Master Thesis
Electrical Engineering
June 2011

Causes of TCP Resets in Mobile Web Browsing

Prasanna Kumar Amburu
Anil Varma Bethalam

School of Computing
Blekinge Institute of Technology
37179 Karlskrona
Sweden

This thesis is submitted to the School of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information

Author(s):
Prasana Kumar Amburu
Anil Varma Bethalam
E-mail: prasanna.amburu@gmail.com,anilvarmab@gmail.com

University advisor(s):

Mr. Junaid Shaikh, PhD Student
COM/BTH

School of Computing
Blekinge Institute of Technology
371 79 KARLSKRONA SWEDEN

Internet: www.bth.se/com
Phone: +46 455 385000
SWEDEN

Acknowledgement

The authors would like to thank our supervisor Mr. Junaid shaikh for his kind guidance and input on various aspects of this work. The authors would like to thank Dr. Patrik Arlos and Mr. Tahir Nawaz Minhas for their assistance with experimental systems used in this work. Many thanks to Ravi Chandra Kommalapti and our friends for their kind support during the thesis work. Finally we would like to thank our parents for their constant love and support .

Abstract

Web browsing has been one of the most popular activities on the internet. The increasing importance of the Web in everyday life calls for device-independent access to existing web sites. Although, there is full Web access available on mobile phones the user experience is often poor when compared to the Web user experience on Personal Computer (PC). Therfore, it is important for the Internet service provider to find the indications of user dissatisfaction from the network. While using the mobile Web, user can normally abort the transfer by pressing the stop or reset buttons in the browsers, leaving the page being downloaded by following a bookmark or can close the connection. Such events can be observed through the TCP reset (RST) flag from traffic on network level.

In this thesis we have investigated the possible causes of TCP RST flags mobile Web session is interrupted. We further analyze up to what extent we can rely on TCP RST flags for being an indication of user dissatisfaction in mobile web browsing. Therefore, an experiment testbed is developed to capture the TCP packet traces during controlled active tests. Results are gathered using mobile devices with four popular mobile operating systems (OS). The trace files captured are analyzed using perl script to dicern the flow and focusing on the TCP RST flag during the flow. Further, TSTAT tool is used to validate our trace files.

Keywords: Transmission Control Protocol (TCP), Reset flag, Hyper Text Transfer Protocol (HTTP), Mobile browsing.

Contents

List of Figures

List of Tables

Introduction

Chapter 1

Introduction

Nowadays, mobile devices are used to access any sort of information and have become a major means of accessing the web. The mobile devices and wireless technology are being upgraded and thus providing variety of functionalities with better service. It is estimated that mobile web usage is set to represent 50% of the total internet usage by 2014 [1]. Even though the number of Internet enabled mobile devices are growing rapidly, not all web sites are available for mobile Internet users. As [2] highlights many users expect the Internet experience on mobiles to be similar to the Internet experience on a PC. However, many popular websites do not provide a mobile version of their sites. This adaption of information to mobile web can be inconsistent [3]. Although there are usability constraints, cheap and durable mobile devices have added flexibility in web browsing by providing users with the possibility of accessing readily available information from anywhere. Quality of experience is a major factor in successful mobile web browsing.

There are many challenges for higher layer networking protocols in the wireless environment for simple applications as web browsing. The two most applicable protocols are the transmission control protocol (TCP) and hypertext transfer protocol (HTTP). The primary transport layer protocol in the Internet protocol (IP) stack is the TCP which provides reliable data delivery. TCP performance for web browsing particularly in a wireless environment can be unpredictable [4]. And also the HTTP protocol used for web downloads can often lead to inefficient TCP performance [5]. All TCP implementations pass through several states from connection establishment to connection termination to enable data transfer on the network.

To ensure reliable data transfer, TCP controls transmission of packets with the help of control flags which occupy a 6-bit field in the TCP header. SYN flag is used to synchronize the sequence number. Both client and server exchange a pair of packets with SYN flags to establish a new connection. ACK represents acknowledgement, this flag is present in all packets except for the first packet. The PSH flag is sent by the sender when it requests

for immediate transfer of data. URG flag is used when the packet contains urgent data and is rarely used. When there is no more data to transmit the sender transmits a FIN flag. Both client and server exchange a pair of packets with FIN flags at the end of the connection. A RST flag is sent when either end wishes to abort the connection. So we consider TCP RST flag as a possible indication of interruption.

1.1 Motivation

However, there are various reasons for the occurrence of RST flags during TCP connection. Therfore it is not completely reliable to consider TCP RST as an indication of user dissastisfaction. Research has been done mostly in the PC environment to find out the different causes of TCP RST flag. A recent study [6] shows that 15-20% of TCP connections observed from a large campus network over the period of one year are abnormal, in that they experience at least one TCP reset flag. Their study concluded to show that browser behavior is the reason behind most of the client side TCP RST flags. In another experiment conducted [7] authors resolve the cause of 10.6% TCP RST flags are from user interruptions due to bad connection speed. Another study [8] defines a methodology to understand TCP flow interruption. A recent study [9] conducted experiments with different webpages containing text, picture and video disapproves this criteria in case of Internet explorer. There is a lot of subjective data available on mobile user behavior and usage as the most widely used data collection method is surveys [10]. Another straightforward method in research studies is TCP/IP traffic measurements. We use this method to obtain objective data from measurements.

Our Current knowledge is missing in the area of detecting what are the main causes which falsely terminate an established TCP connection. The main purpose of the project is to investigate and analyze up to what extent we can rely on TCP resets for being an indication of user dissatisfaction in mobile web browsing. In this process we figure out the causes of TCP reset flags in mobile scenario and the conditions in which the reset flags are generated.

1.2 Research Questions

- What are the various causes of client generated TCP reset in a mobile web browsing session?
- How to differentiate between client generated resets in different scenarios?
- What is the extent to which TCP reset forms an indication of user dissatisfaction?

1.3 Research Methodology

The research methodology implemented in this thesis is empirical (Derived from experiment and observation rather than theory). The primary task involves thorough documentation on collecting the different causes of TCP reset in mobile web browsing. The experiment and observation that is done contains a setup in which the network is monitored for TCP traces. Experiments are conducted in different conditions in which TCP resets occur. Traces from all the results are observed. The secondary task involves the quantitative part of collecting the results from these causes by using the experimental setup. Observations are then narrowed down to a scenario in which we can clearly specify that a TCP reset occurs due to user action or any other condition. Then Results are analysed using Perl tool from which we deduce the user dissatisfaction relating to TCP resets in the mobile web browser.

1.4 Contribution

Our work differs in explaining the behavior of TCP connection and classifying the causes of RST flags in case of interruptions by a mobile user. This thesis investigates up to what extent we can rely on TCP reset flags for being an indication of user dissatisfaction in mobile web browsing. This classification also specifies the difference between a user generated and browser generated RST flag. This assortment of data obtained from performing active tests are expected to help ISPs with a understanding of user behavior based on the RST flag during a session of mobile web browsing.

1.5 Thesis Outline

The rest of the document is as follows: The chapter 2 provides a detailed literature review and related research of web browsing on mobile phones and the impact of TCP RST flag in this process. Chapter 3 consists of the experiment design which explains the equipment used and how the experiment was designed, and in the procedure used to conduct the experiment. Chapter 4 provide the results and detailed analysis of the collected data. Chapter 5 concludes the thesis and gives future directions .

Literature Review

Chapter 2

Literature Review

2.1 Background

This section provides background information of TCP/IP and HTTP connections. The basics of connection management in TCP and HTTP protocols, TCP connection states and TCP RST are explained. Section 2.2 gives related research work done in the past relating to traffic analysis and user behavior in mobile phones.

The two most relevant protocols used for simple applications as web browsing areTCP and HTTP. TCP is a primary transport layer protocol in the IP stack providing reliable transfer for file transfer. TCP is an end to end reliable protocol where the sender retransmits a packet for which no ACK flag is received in a time period called Retransmit Time Out (RTO).

2.1.1 TCP Connection Management

TCP is a connection oriented, end to end protocol designed to support multi network applications and reliable data transfer between computer communication networks. When two processes want to communicate, their TCPs must establish a connection by initializing the status information on each side. The procedure to establish a connection uses the synchronize (SYN) control flag and an exchange of three messages which is called a three way handshake. The connection is initiated by the arriving segment containing a SYN flag. The next packet has two flags set, ACK flag to acknowledge the clients SYN packet and SYN flag to indicate the server also wishes to establish a TCP connection. The third packet from the client has the ACK flag set which sends the connection into synchronized state. After entering into the synchronized state the data transfer between client and server takes place. When the data is transferred, the connection is closed or terminated. The connection tear down is initiated by the server which follows a pair of two way handshake involving FIN bits set in two TCP packets followed by a final acknowledgement packet. The client sends the packet with FIN bit

set after the server acknowledges all the data packets being sent after the connection setup. The server then acknowledges the client to terminate the connection by sending an ACK flag and also a FIN flag to end connection from server end. The connection is completely closed at both ends after the client acknowledges the servers FIN flag by sending an ACK flag.

2.1.2 TCP Connection States

The default state in which each process begins is the CLOSED state where no connection is established. A device is in LISTEN state when it is waiting to receive a SYN flag from client without sending a SYN flag of its own. After the connection is established it enters into synchronized state. The synchronized state consists of FIN–WAIT–1, FINWAIT–2, CLOSE–WAIT, LAST–ACK, and TIME–WAIT [11]. After the connection is established, data can be exchanged freely between both devices until it is closed. The device in FIN–WAIT–1 state sends FIN flag for termination and waits for the ACK for its FIN or termination request from other device. After receiving a FIN the device waits for the application on the local device to acknowledge and generate a request CLOSE–WAIT. A device which has already received a termination request and acknowledged it and sent its own termination request waits in the LAST–ACK state. In the FIN–WAIT–2 state, the device receives ACK flag for its FIN and moves to TIME–WAIT state. The TIME–WAIT state is to confirm the receiving of the last ACK flag to prevent overlap with new connections.

2.1.3 TCP Reset

The RST bit located in the TCP header is used by the TCP to reset a connection. TCP controls the transmission of packets to assure reliability. TCP has control flags which occupy a 6–bit field in the TCP header. Resets are observed to occur in three different cases. A reset is generated when a connection request gets in and there is no process hearing on the destination port, i.e. nonexistent connection. It is possible to abort a connection by sending a RST flag instead of a FIN [12]. A RST is sent in reply to any incoming data packets except another reset after the connection is closed. When a connection is aborted the queued data is rejected and RST flag is immediately sent. The receiver of an RST flag can distinguish if the other user aborted the connection instead of a normal close. RFC793 contains the original specification of TCP in which RST bit is defined. This document explains the function of reset is to prevent duplicate connection initiations from confusing TCPs three way handshake. According to the guidelines in RFC793 a reset (RST) must be sent whenever a segment arrives which is not intended for the current connection. It is stated in RFC1122 that it corrects RC793 and supplements it. Both RFC793 and RFC1122 prohibit

sending a reset for the reason that SYN packet uses reserved flags in TCP header. A reset is validated after its SEQ field is checked i.e. the sequence number is present in the window. Any receiver of the reset first validates it and then changes state, but if the receiver is in LISTEN state it is ignored. If the receiver was SYN–RECEIVED state, upon the reception of a RST it goes back into LISTEN state. For any other state the connection is aborted and goes to CLOSED state.

2.1.4 HTTP 1.0/1.1 Protocol

HTTP is a request/response protocol which uses TCP as its transport protocol. TCP performance is better for long connections but as many web interactions are limited, the bandwidth usage is not utilized to its fullest. HTTP 1.0 used a separate TCP connection for each request. This caused several overhead packets and became a limitation for heavy internet traffic. HTTP communication is usually started at the client end by sending a request to a server achieved by a connection between both. The use of a new HTTP connection sometimes for each image or video embedded in the web page puts response time for the requests. The introduction of parallel TCP connections could not retort as this brought congestion issues. This was solved by introducing persistent connections and pipelining. HTTP 1.1 brings in the concept of hop-by-hop header [13]. These message headers are only applied for each connection unlike HTTP 1.0 which uses it for the entire path. HTTP 1.1 uses a connection header which lists all hop–by–hop headers in a message. This allows the proxy to remove the headers before the message is sent further. Connection header is not recognized in HTTP1.0 and is ignored. HTTP1.1 keeps the keep-alive header in the connection header to signal the maintenance of the persistent connection. By the default properties of persistent connection, the connection is assumed to be open even after the request and response messages. Connection close header is used to inform the end of the connection to the receiver. This helps in reducing network congestion and processing time in routers and clients and servers. Even as multiple requests are allowed on a single connection, the server has to respond to the client in the order of the receiving request. However, the client is allowed to send a large number of requests before receiving any response which helps in maximizing the efficiency of TCP connection. This is called pipelining. A much detailed description of HTTP 1.1 is given in RFC2616.

2.2 Related Work

In this section we explain the relevant research done in the area of user experience in mobile web browsing. The related research in the specific area of user experience in mobile browsing has not been investigated thoroughly.

A lot of subjective research [14], [15] has been done to analyze mobile web user behavior by studying log files generated by users. The publications combining both user experience and mobile web browsing is scattered which motivated us to dive into this specific subject.

In [6] the authors present a one year study of internet packet traffic captured from a large campus network with 15-20% of the connections showing at least one TCP RST. They identify various causes of abnormal TCP connections i.e. the amount of RSTs from these passive measurements. The traffic is captured by *tcpdump* and processed by *Bro*. They further perform active tests on different browsers, operating systems and web servers to analyze TCP connections. They conclude that the browser behavior (irregular management of HTTP persistent connection) is the main contributor for the RSTs.

In [8] authors present a web user behavior study analyzing 2 months of real traffic to know the impact of user behavior during poor web transfer. Several parameters are considered like time of day, file size, throughput. Authors define a methodology to deduce the difference from TCP flow interruption caused by user and other causes. The authors consider round trip time (RTT) between flags and define a criterion to differentiate between different interruptions. They resolve that user throughput is the main factor affecting interruption probability.

Another study [16] defines several aspects of flow structure and its applicability to measurements of TCP traffic. They also define a methodology for profiling traffic flow based on packet traces from multiple networking environments. In [17] authors compared timeouts of 5 and 15 min and found little difference in data exchange duration. Determining when a flow has ended and its associated state information is a significant task. We use this heuristic and allow a timeout period of 5 minutes after each of our individual flows in our experiment.

Sujan shrestha [18] provides an interesting study on user perception of mobile web browsing. Author performs experiments in a laboratory environment to understand user satisfaction and subjective feelings on the usage of mobile phone for web browsing. From the experiments he concludes that the user experience on mobile web usage is not consistent.

In study [7] authors discuss major bandwidth utilization problems due to RSTs generated by impatient users and retransmissions. The authors provide an algorithm to distinguish between a RST of a user and network RST in a HTTP connection. This algorithm is based on an assumption of number of bytes sent before a RST. They provide a detailed analysis on TCP packets discussing various statistics but no reasons on the causes of RST are provided.

In another study [19] authors identify end user behavior in interruption of TCP connections by passive analysis from traffic characteristics. Authors reason that observing RST packet from user is not sufficient in distinguishing

interrupted and complete connections. They define that inter arrival time of connections from the user is a suitable parameter to identify user satisfaction or dissatisfaction in the network.

Experimental Setup

Chapter 3

Experimental Setup

This chapter presents a brief description of the experimental setup and a short description of the components used in the test bed. Section 3.2 explains the experimental procedure and conditions under which the experiments have been performed.

3.1 Experiment Test Bed

The experiment test-bed is based on Distributed Passive Measurement Infrastructure (DPMI) [20]. Figure 3.1 shows the detailed structure of the experiment setup. It allows collection of TCP/IP packet headers, and packet payloads as well. The test-bed contains a web server, a client, a measurement point, a Measurement Area Controller (MArC), and a consumer for storing the capture files. Below is a short description of each component used in the experiment test bed.

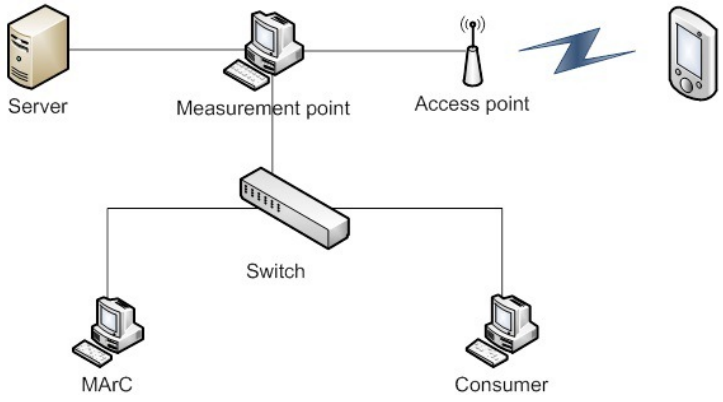


Figure 3.1: Experiment Test-Bed

3.1.1 Measurement Point

The Measurement Point (MP) is a physical device dedicated for packet capture in the network. It is a computer with Linux operating system installed on it. The MP is equipped with two Data Acquisition and Generation (DAG) cards which help in packet capture. DAG cards are relied upon to guarantee 100% traffic capture on any network irrespective of packet size and network load [21]. DAG cards also provide accurate time stamping of each packet. The DAG interfaces are synchronized for accurate network measurements. Network Time Protocol (NTP) was used for this synchronization process.

3.1.2 Consumer

The data collected by the MP is sent to the consumer. Consumer is a physical device with Linux operating system which can store the collected traces. We used the convention shown below for naming the various capture files being stored on the consumer.

T_O_A_W_N_1: Web page type (Text)_ Mobile browser (Opera)_ Web server (Apache) _ mobile operating system (Windows) _ user action (Type of action)_ number of repetition(1)

3.1.3 Access Point

An access point is a Wi-Fi router used for connecting the mobile phones to the web server in the existing LAN network. The clients have been connected with the server using ORINOCO AP-500 access point.

3.1.4 Clients

We have used different clients to consider the current popular mobile operating systems and browsers. As per the recent statistics the Android platform on smartphones is the most popular currently overtaking Symbian and Windows platforms [22]. iphone Operating System (iOS) is also a popular platform with its usage most popular for apple devices. We have used these four most applied platforms.

Table 3.1 shows the different mobile devices and the respective platform and browsers used in our manual testing of the mobile clients. These mobile devices are selected to work on latest updated versions of mobile operating systems. Apart from using the built-in browsers provided in each smartphone, we also use an additional common browser for these operating systems. Opera being one of the most popular browsers was available for Android, Symbian and Windows platform but unavailable for iOS platform on mobiles.

Table 3.1: User End Mobiles and Specifications

Mobile Device	Operating System	Browser Versions
HTC TyTn-II	Windows Mobile 6.0	Internet Explorer 6.0 (Built in browser)
	Windows Mobile 6.0	Opera mobile10.0
Apple IPOD touch	iOS 4.0.2	Safari (Built in browser)
HTC Desire HD	Android 2.2	Opera mobile 10.0
	Android 2.2	Built in Browser
Nokia N8	Symbian 3.0	Built in Browser
	Symbian 3.0	Opera Mobile 10.0

3.2 Experiment Procedure

One of the main parts of this thesis work is to differentiate between user generated TCP RST flag from a bowser generated RST flag. User generated resets are largely based on user impatience and interruptions [8].To discern the differences, we consider two scenarios Uninterrupted and Interrupted. In the Uninterrupted scenarios, the user accesses the web page without any break or discontinuity during data transfer. In the interrupted scenarios, after the user accesses the web a break or discontinuance is introduced while the data transfer is still going on. We consider four possible cases of interrupting the web page transfer at the client end. This could be done by pressing the reset, stop buttons in the browser, by exiting the browser application, and by clicking a URL in the web page. We manually interrupt the transfer while the transfer during the experiment to record the traces. Hence, we conduct experiments in a controlled laboratory environment by providing interruptions at the user end (mobile device). The user actions during the Interruptions are explained below

"Reload"- Interruption is done by pressing the reload button in the browser during the data transfer.

"Stop" - Interruption is done by pressing the stop button in the browser while the data transfer is still going on.

"Close" -The browser application is closed or exited to interrupt the TCP flow during the data transfer is still going on.

"Redirect to another URL"- The URL link in the webpage is clicked before the transfer is completed.

We initiate the TCP connections at the client side from the mobile web browser by requesting the IP address of the server containing the web page. The MP is initiated and the capture is started on the consumer. The server is then accessed through the mobile web browser by entering the URL of the server. We have used Apache server 2.2.11 to host the web pages. The TCP version used at the server is TCP New Reno [23]. This web page is a

HTML page of size 1.3MB which contains text and a web link. We conduct a total of 40 iterations in the uninterrupted scenario and each of the four cases of interrupted scenario. Each iteration is given a 5 minute timeout after the data is transferred in the uninterrupted scenario and after the user interruption action in interrupted scenario. Care is taken to remove the cache before conducting succeeding experiments.

Results

Chapter 4

Results

In this chapter we focus on the results obtained from the DPMI packet traces to present the study of TCP reset behavior, the effects of user experiences on the TCP reset flags .And then we discriminate user generated resets and other causes of resets. To Study the TCP reset behavior, we conducted active tests with different mobile browsers and with local webserver (Apache 2.2.11). Using DPMI packet traces, we then captured and analysed the TCP reset flags that are obtained from the experiments. These experiments contain two set of scenarios interrupted and uninterrupted.

4.1 Uninterrupted Scenario

The uninterrupted scenarios involved an HTTP request from the mobile phone to the local Apache server and leave the connection open for 5 minutes time to study the uninterrupted behaviour. Table 4.1 provides a description of observed TCP flags.

Table 4.1: Summary Of Observed TCP Flags

State	Description
FA_c	FIN from Client
FA_s	FIN from Server
R_c	RST from Client
R_s	RST from Server
R_c+,R_s+	Multiple RST

Table 4.2: Uninterrupted TCP Flow

Client Operating System	Client browser	Termination flow	Behavior	idle time $T_d$seconds
Windows 6.0	I.E 6.0	$(FA_s)(FA_c)$ (R_c)	Server Request for connection termination with FIN handshake after $T_d$seconds of last data packet from the server Followed by RST form client	5.58
Windows 6.0	Opera Mobile 10.0	$(FA_s)(FA_c)$ (R_c)	Server Request for connection termination with FIN handshake after T_d seconds of last data packet from the server Followed by RST form client	5.55
iOS 4.0.2	Safari	$(FA_c)(FA_s)$	Client initiates FIN handshake for connection termination Immediately afterT_d seconds of the last data packet from the server	0.01
Android 2.2	Opera Mobile 10.0	$(FA_s)(FA_c)$	Server initiates FIN handshake for connection termination T_d seconds after the last data packet from the server	5.52
Android 2.2	Default Browser	$(FA_s)(FA_c)$	Server initiates FIN handshake for connection termination T_d seconds after the last data packet from the server	5.54
Symbian 3.0	Default Browser	$(FA_s)(FA_c)$	Server initiates FIN handshake for connection termination T_d seconds after the last data packet from the server	5.65
Symbian 3.0	Opera Mobile	$(FA_s)(FA_c)$	Server initiates FIN handshake for connection termination T_d seconds after the last data packet from the server	5.51

In table 4.2 summarizes the results obtained from uninterrupted experiments. From the table 4.2 we state the following observations:

1. We have observed that Apache server initiates FIN handshake after 5 seconds of receiving the last data packet from the server.

2. The process of termination is different when the client uses iOS with a Safari browser. In this case, the termination action is initiated by the client with a FIN handshake immediately after it receives the last data packet from the server.

3. When the client uses Windows OS, the server requests for termination of connection with FIN handshake five seconds after the last data packet from the sever is sent to the client and it is then followed by a TCP RST flag from the client. The process is similar for both Internet Explorer and Opera Mobile 10.0 browsers as long as the client uses Windows operating system.

4. For the cases, where the operating systems being used by the client are Android and Symbian, the process of termination is similar irrespective of the browsers being used. For both Opera Mobile 10.0 and built-in browser, the server initiates the connection termination with FIN handshake, 5 seconds after the last data packet is sent from the server to the client and the connection is then followed by TCP FIN flag from the client.

From the observations made from table 4.2, we conclude three points:

First, the majority of uninterrupted cases use proper FIN handshake to terminate the idle connection. However, the web browsers used on the Windows OS i.e. Internet Explorer and Opera generate a TCP reset flag (From client) after the FIN handshake used for connection termination.

Second, we observe the following behaviour for the same opera mobile browser 10.0

1. In case of Android and Symbian a proper SYN/FIN is used to close the connection,

2. But in case of Windows client operating system it used a RST flag to close the ideal TCP connection.

As there is no action from the user to terminate an on-going web transfer, we resolve that the RSTs occurring are due to client side implementation. We could have assumed that the TCP RSTs flags are due to the browser, but due to the observations made on Opera browser and Internet explorer we conclude it to be the behaviour of the mobile operating system.

Finally, Safari on iOS showed a different behaviour compared to the other three client operating system. It used a TCP FIN flag from client side

for the connection termination while the others used TCP FIN flag from the server side.

4.2 Interrupted Scenario

In the Interrupted Scenarios the TCP connection can interrupted by the user using following actions: “Stop”, “Close”, “Reload” and “Redirect to URL” actions while the current transfers are going on. A summary of these actions is provided in section . We further discuss the results obtained after performing these actions.

4.2.1 Interruption by Stop Action

In Table 4.3, we summarize the results obtained from stop action in the interrupted experiments. From the Table 4.3, we state the following observations.

In cases when the client is using Windows operating system with Internet Explorer and Opera Mobile 10.0 browsers, the termination process of TCP connection is initiated by the client with a FIN handshake followed by TCP RST flag from the client to the server. In cases where client used iOS operating system with a Safari browser, when the transmission of data is interrupted by stop, the client initiates the termination of TCP connection with a FIN handshake followed by TCP RST flag from the client to the server, just as in the case of a Windows operating system. The termination behaviour for Android OS with built in browser and Opera Mobile 10.0 is similar to that of Windows and iOS. The interruption of a process by stop results in initiation of the termination by the client with a FIN handshake followed by TCP RST flag from client to server. A client using Symbian Operating System with built in browser and Opera Mobile 10.0 shows a termination behaviour in which the client initiates the termination with TCP RST flag followed by a FIN flag and the server responds by sending a TCP RST flag back to the client.

4.2.2 Interruption by Close Action

In Table 4.4, we summarize the results obtained from close action in the interrupted experiments. From the Table 4.4, we state the following observations.

On a Windows operating system with Internet Explorer and Opera Mobile 10.0, when the transmission process is interrupted by close, the termination process is initiated by the client with a FIN handshake followed by TCP RST flag to the server. The process of termination when the data transmission is interrupted by close on iOS and Android Operating Systems is similar to that on Windows. A client using iOS with Safari browser or

Table 4.3: Interruptions Done by Clicking Stop Button in Browser

Client browser	Client Operating System	Termination flow	Behavior	idle time $T_d$seconds
IE 6.0	Windows 6.0	$(FA_c)(R_c)$	Client initiates FIN handshake for connection termination after $T_d$seconds followed by TCP RST flag from client	0.19
Opera Mobile 10.0	Windows 6.0	$(FA_c)(R_c)$	Client initiates FIN handshake for connection terminationafter $T_d$seconds followed by TCP RST flag from client	0.1
Safari	iOS 4.0.2	$(FA_c)(R_c)$	Client initiates FIN handshake for connection termination after $T_d$seconds followed by TCP RST flag from client	0.1
Opera Mobile 10.0	Android 2.2	$(FA_c)(R_c)$	Client initiates FIN handshake for connection termination after $T_d$seconds followed by TCP RST flag from client	0.1
Built-in Browser	Android 2.2	$(FA_c)(R_c)$	Client initiates FIN handshake for connection termination after $T_d$seconds followed by TCP RST flag from client	0.13
Built-in Browser	Symbian 3.0	$(R_c)(FA_c)$ (R_s)	Client sends TCP RST flag to terminate the connection after $T_d$seconds followed Fin from Client and followed byTCP RST flag from Server	0.78
Opera Mobile 10.0	Symbian 3.0	$(R_c)(FA_c)$ (R_s)	Client sends TCP RST flag to terminate the connection after $T_d$seconds followed Fin from Client and followed by TCP RST flag from Server	0.19

Table 4.4: Interruptions Done By Closing Browser Application

Client browser	Client Operating System	Termination flow	Behavior	idle time $T_d$seconds
IE 6.0	Windows 6.0	$(FA_c)(R_c)$	Client initiates FIN handshake for connection termination after $T_d$seconds followed by TCP RST flag from client	0.25
Opera Mobile 10.0	Windows 6.0	$(FA_c)(R_c)$	Client initiates FIN handshake for connection terminationafter $T_d$seconds followed by TCP RST flag from client	0.15
Safari	iOS 4.0.2	$(FA_c)(R_c)$	Client initiates FIN handshake for connection termination after $T_d$seconds followed by TCP RST flag from client	0.13
Opera Mobile 10.0	Android 2.2	$(FA_c)(R_c)$	Client initiates FIN handshake for connection termination after $T_d$seconds followed by TCP RST flag from client	0.11
Built-in Browser	Android 2.2	$(FA_c)(R_c)$	Client initiates FIN handshake for connection termination after $T_d$seconds followed by TCP RST flag from client	0.13
Built-in Browser	Symbian 3.0	$(R_c)(FA_c)$ (R_s)	Client sends TCP RST flag to terminate the connection after $T_d$seconds followed Fin from Client and followed byTCP RST flag from Server	0.88
Opera Mobile 10.0	Symbian 3.0	$(R_c)(FA_c)$ (R_s)	Client sends TCP RST flag to terminate the connection after $T_d$seconds followed Fin from Client and followed by TCP RST flag from Server	0.16

Android with Opera Mobile 10.0 and default browser, initiates the termination with a FIN handshake followed by TCP RST flag to the server from the client. While the operating system being used by the client is Symbian with Opera Mobile 10.0 and default browser, and the transmission process is interrupted by close, then the client initiates termination with TCP RST flag followed by TCP FIN flag and then receives TCP RST flag from the server.

4.2.3 Interruption by Reload Action

In Table 4.5, we summarize the results obtained from reload action in the interrupted experiments. From the Table 4.5, we state the following observations.

While the transmission process on a Windows operating system with Internet Explorer and Opera Mobile 10.0 is interrupted by reload, then the client initiates the termination with a FIN handshake followed by TCP RST flag to the server. Similarly, in the case when the client uses iOS with a Safari browser or Android with its built in browser and Opera Mobile 10.0, the transmission process when interrupted by reload leads to initiation of termination action by the client with a FIN handshake followed by TCP RST flag from client to the server. The process of termination differs on Symbian on its built in browser and Opera Mobile 10.0. When the transmission is interrupted by reload, the client initiates the termination with TCP RST flag followed by TCP FIN flag and finally receives a TCP RST flag from the server.

4.2.4 Interruption by Redirection To Another URL Action

In Table 4.6, we summarize the results obtained from redirection action in the interrupted experiments. From the Table 4.6, we state the following observations.

Redirection to another URL is an interruption method, which results in a similar termination process to other interruption methods. On a Windows operating system, with Internet Explorer and Opera Mobile 10.0, the client initiates the termination process with a FIN handshake followed by TCP RST flag to the server when the transmission is interrupted by redirection to another URL. On iOS with Safari browser and Android with its default browser and Opera Mobile 10.0, the termination process is similar to that on Windows. When the transmission is interrupted by a redirection to another URL, the client initiates the termination action with a FIN handshake followed by TCP RST flag to the server.

Redirection to another URL on a Symbian operating system with its default browser and Opera Mobile 10.0 results in a termination action that is initiated by the client with TCP RST flag followed by TCP FIN flag and

Table 4.5: Interruptions Done by Clicking Reload Button in Browser

Client browser	Client Operating System	Termination flow	Behavior	idle time $T_d$seconds
IE 6.0	Windows 6.0	$(FA_c)(R_c)$	Client initiates FIN handshake for connection termination after $T_d$seconds followed by TCP RST flag from client	0.25
Opera Mobile 10.0	Windows 6.0	$(FA_c)(R_c)$	Client initiates FIN handshake for connection terminationafter $T_d$seconds followed by TCP RST flag from client	0.15
Safari	iOS 4.0.2	$(FA_c)(R_c)$	Client initiates FIN handshake for connection termination after $T_d$seconds followed by TCP RST flag from client	0.13
Opera Mobile 10.0	Android 2.2	$(FA_c)(R_c)$	Client initiates FIN handshake for connection termination after $T_d$seconds followed by TCP RST flag from client	0.11
Built-in Browser	Android 2.2	$(FA_c)(R_c)$	Client initiates FIN handshake for connection termination after $T_d$seconds followed by TCP RST flag from client	0.13
Built-in Browser	Symbian 3.0	$(R_c)(FA_c)$ (R_s)	Client sends TCP RST flag to terminate the connection after $T_d$seconds followed Fin from Client and followed byTCP RST flag from Server	0.88
Opera Mobile 10.0	Symbian 3.0	$(R_c)(FA_c)$ (R_s)	Client sends TCP RST flag to terminate the connection after $T_d$seconds followed Fin from Client and followed by TCP RST flag from Server	0.16

Table 4.6: Interruptions Done by Clicking on URL in Browser

Client browser	Client Operating System	Termination flow	Behavior	idle time $T_d$seconds
IE 6.0	Windows 6.0	$(FA_c)(R_c)$	Client initiates FIN handshake for connection termination after $T_d$seconds followed by TCP RST flag from client	0.25
Opera Mobile 10.0	Windows 6.0	$(FA_c)(R_c)$	Client initiates FIN handshake for connection terminationafter $T_d$seconds followed by TCP RST flag from client	0.15
Safari	iOS 4.0.2	$(FA_c)(R_c)$	Client initiates FIN handshake for connection termination after $T_d$seconds followed by TCP RST flag from client	0.13
Opera Mobile 10.0	Android 2.2	$(FA_c)(R_c)$	Client initiates FIN handshake for connection termination after $T_d$seconds followed by TCP RST flag from client	0.11
Built-in Browser	Android 2.2	$(FA_c)(R_c)$	Client initiates FIN handshake for connection termination after $T_d$seconds followed by TCP RST flag from client	0.13
Built-in Browser	Symbian 3.0	$(R_c)(FA_c)$ (R_s)	Client sends TCP RST flag to terminate the connection after $T_d$seconds followed Fin from Client and followed byTCP RST flag from Server	0.88
Opera Mobile 10.0	Symbian 3.0	$(R_c)(FA_c)$ (R_s)	Client sends TCP RST flag to terminate the connection after $T_d$seconds followed Fin from Client and followed by TCP RST flag from Server	0.16

the server sends back TCP RST flag to the client as its response.

From the results presented in the Tables 4.3, 4.4, 4.5 and 4.6 we come to the following conclusions:

1. First, the majority of interrupted cases used TCP RST flag to tear-down the connection.

2. Secondly, we observe a different behaviour of the same client browser Opera mobile browser 10.0 on different OS. In case of Android and Windows OS, the connection tear- down is done by a TCP FIN flag from the client, but in case of Symbian client operating system it used a TCP RST flag from server to close the TCP connection.

3. Thirdly, in the cases of client using Windows OS, Interruption cases involve connection tear–down by a browser TCP RST flag. For this behaviour we state the following reasons: In several experiment cases the connection termination process begins with TCP FIN flag from the client. Server acknowledges it by sending ACK. This is followed by a single or multiple TCP RST flags from the client upon the acceptance of DATA segments from the server which are already being transferred during the connection. Another reason is that the Client may send RST flag to tear-down the connection (Other than to abort the TCP flow when a problem has occurred) to minimize the number of TIME–WAIT states on the server [6].

4. Finally, in various test cases of client using Symbian operating system we have observed TCP RST flag from the server. One of the possible reasons for this is that connection termination begins with RST flag from the Client which causes server to go to CLOSED state. And upon the reception of FIN flag from the client, server responds with TCP RST flag.

4.3 Interruption Criterion

User can interrupt the on-going web transfer by using one of the actions mentioned in section 3.2 which indicates an interruption event. It is difficult to distinguish among these user interrupted actions as mentioned above. They can be identified by observing the evolution of TCP flow, described in this section.

In an uninterrupted TCP flow, the connection is initiated by a SYN handshake .Then a GET request is sent from the client, followed by a DATA segments from the server .Then the TCP connection tear-down is done by a TCP FIN flag from the server or TCP RST flag from client.

Conversely, user interruption causes an early termination of data flow from the server. The connection termination events depend on the operating

system used at the client side, i.e. MAC and Windows clients initiate the FIN handshake by sending TCP FIN flag to the server. This is followed by a single or multiple TCP RST flags from the client upon the acceptance of DATA segments which are already being sent by the server during the connection. While Android and Symbian client immediately send the TCP RST flag to terminate the connection followed by a TCP FIN flag to which the server replies with RST flag. In most of the cases user interrupted action originates an asynchronous TCP window mechanism.

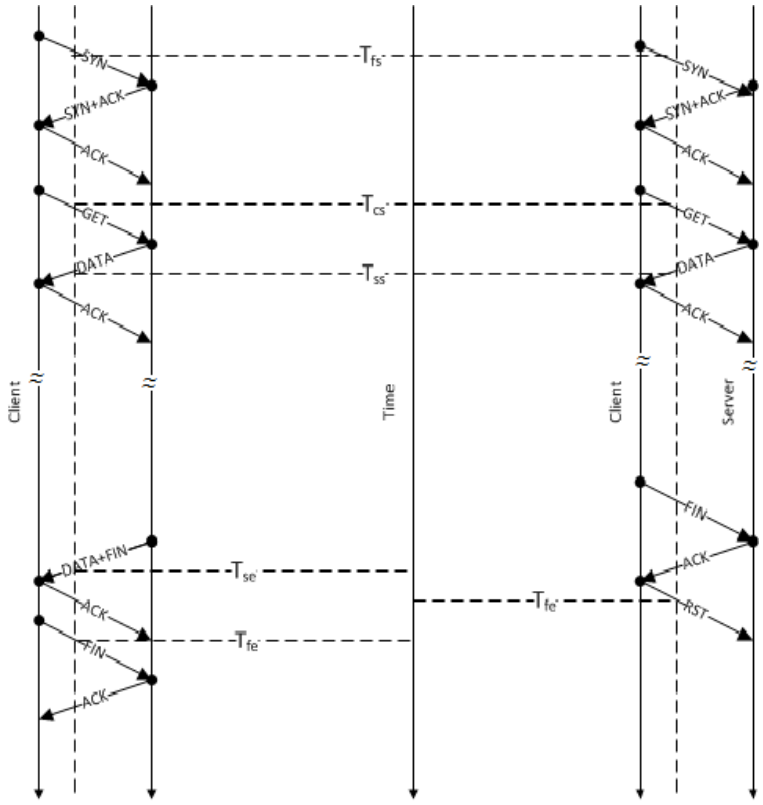


Figure 4.1: Timing Diagram

Figure 4.1 describes the evolution of TCP for interrupted flow versus uninterrupted flow. the vertical line in the middle gives the various time routines that are defined as follows.

- T_{fs} =Time of flow start
- T_{fe} = Time of flow end
- T_{cs} = Client request start time
- T_{ss} = server start time

- T_{se} = server end time

We use the criterion presented in [8] to differentiate TCP RST flags generated in the case of interrupted and uninterrupted connections . A primary condition to categorize the interruption flow is introduced which is called "*Eligibility*". TCP connections in which server sent data, but no TCP FIN or RST flags in the end and client sends an TCP RST flag are considered to be eligible.It is further represented by the following notation in 4.1

$$Eligiblity =: \neg(FINs \vee RSTs) \wedge DATAs \wedge RSTc \tag{4.1}$$

As described in [8] this condition alone is not enough to differentiate between interrupted and uninterrupted connections. As TCP RST flags from client can be observed before the connection tear down by server which occur due to HTTP settings. Any server in general waits for a time to expire after the last DATA packet has been sent before closing the connection according to HTTP protocol settings. After the data transfer is completed, connections closed during this ideal time are also being classified as interrupted.

To measure this difference a parameter t_{gap} is considered. t_{gap} is the difference between the last data packet from the server and actual flow end. However, a majority of connections are closed after receiving the last DATA segment. So, an additional time constraint is taken to distinguish interrupted flows in the eligible flows. t_{gap} is expected to be independent from TCP timings and upper bounded by a function of RTT. Normalized $\hat{t}_{gap}$ is defined in equation 4.2

$$\hat{t}gap = tgap/(\alpha.\mu RTT + \beta.\sigma RTT) \tag{4.2}$$

Where μRTT and σRTT are the average and standard deviation of the connection RTT respectively. From the observations the eligibility criteria are combined with normalized t_{gap} and the interruption flow criteria is defined as follows in equation 4.3.

$$Interrupted := Eligible \wedge (\hat{t}gap \leq 1) \tag{4.3}$$

This criteria is used to differentiate user behaviour based on TCP termination flags and is implemented by a tool called TCP Statistical Analysis Tool (TSTAT) which was developed by the research group at politecnico di torino [8]. Tstat is an analyzer tool which looks into the packet traces to track the connection process.

In Table 4.7 we summarize the results obtained by implementing interruption criterion on our results.

- R_i indicates TCP RST flag in case of interrupted cases.
- R_u indicates TCP RST flag in case of uninterrupted cases.

Table 4.7: Interruption Criterion Results

Client operating system	user action	Client browser	C_a	RSTflag
Windows	uninterrupted	I.E	40	R_u
		Opera	40	R_u
	Stop	I.E	40	R_i
		Opera	40	R_i
	Close	I.E	40	R_i
		Opera	40	R_i
	Reload	I.E	40	R_i
		Opera	40	R_i
	Redirecting to URL	I.E	40	R_i
		Opera	40	R_i
iOS	uninterrupted	Safari	40	–
	Stop	Safari	40	R_i
	Close	Safari	40	R_i
	Reload	Safari	40	R_i
	Redirecting to URL	Safari	40	R_i
Android 2.2	uninterrupted	Default browser	40	–
		Opera	40	–
	Stop	Default browser	40	R_i
		Opera	40	R_i
	Close	Default browser	40	R_i
		Opera	40	R_i
	Reload	Default browser	40	R_i
		Opera	40	R_i
	Redirecting to URL	Default browser	40	R_i
		Opera	40	R_i
Symbian	uninterrupted	Default browser	40	–
		Opera	40	–
	Stop	Default browser	40	R_i
		Opera	40	R_i
	Close	Default browser	40	R_i
		Opera	40	R_i
	Reload	Default browser	40	R_i
		Opera	40	R_i
	Redirecting to URL	Default browser	40	R_i
		Opera	40	R_i

- $C_a$indicates number of iterations satisfying interrupted criterion.

The Results from running our trace files on TSTAT according to Interruption Criterion satisfy 100%. As we observed TCP RST flags in both interrupted and uninterrupted case, we clearly idealized the TCP RST flags that have occurred from user dissatisfaction.

Chapter 5

Conclusions

In this thesis we have found the various causes of TCP RST flags to occur during a mobile Web session. We performed active tests for which an experiment testbed is developed to capture the packet traces. More than 1200 traces files captured are analyzed using perl script to dicern the flow and focusing on the RST flag during the flow. We observe some interesting behaviour as explained in the results and conclude that user behavior shows a huge rise in the number of RSTs and TCP RST can be used to provide a good analysis for user dissatisfaction. From the results presented in section 4.1 and 4.2, we have observed and categorised TCP RST flags in different scenarios in case of mobile web browsing. From these observations we conclude that TCP RST flag cannot be completely relied upon as an indication of user dissatisfaction in mobile web browsing. This is due to the reason explained in the section 4.1 that client operating system is a reason for generating TCP RST in case of uninterrupted scenario. We further differentiate the TCP RST flags generated due to interrupted and uninterrupted connections by implementing our results according to interruption criterion in TSTAT.

Our results can help mobile Internet service providers in finding out user generated RST flags and can immediatly distinguish an interrupted and uninterrupted connection. This can help in improving the network for better service to mobile Web users.

5.1 Future Work

An extension to more complicated scenarios can be done by introducing different pictures and videos in the mobile web pages. As mobile web pages use different languages for different browsers and OS, the web pages can be developed using AJAX, CSS, XHTML and other languages. Traces can be captured in real time for more effective analysis and bulk traffic can be used which is accessed from a remote server.

Bibliography

[1] Cellular News. Mobile internet users to top 1.7 billion by 2013. `http://www.itu.int/ITUD/ict/newslog/Mobile+Internet+Users+To+Top+17+Billion+By+2013.aspx`, June 2008.

[2] Virpi Roto. *Web browsing on mobile phones-characteristics of user experience*. PhD thesis, 2006.

[3] Matt Jones and Gary Marsden. *Mobile Interaction Design*. John Wiley & Sons, February 2006.

[4] A. Omotayo and C. Williamson. Multi-layer analysis of web browsing performance for wireless pdas. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 660 – 667, nov. 2004.

[5] Venkata N. Padmanabhan and Jeffrey C. Mogul. Improving http latency. *Computer Networks and ISDN Systems*, 28(1-2):25 – 35, 1995. Selected Papers from the Second World-Wide Web Conference.

[6] Martin Arlitt and Carey Williamson. An analysis of tcp reset behaviour on the internet. *SIGCOMM Comput. Commun. Rev.*, 35:37–44, January 2005.

[7] Naomi Ramos Concordia Chen, Mrunal Mangrulkar and Mahasweta Sarkarr. Trends in tcp/ip retransmissions and resets.

[8] D. Rossi, M. Mellia, and C. Casetti. User patience and the web: a hands-on investigation. In *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, volume 7, pages 4163 – 4168 vol.7, dec. 2003.

[9] Adeel Ashfaq and Umer Bilal. Study of abnormal tcp/http connection, 2010.

[10] Antero Kivi. Measuring mobile service usage; methods and measurement points. *Int. J. Mob. Commun.*, 7:415–435, March 2009.

[11] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168, 6093.

[12] S. Floyd. Inappropriate TCP Resets Considered Harmful. RFC 3360 (Best Current Practice), August 2002.

[13] Balachander Krishnamurthy, Jeffrey C. Mogul, and David M. Kristol. Key differences between http/1.0 and http/1.1. *Computer Networks*, 31(11-16):1737 – 1751, 1999.

[14] Maryam Kamvar, Melanie Kellar, Rajan Patel, and Ya Xu. Computers and iphones and mobile phones, oh my!: a logs-based comparison of search users on different devices. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 801–810, New York, NY, USA, 2009. ACM.

[15] Jeonghe Yi and Farzin Maghoul. Mobile search pattern evolution: the trend and the impact of voice queries. In *Proceedings of the 20th international conference companion on World wide web*, WWW '11, pages 165–166, New York, NY, USA, 2011. ACM.

[16] K.C. Claffy, H.-W. Braun, and G.C. Polyzos. A parameterizable methodology for internet traffic flow profiling. *Selected Areas in Communications, IEEE Journal on*, 13(8):1481 –1494, oct 1995.

[17] K. Gopalan and Tzi cker Chiueh. Improving route lookup performance using network processor cache. In *Supercomputing, ACM/IEEE 2002 Conference*, page 22, nov. 2002.

[18] Sujan Shrestha. Mobile web browsing: usability study. In *Proceedings of the 4th international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology*, Mobility '07, pages 187–194, New York, NY, USA, 2007. ACM.

[19] I. Din, N.A. Saqib, and A. Baig. Passive analysis of web traffic characteristics for estimating quality of experience. In *GLOBECOM Workshops, 2008 IEEE*, pages 1 –5, 30 2008-dec. 4 2008.

[20] Patrik Arlos, Markus Fiedler, and Arne Nilsson. A distributed passive measurement infrastructure. In Constantinos Dovrolis, editor, *Passive and Active Network Measurement*, volume 3431 of *Lecture Notes in Computer Science*, pages 215–227. Springer Berlin / Heidelberg, 2005.

[21] Endace dag. `http://www.endace.com/endace-dag-high-speed-packet-capture-cards.html`.

[22] Palo Alto. Googles android becomes the worlds leading smart phone platform. `http://www.canalys.com/pr/2011/r2011013.html`, 2010.

[23] Cheskis Alexander. Improvement of tcp connection throughput over noisy environment, 2006.

Appendix

Appendix A

Code space

A.1 Perl Code

```
#!/usr/bin/perl
opendir THEDIR, '.';
while($filename  = readdir THEDIR)
{

print "$filename\n ";
      @in = grep {/\.txt$/}readdir THEDIR ;

for ($i=0; $i<=40 ;$i++)
{

 $value = $in[$i];

print"$value\n";

open (myinput1,"$value") or die "could not open fihdsle";
$nume =0;
while (<myinput1>)
{
$nume++;
my $line =$_;

@details = split/:/,$_;
@required= split/\W/,@details[13];
$flag=@required[2];
$time=@details[3];
$port =@details[15];
$requiredTSsc = $TSsc;
```

```
$requiredTFA = $TFA;
$requiredtF == $tF;
$requiredtimeAFA == $timeAFA;


$requiredtimeRA = $timeRA;



for ($line)
{
if ($flag =~/SA/)
{

 if ($port!= 80)
{
$TSsc=@details[3];
print       "2 S--->C $flag\n";

last;
}
else
{



last;
}

}

if ($flag =~/S/)
{

 if ($port!= 80)
{



last;
}
else
{
 print "1 C--->S $flag\n";
```

```
last;

}
}

if ($flag =~/R/)
{

 if ($port!= 80)
{

print "8 S---->C $flag\n";
last;
}
else
{


print"9 C---->S $flag\n";
last;
}

}



if ($flag =~/F/)
{
if($port!=80)
 {

$TFsc=@details[3];
print       "4 S---->C $flag\n";
last;
}
else
{

$TFcs=@details[3];

print       "6 C---->S $flag\n";
last;
}
```

```
}


if($flag =~/A/)
{
if(($requiredTSsc == $TSsc)&&($TSsc != 0))
{

if($port!=80)
 {

$TAcs=@details[3];

$TSsc =0;

last;
}
else
{

$TAcs=@details[3];
print       "3 C-->S $flag\n";
$TSsc =0;

last;
}
last;

}

}

if($flag =~/A/)
{
if(($TFcs !=0)||($TFsc!=0))
{

if($port!=80)
 {

$TFcs =0;
print           "7 S-->C $flag\n";
last;
}
```

```
else
{
$TFsc =0;
print          "5 C---->S $flag\n";
last;
}
last;

}

if (($flag =~/AP/)&&($port !=80))

{
@timeAA=@details[3];


$requiredtimeAA =$timeAA[-1];
last;

}
}
}
}
}

last;
}
closedir THEDIR;
```

A.2 Uninterrupted TCP Flow

Figure A.1: Text_IE_Apache_Windows_Normal

```
C--->S S
S---->C SA
C--->S A
S--->C FA
C--->S A
C--->S FA
S--->C A
C--->S R
```

Figure A.2: Text_Opera_Apache_Windows_Normal

```
C--->S S
S---->C SA
C--->S A
S--->C FA
C--->S A
C--->S FA
S--->C A
C--->S R
```

Figure A.3: Text_Safari_Apache_MAC_Normal

```
C--->S S
S---->C SA
C--->S A
C--->S FA
S--->C A
S--->C FA
C--->S A
```

Figure A.4: Text_Default browser_Apache_Android 2.2_Normal

```
C--->S S
S---->C SA
C--->S A
C--->S FA
S--->C A
S--->C FA
C--->S A
```

Figure A.5: Text_Opera_Apache_Android 2.2_Normal

```
C--->S S
S---->C SA
C--->S A
C--->S FA
S--->C A
S--->C FA
C--->S A
```

Figure A.6: Text_Default browser_Apache_Symbian_Normal

```
C--->S S
S---->C SA
C--->S A
C--->S FA
S--->C A
S--->C FA
C--->S A
```

Figure A.7: Text_Opera_Apache_Symbian_Normal

```
C--->S S
S---->C SA
C--->S A
C--->S FA
S--->C A
S--->C FA
C--->S A
```

A.3 Interrupted TCP Flow

A.3.1 TCP Flow in Stop Action

Figure A.11: Text_Default browser_Apache_Android 2.2_Stop

```
C--->S S
S---->C SA
C--->S A
C--->S FA
S--->C A
C--->S AR
C--->S R
C--->S R
C--->S R
```

Figure A.8: Text_IE_Apache_Windows_Stop

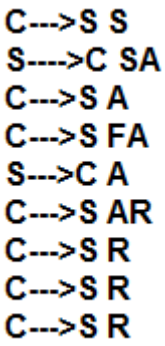


Figure A.9: Text_Opera_Apache_Windows_Stop

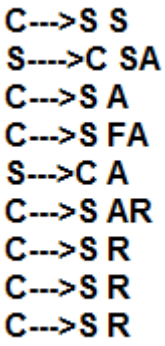
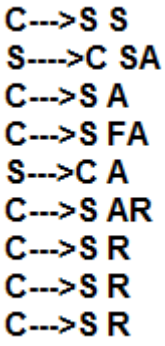


Figure A.10: Text_Safari_Apache_MAC_Stop



A.3.2 TCP Flow in Close Action

Figure A.12: Text_Opera_Apache_Android 2.2_Stop

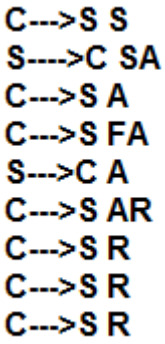


Figure A.13: Text_Default browser_Apache_Symbian_Stop

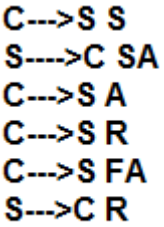


Figure A.14: Text_Opera_Apache_Symbian_Stop

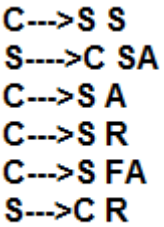


Figure A.15: Text_IE_Apache_Windows_Close

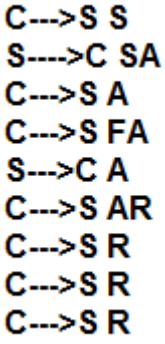


Figure A.16: Text_Opera_Apache_Windows_Close

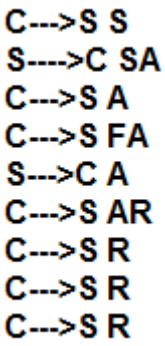


Figure A.17: Text_Safari_Apache_MAC_Close

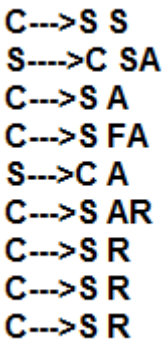


Figure A.18: Text_Default browser_Apache_Android 2.2_Close

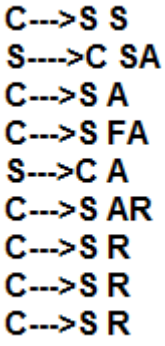


Figure A.19: Text_Opera_Apache_Android 2.2_Close

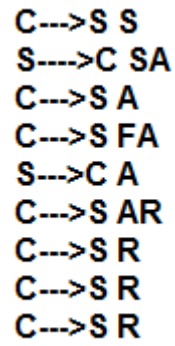


Figure A.20: Text_Default browser_Apache_Symbian_Close

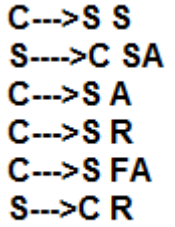
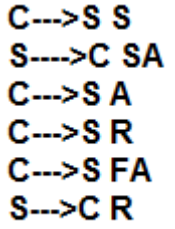


Figure A.21: Text_Opera_Apache_Symbian_Close



A.3.3 TCP Flow in Reload Action

Figure A.22: Text_IE_Apache_Windows_Reload

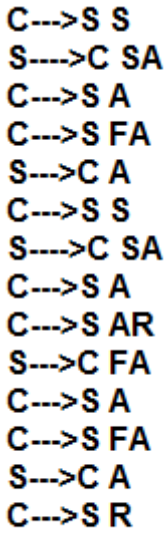


Figure A.23: Text_Opera_Apache_Windows_Reload

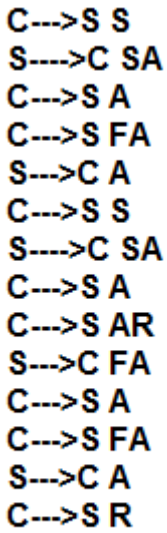


Figure A.24: Text_Safari_Apache_MAC_Reload

C--->S S
S---->C SA
C--->S A
C--->S FA
S--->C A
C--->S S
S---->C SA
C--->S A
C--->S AR
S--->C FA
C--->S A
C--->S FA
S--->C A
C--->S R

Figure A.25: Text_Default browser_Apache_Android 2.2_Reload

C--->S S
S---->C SA
C--->S A
C--->S FA
S--->C A
C--->S S
S---->C SA
C--->S A
C--->S AR
S--->C FA
C--->S A
C--->S FA
S--->C A
C--->S R

A.3.4 TCP Flow in Redirect to Another URL Action

Figure A.26: Text_Opera_Apache_Android 2.2_Reload

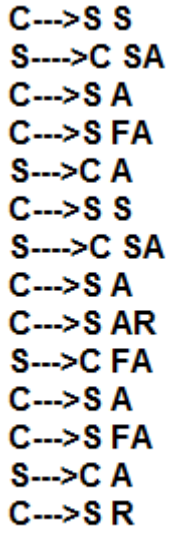


Figure A.27: Text_Default browser_Apache_Symbian_Reload

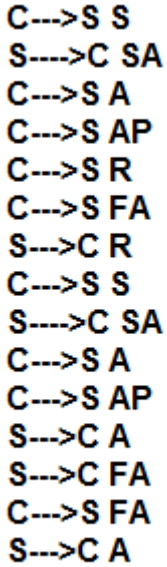


Figure A.28: Text_Opera_Apache_Symbian_Reload

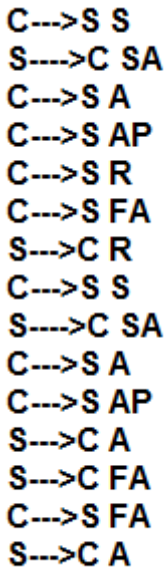


Figure A.29: Text_IE_Apache_Windows_Redirect to another URL

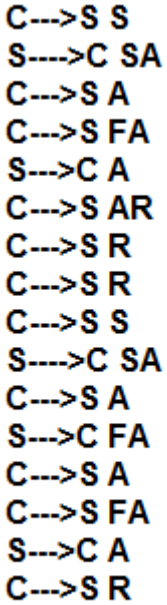


Figure A.30: Text_Opera_Apache_Windows_Redirect to another URL

```
C--->S S
S---->C SA
C--->S A
C--->S FA
S--->C A
C--->S AR
C--->S R
C--->S R
C--->S S
S---->C SA
C--->S A
S--->C FA
C--->S A
C--->S FA
S--->C A
C--->S R
```

Figure A.31: Text_Safari_Apache_MAC_Redirect to another URL

```
C--->S S
S---->C SA
C--->S A
C--->S FA
S--->C A
C--->S AR
C--->S R
C--->S R
C--->S S
S---->C SA
C--->S A
S--->C FA
C--->S A
C--->S FA
S--->C A
C--->S R
```

Figure A.32: Text_Default browser_Apache_Android 2.2_Redirect to another URL

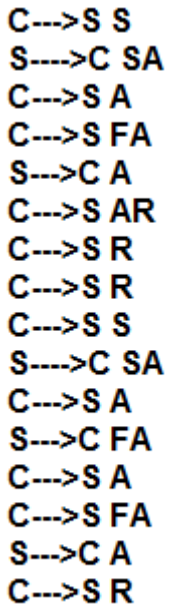


Figure A.33: Text_Opera_Apache_Android 2.2_Redirect to another URL

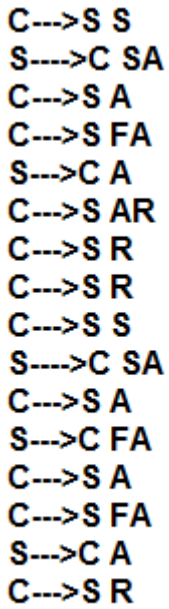


Figure A.34: Text_Default browser_Apache_Symbian_Redirect to another URL

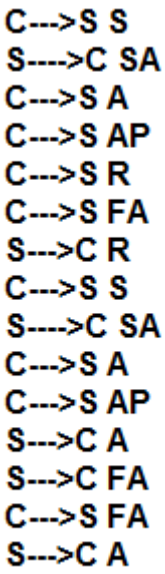


Figure A.35: Text_Opera_Apache_Symbian_Redirect to another URL