CLEMSON UNIVERSITY

# Timeslice tool

CPSC#852 Project

Yaolin Zhang  Yunqing Zhang

29/04/2014

# Introduction

## Description

Timeslice tool intends to analysis network trace and then show categorized network statistics within a certain network community based on the analysis. It will be pretty useful to configure a given network according to those valuable statistics and meet maximum resource utilization rate.

## What we have done

The objectives behind developing this tool were as follows:

- Analysis network trace data and get some useful statistics such as bandwidth in a given time interval, number of web pages in a given time interval and so on.
- Ensure that the program can handle large traces (in the order of gigabytes) as well as multiple parts of a single packet trace.
- Enable the tool to run in different modes, which produce output of different granularity (w.r.t. the level of detail).

# Background

Nowadays, the world actually is web-oriented. It connects everything and everyone in the world. If a network were to be down for a period of time productivity would decline, and in the case of public service departments the ability to provide essential services would be compromised. So network monitoring is a difficult and demanding task that is a vital part of a Network researcher job for a network community which posses large network of data and where network usage is extensive.

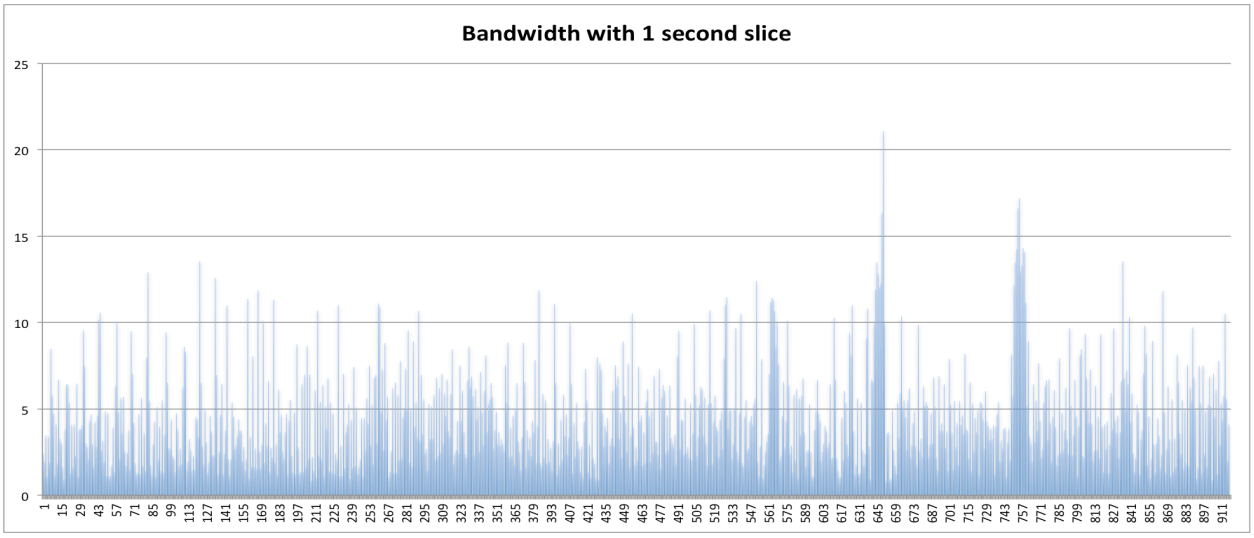
Our timeslice tool takes a large trace file generated by *tcpdump* tool as an input. The primary task of the *tcpdump* tool is to capture raw data at an interface and dump it into a trace file. This trace file containing raw data is basically some packets in their raw form which need to be segregated back into packets simple enough to be used for further processing and analysis. This segregation is achieved using the ***pcap***

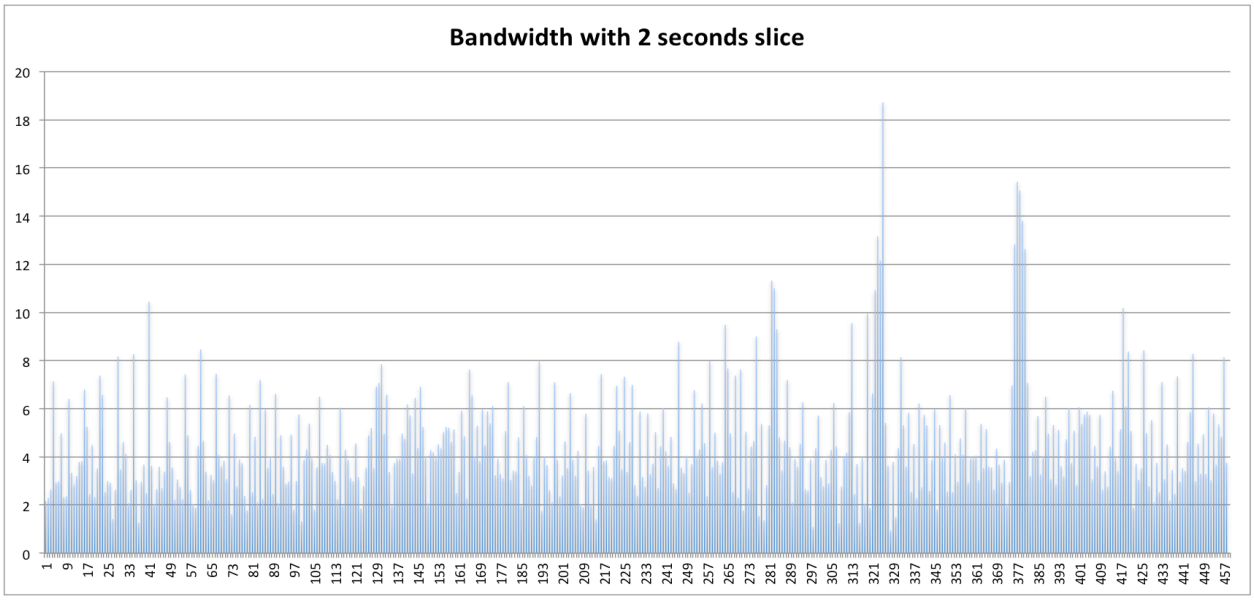
library. The packets returned by the *pcap* library are then passed to the underlying functions.

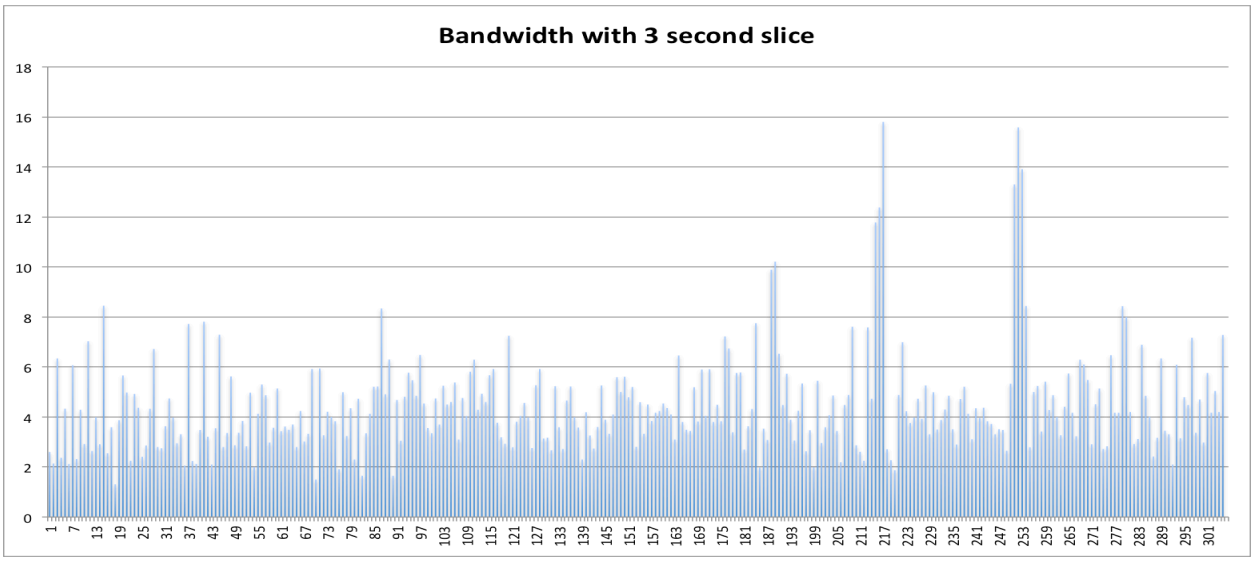
These functions extract information from the packets and place it in the data structures specifically designed to hold the extracted values. After extracting data from all the packets, the data structures are finally processed to provide aggregate statistics.

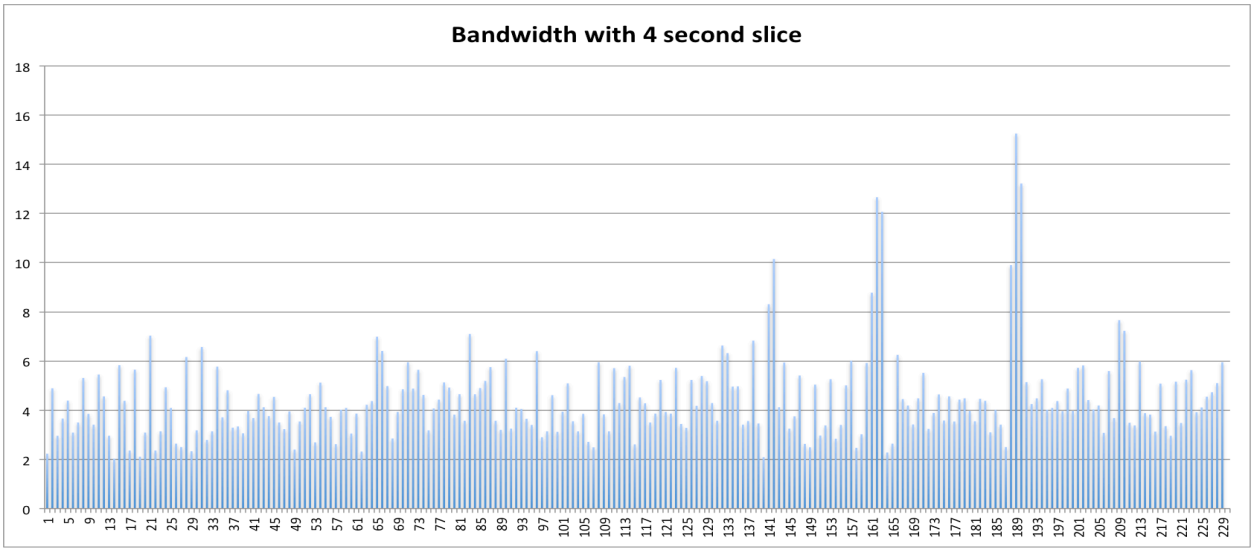
# Result

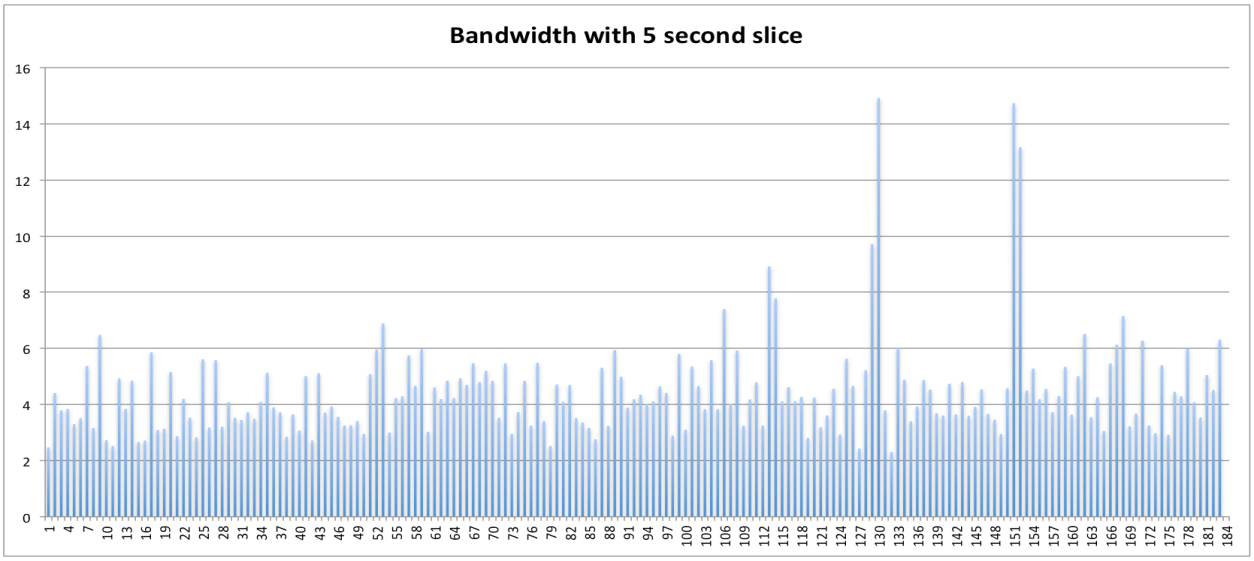
## 1. Bandwidth with different time slice



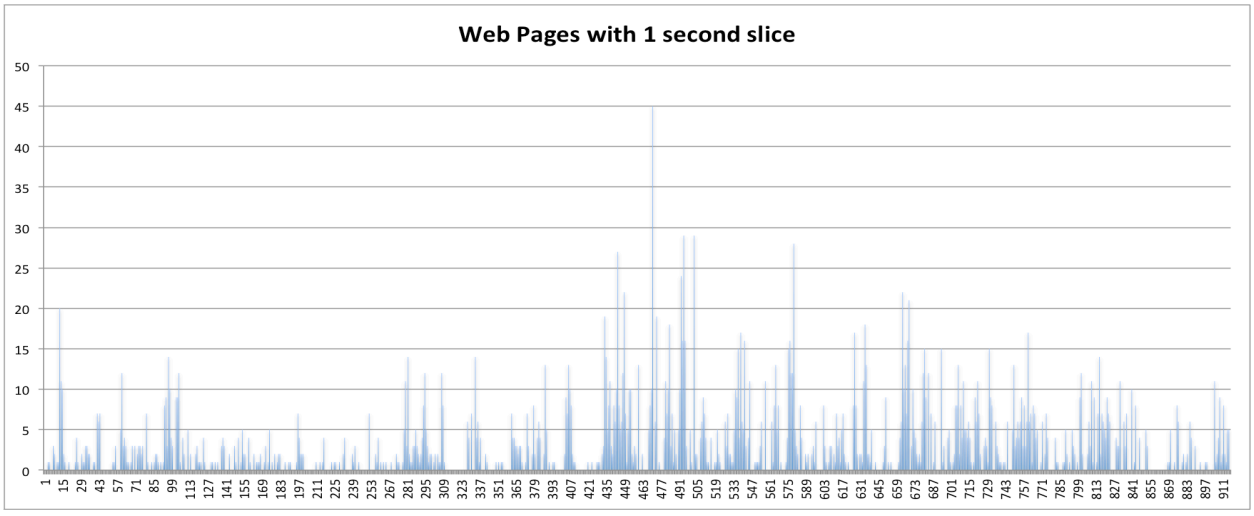



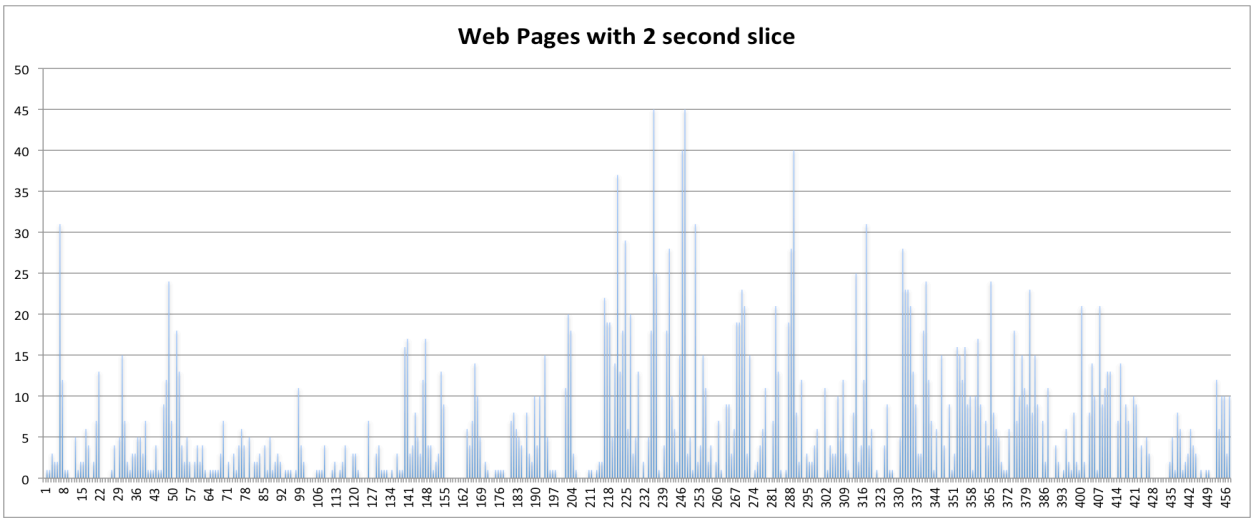

Bandwidth with 3 second slice
0
2
4
6
8
10
12
14
16
18
1 7 13 19 25 31 37 43 49 55 61 67 73 79 85 91 97 103 109 115 121 127 133 139 145 151 157 163 169 175 181 187 193 199 205 211 217 223 229 235 241 247 253 259 265 271 277 283 289 295 301

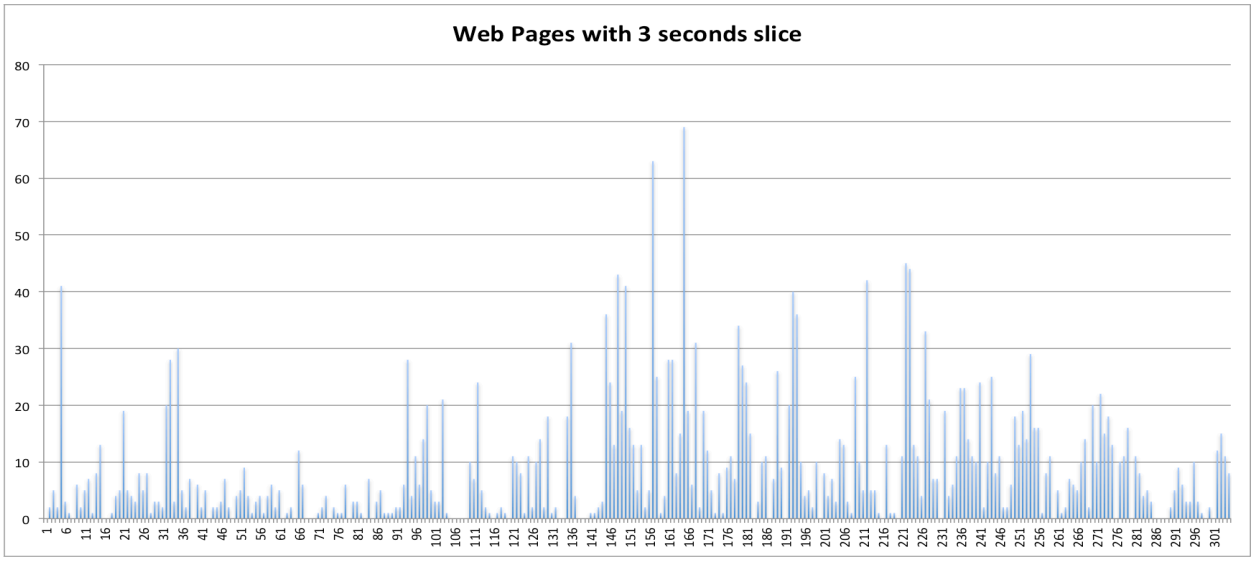

Bandwidth with 4 second slice
0
2
4
6
8
10
12
14
16
18
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97 101 105 109 113 117 121 125 129 133 137 141 145 149 153 157 161 165 169 173 177 181 185 189 193 197 201 205 209 213 217 221 225 229

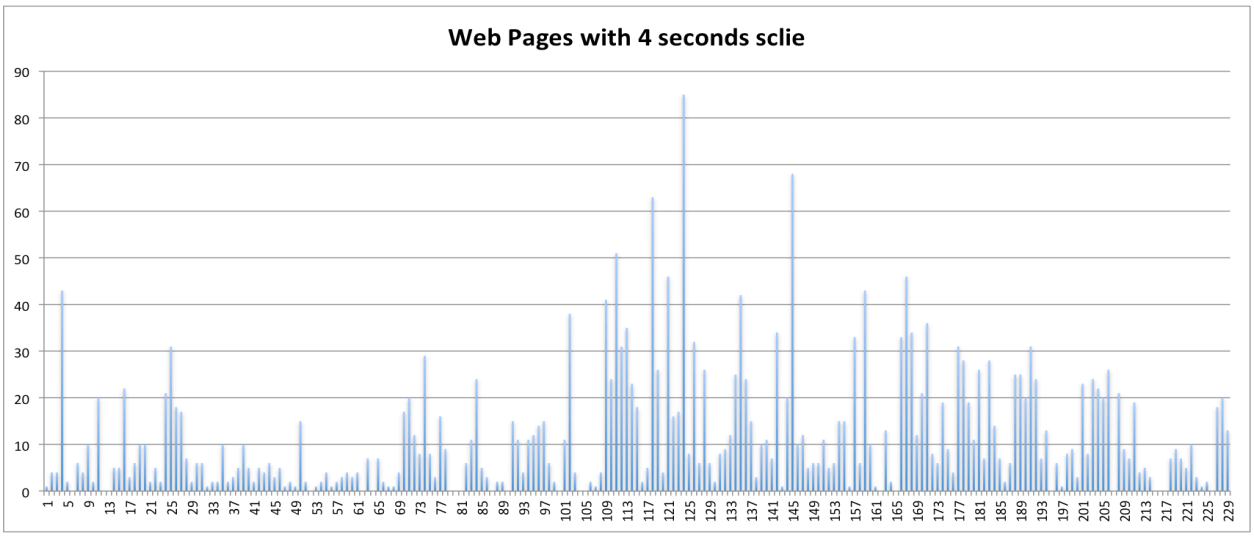

Bandwidth with 5 second slice
0
2
4
6
8
10
12
14
16
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94 97 100 103 106 109 112 115 118 121 124 127 130 133 136 139 142 145 148 151 154 157 160 163 166 169 172 175 178 181 184

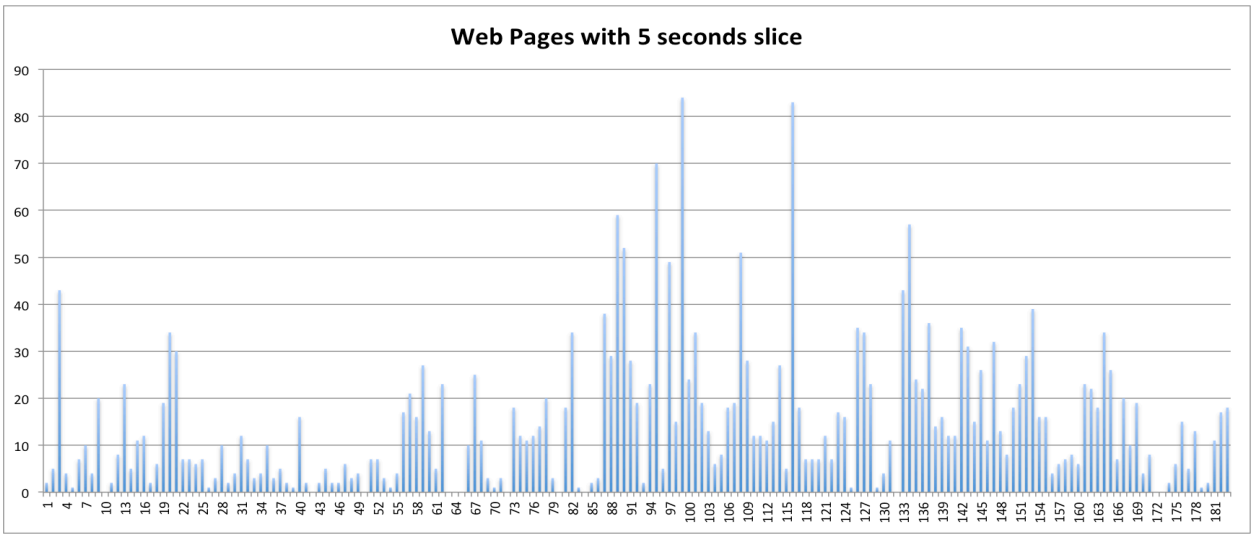
# 2. Web pages with different time slie



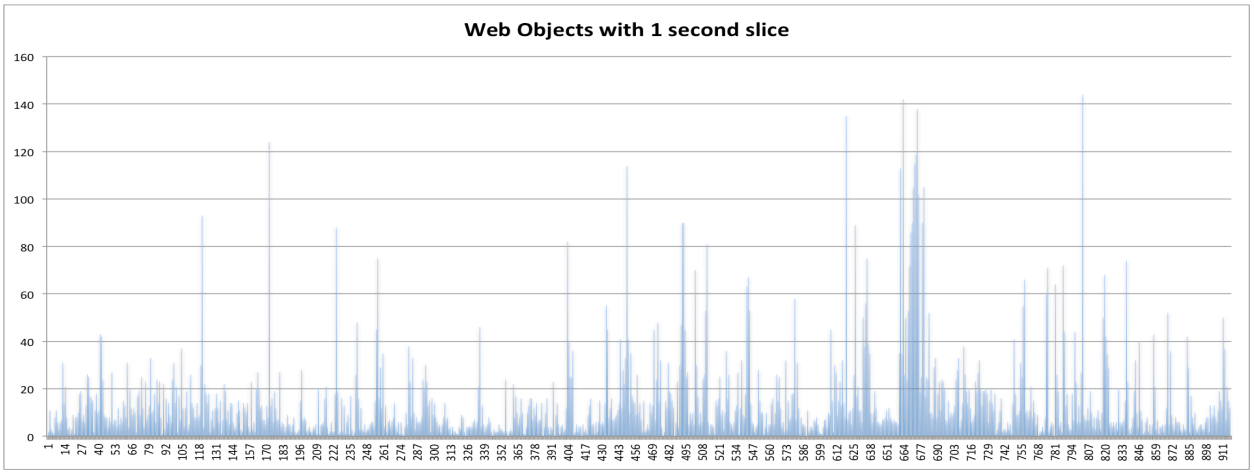



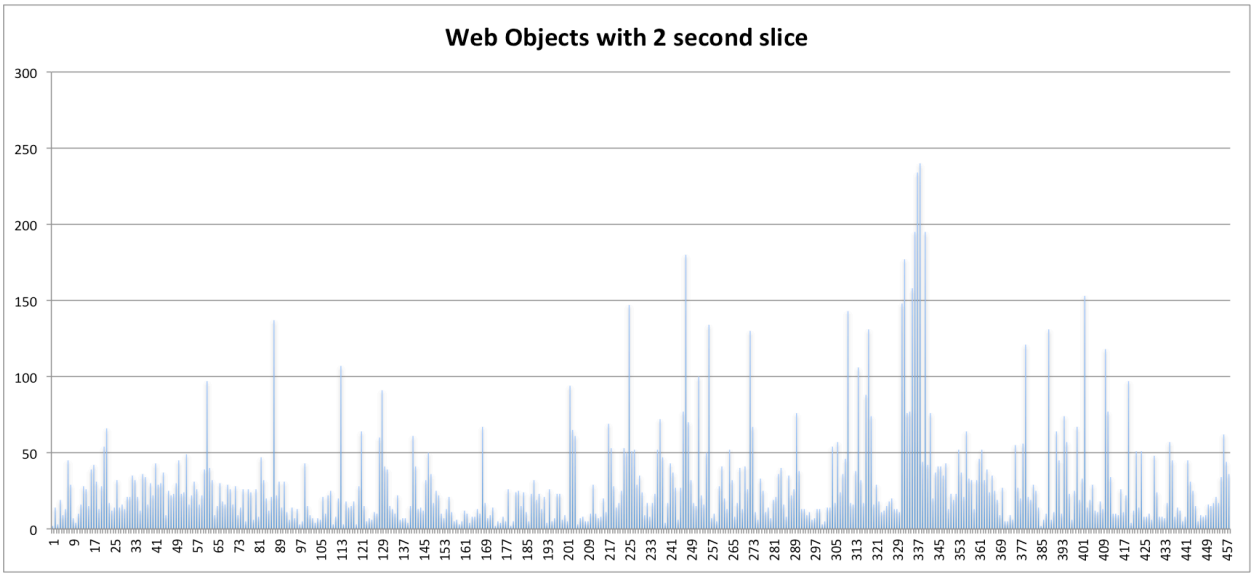



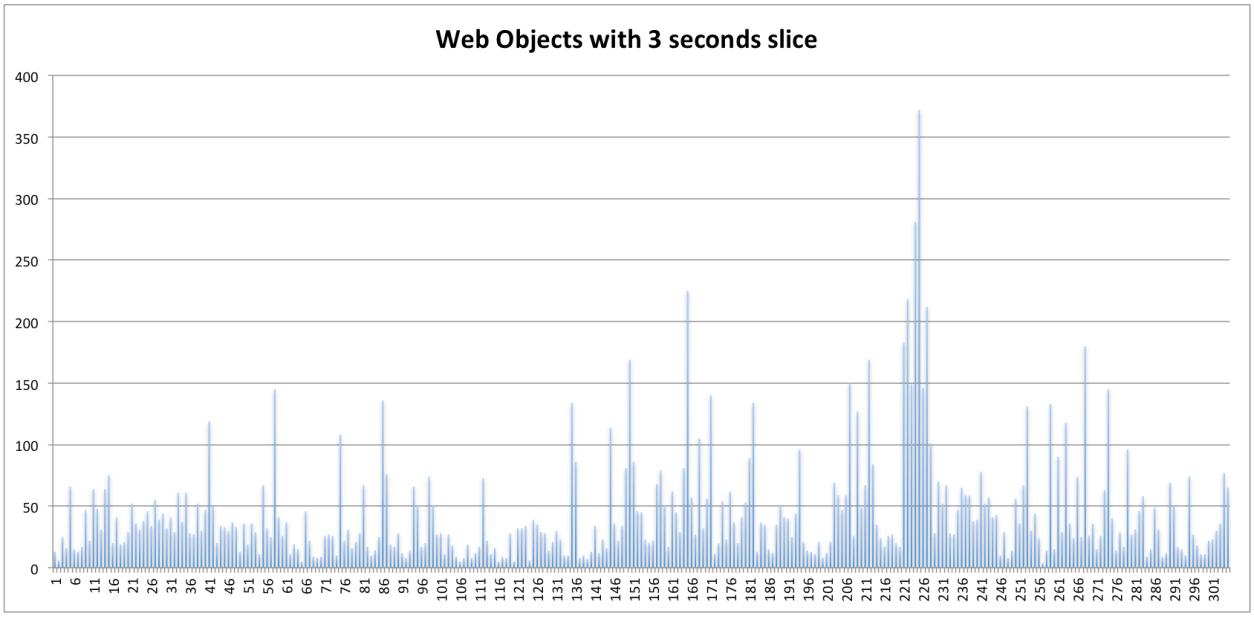

Web Pages with 4 seconds sclie
90
80
70
60
50
40
30
20
10
0
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97 101 105 109 113 117 121 125 129 133 137 141 145 149 153 157 161 165 169 173 177 181 185 189 193 197 201 205 209 213 217 221 225 229

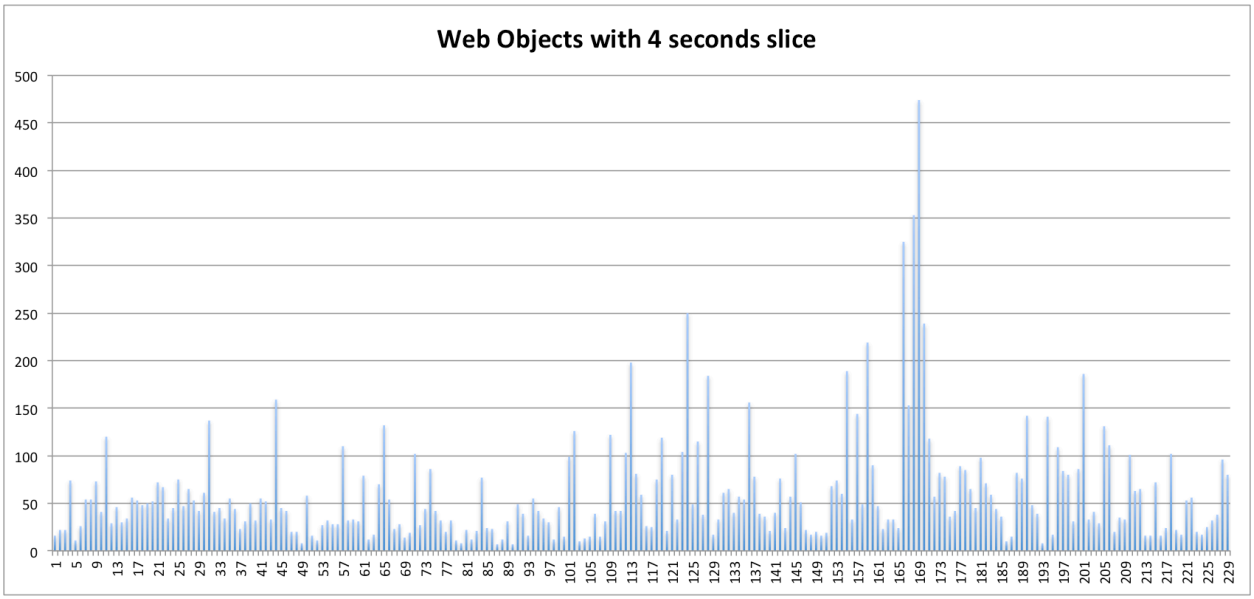

Web Pages with 5 seconds slice
90
80
70
60
50
40
30
20
10
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94 97 100 103 106 109 112 115 118 121 124 127 130 133 136 139 142 145 148 151 154 157 160 163 166 169 172 175 178 181

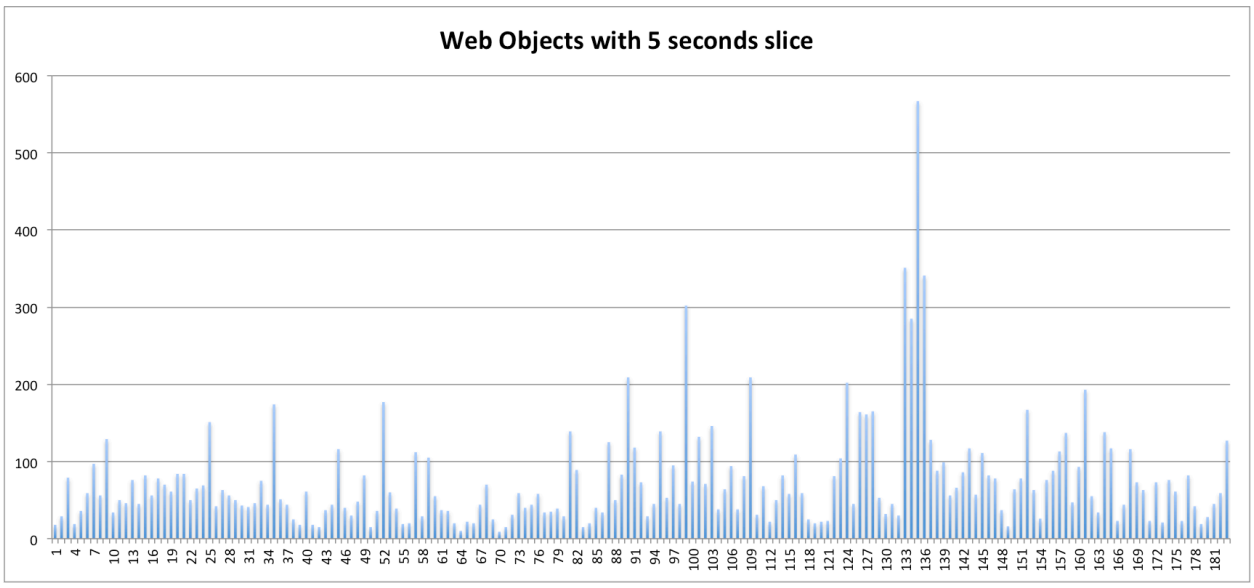
# 3. Objects in Web pages with different time slie






# Future Work

The motivation for future changes to Timeslice can be grouped into three main categories:

- **Improving flow classification based on its application type:** As of now, flows are tagged based on their payload signature and their origin or source endpoints (if traffic from one of the end points has been tagged as BitTorrent, then all the traffic from that endpoint is deemed to be BitTorrent traffic). In future we would like to move to a less invasive mode of profiling which

depends more on traffic patterns and flow characteristics than on payload signatures. The data for making such decisions is already present in the structures created in this tool. Possible ways of classifying data include graph theory, cluster analysis, machine learning etc. Refer to the appendix for more information regarding ongoing research as well as comparable commercial tools.

- **Improving and modifying the input mechanism:**
  As of now, the tool can handle trace files of sizes up through 5 GB. In order to handle larger traces, the tool should be able to store intermediate results in secondary memory (either in a flat file or a database). That shall allow the tool to process information in separate files while providing a global overview. As of now, the tool captures just the TCP packets. In future, we would like the tool to process both TCP and UDP packets. The tool should also have the capability to look at live traces (as opposed to offline dumps).
- **Improving accuracy and code structure**
  The current model of calculating the average RTT for a connection involves storing all the older packets of the connection and going through them to calculate the RTT. This approach consumes a lot of space and is inefficient. A better way of doing this would be to hold the pertinent data in the connection structure itself, reducing the space requirement and improving the accuracy of readings. This is imperative for the correct implementation of the previous requirement.

# Appendix I

We wrote all the codes by ourselves. We also wrote a Makefile to help compile the whole project so you can just type make to compile our project and after that you will get a executable file called run then you can type "./run –h" to show the details about how to run our project.

I have tested my program with the trace files in the website http://people.cs.clemson.edu/~jmarty/traces/ including 20140421

_130127440.pcap and 20140421_130127480.pcap. All the results that show in this paper are got from 20140421_130127440.pcap. You can run our project with any trace file as you want.

An example to run our project:

./run -t 2 -p a -b k trace_file_name_here Run the timeslie with 2 seconds(-t) slice interval, print all the statistic information(-p a), print data size and bandwidth in KB(-b k)

The current version of the timeslice tool has the following code files

| **Filename** | **Purpose** |
|---|---|
| util.h | Contains all the data structures definition and functions definition |
| util.c | Contains implementation of functions which are defined in util.h |
| main.c | Contains the main program, does the option parsing and call the pcap library with appropriate parameters and a designated handler |

## Data structures:

The basic data structures used in this tool are tabulated below:

| Name | Description |
|---|---|
| Struct packet | This is the simplest structure which stores packet level information for each packet returned by the pcap library. Storing that information allows us to abstract out the differences in packet formats to a single function, which actually reads the data into the packet structure. |
| Struct packetNode | This structure holds data for a packet in a packets list |

| | |
|---|---|
| Struct packetListHead | This structure holds data that represents a list of packets that belong to a same slice |
| Struct sliceNode | This structure holds all the information about a slice in a slice list |
| Struct sliceListHead | This structure holds data that represents a list of slice given a trace file as an input |

## Control flow

- The main.c file is the driver as it has the main program contained in it. Firstly, it parses all the parameters and do some necessary initialization. Next, it passes a function pointer to the function *processPacket* to the *pcap* library which is called each time a complete frame is extracted from the trace file.
- The function *processPacket* will maintain some data structure that used to store all the packets information is a slice list. It will call *getPacket* to extract each packet information and store it in a packet structure with in a packet list which belong to the current slice.
- The main function then calls the *print_results* function to process and print slice specific stats.

## Functions

We shall list the important functions present in the source code present in the *main.c* and the *util.c* code files.

| **Function signature** | **Description** |
|---|---|
| **main.c** | |

| | |
|---|---|
| void processPacket(u_char *args, const struct pcap_pkthdr* pkthdr, const u_char* packet) | Pcap filters the packets using the given filter and then calls this function for each valid packet (we only handle TCP packets) a packet struct is created and it is processed and placed into the slice and connection data structs |
| void getCount(u_char *args, const struct pcap_pkthdr* pkthdr, const u_char* packet) | This function is registered as the callback handler if we only want to count the number of packets in the trace (turned on by the –C option). |
| **util.c** | |
| void print_results(struct sliceListHead *slh, int bdUnit, int pringMethod) | Outputs the information for each slice |
| void print_usage_exit(char *name) | Prints the appropriate usage of the tool if invoked incorrectly from the command line. |
| int pcap_dloff(pcap_t *pd) | Get the offset to Network Layer data according to different Data Link Layer type |
| Void getTime(char *tmpLine, struct timeval curTime) | Get printable time given a struct timeval as an argument |
| void getPacket(struct packet *pkt, const struct pcap_pkthdr* pkthdr, const u_char* packet, int sid, int pid, pcap_t *desc) | Get each packet information and store them into pkt |

## Appendix II

Valgrind is an award-winning open source instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile programs in detail.

Wireshark is the world's foremost network protocol analyzer, and is the de facto industrial standard.

## Appendix III

This appendix contains the references to some of the papers which describe ongoing research in network traffic analysis as well as commercial products for carrying out the same:

1. Tools
    a. CAIDA - http://www.caida.org/tools/utilities/flowscan/
    b. Packeteer - http://www.bluecoat.com/
    c. http://www.sandvine.com/
2. Papers
    a. http://www.pittsburgh.intel-research.net/~kpapagia/papers/BLINC.pdf
    b. http://cis.poly.edu/~ross/papers/P2PliveStreamingMeasurement.pdf
3. Alternatives to libpcap
    a. http://www.netronome.com/files/file/Netronome%20-%20Scaling%20Network%20Appliance%20Performance%20White%20Paper%20(3-09).pdf