Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2009, Article ID 530367, 14 pages
doi:10.1155/2009/530367

Research Article

Experiences from Implementing a Mobile Multiplayer Real-Time Game for Wireless Networks with High Latency

Alf Inge Wang, Martin Jarrett, and Eivind Sorteberg

Department of Computer and Information Science, Norwegian University of Science and Technology, 7491 Trondheim, Norway

Correspondence should be addressed to Alf Inge Wang, alfw@idi.ntnu.no

Received 4 May 2009; Revised 14 August 2009; Accepted 23 October 2009

Recommended by Graham Morgan

This paper describes results and experiences from designing, implementing, and testing a multiplayer real-time game over mobile networks with high latency. The paper reports on network latency and bandwidth measurements from playing the game live over GPRS, EDGE, UMTS, and WLAN using the TCP and the UDP protocols. These measurements describe the practical constraints of various wireless networks and protocols when used for mobile multiplayer game purposes. Further, the paper reports on experiences from implementing various approaches to minimize issues related to high latency. Specifically, the paper focuses on a discussion about how much of the game should run locally on the client versus on the server to minimize the load on the mobile device and obtain sufficient consistency in the game. The game was designed to reveal all kinds of implementation issues of mobile network multiplayer games. The goal of the game is for a player to push other players around and into traps where they loose their lives. The game relies heavily on collision detection between the players and game objects. The paper presents experiences from experimenting with various approaches that can be used to handle such collisions, and highlights the advantages and disadvantages of the various approaches.

1. Introduction

Online games like World of Warcraft [1] have become very popular with more than 10 million paying subscribers around the world (2008). Most current AAA titles released on consoles and PCs provide some kind of online multiplayer support. This trend has also been picked up by mobile game developers, but mainly in development of games for Sony Playstation Portable and Nintendo DS. There exist fewer online multiplayer games for mobile phones due to challenges related to latency, packet loss, and low bandwidth. Some examples of commercial and research mobile online games for mobile phones are Pirates of the Caribbean [2], Samurai Romanesque [3], Tibia Micro Edition [4], UbiSettlers [5], Real Tournament [6], and Knockabout [7]. One of the main challenges when developing multiplayer online games is how to handle network latency. The latency of data packets sent between clients and server causes *inconsistencies* of the players' views and can give certain players advantages and ruin the *fairness* in the game [8]. For wired networks, the inconsistency problem has largely been solved using the transaction mechanism *rollback* [9, 10], where the game rolls back to a consistent game state if a conflict of players' views is detected. This approach works quite well in low latency environments, but will not work in high latency environments.

The online multiplayer games for mobile phones on the market today are either turn-based games or slow-paced games to avoid the inherent problem with high latency and low bandwidth of wireless networks. Such games can live with *round-trip delays* of 1–3 seconds without ruining the gameplay. However, real-time multiplayer games require much lower response times, and for online games over wired networks round-trip delays above 150 ms can lead to unsmooth gaming experiences for first-person shooter games [11]. Similarly, the acceptable round-trip delay for military simulations is specified to be 100–300 ms [12]. Especially, in games that involve collision detection between players, it is critical that all game events are updated frequently among the players. The update frequency of game

events depends on the game genre. For instance, fighting and shooting games require higher frequency of game updates compared to strategy and role-playing games.

This paper describes experiences from the BrickBlock project where the focus was on investigating the challenges and opportunities of multiplayer real-time games played over the most commonly available wireless networks today: GPRS, EDGE, UMTS (3G), and WLAN (WiFi). WLAN has successfully been used for multiplayer online gaming for years and was included as a benchmark. The paper consists of two main parts. The *first part* describes network performance tests running instances of the BrickBlock game over various wireless networks. The test was conducted in live wireless networks, and measured the response time and the transfer speed. The paper reports on the practical implications and identified challenges found in the BrickBlock game based on the results of the network performance tests. The *second part* describes experiences from applying various approaches for handling network issues related to collision handling and providing a fluent game experience for the user. Due to the high latency of wireless networks, approaches that introduce unnecessary transmissions cannot be used. The goal of the approaches presented in this paper was to give acceptable game world consistency, a good load balance of CPU-usage between the server and the mobile clients, and minimum transmission overhead.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 describes the mobile multiplayer real-time game BrickBlock. Section 4 describes how the network performance tests were run. Section 5 presents the results of the performance tests, along with a discussion of the implications of the results. Section 6 describes various approaches minimizing the negative effects of network latency. Section 7 describes the results from testing the gameplay of BrickBlock over mobile networks. Finally, Section 8 concludes the paper.

2. Related Work

This section describes research related to networks and games. As there is little work on real-time mobile multiplayer games, papers on real-time online multiplayer games for wired network are also presented. The related work section is split into two parts. The first part looks at how network latency and bandwidth affects games, and measurements of running games over wireless networks. The second part looks at methods and mechanisms to minimize problems related to latency, low bandwidth, and packet loss.

2.1. Games and Network Performance. Busse et al. describe experiences from running a ported online game over GPRS and UMTS using the TCP protocol [13]. The game setup consisted of a two-player game where one client ran on a PocketPC PDA and one client ran on the game server. The game server sent out game states 20 times per second. The result from this test shows that for GPRS the average response time is about 1 second, where as for UMTS the average response time is about 285 ms. The authors did not implement any mechanisms to handle latency issues, and thus concluded that the game was unplayable over both GPRS and UMTS.

Beigbeder et al. investigated the effect of loss and latency on user performance in the online first-person shooter game Unreal Tournament (UT) 2003 running on PCs over wired network [14]. From studying the UT 2003 servers, they found that maximum loss rate was about 3% and maximum latency was about 140 ms. Although the introduction of loss and latency affected the players' performance to some degree, the difference was not statistically significant. The players were able to notice sluggishness in gameplay for latencies as low as 75 ms, and found the game less enjoyable at latencies above 100 ms. Similarly, Quax et al. found that latency above 60 ms or above was experienced as disturbing of players in Unreal Tournament [15].

Dick et al. analyzed how network latency and jitter affected the performance and perception in multiplayer online games (wired) [16]. This paper presents a survey where players state their subjective perceptions for how network latency and jitter affect the performance and gameplay for twelve different games representing four different game genres: first-person shooter, real-time strategy game, sport game, and car racing simulation. The result of this survey shows that the player's perception of the magnitude of latency that is accepted for an unimpaired game is about the same for all game genres: 80.7 ms in average. The perception of how much network latency that can be tolerated before it ruins the gameplay is up to 150 ms with an average of 118 ms. For the FIFA soccer game, 100 ms was the maximum latency tolerated.

Sheldon et al. describe results from a controlled experiment to investigate the effect of latency on user performance in the real-time strategy game Warcraft III [17]. The results of the experiment show that there is no significant effect on the performance of the players when the latency is increased (from 0 to 3500 ms). However, for exploring (determine geographical layout and location of other player's units) there is some correlation between explore time and latency. The results from analyses from playing the full game showed that the round-trip time was about 100 ms. Analysis of users playing the game showed that the users could compensate for latencies up to 500 ms. For latencies above 800 ms, the game appeared erratic which degraded the game experience.

Pantel and Wolf investigated how network latency affected controlling a car over the network in an RC-car simulator [18]. In the experiments, the latency was increased in steps of 50 ms up to 500 ms. Pantel and Wolf found that for beginner and average drivers, lap times get worse from 50 ms. Excellent drivers did not get noticeably longer lap times until the latency was 150 ms. Their conclusion was that latency of 50 ms can hardly be noticed, 100 ms is acceptable if no high demands with respect to realism are needed, 200 ms is clearly observable, and 500 ms is not acceptable.

Chen et al. describe results from studying players playing the massive multiplayer online role-playing game Shen Zhou Online [19]. They found that players that experienced 150 ms latency had an average game sessions lasting four hours compared to players that experienced 250 ms latency had an

average of one hour. Further, their results showed that the players' departure rate from the online game was sensitive to network quality that could be decomposed into latency, latency variation, and loss rate.

Henderson and Bhatti describe results from an experiment of introducing latency in the first-person shooter game Half-Life [20]. On online discussions, first-person shooter players state that they cannot live with latencies above 50 ms or 100 ms. Henderson and Bhatti's results show that players can live with latencies above 250 ms and that most players do not leave game servers until the latency is in average about 300 ms or above.

Nichols and Claypool investigated the effect of latency on online Madden NFL Football game [21]. They found that there is not much effect on user performance for latencies below 500 ms, and that latency below 500 ms is not noticeable to the user. With latencies above 750 ms, the player will feel that the game is "laggy."

Multiplayer online games running over wireless network must cope with latency around 250 ms or more [13]. From the various papers above, we can see that mobile network games like sports and real-time strategy games probably will not give a negative user experience, while others like first-person shooter and racing games will. The following section will describe approaches to minimize the effect of latency in network games.

2.2. Approaches to Minimize the Effect of Latency and Low Bandwidth in Games. Latency in network games introduces challenges mainly in four areas: network efficiency and utilization, visual consistency, game world consistency, and fairness.

The challenge related to *network efficiency and utilization* can be attacked in different ways. Fritsch et al. present how Content Addressable Network (CAN) can be used to improve broadcasting over the Internet tailored to support games by mapping an n-dimensional virtual area to the set of mobile nodes [22]. The main benefit from using CAN is that fewer packages are required compared to simple broadcasting. Zhu et al. present a shift coding approach that can be used in mobile peer-to-peer multiplayer games to efficiently exchange game state updates between neighbor nodes [23]. Experimental results show that this approach can reduce network traffic. Another commonly used technique to lower the bandwidth demand and latency variations in network games is the use of buffering of game state messages [24, 25]. Instead of broadcasting game state messages every time a user event occurs, the events are buffered and transmitted in predefined intervals.

Movement prediction is an approach to overcome warping of game objects on the screen that move around due to loss of data packets or low bandwidth. *Warping* means that moving objects seems to jump from one location to another [26]. The most common approach to deal with *visually inconsistencies* in network games is the *dead reckoning* technique [27, 28] used in the *Distributed Interactive Simulation* (DIS) protocol [12]. In this technique, all clients simulate the game objects of the other players using a defined set of algorithms that mimic the behavior of various players' objects. The precision of the game objects' predicted positions depends on type of algorithm used. Some algorithms give higher accuracy, but might demand more computational resources [29]. The DIS protocol includes a state protocol data unit (PDU), which contains information about a game object like identification, position, velocity, acceleration, orientation, and other state information. This makes it possible for a client to move game objects of other players smoothly using extrapolation based on movement predictions between transfers of PDUs. The price of using dead reckoning is that every client has to run an algorithm to extrapolate each entity in the game. This can be a potential problem for games running on mobile phones with limited CPU and memory. Also, if all the game objects behave unpredictable all the time, dead reckoning offers little gain.

There exist several approaches to solving the issues related to *game world consistency* in network games. One approach is the *bucket synchronization* mechanism where all game event messages are stored by the receiver in a bucket [25]. At a given interval (all clients must be synchronized), all the game event messages in the bucket are used to compute the local view of the global state. This approach is especially useful for peer-to-peer games. Another well-known approach is to add transaction support to deal with inconsistencies [30]. In the case of a detected inconsistency of the local game state, the game can *roll back* to a consistent state using a timewarp algorithm. A similar approach is to build a game upon a transactional distributed shared memory system [31]. Strict transactional approaches or bucket synchronization does not work well for multiplayer games played over mobile networks due to high latency and unreliable connections [7]. It is therefore necessary to tolerate some inconsistency between the players' views and states to enable a real-time experience. Chandler and Finney describe an approach named *Rendezvous* to cope with consistency game state consistency in high latency environment [32]. Rendezvous is an optimistic consistency mechanism that tolerates managed inconsistency between views of shared state within a game, enabling each node to visualize actions as they happen. The consistency is maintained through a shared state convergence mechanism that produces a new global state by averaging values of local sates. An evaluation of testing the implementation of Rendezvous mechanism in a mobile multiplayer soccer game shows that it is possible to obtain an acceptable level of consistency without using a rollback mechanism.

Fairness in network games is measurements to avoid some player to have an advantage over others due to latency [33]. Lin et al. propose a synchronized messaging service named *Sync-MS* to balance the tradeoff between response time and fairness [34]. Sync-MS uses two mechanisms Sync-out and Sync-in. Sync-out is used to queue up a message at the player's client and deliver it to the game application only after the same update message has arrived at all clients. Sync-in is used to enforce a sufficient waiting period on each game state message dynamically to guarantee fair processing of all messages. Zander et al. have developed an application that can be used with existing network games to remove the latency differences, and thus giving players equal network

conditions [35]. GauthierDickey et al. have made a low latency event ordering protocol, named NEO, which divides time into rounds and uses the round duration to bound the maximum latency [36].

All the approaches apart from the Rendezvous described above are targeted for wired networks with low latency and is therefore not well suited for wireless networks. For network games played over wireless networks the challenge is to provide a responsive gameplay with sufficient consistency.

3. BrickBlock—A Mobile Multiplayer Network Game

The BrickBlock game concept was developed to test real-time performance of wireless networks. This section describes the BrickBlock game.

3.1. The Game Concept. In BrickBlock, each player controls his brick around a two-dimensional playfield. The goal of the game is to push other players into certain areas defined as traps. When a player is pushed into a trap he dies, looses points, and after some time respawns (reappears). The winner of the game is the player that has died least number of times within a predefined time. This concept opens for tactical play, as the players most likely will collaborate in order to push and block one targeted player. Further, such alliances must be temporary for one player to become the winner of the game. Ambitious players will most likely jump from one alliance to another several times during a game session to make sure he is always in the best position for the victory. In other words, BrickBlock is a game characterized by its anarchy, chaos, and treachery—attributes that make it an entertaining, unpredictable, and social game.

Figure 1 shows an illustration of the game. When the game starts, the strength, size, and speed of the players' bricks are equal. This will change when a player consumes one of the three kinds of power-ups provided that the *Speed power-up* gives the player increased speed, the *Size power-up* increases the size of the player's brick, and the *Strength power-up* increases the player's pushing strength.

3.2. The Game Architecture. The architecture of the BrickBlock game is a combination of three architectural patterns: the client-server, the layered, and the model-view controller pattern as shown in Figure 2.

The bottom layer consists of the *Communication module* that manages all communication between the server and the client. The same communication and message-parsing interface is used on both sides to provide a uniform communication between the server and the clients, and thus making it easier to support various message formats such as plain text and XML.

The *Test module* is not necessary for running the game itself, but is used to run network performance tests between the server and the client. This module also implements the communication interface and can therefore be used as a communication module by the model layer.

The *Model layer* contains the information needed to represent the current state of the game. It also keeps track of the messages needed to be sent, or being received. The model part on the server side stores and manages information about the game that also is stored and managed by the clients. The server contains the whole view of the game, while the clients have a more local temporary representation of the game.

The *View layer* provides the graphical user interface for the server and the clients. The view module on the server is very simple showing the players being connected and some settings. The view module on the clients displays the game running including screens for a game lobby and the game itself. The server was implemented in Java SE while the client was implemented in Java ME.

4. Testing Real-Time Network Gaming in Live Wireless Networks

One main goal in the investigation of the performance of real-time games in wireless networks was to find the *actual performance* of such games in a *real network environment*. This ruled out the choice of using computer simulations to calculate the network performance. We chose to measure the performance using a real and network-demanding multiplayer real-time game consisting of a server running on a standard PC and mobile clients running on two Sony Ericsson K750i, Sony Ericsson K800i, and Nokia N73 mobile phones. The performance tests were run several times at various times of the day and on various days to capture normal data traffic variations.

The two most important measurements for network performance for real-time games are *response time* and *transfer speed*. The former determines the expected latency of updating game changes across to the clients. The latter determines how much data can be sent between the client and server and limits the number of players that can play the game simultaneously.

4.1. Response Time. The *response time test* measures the *Round-Trip Time* (RTT)—the time a small packet uses from the server to a client and back. The test module generates packets of only 4 bytes containing an id and a separator character. The number of packets generated depends on the number of intervals and the number of packets sent in each interval. These packets are sent with a delay of a fixed mount of milliseconds, which increases with each interval. The test calculates the time values, extracting the highest and lowest times, and calculates the average for the remaining of several runs.

Figure 3 shows how the RTT test was performed by sending a packet in intervals. With more packets per interval, the time between transmissions is not increased before all packets in that interval have been sent. The data packet is represented as a rectangle with a length l. The time the packet uses from the server to the client and back is denoted as t_i, where i represents the number of the packet. Finally, the transmission intervals are denoted as multiples of ΔI.

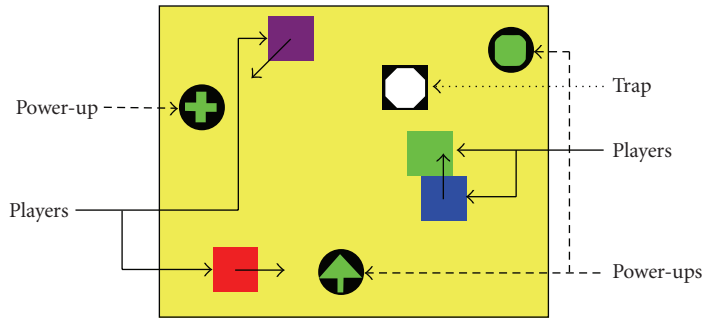

Figure 1: Illustration of the BrickBlock game.

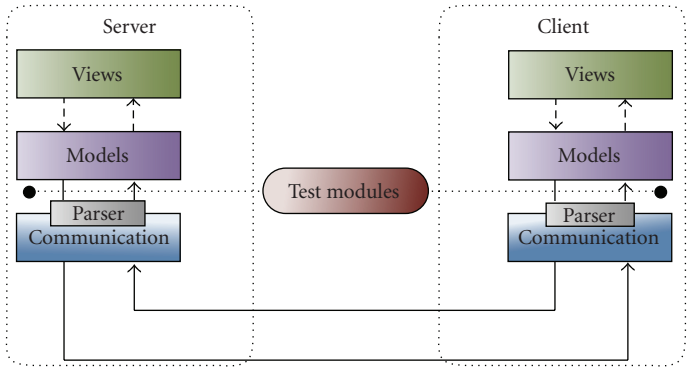

Figure 2: Architectural overview of BrickBlock.

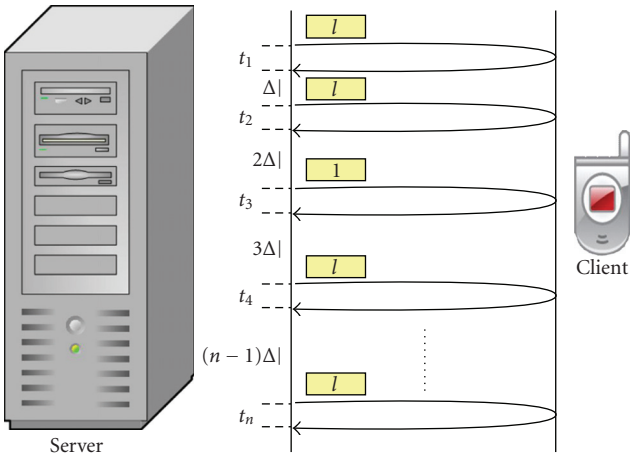

Figure 3: Measurement of response time.

The purpose of the response time test is to find the transmission interval that gives the shortest RTT resulting in low latency enabling smooth gameplay. However, every time the transmission interval is increased, the total time to send a packet is also increased with the same amount of time. The transmission interval with the lowest RTT may therefore not necessarily be the optimal transmission interval.

4.2. Transfer Speed. The *transfer speed test* measures the transfer time (transfer speed) of transmissions of different packet sizes from the server to a client and back. The test uses the same interval setup as the response time test (see Section 4.1). The test transmits a data packet of an initial size and the data packet size is increased with specified amount bytes for each transmission. The packet consists of an id, a separator character, an end-of-message character, and a number of characters to fill up the rest of the packet to the desired size. The delay between each packet was defined according to the optimal interval found in the response time test (see Section 4.1). The packet size is increased after the completion of a transmission from the server to the client and back.

Figure 4 shows how the transfer speed test was performed. The packet to be sent is illustrated by the rectangle and the size of the packet is the initial length l_0, and the increment in size Δl. The time the packet uses from the server and back is denoted as t_i where i represents the number of the packet, and the interval between transmissions is denoted ΔI.

The purpose of the transfer speed was to compare the *actual* transfer speed of different mobile network technologies, and the two transfer protocols TCP and UDP. The transmission time was measured from when a packet was sent from server to client, and back. This test also found the highest and lowest times, and calculated the average for the rest.

5. Results of the Performance Tests

This section presents the results of the network performance tests described in the previous section.

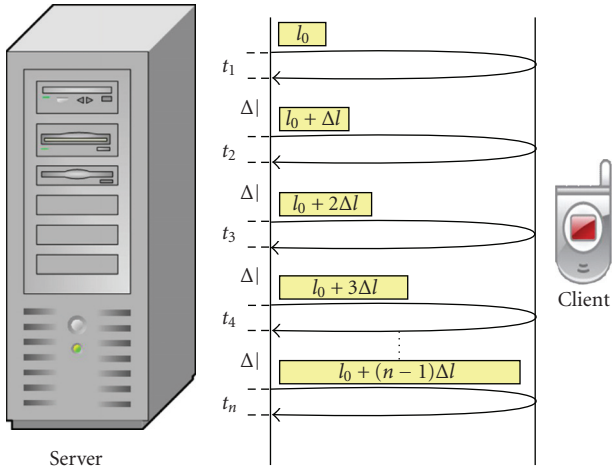

Figure 4: Measurement of transfer speed.

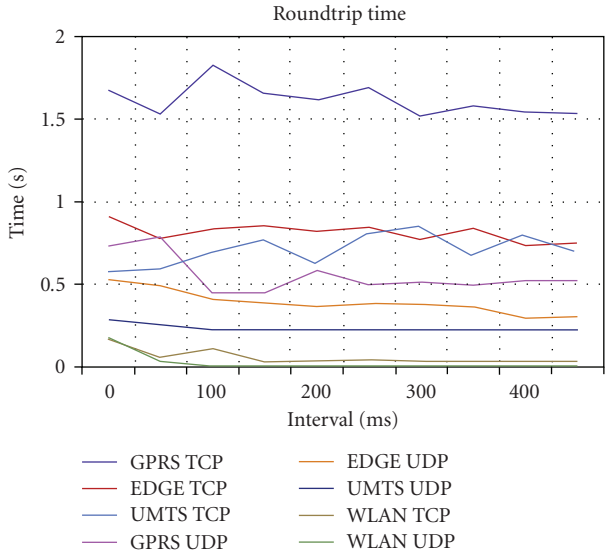

Figure 5: Measured response time.

5.1. Results of the Response Speed Test. For real-time multiplayer games like BrickBlock the response time has high significance, as the game requires small data packets to be sent frequently. Thus, the response time values for the wireless network technology measured in this test indicated the suitability of the network for running real-time multiplayer games.

Figure 5 shows the measured response time for the wireless network technologies GPRS, EDGE, UMTS, and WLAN, and transport protocols TCP and UDP. Figure 6 shows the measured response times including the pause interval.

The charts in Figures 5 and 6 show that UDP performs much better than TCP on all networks. Further, the transmission interval with the lowest response time is between 150 and 200 milliseconds, depending on the mobile network technology used. WLAN has the lowest response time, followed by UMTS, EDGE, and GPRS as expected.

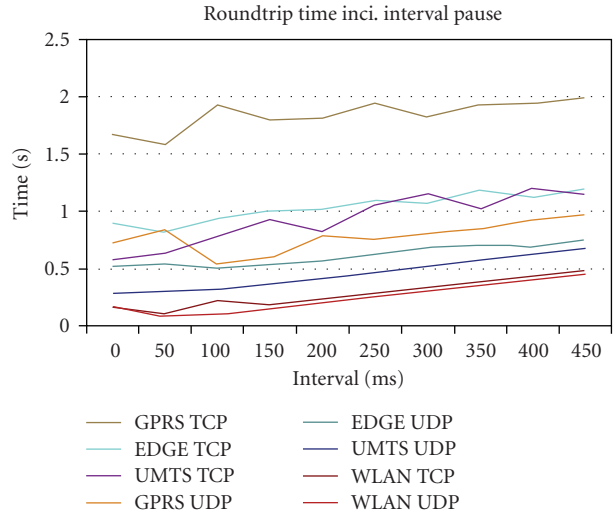

Figure 6: Measured response time including transmission interval.

From both Figures 5 and 6, WLAN clearly outperforms the other network technologies. The WLAN response time always stays below 200 ms seconds for all transmission intervals, and with both transport protocols. The lowest response times measured for the other network technologies using UDP are *217 ms for UMTS, 291 ms for EDGE*, and *445 ms for GPRS*. The UMTS response time is always less than that of EDGE using UDP. With TCP, the distance between the two is significantly less and the EDGE response time is even lower than the UMTS response time for some intervals. GPRS with UDP has close to identical performance as EDGE between 100 ms and 150 ms. However, for the other send intervals, EDGE is closer to UMTS. With TCP, GPRS never has a response time below 1500 ms, which can even make turn-based multiplayer games unresponsive.

In addition to find the optimal transmission intervals for lowest response time, we also ran several tests to look at the variation for transmission intervals between 100 ms to 250 ms. The results showed that *transmission intervals around 250 ms had significant less variation than for shorter intervals*. For UDP, the standard deviation was 80.92 ms for GPRS, 27.59 ms for EDGE, 5.12 ms for UMTS, and 1.19 ms for WLAN. When the TCP protocol was used, the standard deviation increased at least threefold.

Figure 6 shows the response times including the transmission interval; that is, the total response time from the previous packet is sent from the server to the server that receives the return message from the client. This indicates the range of transmission intervals that will provide the lowest total response time. The longer the pause interval is, the longer the total response time will be. Figure 6 shows that *pause interval values between 50 and 250 ms will provide the best total response time*. Table 1 shows the minimum, maximum, and average response times including transmission intervals. The numbers in Table 1 show that the optimal game update intervals over the network are just above 500 ms for GPRS and EDGE, 288 ms for UMTS, and 88 ms for WLAN. If the TCP protocol is used, the optimal

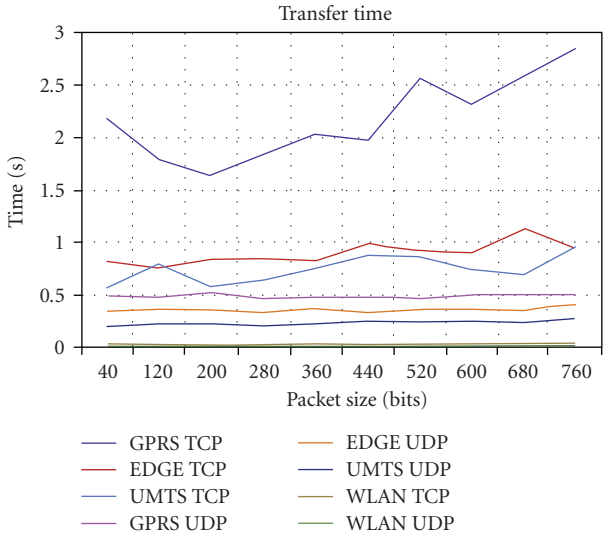

Figure 7: Measured transfer time.

update intervals are 1580 ms for GPRS, 820 ms for EDGE, 580 ms for UMTS, and 11 ms for WLAN.

5.2. Results of the Transfer Speed Test. The transfer speed measures how much data the network is able to transport per second. For a network offering a high transfer speed, larger data packets can be transmitted without loss of performance. This may have significant impact of how much data can be sent between the clients and the server without any lag.

Figure 7 shows the results of the transfer speed tests (short transfer time is best). The test results show the measured transfer time for each packet size. The figure shows that the time a specified network technology applying the UDP or TCP protocol uses to send a packet of a specified size from the server to the client and back.

When using UDP as the transport protocol, the size of the packet does not matter up to a certain size. In Figure 7, the transfer time using UDP is almost constant for all networks. WLAN has the lowest transfer time by far; whereas the UMTS transfer time is around 200–250 ms longer. The EDGE transfer time is additional 100 ms longer, and the transfer time using GPRS is yet another 150 ms. The limit for a satisfactory transfer time depends on the transmission interval chosen and the amount of data sent to the client. With UDP, the four different mobile network technologies all have transfer times around 500 ms or less. This transfer time is measured from the server to the client and back, so the time from the server to the client can be expected to be half the measured transfer time. Packet sizes of the BrickBlock will not affect the response time when UDP is used.

However when using TCP, the packet size affects the transfer time more, except for WLAN. The transfer times with TCP vary more over the different packet sizes. With some packet sizes, EDGE is better than UMTS, but in average UMTS is better. GPRS with TCP has the most fluctuating transfer time. The average of all the packet sizes is more than one second longer on GPRS with TCP than the second worst network technology's average transfer time with TCP (EDGE).

The upper part of Table 2 shows the minimum, maximum, and average transfer times for when varying the packets from 40 to 760 bits. The two rows at the bottom of Table 2 show for what packet sizes the lowest transfer time (min size) and highest transfer time (max size) were found. Table 2 shows that for GPRS with UDP, the packet size is bigger for the lowest transfer time (520 bits) than for the highest transfer time (200 bits). For the other networks and for TCP, the lowest transfer time is for packet sizes smaller than for the higher transfer time. It is not obvious why GPRS using UDP breaks this pattern, but Figure 7 shows that the graph for GPRS using UDP fluctuates almost like a very flat sinus curve with minor variations.

5.3. Large Data Packets. The results in the previous section show that the transfer time does not vary much with typical sizes of data packets used in games like BrickBlock (see Section 5.2). In an additional test, we wanted to find the upper limit for package sizes (beyond 760 bits). This test was only run over UMTS using the UDP protocol, as UMTS has the most suitable characteristics for mobile network games apart from WLAN. Our test showed that the transfer time is relatively stable between 200 ms and 300 ms independent of the packet size up to 11 616 bits. For package larger than 11 616 bits, all packages are lost (practical upper limit for package size using UDP).

5.4. Practical Impact of Network Performance on Gameplay. As expected from the specifications of the four networks considered in this paper, WLAN has the shortest response time and the fastest transfer speed, followed by UMTS, EDGE, and GPRS. Since WLAN is not widely available on mobile phones in Europe, whereas UMTS is, UMTS is currently the most promising network technology suited for multiplayer real-time games that can run on most new phones. The transmission interval has significant effect on the response time. The response time decreases with increased transmission intervals, but this will of course also increase the total response time. In practice, multiplayer games that run over GPRS and EDGE using the UDP protocol can expect game updates every 500–600 ms. This means that only turn-based or slow-paced games like strategy games, role-playing games, and board games can be played over these two networks. If UMTS and UDP are used, game events can be updated about every 300 ms (3 updates per second). Thus real-time games without very fast direct interaction between players can be played over this network. This result also means that mobile multiplayer real-time games should include apply mechanisms to compensate for the network latency to provide a smoother and more consistent gameplay. Multiplayer games over WLAN and UDP can expect game updates every 9 ms (11 updates per seconds), which is sufficient for most real-time games. For wireless networks where the user will be charged for every byte transferred, the costs of playing the game will increase with more frequent game updates and the number of players.

Table 1: Response time including transmission intervals for various wireless networks.

Protocol	UDP				TCP			
Network	GPRS	EDGE	UMTS	WLAN	GPRS	EDGE	UMTS	WLAN
Min	0.545 s	0.510 s	0.288 s	0.088 s	1.578 s	0.823 s	0.576 s	0.110 s
Max	0.971 s	0.751 s	0.674 s	0.457 s	1.980 s	1.198 s	1.195 s	0.485 s
Average	0.779 s	0.614 s	0.458 s	0.215 s	1.840 s	1.036 s	0.933 s	0.282 s

Table 2: Transfer time for various wireless networks (40–760 bits).

Protocol	UDP				TCP			
Network	GPRS	EDGE	UMTS	WLAN	GPRS	EDGE	UMTS	WLAN
Min	0.467 s	0.349 s	0.196 s	0.006 s	1.652 s	0.763 s	0.573 s	0.029 s
Max	0.531 s	0.410 s	0.277 s	0.007 s	2.862 s	1.127 s	0.979 s	0.035 s
Average	0.492 s	0.366 s	0.241 s	0.007 s	2.186 s	0.897 s	0.753 s	0.032 s
Min size	520 bits	440 bits	40 bits	40 bits	200 bits	120 bits	40 bits	200 bits
Max size	200 bits	760 bits	760 bits	760 bits	760 bits	680 bits	760 bits	760 bits

As an example, a two-player game of BrickBlock over UMTS costs in Norway about 15 cents/minute. A four-player game costs 17 cents/minute. We predict that an eight-player game will cost about the double (30 cents/minute) of a two-player game. New pricing policies for mobile online games must be in place to make such games popular.

The transport protocols TCP and UDP perform very differently on the various mobile networks. GPRS using UDP outperforms EDGE and UMTS using TCP. The transfer time tests indicate that the packet size has no influence on the transfer time when using UDP as the transport protocol. Because of this, the package size does not affect the transfer time as long as the size is less than 11 616 bits and UDP is used. This means that for all the networks but WLAN, UDP must be used for real-time games. The disadvantage is that UDP does not handle packet loss. The big difference in performance between UDP and TCP is related to the way the two protocols treat retransmissions. TCP was designed for wired, reliable networks where packet loss is due to congestion in the network. In wireless networks, most packet loss is due to link failure [37]. Even if different TCP implementations perform different over wireless networks, they all performs poorly compared to UDP. Another problem that must be solved when using the TCP protocol over wireless network is to handle variations in delay and rate. This problem can partly be solved by introducing acknowledge buffers that absorb the channel variations and using TCP-aware scheduling and buffer sharing algorithms [38]. Further, TCP traffic over wireless networks can be improved by choosing the optimal package size and managing retransmissions in alternative ways [39]. However, mobile real-time games require frequent updates with minimal latency without retransmissions, thus TCP is not a good solution.

The transfer time test showed that packages up to 11 Kbits could be used without affecting the transfer time. This should be sufficient for most mobile multiplayer games. The amount of data should be kept to a minimum in any case to avoid network congestion issues and to minimize cost.

5.5. Challenges for Mobile Real-Time Multiplayer Games. The goal of the BrickBlock project was to design a game that revealed issues related to network lag and low network bandwidth shown clearly in the gameplay.

(i) It is critical that the positions of the players (the bricks) are correctly reproduced on all the players' screens, as how bricks are positioned on the play area is critical to the gameplay.

(ii) It is critical to detect when a brick (the player's object) hits the walls limiting the play area and collision with other objects such as power-ups and traps.

(iii) It is critical to detect when two or more bricks collide to correctly move the bricks according to the involved physical forces.

From preliminary tests running the first version of the game without any latency compensation mechanisms over a GPRS wireless network, we noticed a number of problems.

(i) The position of the same brick was different on different players' screens. As such, the players did not have one coherent representation of game world.

(ii) The collision detection with walls did not always work, as it was performed on the server to minimize the load of the mobile device [40]. Unfortunately, in some cases the server did not discover when a brick hit the wall in time, and the brick would float outside the play area (unstable state of the game).

(iii) The most noticeable problem was inaccurate detection of collisions between players (bricks). In some cases, players could simply run over other players without any collision detection at all. In other cases, bricks were pushed around when it looked like they did not collide on one of players' screens or the bricks ended up on top of each other (illegal state).

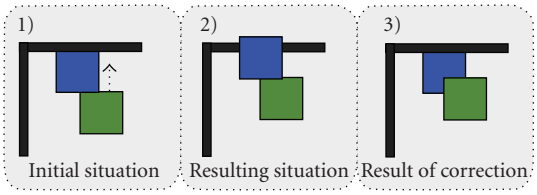

Figure 8: Externally caused wall collision.

6. Handling Network Latency in BrickBlock

This section presents our experiences from overcoming challenges related to latency in handling collision [41] in the BrickBlock game. The focus of this section is the determination of when the server should resolve consistency versus when the clients should resolve consistency while minimizing server involvement. The approaches we use to handle consistency are not new and have previous been demonstrated in both wired LAN games and games over the Internet [6, 7, 10, 11, 14]. The goal of our work was to focus on the engineering experience of wireless game development and find the appropriate mix of server/client responsibility suitable for our wireless BrickBlock game and why this is the case. Our paper does not contain any formal description or validation of consistency, as in our case consistency is measured in how the user experiences the game through the gameplay elements and appearance rather than through an statistical analysis. Traditional conservative approaches using transaction management and roll-back were not considered in this game, as they would introduce additional transmissions, which would increase the already high latency.

6.1. Collision with Walls. Wall collisions occur when a player's brick moves to a position where it is partly or completely located outside the play area. This happens either when the player tries to move to this position, or when another player pushes him to this position. The latter can be solved in two different ways. *One approach* is not to allow a player to move if it pushes another player to be placed in an illegal position. The problem with this solution is that the approximated position of the pushed player is not necessarily completely correct due to network latency and the push should be allowed. *Another approach* is allowing such a move, and only checking the local player's position against the wall. In this case, a player may actually be pushed outside the wall but later corrected by the player being pushed or the server (see Figure 8).

Chosen Approach. Both approaches have drawbacks by introducing temporary displacements of game objects. We chose to implement the first approach letting wall collision be detected by the pushing player avoiding introducing the need for extra correction of game object's position.

6.2. Collision with Power-Up Objects. Another type of collision detection is collisions with power-up objects. When such an event occurs, the power-up needs to be removed from the play area, and the player's attributes need to be updated for all participants. The decision is whether the power-up collision should be performed locally on the client or on the server. The *advantage of client detection* is that the player can get an immediate response when the power-up has been picked up (using sound, graphics, or vibration) and immediate change of the brick's attributes. The *disadvantage* is that multiple players can pick up the same power-up due to latency of broadcasting the pick up to other players. The *advantage of server detection* is that only one player can pick up the power-up (a consistent game model), but the player will not get an immediate response when he has picked up the power-up.

Chosen Approach. Our solution is a hybrid approach doing first a local collision detection, then using the server to ensure consistency. When a power-up collision is detected, player X will get an immediate response (vibration or sound). A message is then sent to the server to check if another player has picket up the power-up. If not, a message is broadcasted to all players that player X has changed attributes and that the power-up must be removed. If two or more players have picked up the power-up at the same time, the first player that first sent a power-up notification to the server will get the power-up.

6.3. Collision with Traps. A trap collision occurs when a player collides with the trap object on the play area. This will usually happen when the player is pushed by other player(s) into the trap, or self-caused accidents. Both these cases are similar to the power-up collisions.

Chosen Approach. Unlike with power-ups, it is not a problem if more than one player collides with the trap at the same time, as several players can die at the same time. However, when a player dies, he must respawn (reappear) in an unoccupied part of the play area. This involves checking all players' positions and finding an available spot. If several players die at the same time, the game must make sure that none of the players respawn at the same place. The server will respawn the players in turn by broadcasting an available position after updating scores and a short timeout.

6.4. Collision with Other Players. Like collisions with power-up objects and traps, collisions with other players are quite simple to detect using the client collision detection. The main difference is that players move around all the time. The simplest case is when one player is standing still while another is pushing. This is just like collision detection of game objects by updating the position of the pushed player by new positions sent by the pushing player. But when both players move at the same time, the situation is more complex. The following three situations might occur as shown in Figure 9.

The left image of each case shows a possible representation of the player positions whereas the right image shows the actual positions of the players. The three situations illustrated

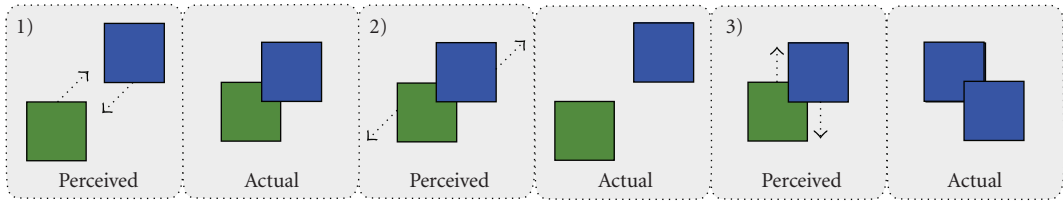

Figure 9: Player collisions with simultaneous movement.

in the figure can arise when (1) an existing collision is not detected because both players have moved into the same area, but the position of at least one player has not yet been received, (2) a nonexisting collision is detected because both players who were in the same area have moved away, but the position of at least one player has not yet been received, and (3) an existing collision is detected, but it is not completely correct since the position of at least one player has not yet been received.

The *first case* may result in two players occupying the same board position for a short period of time, until the new position has been received and the collision is detected. The *second case* is the exact opposite of the first. The consequence can be that a player is pushed even though he has actually managed to get away from the pushing player. If this happens too close to the trap, the player may unintentionally die. However, like in the first case, the correction will come quickly enough to cause any critical damage. For the *third case*, there is no consequence for how the players experience the game. Whether this collision occurs at the edge of or at the centre of the brick, the result is the same that the strongest brick moves the other in the strongest player's movement direction.

Chosen Approach. In BrickBlock, the server holds the most accurate picture of the total game state, while each client knows best its own state. We have chosen to manage player collisions locally to avoid unresponsive gameplay in collisions introduced by server-based collision detection. This means that the collision is not always accurate, but it is very responsive for the user.

6.5. Handling Player Movement in Collisions. Correct action must be taken when a collision between two players is detected. In BrickBlock, the result of such collisions is a change of speed and movement direction for at least one of the players. *First*, the strength ratio between the players involved is calculated. If one of the players is stronger than the other, the strongest player will be able to push the other in the strongest player's movement direction. How much the player can be pushed depends on the strength ratio between the players, as well as the movement speed of the strongest player.

Collision handling like collision detection can be handled on the server with more processing power or on the client with shorter response time. The server has a more accurate game model than the involved clients and can calculate relatively accurate positions for the pushing and the pushed players. However, both players will be able to move forward a short time until the server receives the collision notification and transmits the new positions. Thus, the players will experience that their bricks will be moved backwards seemingly without reason. This solution is illustrated in Figure 10. The figure shows a step-by-step procedure of how calculations will be performed and messages transmitted when the server is responsible for handling player collisions. Step 5 is performed in parallel on the local and remote client. According to the figure, the redrawing of positions is carried out in step 5 on the local client. Steps 2 and 4 are transmission between server and client that with a slow network can issue visible and noticeable latency for the players.

Chosen Approach. To ease this problem, we let the pushing player have responsibility for calculating the results of the collision. Figure 11 shows a step-by-step illustration of this approach. Here, the redrawing of the players occurs already in step 3. Furthermore, no message transmission is necessary before the play area is updated. This approach will give a far more responsive game experience from the pushing player's point of view. The movement of the involved players is calculated as a force vector by the pushing player, which is sent to server and then broadcasted to all clients.

Due to the latency in the network, a player may experience to be pushed without contact between the players displayed on his phone and there may be situations where a push should occur, but does not (similar to the situations illustrated in Figure 9). However, these minor inconsistencies are acceptable and do not ruin the gameplay.

6.6. Movement Prediction. The movement prediction is performed in BrickBlock by calculating a vector based on the two latest known positions of the object. These positions are stored on the client and are replaced whenever a new position is received. When a new position is received from the server, the new position and the previous position are used to calculate the movement vector by subtracting the old coordinates from the new and multiplying the result with the player object's speed property. This movement vector is then used to move the player object while waiting for the next position update from the server. This procedure for movement prediction is further described in Listing 1.

However, this approach has its drawbacks if a player suddenly changes direction as illustrated in Figure 12. The figure illustrates a situation where the player has performed a

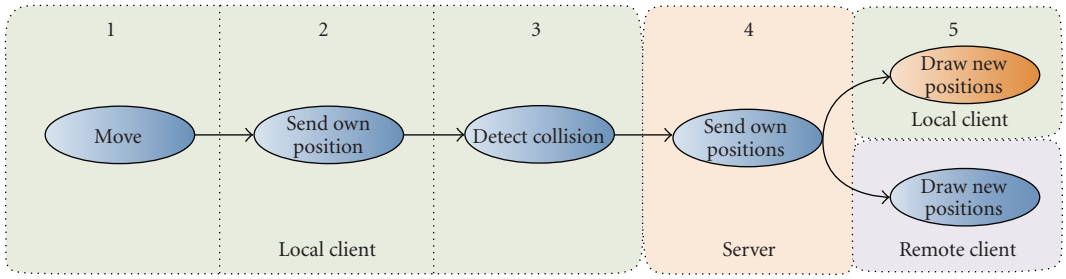

Figure 10: Server-side collision handling.

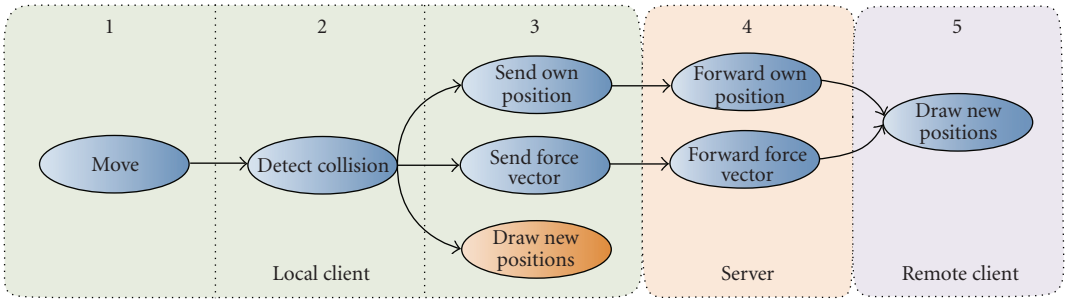

Figure 11: Client-side collision handling.

90° turn. The maximum deviance will be in a 180° turn that cause a warp error distance of twice the movement distance.

In addition to warping of the player on the screen, the usage of movement prediction can also cause problems for calculating collisions with other players, game objects, or with walls. Another negative effect of using movement prediction is that it demands some processing power on the mobile client. If many players are playing the game, the movement prediction might strain the mobile phone's CPU and slow down the game. This is why we have chosen to implement a simple movement prediction algorithm that requires little CPU.

6.7. Message Broadcast. During gameplay, all clients will continuously send updates to the server about their local state. To keep the clients up to date, the server needs to forward these updates to all clients as soon as possible. The simplest approach is to make the server forward the client messages as soon they are received. However, this approach may cause a congestion of messages and an increasing queue of messages to be sent. Our chosen approach was to bundle several client messages into one message with updates that will reduce the number of messages sent considerably. The server broadcasts the bundled message regularly to all clients at the same time ensuring a more consistent view of the game by all players. The main disadvantage with this approach is that it is crucial that the bundled message is not lost, as it contains much more information. This problem can be solved introducing acknowledgment that will double the latency in the system. A simpler and better approach is for the server to send incremental message IDs, so the clients will know if they missed a message and can demand a retransmission.

7. Experiences and Discussion

The BrickBlock game was tested through gameplay sessions over various mobile networks. The playability tests showed that the gameplay was very smooth on WLAN, fully playable with minor latency issues on UMTS using UDP, and even playable over GPRS and EDGE networks. Our approaches to minimize the effect of latency worked well most of the time.

In a mobile client-server system, the server's computational power should be utilized to off-load the less powerful mobile clients. In BrickBlock, we found that we could not let the server be in change of the collision management, as the game would suffer too much from unresponsiveness. Our approach was to let the mobile client of the player executing an action be responsible for detecting the collision with walls, power-ups, traps, and other players. This approach gives a responsive user experience, but can cause some acceptable temporary inconsistencies of the various players' views. The server is mainly used to forward messages, to ensure consistency when picking up power-ups, and to ensure that players do not respawn at the same spot.

Our use of movement prediction worked well most of the time and gave a much more enjoyable and smoother game experience. The main negative effect was occasional warping of players. One solution to eliminate the warping effect is to introduce animation when changing the direction of player controlled game objects. This means that the player cannot do sudden direction changes, but have to wait until the animation is finished. This approach is used in the game "Pirates of the Caribbean" when turning a ship [2]. Similarly, an animation could be used in collisions between two game objects to give time to do a consistency check with the server, and position the game objects correct. Another

```
(1) Last received position update from player A: (x0, y0)
(2) Receive position update for player A: (x1, y1)
(3) Movement X = x1 − x0
(4) Movement Y = y1 − y0
(5) If Movement X! = 0
(6)     Movement X = Movement X/Math.abs (Movement X) //Operate with 0's or 1's
(7) If Movement Y! = 0
(8)     Movement Y = Movement Y/Math.abs (Movement Y)
(9) Each time the play area is redrawn
(10)    A's position = A's position + (A's speed) [Movement X, Movement Y]
```

Listing 1: Procedure for predicting movement.

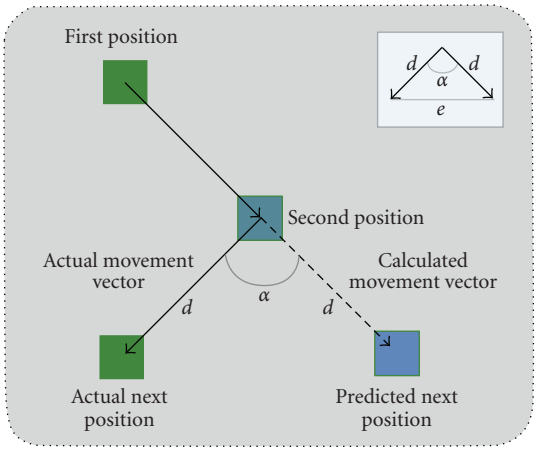

Figure 12: Warping in movement prediction.

simple solution to avoid warping is to reduce the speed of the players' game objects.

Another problem related to our movement prediction occurred occasionally when the game object of a player continued to move on other players' clients after the player had stopped due to a lost data packet. One solution to this problem is to introduce a transaction mechanism for messages that requires an acknowledgement before a message is accepted [30]. Similarly, effective retransmission algorithms can be used [42]. Both these approaches will double the response time of the system and are thus not desirable. For BrickBlock, this problem is not critical as packet losses are rare, and players have to move around all the time.

8. Conclusion

In this paper we have presented results from performance tests running a real-time multiplayer game over various wireless networks. The results show that UMTS can be used for real-time games when the UDP protocol is used, and when the game software compensates for the network latency. For most wired online games, the gameplay gets affected when the latencies are higher than 150 ms. For mobile online games played over UMTS, at least twice the latency must be tolerated. This means that the game must be designed to minimize the effects of latency carefully. For collision handling, it is important that local collision management is used to give the player an immediate response. The server can be used to forward messages and ensure consistencies in player-to-player interaction. Mobile multiplayer real-time games must be design carefully to minimize the negative effects of latency. Important design considerations for such games are the speed of game objects, restriction of sudden direction changes, utilization of animation to camouflage visual or game world inconsistencies, local collision detection, use of movement prediction, and bundling of messages. The purpose of this work presented in this paper was to present a practical implementation of a game designed for stress testing network issues in a wireless environment similar to typical online games. The proposed solution for handling consistency is based on a variety of previous works where we through user tests found a suitable balance between the client and the server for determining and handling collisions. We hope our work can inspire others to pursue development of and research on wireless multiplayer real-time games.

Acknowledgments

The authors would like to thank Richard Taylor at the Institute for Software Research (ISR) at University of California, Irvine (UCI) for providing a stimulating research environment and for hosting a visiting researcher from Norway. The Leiv Eriksson mobility program supported by the Research Council of Norway has sponsored this work. They would also like to thank the editor and the reviewers of IJCGT for providing useful feedback used to improve the paper.

References

[1] T. W. Brignall III and T. L. Van Valey, "An online community as a new tribalism: the world of warcraft," in *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, IEEE Computer Society, Waikoloa, Hawaii, USA, 2007.

[2] mDisney-Studios, "Pirates of Caribbean Multiplayer Game," May 2009, http://disney.go.com/disneymobile/mdisney/pirates.

[3] J. Krikke, "Samurai Romanesque, J2ME, and the battle for mobile cyberspace," *IEEE Computer Graphics and Applications*, vol. 23, no. 1, pp. 16–23, 2003.

[4] CipSoft, "Tibia Micro Edition—the first mobile online role-playing game," May 2009, http://www.tibiame.com.

[5] C. Hiedels, C. Hoff, S. Rothkugel, and U. Wehling, "UbiSettlers—a dynamically adapting mobile P2P multiplayer game for hybrid networks," in *Proceedings of the 5th Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom '07)*, pp. 109–113, White Plains, NY, USA, March 2007.

[6] D. McCaffery and J. Finney, "Low latency optimisation of content based publish subscribe for real-time mobile gaming applications," in *Proceedings of the 4th International Workshop on Distributed Event-Based Systems (DEBS '05)*, vol. 4, pp. 438–443, Columbus, Ohio, USA, 2005.

[7] A. Chandler and J. Finney, "Rendezvous: supporting real-time collaborative mobile gaming in high latency environments," in *Proceedings of the ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, pp. 310–313, ACM, Valencia, Spain, 2005.

[8] J. Brun, F. Safaei, and P. Boustead, "Managing latency and fairness on networked games," *Communications of the ACM*, vol. 49, no. 11, pp. 46–51, 2006.

[9] D. R. Jefferson, "Virtual time," *ACM Transaction on Programming Languages and System*, vol. 7, pp. 404–425, 1985.

[10] M. Mauve, "How to keep a dead man from shooting," in *Proceedings of the 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS '00)*, pp. 199–204, Enschede, The Netherlands, October 2000.

[11] G. Armitage, "Sensitivity of quake3 players to network latency," in *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, Berkley, Calif, USA, November 2001.

[12] IEEE, "IEEE standard for distributed interactive simulation—communication services and profiles," Tech. Rep. IEEE Standard 1278.2-1995, IEEE, Washington, DC, USA, 1995.

[13] M. Busse, B. Lamparter, M. Mauve, and W. Effelsberg, "Lightweight QoS-support for networked mobile gaming," in *Proceedings of the 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, ACM, Portland, Ore, USA, 2004.

[14] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool, "The effects of loss and latency on user performance in unreal tournament 2003," in *Proceedings of the 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, ACM, Portland, Ore, USA, 2004.

[15] P. Quax, P. Monsieurs, W. Lamotte, D. De Vleeschauwer, and N. 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 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, ACM, Portland, Ore, USA, 2004.

[16] M. Dick, O. Wellnitz, and L. Wolf, "Analysis of factors affecting players' performance and perception in multiplayer games," in *Proceedings of the 4th ACM SIGCOMM Workshop on Network and System Support for Games*, ACM, Hawthorne, NY, USA, 2005.

[17] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu, "The effect of latency on user performance in Warcraft III," in *Proceedings of the 2nd Workshop on Network and System Support for Games*, ACM, Redwood City, Calif, USA, 2003.

[18] L. Pantel and L. C. Wolf, "On the impact of delay on real-time multiplayer games," in *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, ACM, Miami, Fla, USA, 2002.

[19] K.-T. Chen, P. Huang, and C.-L. Lei, "How sensitive are online gamers to network quality?" *Communications of the ACM*, vol. 49, no. 11, pp. 34–38, 2006.

[20] T. Henderson and S. Bhatti, "Networked games—a QoS-sensitive application for QoS-insensitive users?" in *Proceedings of the ACM SIGCOMM Workshop on Revisiting IP QoS: What Have We Learned, Why Do We Care?* pp. 141–147, ACM, Karlsruhe, Germany, 2003.

[21] J. Nichols and M. Claypool, "The effects of latency on online Madden NFL football," in *Proceedings of the 14th International Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 146–151, ACM, Cork, Ireland, 2004.

[22] T. Fritsch, H. Ritter, and J. Schiller, "CAN mobile gaming be improved?" in *Proceedings of the 5th ACM SIGCOMM Workshop on Network and System Support for Games*, ACM, Singapore, 2006.

[23] Y. Zhu, Y. Liu, H. Ngan, et al., "Shift coding: efficient state update in mobile peer-to-peer multiplayer games," in *Proceedings of the International Conference on Parallel Processing Workshops (ICPPW '07)*, Xi'an, China, 2007.

[24] T.-C. Chiueh, "Distributed systems support for networked games," in *Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS '97)*, Cape Cod, Mass, USA, 1997.

[25] C. Diot and L. Gautier, "A distributed architecture for multiplayer interactive applications on the Internet," *IEEE Network*, vol. 13, no. 4, pp. 6–15, 1999.

[26] T. Alexander, *Massively Multiplayer Game Development*, Charles River Media, Brookline, Mass, USA, 1st edition, 2003.

[27] H. Batista, V. Costa, and J. Pereira, "Games of war and peace: large scale simulation over the Internet," in *Proceedings of the 7th International Conference on Virtual Systems and Multimedia (VSMM '01)*, Berkley, Calif, USA, 2001.

[28] J. Aronson, *Dead Reckoning: Latency Hiding for Networked Games*, Gamasutra, 1997.

[29] Z. Qu, H. Gao, and Y. Zhu, "Research on high-accuracy position prediction algorithm in online game," in *Proceedings of the International Symposium on Electronic Commerce and Security (ISECS '08)*, pp. 78–81, 2008.

[30] J. Vogel and M. Mauve, "Consistency control for distributed interactive media," in *Proceedings of the ACM International Multimedia Conference and Exhibition*, pp. 221–230, ACM, Ottawa, Canada, 2001.

[31] M. Schoettner, M. Wende, R. Goeckelmann, T. Bindhammer, U. Schmid, and P. Schulthess, "A gaming framework for a transactional DSM system," in *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid '03)*, pp. 502–509, Tokyo, Japan, May 2003.

[32] A. Chandler and J. Finney, "Rendezvous: supporting real-time collaborative mobile gaming in high latency environments," in *Proceedings of the ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACM, Valencia, Spain, 2005.

[33] J. Brun, F. Safaei, and P. Boustead, "Managing latency and fairness on networked games," *Communications of the ACM*, vol. 49, no. 11, pp. 46–51, 2006.

[34] Y.-J. Lin, K. Guo, and S. Paul, "Sync-MS: synchronized messaging service for real-time multi-player distributed games," in *Proceedings of the 10th IEEE International Conference on Network Protocols (ICNP '02)*, Paris, France, November 2002.

[35] S. Zander, I. Leeder, and G. Armitage, "Achieving fairness in multiplayer network games through automated latency balancing," in *Proceedings of the ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACM, Valencia, Spain, 2005.

[36] C. GauthierDickey, D. Zappala, V. Lo, and J. Marr, "Low latency and cheat-proof event ordering for peer-to-peer games," in *Proceedings of the 14th International Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 134–139, ACM, Cork, Ireland, 2004.

[37] M. Berger, S. Lima, A. Manoussakis, J. Pulgarin, and B. Sanchez, "A performance comparison of TCP protocols over mobile ad hoc wireless networks," in *Proceedings of the Electronics, Robotics and Automotive Mechanics Conference (CERMA '06)*, vol. 2, pp. 88–94, Cuernavaca, Mexico, September 2006.

[38] M. C. Chan and R. Ramjee, "Improving TCP/IP performance over third-generation wireless networks," *IEEE Transactions on Mobile Computing*, vol. 7, no. 4, pp. 430–443, 2008.

[39] B. S. Bakshi, P. Krishna, N. H. Vaidya, and D. K. Pradhan, "Improving performance of TCP over wireless networks," in *Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS '97)*, pp. 365–373, Baltimore, Md, USA, May 1997.

[40] M. Satyanarayanan, "Fundamental challenges in mobile computing," in *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing*, pp. 1–7, ACM, Philadelphia, Pa, USA, 1996.

[41] S. Hadap, D. Eberle, P. Volino, M. C. Lin, S. Redon, and C. Ericson, "Collision detection and proximity queries," in *Proceedings of the International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '04)*, ACM, Los Angeles, Calif, USA, 2004.

[42] S.-H. Kim, B.-J. Choi, M.-S. Jung, and K.-S. Park, "A real time network game system based on retransmission of N-based game command history for revising packet errors," in *Proceedings of the 5th ACIS International Conference on Software Engineering Research, Management & Applications (SERA '07)*, pp. 917–923, Busan, South Korea, 2007.