Improving end-to-end quality-of-service in online multi-player wireless gaming networks

Preetam Ghosh*, Kalyan Basu, Sajal K. Das

Center for Research in Wireless Mobility and Networking (CReWMaN), Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, TX 76019-0015, USA

Abstract

In this paper, we present a novel, scalable multi-player gaming architecture that can incorporate mobile nodes. Next we propose an adaptive forward error correction (FEC) and rate-control technique to improve service quality in such a wireless gaming environment. We consider only last-hop wireless links in this paper. It is assumed that the game server and clients (mobile devices) can switch between different prediction levels having different data rates. We introduce a new scheme for estimating the packet loss rates due to congestion and wireless channel conditions and use this information in a cross-layer design to improve the overall service quality. The congestion packet loss probability is used to devise a simple TCP-friendly rate-control algorithm for sending downlink data packets from the game server. We also propose a novel adaptive FEC and dynamic packetization algorithm to alleviate the effects of wireless channel packet losses based on this chosen data rate. Extensive NS-2 simulation results show the efficacy of our scheme in achieving higher throughput.

1. Introduction

Online multi-player gaming poses an interesting application for wireless networks. Particularly, the so-called “first-person shooter” games are some of the most challenging interactive games to be implemented in a wireless network. Such games that involve one or more players taking some form of action against other players in the game, are not usually turn-based. As a result, game play is affected by latencies in the network: players with low latency can actually have an advantage over players exhibiting high latency in the network.

In wireless networks, the likelihood of large discrepancies between players compound the problem. This discrepancy results from the fact that players, particularly in cellular wireless networks, can exhibit a wide variance of latencies depending on their individual wireless channel conditions, mobile class and system load. For instance, a player near the center of a cell may experience much higher throughput than an user at the edge of a cell. Table 1 shows the latency requirements for a few popular games. It can be seen that latencies in the order of ≈200 ms can significantly affect the quality of interactive games. Though there exists a wealth of solutions for online gaming in wire-lined networks, extending the same concepts to wireless networks open up a new set of issues to be addressed. The dynamic fluctuations in available bandwidth of a wireless channel is one such issue that can significantly affect the quality-of-service (QoS) requirements for the underlying access mechanism. The heavy-tailed gaming traffic along with considerably low bandwidth availability (compared to the wire-lined counterpart) calls for some innovative changes in the transport and application-layer protocols to provide better QoS guarantees for online gaming over wireless networks.

To the best of our knowledge, there exists no previous work in the literature for improving quality of gaming specially for the wireless links. Additionally, the main goals of this paper is to provide new solutions with the following properties:

1. Adaptive to the changing network conditions;
2. Incurs minimal overhead in deployment. Ideally, our goal is to provide solutions that require changes in the client (gaming device) and game server softwares without affecting the intermediate nodes in the network;
3. Reduces the feedback overhead as much as possible. Our cross-layer design would require the receiver (mobile host, MH) to send feedback messages to the game server based on observed network conditions, and this feedback overhead in itself can undermine the benefits of adaptive QoS adjustment algorithms if not handled carefully.
4. Fast with low run-time complexity. Nowadays, the trend is to estimate the network conditions at the MH because it has a better view of the network and the wireless channel in...
2. Related works and motivation

The focus of multi-player networked gaming has shifted to support massively multi-player online games (MMPOGs). Griwodz Carsten [6] proposes a proxy-based architecture to separate the different styles of gaming traffic by defining levels of urgency and relevance for each traffic style to cope with scalability problems. A similar proxy-based architecture is also proposed in [2] to reduce the load on the central game server. A mathematical comparison of the average load on the central game server is given in [8] as a function of the number of users participating in the game. They show that the peer-to-peer architecture reduces the amount of load on the game servers heavily and also incurs low latency compared to a completely centralized server architecture. But the inherent problems of state inconsistency resolution becomes very difficult to handle for peer-to-peer architectures. They propose a peer-to-peer with central arbiter architecture model which acts as a hybrid between the two architectures and exploit the advantages of both.

Recently a lot of work is being done to show the detrimental effects of packet loss, delay and jitter both for conventional multi-player games and wireless gaming [3,18,20,21,26–28] through simulation studies. Some recent efforts are directed towards implementing test-bed setups for wireless gaming as well [22–24]. It is known that network latency plays the most important role in determining the user satisfaction in wireless gaming from a survey reported in [19].

Preliminary work towards improving QoS guarantees for networked mobile gaming is due to [7], wherein it has been shown that UMTS is better suited for interactive real-time gaming than GPRS because of overprovisioning problems that increases the delay and jitter for game packets considerably. They propose a combination of statistical multiplexing and QoS guarantees to aggregate multiple game flows and perform reservation for that aggregate. A few other wireless gaming architectures have also been proposed recently [31–33]. But leading game providers of today (e.g., Nokia) are implementing wireless gaming service on EGPRS which motivates us to consider the cross-layer design proposed in this paper. Also, it is important to be able to deploy the gaming services with minimum overhead so that we do not have to change the existing protocol stack at any of the intermediate nodes that are not participating in the game.

Moreover, the transport layer protocol used for gaming is UDP and reliable UDP (RUDP) [9]. RUDP is used to pass on important game state information, and UDP is used for lightweight game packets. Note that TCP is a heavy protocol, due to its complex congestion control algorithms and byte-oriented window scheme, and hence it is difficult to support many concurrent users with this protocol. Furthermore, in TCP, even moderate congestion makes the game data unusable, so it is a particularly inefficient protocol for MMPOGs. On the other hand, UDP is a connectionless protocol that runs on top of the IP layer. Unlike TCP, UDP provides very few error recovery services, offering instead a direct way to send and receive data over the IP layer. However, because these two transport protocols are implemented in almost all network devices, it is easy to deploy new applications using them. RUDP somewhat alleviates the error recovery constraints of UDP by implementing a one-time acknowledgement scheme. The UDP/RUDP combination is hence the default standard for commercial wireless gaming infrastructures [4]. New protocols like the Game Transport Protocol (GTP) [30] are not that popular and hence commercially not viable.

So, the existing end-to-end QoS approaches for supporting multimedia traffic is not suited for wireless gaming. The game traffic being essentially heavy-tailed follow an extreme value distribution [1,5] which calls for new ways of handling QoS for game traffic.

2.1. Our contribution

Our contribution in this paper can be summarized as follows. First, we present a generalized proxy server based game architecture model that can handle both central and peer-to-peer server games. This architecture complies with the one outlined in [4] developed by Sega and to be deployed by Nokia as the first wireless gaming architecture to be functional on EGPRS. Next we propose a new scheme for differentiating between packet losses due to congestion and wireless channel characteristics at the receiver. Our scheme is similar to the one proposed in [11] with a slight modification to increase the efficacy of wireless channel state and congestion loss prediction for the next RUDP message interval. Based on the packet error probability (PER) for congestion losses and wireless channel error, we devise a TCP-friendly rate-control algorithm. TCP-friendliness ensures that the other data streams in the Internet are not unduly penalized during network congestion due to a constantly high transmission rate of UDP game traffic. The central idea is that for multi-player gaming we can have different prediction levels based on dead reckoning, Area of Interest management (AoI) and selective transmission schemes [10]. In fact recent works on game-specific mobility models [25] can actually improve the dead reckoning based prediction schemes significantly. This would essentially reduce the number of UDP game packets to be sent.

<table>
<thead>
<tr>
<th>Real-time strategy (RTS)</th>
<th>250–500 ms (playable)</th>
<th>&gt;500 ms (noticeable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>First-person shooter (FPS)</td>
<td>&lt;150 ms (preferred)</td>
<td>X</td>
</tr>
<tr>
<td>Car racing</td>
<td>&lt;100 ms (preferred)</td>
<td>100–200 ms (sluggish)</td>
</tr>
</tbody>
</table>

Table 1: Latency requirements of a few popular games

Preliminary work towards improving QoS guarantees for networked mobile gaming is due to [7], wherein it has been shown that UMTS is better suited for interactive real-time gaming than GPRS because of overprovisioning problems that increases the delay and jitter for game packets considerably. They propose a combination of statistical multiplexing and QoS guarantees to aggregate multiple game flows and perform reservation for that aggregate. A few other wireless gaming architectures have also been proposed recently [31–33]. But leading game providers of today (e.g., Nokia) are implementing wireless gaming service on EGPRS which motivates us to consider the cross-layer design proposed in this paper. Also, it is important to be able to deploy the gaming services with minimum overhead so that we do not have to change the existing protocol stack at any of the intermediate nodes that are not participating in the game.

Moreover, the transport layer protocol used for gaming is UDP and reliable UDP (RUDP) [9]. RUDP is used to pass on important game state information, and UDP is used for lightweight game packets. Note that TCP is a heavy protocol, due to its complex congestion control algorithms and byte-oriented window scheme, and hence it is difficult to support many concurrent users with this protocol. Furthermore, in TCP, even moderate congestion makes the game data unusable, so it is a particularly inefficient protocol for MMPOGs. On the other hand, UDP is a connectionless protocol that runs on top of the IP layer. Unlike TCP, UDP provides very few error recovery services, offering instead a direct way to send and receive data over the IP layer. However, because these two transport protocols are implemented in almost all network devices, it is easy to deploy new applications using them. RUDP somewhat alleviates the error recovery constraints of UDP by implementing a one-time acknowledgement scheme. The UDP/RUDP combination is hence the default standard for commercial wireless gaming infrastructures [4]. New protocols like the Game Transport Protocol (GTP) [30] are not that popular and hence commercially not viable.

So, the existing end-to-end QoS approaches for supporting multimedia traffic is not suited for wireless gaming. The game traffic being essentially heavy-tailed follow an extreme value distribution [1,5] which calls for new ways of handling QoS for game traffic.

2.1. Our contribution

Our contribution in this paper can be summarized as follows. First, we present a generalized proxy server based game architecture model that can handle both central and peer-to-peer server games. This architecture complies with the one outlined in [4] developed by Sega and to be deployed by Nokia as the first wireless gaming architecture to be functional on EGPRS. Next we propose a new scheme for differentiating between packet losses due to congestion and wireless channel characteristics at the receiver. Our scheme is similar to the one proposed in [11] with a slight modification to increase the efficacy of wireless channel state and congestion loss prediction for the next RUDP message interval. Based on the packet error probability (PER) for congestion losses and wireless channel error, we devise a TCP-friendly rate-control algorithm. TCP-friendliness ensures that the other data streams in the Internet are not unduly penalized during network congestion due to a constantly high transmission rate of UDP game traffic. The central idea is that for multi-player gaming we can have different prediction levels based on dead reckoning, Area of Interest management (AoI) and selective transmission schemes [10]. In fact recent works on game-specific mobility models [25] can actually improve the dead reckoning based prediction schemes significantly. This would essentially reduce the number of UDP game packets to be sent.
from the game server to the mobile clients, thus reducing the UDP data rate. The RUDP data rate is however fixed and cannot be changed as all RUDP messages have to be delivered to the MH. We assume that the RUDP messages from the game server to the MH are sent at regular time intervals which is typically 40 ms for interactive games (this includes the keep-alive messages in case a gamer is not active). Even for variable RUDP intervals, our schemes will work as long as the intervals are not too small. Thus, we can change the UDP data rate based on the estimated congestion PER for the next RUDP interval. Based on this chosen rate, we next devise an adaptive FEC and packetization technique that would maximize the throughput for UDP game packets. Reed–Solomon (RS) codes have been widely used to facilitate application-layer FEC for streaming multimedia traffic. Thus, based on the PER for wireless channel conditions, our adaptive scheme will choose the optimal FEC and transport layer packet size for UDP traffic that maximizes the UDP throughput. We also consider delay sensitive UDP traffic in our algorithms, to present a generalized solution that can handle real-time interactive games. The maximum latency that the game can support is a system parameter to be fixed by the game developers and is an input to our schemes. The loss differentiation, rate control and adaptive FEC and packetization schemes are all implemented at the receiver (i.e., MH) to reduce the cross-layer messaging overhead as will be discussed later on.

3. Proxy-based game architecture model

We use the Sega Network Application Package (SNAP) [4] as the primary case study for the applications of QoS guarantees in a multi-user online gaming environment. SNAP is an application developers’ environment that provides for an efficient means of implementing multi-player online gaming. It recommends a distributed server architecture, which allows for widespread deployment of the gaming network without putting too much burden on the game developer. Its underlying transport protocol is UDP, but allows for reliable transport via a proprietary protocol running on top of UDP known as RUDP (reliable UDP).

Our distributed server architecture allows for easily scalable deployments of gaming applications and can be useful in cellular networks as well. The distributed server elements (known as proxy servers) provide a means for gaming clients to communicate with one another and to one or more central game servers. The concept is depicted in Fig. 1. Note that the proxy servers can be easily added to the gaming network as necessary.

Our game architecture can achieve both the scalability properties of the peer-to-peer model (PP), and use the simple consistency resolution mechanism of the client–server (CS) model. For a simple CS implementation, the game server will maintain the complete game state information and send updates to the MHs through the proxy servers that will function as routers in the network. To implement a PP model, the proxy servers can individually maintain the game state information and send state update messages between themselves without going through the game server. The peer-to-peer with Central Arbiter model (PP-CA) [8] can also be supported with our distributed server architecture. In PP-CA, players exchange updates communicating directly with each other, just as in the PP model. This minimizes the communication delays between players. Each player sends its updates not only to all other players, but also to the central arbiter. The role of the central arbiter is to listen to all player updates, simulate the global state of the game, and detect inconsistencies. In the absence of inconsistencies, the central arbiter remains silent, without sending any messages to the players. When an inconsistency is detected however, the central arbiter will resolve it, create a corrected update, and transmit that update to all players. The corrected players should then rollback to the previous accepted state. The consistency resolution protocol in PP-CA is basically the same with that in the CS architecture. The key difference between the CS server and the PP-CA arbiter is that the former sends a global state update to each player in every player update period, while the latter sends a corrected update to each of the players only when an inconsistency occurs. If inconsistencies are rare events, the bandwidth requirement at the PP-CA arbiter will be significantly lower than the bandwidth requirement at the CS server. In our architecture, the game server can act as the central arbiter and the proxy servers act as the peer-to-peer servers.

Although the SNAP environment provides for ease in authentication, chat applications, and the setting up of game rooms and lobbies, the feature in SNAP that impacts QoS most directly is its transport protocol (RUDP/UDP). Specifically, RUDP provides for a means of guaranteed delivery of packets. It allows also for in-sequence delivery of packets generated from a client or server. However, its bandwidth management mechanism allows for throttling the throughput generated by a client or server based on a fixed parameter, i.e., the available bandwidth, that is fixed before the start of game play. But, in a cellular environment the bandwidth is not only variable from client to client but also variable over time for any individual client.

In our proxy-based architecture, the proxy servers acting as the proxies are located closely to clusters of remote users. The proxy servers communicate with each other and the central server. This has the advantage of decentralizing game play to reduce the load on the central server, but centralizing necessary functions for game play such as authentication. However, no method is proposed in the SNAP for ensuring QoS guarantees among users who are connected to the proxy servers. It is assumed that such users all exhibit the same latency and packet loss probabilities which may not be the case in wireless (particularly cellular) environments. The SNAP environment is access-independent. However, this does not prevent from the development of suitable access-dependent QoS-guaranteeing algorithms within the SNAP architecture. Thus, the QoS guarantee framework that we develop is dependent on the underlying access mechanisms to ensure that relevant parameters affecting the network state can be estimated accurately.

The proposed architecture can support all three proposed gaming architecture models along with the following benefits:

1. It is highly scalable. With increase in the player base, we can add more proxy servers in the network under a common game server.
2. The transport layer protocol used is UDP/RUDP that is compliant with the SNAP and also Nokia’s wireless gaming model.

![Fig. 1. Proposed server architecture.](image-url)
The QoS improvement protocol can be implemented at either the proxy servers or the game server depending on the architecture model that needs to be adopted based on the popularity of the game (i.e., maximum number of players that have to be supported).

The QoS framework discussed in this work is based on the CS model, however it can be easily implemented in both the PP and PP-CA models depending on where the actual game state information is maintained.

4. The QoS guarantee model

Our intention in this section is to devise a TCP-friendly Gaming Protocol for Wireless Internet called TFGWP. Due to network congestion, the gaming application should be able to alter its data rate such that it can fairly share bandwidth with other applications, and also maintain a suitable packet loss level that will not significantly affect the gaming experience of the user. Also, based on the state of the wireless channel, the protocol should be able to apply proper error correction to improve the gaming quality. Thus the basic features of TFGWP can be summarized as follows:

1. Differentiating packet losses due to congestion and wireless channel error. We follow the concept proposed in [11] to use link layer information to identify the packet losses. TFGWP however uses a hybrid approach by combining the ideas of [11,12] and uses the sequence number of the packets to appropriately distinguish the packet loss rate due to congestion and that due to the wireless channel.

2. Estimating the packet loss probability for the next RUDP interval. Two Gilbert models are used to estimate the packet loss probabilities for the next RUDP interval. The Gilbert models help in modeling the loss burstiness lengths due to congestion and that due to the wireless channel.

3. Rate control. Based on the forward congestive packet loss ratio, we devise an additive-increase additive-decrease (AIAD) rate-control algorithm that would switch the sender/receiver pair to a higher prediction level, i.e., a lower UDP data rate if congestion losses are high, and vice versa. Note that, because the normal game packets are sent through UDP protocol, a TCP-like AIMD (additive-increase–multiplicative-decrease) scheme is inappropriate over here. Also, the rate-control algorithm runs at the application layer and even with an AIAD scheme we can sufficiently throttle the traffic to avoid network congestion. However, this will depend on the difference of the UDP data rates at each prediction level as discussed later.

4. Adaptive FEC and packetization. The forward wireless channel PER is used to choose an optimal FEC level packet size at the transport layer that maximizes the throughput of UDP traffic, such that the resulting data rate does not exceed the one specified by the rate-control algorithm. Increasing the FEC level will increase the corresponding redundancy in UDP packets, and the UDP data rate has to be altered accordingly.

A general overview of TFGWP is presented in Fig. 2. Thus, TFGWP computes some optimal parameters for both the transport and application layers. In the following subsections, we describe the basic functionalities and processes of TFGWP in details.

---

![Fig. 2. A general overview of the TFGWP protocol.](image-url)
4.1. Role of game server and receiver (MH)

The TFGWP module is implemented at the proxy server (which sends prediction level change messages to the application layer in the game server) and the receiver (MH). The proxy is generally located in the wired network and delivers UDP game packets at a certain rate. It also sends RUDP packets to the receiver periodically (typically 40 ms). The receiver measures the incoming UDP packets for the entire RUDP interval and sends feedback to the sender by piggybacking it with the ACK message for the corresponding RUDP packet. Based upon the feedback from the receiver, the game server adjusts its transmission rate in a TCP-friendly manner by switching to the required prediction level. The corresponding FEC level and the UDP packet size for the next RUDP interval are also chosen by the proxy. Thus our receiver based protocol has the following steps:

1. estimating loss rate (both congestion losses and wireless channel losses);
2. estimating the available network bandwidth and choosing the best prediction level;
3. choosing the optimal FEC level and packet size and also the corresponding prediction level required to support the redundancy introduced by the FEC;
4. the chosen prediction level, FEC level and packet size are then fed back to the proxy server (piggybacked with the ACK packet) which in turn takes the necessary steps to adjust to the network conditions.

Obviously, implementing the entire logic at the receiver has its own drawbacks. For thin clients the scheme might not prove beneficial. So, our focus is to simplify the calculations at the receiver as much as possible. The algorithms, if implemented at the sender, would require additional feedback messages. Both the proxy (and game server) and the receiver have to know the chosen prediction level to switch to. The prediction is actually done at the receiver to estimate the game state when the game server is sending lesser packets among the erroneous packets by taking a weighted proportion of correct packets. If there are $v-1$ erroneous packets and $u$ lost packets between two adjacent successfully received packets, we will have $v$ time intervals among the erroneous and correctly received packets as follows: $a_1, a_2, \ldots, a_v$. Yang et al. [11] assumes that adjacent packets with larger time interval may contain more lost packets and hence identifies the congestive loss packets among the erroneous packets by taking a weighted proportion of these intervals. But this might not be the best way to identify the congestion loss packets because in general the number of congestive packet losses are skewed being more at higher $a_i$'s and a proportional division might not give good results.

The Biaz scheme [12] uses packet inter-arrival time to differentiate between loss types. We propose a hybrid scheme between Biaz and the one proposed in [11] to identify the congestive loss packets in interval $a_i (i=1, \ldots, v)$. Let $T_{\text{min}}$ denote the minimum packet inter-arrival time observed so far by the receiver during the connection. Now, if $a_i \approx T_{\text{min}}$, we do not have any congestion losses. Whereas, if $a_i \gg T_{\text{min}}$ we should have many congestion losses. Thus, we first exclude the $a_i$'s that are nearly equal to $T_{\text{min}}$ because there should not be any congestive packet losses in these intervals. Next, we choose $a_{\text{max}}$ from the remaining $a_i$'s. The number of congestive packet losses in interval $a_{\text{max}}$ is calculated as the maximum value of $k$ that satisfies $(k+1)T_{\text{min}} \leq a_{\text{max}}$. We do this iteratively until all the $u$ congestion losses are accounted for. The corresponding algorithm is shown in Fig. 4. The packet inter-arrival times are continuously monitored at the MH and $T_{\text{min}}$ is made

![](Fig. 3. Gilbert models for packet losses due to the wireless channel state and congestion.)
Algorithm-ICPL (Identify Congestion-related Packet Losses)

1. for \(i=1,\ldots,v\)
2. remove \(a_i\) from the list of intervals if \(a_i \approx T_{\text{min}}\)
3. end of for loop
4. While \(u \neq 0\)
5. Choose \(a_{\text{max}}\) from the remaining list of \(a_i\')s.
6. Number of congestion packet losses in this interval
   \[= c_{\text{max}} = \max_{k \in \{k\mid k+1 \in \text{set} \}} \{b_{\text{min}} \leq \sigma_{\text{max}}\} \]
7. \(u = u - c_{\text{max}}\)
8. end of While loop

Fig. 4. Algorithm to identify congestive packet losses within the \(a_i\')s.

equal to the current minimum value. The scheme will fail to give good results towards the beginning of the gaming session when we do not have a good estimate of \(T_{\text{min}}\). To alleviate that condition, we can use the algorithm proposed in [11] at the beginning of the game session and then switch to Algorithm ICPL afterwards, when we have a good estimate of \(T_{\text{min}}\). Also, the condition \(a_i \approx T_{\text{min}}\) can be implemented by using thresholds. Ideally, we should check if \((100-t)\% \leq T_{\text{min}} < (100-t)\% + T_{\text{min}}\), where \(t\) gives the threshold percentage. Through NS-2 simulations we have found \(t\approx 10\%\) gives the best results. A point to note is that, we might have \(u > 0\) after allocating the congestion losses at each of the \(v\) intervals. We would then follow the scheme proposed in [11] to allocate the remaining losses.

Thus, loss detection phase identifies \(p_c\) and \(p_w\) which are the estimated congestive PER and wireless channel PER for the next RUDP interval and are used by our adaptive rate control, FEC and packetization control algorithms.

4.3. Game traffic model

Extensive simulations in [1] show that game traffic is essentially heavy-tailed following an extreme value distribution. A mathematical representation of game traffic follows the shifted Lognormal or shifted Weibull distributions the CDF and PDF for which is given as follows:

\[
F(x) = e^{-x^2}; \quad f(x) = \frac{1}{r}e^{-x^2}e^{-\frac{x^2}{r}}; \quad r > 0
\]

We assume that the different mean and peak rates for the different prediction levels of game traffic are known initially to the MH. This can be achieved during client logging into the gameroom phase. Once \(p_c\) and \(p_w\) are known, we can calculate the total packet loss probability \(p\) as \(p = (1 - p_c)(1 - p_w)\). Suppose, the game is at prediction level \(i\) for the MH in question. We can estimate the effective bandwidth, \(EB_i\), at prediction level \(i\) required by the application following Kelly’s formula [17] as follows:

\[
EB_i = q \cdot m_i \left(1 + 3z \left(1 - \frac{m_i}{m_0}\right)\right) \quad \text{if } 3z < \min \left\{\frac{h_i}{m_i}, 3 \frac{h_i}{m_0}\right\}
\]

\[
= q \cdot m_i \left(1 + 3z^2 \left(1 - \frac{m_i}{m_0}\right)\right) \quad \text{if } 3z^2 < \frac{h_i}{m_i}
\]

\[
= q \cdot h_i \quad \text{otherwise}
\]

where, \(q = 1 - \frac{\log 100}{\log 0.1}, \quad z = 2 \log 10 \cdot p_c, \quad m_i = \text{mean rate at prediction level } i, \quad h_i = \text{peak rate at prediction level } i, \quad c = \text{actual channel rate} = B_{\text{max}}\) (i.e., maximum wireless channel bandwidth). The last-hop wireless link is considered to be the bottleneck link and hence the maximum available channel rate is given by \(B_{\text{max}}\). Also, we simply use the congestion PER \(p_c\) in the calculations, because the rate-control algorithm just needs to look into \(EB_i\) and \(p_c\). If the wireless channel conditions degrade leading to an increase in \(p_w\), the UDP data rate does not have to be reduced by switching to a lower prediction level because the adaptive FEC would try to reduce the wireless packet losses. Also, using \(p\) instead of \(p_c\) in the calculation of \(EB_i\) would result in sub-optimal usage of the wireless link which is a poor design. Thus \(p_c\) alone should decide \(EB_i\), which is then fed into the rate-control algorithm to choose the optimal prediction level for the next RUDP interval.

With different prediction schemes possible at the application layer, the traffic characteristics can change depending on how the game server decides to throttle the UDP data rate. This is basically decided by the game developers and can be of the following types:

1. The game server only sends a fixed number of UDP packets every RUDP interval. This would generate a constant UDP data rate from the transport layer perspective and makes our analysis of adaptive FEC and packetization design easier. The effective bandwidth calculation formula still remains the same but with \(m_i = h_i\).

2. The game server drops \(x\) number of packets for every \(y\) number of packets to be sent to the MH. This essentially keeps the traffic model heavy-tailed but with different mean and peak rates.

We assume that the different mean and peak rates for the different prediction levels are known to both the game server and MH at the beginning of the gaming session, and \(EB_i\) can be easily calculated once \(p\) is known for the next RUDP interval.

4.4. Examples of estimating the prediction levels

Before presenting the rate-control algorithm we present a formal definition of the concept of prediction levels here. The examples discussed here consider application of simple cases of dead reckoning and AoI schemes to car-racing games and serve for illustrative purposes only.

1. Effect of dead reckoning on the data rate. We use a simple first order model for estimating a car’s velocity (speed and direction). Suppose, the MH (car 1) tries to predict the positions of the other cars (MHs) in the game based on the last two messages received from the game server regarding car \(j\). The corresponding \(X\) and \(Y\) axes coordinates for these two messages for car \(j\) are denoted by \((x_j, y_j)\) and \((x'_j, y'_j)\) sampled at times \(t_j\) and \(t'_j\), respectively (these timestamps are also generally sent to car 1). Thus the predicted velocities of car \(j\) in the \(X\) and \(Y\) directions (denoted by \(v'_x\) and \(v'_y\), respectively) are as follows:

\[
v'_x = \frac{x'_j - x_j}{t'_j - t_j}; \quad v'_y = \frac{y'_j - y_j}{t'_j - t_j}
\]

Then, if no messages arrive at the MH (car 1) till time \(t_a\), it can predict the positions of car \(j\) till time \(t_a\) as follows:

\[
x'_{\text{est}} = x'_j + (v'_x \ast (t_a - t'_j)); \quad y'_{\text{est}} = y'_j + (v'_y \ast (t_a - t'_j))
\]

Similar estimates can be made for the other MHs (i.e., cars) as well at car 1. Note that based on this prediction scheme at the MHs, the game server can take two strategies of throttling traffic:

- Send \(m\) messages out of every \(N\) messages (where \(N\) is the number of prediction levels supported and \(m \leq N\)). This essentially follows the concept of point 2 in the previous subsection and keeps the game traffic heavy-tailed. Thus the effective data rate can be expressed by \(\text{actual data rate} = 1 - m \leq N\). Note that as the mean and peak rates are known a priori, the new traffic distribution can also be easily estimated by the new values of mean and peak rates as follows:
new mean data rate = \frac{\text{actual mean data rate}}{m},
\text{new peak data rate} = \frac{\text{actual peak data rate}}{m},
\end{equation}

- The game server decides to send \( m \) messages per second, that would result in a constant data rate. Thus if the mean data rate is given by \( \mu \), then the data rate corresponding to prediction level 1 (lowest prediction) is equal to \( \mu \). That corresponding to level \( i \) (\( 1 \leq i \leq N \)) is given by: \( \mu \times (N - i + 1)/N \). Note that the variable \( m \) can be easily related to the data rate at the \( i \)th level as:
\[ m = \mu \times (N - i + 1)/N. \]

(2) Effect of AoI on the data rate. For a simple example of AoI schemes, we define a circular region of interest around a car. Updates on other cars inside that region of interest should be communicated to the original car. The other cars (outside this area) do not affect the movement of the car under consideration and updates on their movements do not have to be transferred. Thus if the mean data rate to car 1 be denoted by \( \mu \), we have \( \pi \times (r1)^2 = \mu \), where \( r1 \) is the radius of the AoI circle at prediction level 1. For each prediction level, the radius is reduced by a step size of \( \frac{r1}{2} \) to reduce the data rate. Assuming, that the minimum radius is 50% of the original radius, we have \( rN = r1/2 \), and data rate at prediction level 1 is given by:
\[ \text{mean data rate at level } i = \pi \times (r1 - (i - 1) \times \frac{r1 - rN}{N - 1}). \]

We next show in Fig. 5 the variation of the mean data rate on number of prediction levels based on the above examples. Note that following [1], the mean data rate is computed as 2.05 kbps with a mean inter-arrival time of 62 ms and mean packet size of 127 Bytes. Dead Reckoning Model 1 is most rigorous and AoI scheme is the least rigorous in reducing the data rate with the Dead Reckoning Model 2 scheme lying in between the two. However, it should be noted that Dead Reckoning Model 1 and AoI schemes do not change the nature of the game traffic, whereas Model 2 achieves a constant data rate.

4.5. Rate control

As discussed in the previous section, the objective of our rate-control scheme is to make TFGWP TCP-friendly, i.e., it should switch to a higher prediction level (resulting in lower UDP data rate) if \( p_c \) increases. Because we can only work with a fixed number of prediction levels, we devise a simple additive-increase additive-decrease rate-control scheme in this paper. Let us suppose that there are \( N \) possible prediction levels for the game under consideration. Again, \( N \) is known to the MH during login phase. The idea is to choose among these \( N \) different prediction levels based on the effective bandwidth \( EB \). As discussed before, the data rates of the individual prediction levels can vary greatly depending on the amount of predictive intelligence implemented in the game. Thus, our scheme still remains TCP-friendly because switching to a higher level does not necessarily additively decrease the UDP data rate. For example, suppose there are three prediction levels with mean UDP data rates as 40, 20, 10 kbps, respectively. Switching from level 1 to level 2 actually decreases the data rate by 50%. Note that the examples of determining the prediction levels shown before in general reduces the data rate appreciably. However, if the difference in the average data rates between successive prediction levels is small, the TCP-friendliness of our algorithm will decrease. It is however possible to reduce the data rates more aggressively in such cases by switching to the \( n^\text{th} \) next higher level \( n = \text{currentLevel} + 1, \ldots, N \). The right value of \( n \) can be determined through simulations and will depend on the game traffic (i.e., the game type) and the data rates of the individual prediction levels and are not reported here. The basic algorithm is shown in Fig. 6.

The main idea of Algorithm Rate Control is to compare the estimated congestive PER, \( p_c \), with the current congestive PER, \( p_c^\text{current} \). If \( p_c \) increases beyond a threshold, we should reduce the UDP data rate. Otherwise, we can infer that the congestion has stabilized, and increase the data rate (i.e., switch to a lower prediction level). Obviously, prediction level 1 offers the highest UDP data rate and level \( N \) offers the lowest one. The threshold \( th \) has been chosen to be 0.2 through simulations. The variable \( tempRate \) stores the current effective bandwidth. Thus when the game switches to a higher prediction level, \( EB_{\text{higherLevel}} \) is calculated based on \( p_c \) (not \( p_c^\text{current} \)) and the corresponding mean and peak rates of the next higher level. When \( p_c \) does not increase, instead of increasing the data rate straight-away, we use an exponential averaging technique with the parameter \( \alpha = 0.8 \). The idea is to smooth out the changes in prediction levels with \( (1 - \alpha) \) acting as the exponential decaying factor. Thus only if the congestive PER does not increase for a sufficiently long period of time, we increase the UDP data rate.

\begin{algorithm}
\caption{Rate Control}
1. If \( p_c > p_c^\text{current} + \alpha \times p_c^\text{current} \)
2. switch to next higher prediction level if not already at level \( N \).
3. \( tempRate = EB_{\text{higherLevel}} \)
4. \( p_c^\text{current} = p_c \)
// end of If part
5. else
6. calculate \( EB_{\text{currentLevel}} \) based on \( p_c \)
7. \( tempRate = E \times tempRate + (1 - \alpha) \times EB_{\text{currentLevel}} \)
8. if \( tempRate > EB_{\text{lowerLevel}} \)
9. switch to next lower prediction level if not already at level 1.
10. \( tempRate = EB_{\text{lowerLevel}} \)
// end of inner If loop
// end of else part
\end{algorithm}
4.6. Adaptive FEC and packetization scheme

Once the prediction level has been specified by Algorithm Rate Control, we concentrate on increasing the throughput of UDP game traffic through our adaptive FEC and packetization techniques. This module of TFGWP tries to mask the effects of wireless channel PER, i.e., \( p_w \), by optimally choosing between the different RS-codes available at the proxy and MH for decoding and the optimal packet size at the transport layer. The redundancy introduced by the FEC design can in effect increase the data rate of UDP traffic if we keep the prediction level same as chosen by the rate-control algorithm. Again, switching to a higher prediction level reduces the UDP data rate and increases the option for choosing a higher FEC level (i.e., with higher redundancy). We thus have a trade-off to consider between the FEC level and the prediction level. Therefore our goal in this section is to maximize the overall throughput of UDP traffic by choosing the optimal FEC and prediction levels, and also the transport layer packet size.

Two different types of FECs are generally used at the application layer to mitigate the effects of \( p_w \). For example, media dependent FECs are used when there is high correlation between the successive data packets, and media independent FECs are used when the data packets are individually coded with block codes like RS-codes. Refs. [13] and [11] show the effectiveness of RS-codes for application layer FEC and hence it is the chosen standard for streaming multimedia applications. Wireless gaming requires similar real-time guarantees and so we propose to use RS-codes at the application layer to increase the successful packet reception probability. One important metric for game traffic is the maximum latency, \( D \), that the UDP game packets can be subjected to, for ensuring smooth gaming quality. \( D \) is again a system parameter provided to the MH at the beginning of the game session. An \((n,k)\) RS-code contains \( k \) data packets in an \( n \) packet block of which the rest \( n-k \) packets are redundant packets. Ideally these \( n-k \) redundant data packets are piggybacked on to the next \( n-k \) normal UDP game packets. Now, any packet error detected at the receiver can only be resurrected if any \( k \) of this block of \( n \) packets are received correctly. Hence, in the worst case, we will have to wait for all the \( n \) packets of the block to be able to correct all the errors. This, however, increases the delay for retrieving the lost packet and hence we have a delay-FEC level trade-off.

Rosenberg et al. [13] derives an expression for the probability of receiving a packet on time (i.e., within the delay bound \( D \)), \( p_R \), as:

\[
p_R = (1-p) \cdot P[n_i \leq D] + (1-(1-p))P[n_i \leq D]
\]

\[
\times \frac{1}{k} \left( \sum_{q=k+1}^{n} (S(g-k) - S(g))P[X_2(g)] + \sum_{i=1}^{n} S(n-j+1)P[X_2(n)] \right)
\]

where:

\[
P[X_2(g)] = \sum_{r=k}^{g-1} \binom{g-1}{r} (1-p)^r p^{g-1-r},
\]

\[
S(j) = P[n_i < D - j],
\]

and \( n_i \) is the actual delay experienced by packet \( i \) to reach the MH from the time of its generation. Eq. (7) calculates \( p_R \) for a given RS-code \((n,k)\), delay bound \( D \) and a fixed data rate at the transmitter where the packets are generated at fixed intervals of \( \Delta \).

The heavy-tailed game traffic however prevents us to use Eq. (7) directly for our analysis, because the inter-arrival times between packets is not constant any more. Ideally, we should use a \( j \)-fold convolution of the inter-arrival times of the traffic distribution PDF and calculate the mean of the corresponding general distribution to estimate the average inter-arrival time between packets.

But, the extreme value traffic distribution can not be integrated in the interval \((0, \infty)\) and hence, we have to use approximations. We assume that the \( j \)-fold convolution of the inter-arrival times follow a gamma-distribution such that for prediction level \( l \), the CDF and PDF are given by:

\[
CDF : F_j^X(x) = P(X \leq x) = 1 - e^{-mX} \sum_{k=0}^{j-1} \frac{(mX)^k}{k!}
\]

\[
PDF : f_j^X(x) = \frac{m_i(mX)^{j-1}e^{-mX}}{(j-1)!}
\]

The mean of this distribution is given by \( \frac{1}{m} \). Let \( s_i \) denote the average packet size generated by prediction level \( i \), \((i=1, \ldots, N)\). Also, let \( L \) denote the transport layer packet size (hence, \( \frac{L}{m} \) denotes the packetization rate). Thus, the actual inter-arrival time between game packets at prediction level \( i \) is given by:

\[
D = s_i + \frac{1}{m} \frac{L}{s_i} = \frac{1}{m} \frac{L}{s_i}
\]

The mean of this distribution is given by \( \frac{1}{m} \). Let \( s_i \) denote the average packet size generated by prediction level \( i \), \((i=1, \ldots, N)\). Also, let \( L \) denote the transport layer packet size (hence, \( \frac{L}{m} \) denotes the packetization rate). Thus, the actual inter-arrival time between game packets at prediction level \( i \) is given by:

\[
D = s_i + \frac{1}{m} \frac{L}{s_i} = \frac{1}{m} \frac{L}{s_i}
\]

The mean of this distribution is given by \( \frac{1}{m} \). Let \( s_i \) denote the average packet size generated by prediction level \( i \), \((i=1, \ldots, N)\). Also, let \( L \) denote the transport layer packet size (hence, \( \frac{L}{m} \) denotes the packetization rate). Thus, the actual inter-arrival time between game packets at prediction level \( i \) is given by:

\[
D = s_i + \frac{1}{m} \frac{L}{s_i} = \frac{1}{m} \frac{L}{s_i}
\]

The mean of this distribution is given by \( \frac{1}{m} \). Let \( s_i \) denote the average packet size generated by prediction level \( i \), \((i=1, \ldots, N)\). Also, let \( L \) denote the transport layer packet size (hence, \( \frac{L}{m} \) denotes the packetization rate). Thus, the actual inter-arrival time between game packets at prediction level \( i \) is given by:

\[
D = s_i + \frac{1}{m} \frac{L}{s_i} = \frac{1}{m} \frac{L}{s_i}
\]

The mean of this distribution is given by \( \frac{1}{m} \). Let \( s_i \) denote the average packet size generated by prediction level \( i \), \((i=1, \ldots, N)\). Also, let \( L \) denote the transport layer packet size (hence, \( \frac{L}{m} \) denotes the packetization rate). Thus, the actual inter-arrival time between game packets at prediction level \( i \) is given by:

\[
D = s_i + \frac{1}{m} \frac{L}{s_i} = \frac{1}{m} \frac{L}{s_i}
\]

The mean of this distribution is given by \( \frac{1}{m} \). Let \( s_i \) denote the average packet size generated by prediction level \( i \), \((i=1, \ldots, N)\). Also, let \( L \) denote the transport layer packet size (hence, \( \frac{L}{m} \) denotes the packetization rate). Thus, the actual inter-arrival time between game packets at prediction level \( i \) is given by:

\[
D = s_i + \frac{1}{m} \frac{L}{s_i} = \frac{1}{m} \frac{L}{s_i}
\]
\[ D_{\text{wired}} = D_{\text{wired-transmission}} + D_{\text{queueing}} \]  
\[ D_{\text{wireless}} = D_{\text{artink}} + D_{\text{interleaving}} + D_{\text{REC}} \]

where, \( D_{\text{wired-transmission}} \) and \( D_{\text{artink}} \) are the wired and wireless link transmission delays, \( D_{\text{queueing}} \) denotes the queuing delays at the intermediate nodes in the wired network, \( D_{\text{interleaving}} \) is a constant interleaving delay at the RLC-MAC layer for EGPRS data packets, and \( D_{\text{REC}} \) denotes the RLC-MAC delay due to retransmissions over the wireless link.

Once, \( n_{\text{mix}} \) is determined, we assume that the network delay follows an exponential distribution with mean \( \frac{1}{\lambda} \) (as we can only estimate one moment of this distribution). Estimating the second moment would result in costly analysis which might make the receiver based approach infeasible. Thus the probability \( P[n_{\text{mix}} \leq D] \) can be calculated by the CDF of this distribution given by:

\[ P[n_{\text{mix}} \leq D] = 1 - e^{-\frac{D}{\lambda}} \]  

(19)

Correspondingly, to determine \( P[n_{\text{mix}} \leq D - j\Delta] \), we need to compute the convolution of the network delay distribution with the approximated gamma-distribution for the \( j \)-fold convolution of the inter-arrival time of game packets, the PDF of which is given by:

\[
c(z) = \int_0^z \frac{m_1(n_1z)^{j-1}e^{-m_1}e^{-\frac{z}{n_1}}}{n_1^j(j-1)!} \, dz \\
= \frac{m_1^j}{n_1^j(j-1)!} \left[ \frac{z^{j-1}}{n_1 - m_1} \sum_{k=0}^{j-2} \frac{(j-2)\cdots(2)(j-k-1)!}{(j-1-k)!} \left( \frac{d}{n_1 - m_1} \right)^{j-1-k} \right] \\
+ \frac{m_1^j}{n_1^j(j-1)!} \left[ \frac{z^{j-1}}{n_1 - m_1} \sum_{k=0}^{j-2} \frac{(j-2)\cdots(2)(j-k-1)!}{(j-1-k)!} \left( \frac{d}{n_1 - m_1} \right)^{j-1-k} \right] \\
= \frac{m_1^j}{n_1^j(j-1)!} \sum_{k=0}^{j-2} \frac{(j-2)\cdots(2)(j-k-1)!}{(j-1-k)!} \left( \frac{d}{n_1 - m_1} \right)^{j-1-k} \]  

(20)

and hence the corresponding CDF gives us \( P[n_{\text{mix}} \leq D - j\Delta] \) as follows:

\[
C(D) = P[n_{\text{mix}} \leq D - j\Delta] \\
= \int_0^D c(z) \, dz = \frac{m_1^j}{n_1^j} \sum_{k=0}^{j-2} \frac{(j-2)\cdots(2)(j-k-1)!}{(j-1-k)!} \left( \frac{d}{n_1 - m_1} \right)^{j-1-k} 
\]

(21)

In the next two subsections we will derive approximate simplified estimates for \( D_{\text{RLC}} \) and \( D_{\text{queueing}} \) that will be used to solve the optimization problem.

4.7. RLC-MAC delay estimation

Chen and Goodman [14] give a detailed theoretical analysis of the estimated RLC-MAC [15] delay. But our goal is to have a simplified expression for \( D_{\text{RLC}} \) because the calculations are to be done at the MH. We first make the following assumptions for the RLC-MAC protocol:

(1) RLC block transmissions are congestion free, hence there is no wait to get hold of a time slot to transmit at the BS for any game packet.
(2) Processing time between receipt and response for any RLC block is fixed and is given by \( \tau \).
(3) There is no blocking at the MH, i.e., no dropping of packets.

(4) The input queue is never empty at the BS, i.e., there is always a packet to transmit. Consequently, the delay analysis does not have to consider vacations.
(5) ACKs and NACKs are never lost.
(6) A constant packet length is used for delay estimation. The game packets have fixed length of \( L \) bytes which is equal to the LLC frame length.
(7) The coding scheme used is CS4 (no FEC at RLC-MAC layer) which requires a block length of 52 Bytes. Based on these assumptions, we can write:

\[ D_{\text{RLC}} = D_{\text{tx}} + D_{\text{error-cor}} \]

(22)

where, \( D_{\text{tx}} \) is the data block's first-time transmission period based on the packet length and coding scheme used and \( D_{\text{error-cor}} \) is the handling time for backward error control including retransmission. Now, we have:

\[ D_{\text{tx}} = \frac{\text{LLC frame length}}{52} \times \tau = n \tau \]

(23)

The LLC frame length can be a maximum of 1520 Bytes as per GPRS standards and since game packets are small, we consider each game packet constitute an LLC frame. \( n \) denotes the number of RLC blocks per LLC frame. Also, we have,

\[ D_{\text{error-cor}} = D_{\text{data}} + D_{\text{back-agg}} \]

(24)

where, \( D_{\text{data}} \) is the retransmission time of the erroneous data blocks, and \( D_{\text{back-agg}} \) is the NACK (Negative Acknowledgement) message processing and transmission time. Now, if \( p_f \) denotes the RLC block error rate, the average number of transmissions, \( T_{\text{avg}} \), is given by:

\[ T_{\text{avg}} = (1 - p_f) + 2p_f(1 - p_f) + 3p_f^2(1 - p_f) + \cdots = \frac{1}{1 - p_f} \]

(25)

Similarly, the average number of retransmissions, \( R_{\text{avg}} \), is given by:

\[ R_{\text{avg}} = p_f(1 - p_f) + 2p_f^2(1 - p_f) + 3p_f^3(1 - p_f) + \cdots = \frac{p_f}{1 - p_f} \]

(26)

Hence, we get \( D_{\text{data}} = \frac{p_f}{1 - p_f} \), and \( D_{\text{back-agg}} = m_2 \tau \). And, finally for one LLC frame, we get the RLC-MAC delay as:

\[ D_{\text{RLC}} = \frac{2m_2 \tau}{52(1 - p_f)} \]

(28)

We next calculate \( p_f \) from \( p_w \) as follows:

\[ p_w = \text{probability that any one of the b blocks are in error} = 1 - \text{probability that all b blocks are received at MH} = 1 - \Pi_{i=1}^{b} 1 - p_f^{i-1} \] 

\[ = 1 - \Pi_{i=1}^{b} 1 - p_f^{i-1} = 1 - \frac{p_f}{1 - p_f} \]

where \( T_{\text{avg}} \) is given by Eq. (25). What we require is the inverse of this expression to have an estimate of \( p_f \) in terms of \( p_w \). So, we can simply calculate \( p_f \) iteratively by comparing against \( p_w \).

4.8. Wire-lined network queueing delay

In this subsection, we consider the delay conditions for the wired network segment. The game server sends game packets that
are routed through the proxy servers into the Internet. The game packets destined to mobile clients are received at the corresponding inter-working gateways (IWG) and then are transmitted through the SGSN and base stations (BS) to the corresponding mobile client. The scenario is depicted in Fig. 7.

1. M/M/1/K model. In the absence of any single model that can track the dynamics of traffic build-up in the entire network, the idea here is to study the effects of congestion selectively in a few strategic nodes in the network assuming a simplistic network model (Fig. 8). One such node is the gateway node between the wireless network and the wireline one, called the IWG. The simplifying assumption here is that all the queues in the network are lumped together into an M/M/1/K queue in the IWG. The IWG node is critical in the sense that there is a serious performance difference (e.g., link speed) between the wireless network with its wireline counterpart.

The M/M/1/K model is chosen because on the input side of the queue, the packet arrival process may be approximated as a Poisson process (however, this is not always true given the nature of WWW sources) with mean arrival rate \( \lambda \). The service rate, \( \mu \), is assumed as Poisson as well for simplicity and is determined from standard Internet delay distributions. A finite buffer size is assumed at the IWG to calculate the blocking probability, \( P_{\text{blocking}} \), of the queue as follows:

\[
P_{\text{blocking}} = \frac{(1 - \rho)\rho^K}{1 - \rho^{K+1}}
\]

where, \( \rho = \frac{\lambda}{\mu} \). Now, obviously we have \( p_{\text{blocking}} = p_\lambda \), and hence \( \lambda \) can be calculated from the above expression. Thus the queueing delay can be estimated as:

\[
D_{\text{queueing}} = \frac{1}{\mu} \left( \frac{(\lambda^K + 1)}{\lambda(1 - p_\lambda)} - \frac{1}{\rho} \right) + \frac{1}{\lambda} + \frac{1}{1 - \rho}
\]

Eqs. 29-30 follow from standard queueing theory results.

2. G/M/1/K model. The M/M/1/K model discussed above is simple to implement in the mobile devices requiring less computational and memory overheads. However, a better model will be to use the G/M/1/K system because the game packets themselves follow a general distribution and hence the total traffic coming into the SGSN should also follow a general distribution. The blocking probability and average delay computations become tedious for these variables as outlined in [16].

The dual queueing system for a GI/M/1/K queue is defined as the M/GI/1/K + 1 queue, in which the role of the arrival and service processes have been interchanged, such that the \( i^{th} \) state of the embedded Markov chain at the departure epochs in the dual queue corresponds to the \( (K-i)^{th} \) state of the embedded Markov chain of the original queue [16]. We assume \( \rho = \lambda_A E[S] < 1 \), where \( \rho \) is the traffic intensity, \( \lambda_A \) = the mean arrival rate and \( E[S] = \) the average service time. Thus the blocking probability, \( p_{\text{blocking}} \), is given by:

\[
p_{\text{blocking}} = \frac{\rho - 1}{(1 + c_A^2)}
\]

where \( Q = \text{mean queue length} \) given by:

\[
Q = \frac{1}{\rho} + \frac{1}{2\rho(p - 1)}
\]

\( c_A \) being the coefficient of variation. The corresponding mean delay for the G/M/1/K system is given by:

\[
D_{\text{queueing}} = \frac{Q}{E[S]} = \frac{1}{\rho} + \frac{1}{2\rho(p - 1)}
\]

The above equations can be easily derived from [16] by noting that \( \rho = \lambda_A \) where \( \rho^M \) is the equivalent traffic intensity for the dual M/GI/1/K system. Now, assuming certain values for \( E[S] \) and \( c_A \) and noting that \( p_{\text{blocking}} = p_\lambda \), we can compute \( \lambda_A \) and correspondingly \( D_{\text{queueing}} \).

4.9. Adaptive FEC and packetization algorithm

The optimal solution to the optimization problem in Eq. (14) is hard to achieve because it is an integer programming problem. The complexity is increased due to the packet size selection requirement. We assume that only a fixed number of packet sizes are available at the transport layer. This largely simplifies the problem and we can indeed find the optimal solution by simply iterating through the different possible values of \( L \), \( n \), \( k \) code pairs and \( L \), i.e., prediction levels and search for the highest throughput efficiency.

However, it is still expensive and unrealistic to obtain the optimal solution (combination of \( L \), \( n \), \( k \)) in real-time. One solution would be to move the computations offline and generate a mapping between the inputs (such as the possible values of \( p_i \) and \( p_{\lambda} \)) to the outputs (i.e., the \( L \), \( n \), \( k \) values). The mapping is stored as a lookup table in the MH. When a MH is executing the algorithm, it only needs to look up the table for the closest input that it can match with the real-time parameters, and use the corresponding pseudo-optimal values as the solution. The results in the next section however were generated by iterating through the possible values for \( L \), \( n \), \( k \) for the observed estimates of \( p_i \) and \( p_{\lambda} \).
5. Performance analysis

For the performance analysis of our TFGWP protocol, we used network simulator (NS) version 2 (NS-2) package. We implemented a standard dumbbell shaped network topology to simulate the Internet traffic which is generally used to simulate performances of TCP-friendly rate-control algorithms (Fig. 9).

The senders and receivers are kept on either side of the bottleneck link. All links (excepting the bottleneck link) are sufficiently provisioned to ensure that network congestion only occurs at the bottleneck link. Also the links are all drop-tail links. The variable wireless environment is simulated by applying a simple selective ARQ protocol to the wireless link. In the wireless channel, all IP-packets are first segmented into several LLC frames and then reassembled on the other side. Any reassembly failure of an IP-packet on the receiver side will be reported to TFGWP immediately so that it knows that an erroneous IP packet was dropped at the RLC layer. In the simulation, the size of IP packets were fixed at 500 Bytes, the LLC layer frame size is also fixed to 500 Bytes and the LLC layer block size adopts CS4 coding scheme with block size 52 Bytes. The background traffic consisted of several infinite-duration TCP-like connections. The game traffic was modeled using the heavy-tailed distribution as discussed in Section 4 with a mean rate of 50 Kb/s and peak rate of 80 Kb/s (for illustrative purposes).

We first demonstrate the efficacy of the end-to-end loss differentiation scheme proposed in this paper. We consider five prediction levels with mean sending rates of 10, 20, 30, 40 and 50 kbps. In the simulation, the bandwidth of the other links were fixed at 10 Mb/s with 10-ms delay, and that of the bottleneck link was set to 0.75 Mb/s, 0.65 Mb/s and 0.5 Mb/s, respectively, for the three plots given in Figs. 10–12. The wireless link was set to 200 Kb/s with 50-ms delay and a uniform error model was applied to the RLC frames in the wireless link to simulate the channel noise. The y-axis in the plots shows the percentage of congestive loss miscalculations, \( \text{per} \). To measure this quantity, we store the intervals \( a_i \) where congestion losses occur in each RUDP interval. If the predicted number of congestive losses be \( \text{pred} \), and actual number of congestive losses be \( \text{act} \), in interval \( a_i \), we have, \( \text{per} = \sum_{i=1}^{n} \left| \text{act}_i - \text{pred}_i \right| \).

Fig. 10 plots \( \text{per} \) against time. We find that initially the scheme in [11] performs better than ours. This is because our scheme needs a good estimate for \( T_{\text{min}} \) and hence performs poorly in the beginning when the observed \( T_{\text{min}} \) is relatively higher than the actual \( T_{\text{min}} \). However, with increase in time a better estimate is made for \( T_{\text{min}} \) and our scheme outperforms the one in [11]. This characteristic is evident in Figs. 11 and 12 as well. The percentage of miscalculations value for the scheme in [11] remains relatively constant in all the three plots with minor fluctuations over all the RUDP intervals for similar wireless link conditions. The main observable difference in the three plots is the fact that with decreasing bottleneck bandwidth the percentage of miscalculations tend to increase. This can be attributed to the corresponding increase in total number of congestive losses due to lesser bandwidth, resulting in a corresponding increase in the miscalculation rate.

Next we show the TCP-friendliness of TFGWP by performing simulations in the wired-line case with the same topology. The wireless link was replaced with a wired-line link with capacity 10 Mb/s and 10-ms delay. The simulations were run with three TCP connections competing with three TFGWP connections. Fig. 13 plots the total bytes received by TCP and TFGWP against time, and we can find that TFGWP can fairly share bandwidth with TCP. The TCP sending rate and mean sending rate of game packets determined by TFGWP (given by the mean sending rate at corre-

Fig. 9. The NS-2 simulation topology.

Fig. 10. Comparison of end-to-end loss differentiation, bottleneck = 0.75 Mb/s.
sponding prediction levels) are plotted against time in Fig. 14 for one TCP and TFGWP connection competing between themselves. We can observe that TFGWP can adjust its sending rate in a smoother manner than TCP.

To demonstrate the performance of TFGWP as a function of the wireless channel errors, we again fix the wireless link to 200 Kb/s with 50-ms delay as before. TFGWP is shown to perform increasingly better than UDP with increasing FER as shown in Figs. 16–18. All the previous results have been obtained using the M/M/1/K model for the wire-lined network. Fig. 15 plots the performance of TFGWP using M/M/1/K model and that using the G/M/1/K model with the latter shown to perform better (under higher processing cost) at FER = 0.3. The mean processing rate for the G/M/1/K model was kept the same as that for the M/M/1/K model, and we assumed 5% variance in arrival rate to generate the results.

For thin clients, the network delay distribution can be computed more simplistically, by maintaining the delays of the last 1000 packets in a queue. Each delay is first quantized, using a linear quantizer with a step size of 5 ms and upper limit of 5 s. The

![Fig. 11. Comparison of end-to-end loss differentiation, bottleneck = 0.65 Mb/s.](image1)

![Fig. 12. Comparison of end-to-end loss differentiation, bottleneck = 0.5 Mb/s.](image2)

![Fig. 13. Comparisons of data received for TCP and TFGWP connections.](image3)

![Fig. 14. Comparisons of TCP sending rate and the mean TFGWP sending rate at corresponding prediction level.](image4)

![Fig. 15. Comparisons based on data received between TFGWP using M/M/1/K and G/M/1/K at FER = 0.3.](image5)
frequency of each delay is maintained in a histogram. When a new packet arrives, the delay of the oldest packet is removed from the histogram, and the delay of the newest is added. The delay distribution is computed using a cumulative sum of the frequencies, and is done only at the end of each RUDP interval.

6. Conclusions

In this paper, we have proposed a QoS guarantee framework to support mobile gamers in multi-player networked gaming. The proposed rate-control scheme is TCP-friendly and thus should work well during network congestion by fairly sharing bandwidth with other connections. The wireless channel fluctuations are mitigated by our adaptive FEC and packetization techniques. We have also stressed on the importance of our receiver-based schemes to minimize the overhead feedback messages. The schemes proposed in this paper are therefore simple and easy to implement and should work well even for thin clients. It is possible to devise more accurate (though complicated) algorithms based on the packet loss rates, that might be difficult to implement in the thin clients that we have nowadays. Our future work includes extensive experiments on real game traffic traces to further elucidate the applicability of our algorithms. Also, some work is required on a formal quantification of the prediction levels based on the possible artificial intelligence based schemes for throttling gaming traffic. We believe that our QoS guarantee model should provide high throughput and make wireless gaming a success in the near future.

References


Peter Quax, Patrick Monseurs, Wim Lamotte, Danny De Vleeschauwer, Natalie Degrande, Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game, in: Proceedings of the Third Workshop on Network and System Support for Games, 2004 (NetGames 2004), pp. 152–156.


