

國立交通大學

電機資訊國際學程

碩士論文

識別跟縮減觸發網通裝置錯誤的網路流量

Identifying and Downsizing Packet Traces Triggering Defects in
Networking Devices

研究生：鍾艾利

指導教授：林盈達 教授

中華民國一百零一年六月

識別跟縮減觸發網通裝置錯誤的網路流量

**Identifying and Downsizing Packet Traces Triggering Defects in
Networking Devices**

研 究 生：鍾艾利 **Student : Pawendtaoré Eliézer ZONGO**

指導教授：林盈達 **Advisor : Dr. Ying-Dar Lin**

國 立 交 通 大 學

電機資訊國際學程

碩 士 論 文

A Thesis

Submitted to EECS International Graduate Program

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master

June 2012

Hsinchu, Taiwan, Republic of China

中華民國一百零一年六月

識別跟縮減觸發網通裝置錯誤的網路流量

學生：鍾艾利

指導教授：林盈達

國立交通大學電機資訊國際學程

摘 要

在將網路設備公開到市場上之前，測試是一種能夠保證產品品質與穩定性的方法，而其中一種測試網路設備的方式是透過重播人工或是真實世界的網路流量來進行測試，相較於使用人工網路流量，透過真實流量的重播測試可以呈現出更多真實的特性。然而大量的真實網路流量及長時間的重播測試，為了減少重播時間及能有效率的重現待測產品的錯誤，如何縮減產生網路產品錯誤的流量是必須面對的問題。我們提出了透過線性搜尋與二元搜尋演算法的線性縮減與二元縮減演算法來進行流量長度的縮減。為了評估我們的方法對流量縮減的有效性，我們使用縮減比率來做為評估單位。從錯誤的分佈評估來看，在所有的錯誤中，ICMP 和 HTTP 造成的錯誤占了所有網路設備錯誤的 60%，另外，我們提出的線性縮減與二元縮減分別對於流量達到了 77%與 80%的縮減比率，根據結果來看，測試的時間也依照同樣的比例減少，證明我們的縮減方法可以對於網路產品測試的效率有大幅度的改善。

關鍵字：網絡設備，缺陷，網路流量，縮減

Identifying and Downsizing Packet Traces Triggering Defects in Networking Devices

Student: Zongo Pawendtaoré Eliézer Advisor: Dr. Ying-Dar Lin

Department of Electrical Engineering & Computer Science / International
Graduate Program

National Chiao Tung University

Abstract

Testing networking devices before releasing them to the market is a way of ensuring their quality and robustness. Replay technique with artificial or real-world traffic is a method used to test networking devices. Using real-world traffic is more preferable as it displays more realistic properties in comparison with artificial traffic. The challenges with testing products with real-world traffic are mainly the high volume of the captured traces and the prolonged time required for the replay testing. In order to efficiently reproduce the defects produced by networking devices and reduce the replay time, it is necessary to reduce the size of the traces that trigger the defect of networking devices. In order to perform a defect-based traces downsizing, we propose a binary and a linear downsizing algorithms respectively based on binary and linear search algorithms. A metric called downsizing ratio is defined in order to evaluate the efficiency of the traces downsizing. The evaluation of the defects distribution show that ICMP and HTTP defects represent around 60% of the total number of defects experienced by networking devices. In addition, the binary downsizing and linear downsizing achieve a downsizing ratio of 77% and 80% respectively. As a result, the time needed to perform testing is reduced by the same amount. This is significant since the downsizing techniques used can improve the testing of networking devices.

Keywords: networking devices, defects, packet traces, downsizing

Table of Contents

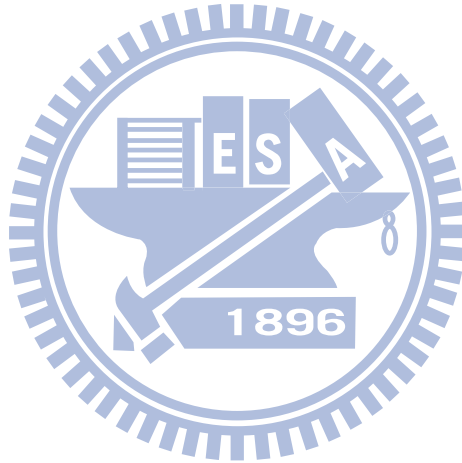
Abstract.....	ii
Table of Contents	iii
List of Figures.....	iv
List of Tables	v
Chapter 1 Introduction	1
Chapter 2 Background	3
2.1 Issues of Traffic Capture and Replay	3
2.2 Overview of Stability Testing	4
2.3 Related Works	5
Chapter 3 Traces Downsizing	7
3.1 Terminologies Definitions	7
3.2 Problem Statement	8
3.3 Issues to Be Addressed.....	8
Chapter 4 Packet Traces Identification and Downsizing	10
4.1 Solution Approach Overall Architecture	10
4.2 Traces Identification.....	11
4.3 Traces Downsizing Procedures	12
Chapter 5 Experiments Studies and Results Analysis.....	15
5.1 Traces Selection and Experimental Testbed	15
5.2 Defects Distribution	17
5.3 Downsizing Ratio (DR).....	20
5.4 Replay Throughput vs. Defects Occurrence	22
5.5 Investigating the Determining Causes of Defects: Case Study	24
Chapter 6 Conclusions and Future Works.....	26
References	28

List of Figures

Figure 1: Captured traces format	3
Figure 2: Stability Testing Architecture and Test-bed.....	4
Figure 3: Traces Collection and Device Testing Flowchart	5
Figure 4: General Solution Approach	10
Figure 5: Sample recorded logs	11
Figure 6: Linear Downsizing Flowchart and pseudocode	13
Figure 7: Binary downsizing algorithm illustration and pseudocode	14
Figure 8: Configuration of the CheckDevice	17
Figure 9: Overview of DUT defects	18
Figure 10: Defects distribution	19
Figure 11: BD and LD Illustrations	21
Figure 12: Two methods of DR evaluation.....	21
Figure 13: Comparison of two downsizing algorithms DR.....	22
Figure 14: Downsizing algorithms: iterations and time.....	22
Figure 15: Replay throughput and Defects occurrence.....	23
Figure 16: Log messages from a DUT running an open-source firmware	25
Figure 17: DUT to web interface messages.....	25

List of Tables

Table 1: Notations Description	7
Table 2: Sample packet traces selection	15
Table 3: Downsized traces	18
Table 4: Replay throughput and DUT failure	23



Chapter 1 Introduction

Networking devices are critical tools as they enable and support information sharing between multiple systems. Testing is one of the methods used to discover product defects in the early stage of development, and aims at reducing the number of defects found by customers after sale [1]. Even though they undergo and pass a series of tests during their development, networking devices still reveal customers found defects when they are put under real-world conditions. This could be explained by the fact that laboratory testing often uses artificial traces which have less realistic properties, diversity and complexity in terms of applications and protocols [2]. For instance applications such as peer-to-peer (P2P), video streaming services, on-line games, etc. often have proprietary protocols, which are hard to model by generated traffic.

Replay techniques fundamentally aim at reproducing and debugging the real-world defects of devices in a controlled environment. In order to improve their accuracy and efficiency, networking devices ought to be tested using real traffic rather than artificial one because there is a higher chance to trigger more device bugs or defects that would not be easily discovered when using artificial traffic.

The major limitations of real traffic replay are twofold: high volume of captured traces and time consumption [3]. For example, during the peak hour, the captured network traffic volume at National Chiao Tung University's Betasite [1] could reach 60 gigabytes (GB) in 30 minutes and even during the regular hours around midnight, the traffic volume would still reach at least 20GB every 30 minutes. Note that replaying several hundreds of gigabytes could take up to 120 hours. Moreover, reproducing a defect triggered by high volume of traffic is time-consuming. Thus, for more efficiency in the testing of networking devices, the downsizing of packet traces turns out to be a very crucial and imperative operation.

However it is not trivial to reduce the size of traces with high volume while maintaining authentic and reliable information capable of reproducing the very same defects triggered by the raw traces. In fact defects-driven traces downsizing is challenging since the downsized traces should be representative of the original entire traces. In addition downsized traces should be able to ensure defect reproduction,

reduce testing time and make device testing more accurate and efficient.

There has been some works on network traffic data reduction or compression. Some traces reduction approaches were based on the concepts of entropy and aimed at reducing traces [3,4] with the objective of improving traffic trace analysis by removing data redundancy in a large trace especially at packet-level. While those two works focused on traffic data reduction, [5-8] emphasized instead on the compression of packet headers. Their goal is to apply the trace compression technique to online packet headers, allowing communication over low bandwidth channels. The purpose of these works is to reduce traces collection storage. While these works help in reducing the volume of network traffic traces, it is important to note that none of them perform a defect-driven traces downsizing.

In our defect-driven traces downsizing approach, we would differentiate the defects that occur with different types of devices. The downsized triggering traces would be evaluated by measuring the downsizing ratio (DR). Variations of DR with respect to different replay throughputs will be highlighted since the Internet throughput varies as well on the real-world environment. The possible causes of the networking devices failure should be addressed by doing a forensic investigation on the downsized traces.

We proceed by first identifying among the raw large traces the ones that trigger devices defects. The testing technique we are using is based on that of In-Lab Real Testing (ILRT) [15] where the devices under test defects are constantly monitored during the replay testing. Once a defect is triggered, the identified traces are downsized using a Linear Downsizing algorithm or a Binary Downsizing algorithm. Linear downsizing algorithm is done through incremental rollback and replay technique. Binary Downsizing is based on binary search algorithm [18], and is used in complement with linear downsizing. Our final step consists on splitting and classifying traces according to the defects triggered.

The remainder of this paper is structured as follows. Chapter 2 presents the background and related works. Chapter 3 describes some terminologies and discusses the problem statements. Chapter 4 describes the design and algorithm of our proposed solution. The experimental results of our works are studied and evaluated in Chapter 5. Finally, in Chapter 6 we draw some conclusions based on our findings and discuss the future works.

Chapter 2 Background

This chapter underlines the issues of traffic capture and replay, as well as an overview of stability testing; finally it discusses the related works.

2.1 Issues of Traffic Capture and Replay

Testing networking devices with real-world network 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 the efficiency of tests performed on products or experimental studies [14]. There exists an environment within National Chiao Tung University campus where the network traffic produced by volunteers is captured. This environment called Beta Site [1] pursues the goal of evaluating products performance with real traffic. Beta Site provides an incentive for students to be volunteers and the daily-generated traffic is captured in files with PCAP format. PCAP stands for "Packet Capture" and is a common packet capture data format. In order to provide a database of traffic traces that can be used for products testing or evaluation, the captured traces in the format of PCAP files are stored in a Pcap Library [13]. However storing multiple PCAP files has a drawback due to the high volume of the traces. In addition replaying large volume of packet traces for products events or defects reproduction is highly time-consuming as mentioned earlier in the introductory chapter. For event analysis, the smaller the volume of traffic replayed to reproduce events, the easier and faster their analysis. [14]. Figure 1 displays a list of captured traces in the format of PCAP files. The nomenclature of the PCAP files is based on the date and the time of capture as follows: ddmmyy-hhmn.pcap. This procedure of assigning names to PCAP files helps in distinguishing traces.

```
total 1.1T
48G 100614-1710.pcap  60G 100614-2040.pcap  61G 100615-0040.pcap  14G anon-100213-0010.pcap  15G anon-100213-0240.pcap
52G 100614-1810.pcap  50G 100614-2110.pcap  61G 100615-0110.pcap  9.5G anon-100213-0040.pcap  15G anon-100213-0310.pcap
63G 100614-1840.pcap  56G 100614-2140.pcap  55G 100615-0210.pcap  8.1G anon-100213-0110.pcap  15G anon-100213-0340.pcap
62G 100614-1910.pcap  58G 100614-2210.pcap  56G 100615-0240.pcap  15G anon-100213-0140.pcap  11G anon-100213-0410.pcap
55G 100614-2010.pcap  58G 100615-0010.pcap  46G 100615-0310.pcap  16G anon-100213-0210.pcap  15G anon-100213-0440.pcap
nbl@nbl-replay-01:/media/1e5613c4-40cb-4c96-bbb7-0866a6aad342/100614$
```

Figure 1: Captured traces format

2.2 Overview of Stability Testing

Networking devices even after they pass the lab testing still face defects challenges in the real-world environment. The defects found after the release of the devices to the market are called customer found defects (CFD). ILRT [15] undertakes testing on Small Office Home Office (SOHO) devices at the Network Benchmarking Lab (NBL) [17]. SOHO routers are stateful devices running NAT functions. The replay tool used to test SOHO routers is a modified version of Tcpreplay [16] with the NAT functions in such a way that it can handle the address and port translation within the device. Stability testing consists of replaying network traffic to devices under test (DUT) while monitoring them with a tool called *CheckDevice*. The *CheckDevice* is a probing tool that continuously sends ARP, HTTP and ICMP queries to DUT. During the testing of networking devices, real traffic traces are replayed to multiple DUTs. Each DUT is connected to a *replayer* (a server that hosts NATReplay and send the traces to the DUT) and a *CheckDevice*.

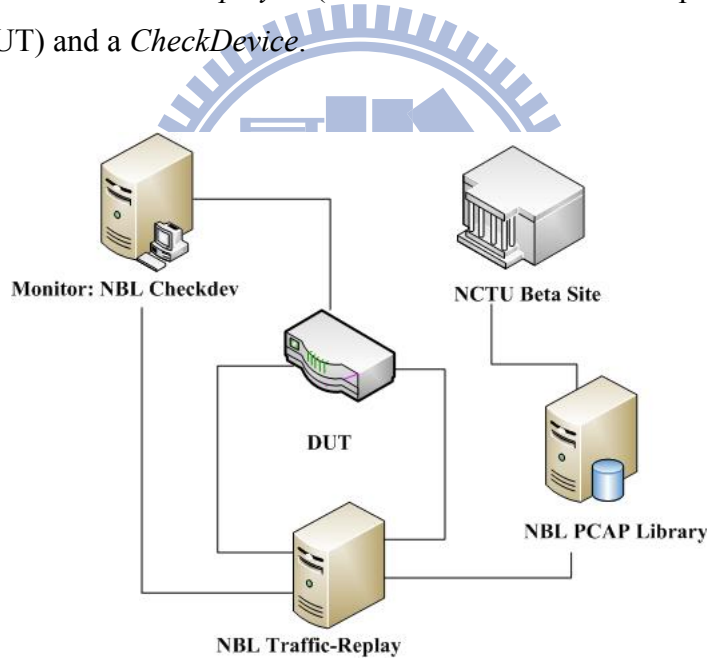


Figure 2: Stability Testing Architecture and Test-bed

Note that the replayed traces are stored in a raw format, meaning that they are not processed or modified. Figure 2 displays the stability testing architecture used by ILRT. The testing environment involves NCTU Beta Site where the network traffic is captured and NBL PCAP Library where the captured traces are stored. Figure 3 portrays the traces collection and the networking devices testing procedure. First, the traces used for replay are captured from the NCTU Beta Site. The traces are then replayed to multiple DUTs. In Figure 3, n represents the number of failed devices.

Thus, whenever two or more DUTs hang, the replayed traffic traces are marked and collected to a repository. These traces called *bug traces* are used to reproduce a detected defect or a malfunction of devices.

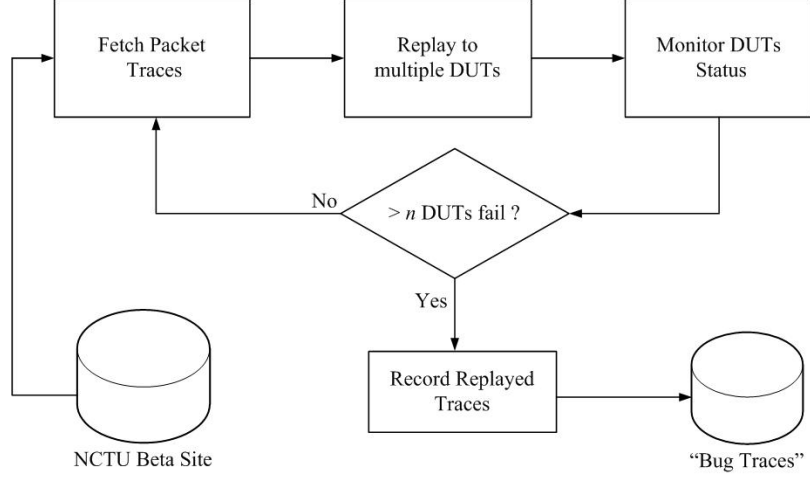


Figure 3: Traces Collection and Device Testing Flowchart

2.3 Related Works

There exist several studies related to network traffic traces reduction. They aim at different objectives and their approaches are diverse.

The main objective of [3] consists of improving data collection and analysis on one hand and reducing original data set with an acceptable loss of properties on the other. It proposes an off-line 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 off-line technique to reduce traffic trace data sets presents the advantage of correctly capturing mean, standard deviation, and marginal distributions of network traffic traces, without compromising time properties.

The study in [5] suggests a compression algorithm that removes this redundancy in order to produce substantially smaller traces. A method to compress and anonymize packet traces [8] 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 described works all share the objective of either reducing or compressing traces in order to either improve data storage or reduce the complexity of traffic traces analysis. However, these works do not meet one of the requirements of our study; their traces reduction is not defect-driven as we are proposing. The purpose of our work first consists of identifying traces that triggers defects of networking devices, then downsizing the traces in order to reduce their volume. Therefore it is a defect-driven downsizing of defectuous traffic traces.



Chapter 3 Traces Downsizing

This chapter highlights the definitions of terminologies specific to our work and discusses the problem and issues that need to be addressed.

3.1 Terminologies Definitions

Terminologies that are specific to our work will be defined according to the context in which they will be used. *Triggering traces identification* is referred to as the process of locating in a bunch of network traffic the traces that are responsible or causing the defects of the networking devices. Triggering traces identification often results in a set of network traffic that is high in volume. *Traces downsizing* is a process that consists of reducing the volume of the traces while keeping the information or characteristics capable of inducing the same defects as the raw traces. One of the objectives of our work is to categorize the traces based on the defects they trigger. In order to achieve this, we perform *traces splitting* by dividing a single trace into multiple traces based on the triggered defects. Table 1 is a description of a list of notations that are further used in our work. P refers to packet traces in PCAP format and L represents the logs that are triggered by the DUTs. P_j and L_i are used to refer to the j -th packet trace and the i -th log respectively. $P_j^{L_i}$ represents packet trace j that have triggered a device defects L_i , but that are not yet downsized. $DP_j^{L_i}$ is the downsized version of $P_j^{L_i}$.

Table 1: Notations Description

Notations	Description
P	Packet trace in a pcap format.
L	Log or defect produced by a device under test.
P_j	j -th packet trace.
L_i	i -th log.
$P_j^{L_i}$	j -th packet trace that triggers the i -th log
$DP_j^{L_i}$	Downsized j -th packet trace that triggers the i -th log

3.2 Problem Statement

Considering the defects in the form of logs produced by networking devices, efficient investigation and testing of these devices require the identification of the suspicious traces. However, as stated in chapter 1 the triggering traces are voluminous, making their analysis or defects reproduction through replay more complex because of the high demand in time and resources.

Therefore, given a set of packet traces and a set of the defects in the form of logs of the networking device, we aim at attaining the following objectives: (1) for each log, identify the triggering traffic traces and (2) downsize the triggering traffic traces in the way that preserve the same characteristics necessary to reproduce the defects.

Hence, the problem could be defined as follows.

Given

- (1) a set of packet traces $\{P_j, j=1\dots m\}$ and
- (2) a set of defectuous logs $\{L_i, i=1\dots n\}$ produced by networking devices,

suppose

- (1) L_i is the i -th log triggered by a device,
- (2) P_j is the j -th packet traces,
- (3) $P_j^{L_i}$ is the j -th packet trace that triggers the i -th log
- (4) $DP_j^{L_i}$ is the downsized trace from $P_j^{L_i}$,

the objectives are

- (1) for each log L_i triggered by a device, identify $P_j^{L_i}$ such that $P_j^{L_i}$ triggers L_i ,
- (2) for each $P_j^{L_i}$, find $DP_j^{L_i}$ such that $DP_j^{L_i}$ triggers L_i .

3.3 Issues to Be Addressed

Defects distribution

The first issue regarding our work consists of determining the defects distribution. This issue aims at differentiating the defects that are produced by networking devices and by analyzing their recurrence. In other terms, we will look at the diversity of defects and the number of times they are produced by networking devices. Defects distribution is justified, as it is a means that could reveal the common defects that are produced by networking devices.

Downsizing ratio

The downsizing ratio (DR) is the relationship between the size of reduced traces and that of the raw traces. It is an expression of the number of times by which raw traces have been reduced. Since the main objective of our work consists of reducing packet traces, to achieve an optimal downsizing, we introduce the DR as a metric to evaluate the downsizing process.

Different networking devices differ in the way they are implemented. They might not function the same way under the same conditions, thus would not produce the same defects under the same conditions of usage. It is therefore important to measure the variation of the DR with respect to DUTs from different vendors.

Replay throughput vs. defects occurrence

This issue aims at determining the different behavior of DUTs under different replay throughput. Device under test when used in real-world conditions encounter high and low bandwidth conditions. In the same way, we ought to test the devices using different throughputs and observe their performance with low and high replay throughputs.

Factors determining causes of devices defects

Finally, a very important aspect worth investigating is the analysis of the causes of devices defects. Packet traces characteristics or properties [9,10] would trigger defective behavior of the networking devices. For instance traffic such as P2P that generates multiple concurrent connections often cause a burden on the devices in term of resources allocation thus affecting their stability. We would look at the behavior of the networking devices as they are handling the real-world traffic traces. Since some networking devices are black boxes, meaning that they do not disclose any information about the causes of their failure, we would perform some simulation on devices with open-source firmware in order to investigate the causes of defects.

Chapter 4 Packet Traces Identification and Downsizing

This chapter discusses the solution approach necessary for packet traces identification, downsizing and splitting. In addition, it discusses the forensic analysis of the downsized traces in order to find the possible causes of device failure.

4.1 Solution Approach Overall Architecture

As illustrated in Figure 4, the design of our solution is composed of two major components. The first component is the *traces identification*. During the testing of networking device using replay technique, the traces that trigger the DUT failure are marked and collected. The second component is the *traces downsizing* where two downsizing algorithms alternatives, namely Linear Downsizing (LD) algorithm and Binary Downsizing (BD) algorithm are proposed. The purpose of traces downsizing is to reduce original traces to smaller traces that are still able to trigger the same failures as the original ones. After the packet traces downsizing, a repository of downsized defect traces is collected and validated.

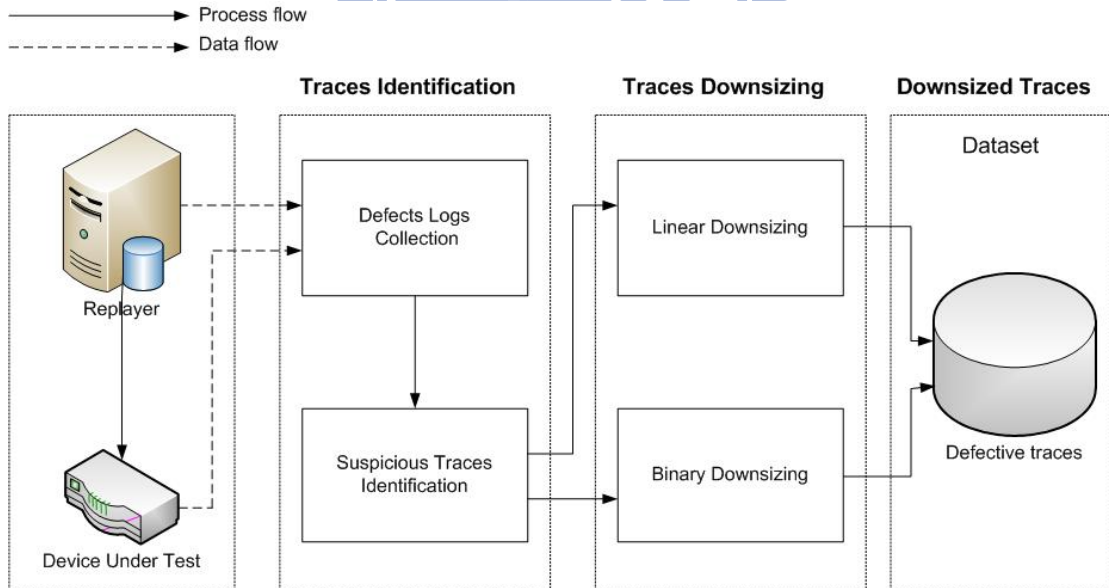


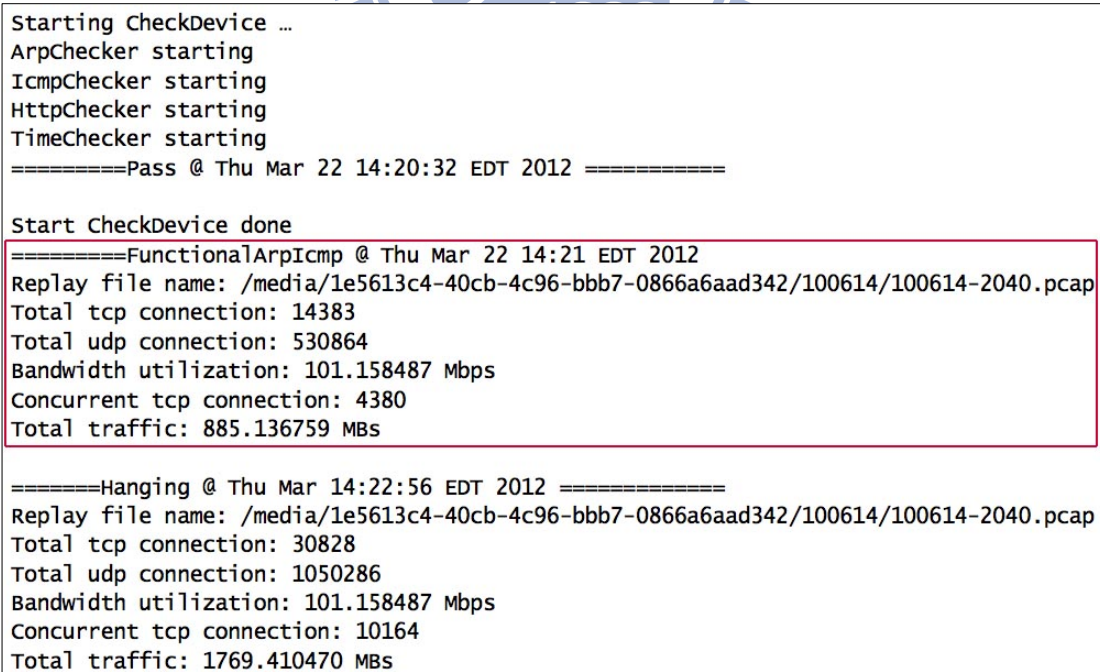
Figure 4: General Solution Approach

After these steps are completed, we perform an analysis of the downsized traces. This is an investigation process in order to depict the properties in the traces that led to the failure of the networking devices.

4.2 Traces Identification

Traces identification is the discovery among multiple replayed traces those triggering defects on the device under test. The goal of traces identification is to obtain $P_j^{L_i}$ from P . Traces identification is useful because it is the first step towards the downsizing of packet traces as it extracts the suspicious traces $P_j^{L_i}$ from raw network traffic. Traces identification process works by replaying network traffic to a DUT. As the traffic is replayed, the status of the DUT is monitored and the failures that it triggers are recorded. Queries of type ICMP, ARP and HTTP are regularly sent to probe the DUT status. When the DUT replies to each of these individual queries within the response time constraint and continuously handles incoming and outgoing packets, it is considered to be working properly. Otherwise it is regarded as failed or crashed. In fact the proper reception of requests and transmission of responses require a good physical connection to the device as well as a properly configured and operating TCP/IP stack.

In case of the device failure, we record the replayed traces information through the monitoring system and the identified traces go through the downsizing process.



```
Starting CheckDevice ...
ArpChecker starting
IcmpChecker starting
HttpChecker starting
TimeChecker starting
=====Pass @ Thu Mar 22 14:20:32 EDT 2012 =====

Start CheckDevice done
=====FunctionalArpIcmp @ Thu Mar 22 14:21 EDT 2012
Replay file name: /media/1e5613c4-40cb-4c96-bbb7-0866a6aad342/100614/100614-2040.pcap
Total tcp connection: 14383
Total udp connection: 530864
Bandwidth utilization: 101.158487 Mbps
Concurrent tcp connection: 4380
Total traffic: 885.136759 MBs

=====Hanging @ Thu Mar 14:22:56 EDT 2012 =====
Replay file name: /media/1e5613c4-40cb-4c96-bbb7-0866a6aad342/100614/100614-2040.pcap
Total tcp connection: 30828
Total udp connection: 1050286
Bandwidth utilization: 101.158487 Mbps
Concurrent tcp connection: 10164
Total traffic: 1769.410470 MBs
```

Figure 5: Sample recorded logs

Figure 5 is an illustration of the recorded logs during the failure of a device under test. The recorded information includes the type of defect that occurred, the information about the traces that are involved as well as the defect occurrence time. It also displays details about the traces number of connections, the total size of replayed traffic. This information is used to find and extract the suspicious traces from the raw traces.

4.3 Traces Downsizing Procedures

As stated in the previous chapters, the traces downsizing purpose is to have traces with reduced size that can trigger the defect in a reduce amount of time. In fact it consists of obtaining $DP_j^{L_i}$ from $P_j^{L_i}$.

Linear downsizing algorithm

There are two concepts involved in the Linear downsizing algorithm: reverse order and rollback-and-replay. In the process of traces downsizing, when a defect is triggered after the replay of traces, $P_j^{L_i}$ is replayed in a reverse and decreasing order. It means that the replay testing starts by replaying the last replayed traces towards the first ones. The second concept involved in LD is the rollback-and-replay, which consists of including an amount of traces each time the replayed traces failed to trigger a defect. Figure 6 shows a flowchart and pseudocode that describes the Linear Downsizing process. There are three steps involved in this flowchart namely rollback, replay and recording. Rollback consists of replaying some amount of the traces that were replayed just before the defect occurred. The rollback size is determined according to the target downsized trace size. When a defect L_i is triggered during the replaying test, the first step towards downsizing is rollback. If these packets do not trigger the defect, we include the adjacent packet traces and redo the replay testing. We keep incrementing packet traces until the defect is triggered. When the defect L_i is triggered, we complete the downsizing process and record these traces as the downsized traces $DP_j^{L_i}$.

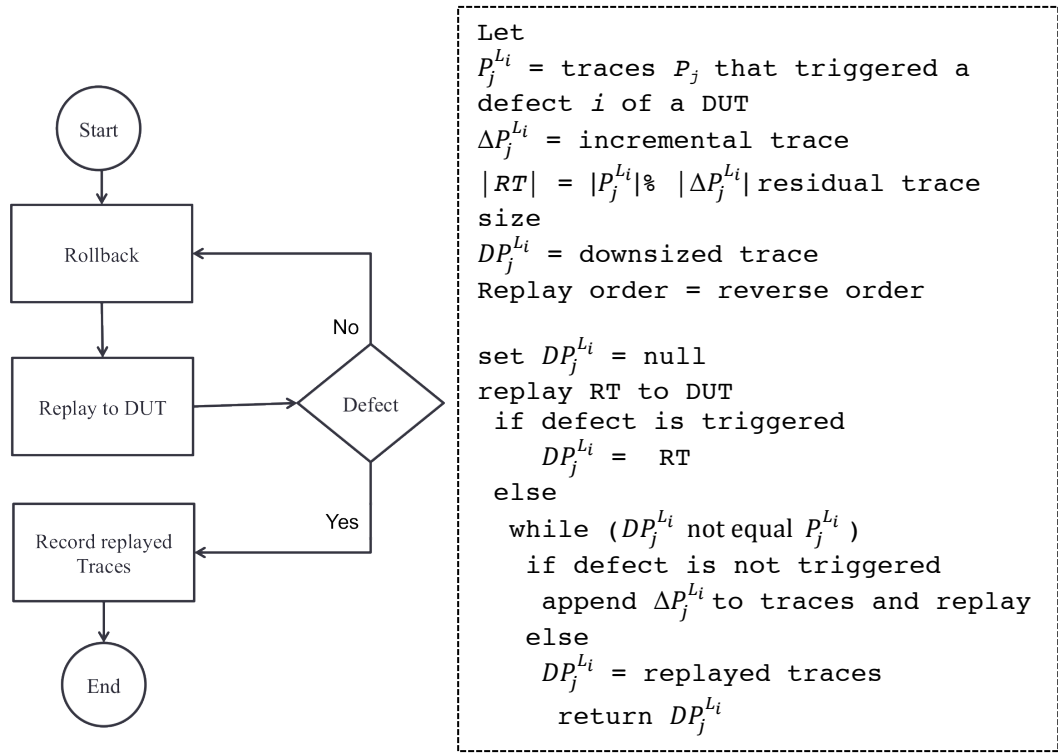


Figure 6: Linear Downsizing Flowchart and pseudocode

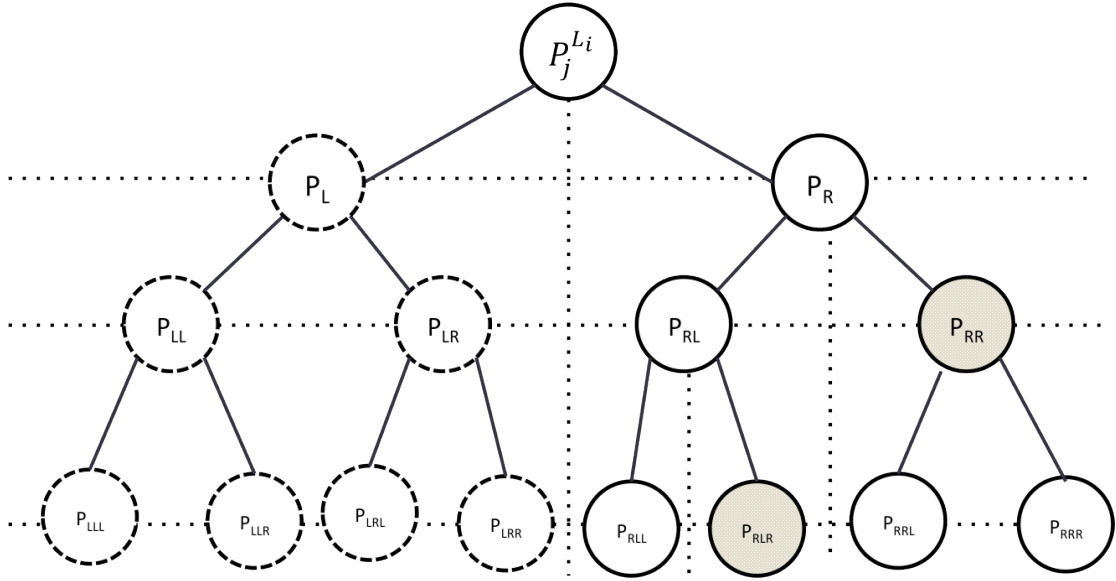
The linear downsizing algorithm is likely to be advantageous if the packet traces that triggered the defect belong to last replayed traces during the test. However when the triggering traces do not belong to the traces replayed at the beginning of the test, an alternative, which is the binary downsizing algorithm, is proposed.

Binary downsizing algorithm

Binary downsizing algorithm is based on binary search algorithm and consists of locating the traces that trigger a device defect by recursively splitting P_j^{Li} in halves and testing the device with each half until the defect is reproduced.

Figure 7 is an illustration and pseudocode of the binary downsizing algorithm. Considering P_j^{Li} as traces that triggered a device defect, we will describe how to downsize P_j^{Li} using the binary downsizing algorithm in order to obtain DP_j^{Li} .

First of all, we split P in half and generate two sets of traces P_R and P_L with the same size. P_R and P_L respectively represent the first and second half of P . After the splitting, P_R is first replayed to the DUT because it is more likely that the traces that trigger the defect belong to the last replayed traces. If P_R triggers the defect of the DUT we further split P_R into two halves where P_{RR} and P_{RL} are respectively the first and second half. As in the previous case, we replay P_{RR} to the DUT.



(a) Binary downsizing illustration

```

Let
 $P_j^{Li}$  = traces that triggered a defect of a DUT
 $\Delta T$  = minimum desired DT
Split  $P_j^{Li}$  in two halves: right_node and left_node
While traces to replay >  $\Delta T$ 
  Replay right_node to DUT
  if defect is triggered
    split traces and replay right_node
     $DP_j^{Li}$  = replayed traces
    return  $DP_j^{Li}$ 
  else
    append previous node right_node

```

(b) Binary downsizing pseudocode

Figure 7: Binary downsizing algorithm illustration and pseudocode

If the replay of P_{RR} does not trigger the device defect, it means that P_{RR} alone cannot trigger the device defect. Therefore, we need to include additional traces to reproduce the defect and we include the second half of P_{RL} , which is P_{RLR} . In this case the replay input is the addition of P_{RLR} and P_{RR} .

In the binary downsizing process, the half-splitting of traces and their replay are continuously done until reduced traces that can trigger the device defect are obtained.

Chapter 5 Experiments Studies and Results Analysis

In this chapter, the experiments conducted to identify the traces that trigger the defects of networking devices are discussed. The outcome of the two downsizing algorithms is compared and a case about a networking device behavior during a hanging defect is studied.

5.1 Traces Selection and Experimental Testbed

Sample traces

To conduct the experiments, we used the network traffic captured from NCTU Beta Site. As mentioned earlier in section 2.2, ILRT testing is performed using real traffic captured by NCTU Beta Site and the recording of suspicious traces is done whenever at least two different types of DUTs failed the test. The recorded traces are stored in a repository of *bug traces*. For our experiment, we selected from the repository sample traces based on their size and the number of failed devices. Our selected sample is a set of 34 PCAP files with a total size of 1.2 TB and causing the failure of 4 DUTs. Table 2 lists a portion of the selected and the 4 commercial DUTs that are used. For the following downsizing process we labeled these DUTs as DUT₁, DUT₂, DUT₃ and DUT₄.

Table 2: Sample packet traces selection

No	File Name	Size	DUTs used
1	100614-1710.pcap	48 GB	DUT ₁ DUT ₂ DUT ₃ DUT ₄
2	100614-1810.pcap	52 GB	
3	100614-1840.pcap	63 GB	
4	100614-1910.pcap	62 GB	
5	100614-2010.pcap	55 GB	
6	100614-2040.pcap	60 GB	
7	100614-2110.pcap	50 GB	
8	100614-2140.pcap	56 GB	
...	
Total size of Traces		1.2 TB	

Experimental testbed

The experimental testbed used is based upon that of ILRT mentioned in Figure 2 in Chapter 2 and mainly comprises a replayer, a DUT and a monitoring tool called *CheckDevice*.

The replayer replays the captured traffic traces to the DUT. The CheckDevice monitors both the replayer and the DUT. Throughout our experiments the sample traces are replayed using a replay throughput of 50 mbps, as it is the case of ILRT environment. However, we use a replay throughput of 100 mbps in our experiments to find the behavior of the DUT under higher throughput.

Overview of the CheckDevice

The CheckDevice is a tool developed by NBL and is used in networking devices testing. It has two main functions. One is that it is used to configure the replayer and the DUT parameters, and the other is that it monitors the status of the replayer and DUT by regularly sending queries to it.

As shown on Figure 8, for the configuration of replay parameters, the CheckDevice is used to set the DUT and replayer IP address, MAC address and the DUT LAN/WAN addresses. In addition it specifies the PCAP path, which is the directory of the traces to be replayed, sets the replay throughput and enables other parameters configuration on the replayer side. The CheckDevice also sets the check interval time, which is the DUT timeout in milliseconds (ms), the execution time and the hanging time tolerance. The hanging time tolerance is the time allowed to the DUT for recovery before stopping the replay testing. For our test, we re-use the settings of ILRT, meaning that the timeout is set to 10,000 ms, the check interval to 2,000 ms and the hanging time tolerance to 3600 s.

The second function of the CheckDevice consists of monitoring the DUT by simultaneously sending three requests namely ARP, ICMP and HTTP requests to the DUT in order to probe its status. The DUT would be regarded as normal if it can reply to all requests within the predefined timeout time; otherwise the DUT is more likely to be experiencing defects.

The screenshot shows the 'CheckDevice' application window. It has a menu bar with 'File', 'View', and 'About'. Below the menu bar is a 'Configuration View' section with a 'Common Config' tab. The 'Common Config' tab contains several input fields and buttons:

- Interface:** A dropdown menu showing 'eth0'.
- Ip Address:** A text field containing '192.168.108.100' and a 'Change Ip Address' button.
- Dut Ip Address:** A text field containing '192.168.103.1'.
- Dut Mac Address:** A text field containing 'e0:cb:4e:92:f5:98' and a 'Get Mac Address' button.
- Timeout(milli-second):** A text field containing '10000'.

Below the 'Common Config' tab is a 'NatReplay' tab. It contains a 'Compatible Mode' checkbox (unchecked) and a grid of configuration fields:

Replayer Ip Address	192.168.103.101	Replayer Port	799
Replay Speed(-r)	100	Avoid Use Ip Address(-7)	
Lan Interface(-i)	eth1	Wan Interface(-j)	eth2
Dut Lan Mac Address(-l)	e0:cb:4e:92:f5:98	Dut Wan Mac Address(-j)	e0:cb:4e:92:f5:99
Cidr(-C)	140.113.0.0/16	Pcap Path	366a6aad342/100614
Wan Ip Address(-5)	10.10.103.1	Lan Network(-5)	192.168.103.0/24
Lan Mac Address(-k)	50:67:f0:5e:c2:3d	Wan Mac Address(-k)	00:22:b0:52:be:c7
<input checked="" type="checkbox"/> Readinfo Port	800		

At the bottom of the 'NatReplay' tab are 'Start' and 'Stop' buttons.

Figure 8: Configuration of the CheckDevice

5.2 Defects Distribution

As shown in Figure 9, there can be distinguished three types of queries: ARP request, ICMP request and HTTP request. Based on the different combination of responses from the DUT, we differentiated 7 different types of device defects.

We classified these defects into 3 groups: single-defect, combined-defect and hanging. A single-defect occurs when the DUT is unable to respond to 1 out of the 3 requests; in the combined defect situation the DUT is unable to respond to 2 out of 3 request, and finally, a hanging occurs when all the 3 requests are not replied to. The single-defect group is composed of an ARP, ICMP and HTTP defect corresponding to a failure of the DUT to respond to an ARP, ICMP or HTTP request respectively. The combined-defect encompasses ARP/ICMP, ARP/HTTP and ICMP/HTTP defects respectively and a crash happens when the DUT fails to respond to any request sent by the CheckDevice. These defects are illustrated in Figure 9 below.

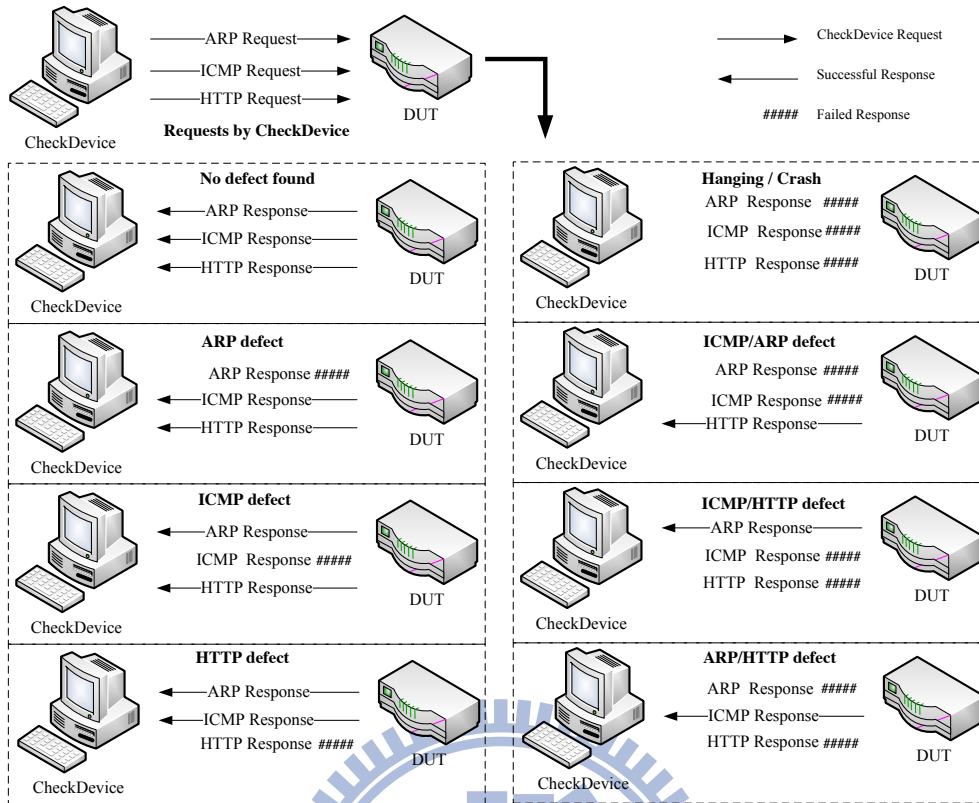


Figure 9: Overview of DUT defects

Table 3 displays a list of downsized traces that triggered defects for each DUT that is used in our experiments. For each trace that triggered a defect of a DUT, we checked with the other three DUTs to see whether the same trace are able to trigger the same defect. In this table, “✓” suggests that the same defect is triggered and “✗” means that the replayed trace did not triggered a defect on other DUTs. The observation that can be drawn indicates that some downsized traces could trigger the same defects on multiple DUTs. In fact downsized traces also trigger defects on multiple DUTs as it is the case for the original “bug traces”.

Table 3: Downsized traces

DUT / DEFECT	DUT1 (✓)			DUT2 (✓)			DUT3 (✓)			DUT4 (✓)		
	DUT2	DUT3	DUT4	DUT1	DUT3	DUT4	DUT1	DUT2	DUT4	DUT1	DUT2	DUT3
Hanging	0.96 GB			326.35 GB			27.35 GB			1200 GB		
	✗	✗	✗	✓	✓	✗	✓	✗	✗	✓	✓	✓
ICMP	0.96 GB			7.7 GB			15.32 GB			24.1 GB		
	✗	✗	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓
HTTP	0.96 GB			0.63 GB			10.38 GB			20.96 GB.		
	✓	✗	✗	✗	✗	✗	✓	✓	✗	✓	✓	✓
ICMP/HTTP	0.96 GB			8.1 GB			16.07 GB			27 GB.		
	✓	✗	✗	✗	✗	✗	✓	✓	✗	✓	✓	✓

DUT / DEFECT	DUT1 (♥)			DUT2 (♥)			DUT3 (♥)			DUT4 (♥)		
	DUT2	DUT3	DUT4	DUT1	DUT3	DUT4	DUT1	DUT2	DUT4	DUT1	DUT2	DUT3
Hanging	8.47 GB			128 GB			32 GB					
	✗	✗	✗	✓	✓	✗	✓	✗	✗	✓	✓	✓
ICMP	1.66 GB			12.7 GB			11.32 GB			64.7 GB		
	✗	✗	✗	✓	✓	✓	✓	✗	✓	✓	✓	✓
HTTP	0.96 GB			2.72 GB			6.02 GB			40.8 GB.		
	✓	✗	✗	✗	✗	✗	✓	✓	✗	✓	✓	✓
ICMP/HTTP	2.3 GB			10.4 GB			11.9 GB			120.96 GB.		
	✓	✗	✗	✗	✗	✗	✓	✓	✗	✓	✓	✓

In Figure 10 we evaluated the overall defects distribution from the experiments using 4 different DUTs.

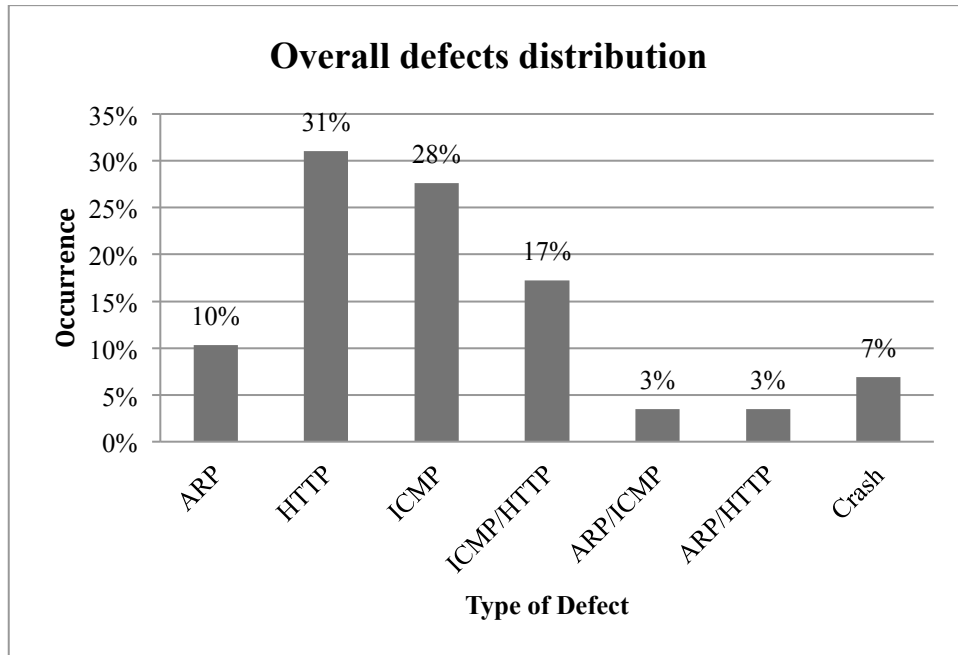


Figure 10: Defects distribution

According to the overall defect distribution, about 70% of defects are single-defect, 23% are combined-defects and 7% are hanging or crashes. In other terms, DUTs are 3 times more likely to experience a single defect than a combined-defect. In addition, a DUT crashes 10 times less frequently than it experiences single defects. The reason could be explained by the fact that the crash occurrence is tightly related to the combination of single-defects. We also observe that HTTP, ICMP defects represent more than half (60%) of the total defects experienced by DUTs. This situation occurs because ICMP and HTTP requests are more resource-intensive than ARP

requests, giving more overhead to DUT. In addition, networking devices sometimes do not reply to those requests in order to preserve resources in order to handle incoming packets.

5.3 Downsizing Ratio (DR)

We introduced a metric to evaluate the effect of the downsizing algorithms. As stated in section 3.3 the downsizing ratio is the relationship in percentage between the size of the reduced traces and that of the original one. We proposed two formulas to evaluate the DR of the two downsizing algorithms. In the first formula, DR_1 compares each single DP_j^{Li} with respect to the corresponding P_j^{Li} . In this case, each defect's DR is the size of the downsized trace over the size of the trace that triggered the defect. DR_1 is expressed as follows:

$$DR_1 = 1 - \frac{\sum_{i,j=1}^n \frac{|DP_j^{Li}|}{|P_j^{Li}|}}{n}$$

In the second formula, DR_2 compares each single DP_j^{Li} with respect to the corresponding P instead. In this case each defect's DR is the size of the downsized trace over the size of the original trace used for the testing. DR_2 is expressed as follows:

$$DR_2 = 1 - \frac{\sum_{i,j=1}^n |DP_j^{Li}|}{|P|}$$

Linear downsizing vs. binary downsizing

Figure 11 is an illustration about how LD and BD actually work. In the illustration, P has the size of 1.2 TB, the size of P_j^{Li} is 8047 MB and the illustrated defect is a hanging of a device. “✓” means that the traces triggered a defect while “✗” suggests that the traces did not. In the case of BD as shown in Figure 11(a), traces are split into halves and replayed for testing while for LD Figure 11(b), they are incremented and replayed until the defect is reproduced. The incremental size is set to 500 MB for the LD algorithm. For these illustration DP_j^{Li} is 1572 MB and 1547 MB for BD and LD respectively.

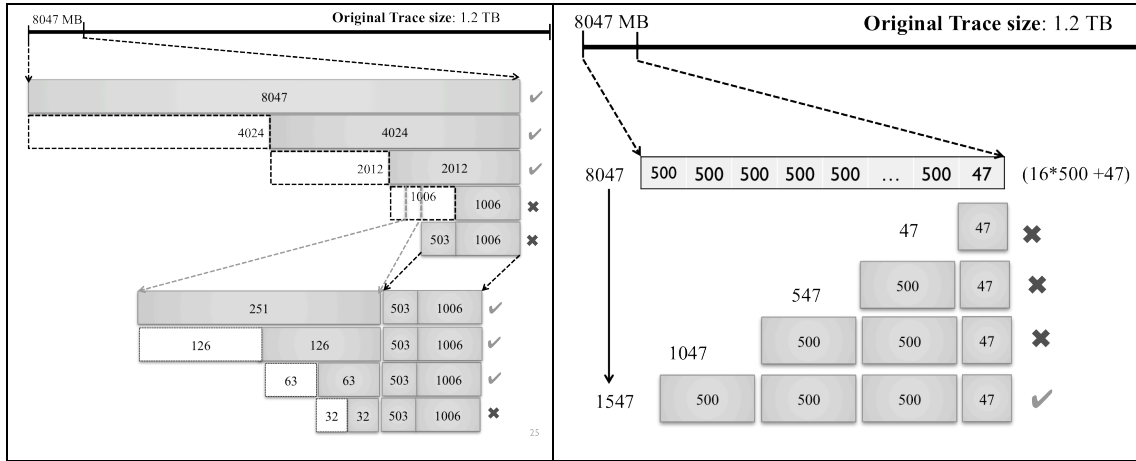


Figure 11: BD and LD Illustrations

We evaluate the overall DR using the two formulas DR_1 and DR_2 , previously introduced and show the result on Figure 12. Using DR_1 we achieve a DR of respectively 74% and 78% for BD and LD. On the other hand, using DR_2 results in 80% and 83% for BD and LD respectively. In comparing, DR_1 and DR_2 we observe that, DR_2 evaluation results in a DR that is slightly higher than that of DR_1 . This can be explained by the fact that with DR_2 the size of the downsized trace is compared against that of the original and entire trace, which is usually high in volume (1.2 TB in our case). Thus the DR_2 somehow depends on the size of the original trace, meaning that the DR might get higher with a higher volume of raw traces.

In Figure 13, we actually compare the two downsizing algorithms in terms of their DRs. We observe that BD and LD respectively achieve DRs 77% and 80%. So, LD has a DR that is slightly higher than that of BD, but in general BD would require fewer iterations than LD.

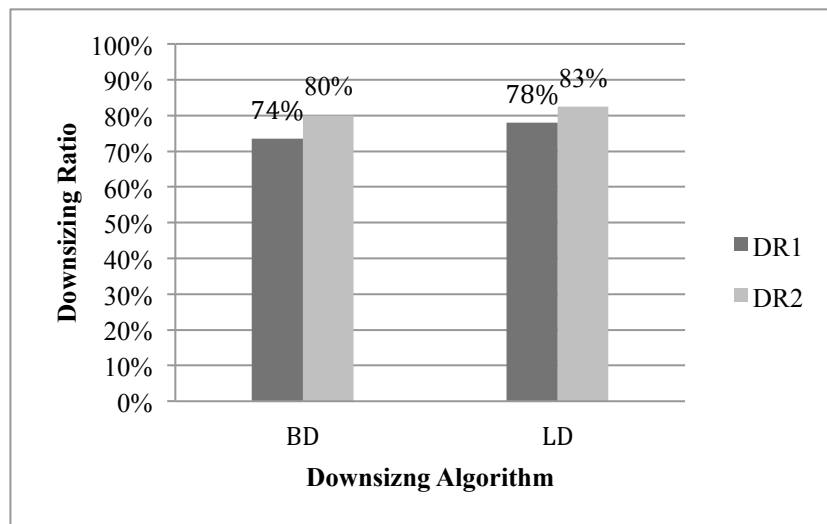


Figure 12: Two methods of DR evaluation

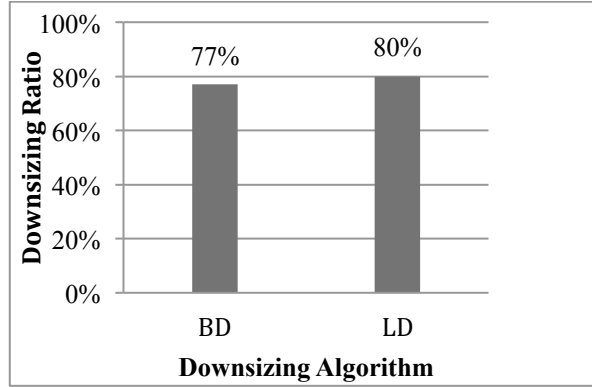


Figure 13: Comparison of two downsizing algorithms DR

Figure 14 presents a comparison of both LD and BD based on the downsizing process number of iterations and time. LD is used as a baseline and is compared to BD. The results show that BD is about 3 times faster than the BD in terms of time of downsizing process. In addition, LD requires 5 times more iterations than BD. Thus BD turns to be faster than LD because it takes advantage of the binary search algorithm.

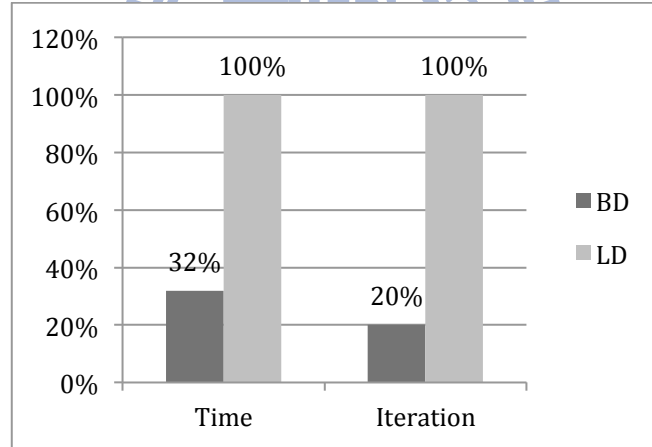


Figure 14: Downsizing algorithms: iterations and time

5.4 Replay Throughput vs. Defects Occurrence

Throughput is a key element in testing networking devices. In the real-life environment, the Internet throughput fluctuates and networking devices should be able to cope with these variations. Therefore we conducted an experiment where DUTs are tested with a throughput that is two times higher than that used in ILRT testing environment. The reason is to find out how replay throughput affects the stability of the DUTs. In Table 4 “✓” means a crash of the DUT while “✗” represents a DUT that is properly functioning. We can observe that there are 3 out of 4 DUTs that

crashed under a replay throughput of 100 mbps as opposed to 2 in the case of 50 mbps. It is also observed through the replay testing that DUTs do not product defects under a replay throughput that is below 50 mbps.

Table 4: Replay throughput and DUT failure

DUT/Replay Throughput	50 Mbps	100 Mbps
DUT1	✓	✓
DUT2	✓	✓
DUT3	✗	✓
DUT4	✗	✗

In more details in Figure 15 we note that there is a higher number of defects when devices are tested with a higher replay throughput. Our results show that networking devices produce 1.5 – 2.8 times more defects in higher throughput (100 mbps) than in lower throughput environment (50mbps). In addition, tested with a replay throughput above 100 mbps, most of the DUTs experience a hanging. In some particular cases, the DUT reduces its throughput to less than 100 mbps to avoid hanging. This could be explained based on the fact that with a higher replay throughput, the number of number of requests per time unit to the DUT is much more likely to increase as well. A high number of requests eventually overload the DUT and results in more defects if the DUT cannot handle all the requests.

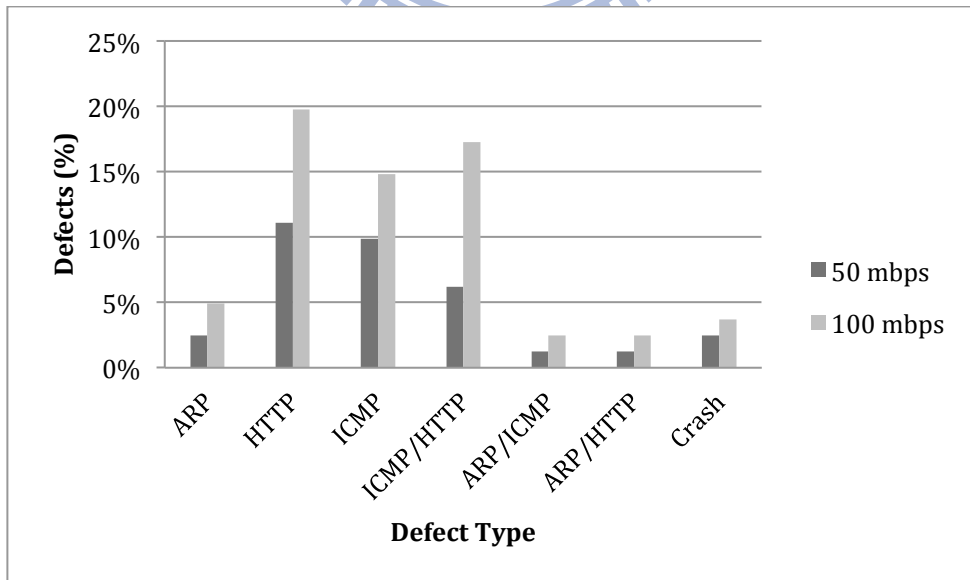


Figure 15: Replay throughput and Defects occurrence

5.5 Investigating the Determining Causes of Defects: Case Study

The DUT that are used in our experiments are SOHO routers and behave like black boxes because they do not display significant information about their failure. Therefore it makes it challenging to discover the causes of their defects. As said in the previous section, replay throughput could certainly explain some defects as their number increases with higher throughput. But in order to find more insight about the causes of the defects, we need to obtain some information about the logging system within the router. So, we conducted a new experiment where we installed an open-source third-party firmware [19] on DUT₄, which is the only DUT that supports and is compatible with the open-source firmware. We labeled it DUT_o. For the replay testing of DUT_o we used the same packet traces, the same throughput with the same environment as the one that was used to test other DUTs for consistency. We are able to monitor more closely DUT_o through its terminal. Figure 16 displays the log messages produced as it crashes. We note that the message displayed is “*nf_conntrack: table full, dropping packet*”. One of the important features built on top of the Netfilter framework is connection tracking. Connection tracking allows the kernel to keep track of all logical network connections or sessions, and thereby relate all of the packets, which may make up that connection. NAT relies on this information to translate all related packets in the same way [20]. In our case, the message means that the NAT table is full as the device tries to build more connections than it can handle. In Figure 17, we can observe from the web interface messages that the DUT_o is actually overwhelmed by the high number of active connections it needs to handle. It has to handle 99% of the maximum number of active connections. This overloads the device resulting in a high CPU usage (94%). All these reasons more likely are the causes of the device crash and could also explain the failure with the other DUT used in previous experiments.

```

DD-WRT kern.warn kernel: nf_conntrack: table full, dropping packet.
DD-WRT kern.warn kernel: printk: 26207 messages suppressed.
DD-WRT kern.warn kernel: nf_conntrack: table full, dropping packet.
DD-WRT kern.warn kernel: printk: 29102 messages suppressed.
DD-WRT kern.warn kernel: nf_conntrack: table full, dropping packet.
DD-WRT kern.warn kernel: printk: 24650 messages suppressed.
DD-WRT kern.warn kernel: nf_conntrack: table full, dropping packet.
DD-WRT kern.warn kernel: printk: 25835 messages suppressed.
DD-WRT kern.warn kernel: nf_conntrack: table full, dropping packet.
DD-WRT kern.warn kernel: printk: 23826 messages suppressed.

```

Figure 16: Log messages from a DUT running an open-source firmware

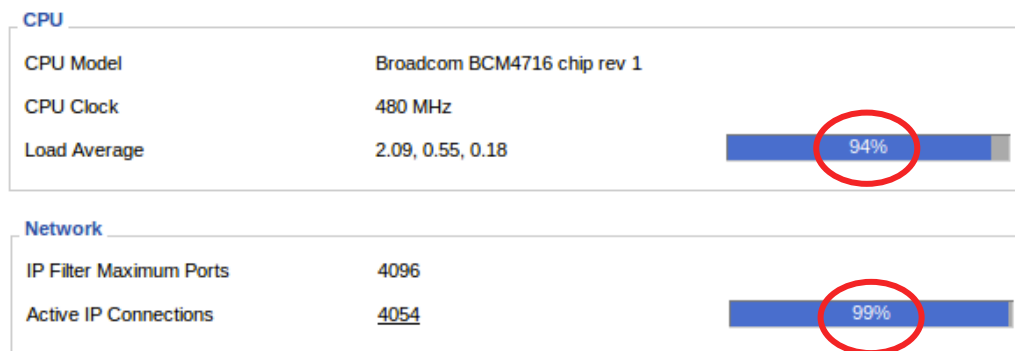
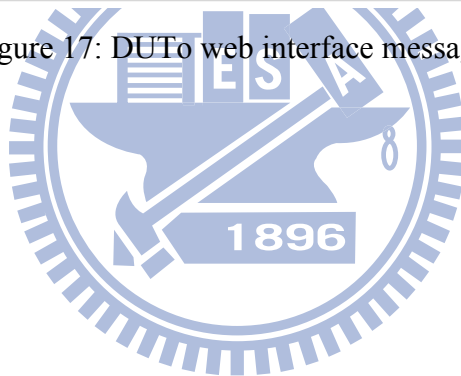


Figure 17: DUT web interface messages



Chapter 6 Conclusions and Future Works

Testing networking devices with real-world traffic is an alternative to testing with artificial traffic given the realistic properties of real-world traffic. However, due to the high volume of captured traces, it is necessary to downsize the packet traces for more efficiency in the devices testing and their defect reproduction.

This work proposed a linear and a binary downsizing algorithm to downsize packet traces triggering the defects of networking devices. From the downsizing experiments, we distinguished 7 different types of devices defects, which we classified into single-defects, combined-defects and hanging. The observed defects result from a failure of the DUTs to respond to the ARP, ICMP or HTTP queries sent to probe their status. According to the results of our experiments, ICMP and HTTP defects represent more than half of the total number of defects produced by the DUTs. This might be explained by the overhead in terms of resources that these requests have on the DUT. We also proposed two downsizing algorithms for packet traces downsizing. The proposed downsizing algorithms achieved a downsizing ratio of 77% and 80% using respectively BD and LD algorithms. The experiments results also showed that BD is about 3 times faster than the BD in terms of time of downsizing process, and LD requires 5 times more iterations than BD.

In our study it is shown that the throughput at which the DUTs are tested could affect the occurrence of defects. In fact the DUTs produced 1.5 – 2.8 times more defects with higher replay throughput (100 mbps) than in lower replay throughput (50mbps). Below a replay throughput of 50 mbps, we do not observe devices defect. This could be explained by the fact that the number of requests per time unit is more likely to increase with a higher replay throughput.

In investigating the cause of DUT's defects, we observed that the connection-tracking table of the DUT gets full during the device hanging. This suggests that the size of the table could be associated with the device eventual failure because it is shrunk as the device tries to build more connections or handle NAT translations. The DUT, which can no longer process incoming requests, ends up overloaded and eventually crashes.

In our work, we define the defects as the DUT's incapacity to respond to probing queries and handle them consequently.

For future works, devices defect could be defined in a more comprehensive way that will consist of monitoring the internal behavior of DUTs. Therefore it will be necessary to introduce probing methods that will be able to disclose more detailed information about the performance of the DUTs. Since the CheckDevice probing technique consists of sending queries to the DUT, it could be limited in detecting the status of the DUT daemons status. Therefore in order to detect more defects from the DUT, the Simple Network Management Protocol (SNMP) could be used in addition the CheckDevice. SNMP is a monitoring tool that monitors the conditions of network-attached devices. However for SNMP to be used, the devices must support it. This could eventually help discover even more defects that would not be revealed through probing queries, and improve the efficiency and accuracy networking devices testing.



References

- [1] Y. D. Lin, I. W. Chen, P. C. Lin, C. S. Chen, C.H. Hsu, "On Campus Beta Site: Architecture Designs, Operational Experience, and Top Product Defects," *IEEE Communication Magazine*, 2010
- [2] M. K. Daskalantonakis, "A Practical View of Software Management and Implementation Experiences within Motorola," *IEEE Trans. Software Eng.*, vol. 18, no. 11, 1992, pp. 998–1009.
- [3] Alessio Botta, Alberto Dainotti, Antonio Pescape, and Giorgio Ventre "Reducing Network Traffic Data Sets" *ICC 2007 proceeding*.
- [4] Antonio Pescape "Entropy-Based Reduction of Traffic Data", *IEEE Communications Letters*, Vol.11, No.2, February 2007.
- [5] Y. Liu, D. Towsley, J. Weng, D. Goeckel, "An Information Theoretic Approach to Network Trace Compression", *UM-CS-2005-003 TR*, Jan 2005
- [6] Y. Liu, D. Towsley, T. Ye, and J. Bolot, "An Information-theoretic Approach to Network Monitoring and Measurement" *IMC 2005*
- [7] G. Iannaccone, C. Diot, I. Graham, N. McKeown. "Monitoring very high speed links," *Proceedings of ACM Internet Measurement Workshop*, November 2001.
- [8] M. Peuhkuri. "A method to compress and anonymize packet traces," *Proceedings of ACM Internet Measurement Workshop*, November 2001.
- [9] Y. Liu, D. Towsley, T. Ye, and J. Bolot, "An Information-theoretic Approach to Network Monitoring and Measurement," *IMC 2005*
- [10] K. C. Claffy, G. C. Polyzos, and H. W. Braun, "Applications of Sampling Methodologies to Network Traffic Characterization," *ACM SIGCOMM 1993*
- [11] Szabó, G., Szabó, I., Orincsay, D "Accurate Traffic Classification," *IEEE WOWMOM, Finland, June 2007*
- [12] Karagiannis, T., Papagiannaki, K., Faloutsos, M "BLINC: Multilevel Traffic Classification in the Dark", *ACM SIGCOMM, Philadelphia, USA, August 2005*
- [13] NBL PCAPLib. [Online]. Available: <http://security.nbl.org.tw/>
- [14] Y-D. Lin, T.-H. Cheng, P.-C. Lin, I.-W. Chen, and Y.-C. Lai, "Low-storage capture and loss-recovery stateful replay of real flows," *IEEE Communications Magazine*, to appear, available upon request.
- [15] Ying-Dar Lin, Jui-Tsun Hung, Ren-Hung Hwang, Chun-Nan Lu "In-Lab Replay Testing with A Case Study on SOHO Routers"

- [16] TCPReplay.[Online].Available: <http://tcpreplay.synfin.net/>
- [17] Network Benchmarking Lab.[Online].Available: <http://nbl.org.tw>
- [18] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms (2nd ed.)
MIT Press 2003
- [19] DD-WRT.[Online].Available: <http://www.dd-wrt.com/>
- [20] Netfilter.[Online].Available: <http://www.netfilter.org>

