Mathematical and Experimental Analysis of Limitations in Creating Artificial Jitter for Networked Usability Trials

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

Philip Branch
Centre for Advanced Internet Architectures
Swinburne University of Technology
Melbourne, Australia
pbranch@swin.edu.au

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

3rd Workshop on the Internet, Telecommunications, and Signal Processing (WITSP'04), Adelaide, Australia, December 20-22 2004

ABSTRACT

Controlled usability trials are frequently desirable when assessing the impact of network latency, loss and jitter on highly interactive networked applications such as online games. This leads to a requirement for accurate (or at least predictable) emulation of IP level latency, loss and jitter on a localized network testbed. This paper reflects on the mathematical and experimental insights we gained when developing a controlled jitter network environment. We used FreeBSD's kernel-resident dummynet module to introduce controlled jitter, but our results generalize to other tools that can introduce dynamically variable delays in an IP packet path. We expect these insights will stimulate further user-experience trials built around low-cost, unix-based networking tools.

Keywords: Jitter, Emulation, Latency, Internet, Online Games, Usability Trials.

1. INTRODUCTION

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]. This research is driven largely by the fact that 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'). 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.

One research approach has been to observe 'real life' game playing conditions – monitoring user behavior and concurrent network conditions from public, online game servers. The resulting data sets often cover hundreds to thousands of played games, and hundreds of self-selected client sites around the Internet. Causal links are then inferred from observed correlations between attributes such as median latency and a player's enjoyment of a particular online game.

Unfortunately, these causal links are tenuous because it is rather difficult to control the network conditions affecting players on public servers, and it is almost impossible to know the types of players frequenting public servers. This leads to a natural desire for running user-experience trials under circumstances where the network conditions can be controlled and the players accurately classified.

Our paper looks at some specific issues surrounding the creation of predictable, controlled jitter in a localized network testbed that emulates the network conditions experienced by typical multi-player game clients (Figure 1). Unlike latency and loss, jitter is harder to produce reliably because it is a statistical variation in latency rather than a completely independent variable. Jitter occurs primarily as the result of queuing delays in network devices such as switches and routers.

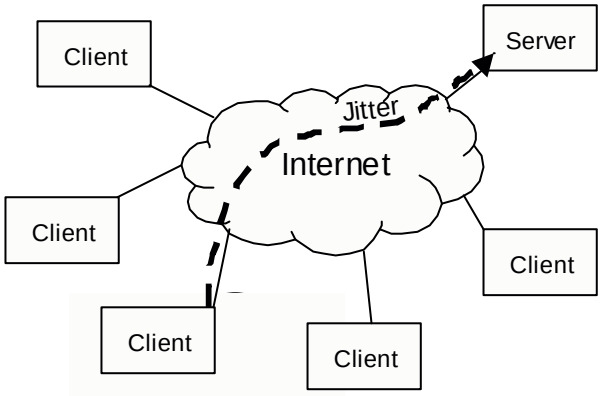

Figure 1. Jitter in the real world

Section 2 begins with some mathematical predictions regarding the consequences of introducing jitter through random fluctuations of latency and measuring jitter through an uncorrelated sampling process. Section 3 describes our experimental testbed for introducing and measuring controlled jitter, section 4 describes our results and the paper concludes in section 5.

2. MATHEMATICAL PREDICTIONS

We can model the behaviour of a uniformly or normally distributed jitter system using fairly simple probability results.

Where the delay in both directions is uniform and the sample is delayed on both the outbound and inbound paths, then the total delay will be the sum of two uniform

distributions. Where random variables are independent, the probability density function of the sum of them is their convolution. It is easy to show that the probability density function of the sum of two uniformly distributed random variables has a triangular distribution [15]. We show this below before developing models for the more complicated scenarios.

Assume the jitter in the outbound and inbound direction is uniformly distributed with jitter between 0 and a seconds. Denote the random component of the outbound jitter by X and the inbound by Y. Then total jitter is:

$Z = X+Y$.

Let the probability density functions for X and Y be f_X and f_Y respectively. The probability density function for Z is therefore given by:

$$f_Z = \int_{-\infty}^{\infty} f_Y(z-x) f_X(x) dx \qquad (1)$$

$$= \begin{cases} \frac{z}{a^2}, \text{ for } 0 \le z \le a \\ \frac{2}{a} - \frac{z}{a^2}, \text{ for } a \le z \le 2a \\ 0 \text{ otherwise} \end{cases} \qquad (2)$$

This describes a triangular distribution as shown in Figure 2.

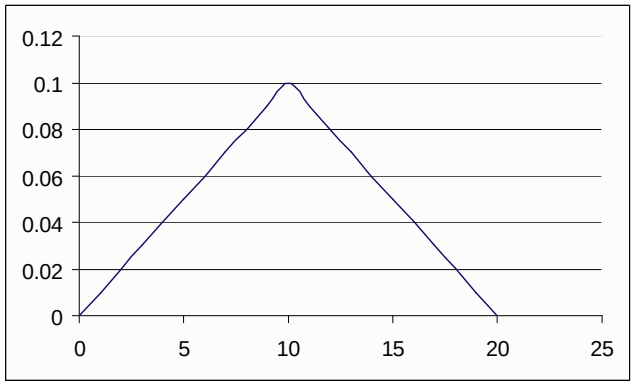

Figure 2. Triangular Distribution

In some of the scenarios described in this paper, not all inbound samples are subject to jitter. In this case we can describe the probability distribution with an impulse function at zero and a uniform distribution on the remainder of the interval [0,a] as follows. We use the same definitions as before and let the probability of the sample not being subject to inbound jitter be u. Denote the delta function at z by $\delta(z)$. The probability density function for Y then becomes:

$$f_Y = \begin{cases} u\delta(y), \text{ for } y = 0 \\ \frac{u}{2a}, \text{ for } 0 \le y \le a \\ 0 \text{ otherwise} \end{cases} \qquad (3)$$

In this case, the probability density function for Z is given by:

$$f_Z = \begin{cases} \frac{u}{2a} + \left(\frac{1-u}{a^2}\right) z, \text{ for } 0 \le z \le a \\ \left(\frac{4-3u}{2a}\right) - \left(\frac{1-u}{a^2}\right) z, \text{ for } a \le z \le 2a \\ 0 \text{ otherwise} \end{cases} \qquad (5)$$

This describes a triangular distribution with a vertical axis displacement as shown in Figure 3. When u is zero, it simplifies to the triangular distribution described in (2). When u is 1 it simplifies to a uniform distribution. For Figure 3 the value of u is 0.8.

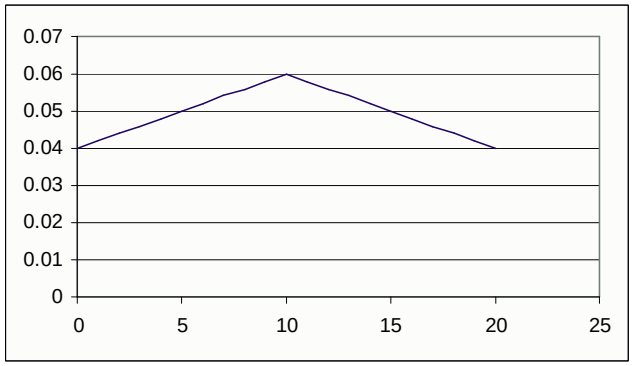

Figure 3. Modified Triangular Distribution

The final scenario we describe is where the inbound jitter and outbound jitter are both from a normal distribution. It is well known that the sum of two normally distributed variables is itself normally distributed, so we would expect the total jitter in that case to have a normal distribution.

Consequently, we would expect the random component of our experimentally observed jitter to have one of the distributions described above or a normal distribution.

3. EXPERIMENTAL METHODOLOGY

3.1 Creating Jitter in the Laboratory

FreeBSD (a stable, widely used, open source unix-like operating system) includes an extremely useful kernel module known as `dummynet` [9] . Dummynet enables the imposition of additional latency, bandwith limits and packet loss to traffic flowing through the kernel's network stack. A FreeBSD-based ethernet bridge can introduce a wide range of controlled latency and packet loss scenarios.

Dummynet is reasonably precise as a latency and loss rate tool [11] (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 through FreeBSD's internal `ipfw` packet filter, and operates

on traffic flows identified as 'pipes'. For example, 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%)

To create jitter we must continuously vary the dummynet pipe rule to cause fluctuations in the configured latency. One goal of this paper is to characterise dummynet's efficacy when used in such a manner, and demonstrate that it can be a suitably predictable and controllable source of network jitter.

We explored the use of other freely available traffic shaping tools such as the BSD family's ALTQ [12] , Linux's IPROUTE2 [13] and NIST Net [14] . Of these, NIST Net appeared to be most promising, as it claims to offer the ability to produce configurable jitter. However, the project has been abandoned for some time (supporting only the now dated Linux 2.4 kernel series). All of the other tools implement essentially the same functionality as dummynet, so we believe our results will generalize to testbeds built with them instead.

3.2 Testbed configurations

Our artificial jitter testbed consisted of three Intel 2.4 GHz Celeron hosts, interconnected by 100Mbit/sec Ethernet as shown in Figure 4. One machine (Pinger) generates ICMP Echo Request traffic through a specially configured bridge to a Target host. All three machines ran FreeBSD 4.9 kernels re-compiled with 1ms tick timers.

Figure 4. Physical network layout diagram

Our experiments explored the three separate delay/jitter configurations shown in **Figure 5**.

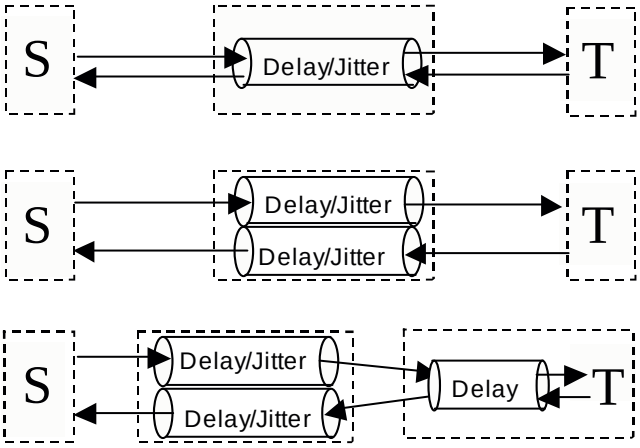

Figure 5. Three test configurations

The first puts all ICMP ping traffic traveling between the source and target in both directions into a single jittering pipe. The second puts Pinger->Target and Target->Pinger traffic in their own separate pipes. The third introduces an additional fixed delay at the Target.

We found no difference between the first two methods at the data rates we used, although the first method could potentially suffer from queue exhaustion and uncontrolled packet loss as traffic loads increased. Consequently we used two pipes for most results presented in this paper.

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

4.EXPERIMENTAL RESULTS

Although it is easy to reconfigure a dummynet pipe at regular intervals from a pseudo-random stream of integer values, 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 (jittering) the dummynet pipe according to a uniform random distribution can create unexpectedly non-uniform distributions of actual latency through your bridge.

4.1 Jitter Constraints

We sampled the actual jitter created along our end-to-end path by using high speed 'ping' trials from Pinger to Target. In this context, high speed means a stream of ICMP Echo Request packets every 5ms, 25ms or 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 (jittered) every X milliseconds, for X = 100, 500, 1000, and 2000. This was intended to emulate moderately fast to slow fluctuations in path latency. As all traffic ran through the bridge twice, ping's estimated round trip time (RTT) is twice the delay configured into the dummynet pipes. When configured with uniformly distributed latencies from 28ms to 52ms (and a mean of 40ms), we created a range of RTTs from 56ms to 104ms with a mean of 80ms. The normally distributed latencies were centered about a mean of 40ms with a standard deviation of 4ms, which approximated the same spread of latencies obtained using the uniform distribution.

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 update packets 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 of such applications is sampled every 40-100ms, constraining the player's awareness of path latency changes.

We tested 2 types of random distribution - uniform and normal. The last three trials were stimulated by uniform linear/ramp distributions.

3rd Workshop on the Internet, Telecommunications, and Signal Processing (WITSP'04), Adelaide, Australia, December 20-22 2004

4.2 Trial Results using Uniform Random Distributions

Figure 6 shows the resulting distribution of RTTs, measured over 100000 jitter transitions, with the bridge jittering every 1000ms (i.e. picking a new, random latency value between 28ms and 52ms every second). Two trials are presented, in which ping packets were generated at 5ms or 50ms intervals, uncorrelated to the jittering transitions every second. The distribution looks reasonably uniform (we achieved an identical result when jittering every 2 seconds). Figure 7 shows the distribution of measured jitter over 100000 uniform random jitter transitions with a jittering interval of 500ms. Figure 7 is beginning to show some of the displaced triangular behaviour predicted in our mathematical model (Figure 3).

This behaviour becomes clearer in Figure 8. We also show in Figure 8 that there is no discernable distinction between using one pipe or two when jittering packets through the bridge (first and second configurations respectively from Figure 5). In addition, the 25ms and 50ms sampling histograms are excluded as they looked exactly the same.

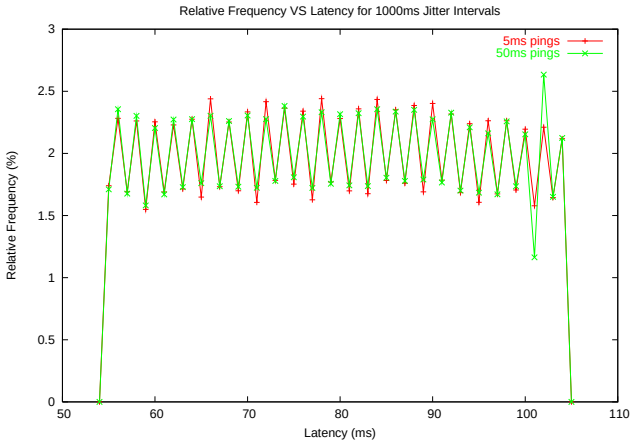

Figure 6. Distribution of measured latency given uniform jitter every 1000ms, 100000jitter samples

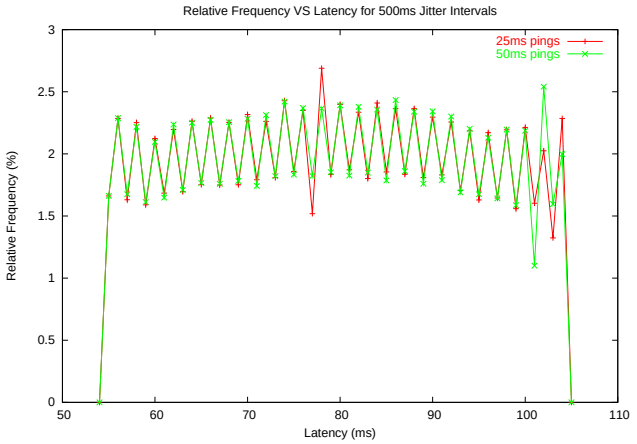

Figure 7. Distribution of measured latency given uniform jitter every 500ms, 100000jitter samples

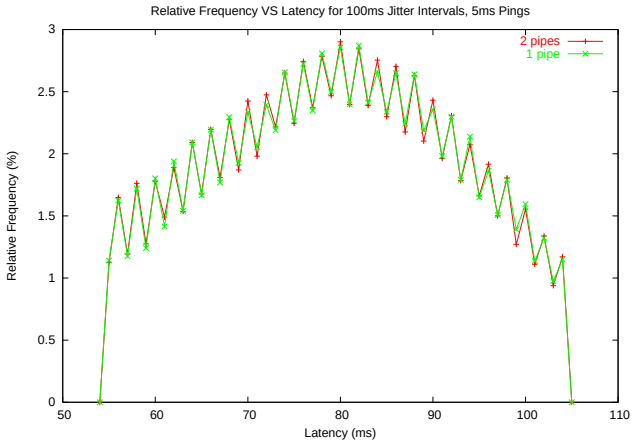

Figure 8. Distribution of measured latency given uniform jitter every 100ms, 5ms sample intervals, 100000jitter samples

Figure 9 is based on the third configuration in Figure 5, with the 40ms mean latency split between a fixed 20ms at the Target and the bridge set to jitter around a mean of 20ms instead of 40ms. We see that splitting the source of jitter and latency across two delay elements does not materially change the results, which suggests an accurate usability testbed need only implement a single bridge to provide the desired jitter and mean latency.

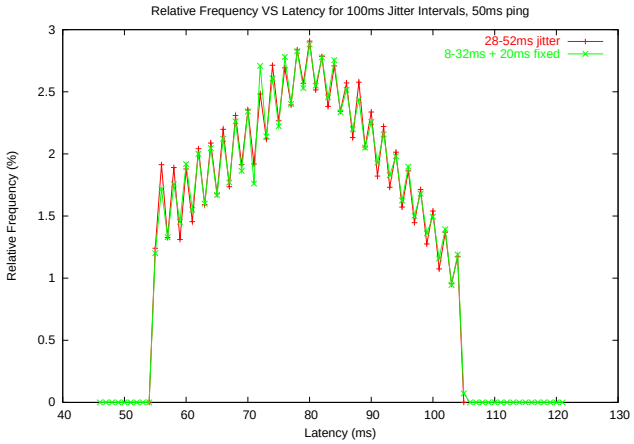

Figure 9. Distribution of measured latency given uniform jitter every 100ms, 2 pipes, 100000jitter samples

Figure 10 provides a more dramatic illustration of what happens when the jittering interval approaches, and then becomes smaller than, the average RTT. There's an increasing chance of one or more packets being in the pipe itself each time dummynet's pipe rule is reconfigured, skewing the effective RTT distribution towards the more triangular shape predicted in section 2. (Given the uniform distribution of jittering and the uncorrelated sampling of 5ms ping intervals.)

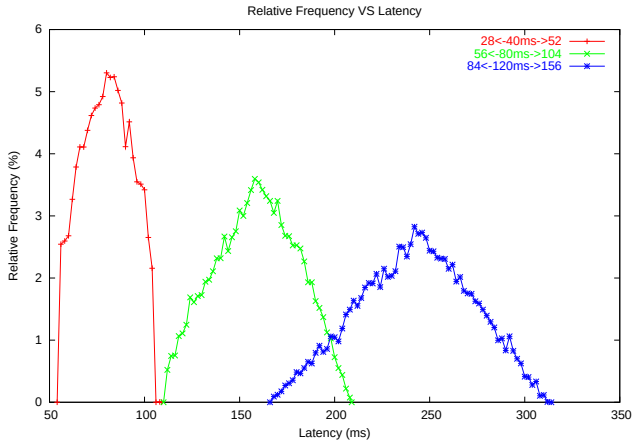

Figure 10. Distribution of measured latency given uniform jitter every 100ms, 5ms sampling, for mean latencies of 40ms, 80ms and 120ms ± 30%

4.3 Trial Results using Random Normal Distributions

The Central Limit Theorem [15] suggests that a Normal distribution should better approximate the spread of jitter present in the Internet, as jitter is caused by the cumulative variation in delay within multiple network devices. (CLT says that when we add multiple variates from the same distribution they will tend to the Normal Distribution.)

Figure 11 shows the resulting distribution of RTTs, measured over 50000 jitter transitions, with the bridge jittering every 100ms (picking a new random latency value normally distributed around the mean of 40ms each time). Two trials are presented, in which ping packets were generated at 5ms or 50ms intervals. As expected, the sum of two normal distributions (jittering of the path in each direction) is also normally distributed.

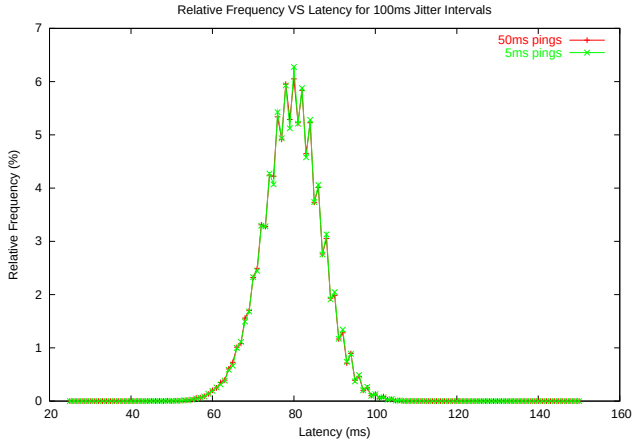

Figure 11. Distribution of measured latency given normally distributed jitter every 100ms, 2 pipes, 50000jitter samples

Figure 12 demonstrates the third test scenario (Figure 5), with the mean 40ms latency component being split into two parts. The bridge was set to jitter the same amount around 20ms instead of 40ms, and a 20ms fixed delay was introduced at the target machine. As with Figure 9 the results suggest an accurate usability testbed need only implement a single bridge to provide the desired jitter and mean latency.

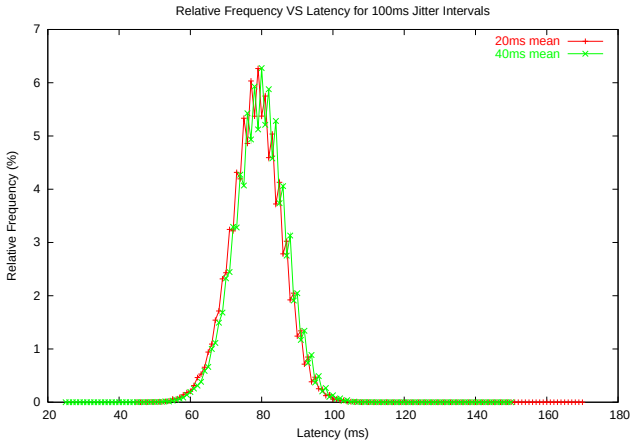

Figure 12. Distribution of measured latency given normally distributed jitter every 100ms, 2 pipes, 5ms sampling, 50000jitter samples

4.4 Trials using Non-random Distributions

We also experimentally demonstrated the convolution of distinctly non-uniform jittering distributions and its impact on actual measured RTTs. Figure 13 shows the results of jittering 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 8 comes about.]

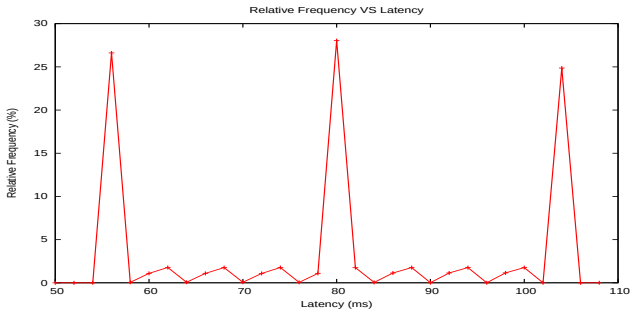

Figure 13. RTT when jitter between two distinct latencies at a 100ms jitter interval

Figure 14 shows the RTT distributions that resulted when we re-ran our trials using three different linear 'ramp' distributions. The RTT distributions are marked '52->28' (jittering with 24 one-millisecond steps from 52ms down to 28ms, then repeat from 52ms), '28->52' (jittering with 24 one-millisecond steps from 28ms up to 52ms, then repeat from 28ms), and '28->52->28' (jittering with 48 one-millisecond steps from 28ms up to 52ms then back down to 28ms again).

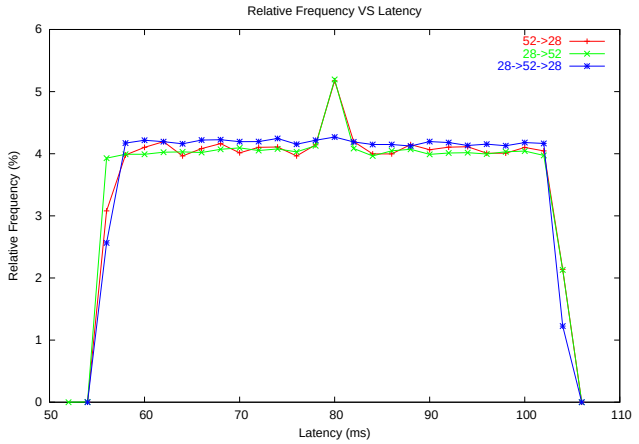

Figure 14. RTT when jitter with ramp distributions and a 100ms jitter interval

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 14 is quite different to Figure 8, even though both were generated by jittering dummynet every 100ms. The '28->52->28' jittering function results in quite a smooth distribution of RTT. Both 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).

5. CONCLUSION

The evaluation of highly interactive networked games (and similar applications) can benefit from usability trials conducted on testbeds that provide controlled and predictable latency, loss and jitter. This paper illustrates some interesting results when emulating jitter using FreeBSD's kernel-resident dummynet module, and provides the mathematical premises behind these observations. If you 'jitter' the dummynet pipe at intervals that approach the pipe's mean latency the resulting distribution of round trip times (RTTs) will be skewed and cluster more around the mean value even though your driving distribution is uniform. This is consistent with the convolution of two uniform random distributions.

It is clear that experimental testbeds should carefully utilize non-uniform (e.g. normal, or otherwise pre-skewed) jittering distributions if they wish to jitter at rates approaching their mean client-server path length. Alternatively, the jittering intervals should be at least ten times longer than the desired mean client-server path length. 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. We expect this paper's insights to augment the development of jittering tools for usability testbeds built around unix-based toolkits.

REFERENCES

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

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

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

[4] 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

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

[6] 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)

[7] 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

[8] Lang, T., Branch, P., Armitage, G. J., "A Synthetic Traffic Model for Quake 3", Proceedings of ACM SIGCHI ACE2004 conference, Singapore, June 2004

[9] 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

[10] 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

[11] 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)

[12] ALTQ, http://www.csl.sony.co.jp/person/kjc/kjc/software.html (as of August 2004)

[13] IPROUTE2, http://snafu.freedom.org/linux2.2/iproute-notes.html (as of August 2004)

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

[15] Ash, R., "Basic Probability Theory", Wiley, NewYork, 1970.