An Unobtrusive Method for Tracking Network Latency in Online Games

Damien Marshall, Brian Mooney, Séamus McLoone, Tomás Ward

Dept. of Electronic Engineering, National University of Ireland Maynooth,
Maynooth, Co. Kildare, Ireland
{dmarshall,smcloone}@eeng.nuim.ie

Abstract

Online games are a very important class of distributed interactive applications. Their success is heavily dependant on the level of consistency that can be maintained between participants communicating in the virtual world. Achieving a high level of consistency usually involves the transmission of a large amount of network traffic. However, if the underlying network connecting participants is unable to process this traffic, then network latency will increase, which will in turn negatively impact on consistency. Many schemes exist which attempt to reduce network traffic, and thus reduce the effect of network latency on the interactive application. However, applications that employ these schemes tend to do so with little knowledge of the underlying network conditions, and assume a worst-case scenario of limited bandwidth. Such an assumption can actually cause these latency reduction schemes to perform sub-optimally, and ironically introduce more inconsistency than they reduce. Hence, it is important that online game applications become aware of network conditions, such as available bandwidth. Existing methods of estimating bandwidth operate by analysing trends in one-way latency, and require that extra data be transmitted between nodes in order to capture the latency trends. Such an approach does not suit online games, as the extra data requirements could increase network latency, and affect the ability of the application to scale to multiple participants. To deal with this issue, this paper proposes a method by which online games can unobtrusively track one-way network latency. This method requires no time-stamping information to be transmitted between participants and operates using data already being transmitted as part of the online game application, meaning that its impact on the network is minimal. NS2 simulations demonstrate that the trends collected by this method can be used to estimate bandwidth under certain conditions.

Keywords: Online games, unobtrusive bandwidth estimation, network latency trends

1. Introduction

Online games have become extremely popular, both socially and commercially in recent years. At their core, they involve multiple participants collaborating and competing within a virtual environment, even though those participants may be located at geographically separate locations. Popular examples of online games include first person shooters, such as Counter Strike, and massively multiplayer online games, such as World of Warcraft.

In order for interaction between participants within the virtual environment to be fruitful, it is important that a sufficient level of consistency is maintained between participants in real time. Consistency is the degree to which each participant experiences the same worldview [1, 2]. One of the key issues limiting the consistency that can be achieved between participants in online games is network bandwidth. Network bandwidth is a measure of maximum throughput of traffic on the network. If the traffic generated by an application exceeds the bandwidth, then data will need to be buffered or dropped until the flow of data decreases [3]. This buffering can occur on a dedicated network node, such as a router, or on the computer hosting the application itself. Latency, or the time taken to transmit a packet from one application to another across a network, and loss of data will increase every time the bandwidth is exceeded.

Many application layer techniques exist which attempt to improve consistency by reducing the latency caused by overloaded network nodes [4, 5]. They do this by introducing an acceptable level of one type of inconsistency, which causes a reduction in network traffic, with the aim of improving overall consistency. A popular example of such a technique is dead reckoning [6]. Under this approach, each participant maintains a simplistic model of their own position information. The behaviour of this model is determined by a parametric model of the actual participant position. The spatial difference between the actual position and the model position is continually calculated. When this value exceeds a predefined spatial error threshold value, the parameters describing the parametric model are updated to reflect the most current behaviour of the actual entity. A network message is transmitted at this time. Upon receipt of this message, each receiving participant then uses the data in this message to update the remote position of the participant entity, and to predict the behaviour of a remote participant until the next update.

The dead reckoning approach is a standard packet reduction technique in online games. Recent work, however, has demonstrated that if techniques such as dead reckoning are not carefully tuned to match the characteristics of the network connecting participants, then they can actually introduce more inconsistency than they reduce [7]. The characteristics of the network are governed by many factors, including the number of users on the network at any one time and the bandwidth of each node on the network. These characteristics govern an optimal region of packet transmission rate for the online game. Transmitting at a rate above this region means that network latency will increase due to overloaded network hardware, while transmitting at a rate below this optimal region means that network latency cannot be improved, and will remain constant. If a packet reduction scheme operates below the optimal region, then inconsistency is unnecessarily introduced, with no subsequent improvement in network latency.

If a packet reduction technique is to perform optimally, it must continually adapt to the characteristics of the network, in particular, network bandwidth. Techniques that measure network bandwidth operate by analysing trends in one-way network latency. Typically, this requires a dedicated standalone application, or extra data to be transmitted as part of an existing application. The extra data requirements could negatively impact on network latency, and reduce the ability of the online game to scale to multiple participants. To deal with this issue, this paper proposes a novel method of unobtrusively tracking one-way network latency trends in online games. This method does not require any clock synchronisation data to be transmitted between participants, and uses data already being transmitted as part of an existing online game application. It's impact on the network beyond that which the online game application is already incurring is minimal. The one-way latency trend information collected from this technique can then be used to estimate network bandwidth under certain conditions.

The rest of the paper is structured as follows. In Section 2, a short review of bandwidth estimation techniques is given. Section 3 describes a method of measuring network latency trends with minimal impact on the underlying network. Results collected from NS2 simulations of the method are presented in Section 4, and show that relative latency measures accurately match the trends in actual network latency. It is also demonstrated that the relative latency measures are useful in estimating bandwidth in certain conditions. The paper finishes in Section 5, with some conclusions and a description of future work.

2. Bandwidth Estimation Techniques

Accurate bandwidth calculation requires direct access to key nodes on a network, such as network routers. As access to these nodes is restricted and is typically not available to the end user, work has concentrated on means of estimating bandwidth from the application layer only. At this layer, techniques operate by transmitting a group of packets from an application on one node on the network to another application on another node. The packets are transmitted at specific intervals. The disturbance of the arrival intervals, due to network latency, relative to the sending intervals is analysed by the application on the receiving node. Bandwidth is inferred from the nature of this disturbance. There are two main categories of bandwidth estimation techniques [8]. Passive approaches estimate bandwidth off-line from trace data collected during the run-time of the application, while active probing approaches attempts to measure bandwidth during application run-time.

One of the earliest active probing techniques is the "Packet Pair" technique [9]. Under this approach, two packets are transmitted in rapid succession from the sending node. The dispersion in the arrival times of the packets measured at the receiving node gives a measure of the link with lowest capacity between the two nodes. The ToPP (Train of Packet Pairs) method operates by transmitting a series of packet pairs [10]. The packet transmission interval is decreased linearly, until a packet rate is found that incurs a significant dispersion in packet arrival time. This dispersion is used to measure link capacity, as well as end-to-end available bandwidth. The SloPS (Self Loading Periodic Streams) technique operates in similar manner to ToPP, and is used to measure end-to-end available bandwidth only [11]. SLoPS, operates by transmitting a stream of 100 packets, each of which is 96 bytes long, with an equal packet interval, which is initially set to 100μsec. The trend in the one-way delay of these packets is analysed. If the bandwidth requirement of the stream is greater than the lowest available network bandwidth on the link connecting the two nodes, packets will need to be queued, and there will be an increasing trend in the one-way network delays. Otherwise, there will be a constant latency trend (taking jitter into account). Depending on the trend, a rate with a greater or smaller packet interval is chosen, and the network is again tested. This process continues until two rates are found, such that the difference between the two rates is an error value set by the user. The lower rate is the highest rate that caused a constant latency trend, while the higher rate is the lowest rate that caused an increasing trend. These values give the range in which available end-to-end bandwidth falls.

All the active probing techniques outlined above are currently implemented as stand alone applications, and require that extra data, beyond typical application data, be transmitted in order to operate. Although applications such as pathChirp attempt to improve the efficiency of the SloPS technique by reducing the amount of data that needs to be transmitted, it still requires extra data beyond necessary application data, to be transmitted [8]. These extra data requirements do not suit latency sensitive applications, such as online games, and could negatively impact on the experience of participants in the game, and the ability of the application to scale to multiple participants. To deal with this issue, a novel technique for analysing trends in the one-way delay experienced by online game traffic is outlined in the next section. This method is unobtrusive, as it uses data that is already being transmitted as part of the online gaming application and also avoids the need to send clock synchronisation information in each packet. It's impact on the network beyond that which the application is already incurring is therefore minimal.

3. Relative Latency

In this section, a method of measuring trends in network latency, which is referred to as relative latency, is described. Consider the case where a client/server architecture is employed in an online game. This is currently one of the most popular architectures in commercial online games. In a client/server scenario, the typical interval between packet transmissions from each client to the server is usually a constant value between 0.05s to 0.025s (20 to 40 packets per second), with a typical packet size of 60 bytes. On the other hand, the packet interval from the server to a client is typically a constant value between 0.1s and 0.06s (10 to 15 packets per second), while packet size is, on average, 100 bytes [12, 13].

Under the relative latency calculation method, each client initially transmits a "Join Request" packet to the server, requesting that they can take part in the current game. Included in this packet, amongst other game specific information such as player name and location, is the client's intended rate of packet generation (for example, 30 packets per second). Upon accepting the client's request, the server responds with a "Request accepted" packet, which also includes its own intended rate of packet generation, amongst other game specific information. Both the client and server record the time when each of these initial packets was received according to their own local clock. This information about the intended transmission rate is the only extra data required to measure relative latency.

Following this initial connection negotiation, both the client and server transmit game specific data, such as position and current action, as normal, at the rate specified in the original connection negotiation message. As both client and server know the packet transmission rate of the other, each can both predict when a packet should arrive. If a packet arrives later or earlier than the predicted time, then latency has increased or decreased relative to the initial latency value. If the packet arrives at the expected time, then latency has remained constant, so relative latency remains the same. In this way, variation in latency relative to the first latency value, or relative latency, is calculated, without the need for any time synchronisation information to be sent between participants.

An example of the approach described above in action can be seen in Figure 1 (a). Here, it can be seen how the initial "Join Request" packet is transmitted to the server from the client. This packet contains, amongst other game specific data, information detailing that the client's transmission rate is 1 packet per second. The server then knows to expect a packet from that client every second. The arrival time of each subsequent packet is compared to the expected time of that packet in order to calculate relative latency. For example, Packet 1 arrives when expected, so relative latency remains at zero. However, Packet 2 is delayed by a 1s, leading to an increase in relative latency. Figure 1(b) shows relative latency and actual one-way latency for the packets transmitted in Figure 1 (a). It can be seen how the trends in latency given by the relative latency measure match that of the actual latency value.

It is important to outline the main benefit of removing the clock synchronisation requirement for latency trend analysis. Traditionally, to calculate one-way latency, clocks must first be synchronised. A time value then needs to be transmitted with each packet. If this time value is transmitted using a 4 byte integer, then the size of a 60 byte packet would increase by 6.66%. Although this seems minimal, it would mean that a client sending at a rate of 30 packets per second would be required to transmit an extra 120 bytes per second, or the equivalent of two extra packets per second, to the server, just to calculate one-way latency. A server sending packets at a rate of 10 PPS would be required to transmit an extra 40 bytes per second to each client. Given that current online game servers can host hundreds, or even thousands, of players, the savings in bandwidth and overall cost to the server provider could be considerable.

To analyse the performance of the technique described in the section, a number of simulations were conducted using a networking simulator, known as NS2. The results collected from these simulations are described in the next section

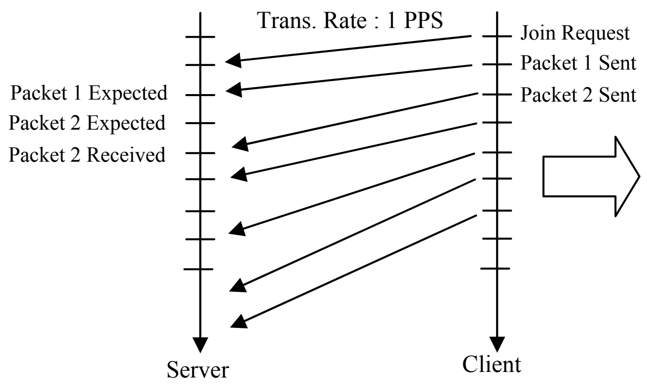


Figure 1(a). The variation in packet arrival rates relative to the first packet latency is calculated

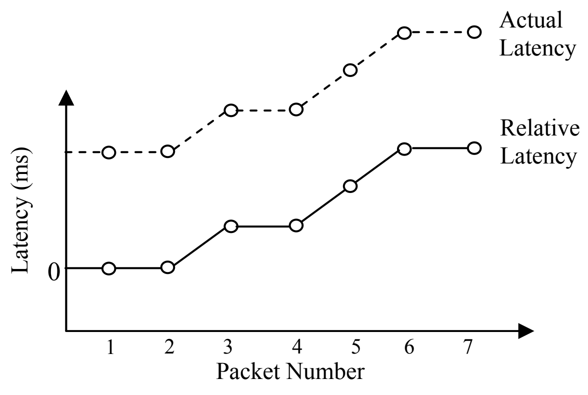


Figure 1(b). The trends in relative latency match that of actual latency.

4. NS2 Simulations and Results

A number of simulations were conducted using a networking simulator known as NS2. This networking simulator has been in constant development and revision for many years, and is currently the basis for much research in the area of networking [14]. Using NS2, a typical online game with a client/server architecture was simulated. Packet size for the client to server connection was set to 60 bytes with a packet interval of 0.033ms, resulting in a total transmission rate of 1.8 kilobytes/s per client. Packet size for the server to client connection was set to 100 bytes, with a packet interval of 0.1ms, resulting in a total rate of 1 kilobytes/s to each client. For simplicity, a single server and two clients were simulated. An overview of the network connecting the participants is shown in Figure 2. Latency was set to 0ms, with jitter of 20ms. The amount of bandwidth on the network was varied on the bottleneck link between the server and the clients.

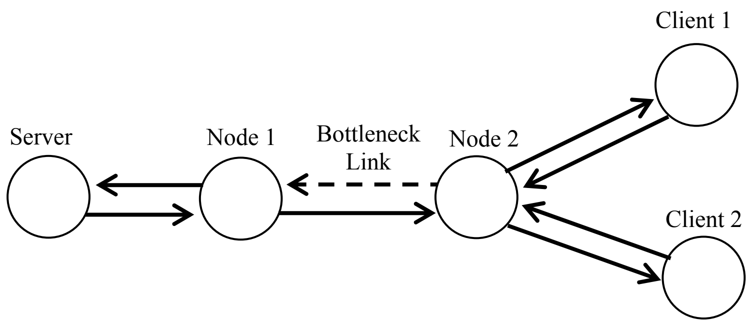
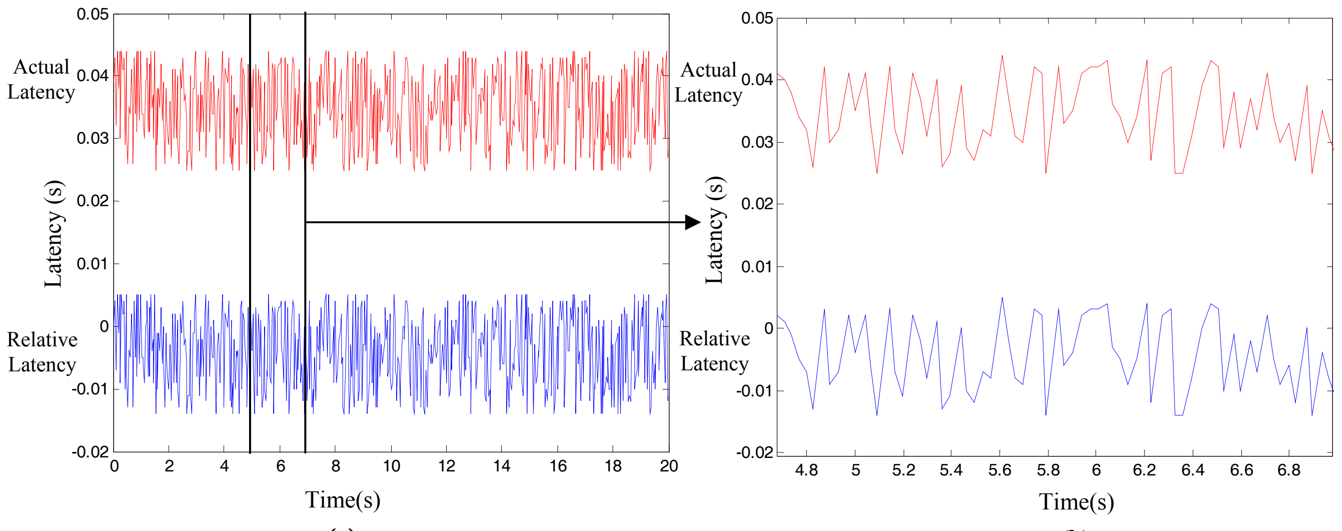


Figure 2. Overview of simulation network

In the first simulation, only Client 1 transmitted data to the server. The downstream bandwidth was set to 2.5 kilobytes/s on the server side (connection from Node 1 to Node 2 in Figure 2). One-way latency using synchronised clocks and one-way relative latency was recorded, and is shown in Figure 3(a). Here we can see that, as the data being transmitted by Client 1 does not exceed the network bandwidth, latency remains constant, taking jitter into account. Minimum latency is 0.024s, as this is the length of time a 2.5 kilobyte/s link takes to process a 60 byte packet. It can be also seen how the relative latency accurately tracks the trends in the actual latency values. For further clarity, Figure 3(b) shows a zoomed in section of Figure 3(a). Again, it can be seen how the relative increases and decreases can be accurately tracked, without the need for transmission of clock synchronisation information.



(a) **(b)**

Figure 3(a) Latency trends with a total rate of 1.8 kilobytes/s and a bandwidth of 2.5 kilobytes/s (b) Zoomed in section of Figure 3(a)

A second simulation was conducted to analyse how the relative latency tracking scheme reacts to varying latency trends. In this case, the bottleneck link bandwidth was again set to 2.5 kilobytes/s Client 1 joined the server as normal at time 0, and transmitted 60 byte packets with an interval of 0.033 ms. At a time of 10 seconds, Client 2 joins the server, and begins transmitting data at the same rate and packet size as Client 1. At 10.5 seconds, Client 2 leaves the server, and stops transmitting data. During the time interval when both clients are sending data to the server, a total of 3.6 kilobyte/s is transmitted. This is greater than the available bandwidth on the bottleneck link, and causes network latency to increase. This is clearly evident from Figure 4. Here, latency and relative latency are again plotted against time for the connection between Client 1 and the server. It is clear from Figure 4 how the relative latency trends accurately match the increasing network latency at time 10 seconds, and the decreasing network latency when Client 2 stops transmission at a time of 10.5 seconds. Figure 5 shows the latency trends calculated when bandwidth on the bottleneck link is set to 3.125 kilobytes/s. Again, it can be seen how the relative latency measure accurately captures all the nuances of the actual latency value.

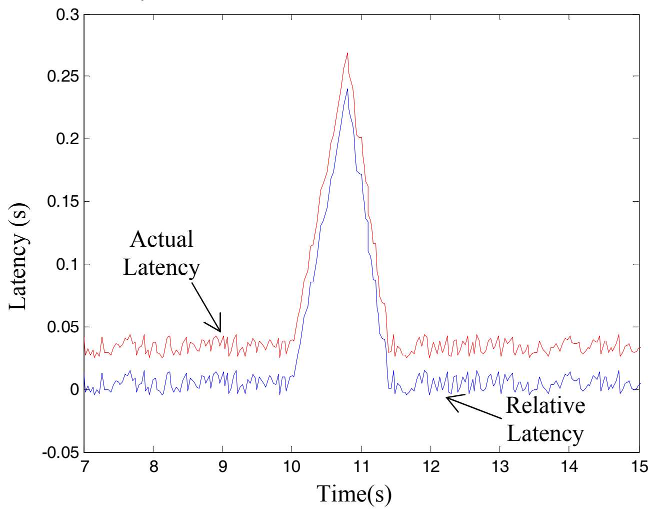


Figure 4. Latency trend with a total rate of 3.6 kilobytes/s and a bandwidth of 2.5 kilobytes/s

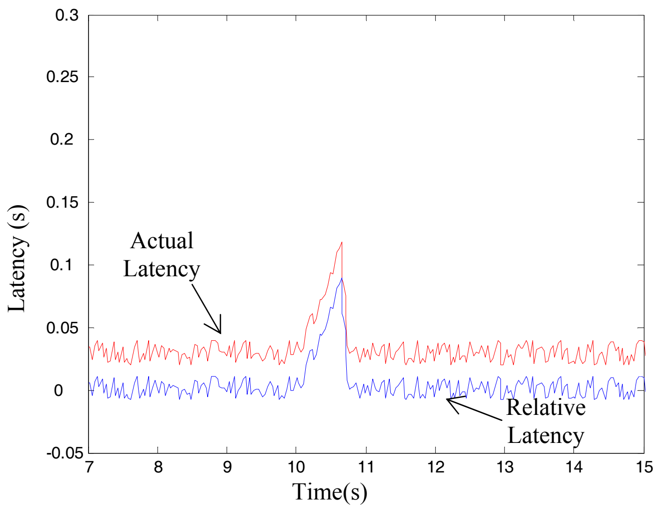


Figure 5. Latency trend with a total rate of 3.6 kilobytes/s and a bandwidth of 3.125 kilobytes/s

An increasing trend in one-way network latency, such as those seen in Figures 4 and 5, is an indication that one of the nodes on the network is currently being overloaded. The nature of this trend can be used to provide an estimate of the available network bandwidth on the connection between the client and the server. In the next subsection, a method of estimating bandwidth from the nature of the trends in network latency is explored in further detail.

4.1 Bandwidth Estimation

Both the SloPS and ToPP approach use an iterative approach to determine bandwidth. Although they can quickly converge to an estimated bandwidth value, such an iterative approach is not suited to online games, which have very strict time constraints. In such a case, it is necessary to establish a relationship between the application’s packet transmission rate and the measured trends in network latency in real time.

A relationship between network latency and transmission rates can be established using basic queuing theory. Consider a queuing system with a deterministic arrival rate and deterministic service rate, also known as a D/D/1 queuing system. In this case, if the arrival rate, R_A, is greater than the service rate, R_S, at a node, then messages will be added to a queue and delayed. In this case, the system is said to be unstable, and the queue length and delay will grow indefinitely. The increase in delay per packet, L_P is given by:

$$L_P = \frac{1}{R_S} - \frac{1}{R_A} \qquad (1)$$

The total increase in latency in a given time interval can be found by multiplying Equation (1) by the number of packets received during that interval. In a D/D/1 queuing scenario, the server node would process R_S packets per second, meaning that any recipients connected to this node would also receive R_S packets per second. The increase in latency over a second, as measured by one of these recipients, would be then be given by R_SL_P. Using some straightforward simplifications, the increase in latency per second, L_S, can be found in terms of the arrival and services rate, R_A and R_S respectively, as shown in (2).

$$
\begin{aligned}
&L_S = R_S L_P \\
&\Rightarrow L_S = R_S\left(\frac{1}{R_S} - \frac{1}{R_A}\right) \\
&\Rightarrow L_S = R_S\left(\frac{R_A - R_S}{R_S R_A}\right) \\
&\Rightarrow L_S = \frac{R_A - R_S}{R_A} \qquad (2)
\end{aligned}
$$

In the simulations presented in the previous section, L_S, can be easily calculated from the slope of the relative latency trend. As R_A, the packet transmission rate, is already known, Equation (2) can be used to estimate the service rate of the bottleneck link, R_S.

To test the accuracy of Equation (2), a number of simulations were again conducted. The multiple client scenario as detailed in the previous section was again employed, resulting in a total transmission rate of 3.6 kilobytes/s. The bandwidth of the bottleneck link was varied. The slope of the network latency values over time was calculated within the region where network latency was increasing. Example results from these simulations are presented in Table 1.

It is clear from Table 1 that bandwidth can be accurately estimated using only the slope information and total transmission information. However, it should be noted that this is only a relatively simple approach to estimating bandwidth, and does not consider the effect of cross traffic or varying queuing strategies of network nodes, for example. Improving the robustness of this estimation technique remains the focus of future work.

Total Transmitted (R_A)	Slope (L_S)	**Estimated Bandwidth (R_S)**	**Actual Bandwidth**
3.6 kilobytes/s	0.47916666	**1.87500024 kilobytes/s**	**1.875 kilobytes/s**
3.6 kilobytes/s	0.30555654	**2.49999643 kilobytes/s**	**2.5 kilobytes/s**
3.6 kilobytes/s	0.13194531	**3.12499687 kilobytes/s**	**3.125 kilobytes/s**

Table 1. Bandwidth estimation from relative latency trends

5. Conclusions

Maintenance of an acceptable level of consistency between participants in an online game can require the transmission of a high level of network traffic. If this traffic is greater than the capacity of the nodes connecting participants, then latency will increase, causing a subsequent decrease in consistency. To deal with this issue, many techniques have been developed which reduce network traffic by lowering one level of consistency, with the result of reducing overall network traffic requirements.

However, unless these techniques are carefully tuned, they can actually introduce more consistency issues than they solve. It is important, therefore, that applications employing these techniques become aware of the characteristics of the network in which they operate. This, however, can introduce a significant overhead in terms of extra data transmission requirements, which could in turn negatively impact on the performance of the online game.

To deal with this issue, this work presented a novel method of detecting latency trends in online game traffic that avoids the traditional data requirements of one-way latency analysis. This technique is unobtrusive, as it operates using data that is already being transmitted as part of the online game application. It operates by predicting when a network packet should be received, and comparing this predicted value to when the packet is actually received. The result of this technique is a measure of relative latency, which is an accurate recreation of the trends in actual network latency values. Analysis of these trends can then be used by the online gaming application to estimate bandwidth in real time.

Future work will investigate how this scheme could be employed in a real world gaming scenario, and will examine how it can be used to optimise the performance of packet reduction techniques, such as dead reckoning.

6. Acknowledgements

This work is supported by Science Foundation Ireland and Enterprise Ireland - grant no. IRCSET/SC/04/CS0289.

7. References

[1] Delaney, D., T. Ward, and S. McLoone. (2006). On consistency and network latency in distributed interactive applications: A survey - Part I. *Presence: Teleoperators and Virtual Environments*, 15(2): 218-234.
[2] Delaney, D., T. Ward, and S. McLoone. (2006). On consistency and network latency in distributed interactive applications: A survey - Part II. *Presence: Teleoperators and Virtual Environments*, 15(4): 465-482.
[3] Roehle, B. (1997). Channeling the data flood. *IEEE Spectrum*, 34(3): 32-38.
[4] Delaney, D., T. Ward, and S. McLoone. (2003). *Reducing Update Packets in Distributed Interactive Applications using a Hybrid Mode*. In 16th International Conference on Parallel and Distributed Computing Systems, August 13-15, Reno, USA: 417-422.
[5] McCoy, A., T. Ward, S. McLoone, and D. Delaney. (2006). Multi-step-ahead Neural-Networks for Network Traffic Reduction in DIAs. To appear in. *ACM Transactions on Modelling and Computer Simulation (TOMACS).*
[6] IEEE. IEEE Standard for Distributed Interactive Simulation - Application Protocols IEEE Std 1278.1-1995 IEEE, 1995.
[7] Marshall, D., S. McLoone, T. Ward, and D. Delaney. (2006). *Does Reducing Packet Transmission Rates Help to Improve Consistency in Distributed Interactive Applications?* In 9th International Conference on Computer Games: AI, Animation, Mobile, Educational & Serious Games, 22nd-24th November 2006, Dublin, Ireland: 88-92.
[8] Ribeiro, V., R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. (2003). *pathChirp: Efficient Available Bandwidth Estimation for Network Paths*. In Passive and Active Measurements (PAM) workshop, April 2003
[9] Keshav, S. (1991). A control-theoretic approach to flow control. *ACM SIGCOMM Computer Communication Review*, 21(4): 3-15.
[10] Melander, B., M. Bjorkman, and P. Gunningberg. (2000). *A new end-to-end probing and analysis method for estimating bandwidth bottlenecks*. In Global Telecommunications Conference, 2000., Nov. 27 - Dec 01, San Francisco, CA, USA: 414-420.
[11] Jain, M. and C. Dovrolis. (2002). *End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput*. In Proceedings of the 2002 SIGCOMM conference, Pittsburgh, Pennsylvania, USA: 295-308.
[12] Lang, T., P. Branch, and G. Armitage. (2004). *A synthetic traffic model for Quake3*. In Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology, Singapore: 233-238.
[13] Färber, J. (2004). *Traffic Modelling for Fast Action Network Games*. In Multimedia Tools and Applications, Bruanschweig, Germany: 31-46.
[14] Breslau, L., D. Fall, K. Floyd, S. Heidemann, J. Helmy, A. Huang, P. McCanne, S. Varadhan, and K. Yu. (2000). Advances in network simulation. *IEEE Computer*, 33(5): 59-67.