Some Thoughts on Emulating Jitter for User Experience Trials

Grenville Armitage
Centre for Advanced Internet Architectures
Swinburne University of Technology
Melbourne, Australia
garmitage@swin.edu.au

Lawrence Stewart
Centre for Advanced Internet Architectures
Swinburne University of Technology
Melbourne, Australia
lastewart@swin.edu.au

ABSTRACT
It is usually hard to control the network conditions affecting public online game servers when studying the impact of latency, loss and jitter on user experience. This leads to a natural desire for running user-experience trials under controlled network conditions, and hence a requirement for accurate (or at least predictable) emulation of IP level latency, loss and jitter on a localized network testbed. In this short paper we reflect on some experiences with running user-experience trials, and specifically evaluate the utility and limitations of using FreeBSD's kernel-resident dummynet module to introduce controlled jitter. We expect these insights will stimulate further user-experience trials built around low-cost, unix-based networking tools.

Categories and Subject Descriptors
C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design – *network communications, packet-switcing networks*
C.2.3 [**Computer-Communication Networks**]: Network Operations – *network monitoring*

General Terms: Measurement, Documentation, Experimentation, Human Factors, Verification.

Keywords: Online, Games, Jitter, Latency, Internet.

1. INTRODUCTION
Highly interactive, real-time online games represent both a challenging consumer-driven requirement (creating demand for broad-band access and predictable network performance) for Internet Service Providers (ISPs) and a potential market for 'premium' services (read 'higher fee for predictable network characteristics'). As a consequence, there has been an increase in the number of research projects attempting to quantify the impact of game traffic on networks [3,4,5,8,9] and network conditions (such as path latency, packet loss and path jitter) on the game player's online experience [1,2,6,11]. Understanding the qualitative and quantitative relationships between user experience and network characteristics helps ISPs develop suitable network engineering guidelines, and helps future game server operators optimally locate and provision their services around the Internet.

In order to observe 'realistic' game playing conditions, many of us have focussed on instrumenting public, online game servers and gathering data about user behavior and concurrent network conditions. From these data sets – often covering hundreds to thousands of played games, and hundreds of self-selected client sites around the Internet - we draw inferences from observed correlations between things like median latency and a player's enjoyment of a particular online game.

Unfortunately, it is rather difficult to control the network conditions affecting players on public servers, and it is hard (if not impossible) to know the types of players who choose to frequent your servers. This leads to a natural desire for running user-experience trials under circumstances where the network conditions and players themsevles can be accurately classified (and controlled, in the case of the network). This leads to a requirement for accurate (or at least predictable) emulation of IP level latency, loss and jitter on a localized network testbed. In this paper we reflect on some of our experiences with running user-experience trials, and specifically evaluate the utility and limitations of using FreeBSD's kernel-resident dummynet module [10] to introduce controlled jitter through FreeBSD-based Ethernet bridges. Our trials were run using FreeBSD 4.9, the current production release of FreeBSD at the time. We hope these insights will stimulate further user-experience trials built around low-cost, unix-based network tools, and lead to a far greater body of knowledge in this industry sector.

1. USER EXPERIENCE TRIALS
A good user-experience trial will typically involve a number of PCs or game consoles set up in a known network configuration, a,nd with a user/player community selected for their willingness ,to stay around for the duration,, of the trials. Despite the attraction of jumping straight into designing the network testbed, it is worth taking a moment to reflect on the human aspect of a good user-experience trial.

Because user experience fluctuates from minute to minute and hour to hour even with the same person, a statistically valid user-experience trial must repeat each set of game-play/player/network-condition combinations multiple times. Because experience also varies between individuals you also need to ensure a reasonably large sample space of players – more like 10, 20 or 30+ players rather than two or three of your close friends.

In late 2003 we ran user experience trials at our University research centre for the XBox game 'Halo' [7] and the PC game Quake3 (results unpublished). For the Halo trials we tried to establish four groups of eight players who would volunteer to play Halo for two hours a night, one night a week for four weeks. As it turned out, we could get volunteers for only two groups, and suffered attrition as RealLife(tm) intruded into people's evening schedules. For the Quake3 trials we arranged for a group of 8 players to dedicate most of a single Saturday to game play. It would be fair to say that each player's mental state was significantly compromised by the end of such a long day.

The primary lesson here is that we need to be careful and creative when establishing a pool of players whose performance and consistency is statistically valid. Rather than discuss how we might achieve these goals, the rest of this short paper focuses on the other aspect of user-experience trials – emulating poor network conditions.

2. CONTROLLING NETWORK CHARACTERISTICS

FreeBSD is a stable, widely used open source unix operating system, freely available much like any good Linux distribution and the other 'BSD' projects (NetBSD and OpenBSD). FreeBSD includes an extremely useful kernel module known as `dummynet` [10], enabling the addition of configurable latency, bandwith limits and packet loss to traffic flowing through the kernel's network stack. From the command line of a FreeBSD-based Ethernet bridge between client hosts and a game server, dummynet allows us to repeatably emulate a wide range of latency and packet loss scenarios.

One of our colleagues demonstrated dummynet to be reasonably precise as a latency and loss rate tool [12] (to within one millisecond once the FreeBSD kernel is recompiled with its internal 'tick' counter set to 1000 Hz rather than its default of 100Hz). Dummynet is controlled as an extension to FreeBSD's internal `ipfw` packet filter, and operates on traffic flows identified as 'pipes'. Without going into the details of how pipes are allocated (some instructions can be found in Appendix 1 of [12]) one can configure particular latency or loss with commands like:

`ipfw pipe 1 config delay 10ms` (to instantiate an additional latency of 10 milliseconds)

or

`ipfw pipe 1 config plr 0.15` (to instantiate a loss rate of 15%)

However, dummynet does not directly support configurable jitter. To create jitter we need to dynamically and continuously vary the dummynet pipe rule to cause fluctuations in the configured latency. Our goal for this paper is to characterise dummynet's efficacy and consequences when used in such a manner, and demonstrate that it can thus act as a suitably predictable and controllable source of client-server jitter.

(An alternative tool for network testbeds using Linux-based routers is NIST Net [13], maintained up to and including the Linux 2.4 kernel series. Contrasting FreeBSD's dummynet/ipfw with NIST Net for introducing controlled jitter is the subject of ongoing work.)

3. CREATING JITTER

It is not particularly difficult to write a small program in C or C++ that generates a pseudo-random stream of integer values, which are then used to update a dummynet pipe at regular intervals (using 'ipfw pipe'). However, our experiments show that you should choose your random distribution carefully, keeping in mind how it can interact with your pipe re-configuration interval. Simply fluctuating the dummynet pipe according to a uniform random distribution can create unexpectedly non-uniform distributions of actual latency through your bridge.

3.1 Jitter Constraints

To test the actual jitter created along our end-to-end path, we ran high speed 'ping' trials from one machine to another through the dummynet-equipped bridge. The standard FreeBSD 4.9 implementation of ping was used to generate a stream of ICMP Echo Request packets every 5ms, 10ms, 25ms and 50ms apart – essentially sampling the path's roundtrip time at those intervals.

Each test involved many tens of thousands of pings while the bridge's dummynet pipe rule was reconfigured every X milliseconds, for X = 100, 500, 1000, and 2000. This was intended to emulate moderately fast to slow fluctuations in path latency. Since we ran all bridge traffic through dummynet's pipe in both directions, the round trip time (RTT) estimate by ping would be twice the delay configured into the dummynet pipe at any given time. Configured latencies ranged from 28ms to 52ms with a mean of 40ms, creating a range of RTTs from 56ms to 104ms with a mean of 80ms.

We decided not to explore jittering intervals less than 100ms for now, even though jitter induced by real network congestion events can have much shorter intervals. Many fast-paced online games only send server to client updates every 40 to 100ms (e.g. 40ms for XBox Halo on a LAN, 50ms for Quake3, 60ms for Half-Life and 100ms for older games such as Quake2). In other words, the client-server network path is sampled every 40-100ms, constraining the player's perception of how frequently the path latency changes.

For all trials we used a uniform distribution - the bridge would spend roughly an equal amount of time configured for each possible latency value between the specified upper and lower bounds. The actual distribution was random and uniform for all but the last three trials. The last three trials were stimulated by uniform linear/ramp distributions.

Finally, we also confirmed that our jittering scheme did not induce any re-ordering of the packets flowing through a given dummynet pipe.

3.2 Trial Results using Random Uniform Distributions

Figure 1 shows the resulting distribution of RTTs, measured over 300000 ping packets, with the bridge jittering every 2000ms (i.e. picking a new, random latency value between 28ms and 52ms every 2 seconds). Ping packets were generated every 50ms. The distribution looks reasonably uniform. Using 51000 ping packets, Figure 2 shows a similar set of distributions with the bridge jittering every 1000ms, and Figure 3 shows the same set of trials with the bridge jittering every 500ms. The results still look reasonably uniformly distributed.

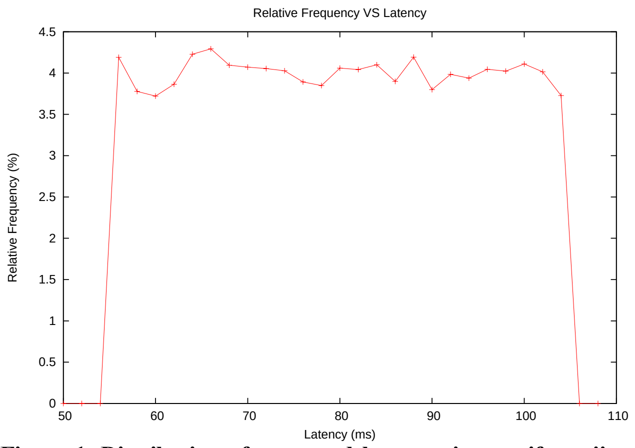

Figure 1. Distribution of measured latency given uniform jitter every 2000ms

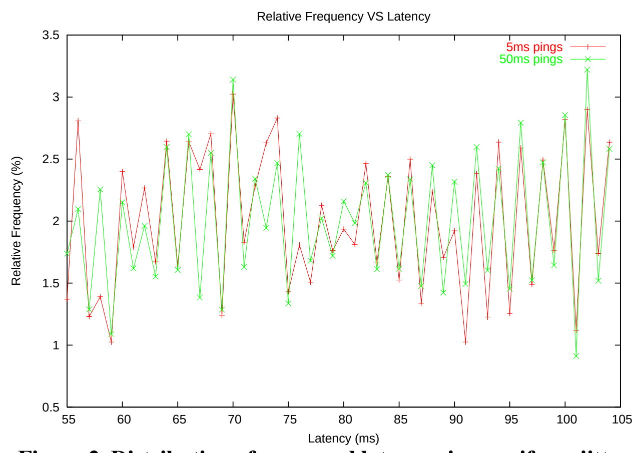

Figure 2. Distribution of measured latency given uniform jitter every 1000ms

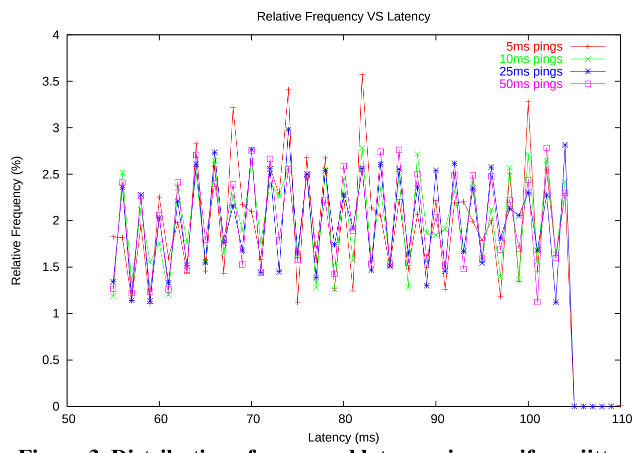

Figure 3. Distribution of measured latency given uniform jitter every 500ms

However, things begin to get rather interesting in Figure 4, where the bridge is now jittering every 100ms. The actual effective distribution is anything but uniform, with a strong peak around the centre of the expected RTT range. Arguably this is not a good situation if one is trying to emulate high speed jittering with a uniform distribution.

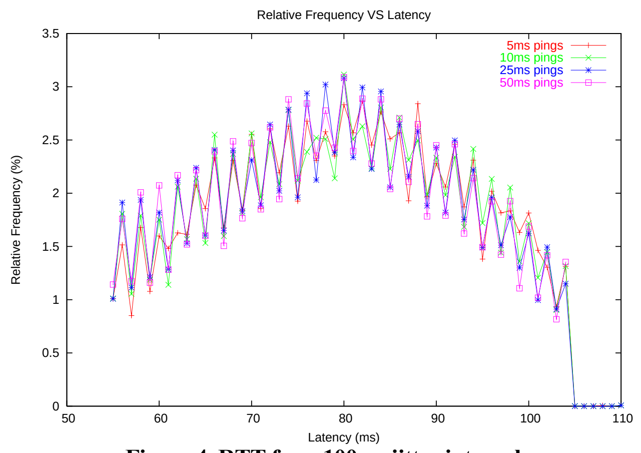

Figure 4. RTT for a 100ms jitter interval

Further investigation suggests that this skewing of the measured RTT distribution is a consequence of the jittering interval being a close fraction of the actual latency configured into the bridge. Figure 5 shows how the skewed distribution (given 100ms jitter interval on the bridge) becomes more pronounced for higher average RTT (i.e. when the bridge's latency is being jittered around a larger mean value).

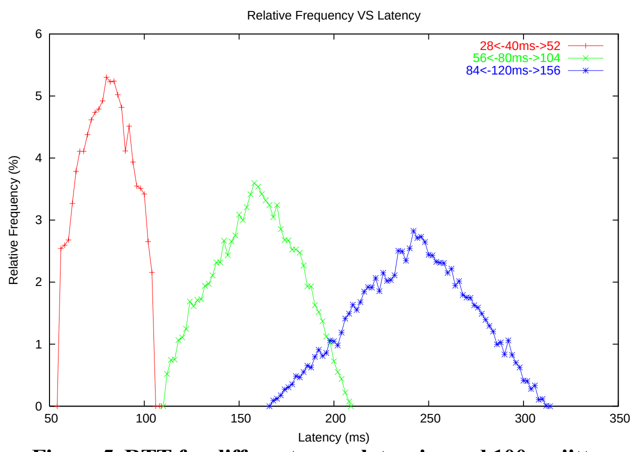

Figure 5. RTT for different mean latencies and 100ms jitter interval

This unusual skewing appears to be related to the particular random sequence of dummynet latency settings. If the pipe's latency is being changed relatively frequently (compared to the average configured latency of the pipe itself) there's an increasing chance of one or more packets being in the pipe itself each time dummynet's pipe rule is reconfigured. This skews the effective RTT – packets that were in the pipe when the pipe's delay is shortened experience an actual latency somewhere between the previous pipe delay and the new pipe delay.

3.3 Trials using Non-random Distributions

The skewing of RTTs is quite dramatically demonstrated in Figure 6. We alternated dummynet between two specific latencies (28ms and 52ms) every 100ms, and probed the path with 51000 ping packets. There are two spikes in the RTT histogram corresponding to the two actual latencies (at 56ms and 104ms), and another spike halfway between these two values (at 80ms). If you stood back and superimposed a number of these graphs on top of each other (as would be the case with a uniform random distribution of possible dummynet latency transitions) you begin to see how the RTT distribution in Figure 4 comes about.

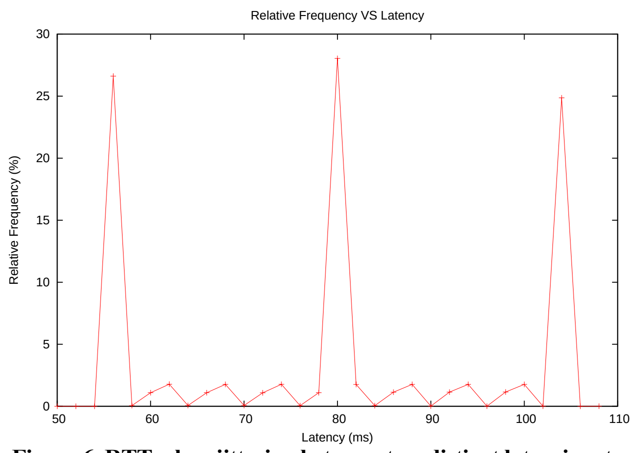

Figure 6. RTT when jittering between two distinct latencies at a 100ms jitter interval

In order to test this theory further, we re-ran our trials using three different linear 'ramp' distributions and show the results in Figure 7. The distributions are marked '52->28' (24 one-millisecond steps from

52ms down to 28ms, then repeat from 52ms), '28->52' (24 one-millisecond steps from 28ms up to 52ms, then repeat from 28ms),and '28->52->28' (48 one-millisecond steps from 28ms up to 52ms then back down to 28ms again).

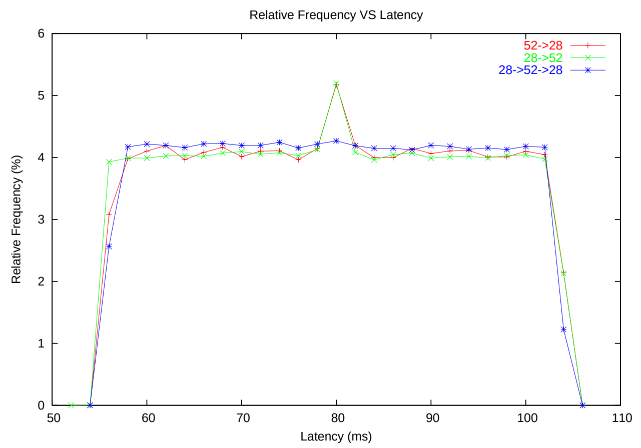

Figure 7. RTT when jittering with ramp-distributions and a 100ms jitter interval

These ramps are also uniform distributions. The key difference is that each transition from one dummynet latency to the next is only one millisecond (except for one transition at the end of each 24 step cycle for '52->28' and '28->52'). Figure 7 is quite different to Figure 4, even though both were generated by jittering dummynet every 100ms. The '28->52->28' jittering function results in quite a smooth distribution of RTT. The '52->28' and '28->52' jittering functions create a mostly smooth distribution of RTT, with a small spike around 80ms corresponding to the large jumps from 28ms to 52ms (or vice-versa) every 24 jittering intervals (every 2.4 seconds).

4. CONCLUSION

This paper touches on some qualitative issues associated with running user-experience trials, and illustrates some unexpected results when emulating jitter using FreeBSD's kernel-resident dummynet module. If you 'jitter' the dummynet pipe at intervals that approach the pipe's mean latency the actual resulting distribution of latencies will be skewed and cluster more around the mean value even though your driving distribution is uniform. Fast jittering might be better performed using non-uniform (pre-skewed) distributions of latency values. For predictable, smooth jitter your jittering intervals should be at least ten times longer than your mean network latency. Although this would appear to limit one's ability to emulate jitter occurring at very short intervals (e.g. sub-50ms) this shouldn't be a great limitation as many interactive games typically don't 'sample' the network more frequently than every 40-100ms anyway.

5. REFERENCES

[1] Armitage, G., Stewart, L., "Limitations of using Real-World, Public Servers to Estimate Jitter Tolerance Of First Person Shooter Games", ACM SIGCHI ACE2004 conference, Singapore, June 2004

[2] Armitage, G., Stewart, L., "Limitations of using Real-World, Public Servers to Estimate Jitter Tolerance Of First Person Shooter Games", ACM SIGCHI ACE2004 conference, Singapore, June 2004

[3] Borella, M. S., "Source models of network game traffic", proceedings of networld+interop '99, Las Vegas, NV, May 1999

[4] Faerber, J., "Network game traffic modelling", proceedings of the 1st ACM workshop on Network and System Support for games, April 2002

[5] Feng, W., Chang, F., Feng, W., Walpole, J., "Provisioning\On-line Games: A Traffic Analysis of a Busy Counter-Strike Server", SIGCOMM Internet Measurement Workshop, November 2002

[6] Henderson, T., "Latency and user behaviour on a multiplayer games server", proceedings of NGC 2001, London, UK, pp1-13, November 2001

[7] Lang,T., "User Experience while playing Halo with network delay or loss", Technical Report CAIA-TR-031205A, Centre for Advanced Internet Architectures, Swinburne University of Technology, December 2003 (http://caia.swin.edu.au/reports/031205A/CAIA-TR-031205A.pdf)

[8] Lang, T., Armitage, G., Branch, P. and Choo, H., "A Synthetic Traffic Model for Half-Life", Australian Telecommunications Networks & Applications Conference 2003, Melbourne (ATNAC 2003), Australia, December 2003

[9] Lang, T., Branch, P., Armitage, G. J., "A Synthetic Traffic Model for Quake 3", (accepted for publication) ACM SIGCHI ACE2004 conference, Singapore, June 2004

[10] Rizzo, L., "Dummynet: a simple approach to the evaluation of network protocols", ACM Computer Communication Review, Vol. 27, No.1, pp.31-41, January 1997

[11] Sheldon, N., E. Girard, S. Borg, M. Claypool and E. Agu, "The Effect of Latency on User Performance in Warcraft III", Technical Report WPICS- TR-03-07, Computer Science Department, Worcester Polytechnic Institute, March 2003

[12] Vanhonacker, W. A., "Evaluation of the FreeBSD dummynet network performance simulation tool on a Pentium 4-based Ethernet Bridge", Technical Report CAIA-TR-031202A, Centre for Advanced Internet Architectures, Swinburne University of Technology, December 2003 (http://caia.swin.edu.au/reports/031202A/CAIA-TR-031202A.pdf)

[13] "NIST Net home page," National Institute of Standards and Technology, USA, http://snad.ncsl.nist.gov/itg/nistnet/ (as of June 2004)