Using Wireshark to Create Network-Usage Baselines

10 June 2007

Joshua L. Davis
Georgia Tech Research Institute
Georgia Institute of Technology
Atlanta, GA 30332

Email: joshua.davis@gtri.gatech.edu
Phone: 404.407.7554

Wireshark is the world's foremost network protocol analyzer, and is the standard in many industries. It is the continuation of a project that started in 1998. Hundreds of developers around the world have contributed to it, and it is still under active development.[1]

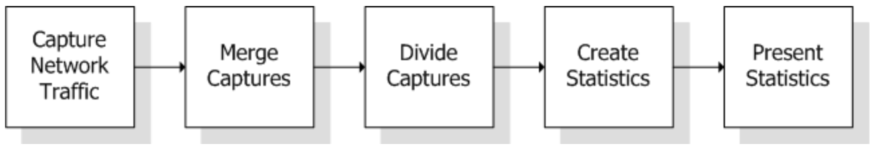
A *Network-Usage Baseline*, within this paper, is defined as a profile of the characteristics of a communication network within a particular window of usage. For example, these characteristics can include utilization, network applications, number of users, etc. These baselines can be used for forecasting and planning, as well as optimization and troubleshooting. This paper is used to document a methodology for creating network-usage baselines using Wireshark[2], its packaged toolsets (e.g. dumpcap.exe, tshark.exe), an advanced text editing tool and a common spreadsheet application[3].

Notwithstanding Wireshark's many capabilities, there are limitations when attempting to analyze extremely large capture files (GB+). The task of creating a baseline on modern, multi-user networks normally requires the manipulation and analysis of many very large, multi-GB data captures. This methodology was developed based on roughly 72 hours of captured data, at three distinct intersections in the network architecture. The resulting data equated to 52GBs of libpcap capture files. All of these files were captured using the tool `dumpcap.exe`, which accompanies the Wireshark installed. During the capturing, each packet was intentionally "snapped" at 50 bytes to minimize the size of the resulting capture files. To the author's current knowledge, this "snapping" at 50 bytes does not impact the statistics being generated. The example charts at the end of this paper support this understanding.

Methodology Overview

Diagram 1 depicts the high-level process for capturing the data and producing a baseline. Difficulty arises in the process from the sheer quantity of data that must be managed and limitations in Wireshark to handle very large libpcap files.[4]

Diagram 1: The Process



Current limitations within the Wireshark software prevent reasonable management and analysis of the multiple GB size files within the main application. This approach leverages multiple scripts to automate the processing of the capture files and creation of the baseline statistics. Depending on the size of the captured data, the processing at merge captures, dividing captures and creating statistics can take days or even weeks. However parallelization of the processing is possible on separate computers and could significantly decrease the time required.

Capture Network Traffic

When creating a baseline, it is common to distribute the capturing to multiple points within the network to provide the most comprehensive view into its performance and utilization. The goal, when possible, should be to tap into the network in such a way that you are granted a view into all of the traffic. If this is not possible, capturing at interface points between distributed communication points will likely be sufficient given that modern 1000Mbit and 100Mbit LANs are normally not the limiting factor in the network's architecture.

[1] Description from *Wireshark*'s web site (http://www.wireshark.org/about.html)

[2] Wireshark Version 0.99.5 - downloaded from http://www.wireshark.org

[3] *Textpad* was the advanced text editing tool and *Microsoft Excel 2003* was the spreadsheet application used during the development of this methodology (http://www.textpad.com)

[4] From this initial experiment, 150MB files were the largest that Wireshark could handle effectively on reasonably high-end desktops (2GB+ in RAM running Windows XP Professional).

To enable capturing of all of the traffic on a switch based network, it typically necessary to configure the primary switch or router to “mirror” all of the traffic to a particular port. Some more advanced devices already possess a monitor port which provides this “mirroring.” Unfortunately, in many cases, it will be entirely impossible to capture all traffic given that the one single port’s capacity is significantly less than all of the available bandwidth on a switched network. However, it will provide the best insight in spite of the incompleteness at high utilization loads. For this particular situation, it was necessary for the engineers to configure the individual LAN/WAN segments switches to permit traffic monitoring.

Once the monitoring devices are connected, Wireshark can be used for capturing the network traffic. To simplify the process, make it consistent and minimize use of computing resources, a Windows XP command line batch script was created that executed the command line capture tool to capture the target network traffic. The following lists the command line help for `dumpcap.exe`.

```
Dumpcap 0.99.5 (SVN Rev 20677)
Capture network packets and dump them into a libpcap file.
See http://www.wireshark.org for more information.

Usage: dumpcap [options] ...

Capture interface:
  -i <interface>           name or idx of interface (def: first none loopback)
  -f <capture filter>      packet filter in libpcap filter syntax
  -s <snaplen>             packet snapshot length (def: 65535)
  -p                       don't capture in promiscuous mode
  -B <buffer size>         size of kernel buffer (def: 1MB)
  -y <link type>           link layer type (def: first appropriate)
  -D                       print list of interfaces and exit
  -L                       print list of link-layer types of iface and exit

Stop conditions:
  -c <packet count>        stop after n packets (def: infinite)
  -a <autostop cond.> ...  duration:NUM - stop after NUM seconds
                           filesize:NUM - stop this file after NUM KB
                              files:NUM - stop after NUM files
Output (files):
  -w <filename>            name of file to save (def: tempfile)
  -b <ringbuffer opt.> ... duration:NUM - switch to next file after NUM secs
                           filesize:NUM - switch to next file after NUM KB
                              files:NUM - ringbuffer: replace after NUM files
Miscellaneous:
  -v                       print version information and exit
  -h                       display this help and exit

Example: dumpcap -i eth0 -a duration:60 -w output.pcap
"Capture network packets from interface eth0 until 60s passed into output.pcap"

Use Ctrl-C to stop capturing at any time.
```

It should be noted that determining the correct network interface it not entirely intuitive. Fortunately `dumpcap.exe` provides a handy option (`-D`) that provides a list of all known interfaces. Another useful option, especially when drive space is limited is the “snap” option (`-s 50`). “Snapping” limits the max size of the packet captured to the specified value. The default is 65,535 bytes. As stated previously, this operation does not affect the resulting statistics.

Another useful option to note is the max file capture size option (`-b filesize:50000`). During initial experiments with Wireshark, the engineers experienced that roughly around 150MB the files would throw a documented and well known application error[5]. In many instances when files close to 150MB were opened, it would take far too long to permit reasonable analysis. From this experience, 50MB was deemed a manageable size. This size can be quickly manipulated and viewed within the Wireshark application. The following is an example using `dumpcap.exe`.

[5] Error: Out of Memory! Sorry, but Wireshark has to terminate now! Some infos /workarounds can be found at http://wiki.wireshark.org/KnownBugs/OutOfMemory

```
dumpcap.exe -i \Device\NPF_{E69F0654-47F6-475B-BED4-7DC94F410207} -s 50 -w
CaptureLocation.pcap -b filesize:50000
```

Merge Captures

With a distributed set of captures and a divided set at each capture point, it is now necessary to merge, in one form or another, all of the sets to create combined statistics over evenly spaced time intervals and appropriate distributions. Depending on the intended use of the baseline, it might be necessary to merge all captures or just subsets based on capture point, times/events, etc. Wireshark provides a command line for merging captures. The following lists help for the command line tool `mergecap.exe`.

```
Mergecap 0.99.5 (SVN Rev 20677)
Merge two or more capture files into one.
See http://www.wireshark.org for more information.

Usage: mergecap [options] -w <outfile|-> <infile> ...

Output:
  -a                files should be concatenated, not merged
                    Default merges based on frame timestamps
  -s <snaplen>      truncate packets to <snaplen> bytes of data
  -w <outfile|->    set the output filename to <outfile> or '-' for stdout
  -F <capture type> set the output file type, default is libpcap
                    an empty "-F" option will list the file types
  -T <encap type>   set the output file encapsulation type,
                    default is the same as the first input file
                    an empty "-T" option will list the encapsulation types

Miscellaneous:
  -h                display this help and exit
  -v                verbose output
```

This command is not very complex, however the quantity of files to be merged might create difficulties. With large quantities, it might be helpful to write a script to handle the merge. For this experiment, a Windows XP command line batch script was written.

In regards to disk space, the resulting file will be enormous and efforts should be made to ensure adequate space is available. The following shows an example of merging three files into one using the command line tool `mergecap.exe`.

```
mergecap.exe -w MergedFile.pcap capture1.pcap capture2.pcap capture3.pcap
```

Divide Captures

With the captures merged, it is time to divide the whole capture into even segments representing the target time intervals. Depending on the desired granularity viewed, i.e. the “bin width”, and the quantity of data at any given moment during the capture, one might be limited to the size of the interval due to limitations in `tshark.exe`, the command line Wireshark application. The best rule of thumb found so far is to not have any interval exceed 125MBs. This number has not been tested extensively but was successful during the experiment using typical high-end workstations. At the create statistics step, processing an interval file larger than 125MBs might throw an error[6] similar to the previous error mention resulting in no statistics being produced for that interval.

Depending on the size of the file and the intervals desired, it is likely that a slightly more complex script should be used to support this process. Provided in Appendix A is the script used to separate the large libpcap files into one minute segments. Each one of these individual interval libpcap files will be used to generate one row of statistics for that interval of time.

[6] Error: Unhandled exception (group=1, code=6). This application has requested the Runtime to terminate it in an unusual way. Please contact the application's support team for more information.

Create Statistics

During this step in the process, each individual time segment will be processed via the command line application `tshark.exe` to produce statistics for this segment. The resulting output from this step in the process will be a comma separated value (CSV) file that may be opened by many common tools for manipulation and creation of visualization of data. It is very important to process these time intervals in the correct sequence to ensure accurate resulting statistics. Ignoring this can result in values that cannot be trusted and likely useless. This command will be used with filters that are appropriate for the given network-usage baseline being produced. The following lists help for the command line tool `tshark.exe`.

```
TShark 0.99.5 (SVN Rev 20677)
Dump and analyze network traffic.
See http://www.wireshark.org for more information.

Copyright 1998-2007 Gerald Combs <gerald@wireshark.org> and contributors.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Usage: tshark [options] ...

Capture interface:
  -i <interface>           name or idx of interface (def: first non-loopback)
  -f <capture filter>      packet filter in libpcap filter syntax
  -s <snaplen>             packet snapshot length (def: 65535)
  -p                       don't capture in promiscuous mode
  -B <buffer size>         size of kernel buffer (def: 1MB)
  -y <link type>           link layer type (def: first appropriate)
  -D                       print list of interfaces and exit
  -L                       print list of link-layer types of iface and exit

Capture stop conditions:
  -c <packet count>        stop after n packets (def: infinite)
  -a <autostop cond.> ...  duration:NUM - stop after NUM seconds
                           filesize:NUM - stop this file after NUM KB
                              files:NUM - stop after NUM files
Capture output:
  -b <ringbuffer opt.> ... duration:NUM - switch to next file after NUM secs
                           filesize:NUM - switch to next file after NUM KB
                              files:NUM - ringbuffer: replace after NUM files
Input file:
  -r <infile>              set the filename to read from (no pipes or stdin!)

Processing:
  -R <read filter>         packet filter in Wireshark display filter syntax
  -n                       disable all name resolutions (def: all enabled)
  -N <name resolve flags>  enable specific name resolution(s): "mntC"
  -d <layer_type>==<selector>,<decode_as_protocol> ...
                           "Decode As", see the man page for details
                           Example: tcp.port==8888,http
Output:
  -w <outfile|->           set the output filename (or '-' for stdout)
  -F <output file type>    set the output file type, default is libpcap
                           an empty "-F" option will list the file types
  -V                       add output of packet tree        (Packet Details)
  -S                       display packets even when writing to a file
  -x                       add output of hex and ASCII dump (Packet Bytes)
  -T pdml|ps|psml|text     output format of text output (def: text)
  -t ad|a|r|d|e            output format of time stamps (def: r: rel. to first)
  -l                       flush output after each packet
  -q                       be more quiet on stdout (e.g. when using statistics)
  -X <key>:<value>         eXtension options, see the man page for details
  -z <statistics>          various statistics, see the man page for details

Miscellaneous:
  -h                       display this help and exit
  -v                       display version info and exit
  -o <name>:<value> ...    override preference setting
```

The following is a simple example that creates statistics for a 10 minute window from a 10 min interval divided from a "whole" capture file.

```
tshark.exe -q -z "io,stat,600, ,tcp.port == 80" -r DividedFile_1.pcap
```

The first filter is an empty filter which means it will sum the frames and bytes for that interval for all traffic. The second filter is for any traffic on TCP port 80, typically web server traffic. This command will permit a seemingly infinite amount of statistics to be generated[7]. From this experiment, 31 unique filters were used to build the resulting baseline. Obviously a user will need to develop a working understanding of Wireshark's display filtering for this step to be successful. The output from the previous example results with the following:

```
===================================================================
IO Statistics
Interval: 600.000 secs
Column #0:
Column #1: tcp.port == 80
                |   Column #0    |   Column #1
Time            |frames|  bytes  |frames|  bytes
000.000-600.000  281214 128192886  51408  28494414
===================================================================
```

For each filter applied, a column of stats will be created. For each column, two sub-columns will also be produced. The first being the number of frames and the second being the number of bytes. Now, the strategy used to produce multiple records of data from multiple divided capture files is listed in the following process:

1. Create a Windows XP command line batch script, or something more advanced, that will produce interval statistics for each divided capture file.
2. Append the output from `tshark.exe` to the same file.
3. Use an advanced text editor (e.g. TextPad) to search and replace all unnecessary text leaving only the desired rows in order.
4. Replace all double spaces ' ' with a single space ' ' until only single spaces remain.
5. Replace all single spaces ' ' with commas ','.
6. Create an appropriate header for the columns, using similar search and replace strategies. This will certainly be necessary when multiple filters are used.
7. Save the newly edited file as a comma separated value (CSV) file.

The following sections provide an example of the process:

Script used for creating the statistics

```
tshark.exe -q -z "io,stat,600, ,tcp.port == 80" -r DividedFile_1.pcap > stats.txt
tshark.exe -q -z "io,stat,600, ,tcp.port == 80" -r DividedFile_2.pcap >> stats.txt
tshark.exe -q -z "io,stat,600, ,tcp.port == 80" -r DividedFile_3.pcap >> stats.txt
```

[7] There is a length limit that was experienced at around 4000 characters. However, refer to Wireshark's website for details on this limit. The engineers in this experiment responded to this error by splitting the statistic requests into multiple processing batch scripts to be merged later within Microsoft Excel.

Resulting output - stats.txt

```
===================================================================
IO Statistics
Interval: 600.000 secs
Column #0:
Column #1: tcp.port == 80
                |   Column #0    |   Column #1
Time            |frames|  bytes  |frames|  bytes
000.000-600.000  981214 768192886  51708  67494414
===================================================================

===================================================================
IO Statistics
Interval: 600.000 secs
Column #0:
Column #1: tcp.port == 80
                |   Column #0    |   Column #1
Time            |frames|  bytes  |frames|  bytes
000.000-600.000  281214 128192886  51408  28494414
===================================================================

===================================================================
IO Statistics
Interval: 600.000 secs
Column #0:
Column #1: tcp.port == 80
                |   Column #0    |   Column #1
Time            |frames|  bytes  |frames|  bytes
000.000-600.000  344214 128342886  431408  283454414
===================================================================
```

“Search and Replace” Unnecessary Text Example - stats.txt

```
                |   Column #0    |   Column #1
Time            |frames|  bytes  |frames|  bytes
000.000-600.000  981214 768192886  51708  67494414
===================================================================
                |   Column #0    |   Column #1
Time            |frames|  bytes  |frames|  bytes
000.000-600.000  281214 128192886  51408  28494414
===================================================================
                |   Column #0    |   Column #1
Time            |frames|  bytes  |frames|  bytes
000.000-600.000  344214 128342886  431408  283454414
===================================================================
```

“Search and Replace” Unnecessary Text Results - stats.txt

```
000.000-600.000  981214 768192886  51708  67494414
000.000-600.000  281214 128192886  51408  28494414
000.000-600.000  344214 128342886  431408  283454414
```

“Search and Replace” Spaces with Commas Example - stats.txt

```
000.000-600.000,981214,768192886,51708,67494414
000.000-600.000,281214,128192886,51408,28494414
000.000-600.000,344214,128342886,431408,283454414
```

Add Column Headers and Saved as “stats.csv”

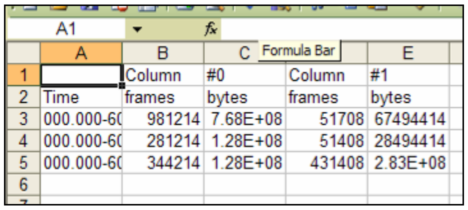
```
,Column,#0,Column,#1
Time,frames,bytes,frames,bytes
000.000-600.000,981214,768192886,51708,67494414
000.000-600.000,281214,128192886,51408,28494414
000.000-600.000,344214,128342886,431408,283454414
```

The resulting file will be one that anyone with experience using Microsoft Excel can produce useful results.

Present Statistics

This section is focused on providing a few tips for Microsoft Excel users in preparing this data for manipulation, graphing and charting. Additionally, if the resulting CSV file exceeds 65,000 rows, it will likely be necessary to manually import these lines into a single Microsoft Excel document using multiple worksheets or some other application/spreadsheet/database that does not have this limitation.

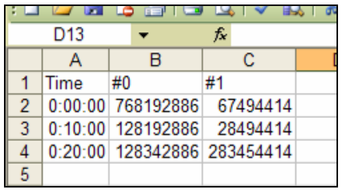
To begin using the output, open the CSV file in Microsoft Excel. The output should look something like the following:



A1 · Formula Bar

	A	B	C	D	E
1		Column	#0	Column	#1
2	Time	frames	bytes	frames	bytes
3	000.000-6(	981214	7.68E+08	51708	67494414
4	000.000-6(	281214	1.28E+08	51408	28494414
5	000.000-6(	344214	1.28E+08	431408	2.83E+08
6					

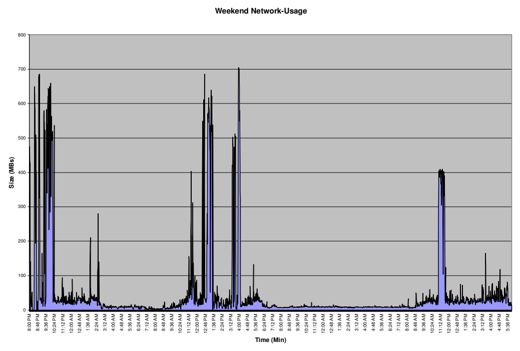
Immediately it is recommended that row two be deleted and column A (Time) be cleared and the appropriate interval range to be entered. This should be very easy in Microsoft Excel formatting the column as Time and using the `Time(H,M,S)` command. Additionally, it is suggested that all even columns (B, D, etc.) be deleted or hidden unless one plans to use frame count in analysis operations. Refer to the following as an example (even columns were deleted):

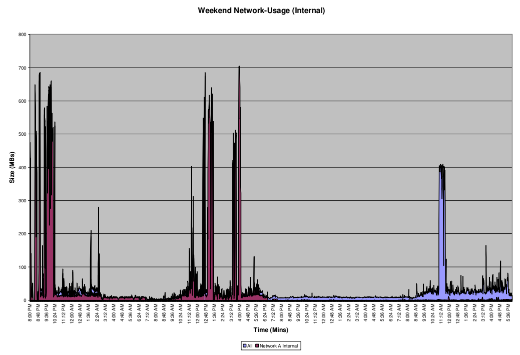


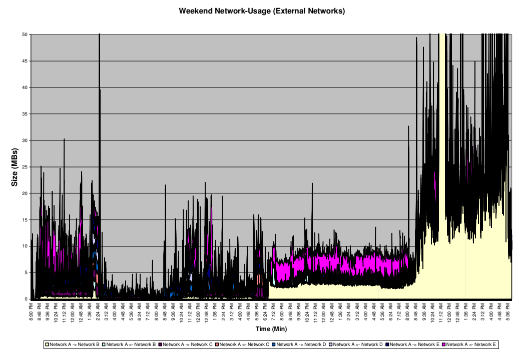
D13

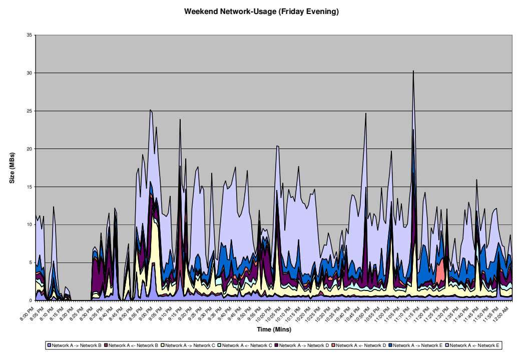
	A	B	C
1	Time	#0	#1
2	0:00:00	768192886	67494414
3	0:10:00	128192886	28494414
4	0:20:00	128342886	283454414
5			

The resulting data set will yield nicely to charting and many other data manipulation and presentation operations in Microsoft Excel. Lastly, it is recommended at this point that one saves the resulting file as a Microsoft Excel file to prevent any loss of data. Below are a few examples resulting from initial use of this process.









Conclusion

Despite limitations in Wireshark for handling large capture files, this paper has proven that there is a way to manipulate such data to create comprehensive network-usage baselines. Through the development of this methodology, the author hopes to begin some open source projects to help fill this void while also intending on improving Wireshark's capabilities. If you are interested in participating in development and use of these future toolsets please contact the author directly.

Hopefully this paper has been useful. Please share it with others and notify the author of any discrepancies, questions, suggestions and successes.

Appendix A – Script to Divide Large PCAP Files by Time

The following program was written for Python 2.5 and used default libraries. The Python interpreter can be downloaded from http://www.python.org. To use this script, copy and paste the following text into a file named *parse_pcap.py*. Change the filename "example.pcap" in this script to be the target file that is to be parsed.

```
import os
import datetime

print "===Parse PCAP Files ===\n"

startDate = datetime.datetime(2007, 04, 23, 16, 47, 56)
endDate = datetime.datetime(2007, 04, 23, 23, 43, 16)

startWindowDate = datetime.datetime(2007, 04, 23, 16, 47, 56)
endWindowDate = datetime.datetime(2007, 04, 23, 16, 48, 56)
dateTimeOffset = datetime.timedelta(minute=1)
fileCounter = 0

program = 'C:/Progra~1/Wireshark/editcap.exe'

while (endWindowDate < endDate):
        arg0 = [' ', "example.pcap " + str(fileCounter) + '.pcap' + ' -A \"' +
startWindowDate.isoformat(' ') + '\"' + ' -B \"' + endWindowDate.isoformat(' ') + '\"']
        print arg0
        os.spawnv(os.P_WAIT, program, arg0)

        fileCounter = fileCounter + 1
        startDate = startWindowDate + dateTimeOffset
        endDate = endWindowDate + dateTimeOffset
```