Monitoring of the Local Transmission Control Protocol's State Variables Using L3DGEWorld

Michael Allwright*, Grenville Armitage
Centre for Advanced Internet Architectures, Technical Report 100820C
Swinburne University of Technology
Melbourne, Australia
5704634@student.swin.edu.au, garmitage@swin.edu.au

***Abstract*—The monitoring of the Transmission Control Protocol (TCP) state variables can provide an understanding of how various traffic flows on a network interact. As discussed in previous work [1] the rate and diversity of the raw information available is incomprehensible in real time, considering human cognitive processing limitations. In the technical report we will demonstrate a solution to the problem of monitoring the attributes of TCP flows by projecting them as orthogonal visual characteristics on 3D entities displayed inside a modified game engine, known as L3DGEWorld.**

***Index Terms*—L3DGEWorld, TCP, Network Monitoring, SIFTR, Visualisation, Game Engine**

I. Introduction

The interactions between network traffic directly effects the end users experience of the network. This particularly holds true for the Internet, where latency and bandwidth limitations due to network loading and the underlying infrastructure, can adversely effect the quality of a Voice over Internet Protocol (VoIP) call, an online multiplayer game or a remote desktop connection. To solve this problem, many implementations based around the concept of Quality of Service (QoS) provide scheduling and prioritisation for specific traffic which is sensitive to certain network conditions. One challenge which arises from this solution is monitoring the performance of QoS and it's various implementations and configurations.

Whether or not QoS is enabled, the monitoring of network traffic is a non-trivial task. For example, the Transmission Control Protocol (TCP) implementation's state machine in FreeBSD has more than 20 different variables which describe the performance of a given outbound transfer. Some of these parameters are updated on a packet-to-packet basis, and when multiplexed with other outbound flows moving thousands of packets per second, as found on a file server, the shear volume of raw data obtainable is incomprehensible in real time. Solutions to similar problems regarding network security are outlined in [1]. This paper looks at various existing methods for administrating networks from a security perspective, and proposes the use of a modified 3D game engine to make presence of hackers attacking a network visually available in real-time.

In this technical report, the concept of using game engines to represent large amounts of data has been utilised to monitor TCP traffic flows as a proof of concept that aims to illustrate a more detailed understanding of the interactions of various network traffic on a macroscopic scale and furthermore, the impact of various QoS implementations and configurations.

This technical report is organised as follows: Section II provides background information about TCP, and an introduction to SIFTR [2], a FreeBSD kernel module which logs information about inbound and outbound TCP packets. Section III discusses the challenges of data visualisation and provides an introduction to L3DGEWorld [3]. Section IV details the structure of the application LTCPMON which provides a proof of concept by creating an interface between the raw TCP data generated by SIFTR and forwarding it into L3DGEWorld. Section V details future work and potential improvements, while Section VI concludes this technical report.

II. Transport Layer Protocols and SIFTR

A. The Transmission Control Protocol

Almost all communication over the Internet is provided by two transport layer protocols. These are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP), which provide different features

*Author is currently an engineering student at Swinburne University of Technology. This report was written during the author's winter internship at CAIA in 2010

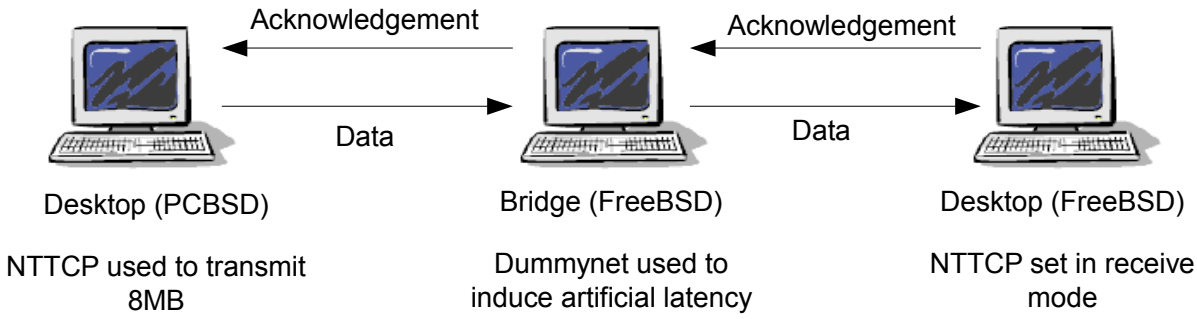


Fig. 1. Test bed to induce artificial latency

and are utilised by applications accordingly. UDP provides applications with connectionless communication and multiplexing. This means that while UDP can not guarantee the reliable transfer of data, it is lightweight and well suited to applications like online gaming and Voice over IP (VoIP). Meanwhile TCP provides mechanisms that guarantee reliable transfer of data and congestion avoidance. The reliable transfer of data makes TCP ideal for browsing the web, sending and receiving email and file-sharing. One potential drawback of TCP is that the connection throughput will fluctuate over time, this is because TCP probes the network's capacity in order to maximise throughput.

B. Macroscopic Impact of Multiple Traffic Flows

On a typical corporate or home network, there is bound to be both TCP and UDP flows present. And on these networks there is often a finite amount of bandwidth and routing resources which must be shared by both of these transport layer protocols. It is the interaction of network traffic from both these protocols that adversely affects performance at the application level. This holds true not only for the interaction of TCP and UDP flows, but also for the effect of a TCP flow on other TCP flows.

One characteristic of TCP which is often of interest is the Congestion Window (CWND). The CWND defines the amount of in-flight data that is allowed to be sent before an acknowledgement is received. TCP will grow the CWND every time an acknowledgement is received. If an acknowledgement for a piece of data is not received within a given time, TCP will enter a recovery mode before setting CWND to the slow start threshold value (typically half of CWND). This behaviour of probing the network to maximise throughput and backing off when packet loss is detected is TCP's congestion avoidance mechanism.

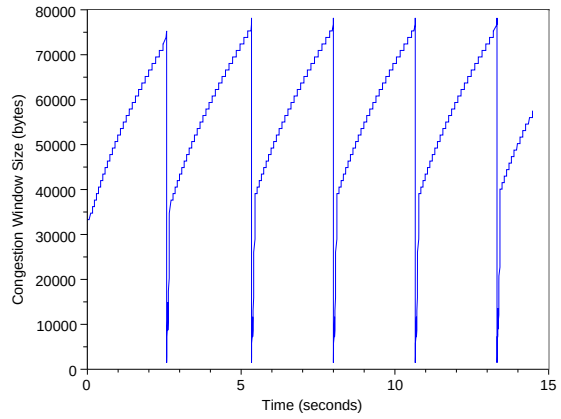
TCP is designed to operate on networks where the loading can vary with time. This can be observed by monitoring the throughput during a connection, where depending on network loading and infrastructure, the performance will decrease with the introduction of additional latency and bandwidth constraints. To demonstrate this, the test bench in Figure 1 was used to artificially induce latency via the use of Dummynet [4], a FreeBSD kernel module which emulates static network conditions though the use of virtual pipes.

Figure 2 shows the effect of increasing latency on CWND. As shown in the Figures, CWND's rate of growth is proportional to the latency of the underlying network (or in this case, the Dummynet pipe). This proportionality comes about as TCP is typically designed to only grow CWND for every acknowledgement received. Contrasting the duration to transfer a fixed amount of data in Figures 2(a) and 2(c), it is seen that excessive latency on a link can result in a heavy performance penalty.

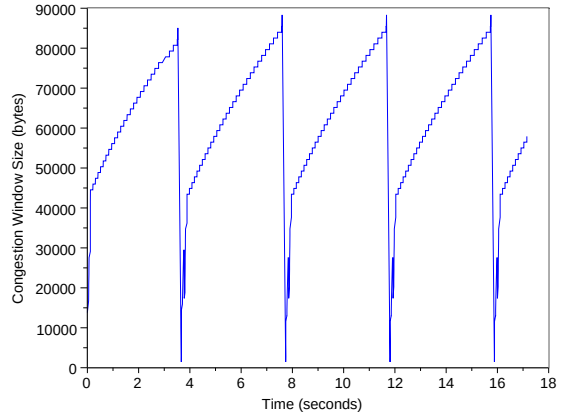
C. Measuring TCP Performance with SIFTR

SIFTR, Statistical Information for TCP Research [2], is a FreeBSD kernel module developed and maintained by CAIA [5]. The kernel module works by connecting to the FreeBSD packet filter interface [6], which allows SIFTR to take a snapshot of the TCP state machine on every inbound and outbound packet traversing through the TCP/IP stack.

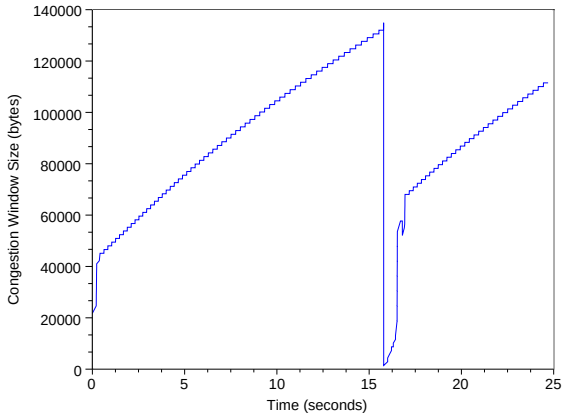
Once SIFTR is compiled, two kernel modules, *alq.ko* and *siftr.ko*, must be loaded into the kernel in order to use SIFTR. However the loading of *siftr.ko* will automatically result in the loading of *alq.ko*. In FreeBSD these modules can be installed into the /boot/modules directory and loaded dynamically using the command:



(a) No added latency



(b) Symmetric Latency of 20ms added



(c) Symmetric Latency of 100ms added

Fig. 2. Using Dummynet to induce artificial latency

```
root@mallwright# sysctl net.inet.siftr
net.inet.siftr.enabled: 0
net.inet.siftr.logfile: /var/log/siftr.log
net.inet.siftr.ppl: 1
net.inet.siftr.genhashes: 0
```

Fig. 3. SIFTR's sysctl configuration interface with default value set

```
kldload module_name.ko
```

Once the kernel module for SIFTR is loaded, the module can be enabled and configured dynamically by the sysctl interface [7]. As shown in Figure 3, the interface allows the configuration of the path to the log file (*net.inet.siftr.logfile*), the option to define how many packets must be detected before the log file is appended to (*net.inet.siftr.ppl*), and whether or not to generate the hashes for the packets which caused the snapshot (*net.inet.siftr.genhashes*). These values can be set with appropriate privileges using the command:

```
sysctl name=value
```

Once SIFTR is enabled (following setting the *net.inet.siftr.enabled* parameter to 1) - an ASCII formatted CSV log file will be generated and appended to with details about the state variables for every inbound and outbound packet (or as set by the *net.inet.siftr.ppl* variable). The CSV log file contains 25 comma separated values. Amongst the state variables, these also include the time-stamp and information about the packet which triggered the snapshot. Detailed information about these values can be found in the SIFTR manpage included in the tarball on the CAIA website [2].

III. Utilisation of Game Engines to Visualise Network Performance

The use of modified game engines to present dynamic data has been an active area of research. This section will give a short review of the previous work [1], [8] researched at CAIA.

Figure 4 shows the output of several common network analysis tools which are installed or readily available under the FreeBSD operating system. Although these tools provide different perspectives of network conditions, one thing that these tools share in common is ASCII formatted output (with the exception of tcpdump which can also output binary data in pcap format).

Depending on the formatting present on the standard output, this data can be more or less difficult to read. A further reality to this is that in a typical scenario, there will be multiple traffic flows multiplexed together, and

SIFTR - Statistical Information For TCP Research

```
root@host# head siftr.log
enable_time_secs=1278293075 enable_time_usecs=869497 siftrver=1.2.2 hz=1000
tcp_rtt_scale=32 sysname=FreeBSD sysver=800107 ipmode=4
o,0x9bc4e9ae,1278293075.871738,192.168.0.3,22,192.168.0.1,13845,1073725440,3801,6144...
o,0x1ff83e09,1278293075.871893,192.168.0.3,22,192.168.0.1,11927,1073725440,3577...
i,0x385b7fcb,1278293075.936369,136.186.229.102,5037,136.186.229.98,20928,1073725440...
```

tcpdump

```
root@host# tcpdump -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on em0, link-type EN10MB (Ethernet), capture size 96 bytes
17:32:23.137718 IP 136.186.229.98.22 > 136.186.229.103.49596: Flags [P.], ack...
17:32:23.137731 IP 136.186.229.98.22 > 136.186.229.103.49596: Flags [P.], ack 1...
17:32:23.138090 IP 136.186.229.103.49596 > 136.186.229.98.22: Flags [.], ack 116...
```

SPP - Synthetic Packet Pair

```
root@host# head spp.log
 1 1278292405.310012 0.007890 0.000024
 2 1278292405.322705 0.006496 0.000051
 3 1278292405.330009 0.007897 0.000023
```

Ping

```
root@host# ping -i0.1 136.186.20.9
PING 136.186.20.9 (136.186.20.9): 56 data bytes
64 bytes from 136.186.20.9: icmp_seq=0 ttl=63 time=0.293 ms
64 bytes from 136.186.20.9: icmp_seq=1 ttl=63 time=0.192 ms
64 bytes from 136.186.20.9: icmp_seq=2 ttl=63 time=0.192 ms
```

Fig. 4. ACSII output of various network monitoring tools

all printed to the same standard output. This leaves the task of separating flows to the network administrator, and considering the volume of traffic that is flowing in and out of a home network, let alone a corporate one, the task of visually identifying a single parameter from a single flow in real time, is nearly impossible.

A. Conventional Graphing Techniques

As discussed previously, it is an unrealistic expectation that network administrators will be able to make productive decisions based purely on a blur of high speed ASCII log messages appearing at their terminal. To solve this problem, software packages like Scilab [9] and R [10] have been used to graph this flow of incomprehensible data into a visual format. However conventional graphing techniques have some limitations. The TCP state machine contains more than 20 different variables, most of which are changing on a packet-to-packet basis. Using conventional graphing techniques, we can plot three of these variables in an area where the distances between the orthogonal components X, Y and Z and the origin, could each represent a single variable or metric. In this space, a point for each TCP flow could be plotted, and furthermore this plot could then be animated by allowing the points to move with time, creating a real-time representation of the network traffic. However, given the number of packets and the number of flows to which these packets belong, the representation would be familiar to that of an blurred electron cloud, which is only a marginal improvement over the ASCII text blur. There is however another solution. If the points were replaced with objects, where instead of spatial positioning, orthogonal visual characteristics were used to represent the underlying metrics of interest, it is possible that this visualisation would be far more intuitive in real-time. To do this, a new software package would have to be developed to provide a 3D area in which these objects or entities could exist, or alternatively an existing genre of software platforms may already provide most of this required functionality.

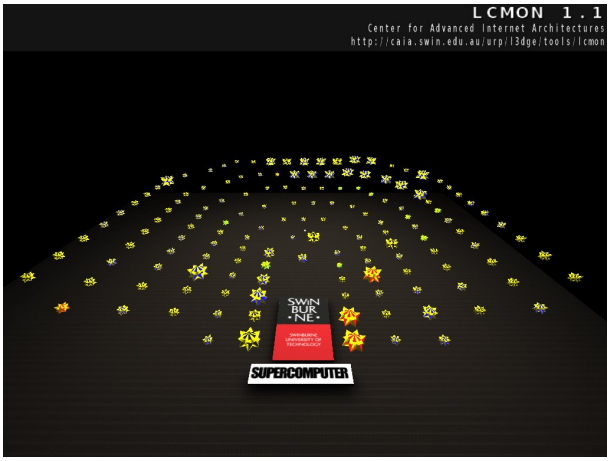


Fig. 5. LCMON - L3DGEWorld Clusternode Monitor

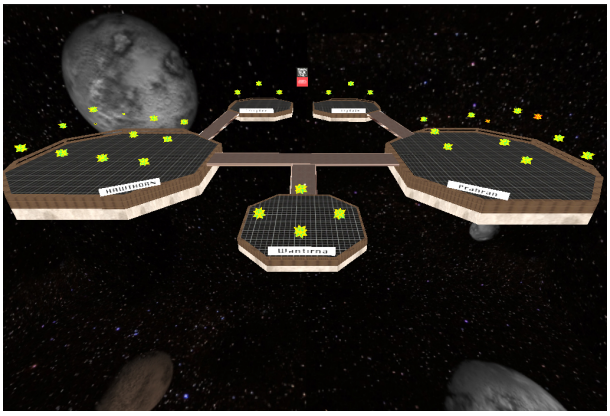


Fig. 6. LupsMON - L3DGEWorld Uninterruptible Power Supply Monitor

B. 3D Game Engines for Visualisation

The adoption of game engines for visualisation is a concept which been gaining momentum over the past years. 3D game engines support an attractive set of *off the shelf* features such as:

- Inbuilt multi-user interaction including chat
- Simple physics
- Support for entities with many orthogonal visual characteristics (avoids ambiguous interpretation)
- Native server-client infrastructure for viewing at remote locations

One example of this is is L3DGEWorld [3], which has been developed by CAIA.

C. L3DGEWorld

L3DGEWorld is a data visualisation utility based on Open Arena, a GPL'd game that makes use of the Quake III Arena (Q3A) game engine. The software contains various modifications which allow entities to be controlled via a UDP based API. The entities in the virtual world can be set to bounce, spin, change colour and even change positions with time. L3DGEWorld has been previously used in the monitoring of the Swinburne Supercomputer 'The Green Machine' [11], and the Swinburne Uninterruptible Power Supply (UPS) Network [12]. As shown in Figures 5 and 6, various underlying parameters from both systems have been mapped to visual metrics inside the L3DGEWorld application.

L3DGEWorld allows network administrators to import data from multiple sources and map particular values inside this data to orthogonal visual characteristics on the displayed entities in the virtual world. L3DGEWorld comes with L3DGEComms [13], a C library which can be used to provide a simple interface to L3DGEWorld. The library handles trivial tasks like establishing connections and authentication, sending messages and finalising connections. L3DGEWorld also provides output mechanisms to gain control over (or to modify) the underlying network, this functionality has not been utilised in LTCPMON.

IV. LTCPMON

LTCPMON (L3DGEWorld Transmission Control Protocol Monitoring) demonstrates the use of combining L3DGEWorld with SIFTR to provide a qualitative insight into how multiple outbound TCP flows interact on the FreeBSD TCP/IP stack. LTCPMON is a C++ console application which has been built following object orientated principles to promote software reuse by ensuring that high cohesion and low coupling exists between all objects. This means that many parts of the software can be easily reused in other applications working with SIFTR or L3DGEWorld.

A. Input Files

When LTCPMON starts, it parses the command line, making the assumption that the last three given arguments are the input files to the program. These files must be provided and must be in order, following the syntax message shown in Figure 7. Each of these files has a parser class which is used to import the ASCII formatted data into the Standard Template Library (STL) container classes.

As shown in Figure 8, the first file defines mappings between state variables and orthogonal visual characteristics via a given statistics function. This means that for every update LTCPMON sends to L3DGEWorld, the selected statistic function will calculate its value from

```
LTCPMON - SIFTR to L3DGEWorld Interface Utility
usage: ltcpmon [options] <map_file> <entity_file> <siftr_logfile>

options:
  [-s <time>] - visualisation start time offset since beginning of the SIFTR log
  [-d <duration>] - duration of visualisation
  [-u <period>] - period at which updates are sent to L3DGEWorld
  [-r <ratio>] - real:visualisation time ratio
  [-a <ip address>] - IP address of the L3DGEWorld Server
  [-p <port>] - Communication port of the L3DGEWorld Server
  [-v] - Enable verbose mode

files:
  <map_file> - contains mappings of TCP parameters to visualisation metrics
  <entity_file> - contains a list of entities available in the server map
  <siftr_logfile> - contains the SIFTR data to be visualised
```

Fig. 7. Command Line Syntax Message for LTCPMON

the dataset containing the selected state variable's values since the last update. The output value to statistic function is then dynamically scaled and sent to L3DGEWorld via L3DGEComms.

The second file defines what entities are available on the map. This file is typically known as *l3dgehosts.conf* or *<mapname>.conf* and must be the same file in use on the server. The file contains three space separated strings which respectively define the entity's name, it's unique identification number and it's administrative weight. The entity name can be set to be any string (without spaces), while the unique identification number should represent the entities on the map (this is done by using the same file for server, as is loaded into LTCPMON). The administrative weight is for a unused feature in L3DGEWorld, where the player's interactions would influence the underlying network configuration. The administrative weight is not used in LTCPMON and discarded during parsing.

The final input file, as previously mentioned, is the CSV formatted SIFTR log file. As shown previously (Figure 4), this file is the output of the SIFTR kernel module. It contains 25 fields which provide the values of the TCP state variables, alongside information about the packet which triggered the log message. For details of the fields, see the README file for SIFTR [2].

B. Program Flow

As mentioned in the previous section, when LTCPMON starts it parses the command line to find the required files and any additional command line switches. These command line switches are shown in Figure 7 and are optionally used to configure the location and port of L3DGEWorld server (by default 127.0.0.1 on 27960), the playback speed, the start time offset and the duration of playback as desired.

The application sets up it's environment using the optional command line switches before parsing the provided files. This gives LTCPMON a complete configuration. LTCPMON then creates a connection with L3DGEWorld Server and enters it's main loop. Inside the main loop the program parses a variable number lines of the SIFTR log file. The number of lines parsed is dependant on how much traffic was flowing in and out of the TCP/IP stack over a given interval. This is because LTCPMON looks at the timestamps inside the log file to determine when a finite amount of time has expired. Once this time has expired the application waits for an update timer to expire, before calculating the statistics on the data. Hence, the playback ratio switch controls the speed at which the log file data is parsed and displayed, while the update period switch controls the resolution of the updates.

LTCPMON will terminate once it has reached the end of the log file, or in case of a pipe, when it has consumed it's buffer. This flow is illustrated in Figure 9.

C. Design Challenges

One of the most significant challenges faced during the development of LTCPMON was designing the application in away that it could run in real time, i.e. run concurrently while data was being obtained. This presents an interesting challenge of scaling the attributes from the output statistics, due to the infrastructure and

```
# Example mappings file
# format follows variable:statistic_mode:visualisation_parameter

# Map the median congestion window size to scale
map currentCongestionWindow:median:scale

# Map the standard devation of inflight data to spin rate
map inflightBytes:stddev:spinRate

# Map the median rttEstimate to the colour of the object
map rttEstimate:median:colour
```

Fig. 8. Contents of mapping file for LTCPMON

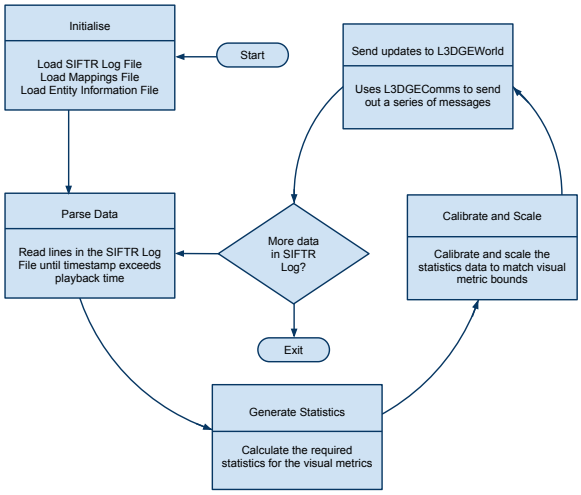


Fig. 9. Overview of LTCPMON's program flow

capacity of the underlying network being indeterminate at run time. Given this, it is not sufficient to statically scale the TCP state variables into the orthogonal visual characteristics displayed by L3DGEWorld, as static scaling would require making assumptions about the underlying network infrastructure and capacity.

To overcome this challenge, a dynamic scaling module was implemented. The dynamic scaling module in LTCPMON scans through the current dataset and detects maximum and minimum values of a statistic of a given TCP state machine parameter. The scaling starts by setting the maximum and minimum values to a uninitialised state, and then calibrating these values as new data is obtained. However this method does have the following drawbacks.

- Dynamic scaling causes the quantitative aspect of a particular visual metric, such as size or spin rate, to have a varying meaning over time, this can be ambiguous with actual changes in network conditions
- The current configuration of the dynamic scaling mode, is to take the maximum and minimum values outputted by the statistics module. This is very sensitive to noise.

V. Future Work

As of this Technical Report, LTCPMON is in it's first alpha release. While the application is functional, significant testing is required to determine if various components are working correctly. There are a few known bugs in the application as follows:

1) Update time resolution is limited to an integral amount of seconds, this is caused as the time() function in *time.h* returns an integer representing the number of seconds since the 1st of January, 1970 (Unix Epoch).
2) Dynamic scaling is sensitive to noise. An improvement to this will be mapping the lower fifth and upper ninety fifth percentiles of the statistics module output to the maximum and minimum values of the visualisation metric rates, truncating the outliers beyond this range.
3) Reset logic of entities is currently ineffective. This functionality is crucial and must be debugged before the beta release.

The tarball available on the CAIA/LTCPMON website, contains the latest snapshot of the source code. Throughout the source code, comments have been added to highlight potential weaknesses. These are planned to be fixed before the beta release.

VI. Conclusion

In this Technical Report we have seen that the productive monitoring of network traffic is a non-trivial task. One potential solution to this problem is to use game

engines to move away from conventional graphing techniques, and start representing various network metrics as visually orthogonal characteristics projected on to 3D objects inside a virtual world. This technical report also introduces LTCPMON, an example of how a modified game engine like L3DGEWorld can be controlled by a third party application. LTCPMON is responsible for relaying data onto the entities inside the virtual world, projecting a configurable set of TCP state variables on to visually orthogonal characteristics.

REFERENCES

[1] W. Harrop and G. Armitage, "Real-Time Collaborative Network Monitoring and Control Using 3D Game Engines for Representation and Interaction," in *VizSEC'06 Workshop on Visualization for Computer Security*, Virginia, USA, 03 November 2006, pp. 31–40. [Online]. Available: http://doi.acm.org/10.1145/1179576.1179583

[2] "NewTCP project tools," Aug. 2010, accessed 3 August 2010. [Online]. Available: http://caia.swin.edu.au/urp/newtcp/tools.html

[3] "L3DGEWorld," Aug. 2010, accessed 6 August 2010. [Online]. Available: http://caia.swin.edu.au/urp/l3dge/tools/l3dgeworld/

[4] M. Carbone and L. Rizzo, "Dummynet revisited," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 2, pp. 12–20, 2010.

[5] "CAIA - Centre for Advanced Internet Architectures," Aug. 2010, accessed 15 August 2010. [Online]. Available: http://caia.swin.edu.au/

[6] *PFIL FreeBSD Kernel Developer's Manual*, Aug. 2010, accessed 5 August 2010. [Online]. Available: www.freebsd.org/cgi/man.cgi?query=pfil

[7] *sysctl - FreeBSD Kernel Developer's Manual*, Aug. 2010, accessed 6 August 2010. [Online]. Available: www.freebsd.org/cgi/man.cgi?query=sysctl

[8] W. Harrop and G. Armitage, "Intuitive Real-Time Network Monitoring Using Visually Orthogonal 3D Metaphors," in *Australian Telecommunications Networks & Applications Conference 2004 (ATNAC 2004)*, Sydney, Australia, 8-10 December 2004, pp. 276–282. [Online]. Available: http://caia.swin.edu.au/pubs/ATNAC04/harrop-armitage-ATNAC2004.pdf

[9] "Scilab - The Free Platform for Numerical Computation," Aug. 2010, accessed 8 August 2010. [Online]. Available: http://http://www.scilab.org/

[10] "The R Project for Statistical Computing," Aug. 2010, accessed 8 August 2010. [Online]. Available: http://www.r-project.org/

[11] "LCMON - L3DGEWorld Cluster-node Monitoring," Aug. 2010, accessed 6 August 2010. [Online]. Available: http://caia.swin.edu.au/urp/l3dge/tools/lcmon/index.html

[12] "LupsMon - L3DGEWorld Uninterruptible Power Supply Monitoring," Aug. 2010, accessed 6 August 2010. [Online]. Available: http://caia.swin.edu.au/urp/l3dge/tools/lupsmon/index.html

[13] L. Parry, "L3DGEWorld 2.3 Input & Output Specifications," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 080222C, 22 February 2008. [Online]. Available: http://caia.swin.edu.au/reports/080222C/CAIA-TR-080222C.pdf