Content Scheduling in Multimedia Interactive Mobile Games

M. Mohsin Saleemi, Kristian Nybom, Jerker Björkqvist, Johan Lilius
Department of Information Technologies
Åbo Akademi University, Finland
msaleemi@abo.fi knybom@abo.fi jbjorkqvist@abo.fi jlilius@abo.fi

Abstract

In this paper, we study how to implement interactive multimedia services using a DVB-H broadcast channel combined with a point-to-point channel, such as 3G or GPRS. We study the problem in the context of a location-based interactive mobile game. The technical challenge is to schedule the sending of data over the broadcast channel while maintaining Quality-of-Service, that is, sending the right data to the right user at the right time to provide a seamless interactive experience. We explore design issues and problems related to the scheduling of content in the game, present a usecase study to describe scheduling problems and propose a content scheduling algorithm to solve these problems. Moreover, we provide a simulation of the system and the experimental results to show how different game parameters influence the in-time delivery of the multimedia content to the players. We conclude that most of the problems involved with our approach can be expressed as the problem of defining delivery deadlines for a scheduling algorithm.

Author Keywords

Interactive applications; content scheduling; algorithm; multimedia; location-based game.

Introduction

With the evolution of handheld devices, positioning techniques and new transmission technologies, a new class of location-based mobile applications (Dix, 2000) has been made feasible and has become an evolving research area. These mobile location based games involve wireless networks, navigation techniques, movement and location of the players, and hypermedia content that is used by the players while playing the game (see Boll, 2003 and Klante, 2005 for more detail). These games are realized with wireless handheld devices together with GPS receivers in a PtP communication network over Web infrastructure. In our interactive location-based mobile game, we use a new wireless broadcast technology (DVB-H) with the old theme of traditional location-based games. The motivation for using DVB-H is that the substantially larger bandwidth will provide a more immersive environment for the players.

The technical challenge is to schedule the sending of the data over the broadcast channel while maintaining Quality-of-Service (QoS) to provide a seamless interactive experience. These kinds of applications are sensitive to delay as the players would need to wait unnecessarily for the content to identify their next tasks which may in turn be the difference between winning and

losing. In this paper we present a solution to this problem. We discuss the challenges and problems that are involved in designing these kinds of interactive applications due to the usage of wireless broadcast technologies. The paper justifies the need of dynamic scheduling in the game, explores the scheduling issues, and presents a scheduling algorithm. Moreover, it exposes the overall interactive communication setup to illustrate how the broadcast channel is used in this interactive mobile game. We chose the interactive mobile game platform to explore the issues of content scheduling because this kind of real-time system involves dynamic scheduling based on the active interactions of the players. Using this approach, we investigate the concepts and issues involved in this new converged environment. Most of the literature in real time systems deals with periodic deterministic tasks which may be prescheduled (Chung, 1988; Shin 1988). The aperiodic scheduling has different behavior in which dynamic scheduling is performed when some event or interaction occurs and it can not be prescheduled as it depends on time of interaction. Aperiodic scheduling is required in multiplayer interactive multimedia games where scheduling of large content is performed according to the associated deadlines and it has not been thoroughly investigated. This paper models the scheduling problem in broadcast network environment and deals with the design issues to provide dynamic scheduling to handle the interactions and content demands of the users. The main contribution of this work includes: (1) modeling the scheduling problem for QoS-sensitive games; (2) proposing a scheduling solution based on the priority assignment strategy. The proposed approach is validated through a simulation model and the experimental results.

The rest of the paper is organized as follows: Section 2 provides the definition and categories of interactivity in this kind of system. Section 3 describes the game model and motivation behind the game scenario. It also provides the logical structure of the game server and its overall implementation sketch. Section 4 highlights the importance of multimedia content scheduling in this kind of system. It also provides usecase to describe the scheduling problem and propose an algorithm to solve the content scheduling. Section 5 introduces the simulation setup used to evaluate the system and the experimental results. Finally, we describe the future work and conclude the paper in Section 6.

2-Definition of Interactivity

Many forms of interactive genres, formats and content in the context of Interactive TV are discussed by Jensen (2005). In this paper we are not so much concerned with different kinds of interactive TV, but more with the technological issues in the implementation of interactivity in a TV environment. In this paper, an interactive service is defined as a service where the action of one user affects the internal status of the server providing the service, allowing all other users to experience the changes. Using this definition, a web discussion forum, for example, is an interactive service. However, the interactivity in such a service is dependent on the user's actions, i.e. the user has to request some data in order to be interactive. This can be defined as pull-type interactive service. Here our special interest is when the interaction affects a common stream of data to many users. A typical example of such an interactive service is the SMS-based chats in television, where you can send SMS messages to a server, which then broadcasts the messages as a video stream. This can be defined as a push-type interactive service.

We can also define interaction from the point of view of the user. In this case we can distinguish between passive and active interaction. The passive interaction is used for interacting with the server periodically, i.e. sending the state information with a fixed interval. The active interaction, on the other hand, is initiated when the user has to make a decision that affects the state of the game and other users. An active interaction can for instance be when a player requests some information from the server to perform an action or when a player makes a choice to accept or reject some action that has influence on other players. An example of an active interaction in the game is when the player accepts a quest that is no longer available to other players. In this interactive game we are dealing with both active and passive interactions.

3-Game model

Åbot is a multiplayer interactive mobile game in which the movements and locations of the players trigger all the actions. GPS is used for obtaining the location information. The communication between the terminals and the game server is done using two different channels. GPRS/3G is used as the uplink to the server to send information from the terminals to the server using an IP connection whereas a wireless broadcast medium (DVB-H) is used to communicate with the players. The real physical environment serves as playing area for the game. The conceptual game platform is shown in Figure 1. The game provides the bridging of the physical locations and objects to a virtual map on players' handheld screens. The game has two different kinds of tasks for the players. The players can collect bots, which are virtual containers that are dispersed on the game map by the server. Bots contain hints, points, equipment etc, which are useful for solving the quests. Quests are the second kind of tasks in the game. Quests in their simplest form involve finding the locations by solving the puzzles, and they can also be constrained in time. Quests also form the story of the game. The players in the game are equipped with the handheld terminals having mobile phone networks and capable of receiving wireless broadcast data. Additionally, the players are equipped with a GPS device to get the accurate positions on their mobile phones. The overall client-server communication system and the communication methodology of the game are explored in detail in Nybom (2007).

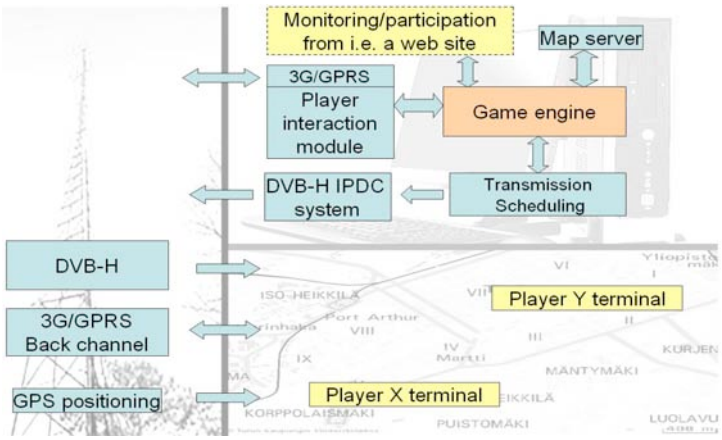

Figure 1: Game Platform

The novelty of the game is that we use DVB-H, a broadcast medium, to deliver data to the players. Our motivations for using DVB-H are the following. Firstly, DVB-H is optimized for the handheld terminals and offers much higher data rates than cellular networks, like 3G or GPRS. In the game design we want to explore this possibility of giving a more immersive experience for the players, but this is not the topic of this paper. Secondly, DVB-H is a relatively new technology. We want to use the context of a location based game to explore this technology in a converging environment where both, the cellular and the broadcast, technologies are used. In this way we hope to expose new research issues and challenges

Implementation sketch

The implementation is realized with a PC connected to the Internet, which is used for delivering IP data unidirectionally to the broadcast operator and bidirectionally to the cellular operator. The broadcast channel uses UDP and the PtP point-to-point uses TCP. Figure 2 shows the implementation sketch. The players connect to the server using the PtP connection and negotiate a subscriber account, which contains all player specific information. The implementation sketch of the system also includes an overview of the server architecture and continues with a brief overview of the content scheduling methodology.

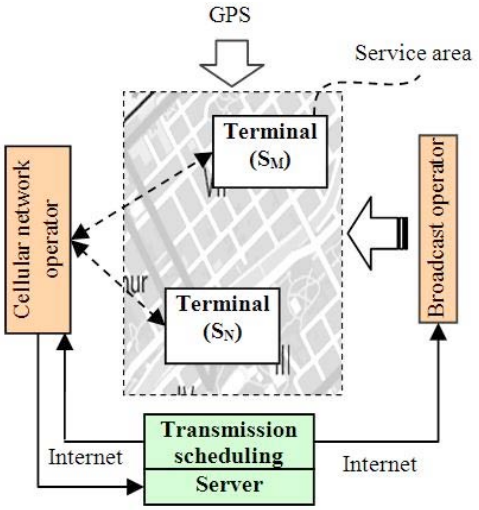

Figure 2: Implementation sketch

Server architecture

Figure 3 shows the logical block diagram of the server implementation. The game engine is the entity which keeps track of the state of the game. Its main tasks are to update the positions of the subscribers and based on that to decide on what new information each subscriber (or group of subscribers) needs next. This also involves predicting ahead what data a particular subscriber will require in the near future. This is explained in more detail in the next section. The media engine is the entity which handles all the objects, in the media database, associated with the service. The media engine produces all messages which contain service objects. In our motivating game scenario such objects can be documents, images, storyline animations done in Flash, video clips, etc. In addition, the game engine may compose dynamically directed live video and audio feeds reflecting for instance the state of the game. The system encourages the subscribers to produce their own content to the service by uploading digital images or video clips. The subscriber manager keeps track of users or players that are currently using the service

interactively. The subscriber assigns correct IP addresses to the packets encapsulating data destined to a certain subscriber or group. In addition, the subscriber manager handles congestion control for individual terminals. During login, the memory capacity of each terminal is negotiated to the server and taken into account before transmission, in order to avoid buffer overflow in the receiving terminals.

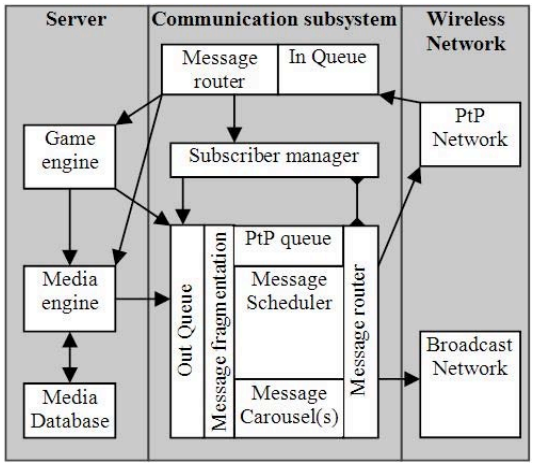

Figure 3: Server logical block diagram

The message handling in the communication subsystem is realized using message queues for input and output messages, a message carousel and a message router. Messages may be of any length and have arbitrary deadline for delivery. This imposes the need for a pre-emption mechanism for message transmission. In other words, an urgent message must be able to pre-empt a less important message. The solution to this in the presented system is to use a message scheduler.

In addition, we assume that the number of available downstream channels is arbitrary, that they possess different characteristics, and that these should be taken into consideration when the message routes are chosen. In case of a PtP network, the communication is reliable and straightforward. However, a broadcast channel such as DVB-H is unidirectional and unreliable. Also, the reception condition for each terminal is different and from this point of view, the subscribers cannot be seen as a homogenous group. Using the broadcast channel for delivering messages to different sets of subscribers, even individuals, requires intelligent decisions on usage of the available bandwidth. Furthermore, since the broadcast channel is unreliable, the message should, if needed, be retransmitted. Retransmission in the server is handled by the message carousel which contains data that is either not yet transmitted or not yet acknowledged by all recipients. Raptor codes can be used for reliable delivery using broadcast channel. A Raptor code is a Forward Error Correction (FEC) code standardized in DVB-H as an optional code for IP Datacasting. Raptor encoded messages in the carousel are not retransmitted but are instead replaced with new encoding symbols. Longer messages need to be fragmented into smaller messages that the scheduler can handle. The fragmentation process preserves the requirements of the original message.

The message router for the output messages receives messages from the PtP queue and the message carousel. The messages are encapsulated into IP packets and routed to the address

acquired from the subscriber manager. The predefined network is consequently used for delivering the message. Input messages are always received using the PtP network. The messages are then routed, based on type of message, to the correct module, which then handles it accordingly.

4-Scheduling of content

One challenge for this game environment is to provide all user terminals with data that is needed at the particular moment. The assumption is that the available bandwidth and the user terminal memory capacity are limited, hence the user terminals must be provided with data on demand. Additionally, some of the data sent is dependent on the status of the game; hence it is not available before it is needed. The demand for data is assumed to be predicted by the game server. Thus the problem of providing media to the players at the right time is solved by proposing a content scheduling algorithm. Using this algorithm, it can be made sure that every player receives the data intended to him/her before its deadline. The game server receives position messages from each player after every 5 seconds and periodically broadcasts location updates of all the players logged into the game. The server predicts the movements of the player towards the possible quest locations and also calculates the distances and expected arrival times of the players to these quest locations based on the received position messages. We use a threshold time as a decision point at which the data should be scheduled by the server so that it can be sent to the players at the quest location before its deadline. The threshold value gives the latest time at which the data should be scheduled so it guarantees the delivery of the content before its deadline. This threshold value is globally selected and is same for all the quest locations. The minimum value of the threshold is calculated as

$$\textit{Minimum threshold} = \textit{Size of data to be sent} / \textit{Bandwidth available}$$

In this paper, we are only considering scheduling the quest data that is to be sent by the server when a player moves to a quest location. Other data related to player's fighting and server queries about some choice are not dealt with in this work. Quests have fixed locations and the game server knows all these locations before the game starts. The server should predict the next location of the player by observing its movement and send the data before the player actually requests it. Figure 4 describes the usecase for the scheduling problem.

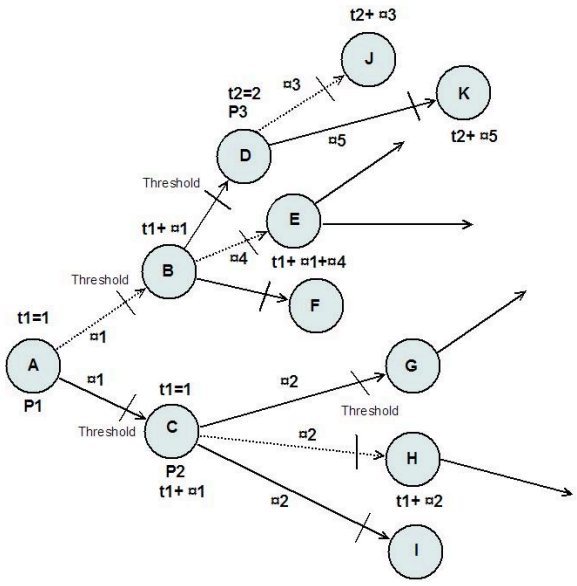

Figure 4: Usecase

Assume that the player P1 is at location A at some specific time, say t=1. He has two choices at point A as to where he can go and get the quests. These locations are referred to as quest locations and denoted by B and C in Figure 4. Let's assume that the distance to these locations from A is the same e.g. 700 m and the expected time of arrival (ETA) is Φ1=10 minutes. As each player sends their position updates to the server after every 5 seconds, the server knows the current location of the player and its movement towards the quest location. At each point towards the quest location, the server recalculates the ETA and updates the position of the player. A globally selected threshold value is associated with each quest location in the game. Let's assume that the threshold value is 1 minute. When the player reaches the threshold point of the quest location B, the server schedules the data needed by the player at that location. The threshold value is dependent on the data to be sent and available bandwidth to send data. We assume error free transmission to simplify the scheduling problem. Therefore, the threshold value guarantees the delivery of content before its deadline. If another player is at location C at the same time t=1, he has three choices of G, H and I where he can go to get the quests. Let's assume the distance from C to G, H and I is the same, 1 km, ETA is Φ2= 12 minutes and the threshold value is 1 minute. When the player P2 reaches the threshold point of any of these quest locations, the server schedules the content for delivery. As the ETA of P1 is less than that of P2, the content needed by P1 is scheduled before P2. The server should schedule the data in such a way that the player at point A moving to any location, which has shorter ETA, should get priority over the player moving from location C to G, H or I. If another player P3 is at location D at time t=2 and moving to a location which has distance 100 m, ETA Φ3= 3 minutes and threshold value 1 minute, when P3 reaches at the threshold point of J or K the server should calculate its proper position in the schedule and reschedule the data if needed before sending. In this way, the game evolves for every player who is moving towards the quest locations and the quest data that is needed by the players in near future should be efficiently sent by the server before it is actually requested.

The terminals need to discard data that is not intended for them but scheduled and sent due to unpredictable behavior of the players when they have choices for the next locations which are close to each other. The dotted lines in the figure show movements of the players towards the

next locations. The scheduling algorithm must be efficient enough to schedule the data for each player according to its deadlines and reschedule the data if the player changes his mind in the way and starts moving to other quest location. In that case when the game server predicts the player movement towards the next quest location, the scheduled data for not-taken options will be discarded by the scheduler.

The main steps and pseudo code of the scheduling algorithm are mentioned below.

For every position message, do:

```
1. Update position of the player
2. Calculate the distances to next quest locations
3. Iterate through quest locations list
   if ( ! (questLocList [index] == player.Loc)
   {
    calculate ETA to next quest locations
     if ( ETA <= thresholdETA)
      {
       // prepare and schedule data for player
       outQueue.pushMessage (questData)
      }
   }
4. For each message to be inserted in outQueue,
 // iterate through the outQueue
 for (index=0 ; index < this.index ; index++)
 {
 insert message according to ascending ETA
 }
5. Send data with the highest priority to carousel
    while (outQueue.length>0)
       {
        Carousel.send(outQueue.pop())
       }
```

5-Simulation Setup

For the evaluation of the proposed algorithm we modeled our system in Ptolemy II tool, a Java-based component assembly framework with a graphical user interface. We selected Ptolemy as it facilitates modeling, simulation and design of real-time embedded systems and it meets requirements of our system. We modeled the system in the Discrete Event (DE) domain of the Ptolemy. The reason for choosing the DE domain is that it provides a general environment for time-oriented simulations of queuing systems and communication networks. In this domain, actors communicate by sending tokens across connections. The token sent and the time at which the action took place constitutes an event in the DE domain. Upon receiving an event, the destination actor is activated and a reaction takes place. The reaction may change the internal state of the actor and possibly generate new events, resulting in further reactions. A DE domain

scheduler ensures that events are processed chronologically. All these features fulfill the requirements of our system in which players move to the different quest locations which are dispersed by the game server. The game server knows about each quest location and the multimedia data need to be sent at these locations before the game start. In this simulation we make the following assumptions: 1) We assume that the quest locations are randomly dispersed - it helps to trigger the player's interactions at arbitrary times and hence causes aperiodic scheduling in which data is rescheduled in the queue upon arrival of higher priority data and sent through the carousel once the channel is available; 2) For each experiment performed, the data size is kept same for all the quest locations. This means that all the players will receive the same amount of data when they reach at a quest location; 3) We perform the experiments under the assumption that the channel is error free; 4) It is also assumed that the game playing area is relatively small to better analyze the behavior of the system. This small playing area defines the pace of the game and how quickly the game evolves. It is an important factor as it influences the bandwidth required to send data by specifying the load on the network. For example, if we have a very small playing area where the quest locations are very near to each other then we can not send large amount of multimedia content and the number of players are quite restricted. On the other hand, if we have very big playing area where quest locations are dispersed at longer distances to each other then large amount of data and larger number of players can be supported and the game can be thought of as a massively multiplayer game. This factor is related to the design parameter of the game and the game model simulation presented in this paper will work for both cases as the objective here is to observe the delay in sending the data to the game players by systematically varying the different parameters. In this simulation we concentrate on Quality of service (QoS) issues of the system to study the effects of changing the number of players, data size, locations and threshold values and check the bandwidth utilization and delay in sending data to the players. The behavior of the system is observed to check how these factors affect the in-time delivery of data to the players and how much bandwidth is needed for a certain number of players and quest locations.

Performance metrics

In these experiments, we focus on the most important factor of QoS, which is network latency or delay. By varying different parameters, we observe how they affect the delivery of multimedia data to the players. The data to be delivered at each quest location has its expected time of arrival and the actual time of delivery should not be greater than the expected time in order to maintain immersive experience of the game play. We used a number of metrics to monitor the performance and behavior of the system. These metrics are the following.

- Players --- number of players logged in the game server
- Locations --- quest locations at which data needs to be delivered
- Threshold --- threshold value at which data needs to schedule in the queue
- Data --- amount of data that needs to be sent to the players

As we are mainly concerned with the delay in these experiments, we use the actor TimedPlotter of Ptolemy to plot the signals as in this signal plotter the horizontal axis represents the time. For the game, we use one channel of DVB-H which has bandwidth of 384 Kbps which is used to send the data to the players. The size of the multimedia content that is to be sent at each quest location depends largely on compression technique used. In general, the size of a 50 seconds

long animated clip in QCIF format with resolution 176*144 using H.264 compression techniques is approximately 190 Kb. In this simulation we assume that each player needs 200 Kb data to be sent when it is at the threshold point of a particular quest location. This gives a better understanding of the system to explore the system behavior when the different parameters are changed. The goals of these experiments are to analyze how many players the server can accommodate under a certain limited bandwidth while maintaining the QoS, assess the worst case when there is most load in the channel, to determine how threshold value affects the in-time delivery of the data and find the effects on data delivery time when the channel is busy.

Experimental results

We perform the basic experiments on the modeled system by varying the different parameters one by one and analyze the results that how they affect the delay of sending required data to the players before the deadline. The actual time of data arrival is the time when the data is actually sent to the players through the broadcast channel, whereas the expected time is calculated by the server based on the player's movement and speed towards a quest location.

Figure 5 shows the results of first case in which we have 10 quest locations and 10 players are logged into the game. The bandwidth is fixed at 384 Kbps. Each player requires a 50 seconds animated clip, that is, 200 Kb data when it is at the threshold point of any quest location.
We see that with this configuration, the server handles all the 10 players without missing any deadline as the actual arrival time of data is always less than the expected time for all the players; hence, the sever sends data to the players before their respective deadlines.

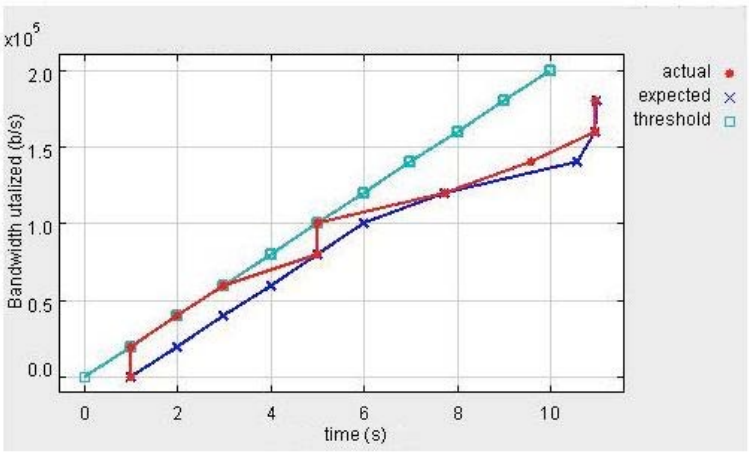

Figure 5: B/W = 384 Kbps, players = 10, locations = 10, data sent to each location = 200 Kb

Figure 5 also shows the bandwidth utilization for sending the data to 10 players. We see that only 175 kbps bandwidth is used for sending data to all these players.

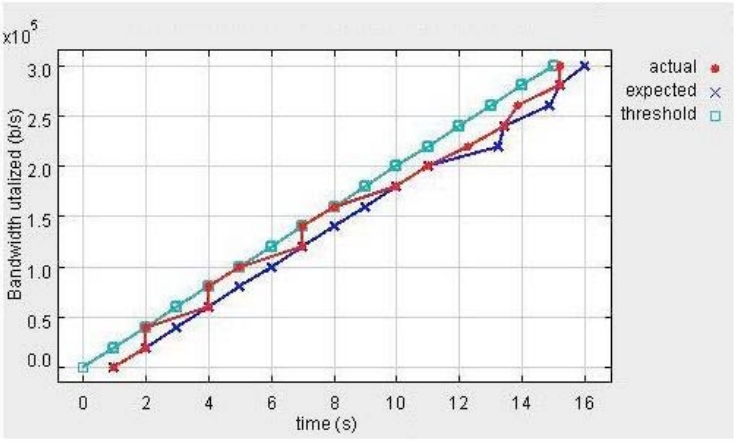

Figure 6: B/W = 384 Kbps, players = 15, locations = 10, data sent to each location = 200 Kb

We then increase the number of players to 15 keeping the data 200 Kb for each location and run the simulation again. Figure 6 shows that the deadlines for all 15 players are met as in the case of Figure 5. The only difference is the bandwidth utilization. The bandwidth utilization is 300 Kbps for this case which was 175 Kbps for the previous case. The results show that increasing the number of players by 50 percent causes the increase in the bandwidth utilization about 75 percent. It means that the bandwidth utilization does not increase linearly with the increase in number of players.

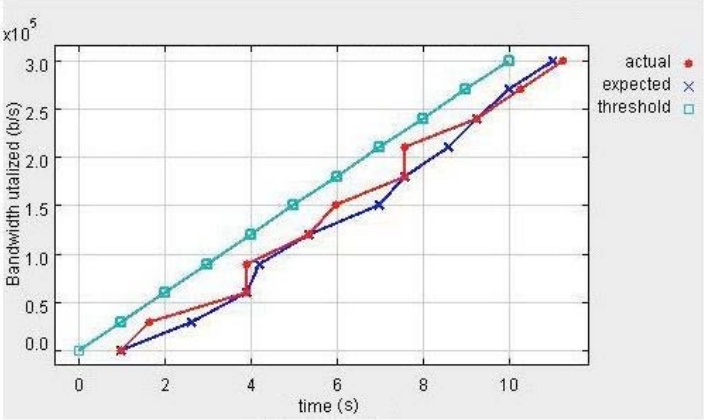

Figure 7: B/W = 384 Kbps, players = 10, locations = 10, data sent to each location = 300 Kb

In the next step, we assume that each of the 10 players require an animated clip of 75 seconds instead of 50 seconds at the quest locations. Hence, the data size to be sent to each player has increased to 300 Kb. This configuration increases the average play time of the game. Figure 7 shows the results. We see the overhead at point 9 where the deadlines are beginning to miss. The reason for this delay is due to the position updates that are sent by the server to each player. The size of these position updates is small, i.e. a few Kb as compared to the data size and it depends on the number of players. In this case, the remaining bandwidth is used by the server to send position updates which causes the deadlines to miss.

In the next experiment, we keep the configurations of last experiment the same and increase the threshold value. Figure 8 shows the results. In this case, we see that all the deadlines for 10 players are met. The reason is that increasing the threshold value means the data will be scheduled a little ahead of time and can wait more for the empty slots in the carousel hence meeting the deadlines.

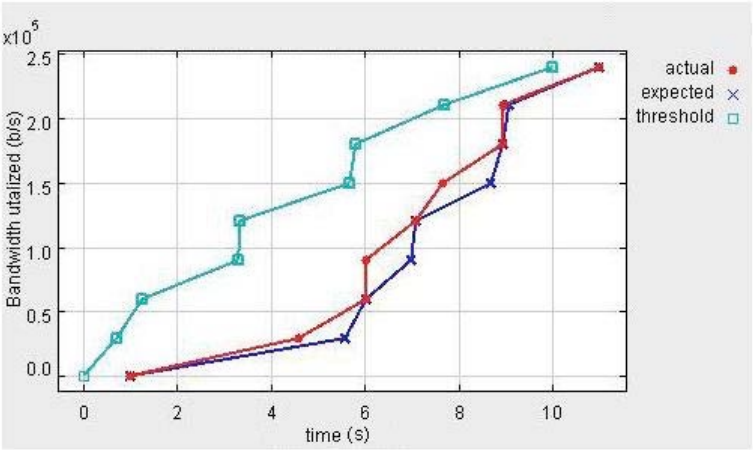

Figure 8: B/W = 384 Kbps, players = 10, locations = 10, data sent to each location = 300 Kb, threshold increased

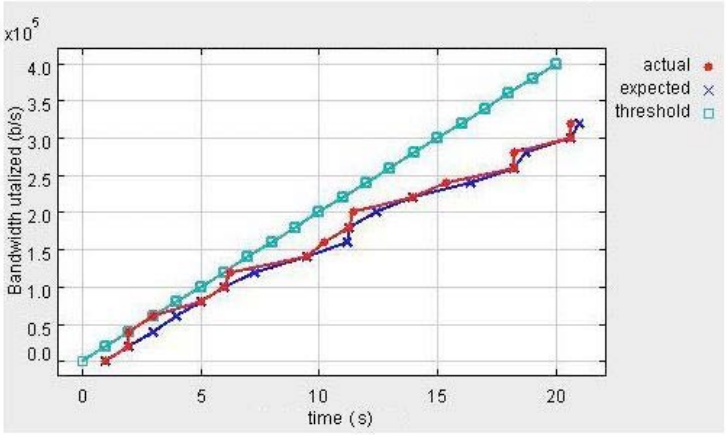

Figure 9: B/W = 384 Kbps, players = 20, locations = 10, data sent to each location = 200 Kb

In the next experiment, we increase the number of players to 20 keeping number of locations at 10 and data size 200 Kb. Figure 9 shows that all the deadlines for 20 players are met. The reason is that as the quest locations are fewer than the number of players, more players move to the available quest locations at almost same time. The server multicasts the same data to players who move to the same quest location, hence avoiding multiple sending of same data, which reduces the bandwidth utilization.

To check the worst case, we increase the number of quest locations to 20 and each of the 20 players move to different quest locations at almost same time. In this case the data sending deadlines are very close to each other. Figure 10 shows the results. We see that at point 7 and after 12 there are sharp increases in the delay and the deadlines are missed for 5 players as the bandwidth limitation of 384 Kbps is reached.

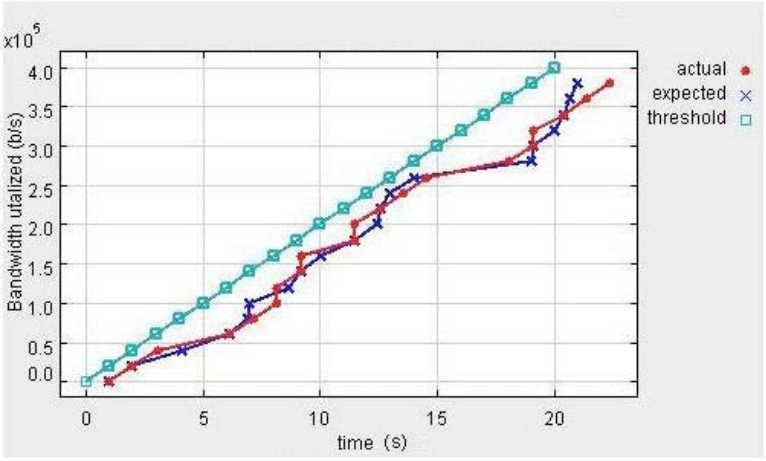

Figure 10: B/W = 384 Kbps, players = 20, locations = 20, data sent to each location = 200 Kb

In the next step, we increase the number of locations to 40 while keeping the number of players 20 and data size 200 Kb. The results of these configurations are very interesting as we just increase number of locations while number of players is the same at 20 but there are many deadlines missed as shown in Figure 11. The reason for this delay is that as the playing area is small and when we increase the number of quest locations, some locations become so near to each other that a single player is in the range of threshold of more than one location. In this case, the server has to schedule and send data of both locations although the player moves to only one location. It not only increases the overhead on server but terminals also need to discard data that is not intended for them.

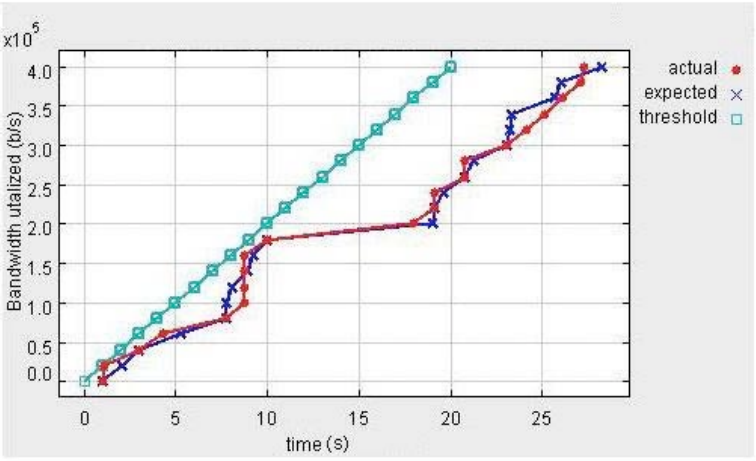

Figure 11: B/W = 384 Kbps, players = 20, locations = 40, data sent to each location = 200 Kb

From the simulation results, one can see that the proposed scheduler works reasonably well in a small playing area, as long as the number of players remains below 20 and the data sizes for each quest location is below 300 Kb. Clearly, the worst case occurs when all players simultaneously move to distinct nearby locations and therefore require different data, since the amount of data that has to be sent is drastically increased. However, since the scheduler is designed for a location based game in the physical world, one can assume that this is a rare event. A more probable event is that a number of players are moving to the same location and therefore require the same data, which improves the performance of the scheduler since data is broadcasted, i.e. this specific data is sent only once.

One should note that in the simulations, the data sizes are quite small compared to the available bandwidth. If the data sizes were larger, this would increase the need for the scheduler, since the goal is that data should be available at every receiver when the user reaches a special location. In effect, this would increase the thresholds and the requirement on content fragmentation. The most difficult situation for the scheduler to cope with is the situation where there are several special locations close to each other. This is because the scheduler should predict the most probable destination for every player and if a player moving to one of these locations suddenly changes his mind, the prediction goes wrong and new data has to be transmitted.

6-Conclusions and Future Work

In this paper we have introduced a data scheduler for an interactive location based game, which relies on a wireless broadcasting technology for delivering the data. The benefit of using a broadcasting technology is the high transmission rate and the possibility to send the same data to multiple receivers simultaneously. However, the major issue in this scenario is to be able to meet deadlines when certain data should be available at certain receivers. The proposed scheduler uses the receivers' GPS locations to predict the most probable destination to where each user is headed. Through this information, the scheduler derives the latest point in time, i.e. the threshold, when the data should be transmitted to meet the deadline. We have shown that the threshold is dependent on the number of players in the game, on the number of possible special locations, and on the size of the data that should be transmitted for every location.

Simulations for the scheduler have been performed under the assumption that data is delivered without errors. This is slightly unrealistic, but it gives an indication on whether the proposed scheduler performs as expected. Using the proposed scheduler, deadlines are met as long as the number of players remains below 20 and the data sizes are below 300 KB. The simulation results indicate that in order to cope with the varying nature of the game, the thresholds should be chosen with some slack whenever possible. If the erroneous nature of wireless broadcast networks is taken into account, the slack becomes even more important to give receivers sufficient time to recover the originally transmitted message. Clearly, this complicates the threshold determination as every user experiences different reception conditions. This, however, remains for future work. The same idea of content scheduling could also be used in the context

of ambient intelligence and profile-based TV programming i.e. programs whose content adapts according to the viewers preferences, location and time. We aim to work in this area in future.

References

Boll, S. and Krösche, J. and Wegener, C. (2003). Paper Chase Revisited - a Real World Game Meets Hypermedia. *In proceedings of the 4th ACM conference on Hypertext and Hypermedia*. Nottingham, UK.

Chung, E-Y. and Liu, J.W.S. (1988). Algorithms for scheduling periodic jobs to minimize average error. *Real-Time Systems Symposium*, pp.142-151.

Dix, A. and Rodden, T and Davies, N. (2000). Exploiting Space and Location as a Design Framework for Interactive Mobile Systems. *In ACM Transactions on Computer-Human Interaction (TOCHI)*, pp. 285-321.

Jensen, J.F. (2005). Interactive television: new genres, new format, new content. *In proceedings of the IE2005 Conference,* Sydney, Australia, pp. 89-96.

Klante, P. and Krösche, J. and Boll, S. (2005). Evaluating a mobile location-based multimodal game for first-year students. *In proceedings of the SPIE 2005*.

Nybom, K. and Kempe, J. and Saleemi, M. and Bjorkqvist, J. and Lilius, J. (2007). A Communication Methodology for Interactive Location-Based Mobile Games. *In Workshop on Interactive Applications for Mobile TV, EuroITV 2007*.

Shin, K.G. and Krishna, C.M. and Lee, Y.-H. (1988). Optimal resource control in periodic real-time environments. *Real-Time Systems Symposium*, pp. 33-41.