Real Time Traffic Classification and Prioritisation on a Home Router using DIFFUSE

Nigel Williams, Sebastian Zander
Centre for Advanced Internet Architectures, Technical Report 120412A
Swinburne University of Technology
Melbourne, Australia
njwilliams@swin.edu.au, szander@swin.edu.au

***Abstract*—Quality of Service (QoS) is important in multi-user residential networks that share a common bandwidth-constrained Internet connection. A limited upstream bandwidth can become a bottleneck and the performance of delay-sensitive applications – such as VoIP and online games – can degrade significantly when latency is introduced by other traffic flows. Current residential router QoS support must be manually configured and set up is difficult for the average home user. The DIFFUSE (DIstributed Firewall and Flow-shaper Using Statistical Evidence) architecture has been developed to provide automated IP traffic classification and treatment based on statistical flow properties. This paper provides some preliminary performance results of DIFFUSE on a TP-Link TL-WR1043ND home Internet router running OpenWRT Linux. We find that DIFFUSE can be successfully built-for and run on the TL-WR1043ND. The router is able to perform flow classification for at least 5,000 concurrent flows. With prioritisation enabled an upstream throughput of 24.5 Mbps is achieved. The router is able to identify and prioritise flows in a realistic home-network scenario.**

I. Introduction

Quality of Service (QoS) is important in multi-user residential networks that contain mixed traffic types and share a common bandwidth-constrained Internet connection (e.g. ADSL1 or ADSL2+ services with uplink speeds limited to 256 kbps–1.5 Mbps). A limited upstream bandwidth can become a bottleneck when multiple upstream flows are active, causing additional latency. The performance of delay-sensitive applications – such as VoIP and online games – can degrade significantly as latency is introduced [1]. The identification and separation of delay-sensitive and delay-tolerant traffic at the upstream link is highly desirable in order to preserve the performance of these delay-sensitive applications.

QoS support is available in current home routers, but configuration must be performed manually. This requires some networking knowledge, such as which TCP or UDP ports represent application traffic that requires prioritisation. Thus the average home-user may be unable to configure QoS support.

The DIFFUSE (DIstributed Firewall and Flow-shaper Using Statistical Evidence) architecture has been developed to provide automated IP traffic classification and treatment based on statistical flow properties and Machine Learning (ML). The use of statistical evidence for classification distinguishes DIFFUSE from existing classification schemes, which commonly employ packet-payload inspection or port number matching. DIFFUSE can provide transparent QoS for residential broadband users without requiring the end-user to know specific details such as application port numbers. Using a pre-built classification model, the system is able to identify priority flows and reconfigure Internet access hardware without user intervention.

DIFFUSE features the de-coupling of classification and flow prioritisation. For example a flow might be classified within the network core on a high-performance Classifier Node (CN) classifying tens of thousands of concurrent flows, providing prioritisation updates to multiple low-performance Action Nodes (AN) at the network edge. The ANs act on the updates provided by the CN. The CN and AN could also be on the same device – a bridge or router may classify forwarded traffic and reconfigure prioritisation rules (CN+AN). The architecture is described in [2].

DIFFUSE has thus far been tested only on PC workstations running FreeBSD and Linux. This paper provides some preliminary performance results of DIFFUSE on a TP-Link TL-WR1043ND home Internet router running OpenWRT[1] Linux. We examine the throughput, CPU load and memory consumption of the TL-WR1043ND under different network conditions with and

[1] A Linux distribution for embedded devices: www.openwrt.org

without DIFFUSE. We aim to determine whether it is practical to run DIFFUSE on the router.

We find that DIFFUSE can be successfully built-for and run on the TL-WR1043ND router. With DIFFUSE enabled the router is able to identify and prioritise flows in a realistic scenario. When running as a CN+AN the router is able to sustain in excess of 5,000 concurrent flows. Throughput measured when configured as a CN+AN was 24.5 Mbps with $\tilde{8}0\%$ CPU utilisation. As an AN the router is able to sustain more than 16,000 concurrent flows. A throughput of 32.8 Mbps was achieved when tracking 20 priority flows with a maximum of 80% CPU utilisation. We identify several optimisations for improving the CPU and memory performance of DIFFUSE.

The report is organised as follows. We first provide background details on the DIFFUSE architecture, OpenWRT and the TP-Link TL-WR1043ND in Section II. In Section III we describe the equipment and experimental method. The results are presented and discussed in Section IV. Section V outlines future work and Section VI concludes.

II. Background

A. DIFFUSE

DIFFUSE makes IP flow classification decisions based on statistical flow properties (*features* in ML terminology) and a classification model that has been trained on example traffic to recognise these features [2]. By calculating features of an IP traffic flow (such as packet length distributions or packet inter-arrival times) and comparing these with a classification model, DIFFUSE is able to provide a prediction as to the nature of the traffic flow (such as whether it is delay-sensitive or delay-tolerant[2]).

This approach is distinct from current popular forms of IP flow classification, which include transport-layer header inspection (e.g. TCP/UDP port numbers) and packet payload inspection. Port number identification can be ineffective in the presence of NATs or other middleboxes that can alter packet headers, while port-agile applications may use different ports for different flows. The use of packet payload inspection can be limited by performance or legal issues. A shared difficulty with both techniques is the requirement of application-specific knowledge (such as port number, or payload signatures) in order to identify traffic.

A key feature of DIFFUSE is that it is able to produce a generalised classification model for traffic *types*, rather than for single applications only (though this is also possible). The process of training a classification model is automated. As delay-sensitive applications often exhibit similar traffic characteristics [1], a classification model can be built that will identify multiple applications of the same type (different online games of the same genre), potentially including applications that were not known at the time of model creation.

The classification model used in the experiments divides traffic into two categories: *First Person Shooter (FPS)* and *Other*. The FPS class was trained using data representative of a number of FPS games, while the Other class contains a mix of Web, file-sharing, email and other delay-tolerant traffic. The model uses two features to make decisions: *packet length* and *packet count*.

Feature calculations are performed on *windows* of n packets. There are two window configuration options for this, *sliding* and *jumping*. A sliding window will calculate features as each new packet arrives. A jumping window will wait for an entire new window of n packets to arrive before performing feature calculation (features are calculated once for every n packets).

We use the DIFFUSE 0.4 distribution for Linux [3]. This version of DIFFUSE is integrated with IPFW and uses Dummynet for prioritisation.[3] It is planned that future releases of DIFFUSE will allow the use of other packet filters/firewalls for prioritisation.

B. OpenWRT

OpenWRT is a Linux distribution for embedded devices. It is commonly used to replace the factory firmware found on residential Internet routers. By replacing the factory firmware, users gain access to additional configuration options and are able to extend the functionality of the router.

There are a variety of hardware configurations amongst residential routers and a customised firmware image must be created for specific devices. Pre-built firmware images are available and there is currently support for a wide range of hardware from a number of manufacturers [5].

OpenWRT uses a package management system called *opkg* to install binary packages [6]. OpenWRT Buildroot [7] is also available to fetch and cross-compile software

[2]Delay sensitive traffic includes Online Games and Telephony. Delay-tolerant traffic includes Web Browsing, email, FTP.

[3]IPFW (ipfirewall) [4] is the FreeBSD IP packet filter, which has been ported to Linux. Dummynet provides queuing, bandwidth shaping to IPFW.

for different platforms. The SDK framework [8] allows for the creation of new binary packages and firmware images for different target devices. Packages can be pre-compiled into the customised firmware images.

The OpenWRT SDK was used to build DIFFUSE binary packages and the DIFFUSE-OpenWRT firmware. The firmware image is based on the trunk (December 2011, 'Attitude Adjustment'[4]) version of OpenWRT. We also used the SDK to build profiling tools used in our experiments. These files and instructions for building and installing DIFFUSE packages and firmware can be found on the DIFFUSE website [3].

C. TPLink TL-WR1043ND

The TL-WR1043ND (see Figure 1) is a Gigabit Ethernet residential router with one WAN port, four LAN ports and support for 802.11b/g/n wireless. It is based on the Qualcomm Atheros AR9132 SoC running at 400Mhz. There is 8MB of flash storage for firmware and 32MB of system RAM. The OpenWRT page for this router can be found at [9]. The hardware specifications of the TL-WR1043ND are typical of the mid-level routers supported by OpenWRT.

A serial port header was added to the router in order to gain access to the default firmware and as an alternate method of installing new firmware. A serial port is not required to install OpenWRT, as new firmware can be loaded from the routers web interface. It can however be useful for development or experimental work (for instance router images can be loaded directly over serial via tftp).

For information on how to add a serial port to the router and gain root access to the default firmware, see Section VIII-A.

III. Equipment and Test Methodology

Several test-bed configurations were used to test the throughput, CPU and memory performance of the router. A common set of hardware was used across these testing scenarios, listed in Table I. The hosts used as traffic generators/sinks were both standard desktop workstations. We use the Ninjabox 500[5] for throughput testing and generating traffic to place the router under load. It is a commercial-grade high performance network capture and playback device that is able to generate flows at Gigabit speeds with precise inter-frame timing. The network topology was reconfigured for each of the tests, as described in the following sections.

Fig. 1. TP-Link TL-WR1043ND

A. Real-time Classification on 1 Mbps and 15 Mbps Links

To determine whether the router was able to run as a CN+AN and perform traffic prioritisation, we devised a test simulating a simple 'home network' scenario. In this scenario one user is playing an online game while another user is performing a bulk TCP upload (e.g P2P file-sharing). The online game traffic represents a real-time interactive application, while the TCP transfer represents a non-interactive flow. TCP is commonly used for delay-tolerant applications such as P2P file sharing (e.g. BitTorrent [11]). We measured the one-way-delay (OWD) and packet loss of the game traffic flow (client to server) with and without DIFFUSE to determine the effect of the upload. We also measured the overall packet loss. The topology is shown in Figure 2.

To simulate an ADSL home broadband connection, an upstream 1 Mbps bandwidth limit was placed on the WAN interface using Dummynet. A two-class weighted priority queuing system was configured in IPFW using the commands in Appendix VIII-C. With this configuration any packet labelled as 'priority' was placed in the high priority queue, with all other packets being placed in the low priority queue.

With the construction of the National Broadband Network (NBN) in Australia, some Internet providers are now able to offer asymmetric connections of up to 100/40 Mbps [12]. Thus we repeated the test with a 15 Mbps upstream link, representing a mid-level NBN access speed.

[4] Attitude Adjustment OpenWRT 2.6.39.4 Dec 2011, bleeding edge r29537

[5] Built by Endace Measurement Systems [10]

TABLE I
EQUIPMENT SPECIFICATION

Hardware	Specification
Ninjabox 500: Traffic generator	2x Intel Xeon 5130 2.00 GHz 4GB RAM Ninjabox Linux 2.6.x 64bit 2x DAG 4.5 G2 Cards
Host A: Traffic Generator/Sink	Intel Core i5 2.80 GHz 4GB RAM FreeBSD 8.2-RELEASE 1x Intel Pro 1000 PCI NIC
Host B: Traffic Client/Server	Intel P4 2.66 GHz 2GB RAM FreeBSD 8.2-RELEASE 2x Intel Pro 1000 PCI NIC 1x Intel Pro 100 PCI NIC
TP-Link Router	TL-WR1043ND Atheros AR9132 400Mhz (MIPS) TP-Link Linux 2.4 (v3.13.4) OpenWRT 2.6.39.4 (Trunk) r29537 32MB RAM 8MB Flash 4x GigE LAN Ports 1x GigE WAN Port 1x GigE LAN/WiFi bridge

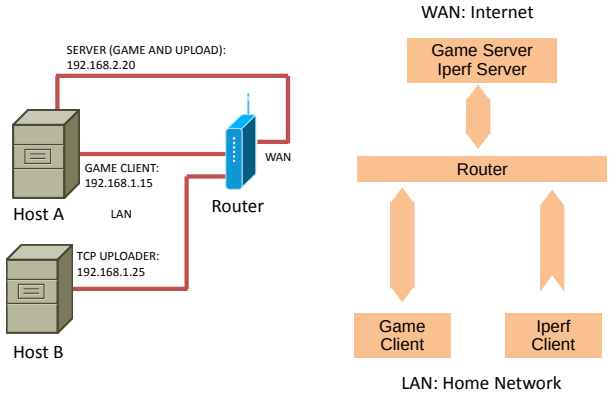

Fig. 2. Host A has two interfaces (one on the LAN subnet and one on the WAN). The WAN interface sends game client server traffic and acts as an Iperf server. Host B simulates a bulk TCP upload to the Iperf server. The game server and game client are hosted on the same workstation so that the OWD can be calculated without the need for clock syncronisation.

The tool tcpreplay[6] was used to replay traffic from the online first person shooter game 'Return to Castle Wolfenstien: Enemy Territory' (ET). TCP upload traffic was generated at run-time using Iperf[7].

The previously captured online game traffic was from the SONG [13] database. It consisted of a single bi-directional ET flow. Flows from online games such as ET provide a good representation of the type of real-time interactive traffic that can benefit from QoS schemes. Although the bandwidth requirements are low (61 kbps), it is highly sensitive to delay and the user experience can degrade significantly as latency increases [14] [15]. This sensitivity to delay is common to other types of real-time interactive traffic, such as VoIP [16].

To separate playback of the client and server game traffic a tcpreplay cache file was created using tcpprep. Tcprewrite was used to reformat the flow endpoints to work within the testbed address space.[8] We used tcpdump to capture packets addressed to the game server at the client and server interfaces, and calculated the OWD of each packet.

The commands used to replay the interactive and non-interactive traffic are in Appendix VIII-D and VIII-E.

B. *Forwarding Throughput*

We measured the maximum throughput of the router when forwarding packets upstream (LAN to WAN) using

[6]Tcpreplay is a suite of tools that allows the replay of previously captured traffic. Including (but not limited to) TCP and UDP flows. http://tcpreplay.synfin.net

[7]Tool for measuring maximum TCP or UDP performance on a link. http://sourceforge.net/projects/iperf

[8]Both tcpprep and tcprewrite are part of the tcpreplay suite.

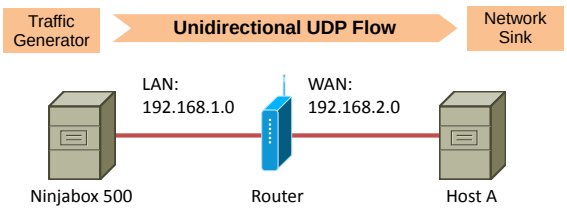

Fig. 3. Throughput testing: The Ninjabox generates a stream of UDP packets of fixed size at different packet rates

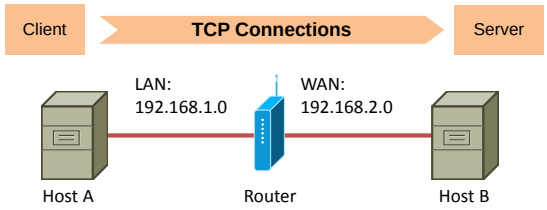

Fig. 4. Measuring maximum concurrent connections: Host A initiates TCP connections to Host B and detects when the router has reached the maximum number of connections

both the factory firmware and OpenWRT. Other than disabling DHCP and wireless, both firmware images are in 'out-of-the-box' configurations. The testbed is shown in Figure 3.

The Ninjabox replays a single uni-directional UDP flow with a fixed inter-frame gap. This flow is forwarded to the network sink on the WAN interface, where it is null routed. To determine the maximum throughput, we alter the frames-per-second sending rate to find the highest rate for a given frame size for which no frames are dropped.

Each frame rate is sustained for 30 seconds, with a 30 second wait time between trials. The Ethernet frame sizes used in the tests were (in bytes): 128, 256, 512, 768, 1024, 1280, 1344, 1500.

This methodology is based on sections 5, 9.1 and 26.1 of [17]. Our approach differs from [17] in a number of ways: we use a trial period of 30 seconds rather than 60 seconds, a minimum Ethernet frame size of 128 bytes rather than 64, and a uni-directional flow rather than a bi-directional flow (from LAN to WAN).

C. Concurrent Connections

We tested the number of concurrent connections the router can handle when running the factory firmware and OpenWRT firmware, and when running OpenWRT with DIFFUSE. As the router maintains a table of all current flows in memory, the maximum number of flows is physically limited by the available RAM (commonly software limited in Linux systems by the Netfilter[9] option 'nf_maxconntrack'). Table II shows the default configured nf_maxconntrack for the two firmwares.

To test the maximum number of connections we used a tool that initiates TCP flows across the router between a client and server host. The setup is shown in Figure 4.

The client first establishes a connection and then transmits a single packet, which the server echoes. If the client receives the echo, it creates a new connection to the server with a new source port. This process is repeated until the client is not able to receive the echo, which indicates that the router has reached the maximum configured connections, or has run out of memory. A 60-second flow timeout was configured for both Netfilter and IPFW to ensure that flows did not time out while establishing the connections.

TABLE II
DEFAULT MAXIMUM CONCURRENT CONNECTIONS

Firmware	Default connections
TPLink 2.13.4	5,120
OpenWRT 2.6.39.4 #5 r29537	16,384

We use Slab memory statistics from */proc/meminfo* to record the memory use before and immediately after running the tool. As both Netfilter and IPFW operate in kernel space, the flow tables are stored in the kernel Slab memory cache, and thus by measuring the Slab size we are able to estimate the memory allocated per flow. More common methods of determining kernel module memory (*/proc/slabinfo* or the tool *slabinfo*) were not available for the version of OpenWRT used. However, using the Slab size to estimate per-flow memory allocation provided an acceptable approximation as there were no additional processes running on the router and the Slab size was constant when the router was idle. The Slab size was sampled multiple times during testing to ensure that it had stabilized before a value was recorded.

D. CPU Load Measurement

We measured the CPU load under different router configurations and packet rates to determine the processing overhead of running DIFFUSE. The topology is shown in Figure 3. We tested the router configured as a CN+AN and as an AN.

We measured the CPU load with the following router configurations:

- Baseline performance

[9]Netfilter is the Linux packet filtering framework for firewalls and connection tracking.

  - Netfilter only
  - Netfilter with IPFW
- Performance as a CN+AN
  - Netfilter with IPFW, DIFFUSE
  - Netfilter with IPFW, DIFFUSE and Dummynet
- Performance as an AN
  - Netfilter, IPFW and Dummynet with a varying number of unique rule entries for priority flows

The commands used to configure DIFFUSE are detailed in Appendix VIII-B.

To create CPU load, we used the Ninjabox to send a single uni-directional UDP stream to the network sink. The frame size remained fixed while the frame per second rate was varied from 2,000 to 8,000 in increments of 1,000. The duration of each trial was 120 seconds, with a 30 second waiting period between trials. No bandwidth limiting was applied to the WAN port.

We used vmstat[10] to measure the CPU load during the test. Although the results provided in this paper are taken from these vmstat measurements, we also used mpstat[11] and statistics from */proc/stat*[12] for cross-checking. We found the CPU load measurements to be consistent across the three techniques.

E. Memory Use

We measured memory use with DIFFUSE enabled, and compared this with the memory use of Netfilter only. Using the connection-creation tool described in Section III-C, we measured the memory use between 1,000 and 10,000 flows, at intervals of 1,000. After the connections were set up, the flows were idle (did not actively send packets). We also measure memory with an increasing number of active flows (at least one packet per second per flow) at a fixed aggregate packet rate.

Tcpreplay was used to send uni-directional UDP flows in the upstream direction to the network sink. All flows originated from a single IP+port pair and were differentiated by the destination IP+port pair. To create the tracefiles a single UDP flow was generated using Iperf, and tcpmerge and tcprewrite were used to duplicate the flow and change the destination IP+port pair to create different flows. The setup is shown in Figure 5.

The total packet rate for each of the tracefiles was 5,000 packets per second[13]. The tracefile details are shown in Table III.

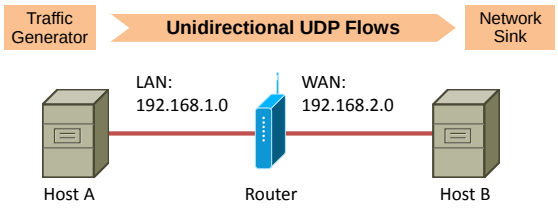

Fig. 5. Per-flow memory use with active flows. Host A generates multiple UDP flows that are null routed at Host B. The number of flows is changed but the aggregate packet rate is always 5,000 pps.

TABLE III
TRACES USED TO MEASURE MEMORY CONSUMPTION

Number of flows	Packets per second per flow
1000	5
1250	4
2500	2
5000	1

We again used Slab statistics to provide an estimate of the amount of memory being consumed by Netfilter and IPFW/DIFFUSE while tracking the flows.

IV. Results

A. Real-time Classification on 1 Mbit and 15 Mbit links

Figure 6 plots the CDF of client to server OWD measured for a single game traffic flow without additional traffic and with additional client to server TCP cross traffic (with and without DIFFUSE traffic prioritisation).

Without additional traffic the game flow has a mean OWD of 3 ms (median 1.5 ms) and there is no packet loss. Adding a single upstream TCP flow causes the mean OWD of the game traffic to increase to 344 ms (median OWD 351 ms), with 7% packet loss. Such a high OWD would negatively impact the game experience of the user [18].

With DIFFUSE enabled the mean OWD is reduced to 32 ms (median 31 ms) without packet loss. This is a significant improvement and would allow a user to play an online game while a file upload is in progress. Figure 7 plots the OWD of the game flow over time, with DIFFUSE and without. We can see that with DIFFUSE at the very beginning of the flow OWD values approach 200 ms – this is due to the flow not being classified as 'priority' at this point. The classification packet window (see Section II) must be filled before a classification can be made and the flow prioritised. However once the game flow has been identified and prioritised (a process that takes less than two seconds) the OWD is more consistent.

[10] A tool that provides process, memory and I/O statistics.

[11] Another process and memory measurement tool.

[12] The /proc/stat file tracks kernel activity, including CPU time, from which CPU load can be calculated.

[13] We also used the tcpreplay option "–pps 5000" to ensure that the packet rate was consistent

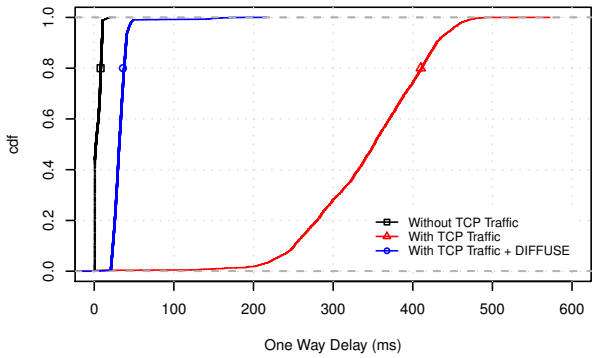

Fig. 6. CDF of one way delay for game flow on a 1 Mbps link. Measured in the client-to-server direction.

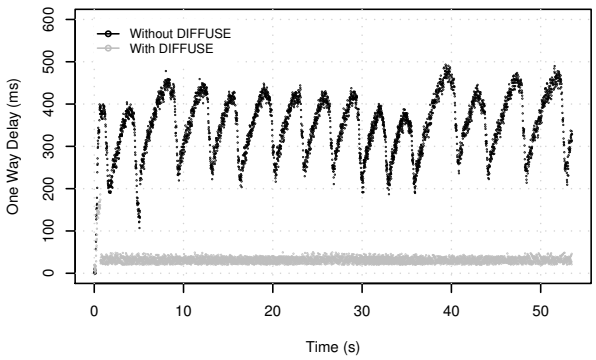

Fig. 7. With and without DIFFUSE: per-packet one way delay for game flow with TCP cross traffic on a 1 Mbps link. Measured in the client-to-server direction

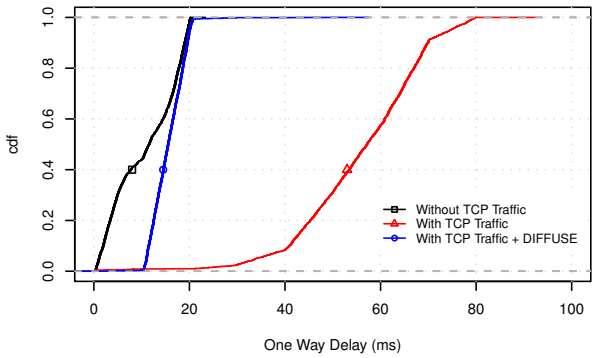

Fig. 8. CDF of one way delay for game flow on a 15Mbit link. Measured in the client to server direction.

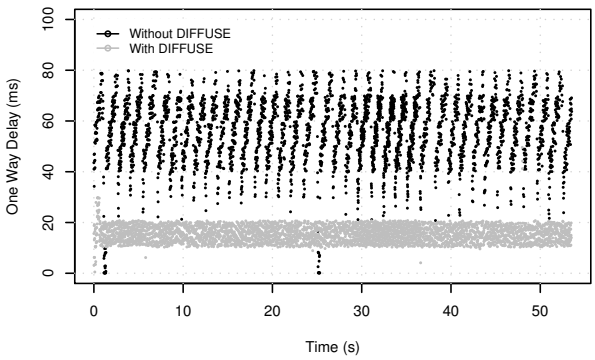

Fig. 9. With and without DIFFUSE: per-packet one way delay for game flow with TCP traffic on a 15Mbit link. Measured in the client to server direction

Without DIFFUSE the game flow packets are queued and delayed behind the TCP packets as they exit the router, seen here in the cyclic pattern of measured OWD values (ranging from 200 ms to 500 ms) as the upstream queue fills and empties. TCP will increase its throughput (congestion window size) until the queue is filled and packets are dropped. Once packets are dropped TCP will back-off, reducing the congestion window.

We also measured the CPU load and memory use during the test. CPU load was found to be less that 1%, while memory usage was 16–20 kB.

Figures 8 and 9 plot the cumulative OWD and OWD over time with a 15 Mbps connection. The single TCP upload increases the OWD of the game traffic flow to a mean of 55 ms (median 57 ms) with 5% packet loss from the game client to server. With DIFFUSE enabled the mean and median OWD was 15 ms, without packet loss.

The TCP cross traffic again induces a cyclic OWD pattern on the game flow, though the period between the queue filling and emptying is shorter. This speeding up is due to the lower overall OWD (the serialisation time for a 15 Mbps link is shorter than that of a 1 Mbps link). Lower OWD allows the TCP congestion window to grow more quickly before packet loss occurs and the window collapses (allowing the queues to empty).

Although the overall latency of the game flow is not high without prioritisation (a maximum OWD of 80 ms), packet loss did occur and the cyclic pattern of the measured OWD indicates that a large amount of jitter[14] was introduced. Jitter can have a negative impact on delay-sensitive applications [16].

Multiple TCP flows (as might be expected on a high speed link) would further impact the real-time flow. Enabling DIFFUSE can prevent the packet loss and reduce the delay caused by TCP flows filling a shared upstream buffer.

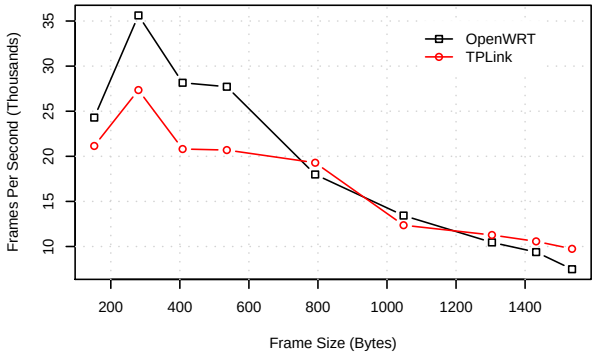

Fig. 10. Frame forwarding rates at different frame sizes

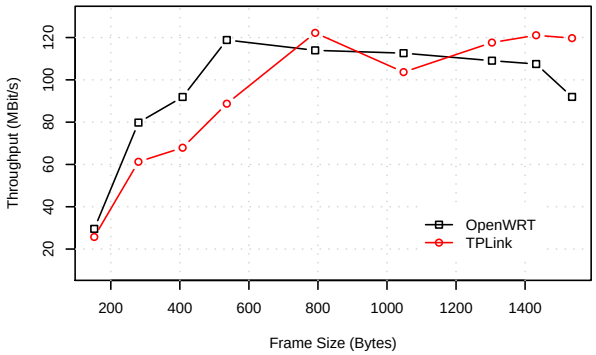

Fig. 11. Throughput at different frame sizes

B. Throughput

Figure 10 compares the frames per second performance against frame size when running default configurations of the factory firmware and OpenWRT firmware. The throughput in Mbps is shown against frame size in Figure 11.

At frame sizes larger than 800 bytes, there is not a significant difference in throughput between the firmware images, and both were able to sustain throughput above 100 Mbps for the test duration. However throughput decreases significantly as frame size decreases and with 128 byte frames approaches 30 Mbps.

In the range of 256–512 bytes the OpenWRT firmware outperforms the TP-Link firmware by approximately 20 Mbps as it is able to sustain a higher frame per second rate. An interesting observation is the peak in performance with 256 byte packets. There is no immediate indicator as to the cause of this behaviour, though it was common to both firmware images.

Although the forwarding throughput did not approach the gigabit speed of the underlying switching hardware, it is adequate for ADSL-like speeds of 1–24 Mbps. Overall the performance of the router does not appear to be substantially different when running OpenWRT firmware in place of the original firmware.

[14] Or packet delay variation, the difference in delay between successive packets in a flow.

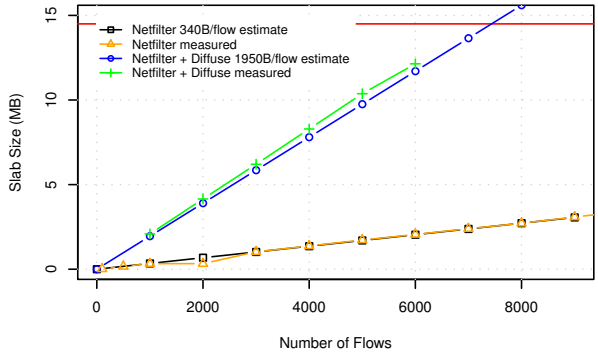

Fig. 12. Projected and measured memory use with multiple concurrent flows. The red line indicates the available memory once DIFFUSE has been loaded.

C. Concurrent Connections

We compared the measured maximum number of concurrent connections against the default 'nf_maxconntrack' values of Netfilter for the TP-Link firmware and OpenWRT firmware.

In both instances we were able to open the maximum number of connections as defined in nf_maxconntrack. By measuring the change in Slab size before and after opening the connections, the memory use per flow table entry for Netfilter was estimated to be approximately 340 bytes.

OpenWRT leaves approximately 15 MB of memory free once it has loaded. Given this we were able to change nf_conntrackmax and open 32,000 concurrent connections. It would appear that the factory firmware limit of 5,120 connections reflects the target market of the device rather then any hardware/memory limitations.

D. Memory Use

Figure 12 shows memory use against the number of current flows, as measured with the connection creation tool. We also extrapolate the memory use based on an estimate of 340 bytes/flow for Netfilter, and 1950 bytes/flow when running DIFFUSE as a CN+AN. The memory use per flow when configured as an AN is not significantly larger than that of Netfilter alone (as the AN only keeps flow state for priority flows, not every flow).

The size of the DIFFUSE kernel module is 131 kB. However after loading the module total memory available is approximately 14.5 MB. We thus use 14.5MB

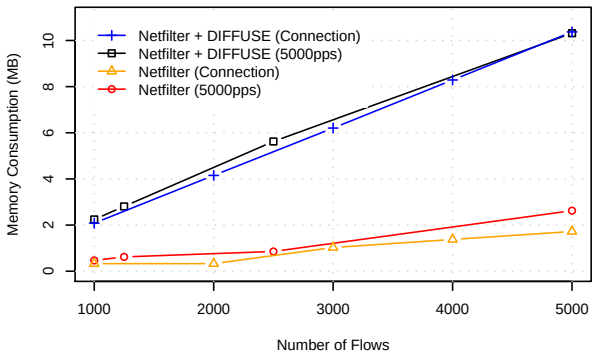

Fig. 13. Comparing concurrent flow memory use of flows with 5,000 packets per second against flows that are established but idle (no packets being transmitted).

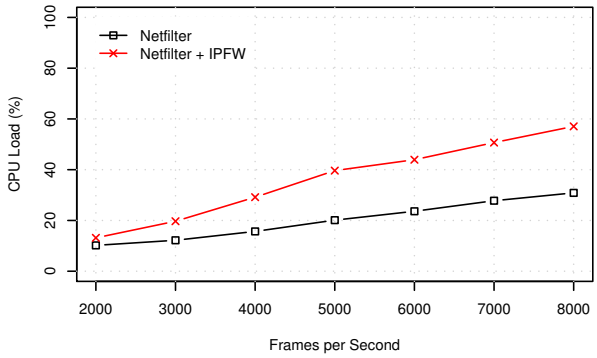

Fig. 14. CPU load depending on the frame forwarding rate for Netfilter and Netfilter with IPFW.

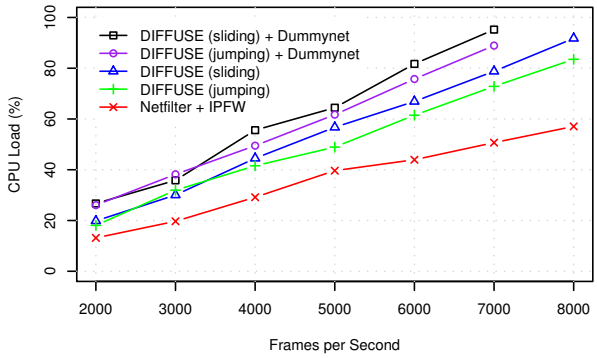

Fig. 15. CPU load against frame rate for DIFFUSE (sliding and jumping window feature calculations), with and without Dummynet priority queuing

as a 'ceiling', as indicated by the horizontal red line in Figure 12.

The estimated values for per-flow memory use appear to be accurate, and match closely with the actual measured memory use. DIFFUSE in CN+AN mode uses considerably more memory than Netfilter alone. The increase in per-flow memory use and the reduction in the amount of memory available once the DIFFUSE kernel module is loaded means that the router is limited to 6,000 concurrent flows with DIFFUSE enabled. This still exceeds the factory default setting of the router and is on par with a number of residential router devices [19].

We also measured memory use with multiple flows that have an aggregate packet rate of 5,000 pps and at least one packet per second for each flow (see Table III). The results are shown against those measured using the connection utility in Figure 13.

Per-flow memory use is higher with continuously active flows. This can be expected as the kernel must in this case process 5,000 packets each second in addition to keeping flow state in the flow tables. Given the memory increase measured when flows are active, an appropriate nf_maxconnection value when using DIFFUSE might also be 5,120 (TP-Link default). With the 12.5 MB available, this would provide enough concurrent connections to be established while leaving some memory overhead for packet processing.

E. CPU Load

Figure 14 plots the CPU utilisation for a range of packet rates when running Netfilter only and Netfilter with IPFW. DIFFUSE is not enabled. We used a single uni-directional UDP flow with a fixed Ethernet frame size of 512 bytes.

By enabling IPFW the router is essentially running two packet filters – Netfilter and IPFW. This results in a significant increase in CPU usage, with utilisation over 50% at 8,000 pps with IPFW enabled, against 30% with Netfilter alone.

Figure 15 plots the CPU load when running DIFFUSE with a classification model loaded and also with Dummynet enabled for prioritisation. We see a higher CPU utilisation when classification is enabled. Using sliding windows results in a 5% increase in CPU load over using jumping windows (see Section II for an explanation of windows). Using the features available in DIFFUSE 0.4, a jumping window does not perform feature calculation as often as a sliding window and thus will generally result in a lower load. Enabling prioritisation using Dummynet results in a 20% increase in CPU load for jumping and sliding window classifiers.

Figure 16 shows the CPU Load against the number of concurrent flows with a different number of buckets configured for the DIFFUSE flow table (flow state is kept in a hash table). The aggregate packet rate is always

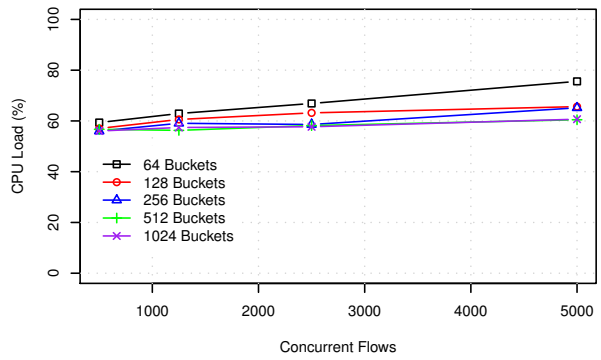

Fig. 16. CPU Load against the number of flows with different bucket sizes for the DIFFUSE flow table. The packet rate is 5,000 pps.

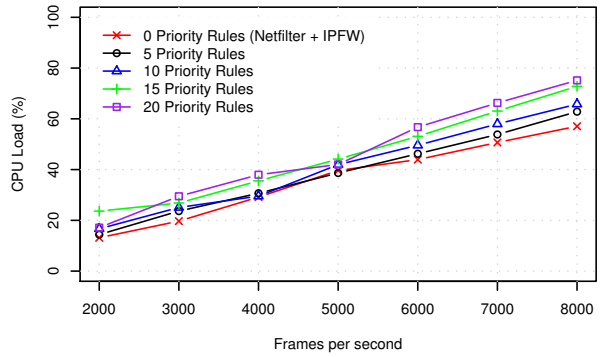

Fig. 17. Router as an Action Node. CPU Load against frame rate with increasing numbers of priority flow rules. IPFW and Dummynet provide prioritisation

5,000 pps. We see that the CPU performance scales well and is not dependant on the number of flows, provided an appropriate bucket size is specified for the flow table. With 512 buckets configured, CPU load does not alter significantly when the number of concurrent flows increases from 250 to 5,000 flows. For a CN+AN the number of concurrent flows is limited to approximately 5,000 (see Section IV-D), hence 512 would be a suitable bucket size that balances flow table memory usage and CPU performance.

Enabling DIFFUSE CN+AN and prioritisation significantly increases CPU load. The increased CPU utilisation results in a reduced maximum throughput when compared with the factory firmware and the default OpenWRT configuration. With a 512 byte frame and nearing 100% CPU load, the DIFFUSE enabled router has a throughput of 28.6 Mbps, while the factory firmware and default OpenWRT measured 90 Mbps and 120 Mbps respectively.

Figure 17 shows the CPU load when running the router as an AN. To simulate an AN we enable IPFW and Dummynet and add a number of static rules that represent active priority flows. As with previous tests Netfilter was also active.

Our measurements represent the worst case scenario – the time taken for a packet to check against all priority rules before matching a default 'non-priority' rule. With a frame size of 512 bytes, for a throughput of 32.8 Mbps we can expect less than 80% CPU load with 20 priority flow rules.

A higher maximum throughput can be obtained when running as an AN only, as the router does not need to calculate features and classify flows. It is also not expected that users would frequently have 20 concurrent priority flows, thus in most cases the CPU load would be lower. Using the router as an AN results in a saving of 20–30% in CPU load against using the router as a CN+AN. A significant proportion of CPU load (up to 20%) can be attributed to enabling prioritisation through Dummynet.

Although enabling DIFFUSE does increase CPU load, the router is able to sustain ADSL-like speeds when configured as an AN or as a CN+AN. The ability of DIFFUSE to decouple AN and CN functionality is clearly beneficial where devices with limited CPU performance must be used as ANs. With further optimisations DIFFUSE may be able to support higher throughput.

V. DIFFUSE Optimisations and Future Work

There are a number of areas for optimisation that may lead to increased performance in the future.

Currently DIFFUSE is integrated with IPFW. When ported to OpenWRT this results in two packet filters – IPFW and Netfilter – running concurrently, adding CPU load. In the future DIFFUSE could be integrated with Netfilter. As shown in Figure 17, using Dummynet for prioritisation increases CPU load by up to 20%. CPU Load could be reduced in the future by having DIFFUSE communicate directly with Linux Traffic Control (TC) for prioritisation.

Code changes may also allow future versions of DIFFUSE to be more memory efficient. Currently DIFFUSE allocates memory in the kernel cache using *kmalloc*. Kmalloc allocation occurs with fixed blocks ranging in size from 8 bytes to 8192 bytes. This can result in inefficiency when the amount of memory required does not fill an entire kmalloc block (more memory is allocated then actually needed).

For example when using kmalloc to allocate 800 bytes of memory, a 1024 byte object is created, resulting in a "loss" of 224 bytes. Allocating 600 bytes would also

create a 1024 byte object – 424 bytes are allocated that are not needed. DIFFUSE could be modified to use custom-sized kernel cache memory blocks rather than the default allocators, which may lead to increased memory efficiency.

Memory use and CPU performance depends on the classification model and the features used. In this paper we used a single model (with packet length and packet count features) and tested sliding and jumping windows with the same window size. Further tests of different classifiers with parameter tuning (e.g. different window sizes, additional features) could provide more insight as to the processing cost of DIFUFSE.

VI. Conclusions

We evaluated the impact of using DIFFUSE on the TP-Link TL-WR1043ND residential router running the OpenWRT operating system. We tested the router in Action Node (AN) and Classifier Node (CN+AN) configurations, and compared forwarding performance against the baseline router performance under several scenarios. The router is able to run DIFFUSE, perform automated classification and prioritise flows without pre-configuration of static QoS rules. Enabling DIFFUSE provides a substantial reduction in latency and jitter experienced by online game traffic flows when exposed to TCP cross traffic. Although there is a decrease in the forwarding throughput and an increase in memory use, performance is suitable for typical ADSL or ADSL2+ connections.

In testing under a realistic use-case scenario on a 1 Mbps link, the mean OWD of an online game flow was reduced from 344 ms to 32 ms after DIFFUSE was enabled. Packet loss from the game client to server was eliminated (originally 7%). Enabling DIFFUSE on a 15 Mbps link reduced the mean OWD from 55 ms to 15 ms. Game flow packet loss was also eliminated on this link (from 5%).

Enabling DIFFUSE classification and prioritisation (CN+AN) results in up to a 40% increase in CPU load over the baseline router performance for a given packet rate. At near 100% CPU utilisation with a constant flow of 512 byte Ethernet frames, the forwarding throughput when running DIFFUSE was 28.6 Mbps, around 90 Mbps less than OpenWRT in default configuration (24.5 Mbps for the CN+AN at 80% load). Configuring the router as an AN provided higher throughput of 32.8 Mbps at 80% CPU load.

Per-flow memory use was found to be approximately six times higher with DIFFUSE enabled (1950 bytes per flow compared to 340 bytes per flow). As a CN+AN, the router was able to handle 5,000 concurrent flows at approximately 60% CPU load. This is comparable to the 5,120 maximum concurrent flows configured for the router 'out-of-the-box'.

Despite the additional CPU load and memory usage, we find that running DIFFUSE on a residential router would be practical. With increased CPU clock speeds and RAM (TP-Link's more recent TL-WR2043ND has 64MB), and with some code optimisations, DIFFUSE could be run as both a CN+AN or AN with an even greater number of concurrent connections and higher throughput.

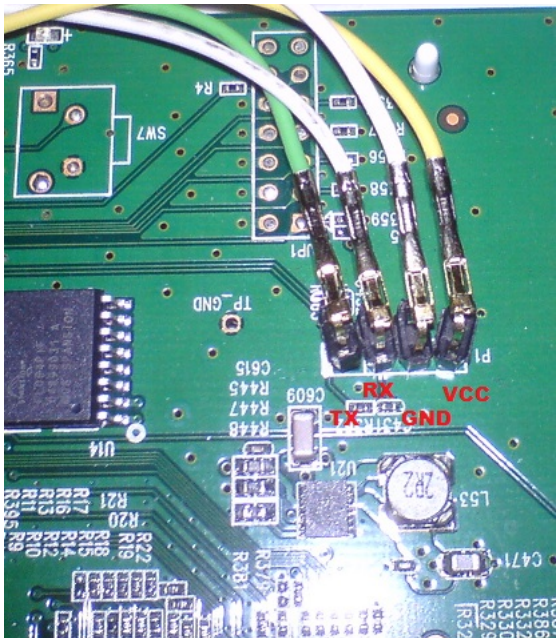

Fig. 18. A modified WR-1043ND. Header pins are soldered to the serial connectors on the motherboard. The transmit, receive, ground and Vcc are indicated. Photo courtesy of Warren Harrop.

VII. Acknowledgements

The authors would like to acknowledge the assistance of Grenville Armitage who provided valuable feedback and advice over the course of the experiment.

VIII. Appendix

A. Adding a serial port to gain access to factory firmware

Adding a serial port to the TL-WR1043ND is not difficult, but does require soldering directly onto the router motherboard. Specific details on the process can be found at the OpenWRT TL-WR1043ND page [9].

Figure 18 shows the WR-1034ND motherboard with serial header pins added.

In addition to adding a serial port, a serial cable must be purchased (or made) that can connect from a PC to the router. We use a USB to 3.3V TTL-serial[15] that has been modified from a Nokia USB phone cable[16].

To gain root access to the stock firmware, power on the router with the serial cable connected. Use *cu* (available on BSD/Linux) to connect to the router. The commands are shown in Table IV. To log in, use the username *root* with password *5up*.[17]

B. Loading the DIFFUSE classifier

When logged into the router (and after loading the IPFW kernel module if necessary), we used the commands in Table V to load DIFFUSE and a classification model. By default DIFFUSE will perform sliding window calculations. To enable jumping windows, the "jump-windows" parameter is used.

C. Setting up priority queuing

Table VI lists the commands used to match DIFFUSE classifications and prioritise packets. In the example the bandwidth is limited to 1 Mbps. The parameter *match-if-class myclass:rt* means that the rule will evaluate true for packets tagged as 'rt', (i.e. *real-time*). The label 'other' refers to all other traffic types. We include a default rule sending any traffic that has not been classified to the low-priority queue. See the IPFW/Dummynet homepage for a guide to IPFW syntax.

D. Sending traffic with tcpreplay

Tcpreplay was run with the "–cachefile" option to replay the server and client packets out of the appropriate "client" and "server" interfaces and across the router. A cachefile is required when replaying bi-directional traces out of multiple interfaces. Table VII shows the command used to replay the game traffic.

E. Generating TCP Cross Traffic with Iperf

The Iperf commands used are shown in Table VIII. These commands generate a TCP upload from the client host to the server for a duration of 60 seconds. Given the same congestion control mechanism, TCP will behave similarly across different applications, so Iperf provides a convenient emulation of activities such as uploading a video or transferring files to a cloud services.

TABLE VIII
IPERF COMMAND

LAN Client: iperf -c 192.168.2.20 -t 60 WAN Server: iperf -s
-c : Act in client mode, connect to 192.168.2.20 -t : Send data for 20 seconds -s : Act as server and wait for connections

[15]TTL Serial operates between 0V and 3.3–5V. The TL-WR1043ND uses 3.3V (a TTL-5V cable may damage the router. A 15V RS-232 cable is likely to cause damage)

[16]http://www.dealextreme.com/p/usb-cable-for-nokia-n1200-1208-1650-2630-2670-13638

[17]This password has been tested for factory firmware 3.13.4.

REFERENCES

[1] L. Stewart, G. Armitage, and A. Huebner, "Collateral Damage: The Impact of Optimised TCP Variants On Real-time Traffic Latency in Consumer Broadband Environments," in *IFIP/TC6 NETWORKING 2009*, Aachen, Germany, 11-15 May 2009. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01399-7_31

[2] S. Zander and G. Armitage, "Practical Machine Learning Based Multimedia Traffic Classification for Distributed QoS Management,," in *36th Annual IEEE Conference on Local Computer Networks (LCN 2011)*, Bonn, Germany, 4-7 October 2011.

[3] S. Zander. (2012, March) Distributed firewall and flow-shaper using statistical evidence (diffuse). [Online]. Available: http://caia.swin.edu.au/urp/diffuse/

[4] L. Rizzo, "Porting ipfw to linux and windows." Presented at BSDCan Conference 2010, 2010.

[5] OpenWRT.org. (2012, March) Table of hardware. [Online]. Available: http://wiki.openwrt.org/toh/start

[6] OpenWRT.org. (2012, March) Opkg package manager. [Online]. Available: http://wiki.openwrt.org/doc/techref/opkg

[7] OpenWRT.org. (2012, March) Openwrt buildroot. [Online]. Available: http://wiki.openwrt.org/doc/howto/build

[8] OpenWRT.org. (2012, March) Openwrt buildroot. [Online]. Available: http://wiki.openwrt.org/doc/howto/obtain.firmware.sdk

[9] E. M. S. Ltd. (2012, March) Tp-link tl-wr1043nd. [Online]. Available: http://wiki.openwrt.org/toh/tp-link/tl-wr1043nd

[10] Endace. (2012, March) Endace measurement systems ltd. [Online]. Available: http://www.endace.com/

[11] B. Cohen. (2009, June) The bittorrent protocol specification. [Online]. Available: http://bittorrent.org/beps/bep_0003.html

[12] iiNet. (2012, March) iinet nbn plans. [Online]. Available: http://www.iinet.net.au/nbn/nbn-plan-residential.html

[13] G. Armitage. (2012, March) Song - simulating online networked games database. [Online]. Available: http://caia.swin.edu.au/sitcrc/song/

[14] G. Armitage, "An Experimental Estimation of Latency Sensitivity in Multiplayer Quake 3," in *11th IEEE International Conference on Networks (ICON 2003)*, Sydney, Australia, 28-1 September 2003, pp. 137–141. [Online]. Available: http://dx.doi.org/10.1109/ICON.2003.1266180

[15] S. Zander and G. Armitage, "Empirically Measuring the QoS Sensitivity of Interactive Online Game Players," in *Australian Telecommunications Networks & Applications Conference 2004 (ATNAC 2004)*, Sydney, Australia, 8-10 December 2004, pp. 511–517. [Online]. Available: http://caia.swin.edu.au/pubs/ATNAC04/zander-armitage-ATNAC2004.pdf

TABLE IV
CONNECTING TO THE ROUTER USING CU

cu -s 115200 -l /dev/cuaU0
-s : The baud rate is 115200 -l : the serial line to connect to. We use a USB serial cable – on FreeBSD this begins with /dev/cuaU0. If using the DIFFUSE Ubuntu Linux VirtualBox image, the serial line will appear similar to /dev/ttyUSB0

TABLE V
LOADING DIFFUSE AND A CLASSIFIER MODEL

With Sliding Windows
ipfw feature delete plen ipfw feature plen config module plen window 20 payload-len ipfw feature delete pcnt ipfw feature pcnt config module pcnt window 40 ipfw mlclass myclass config algorithm c4.5 confirm 20 –line continued– model /root/fps_model/rt_vs_other_plen+pcnt.c45.diffuse
With Jumping Windows
ipfw feature delete plen ipfw feature plen config module plen window 20 jump-windows payload-len ipfw feature delete pcnt ipfw feature pcnt config module pcnt window 40 jump-windows ipfw mlclass myclass config algorithm c4.5 confirm 20 –line continued– model /root/fps_model/rt_vs_other_plen+pcnt.c45.diffuse

TABLE VI
IPFW QUEUE SETUP: USING DUMMYNET TO CONFIGURE A 1 MBIT PIPE WITH TWO QUEUES

ipfw add pipe 1 ip from any to any ipfw pipe 1 config bw 1Mbit ipfw queue 1 config pipe 1 weight 100 ipfw queue 2 config pipe 1 weight 5 ipfw add 10 queue 1 ipv4 from any to any match-if-class myclass:rt in ipfw add 10 queue 2 ipv4 from any to any match-if-class myclass:other in ipfw add 10 queue 2 ipv4 from any to any in

TABLE VII
TCPREPLAY ET GAME TRAFFIC

tcpreplay –intf1=em1 –intf2=em0 –cachefile=ET_trace.cache ET_Trace.dmp
–intf1 : server traffic interface. The server traffic will be played back via this interface –intf2 : client traffic interface. The client traffic will be played back via this interface –cachefile: This cache file defines the client and server endpoints for flows.

[16] L.Stewart, D. Hayes, G. Armitage, M. Welzl, and A. Petlund, "Multimedia-unfriendly TCP Congestion Control and Home Gateway Queue Management,," in *ACM Multimedia Systems Conference (MMSys 2011)*, San Jose, California, 23-25 February 2011. [Online]. Available: http://portal.acm.org/citation.cfm?id=1943558

[17] S. Bradner and J. McQuaid, "Benchmarking methodology for network interconnect devices," United States, 1999.

[18] M. Dick, O. Wellnitz, and L. Wolf, "Analysis of factors affecting players' performance and perception in multiplayer games," in *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, ser. NetGames '05. New York, NY, USA: ACM, 2005, pp. 1–7. [Online]. Available: http://doi.acm.org/10.1145/1103599.1103624

[19] S. N. Builder. (2012, March) Router charts: Maximum simultaneous connections. [Online]. Available: http://www.smallnetbuilder.com/lanwan/router-charts/bar/77-max-simul-conn