	0.15	KuGoo	0.06	Warcraft	0.02
BitTorrent	3.77	eDonkey	0.96	iTunes	0.13	Telnet	0.05	SMB	0.02
Funshion	3.69	Skype	0.56	RTP	0.13	Direct Play7	0.05	TTPlayer	0.01
Xunlei	3.43	HTTPS	0.49	TeamViewer	0.12	Gnutella	0.05	IMAPs	0.01

Table 1. The application profile of the original raw traces.

er than 0.01 percent are listed. The four WLAN routers are used as the DUTs. They are labeled DUT1, DUT2, DUT3, and DUT4.

Testbed Configuration

The experimental testbed used is based on that of ILRT, and throughout our experiments the replay throughput is set to 50 Mb/s. CheckDev is configured with the information of medium access control (MAC) and IP addresses of the DUT and NATreplay, the LAN and WAN subnets, and the *timeout*. The timeout is the maximal allowable interval between the probe and the corresponding response from the DUT. If CheckDev cannot collect the responses from the DUT in time, the issued probes are judged to have failed. The traffic captured in PCAP files is separated into two sides, primary and secondary, and the network addresses contained are bound to different interfaces of NATreplay. In this way, the DUT can see that the traffic a host in the primary side initiates to another host in the secondary side will go through it one way, and the response will go through it the other way.

Failure Distribution

The DUT would be regarded as normal if it could reply to all probes within the timeout interval; otherwise, it is likely that the DUT has suffered some failures. These results show that most DUTs still have failures when measured against their specifications even though they passed the laboratory test. For the single-failure group, the individual occurrence of the ARP, HTTP, and ICMP failures is 10, 31, and 29 percent, respectively; for the combined-failure group, the occurrence of ICMP/HTTP, ARP/ICMP, and ARP/HTTP failure is 17, 3, and 3 percent, respectively. Finally, the percentage of the ARP/ICMP/HTTP failure (i.e., the hanging failure) is 7 percent. The sum of single-failure groups (i.e., 70 percent) dominates the total failures because:

- They are the most fundamental probes, and in general cases, if a networking device works, it must be able to respond to these probes.
- Compared to ARP requests, ICMP and HTTP requests require more CPU resources, which is difficult when the CPU is overburdened with heavy loads.

Downsizing Ratio

Two equations are proposed to evaluate the DR of the two downsizing algorithms. The first equation, DR_1, compares each single downsized trace with respect to its corresponding failure. In this case, each failure's DR is the size of the downsized trace over that of the trace triggered the failure. DR_1 is expressed as

$$DR_1 = 1 - \frac{\sum_{i,j=1}^{n} \frac{\left|\left(P_j^{L_i}\right)_d\right|}{\left|P_j^{L_i}\right|}}{n},$$

where $(P_j^{Li})_d$ represents the *downsized* jth trace P_j, which triggered the ith failure, log L_i. $|P_j|$ means the size of P_j, and n is the total number of triggered failures.

DR_2 compares each single downsized trace with respect to the original trace, P, instead. In this case each failure's DR is the size of the downsized trace over that of the original trace used for the testing, and DR_2 is expressed as

$$DR_2 = 1 - \frac{\sum_{i,j=1}^{n}\left(P_j^{L_i}\right)_d}{|P|}.$$

The overall DR is evaluated by using the two equations DR_1 and DR_2, and the rollback size is set to 500 Mbytes. Using DR_1, the DR of BD and LD is 77.2 and 79.6 percent, respectively. With DR_2, the DR of BD and LD is 81.9 and 84.1 percent, respectively. Based on these results, DR_2 evaluation results in a DR slightly higher than that of DR_1. This is because with DR_2 the size of the downsized trace was compared against that of the whole original trace, which is usually high in volume (1.2 Tbyte in our case). Hence,

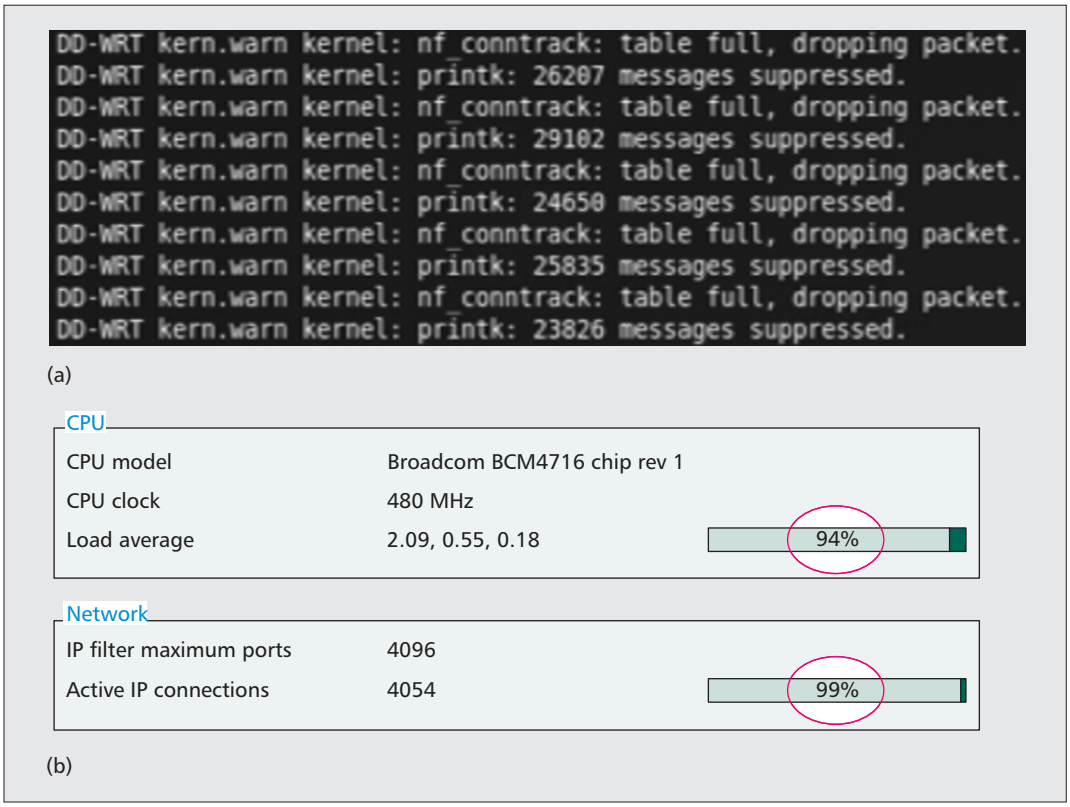

Figure 4. Observations by DD-WRT: a) logs observed; b) internal status observed.

If there are too many resource allocation requests and garbage collection tasks burst out in a short time and their execution orders are not well-scheduled, it is very likely to trigger failures, resulting in performance degradation or even DUT crashing.

LD is inferred to have a slightly higher DR than that of BD, 81.85 (LD) and 79.55 percent (BD), but BD generally would require fewer iterations than LD, which only requires 64.3 percent of LD in time.

Investigating the Determining Causes of Failures: Case Study

The DUTs used in our experiments behave as black boxes because they do not disclose detailed information about the triggered failures. In order to gain more insight about the failures, an open source third-party firmware, *DD-WRT* [12], has been installed on DUT_4, and then DUT_4 was used to redo the replay testing using downsized traces. Figure 4a shows the logs as the testing proceeded, and one message, *nf_conntrack: table full, dropping packet*, was shown repeatedly, generated by the function of connection tracking of the Netfilter framework [13]. Connection tracking allows the kernel to keep track of all logical network connections or sessions, and thereby relates all of the packets, which is used by NAT to translate all related packets to make up the connections. The repeated message means that the NAT table was too full to receive any more new packets, so the DUT was forced to drop subsequent packets. In Figure 4b, the other observed messages show that the DUT had to handle up to 99 percent of the maximum number of acceptable connections, which almost overloaded the DUT, resulting in a high CPU utilization of 94 percent, and consequently failed to respond to any probes.

With the help of product developers, we further explore the observations and find that they are mostly caused by the mutual interaction between resource allocation and garbage collection. If there is a new connection to build, resource allocation would be invoked to allocate resources; if there is a dated one to clean up, garbage collection would be invoked to retrieve the resources allocated previously. If too many resource allocation requests and garbage collection tasks burst out in a short time, and their execution orders are not well scheduled, it is very likely to trigger failures, resulting in performance degradation or even DUT crashing.

Conclusions

Replaying artificial or real-world traffic is an alternative way to test networking devices. Using real-world traffic is preferable as it uncovers more realistic properties. However, the high volume of captured traces and the lengthy replay time are unavoidable obstacles. Therefore, it is necessary to downsize the captured bug traces to more efficiently reproduce the triggered failures.

This work analyzed bug traces collected from the NCTU Beta Site and distinguished seven different types of device failures, which were later classified into three groups: *single-failure*, *combined-failure*, and *hanging*. The triggered failures correspond to three types of probes, ARP, ICMP and HTTP queries sent to probe the DUTs. The corresponding percentage of the total failures for the three groups is 70 , 23 , and 7 percent, respectively. According to these results, most

The context of real-world traffic is more complicated and harder to track, which always forces CPU and the resource allocation mechanisms to decide which connections should be abandoned or executed.

DUTs still have failures in this real-world replaying testbed, even they have passed laboratory testing. The single-failure group (70 percent) dominates the total failures because:

- They are the most fundamental probes, and in general cases, if a networking device works, it must be able to respond to these probes.
- Compared to ARP requests, ICMP and HTTP requests require more CPU resources, which is difficult when the CPU is overburdened with heavy loads.

Two downsizing algorithms, Linear Downsizing and Binary Downsizing, were proposed to downsize the bug traces that triggered the failures of networking devices. From the downsizing experiments, LD was inferred to have a slightly higher DR than that of BD, 81.85 (LD) and 79.55 percent (BD), but BD generally would require fewer iterations than LD, which only requires 64.3 percent of LD in time. PCAP Lib [14], a library of packet traces developed based on the framework of bug trace collections, is the concrete implementation and has been a test service of NBL.

In investigating the causes of the failures, we observed that the failures were easier to trigger while the NAT table was filled with the connections and the CPU was overloaded for a long time, which might not be easy to generate in a laboratory test. The context of real-world traffic is more complicated and harder to track, which always forces the CPU and resource allocation mechanisms to decide which connections should be abandoned or executed.

Lessons Learned

In this work, the distinct observed failures were used as key features of classification on the premise that we had no a priori knowledge about the implementation details of DUTs. If the details can be obtained, the development defects can be included as features as well. Although it may happen that multiple triggered failures all map into a single defect, it is worthwhile to retain all triggered failures for further analysis. Even for the same defect, different triggered failures may imply distinct logic flaws, which should be highly valued. For example, when the NAT table is too full to receive any more new packets, the system may become unstable and start to trigger many failures. However, a full NAT table may be caused by the following:

- A connection that is uncompleted but selected to be removed may reenter the NAT table soon because of inadequate victim-connection-selection operations.
- Imperfect resource recovery operations that the resource allocated to a completed connection should recover correctly and quickly.

Thus, in order to retain the completeness and correctness of a test, only the failures confirmed by product developers are removed from our bug trace.

Apart from the above conclusions, we also learned several valuable lessons in replay testing with real-world traffic and list them as follows. For product developers:

1) A user's behavior is highly complicated and cannot be imagined, let alone that of thousands of users. Unpredictable contexts of an individual connection and numerous connections orthogonally affect the resource reservation and performance of a networking device. The context of each trace that triggered failures once should be confirmed, clarified, and revised to improve the quality of products. Meanwhile, a library of user behavior can be established for other development and testing of products.

2) Resource allocation and recovery operations are critical and likely to be invoked frequently during the operation. However, each invocation could be far from perfect and would accumulate side-effects. Whether the DUTs remain stable under such circumstances is difficult to quantify. The unpredictable and frequent access DUTs of real-world users can help to test product stability.

3) A networking device should be able to respond to all ARP, ICMP, and HTTP probes based on specifications. It should be more aggressive to consume the requests that are waited to be handled. Passively dropped or discarded requests would be misunderstood as failed. Too many failed probes may alter the network topology and ultimately degrade the performance.

For product testers:

1) The DUTs passing laboratory tests are not guaranteed to be free from failures. Replay using real-world traffic can be used to complement the insufficiency in using artificial traffic for the diversity and completeness of prerequisite test cases. Replaying using artificial traffic is suitable for specified or small-scale coverage because it can respond in a short time; replaying using real-world traffic is suitable for large-scale or comprehensive tests because it can trigger more logic or consequent flaws that are not easy to find.

2) Replaying real-world traffic is not particularly targeted, which is a comprehensive test for the DUTs. However, replaying using the original raw traces captured from the real world wastes storage and is time-consuming. BD can be invoked to quickly downsize the raw traces while retaining the triggered failures.

3) Bug traces can be used to efficiently reproduce similar failures. For the DUTs of the same family, utilizing bug traces can save a lot of testing time.

Based on lesson 2 for developers, we found an interesting open research question on how to design a protection mechanism for advanced features in resource-limited devices. Due to resource limitation, if the operations on advanced features are not carefully designed, it is very possible to affect the performance or stability of the system and even crash the system in the end. For example, if the resource allocation or recovery operations cannot operate well for a networking device, after frequently interleaved invocations of the two kinds of imperfect operations, the accumulated side-effects, like resource leakage, would finally crash the system. Hence, the question of how to design a mechanism that can ensure the execution correctness of each operation to protect the system is worth exploring.

Acknowledgments

This work was supported in part by the National Science Council, and in part by ZyXEL Inc. and D-Link Corp. The authors would also like to thank the staff engineers at Network Benchmarking Lab (NBL, www.nbl.org.tw) for their continuous laboratory technical support.

References

[1] Y.-D. Lin *et al.*, "On Campus Beta Site: Architecture Designs, Operational Experience, and Top Product Defects," *IEEE Commun. Mag.*, vol. 48, no. 12, Dec. 2010.
[2] M. K. Daskalantonakis, "A Practical View of Software Management and Implementation Experiences within Motorola," *IEEE Trans. Software Engineering*, vol. 18, no. 11, Nov. 1992, pp. 998–1010.
[3] Alessio Botta, Alberto Dainotti, Antonio Pescape, Giorgio Ventre, "Reducing Network Traffic Data Sets," IEEE ICC, 2007.
[4] A. Pescape, "Entropy-Based Reduction of Traffic Data," *IEEE Commun. Letters*, vol. 11, no. 2, Feb. 2007, pp. 191–93.
[5] Y. Liu *et al.*, "An Information-theoretic Approach to Network Monitoring and Measurement," *Proc. 5th ACM SIGCOMM Conf. Measurement*, 2005.
[6] G. Iannaccone *et al.*, "Monitoring Very High Speed Links," *Proc. 1st ACM SIGCOMM Wksp. Internet Measurement*, 2001.
[7] M. Peuhkuri, "A Method to Compress and Anonymize Packet Traces," *Proc. 1st ACM SIGCOMM Wksp. Internet Measurement*, 2001.
[8] Y.-D. Lin *et al.*, "In-Lab Replay Testing with a Case Study on SOHO Router," submitted to *J. Internet Tech.*, available upon request.
[9] Y.-D. Lin *et al.*, "Low-Storage Capture and Loss-Recovery Selective Replay of Real Flows," *IEEE Commun. Mag.*, vol. 50, no. 4, Apr. 2012, pp. 114–21.
[10] Network Benchmarking Lab, http://nbl.org.tw.
[11] TCPReplay, http://tcpreplay.synfin.net/.
[12] DD-WRT, http://www.dd-wrt.com/.
[13] Netfilter, http://www.netfilter.org.
[14] PCAP Lib, http://security.nbl.org.tw.

Biographies

Ying-Dar Lin [F'13] is a Distinguished Professor of computer science at National Chiao Tung University (NCTU), Taiwan. He received his Ph.D. in computer science from the University of California at Los Angeles (UCLA) in 1993. He served as a visiting scholar at Cisco Systems, San Jose, California, during 2007–2008. Since 2002, he has been the founder and director of the Network Benchmarking Lab (NBL), which reviews network products with real traffic. His research interests include quality of services, network security, deep packet inspection, P2P networking, and embedded hardware/software co-design. His work on multihop cellular was the first along this line, and has been cited over 600 times and standardized into IEEE 802.11s, WiMAX IEEE 802.16j, and 3GPP LTE-Advanced. He is an IEEE Distinguished Lecturer for 2014 and 2015, and currently on the editorial boards of several IEEE journals and magazines. He published a textbook, *Computer Networks: An Open Source Approach*, with Ren-Hung Hwang and Fred Baker (McGraw-Hill, 2011).

Chun-Nan Lu (cnlu@cs.nctu.edu.tw) received his B.S. and M.S. degrees in computer science from National Tsing Hua University in 2000 and 2002. He is a Ph.D. student in computer science at National Chiao-Tung University. His research focuses on network security and traffic measurement/analysis.

Yuan-Cheng Lai (laiyc@cs.ntust.edu.tw) received his Ph.D. degree in computer Science from NCTU 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 content networking.

Zongo Pawendtaore Eliezer (zepaw@hotmail.com) received his M.S. in computer science from the Department of Computer Science of NCTU.

If the resource allocation or recovery operations cannot operate well for a networking device, after frequently interleaved invocations of the two kinds of imperfect operations, the accumulated side effects, like resource leakage, would finally crash the system.