Mongoose: Throughput Redistributing Virtual World

Iain Oliver, Alan Miller and Colin Allison
School of Computer Science
University of St Andrews
Email: {iao, ca, alan.miller}@st-andrews.ac.uk

***Abstract*—Metaverses provide a framework for developing distributed 3D Internet applications where users gain presence through the proxy of an *avatar*. They offer much of the engagement of on line 3D games but support heterogeneous applications. From the network perspective metaverses are similar to games in that timeliness is important but differ in that their traffic is less regular and requires more bandwidth. The motivation for our study flows from using virtual worlds to support experiential learning and to promote cultural heritage; the applicability of the results is wider. The responsiveness of the system is effected by interactions between avatar activity, application traffic regulation and network conditions. Through measurement and analysis current virtual world traffic regulation is evaluated and compared with Transmission Control Protocol fair rate. The measurement study motivates the design of Mongoose, which adds measurement based packet regulation to open virtual world clients and servers. Mongoose combines isolating distinct functional components with efficient use of network resources and fairness to other traffic.**

I. Introduction

The work in this paper is motivated by our experience in using Multi User Virtual Environments to create interactive applications that support experiential collaborative learning about computer networking and other subjects. Our experience is that such applications add a new dimension to learning that students find engaging and motivating. Multi-User Virtual Environments (MUVE) are a relatively new class of application. They offer much of the engagement associated with computer games and provide a 3D environment within which the user is represented by an avatar. Through this proxy users achieve presence, they can navigate the world and interact with the environment whilst cooperating with other users. Modelling tools allow terrain, buildings and artifacts to be created and edited. The environment is programmable and supports multi media. These characteristics make MUVEs a powerful platform for developing 3D educational applications.

Computer networking is a naturally engaging subject; routes are decided dynamically and may change rapidly, packets collide and are retransmitted, protocols interact. Unfortunately this interaction can be difficult for students to engage with directly. We have developed a set of virtual world resources that address this. WiFi Island is a virtual laboratory that the enables exploration of WiFi protocols. Learners set up virtual wireless networks, experiment with them and solve the "hidden node" problem [1]. In Routing Island routers, sinks, sources and links are all created with the click of a mouse. Learners add, delete or alter the "weight" of links, observe the effect on routing tables and traffic flows and see routing algorithms in action.

OS island	Country	Downstream (Kbits/s) St	W	F	T
Cathedral	UK	189	253	434	357
Sparta	UK	51.0	214	224	269
Linlithgow	UK	102	278	292	369

TABLE I
Throughput for Standing, Walking, Flying and Teleporting for Three OpenSim Islands.

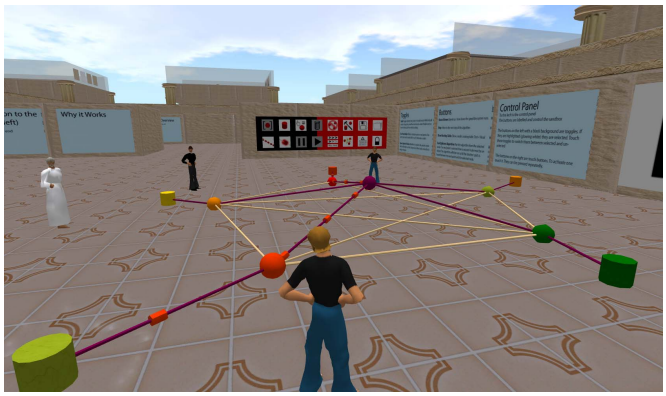
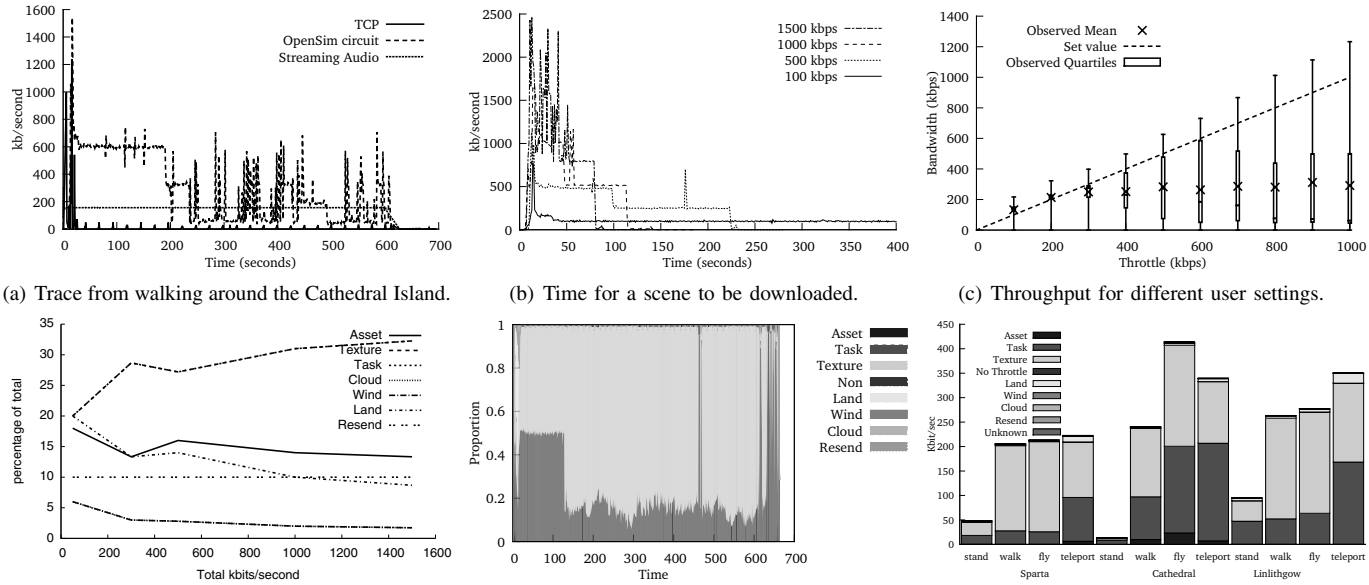


Fig. 1. Computer Science students simulate a routing algorithm in OpenSim.

We have also developed a Virtual Humanitarian Disaster, which enables management students to safely explore dilemmas aid workers face, and a Virtual Archaeological Excavation [2], which enables Classics students to explore a Byzantine Basilica. Work in collaboration with Historic Scotland[1] and Education Scotland[2] is ongoing to develop reconstructions of ruined monuments of major historical significance.

The virtual world platform used is OpenSim[3] an open source 3D application server which aims to become an "extensible server of the 3D web". It is compatible with clients for the commercial virtual world service Second Life (SL)[4] [3]. OpenSim allows organisations to host their own virtual world servers. Using OpenSim allows virtual world environments to be created with functionality and on a scale that would be prohibitively expensive in SL. OpenSim based work is also

[1]http://www.historic-scotland.gov.uk
[2]http://www.educationscotland.gov.uk
[3]http://opensimulator.org
[4]http://secondlife.com



(a) Trace from walking around the Cathedral Island.

(b) Time for a scene to be downloaded.

(c) Throughput for different user settings.

(d) Client bandwidth allocation between channels.

(e) Variation of channel bandwidth share.

(f) Standing, walking, flying and teleporting.

Fig. 2. Measurements of OpenSim network traffic: showing variation in throughput and bandwidth allocation between channels within a circuit.

underway to support: content creation, application development, resource sharing and service provision [4]. This paper presents the Mongoose Client and Server, which add TCP Fair Quality of Service aware traffic management to open virtual worlds.

1) Virtual Worlds and Network Communication: The throughput required by MUVEs is typically orders of magnitude greater than Computer Games [5]. The throughput is variable and depends on factors such as the composition of the space and avatar activity [5], [6]. Whereas First Person Shooters' (FPS) and Massively Multiplayer Online Role Playing Games (MMORPG) clients maintain a model of the environment, in MUVEs this is maintained primarily at the server. Consequently, as Avatars travel within a MUVE information about the environment such as topography of the land, the shape of objects and their textures, is communicated across the network. The trace shown in Fig. 2(a) shows variation in throughput for one client during an OpenSim session, TABLE I shows the average throughput for the activities of standing, walking, flying and teleporting for three OpenSim islands.

Virtual world traffic can be decomposed into components: texture traffic transports images for object surfaces; updates to objects define the position and shape of environment primitives; Avatar Control Traffic (ACT) provides updates to avatars locations. ACT fulfills the same function as and exhibits similar behaviour to FPSs' [3], [7] and MMORPGs' [8] traffic, is made up of a large number of small packets and is sensitive to delay. Providing bandwidth guarantees for ACT helps preserve interactivity. Object and texture traffic is less regular, has higher bandwidth requirements, larger packets and is less sensitive to delay. Fig. 2(b) represents traces from an OpenSim island downloading at a range of throughput allocations, it shows that increasing a client's bandwidth limit can reduce download times.

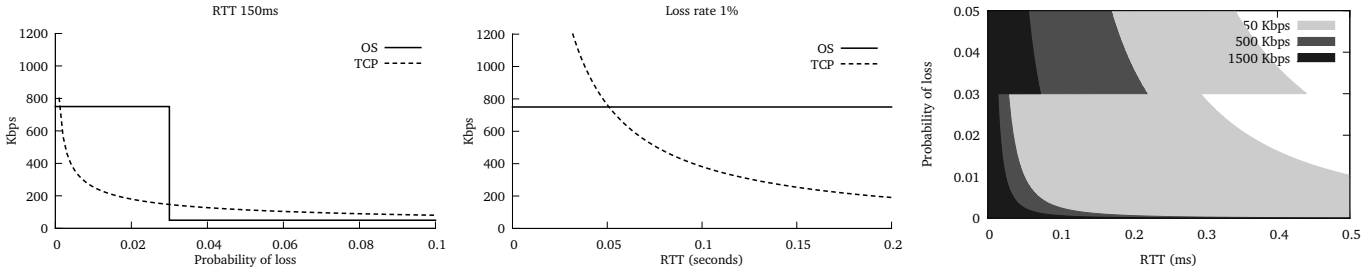
2) TCP and Internet Congestion Control: For the last twenty years TCP's congestion [9] control algorithms have helped prevent congestion and distribute Internet bandwidth fairly. The Transmission Control Protocol is an end to end protocol, which provides a reliable controlled service between applications. In the late 1980s Raj Jain defined a notion of Max-Min fairness [10], and a class of Additive Increase Multiplicative Decrease (AIMD) [11] algorithms. These operate the network at an optimum level and distribute resource in a Max-Min fair manner. Van Jacobsen designed Slow Start and Congestion Avoidance algorithms, which were incorporated into TCP [9]. The combination of Congestion Avoidance and duplicate acknowledgments form an AIMD algorithm and achieves statistical fairness.

In [12] it is shown that, for long lived TCP Reno flows, the data transfer rate is inversely proportional to the square root of packet loss and can be characterised by equation 1 [5]. This equation is used as a reference point for evaluating the behaviour [15], [16] of new congestion control algorithms to determine whether they are *TCP Fair*.

$$X = \frac{S}{RTT\sqrt{p}} \tag{1}$$

TCP Reno is unable to quickly utilise the full capacity of high bandwidth paths for bulk data transfers, this has been addressed by Compound TCP [17] used in Vista and Windows 7 and CUBIC [18] used in Linux kernels. These are shown to be TCP Fair in [19] and elsewhere. Virtual world traffic

[5]TCP Friendly Rate Control [13] uses a more complex formula [14] that models the specific behaviour of TCP RENO and takes into account such factors as flow control window and multiple slow start phases.



(a) Loss against throughput. (b) RTT against throughput. (c) RTT against Loss.

Fig. 3. TCP and OpenSim client algorithm comparison: at low to medium loss levels and larger RTTs OpenSim uses more bandwidth than TCP

is streamed and application limited, consequently the Gbps transfer rates these TCP variants are designed for do not arise and their enhancements are not necessary for virtual worlds.

Real time media applications often do not require the reliability provided by TCP. TCP Fair Rate Control (TFRC) [13], [20] and the Datagram Congestion Control Protocol (DCCP) [21], [22] are examples of TCP Fair congestion control schemes which are intended for use with non-TCP applications.

Virtual world traffic differs has soft real-time constraints. It differs from games in that bandwidth requirements are higher, and from audio and video streaming traffic in that it is more sensitive to delay and jitter. Hence designing traffic management for virtual world applications is a new challenge. This paper makes contributions to meeting the challenge, these are:

- measurements of OpenSim traffic, which motivate TCP Fair QoS aware traffic management for virtual worlds,
- the design, implementation and evaluation of the Mongoose virtual world client, which uses a client side "window tracking, rate" algorithm,
- the design, implementation and evaluation of the Mongoose virtual world server, which uses dual layer token buckets for throughput redistribution.

Our methodology combines, measurement, experimentation, analysis, development and evaluation. in order to understand OpenSim traffic we have made measurement of live OpenSim systems, carried out experiments where avatars performed a series of common tasks and also analysis of source code. A measurement infrastructure was developed to support the capture and analysis of OpenSim traffic. In Section II analysis of OpenSim applications' network requirements and demands as well as a comparison with TCP congestion control algorithms is discussed.

These measurements and analysis motivate the need for and inform the design of Mongoose the TCP Fair virtual world client discussed in Section III. The evaluation is against standard OpenSim servers and ascertains whether Mongoose is able to determine a TCP Fair rate and to evaluate how quickly it is able to adapt to changes in network conditions. The results are compared with an unmodified client and TFRC.

In Section IV interactions between OpenSim server and Mongoose are investigated through measurement and analysis. This motivates a dual level token bucket design for server traffic regulation. This has been implemented and an evaluation is presented, before demonstrating that the Mongoose Client also achieves TCP Fairness with SL servers.

II. Comparison of TCP and OpenSim

In this section the behaviour of TCP's and OpenSim's[6] congestion control algorithms are compared.

When a user logs onto a virtual world, factors such as the access technology used, length of path and amount of competing traffic mean loss levels and the Round Trip Time (RTT) may vary by several orders of magnitude. If the user is connecting via a local area network the RTTs may be a few milliseconds and the TCP Fair bandwidth maybe many Mbps, whereas on a mobile Internet connection with a poor signal the TCP Fair bandwidth might be only a few tens of kbps.

In order to balance application needs with the available bandwidth the flow of data between client and server can be regulated. In OpenSim the client decides upon rate limits and communicates these to the server. The server uses these limits as inputs to a rate throttling algorithm. The trace of a session on the OpenSim Cathedral Island Fig. 2(a) shows the throughput cap limiting bandwidth usage in the first 200 seconds.

There are three factors that a client uses to decide what rate limits to use: a default value; a user may input a rate; and the system may adjust the rate in response to network conditions. The default user setting is 500 Kbps. This will only be changed if the user explicitly sets a value or if packet loss rises above 3%. The user can specify the throughput by adjusting a slider in the network control dialog box. The minimum setting is 50 Kbps and the maximum is 1500 kbps. The value requested from the server is set to the minimum of 1,500 Kbps and 1.5 times the user limit. Box plots of throughput against user settings in Fig. 2(c) demonstrate that increasing the throttle setting increases throughput.

The loss detection system used in OpenSim clients is a timeout based system that operates at the receiving end of the connection and relies on detecting gaps in the sequence range. In each one second interval:

[6]Second Life clients are open source and may be used to connect to OpenSim servers, there are also third party clients such as Imprudence, Hippo, Phoenix and Firestorm, which provide added capabilities with OpenSim servers. These clients reuse the SL client side traffic management code.

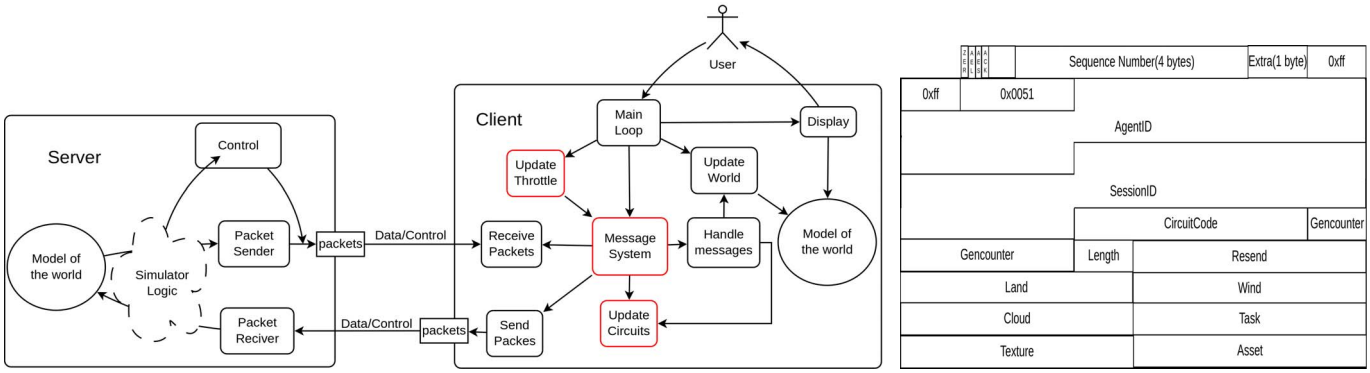


Fig. 4. OpenSim Traffic Control: a) Client requests a bandwidth allocation from the Server. b) Traffic Control Packet with individual channel fields.

- when the loss rate is less than 0.5% and the cap is less than the starting value the rate is increased by 0.1 times the set rate,
- when the loss rate is greater than 3% the sending rate is reduced by 0.1 times the set rate.

This is an Additive Increase Additive Decrease (AIAD) algorithm. Each time the rate is changed by a fixed amount. This does not have the same fairness properties of TCP's AIMD steady state behaviour. The transmission rates of competing streams sharing the same network path, will not converge [11]. The use of a 3% threshold means that loss below this level will not reduce throughput. The transmission rate is independent of RTT, making the algorithm less responsive to short term changes in RTT and meaning that, unlike TCP, throughput is independent of network path length.

The throughput cap is allocated between seven separate channels: Asset, for non physical information about objects; Task, updates the movement and shape of objects and avatars; Texture, includes images that are applied to the surface of objects; Land, Wind, Cloud and Resend, the distribution for different throughput cap levels is shown in Fig. 2(d). Measurements of how bandwidth is allocated between channels during a session are shown in Fig. 2(e). Stacked graphs of total and per channel average, for the activities of standing, walking, flying and teleporting are shown in Fig. 2(f) for three Islands. This illustrates that allocating throughput to each channel ensures that no one channel becomes choked by a surge in traffic from another channel and that different types of elements within the virtual world can be updated in a timely manner.

To summarise OpenSim has the ability to adjust its communication in response to both user input and feedback from the network. There is support for user preferences and isolation of traffic types from each other. However, it is not clear that total throughput is TCP Fair.

A. Comparing the TCP and OpenSim Client Algorithms

Three assumptions were made to allow the analysis to focus on the behaviour of the TCP and OpenSim client algorithms: that the data source is not application limited, loss is deterministic and connections are long lived. These assumptions are the same as those made in [23]. For TCP a simple simulator was written, which takes as input a sequence of packets arranged in a deterministic loss pattern and gives as output the average steady state window size. This simulation was repeated for RTTs from 1 ms up to 500 ms and for packet loss rates from 0.1% to 10%. The throughput for OpenSim was calculated for the same range of parameters [24].

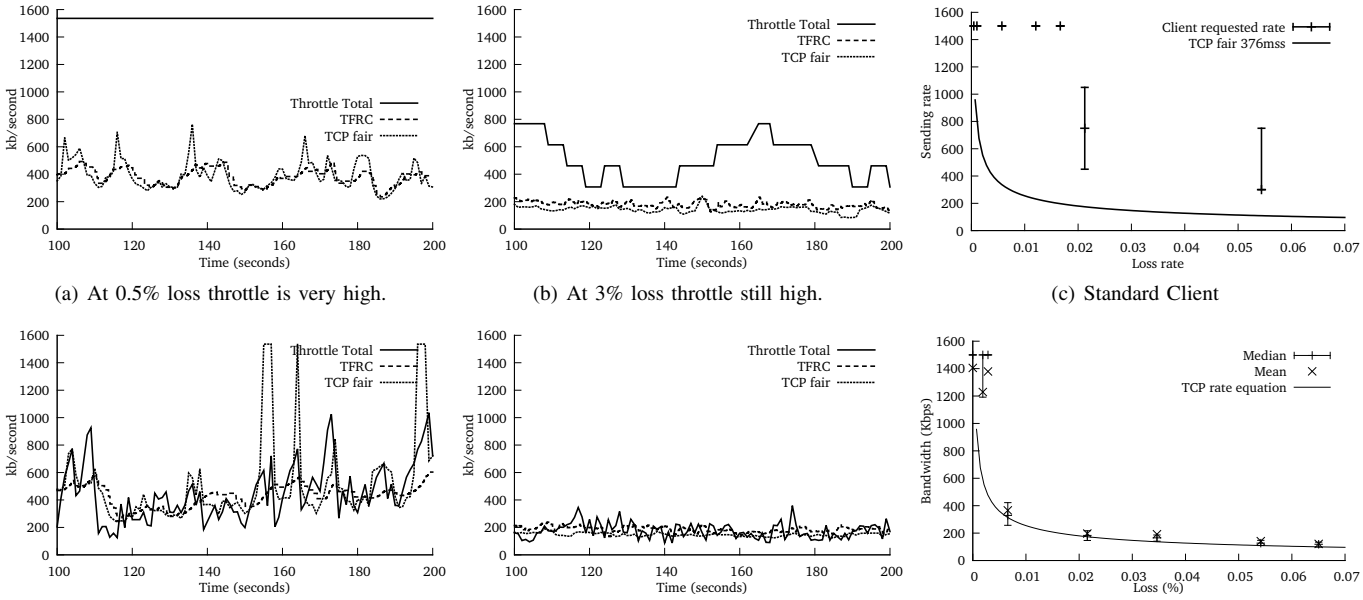
The throughput of OpenSim and TCP for a RTT of 100ms and congestion levels from 0.1% to 10% are shown in Fig. 3(a). The OpenSim user setting is the default 500 Kbps. When congestion is between 0.3% and below 3% the OpenSim algorithm is more aggressive than TCP. When congestion is less than 0.3% it prevents TCP Fair throughput being reached. The throughput against RTT for a loss rate of 1% is shown in Fig. 3(c). Here OpenSim is more aggressive than TCP at RTTs greater than 53ms.

The throughput of OpenSim and TCP, for RTTs up to 500 ms and for loss regimes up to 5%, are compared in Fig. 3. Three lines are shown: the leftmost is for the maximum OpenSim user throughput setting of 1500 Kbps, the middle is for the default setting of 500 Kbps and the line on the right is for the minimum user setting of 50 Kbps. Each line shows where TCP and OpenSim have the same throughput, in the area to the left TCP uses more bandwidth and to the right OpenSim is more aggressive than TCP.

Consider the default user setting. If the packet loss rate never exceeds 3% the throughput cap will remain at 750 Kbps. For an loss rate of 3% and a round RTT of 100ms TCP would allow 150 Kbits more to be sent every second than SL. If the packet loss rate is 4% OpenSim would reduce its sending rate to 50 kbps, significantly lower than the TCP Fair rate. From this analysis it can be concluded it would be desirable to develop a TCP Fair traffic management system appropriate for OpenSim based virtual worlds.

III. The Mongoose Virtual World Client

In the existing OpenSim traffic control system the client calculates a rates for channels within a circuit and sends these to the server, which regulates data flow. Fig. 4(a) shows the system structure.



(a) At 0.5% loss throttle is very high.

(b) At 3% loss throttle still high.

(c) Standard Client

(d) At 0.5% loss Mongoose is TCP Fair.

(e) At 3% loss Mongoose is TCP Fair.

(f) Mongoose Client.

Fig. 5. Client calculated throttle values and TFRC values: Mongoose requests TCP Fair bandwidth across the range of loss regimes.

TCP Fair congestion control can be achieved using rate or window based control. The rate or window size can in turn be determined through an equation based system or through a window tracking system, which reacts directly to notifications of the presence or absence of congestion. TCP itself is a reactive window based system. Streaming media often uses an equation based rate system. An example of a tracking system which uses a similar increase and decrease algorithm to TCP is DCCP Congestion Control IDentifier (CCID) 2 [21]. A congestion window is maintained to control the number of packets that are in the network at any one time.

An equation based congestion control system may calculate the probability of loss so that it can determine the correct window size. In TFRC a weighted average of 8 congestion events is calculated. An example of a rate based formula system is DCCP's CCID 3 [22], a variation of TFRC [13], [20]. It is designed to compete fairly with TCP whilst providing a smooth packet sending rate.

When congestion is low loss events occur infrequently. It is therefore difficult for loss driven equation based systems like TFRC to adjust quickly to a reduction in congestion. Schemes based on tracking changes in latency such as TCP VEGAS [25] and its derivatives are prone to being squeezed by competing TCP Reno traffic in the presence of loss [17]. However, for window tracking schemes based on detecting loss and receiving acknowledgments, packets succesfully received will cause the congestion window to open, prompting a timely recovery.

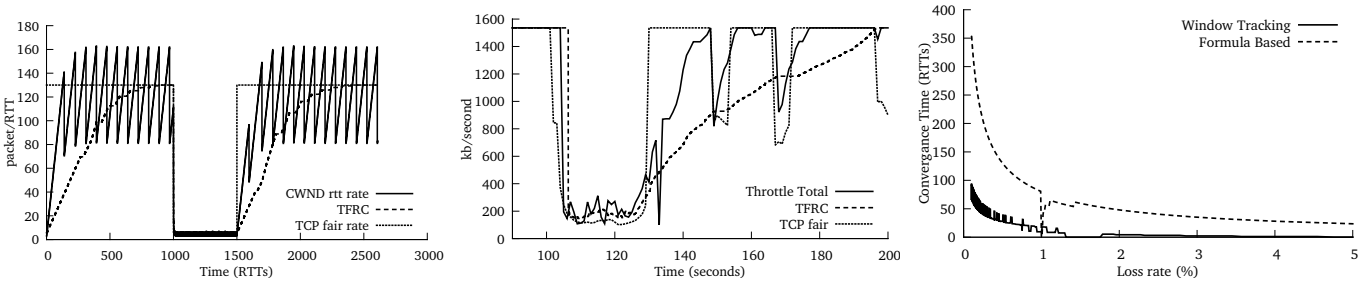
These observations lead to the design of Mongoose as a rate based, window tracking system, maintaining a congestion window at the client and calculating a sending rate from this value using the RTT. A congestion window (CWND) and a slow start threshold (SSTHRESH) are maintained. When loss is detected the CWND and SSTHRESH are reduced to half the CWND. When an in_sequence packet is received the CWND is increased. If the CWND is less than SSTHRESH it is increased by one average packet size. If the CWND is larger than SSTHRESH then it is increased by, the square of the packet size divided by the CWND. The circuit rate is calculated by dividing the CWND by the RTT. The calculated circuit rate is then divided between channels and communicated to the server in a traffic control packet Fig. 4(b).

The detection of loss and changing of the value of the CWND is implemented in Update Throttle, see Fig. 4(a), which is called to process the ID of a packet that has been received. The function performs a check to determine if there are gaps in the ID space. A loss event is confirmed as occurring when three or more packets are received after a gap in the sequence space. If the function determines that the packet is in sequence the CWND is increased. The calculation of the rate is implemented in the function updateDynamicThrottle. The circuit throttle rate is calculated as CWND over RTT. The value is clamped between the upper and lower limits and then checked against the previous value; if it is the same then no new value is sent.

The aim of the following experiments is to evaluate the Mongoose client's ability to determine a fair rate under a range of loss regimes and to compare it against standard client, TFRC and TCP steady state behaviour. The ability of Mongoose to adapt to changes in loss regime is also evaluated and compared with TFRC.

A. Comparison of Mongoose and OpenSim Client Requests

Here the behaviour of Mongoose and standard client algorithms are compared. A series of controlled experiments creating connections to OpenSim servers, where the avatar



(a) 0.01% and 5% Deterministic Simulation. (b) 0.1% to 5% Measured Trace. (c) 1% to multiple loss levels% Measured Trace.

Fig. 6. Time to convergence for Equation and Window Tracking algorithms: Mongoose promptly recovers from congestion episodes.

performs a series of defined actions were conducted. The island used was called the Cathedral Island. The traffic that was received and sent from the client was captured for post processing. The activity undertaken was walking along a set path. Each scenario lasts for ten minutes.

The server traffic goes through two routers, running the netem network emulator [26], before reaching the client machine. The netem setup allows the traffic to be manipulated in both directions. Loss levels of 0.1%, 1%, 3% and 5%, were used and an additional delay of 100 ms was introduced. The data was captured at both sides of the traffic shaping network. This allows the introduced loss rate per unit time to be calculated accurately. The round trip times (RTT) were calculated from the captured traffic. This also allows packet drops to be detected and the loss rate to be calculated. The loss rate value was then run through the TCP rate equation 1 to produce a rate for that level of loss. This allowed the TCP Fair value to be tracked.

The TFRC [13] value was calculated by running through the trace and averaging over the previous 8 loss intervals. A loss interval is the period between the beginning of a loss event and the beginning of the next. Each loss rate was then put into the TCP Fair equation, shown in equation 1, to produce a TFRC rate.

In Fig. 5(a), the loss level is set at 0.5%. Here the bandwidth requested by the standard client remains at the user setting. This is almost four times the TCP Fair amount. The next Fig. 5(b) shows a session where packet loss is set at 3%. The bandwidth requested by the client has been significantly reduced, showing client congestion control being active. However, the bandwidth requests are still higher than a TCP Fair value. The third graph in Fig. 5(c) shows box plots for six sessions at a range of congestion levels. For these scenarios the standard client consistently asks for more bandwidth than is TCP Fair. At the lowest loss level it is less aggressive than TCP.

Next consider graphs illustrating the behaviour of Mongoose. In Fig. 5(d) and Fig. 5(e) the bandwidth requested fluctuates around the TCP Fair level. This result is confirmed in Fig. 5(f) where boxplots for a range of sessions are shown along with a guideline for TCP Fair behaviour. The requests follow the TCP Fair behaviour closely. This evaluation demonstrates that the Mongoose client is able to consistently determine a TCP Fair rate, and to request this rate from the server.

B. Adapting to Changes in Network Conditions

In comparing the behaviour of Mongoose and TFRC, how quickly the algorithm adapts to the changes in network conditions is considered. Mongoose is evaluated by comparing the throughput it requests with that requested by the TFRC algorithm and the TCP Fair rate.

A simulation of the scenario where the congestion level periodically switches between 0.01% and 5% is shown in Fig. 6(a). In this scenario there is little difference in convergence when there is an increase in congestion, both Mongoose and TFRC adjust quickly. When there is a reduction in congestion, TFRC takes a long time to utilise the increase in fair bandwidth that is available. The Window Tracking algorithm is able to adjust significantly faster.

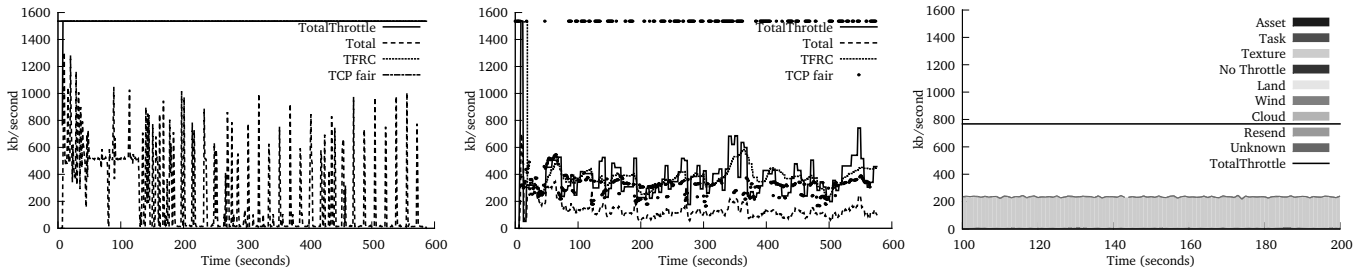
A trace of measured traffic is shown in Fig. 6(b). Here congestion starts at a low level of 0.1%, packet loss is then sharply increased to 5%. After a while packet loss is again reduced to 0.1%. It can be observed that again both TFRC and Window Tracking are able to adjust quickly to the increase in packet loss. As in the simulation, Window Tracking is quicker at adjusting to the reduction in loss.

The results of multiple simulations are shown in Fig. 6(c). Each point represents the time it takes to converge from 1% packet loss to a new congestion level. Lines for Window Tracking and TFRC algorithms are shown. This further demonstrates that TFRC is slow to adjust to decreases in loss.

For streaming applications a relatively smooth bandwidth allocation and consistent quality of service are appropriate [27], TFRC fits with these requirements. However, OpenSim traffic contains several components. Avatar control traffic has small and consistent bandwidth requirements. Textures and object descriptions benefit from being delivered as soon as possible. Consequently, if bandwidth is available to OpenSim, the application will often benefit from utilising it. The ability of the Window Tracking to quickly recover from periods of congestion is in this context an advantage and justifies its use in Mongoose.

IV. The Mongoose Virtual World Server

This section looks at server side aspects of virtual world network usage: it first discusses how an OpenSim server



(a) 0% loss: throughput is application limited. (b) 0.5% loss: circuit allocation is not fully utilised. (c) The channel bandwidth stacked.

Fig. 7. The bandwidth usage and throttle values: a circuit's allocation is underused because unneeded bandwidth is not reallocated between channels.

responds to bandwidth requests from a client, this discussion is supported both by measurements and through inspection of source code. Next the design of a traffic control system is proposed, which enables individual channel's bandwidth to be protected and the TCP Fair allocation to be fully utilised. The design has been implemented and its evaluation is the final topic in this section.

An OpenSim server creates a circuit for each client it is connected to. As described above each client determines a global throughput limit, and allocates this between channels. The client periodically sends a Traffic Control Packet, containing the channel throughput throttle values to the server. The server maintains a separate token bucket for each channel. Tokens accumulate at the rate of the bandwidth requested by the latest Traffic Control Packet. When a data packet is ready to send, the server checks the credit accumulated in the token bucket for the packet's channel. If there is sufficient credit the packet is sent, otherwise it is queued until there is sufficient credit. This system has the benefit of protecting the throughput available to individual channels. However, if one channel has unused capacity it is not reallocated to another that may have excess demand. Consequently, it is possible for extra delays to be introduced in servicing client requests while much of the circuit's throughput allocation is unused. When a user logs on, or an avatar teleports to a new region, it will often be the case that the task channel allocation is under used whilst there are lots of textures queued for download. This increases the time the user has to wait before they can see their environment and therefore reduces the usability of the system.

Two sets of experiments were conducted with the Mongoose client and a standard OpenSim server. In the first the throughput achieved for loss levels, between 0% and 10%, were measured and compared to the circuit throttle and TCP Fair throughputs. This allows evaluation of whether in practice the absence of a circuit throughput reallocation mechanism impacts on global throughput. The second set of experiments measured the bandwidth allocated to individual channels.

The values for a session where the loss rate is set to 0% are shown in Fig. 7(a): the mean bandwidth is 218 Kbps, the Mongoose client is calculating a throttle value greater than the maximum throttle of 1500 kbps and the throughput is always well below the throttle value. The loss rate is set to 0.5% in Fig. 7(b). The loss is being detected and the client is reacting to it. The mean bandwidth is 139 Kbps. The circuit throttle value is close to the TCP Fair values. The throughput is less than both the circuit throttle value and the throughput when loss was 0%. The traffic of a trace split into its component channels is shown in Fig. 7(c). Most of the traffic is textures, which reaches but does not exceed the channel throttle. The other channels have very little traffic, and their allocation remains unused.

These measurements verify that the OpenSim throttle system does protect the throughput of individual channels but often prevents the circuit throttle throughput being fully used.

A. Mongoose Server Design

The design goals are threefold: that the server limit its traffic to its TCP Fair share, that it be able to use all of its TCP Fair share when it has sufficient data to send and that each channel's allocated share is protected.

The design is illustrated in Fig. 8. A two level token bucket system is used. There is a token bucket for each channel and a parent bucket that limits the overall throughput for the circuit. The tokens for each channel bucket accumulate at the channel throttle rate and the parent bucket's tokens accumulate at the circuit throttle rate. There is a burst rate value for each bucket which limits the number of unused tokens that can accumulate and therefore the burst size. When a packet is ready to send the parent bucket's token credits are checked. If there are enough credits the packet is sent out otherwise it is queued.

The channel level buckets only limit transmission when there is a queue. The token buckets then distribute the bandwidth between channels. The system iterates through the channels in a round robin fashion. When there are sufficient circuit channel credits for a packet to be sent the next channel is checked. If the channel has a packet queued and sufficient channel credits have accumulated the next packet from that channel is sent, otherwise the same algorithm is applied to the next channel. With this approach when there is sufficient demand throughput reaches the global throttle limit and each channel's allocation is protected. Consequently, a request to download lots of textures can be met quickly without interfering with the responsiveness of avatar controls.

B. Mongoose Server Evaluation.

In this section the Mongoose server is evaluated. The experiments with Mongoose were performed using the same

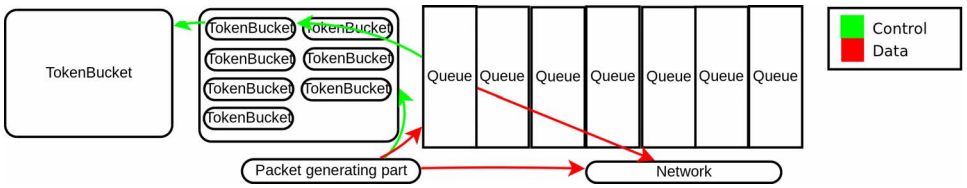
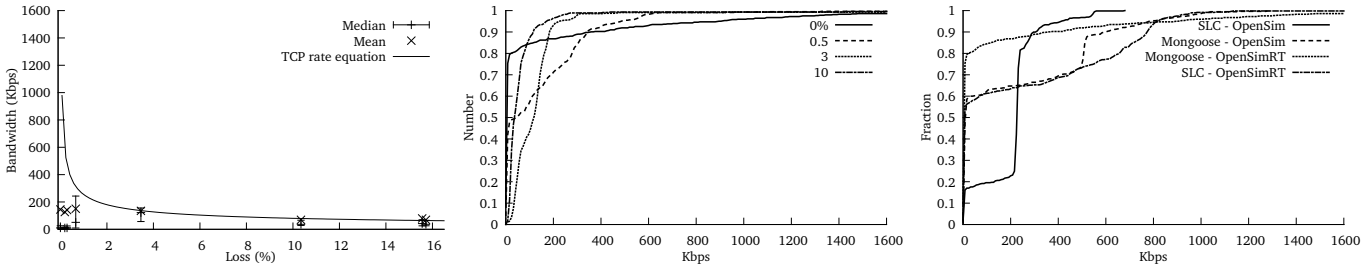


Fig. 8. Dual level token buckets in the Mongoose Server, showing the relationship between buckets and queues.



(a) Throughput comparison with TCP. (b) CDF for Range of Loss Regimes. (c) Comparison of Client Server Combinations.

Fig. 9. Measured throttle values and bandwidth usage against loss rate for the Mongoose Client and Server.

infrastructure as the unmodified OpenSim server. The traffic was directed over a test network that allowed the loss and RTT to be controlled. The traffic was captured at both ends of the network. A series of experiments were conducted which show that like the unmodified OpenSim server when bandwidth allocation is limited the Mongoose client distributes bandwidth between channels. The measurements discussed below show that unlike the unmodified OpenSim server Mongoose is able to make use of, and not exceed the client's (TCP Fair) throughput requests.

The throughput values against loss rates are shown in Fig. 9(a). The x-axis shows loss and the y-axis shows bandwidth. The guide line shows the TCP Fair equation for a RTT of 150 ms and a packet size of 375 bytes (the measured OpenSim mean packet size) . The mean bandwidth usage does not change significantly between 0% loss and 3% loss, the median bandwidth is however much smaller at 0% loss. The low loss level traces have lower median bandwidth and higher bursts in traffic. The shorter bursts mean that when a texture is requested it is transferred quickly. The small difference between the mean, median and quartiles at higher loss levels shows that the traffic is less bursty and is limited by the throttle system rather than the application. The usage at medium to high loss levels is TCP Fair. The usage at low loss levels is below the TCP Fair value as it is application limited.

The next two graphs show cumulative throughput distributions. A comparison of the four combinations possible of OpenSim server, Mongoose server (OpenSim TR), Mongoose client and SL client is shown in Fig. 9(c) with 0% packet loss. This demonstrates that the Mongoose Server and Client combination makes more efficient use of network resource. Throughput levels are low for 80% of the time, and there are short bursts of high throughput. Thus available network resource is best utilised and the user receives a more responsive experience.

The cumulative distribution frequency of the bandwidth for different loss levels is shown in Fig. 9(b). Values for 0%, 0.5%, 3% and 10% loss are shown. At low loss levels the bandwidth level is mainly low but there are a small proportion of high bandwidth bursts. At medium and high loss levels there are more samples in the middle, due to server imposed rate limitation, and the bandwidth is more stable. Thus traffic produced by the Mongoose server and Mongoose client is bursty at low loss levels, when the network is not heavily loaded and can handle this type of traffic. When the network is under heavy load traffic is TCP Fair and less bursty. It does not put undue stress on the network. The combination of the Mongoose client and Mongoose server does the right thing at low, medium and high loss levels.

C. Mongoose with Second Life Servers

Here the bandwidth that a Mongoose client receives from SL servers is discussed. A scatter plot of readings, showing the throughput against TCP fair values, are given in Fig. 10(a), where each point corresponds to one second of activity. The next graph 10(b), shows boxplots for a number of sessions. They show a good correspondence between TCP Fair behaviour and actual throughput. At low levels of loss the bandwidth utilised dips significantly below TCP Fair levels. Here the effect of application limitation is seen. The final graph 10(c) plots throughput against loss with a TCP Fair guideline. Boxplots for the throughput are shown. These show a good correspondence between the TCP Fair guideline and Mongoose traffic and demonstrate that Mongoose is TCP Fair when used with SL servers.

V. Conclusion

Virtual worlds offer the potential of providing engaging educational experiences and addressing important social priorities such as promoting cultural heritage. Open virtual worlds

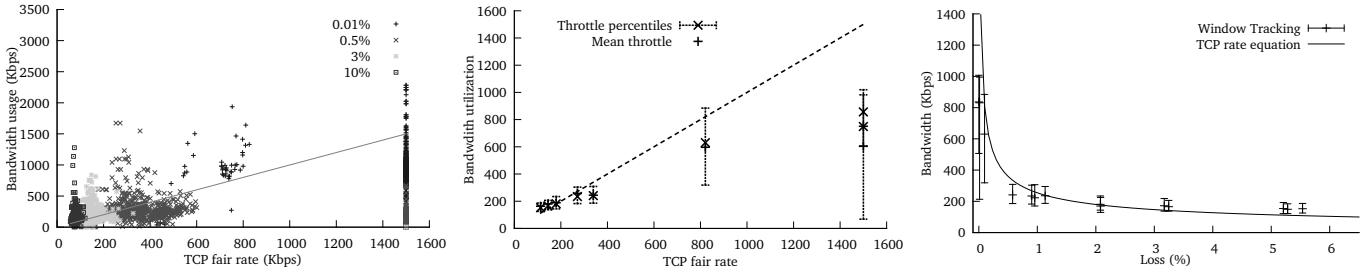


Fig. 10. Mongoose with SL servers: a) per second readings. b) per connection box plots. c) TCP Fair behaviour.

such as OpenSim enable the cost efficient deployment of these applications as part of the emerging 3D Internet. In deploying new applications there is a responsibility to study their network traffic so that impact on the Internet can be understood.

Measurement based congestion control that uses a rate based tracking algorithm can achieve TCP Fairness, whilst improving the responsiveness of virtual worlds. Furthermore virtual world traffic is decomposed into categories, which have distinct characteristics and requirements from the network. A dual layer token bucket system has been implemented and shown to: regulate the overall rate within TCP Fair bound; provide guarantees to specific channels; and redistribute unused bandwidth. It provides protection to delay sensitive Avatar Control Traffic without unduly delaying the download of textures.

We conclude that the differential QoS aware congestion control outlined in this paper meets the specific needs of virtual world applications and is fair to competing traffic. The Mongoose Server and Client, traffic traces and virtual world environments discussed in this paper are all accessible via: http://openvirtualworlds.org/.

REFERENCES

[1] C. Allison, A. Miller, T. Sturgeon, J. Nicoll, and I. Perera, "Educationally enhanced virtual worlds," in *Frontiers in Education Conference (FIE), 2010 IEEE*, oct. 2010, pp. T4F–1 –T4F–6.

[2] K. Getchell, A. Miller, J. R. Nicoll, R. Sweetman, and C. Allison, "Games methodologies and immersive environments for virtual fieldwork." *IEEE Transactions on Learning Technologies*, 2010.

[3] I. A. Oliver, A. H. Miller, and C. Allison, "Virtual worlds, real traffic: interaction and adaptation," in *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, 2010.

[4] A. Sanatinia, I. A. Oliver, A. H. D. Miller, and C. Allison, "Virtual machines for virtual worlds," in *2nd International Conference on Cloud Computing and Services Science, CLOSER 2012*, 2012.

[5] J. Kinicki and M. Claypool, "Traffic Analysis of Avatars in Second Life," in *Proceedings of the 18th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Braunschweig, Germany, May 2008.

[6] S. Fernandes, F. Antonello, J. Moreira, D. Sadok, and C. Kamienski, "Traffic Analysis Beyond This World: The Case of Second Life." in *7th International workshop on Network and Operating Systems Support for Digital Audio & Video (NOSSDAV) 2007*, 2007.

[7] T. Henderson and S. Bhatti, "Networked games: a QoS-sensitive application for QoS-insensitive users?" in *RIPQoS '03: Proceedings of the ACM SIGCOMM workshop on Revisiting IP QoS*. New York, NY, USA: ACM, 2003, pp. 141–147.

[8] K. Chen, P. Huang, C. Huang, and C. Lei, "Game traffic analysis: an MMORPG perspective," in *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM, 2005.

[9] V. Jacobson, "Congestion Avoidance and Control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 314–329, 1988.

[10] R. Jain, D. M. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared systems," Digital Equipment Corporation, Littleton, MA, Tech. Rep. DEC TR-301, 1984.

[11] K. K. Ramakrishnan and R. Jain, "A binary feedback scheme for congestion avoidance in computer networks," *ACM Trans. Comput. Syst.*, vol. 8, no. 2, pp. 158–181, 1988.

[12] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP Congestion Avoidance algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 3, pp. 67–82, 1997.

[13] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," RFC 3448 (Proposed Standard), Jan. 2003.

[14] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," in *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 4, ACM. Amherst, MA, USA: University of Massachusetts, 1998, pp. 303–314.

[15] D. Bansal and H. Balakrishnan, "Binomial congestion control algorithms," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, 2001, pp. 631–640.

[16] H. Balakrishnan and S. Seshan, "The Congestion Manager," RFC 3124 (Proposed Standard), Jun. 2001.

[17] K. Tan and J. Song, "Compound TCP: A scalable and TCP-Friendly congestion control for high-speed networks," in *in 4th International workshop on Protocols for Fast Long-Distance Networks (PFLDNet), 2006*, 2006.

[18] S. Ha, I. Rhee, and L. Xu, "Cubic: a new TCP-Friendly high-speed TCP variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.

[19] I. Abdeljaouad, H. Rachidi, S. Fernandes, and A. Karmouch, "Performance analysis of modern TCP variants: A comparison of Cubic, Compound and New Reno," in *Communications (QBSC)*, 2010.

[20] F. Nivor, "Experimental study of DCCP for multimedia applications," in *Proceedings of the 2005 ACM conference on Emerging network experiment and technology*, ser. CoNEXT '05. New York, NY, USA: ACM, 2005, pp. 272–273.

[21] S. Floyd and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control," RFC 4341 (Proposed Standard), Mar. 2006.

[22] S. Floyd, E. Kohler, and J. Padhye, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)," RFC 4342 (Proposed Standard), Mar. 2006.

[23] S. Floyd, M. H, and J. Padhye, "A comparison of equation-based and AIMD congestion control," ACIRI, Tech. Rep., May 2000.

[24] A. Ruddle, C. Allison, and P. Lindsay, "Visualising interactions between TCP's congestion and flow control algorithms," in *Computer Communications and Networks, 2002. Proceedings. Eleventh International Conference on*, Oct. 2002, pp. 34–38.

[25] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: new techniques for congestion detection and avoidance," *SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 4, pp. 24–35, oct 1994.

[26] S. Hemminger, "Network emulation with NetEm," in *Linux Conf Au*, 2005, pp. 18–23.

[27] A. Bouch, A. Kuchinsky, and N. Bhatti, "Quality is in the eye of the beholder: meeting users' requirements for internet quality of service," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2000.