Construction and Scheduling of Extrapolated Parity Packets for Dead Reckoning in Network Gaming

Gene Cheung
Hewlett-Packard Laboratories Japan
3-8-13, Takaido-higashi
Suginami-ku, Tokyo, Japan 168–0072
gene-cs.cheung@hp.com

Takashi Sakamoto
Hewlett-Packard Laboratories Japan
3-8-13, Takaido-higashi
Suginami-ku, Tokyo, Japan 168–0072
takashi.sakamoto@hp.com

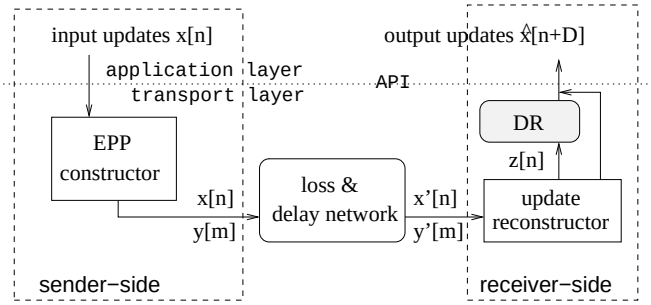

Figure 1: System Overview of EPPs

ABSTRACT

Game update packets of typical First Person Shooting (FPS) network games, small and of delay-sensitive and self-healing characteristics, are disseminated periodically among players for dead reckoning (DR) to predict the current locations of the opposing avatars. To improve delivery of these updates over networks with substantial packet losses and transmission delays, we propose the use of Extrapolated Parity Packets (EPPs). EPPs are application-aware parity packets that can fully recover missing game updates or prediction information. Our proposed scheduling of EPP makes maximal use of the available network bandwidth. Experiments showed noticeable improvements in prediction accuracy at receiver using EPP schemes over non-EPP schemes.

1. INTRODUCTION

In a typical First Person Shooting (FPS) network game [1, 2], game updates of a player's avatar are packetized and disseminated to all others periodically [3]. The most recent window of updates, containing time-varying information like an avatar's coordinates in the virtual world, are used by a position prediction procedure called *dead reckoning* [4], running at each player's terminal, to predict current locations of opposing avatars' for rendering on a player's display screen.

NetGames '07, September 19-20, 2007, Melbourne, Australia
.

Game update packets are, first of all, *very small* — typically much fewer than 100 bytes — when compare to other multimedia data packets like video. They are *delay-sensitive*; the sooner they are delivered, the more accurate dead reckoning can predict an avatar's current location based on them. They are also *self-healing*; lost updates in a window will be subsumed by the next window, which provides more recent and hence more valuable information. We focus on the delivery of game updates in this paper.

To improve transport of this unique data type for networks with substantial packet losses and/or transmission delay like 3G cellular networks [5], we propose the use of a new, game-update specific redundancy packet type called *Extrapolated Parity Packets* (EPPs). Like a parity packet in a Reed Solomon (RS) $(n, n-1)$ code for packetized multimedia streaming [6], redundant information contained in each EPP can be used to recover perfectly information lost in one of $n-1$ source packets. Moreover, each individual EPP can also be used to convey prediction information even in multiple-loss scenarios.

The outline of the paper is as follows. After a review of related works in Section 2, we present a theoretical discussion of dead reckoning in Section 3. We then give an overview of EPPs in Section 4. We discuss construction of EPPs in Section 5. In Section 6, we discuss different packetization methods for EPPs. In Section 7, we show how to schedule EPPs to maximally utilize bandwidth for performance. Finally, results using theoretical models and real game traces and conclusions are presented in Section 8 and 9, respectively.

2. RELATED WORK

Latency has long been casted as a major hurdle for network games [3]. Dead reckoning (DR) [4, 7]—predicting the current position of an avatar based on previous location updates—is often used to create the illusion of low-delay interactivity. Among published works, [4] evaluated different DR schemes experimentally for sports, action and racing games, while [7] proposed the use of globally synchronized clocks to improve accuracy of DR schemes. No matter what DR scheme is used, DR has the inherent drawback that the differences between predicted and actual avatar locations can lead to temporary inconsistencies of global game states at different players. This can potentially lead to bizarre phenomena, like "A Dead Man that Shoots" and "A Flying Tank" examples in [8]. These global state inconsistencies must be fixed using time traceback algorithms like *timewarp* [9] to reconstruct the definitive order of state changes by

different players, who disseminate *game events* [8] containing these state changes to other players or an authoritative server. Instead of game events, in this paper we narrow our focus to improving the delivery of *game updates* to minimize DR prediction errors.

In addition to latency, packet losses in the delivery network can also influence user performance [10] and game consistency [11]. Among traditional transport repair mechanisms, ARQ (Automatic Retransmission Request) would work poorly due to the mismatch of long retransmission delay and the hyper delay sensitivity of game updates. FEC (Forward Error Correction) like the popular Reed Solomon code [6], on the other hand, is feasible, and in fact EPP can be viewed as an application-aware form of FEC.

3. OPTIMAL DEAD RECKONING

In this section, we first introduce theoretical motion models of avatars that we will use extensively for our analysis. Avatar motion models are random processes that model actual movements of avatars over time probabilistically. We then discuss the derivation of the optimal DR scheme for a given motion model in a statistical sense. Motion models of avatars and corresponding optimal DR schemes are used in the development of EPP later in Section 5.

3.1 Avatar Motion Models

The first motion model is a *random displacement* model (`rd`), where the position of avatar at the next time unit $x[n+1]$ is the current time unit $x[n]$ plus a random variable $w[n]$. $w[n]$ is zero-mean with variance σ_w^2. `rd` is written as:

$$x[n+1] = x[n] + w[n] \tag{1}$$

D time unit later, $x[n+D]$ can be written alternatively as:

$$x[n+D] = x[n] + \sum_{i=1}^{D} w[n+D-i] \tag{2}$$

We can similarly define a *random velocity* model (`rv`), where the *instantaneous velocity* of the next point $v[n+1]$ is the current instantaneous velocity $v[n] = x[n] - x[n-1]$ plus random variable $w[n]$. The new position $x[n+1]$ is then the previous position $x[n]$ plus the change resulted from the new instantaneous velocity $v[n] + w[n]$:

$$\begin{aligned} x[n+1] &= x[n] + (v[n] + w[n]) \\ &= 2x[n] - x[n-1] + w[n] \end{aligned} \tag{3}$$

D time unit later, $x[n+D]$ can be written alternatively as:

$$x[n+D] = (D+1)x[n] - Dx[n-1] + \sum_{i=1}^{D} i\ w[n+D-i] \tag{4}$$

Finally, we can define a *random acceleration* model (`ra`), where the *instantaneous acceleration* of the next point $a[n+1]$ is the current instantaneous acceleration $a[n] = v[n] - v[n-1]$ plus a random variable $w[n]$. The new velocity is then $v[n] + a[n]$, and the new position $x[n+1]$ is:

$$\begin{aligned} x[n+1] &= x[n] + (v[n] + a[n]) \\ &= 3x[n] - 3x[n-1] + x[n-2] + w[n] \end{aligned} \tag{5}$$

D time unit later, $x[n+D]$ can be written alternatively as:

$$\begin{aligned} x[n+D] &= \alpha_0[D]\ x[n] + \alpha_1[D]\ x[n-1] + \alpha_2[D]\ x[n-2] \\ &\quad + \sum_{i=1}^{D} \alpha_2[i]\ w[n+D-i] \end{aligned} \tag{6}$$

where the coefficients $\alpha_0[k]$, $\alpha_1[k]$ and $\alpha_2[k]$ are:

$$\alpha_0[k] = 3 + \sum_{i=2}^{k}(i+1) \quad \alpha_1[k] = -3 - \sum_{i=2}^{k}(2i+1) \quad \alpha_2[k] = \sum_{i=1}^{k} i$$

respectively.

3.2 Deriving Optimal DR Schemes

Given an avatar motion model, we now derive the optimal DR scheme in statistical sense. More specifically, given the current set of game updates, $\{x[n], x[n-1], \ldots\}$, we find the optimal prediction $\hat{x}[n+D] = A_0 x[n] + A_1 x[n-1] + \ldots$, where $D \geq 1$ is the time unit into the future we are predicting, such that the expected prediction square error $\mathcal{E} = E[(\hat{x}[n+D] - x[n+D])^2]$ is minimized. For `rd`, if we set $A_0 = 1$ and $A_i = 0, i \geq 1$, using (2) we get:

$$\begin{aligned} \mathcal{E} &= E\left[\left(x[n] - \left(x[n] + \sum_{i=1}^{D} w[n+D-i]\right)\right)^2\right] \\ &= E\left[\left(\sum_{i=1}^{D} w[n+D-i]\right)^2\right] = DE[w^2[n]] = D\sigma_w^2 \end{aligned}$$

where line 2 follows because $w[n+D-1], \ldots, w[n]$ are zero-mean, independent and identically distributed random variables. Because $w[n+D-1], \ldots, w[n]$ are also each independent from set $\{x[n], x[n-1], \ldots\}$, we can do no better, and $\hat{x}[n+D] = x[n]$ is the best prediction in statistical sense for model `rd`. We denote DR scheme $\hat{x}_{cd}[n+D] = x[n]$ as *constant displacement* `cd`.

Using similar argument, we can derive the optimal DR schemes for motion model `rv` and `ra` respectively as:

$$\hat{x}_{cv}[n+D] = (D+1)x[n] - Dx[n-1] \tag{7}$$

$$\begin{aligned} \hat{x}_{ca}[n+D] &= \alpha_0[D]\ x[n] + \alpha_1[D]\ x[n-1] \\ &\quad + \alpha_2[D]\ x[n-2] \end{aligned} \tag{8}$$

where in each case, DR makes a prediction such that the expected prediction square error only involves random variable $w[n+D-1], \ldots, w[n]$. We denote these corresponding optimal DR schemes as *constant velocity* `cv` and *constant acceleration* `ca`, respectively.

4. OVERVIEW OF EXTRAPOLATED PARITY PACKETS (EPP)

We first overview the operation of EPPs shown in Figure 1. The sender-receiver pair can be peer-peer in a P2P system, or server-client if updates are routed through a game server in a server-client architecture. At sender, game updates $x[n]$'s of period T, generated at the *application layer*, are passed down to the *transport layer* via a set of predefined *Application Programming Interfaces* (APIs). *EPP Constructor* constructs *Extrapolated Parity values* (EP) $y[m]$'s using updates $x[n]$'s, packetizes and transmits them over the

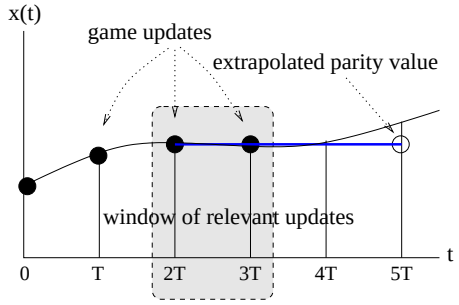

Figure 2: Extrapolated Parity Value (EP) for $D = 2$

delivery network. The receiver's *update reconstructor* receives delayed packets containing $y'[m]$'s and $x'[n]$'s, reconstructs updates $z[n]$'s, and forwards them to DR to predict $\hat{x}[n+D]$'s, where D is the average network delay in update periods from sender to receiver. Alternatively, update reconstructor can reconstruct and send $\hat{x}[n+D]$'s directly to the application, bypassing DR. Note that in our system, DR resides in the transport layer, meaning that the receiver only needs to display avatars at location $\hat{x}[n+D]$ at time $(n+D)T$ *without* prediction. The problem is how to construct the transport blocks such that the receiver's expected prediction square error is minimized.

We assume a sender-receiver connection of available bandwidth C bits per second (bps), while the transmissions of game updates of period T, if packetized individually, consume only R bps, where $R < C$. C can be found, for example, using a TCP-friendly Rate Control (TFRC) algorithm [12] on the wired network, or obtained during session setup between wireless client and the UMTS basestation of 3G network [5]. We packetize and schedule EP $y[m]$'s using extra bandwidth $C - R$. Note that the extra bandwidth cannot be filled simply with more frequent updates with period $T' < T$ for several reasons. First, bandwidth C is a fast time-varying quantity, and it is unreasonable to expect the application to quickly adapt as C varies. Second, as shown in [13], players receiving different frequencies of updates can lead to unfair game play. That means update frequencies should be fixed at the application in accordance to computation and fairness criteria. We next discuss the construction of Extrapolated Parity values (EP).

5. CONSTRUCTION OF EP

We perceive game updates as discrete samples $x[n]$'s, $n \in \mathcal{I}$, of a continuous time signal as shown in Figure 2, where $x(t)$ can be the x-coordinate of an avatar's location at time t. To construct an EP, we essentially perform DR at sender: we predict the D^{th} sample into the future $y[n+D]$ (white circle) using a window of known samples $x[n], \ldots, x[n-l]$ (black circles) called the *window of relevant updates*. EP $y[n+D]$ performs two functions:

1. In scenario of single loss in $x[n], \ldots, x[n-l]$, EP acts as *parity* so that the lost update in the window of relevant updates can be perfectly reconstructed.

2. In scenario of multiple losses in $x[n], \ldots, x[n-l]$, EP acts as *prediction* $\hat{x}[n+D]$ so that $x[n+D]$ can still be accurately predicted as if no loss has occurred.

While RS$(n, n-1)$ code can also perfectly recover one datum loss in $n-1$ data by adding one parity, EP can lead to the same predicted $\hat{x}[n+D]$ even in multi-loss scenarios by being prediction $\hat{x}[n+D]$ itself. Implementationally, when EP acts as parity, update reconstructor passes the reconstructed $x[n]$'s to DR to predict $\hat{x}[n+D]$. When EP acts as prediction, update reconstructor bypasses DR and sends EP as predicted $\hat{x}[n+D]$ directly to application.

Corresponding to the three motion models in Section 3.1, we now discuss in details three EP (re)construction methods.

5.1 Extrapolated Displacement

For motion model `rd`, by performing `cd` at sender, we basically set EP $y[n+D]$ to the most recent update $x[n]$. Obviously, if $x[n]$ is lost in the network, $y[n+D]$ can be used as an identical replacement for $x[n]$, and the delivery of either $x[n]$ or $y[n+D]$ at receiver would lead to the same prediction $\hat{x}[n+D] = x[n]$ at time $(n+D)T$. We call this EP (re)construction method *extrapolated displacement* `xd`.

5.2 Extrapolated Velocity

For motion model `xv`, by performing `cv` at sender, we set EP $y[n+D] = \hat{x}_{cv}[n+D]$ using (7). If only $x[n]$ is lost, we can perfectly recover $x[n]$ using $y[n+D]$ and $x[n-1]$:

$$x[n] = \frac{y[n+D] + Dx[n-1]}{D+1} \tag{9}$$

Alternatively, if only $x[n-1]$ is lost, we can perfectly recover $x[n-1]$ using $y[n+D]$ and $x[n-1]$ by:

$$x[n-1] = \frac{(D+1)x[n] - y[n+1]}{D} \tag{10}$$

Hence $y[n+D]$ fulfills the role of parity. If both $x[n]$ and $x[n-1]$ are lost, then $y[n+D] = \hat{x}_{cv}[n+D]$ will act as prediction and be sent directly to the application by update reconstructor at time $(n+D)T$. We call this EP (re)construction method *extrapolated velocity* `xv`.

5.3 Extrapolated Acceleration

For motion model `xa`, by performing `ca` at sender, we set EP $y[n+D] = \hat{x}_{ca}[n+D]$ using (8). If only $x[n]$ is lost, we recover $x[n]$ at receiver using other received data $y[n+D]$, $x[n-1]$ and $x[n-2]$ as follows:

$$x[n] = \frac{y[n+D] - \alpha_1[D]\ x[n-1] - \alpha_2[D]\ x[n-2]}{\alpha_0[D]} \tag{11}$$

If only $x[n-1]$ is lost, we recover it using:

$$x[n-1] = \frac{y[n+D] - \alpha_0[D]\ x[n] - \alpha_2[D]\ x[n-2]}{\alpha_1[D]} \tag{12}$$

Finally, if only $x[n-2]$ is lost, we recover it using:

$$x[n-2] = \frac{y[n+D] - \alpha_0[D]\ x[n] - \alpha_1[D]\ x[n-1]}{\alpha_2[D]} \tag{13}$$

Hence $y[n+D]$ fulfills the role of parity. If two or more data among set $\{y[n+D], x[n], x[n-1], x[n-2]\}$ are lost, then EP $y[n+D]$ will act as prediction and be sent directly to the application by update reconstructor. We call this EP (re)construction method *extrapolated acceleration* `xa`.

6. PACKETIZATION OF EP

We assume that before an EP can be generated, updates $x[n]$'s are sent as individual update packets T seconds apart as they normally would under lossless environment. Before an EP is sent, we can estimate the expected prediction square error over a delayed and lossy network by different optimal DR schemes corresponding to different motion models as follows. For `rd`, we approximate the expected error $E[\theta_{cd}]$ resulting from DR scheme `cd` at receiver by conditioning on whether packet $x[n]$ sent at time nT arrives at $(n+D)T$; we denote by $p[D]$ the transmission probability that a packet is correctly delivered from sender to receiver in time interval $D*T$. If not, we assume packet $x[n-1]$ arrived in time interval $(D+1)T$; this is a good approximation when the probability of transmission failure $1-p[D]$ is small. Using (2), we write:

$$E[\theta_{cd}] \approx p[D]\sigma_w^2 D + (1-p[D])p[D+1]\sigma_w^2(D+1) \quad (14)$$

For `rv`, we can approximate the expected error $E[\theta_{cv}]$ similarly, except now we need to consider two packets, $x[n]$ and $x[n-1]$ for example, for each term. Using (4), we write:

$$\begin{aligned} E[\theta_{cv}] &\approx \prod_{i=0}^{1} p[D+i]\sigma_w^2 \sum_{i=1}^{D} i^2 \quad (15)\\ &\quad +(1-p[D])\prod_{i=1}^{2} p[D+i]\sigma_w^2 \sum_{i=1}^{D+1} i^2 \end{aligned}$$

For `ra`, we now need to consider three packets, each sent T seconds apart. Using (6), we write:

$$\begin{aligned} E[\theta_{ca}] &\approx \prod_{i=0}^{2} p[D+i]\sigma_w^2 \sum_{i=1}^{D} \alpha_2[i]^2 \quad (16)\\ &\quad +(1-p[D])\prod_{i=1}^{3} p[D+i]\sigma_w^2 \sum_{i=1}^{D+1} \alpha_2[i]^2 \end{aligned}$$

When an EP $y[n+D]$ is generated using the window of relevant updates $x[n],\ldots,x[n-l]$, we have two options to packetize $y[n+D]$, which we discuss next.

6.1 Piggyback Packetization

First, we can piggyback $y[n+D]$ into the same packet as $x[n]$; we call this *piggyback packetization of EP*. As mentioned in the Introduction, unlike other multimedia data packets like video, the size of a game update packet is very small, with the majority consumed by packet headers. By piggybacking EP $y[n+D]$ onto update packet $x[n]$, we can amortize the cost of packet header across two data. The drawback of piggyback packetization is that the loss of one piggyback packet will mean the loss of both update $x[n]$ and EP $y[n+D]$.

For motion model `rd`, piggybacking prediction $\hat{x}_{cd}[n+D]$ onto update $x[n]$ make no sense, because $\hat{x}_{cd}[n+D] = x[n]$.

For motion model `rv`, by piggybacking prediction $\hat{x}_{cv}[n+D]$ onto update $x[n]$, we can estimate the receiver's expected prediction square error $E[\Theta^P_{xv}]$ as follows:

$$\begin{aligned} E[\Theta^P_{xv}] &\approx p[D]\ \sigma_w^2 \sum_{i=1}^{D} i^2 + \quad (17)\\ &\quad +(1-p[D])\prod_{i=1}^{2} p[D+i]\ \sigma_w^2 \sum_{i=1}^{D+1} i^2 \end{aligned}$$

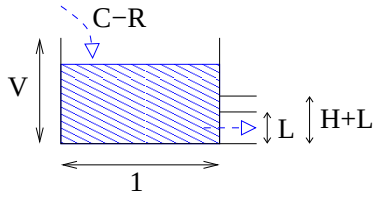

Figure 3: Token Bucket for Packetization and Scheduling of EPs at Sender

For motion model `ra`, by piggybacking prediction $\hat{x}_{ca}[n+D]$ onto update $x[n]$, we can estimate error $E[\Theta^P_{xa}]$ as:

$$\begin{aligned} E[\Theta^P_{xa}] &\approx p[D]\ \sigma_w^2 \sum_{i=1}^{D} \alpha_2[i]^2 + \quad (18)\\ &\quad +(1-p[D])\prod_{i=1}^{3} p[D+i]\ \sigma_w^2 \sum_{i=1}^{D+1} \alpha_2[i]^2 \end{aligned}$$

Note that within a piggyback EPP, differential coding can be used among data to reduce the size of the packet payload. We consider the effect secondary and did not pursue this direction in this paper, however.

6.2 Independent Packetization

Alternatively, we can send EP $y[n+D]$ separately as an independent packet immediately after $x[n]$ is sent. While this *independent packetization* induces the cost of a packet header, the loss of update $x[n]$ will not mean the loss of EP $y[n+D]$ because they are sent as separate packets.

For motion model `rd`, the expected error $E[\Theta^I_{xd}]$ is:

$$E[\Theta^I_{xd}] \approx (p[D] + (1-p[D])p[D])\ \sigma_w^2\ D \quad (19)$$

For motion model `rv`, the expected error $E[\Theta^I_{xv}]$ is:

$$E[\Theta^I_{xv}] \approx \left(p[D] + (1-p[D])\prod_{i=0}^{1} p[D+i]\right)\ \sigma_w^2 \sum_{i=1}^{D} i^2 \quad (20)$$

For motion model `ra`, the expected error $E[\Theta^I_{xa}]$ is:

$$E[\Theta^I_{xa}] \approx \left(p[D] + (1-p[D])\prod_{i=0}^{2} p[D+i]\right)\sigma_w^2 \sum_{i=1}^{D} \alpha_2[i]^2 \quad (21)$$

where in (19), (20) and (21) we again consider only event of receipt of the first packet, and event of loss of the first packet *and* receipt of subsequent window of relevant updates.

7. SCHEDULING OF EP

Given there are two methods to packetize EPs, in this section we discuss scheduling of EPs—when to use which method at sender, given extra bandwidth $C-R$, to minimize expected receiver's prediction square error. We first discuss how to optimize the selection of packetization methods in Section 7.1. We then discuss how the optimized selection is implemented in Section 7.2.

7.1 Optimizing Packetization Method

Aside from motion model `rd` where `xd` EPs must be sent as independent packets, the selection between piggyback and

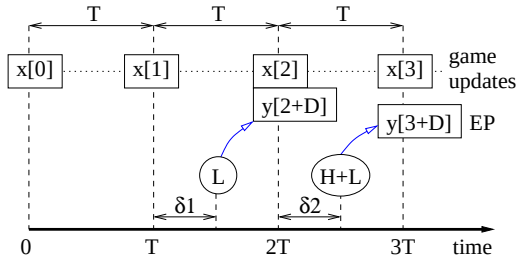

Figure 4: Example Scheduling of EPs at Sender

independent packetization involves a tradeoff between expected error and bandwidth usage. Piggyback packetization amortizes packet header among two data, but the loss of the same packet leads to two data losses, resulting in a high expected error. In contrast, the bandwidth cost of independent packetization is high due to the need of a packet header, but the independent delivery of updates and EPs lead to a lower expected error. In general then, the optimal packetization method is one that optimizes this tradeoff.

Let H be the size of a packet header and L be the size of the EP packet payload containing an EP. When we use independent (piggyback) packetization, given extra bandwidth $C - R$, the smallest period T^I (T^P) at which we can send EPs is:

$$\frac{1}{T^I} = \frac{C-R}{H+L} \qquad \frac{1}{T^P} = \frac{C-R}{L} \tag{22}$$

The resulting expected error of each packetization method is scaled by how often the packetization method can be used given its period and the game update period T. The optimization problem is to simply find the method with the smaller of the two resulting expected errors. For motion model `rv`, we can write:

$$\min_{m\in\{I,P\}} \left\{ \left(\frac{T^m - T}{T^m}\right) E[\Theta^m_{xv}] + \left(\frac{T}{T^m}\right) E[\Theta_{cv}] \right\} \tag{23}$$

for $T^m > T$. Same (23) can be used for motion model `xa`, with subscripts `xv` and `cv` replaced by `xa` and `ca`.

7.2 Scheduling Implementation

To implement the schedule in Section 7.1 such that excess bandwidth $C - R$ is maximally utilized, we implement the timed release of an EPP using a *token bucket*, shown in Figure 3. A continuous input flow of tokens of $C - R$ bps fills a bucket. When the volume in the token bucket reaches the size of an EPP (L or $H + L$, depending if piggyback or independent packetization is used), an EPP is permitted immediately; however, it is generated and sent at the next update transmission time. This is done so that the prediction $y[n + D]$ is sent as close to the most recent update $x[n]$ as possible to maximize its effectiveness. When an EPP is sent, volume L or $H + L$ of the bucket is drained.

An example usage of the token bucket for the scheduling of EPs is shown in Figure 4. Game updates $x[0], \ldots, x[3]$ are sent period T apart. At time $T + \delta_1$, a piggyback packet is permitted by the token bucket. Sender waits till time $2T$ before constructing prediction $y[2 + D]$ and sending it with $x[2]$ in the same packet. Volume L is then drained from the token bucket. Alternatively, at time $2T + \delta_2$, an independent packet is permitted. Sender waits till time $3T$ before constructing $y[3+D]$ and sending it as an independent packet. Volume $H + L$ is then drained from the bucket.

At receiver, based on arrived packets at time $(n + D)T$, update reconstructor essentially sends the best prediction $\hat{x}[n + D]$, whether from DR or from received EPs, to the application for scene rendering.

Table 1: DR Prediction Error for Different Avatar Motion Models (update rate = $10/s$)

	rd .05	rd .1	rv .05	rv .1	ra .05	ra .1
raw	4.83	4.95	5.71	6.28	7.53	10.97
EP-p	—	—	5.55	5.92	6.41	7.28
EP-i	4.51	4.55	4.63	4.74	4.76	4.98

8. SIMULATION EXPERIMENTS

We discuss the simulation experiments we performed to test the effectiveness of EPPs. We first detail the experimental setup in Section 8.1. We then present theoretical and game trace results in Section 8.2 and 8.3, respectively.

8.1 Setup

For the theoretical experiments, we use the following definition for random variable $w[n]$:

$$w[n] = \begin{cases} 0 & \text{with } Pr = p \\ U[0,S) & \text{with } Pr = 0.5(1-p) \\ -U[0,S) & \text{with } Pr = 0.5(1-p) \end{cases} \tag{24}$$

where $U[0, S)$ is the uniform random variable between 0 and S, where $S = 10$ is a pre-defined scaling constant. $w[n]$ is weighted so that $w[n] = 0$ with probability $p = 0.5$.

For $p[D]$, the transmission probability that a packet will be correctly delivered in interval $D * T$, we use:

$$p[D] = (1-q)\int_0^{DT} \Gamma(\tau)d\tau \tag{25}$$

where q is the packet loss probability and network delay $\Gamma(\tau)$ is the shifted Gamma distribution with parameters $(\kappa, \alpha, \lambda) = (50ms, 3, 0.1)$ and mean $80ms$.

For the game trace experiment, we first collected 10 minutes worth of game updates from FPS games `bzflag` [1] and `sauerbraten` [2]. The method used in capturing the updates is as follows. Two clients were actively participating in the FPS game over a low-delay, low-loss LAN. When periodic updates from `client2` reached `client1`, a packet filter was used at `client1` to capture the arriving update packets.

We assume packet header is 28 bytes and packet payload is 25 bytes for all experiments. We assume both x and y coordinates are generated, and the calculated *Euclidean distance* between actual and prediction location is used as metric of performance. Each data point is average over 100 trials of more than 1000 avatar locations each.

8.2 Theoretical Results

We first examined the performance of EPP when game updates were generated using the avatar motion models described in Section 3.1. Using update rate $10/s$ and bandwidth of $10kbps$, we calculated the average DR prediction error in Euclidean distance for no-EPP scheme (`raw`), scheme using EPP with piggyback packetization (`EP-p`) and scheme using EPP with independent packetization (`EP-i`) for differ-

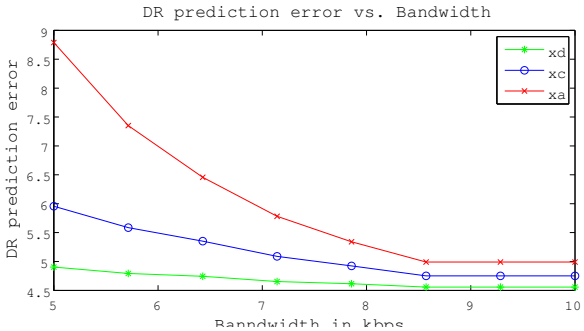

Figure 5: Prediction Error vs. Available Bandwidth for Different Motion Models

Table 2: DR Prediction Error for Different Avatar Motion Models (update rate = $15/s$)

	rd .05	rd .1	rv .05	rv .1	ra .05	ra .1
raw	6.44	6.54	10.00	10.72	15.74	21.51
EP-p	—	—	9.78	10.20	15.65	16.64
EP-i	6.12	6.17	8.77	8.97	14.60	14.78

ent combination of motion models and loss rates in Table 1.

We see that by using EPP, we can reduce prediction error by 8.2% for motion model `rd`, by 24.5% for `rv`, and by 54.6% for `ra`. We see also that independent packetization performed better than piggyback packetization for the experimental parameters we have chosen.

For the same experimental parameters, we also plotted the performance of `EP-i` as function of available bandwidth in Figure 5. We see that as bandwidth increased from $5kbps$ to $10kbps$, the prediction error decreased dramatically, particularly for `xa`. We also see the performance improvement trailed off after a certain point.

For the same experimental setup, we increased the update rate to $15/s$ and calculated the performance for the three schemes in Table 2. We see that the performance improvements for the three motion model `rd`, `rv` and `ra` over non-EPP scheme have decreased to 5.6%, 16.3% and 31.3%, respectively. We conjecture the following: as update rate increases, D increases for the same average network delay, and so it becomes more difficult for DR to predict an update further into the future.

8.3 Game Trace Results

We now compare the performance of EPPs using collected game traces. In Table 3, we see a comparison of the different methods for the two FPS game data sets for two different loss rates. We first observe that performances with EPP exceeded performance without EPP. We next observe that `xv` performed the best: up to 47.0% for `bzflag` and up to 57.6% for `sauerbraten` over non-EPP `raw`. This can be explained by the fact that motion model `rv` matches the movements in both game traces the best.

Table 3: DR Prediction Error for Real Game Traces using Different EPs (update rate = $10/s$)

	Bzflag .05	Bzflag .1	Sauer .05	Sauer .1
raw	13.47	14.04	3.39	3.57
xd	12.02	12.24	3.37	3.43
xv	**7.29**	**7.45**	**1.51**	**1.51**
xa	12.35	12.80	2.38	2.42

9. CONCLUSIONS

In this paper, we discuss the construction and scheduling of Extrapolated Parity Packets (EPPs) for network gaming over lossy networks with delays. Unlike like Reed Solomon $(n, n-1)$, EPPs can correctly convey prediction information even when multiple losses occur.

10. REFERENCES

[1] "bzflag," http://www.bzflag.org.
[2] "sauerbraten," http://sauerbraten.org.
[3] G. Armitage, M. Claypool, and P. Branch, *Networking and Online Games: Understanding and Engineering Multiplayer Internet Games*, John Wiley and Sons Ltd, 2006.
[4] L. Pantel and L. Wolf, "On the suitability of dead reckoning schemes for games," in *ACM SIGCOMM NetGames*, Braunschweig, Germany, April 2002.
[5] H. Holma and A. Toskala, Eds., *WCDMA for UMTS: Radio Access for Third Generation Mobile Communications*, Wiley, 2001.
[6] P. Frossard, "FEC performance in multimedia streaming," in *IEEE Communications Letters*, March 2001, vol. 5, no.3.
[7] S. Aggarwal, H. Banavar, and A. Khandelwal, "Accuracy in dead-reckoning based distributed multi-player games," in *ACM SIGCOMM NetGames*, Portland, OR, August 2004.
[8] M. Mauve, "How to keep a dead man from shooting," in *Interactive Distributed Multimedia Systems and Telecommunication Services*, 2000, pp. 199–204.
[9] M. Mauve, J. Vegel, V. Hilt, and W. Effelsberg, "Local-lag and timewarp: Providing consistency for replicated continous applications," in *IEEE Transactions on Multimedia*, February 2004, vol. 6, no.1, pp. 47–57.
[10] T. Beigbeder, R. Coughlan, C. Lusher, and J. Plunkett, "The effects of loss and latency on user performance in unreal tournament 2003," in *ACM SIGCOMM NetGames*, Portland, Oregon, August 2004.
[11] T. Yasui, Y. Ishibashi, and T. Ikedo, "Influences of network latency and packet loss on consistency in networked racing games," in *ACM SIGCOMM NetGames*, Hawthorne, New York, October 2005.
[12] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *ACM SIGCOMM*, Stockholm, Sweden, August 2000.
[13] S. Aggarwal, H. Banavar, S. Mukherjee, and S. Rangarajan, "Fairness in dead-reckoning based distributed multi-player games," in *ACM SIGCOMM NetGames*, Hawthorne, New York, October 2005.