Skype4Games

Tonio Triebel
Praktische Informatik IV
University of Mannheim
Germany
triebel@informatik.uni-
mannheim.de

Benjamin Guthier
Praktische Informatik IV
University of Mannheim
Germany
guthier@informatik.uni-
mannheim.de

Wolfgang Effelsberg
Praktische Informatik IV
University of Mannheim
Germany
effelsberg@informatik.uni-
mannheim.de

ABSTRACT

We propose to take advantage of the distributed multi-user Skype system for the implementation of an interactive online game. Skype combines efficient multi-peer support with the ability to get around firewalls and network address translation; in addition, speech is available to all game participants for free. We discuss the network requirements of interactive multi-player games, in particular concerning end-to-end delay and distributed state maintenance. We then introduce the multi-user support available in Skype and conclude that it should suffice for a game implementation. We explain how our multi-player game based on the Irrlicht graphics engine was implemented over Skype, and we present very promising results of an early performance evaluation.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems - *distributed applications*

General Terms

Algorithms, Design, Experimentation

Keywords

Distributed Interactive Applications, Peer-to-Peer, Skype

1. INTRODUCTION

More and more players are attracted to interactive multi-player games. This is partly due to the young generation who grew up with computer-based video games and continues to play them and partly to the increasing performance of modern computers that allow the creation of attractive new games with better-than-ever graphics and real-time performance. There are now even professional game players who earn their living playing games.

However, modern networked multi-player games do not scale well. The reason is that they have a client-server architecture: the game status is maintained on the server and

distributed to all clients, either periodically or on request. With such an architecture, the number of clients attached to each server and therefore the maximum number of players is limited. Smed *et al.* [15] estimate that a server with a 10 Mbps ethernet connection can provide serializability for at most 2,551 clients. In their scenario each client sends 5 packets per second using the IPv6 communication protocol. Thus we decided to investigate the feasibility of distributed scalable multi-player games, based on a peer-to-peer approach, that scale well while still maintaining an acceptably short latency.

In order to implement such a distributed game we need a game idea, a graphics engine and a peer-to-peer system that we can use for distributed state maintenance. Developing all that from scratch would clearly go beyond our resources. Thus we decided to use an existing game idea (derived from the game *Descent*), the *Irrlicht* game engine for the graphics [9], and *Skype* as the peer-to-peer system [14]. Irrlicht and Skype have the advantage that they are free to use and both have an application programming interface, allowing us to put Irrlicht on top of Skype. An additional advantage of Skype is that it allows users to talk to each other while playing the game.

The remainder of this paper is structured as follows: In Section 2 we analyse current online games, including previous work, and we derive the requirements for our distributed multi-player game; Section 3 gives a brief introduction into the Irrlicht game engine as well as into Skype, In Section 4 we describe the architecture and implementation of our game, including our approach to distributed state maintenance; Section 5 presents initial, encouraging results of our performance measurements;Section 6 describes related work; and Section 7 concludes the paper.

2. ONLINE GAMES AND REQUIREMENTS

2.1 Game Fundamentals

Current online games can be categorised into four classes: role-playing games (RPG), real-time strategy (RTS), sports games (SG), and first person shooters (FPS). Each class has a different game flow and therefore creates a different network traffic. The most challenging demands are a great number of players, as they are common in RPG, and a low network latency, which is necessary in FPS or SG. The game we are building is a FPS with a possibly large number of participants.

Environment

Typically the environment of an online game consists of the game landscape and the entities such as space ships, objects, weapons and projectiles. Each entity has a model and characteristics like size, position and orientation. In addition, online multi-player games have to display both the entities of one's own game environment and those of all other participants visible in the actual scene.

Avatars

The virtual identity of a player is called *avatar*. By controlling an avatar the player can see, hear and interact with the game world or other human players. The range of an avatar's perception and interaction forms the *area of interest* (AOI). All actions must be guaranteed to be communicated to all other players located in the AOI and vice versa.

Network Support

The game traffic consists of entity data, control messages and player communication, such as text or voice. The entity data consists of information such as position, orientation or velocity of an object. This can be a large amount of data, due to a high number of entities. In fast games like FPS the transmission of entity data can be time-critical, and efficient multi-peer communication as well as a *low network latency* is required.

One further need of online games is *reliable data transmission*. Important events like the joining or the death of a player must be reflected consistently in each relevant instance of the game application. This does not hold for normal entity data where a retransmitted position of a fast-moving object would be obsolete by the time it arrives.

Speech

In order to coordinate complex game situations with large groups of players, it becomes more and more popular to use *speech*. Spoken commands are faster and more direct than written text. But the transmission of speech would further increase the load on the central server; as a consequence, there is no support for voice communication in many current games and additional software must be used.

2.2 Increasing Number of Online Players

The number of online gamers is increasing fast. A well known example is the MMORPG (Massively Multi-player Online Role-Playing Game) *World of Warcraft*. On January 11, 2007 Blizzard Entertainment announced: "World of Warcraft surpasses 8 million subscribers worldwide" [4]. Furthermore, the desire arises to bring together such a great number of players into one single game world.

Not only the number of players is changing, but also their gaming behaviour. In [6] the authors introduce a new category of players, the *hardcore gamers*. They spend more than 32 hours per week on gaming (more than 70% of their leisure time). And even further, now there are professionals who earn their living entirely from online games, such as electronic sports events and tournaments. In many countries they are celebrated like pop stars, and the prize money they can potentially obtain is substantial. On the Electronic Sports World Cup 2006 in Paris there was an overall prize money of 400,000 USD to be won. These professionals have developed a discerning sensitivity to network delays. We argue that the increase of simultaneously playing gamers and

their demands on the network lead to tasks that current game architectures cannot handle anymore.

2.3 Game Requirements

2.3.1 Latency

In [7] Henderson reports detailed measurements of latencies with the online game Half-Life. According to his results, most gamers experience delays of $50ms$ to $300ms$, with 95 % having delays of less than $533ms$. He also notices that most players are quite tolerant to longer absolute delays, provided that other players have similar delays.

With a peer-to-peer architecture the situation is slightly different, since there is no server and thus absolute or relative delay occure in a different context. Therefore the results can only be used as an approximate value.

2.3.2 Distributed Game Architecture

Current online games are built on a client-server architecture. The server maintains the entire state of the game world. The state information of all entities and players is collected from the clients, merged together into one consistent world state and communicated back to the clients. The game scene is rendered by the clients using the most recent state information.

This architecture works well for existing games, but has one major disadvantage: it does not scale. The server has to communicate with all the clients in order to fulfil this task. This induces an enormous amount of traffic leading to a bottleneck. One approach to handling this problem is building large server farms and dividing the game world into several independent cells, each being in one server's field of responsibility. This measure still faces three scalability issues: insufficient total resources, high user density and excessive inter-server communications [8].

For a more general solution, a new game architecture is necessary. In order to achieve scalability different approaches exist that are based on the peer-to-peer paradigm [8, 18]. We also propose a *distributed game architecture*: Every instance of an application is a peer in a peer-to-peer system. Each peer manages the part of the game world relevant to the players in it. Information only needs to be transmitted to other peers when a player or an entity is in that scope. Since playing a distributed game should not differ from playing a client-server game, the distributed game architecture should be capable of implementing all game features of current client-server games, provide network latencies of less than $300ms$, and offer security.

3. IRRLICHT AND SKYPE

Developing and implementing a complete game infrastructure that is able to fulfil all the aforementioned tasks would go beyond our resources. Hence we decided to use existing components in order to create a fully distributed game application and evaluate its potential.

3.1 Irrlicht

In order to create a realistic game environment, we decided to use the *Irrlicht* engine [9]. It is an open source and easy to use 3D engine written in C++. All state-of-the-art 3D engine features (e.g. texture animation, light maps, parallax mapping, dynamic shadows) are implemented. The API wraps up complex concepts of computer graphics and

presents the programmer with only a set of abstract classes with well-defined interfaces. Thanks to this, it is possible to quickly implement advanced 3D features into a game without requiring a deep understanding of the underlying mechanisms. In addition, there is a great *Irrlicht* community and comprehensive documentation is available.

Although we have only tested the engine under MS Windows, the developer site states that the library is also available for Mac OS and Linux.

3.2 Skype

Skype is a very popular voice-over-IP application, based on the peer-to-peer paradigm [14, 2]. Apart from the telephone directories, it has no central components. One of its nice properties is that it even works in the presence of firewalls and network address translation. Skype has an application programming interface that allows new applications to use the existing peer-to-peer infrastructure. We thus decided to use this API as the network interface for our distributed online game. This gives us the additional advantage that voice communication and text messages are available to our gamers for free.

The data transmission is being implemented using the concept of virtual applications and data streams. Unreliable and reliable data traffic, as well as group communication via collections of streams are available.

Last but not least Skype employs high security standards [2]. It is important to transmit data securely to avoid manipulation by cheaters. Data encryption, however, comes at the cost of higher network delays.

4. ARCHITECTURE AND IMPLEMENTATION OF THE DISTRIBUTED GAME

4.1 Game Idea

In order to test the game architecture, we chose a suitable game idea that is able to demonstrate the quality of the graphics engine and, in particular, the performance of the underlying Skype peer-to-peer network. We use a common game scenario known from one of the early multi-player games *Descent*. Each player controls a space ship in a 3D environment. The goal is to destroy the other space ships. This idea is simple but sufficient for testing the underlying network. The game requires fast reactions of the players and thus low latencies. There is no limit to the number of players.

4.2 Application Logic

There exists a number of concepts concerning the synchronisation of game states across networks that are commonly implemented in modern online games. In this paper we refer to these techniques as *application logic*.

A problem inherent in distributed games is inconsistency. Local changes to the state of a game through user interaction must be transmitted over the network to the other instances. Such a transmission is subject to a delay and delay jitter. This means that it is no longer guaranteed that the ordering of actions performed at two different sites will be retained after being sent to the other instances of the game. Confusing up the order in which events take place induces an inconsistency and necessitates consistency control mechanisms [11, 16].
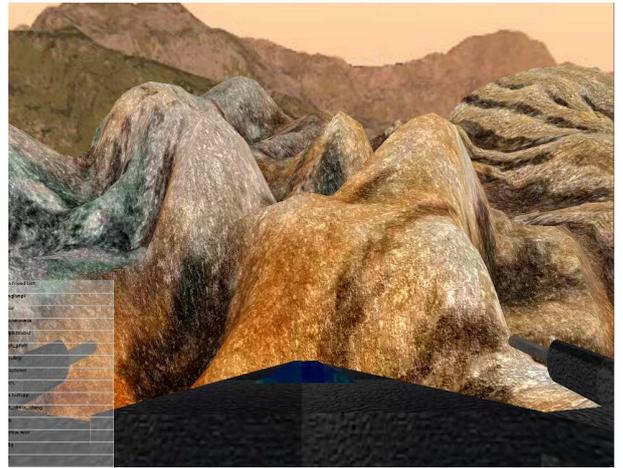
Figure 1: Screenshot of our multi-player game

Distributed applications often allow users to join and leave at any point in time. In the case of a late join, the new user misses all the previously sent data. It is then necessary to get the new user's game into a consistent state as fast as possible. Algorithms dealing with this situations are called late join algorithms [17].

For our sample application we restricted the implementation to clock synchronisation which is the basis for many consistency control mechanisms. To accomplish this, we use a simplified version of the network time protocol (NTP) in symmetric mode [12]. Data packets containing three timestamps are sent periodically to each other player in the current cell. The receiver includes the reception time as the fourth timestamp and uses all four values to compute a round trip delay and a clock offset relative to the sender. The eight most recent delays and offsets are then stored in a list. After collecting enough data, the clock offsets relative to all players in the current cell are combined into one final value that is used to adjust the local clock.

4.3 Skype4Scalability

Simply using the Skype API and point-to-point connections for data transmission would lead to a fully meshed network. In such a network, the number of connections grows at $O(n^2)$ with user size, compared to the growth at $O(n)$ in a client-server architecture. Therefore the set of players to communicate with must be reduced. In the best case, state updates would only be transmitted to receivers that need the information. Hu et al. [8] propose to use a *Voronoi-Based Peer-to-Peer Design* to maintain a list of receivers located in the player's AOI. Yu et al. [18] use a combination of neighbour list exchange, distributed hash table and direct transfer for interest management.

We follow a simpler but less optimal approach by dividing our game map into *cells*. Cells can be physically independent areas, like caves or buildings. It is also possible to let the cells overlap so that a player's avatar does not suddenly spawn in a new cell it enters.

In Figure 2 we show a quadratic world map divided into four cells A, B, C and D. Suppose that a player P_1 is initially located in cell A. Each player in this cell receives state updates from P_1 and vice versa. Communicating the
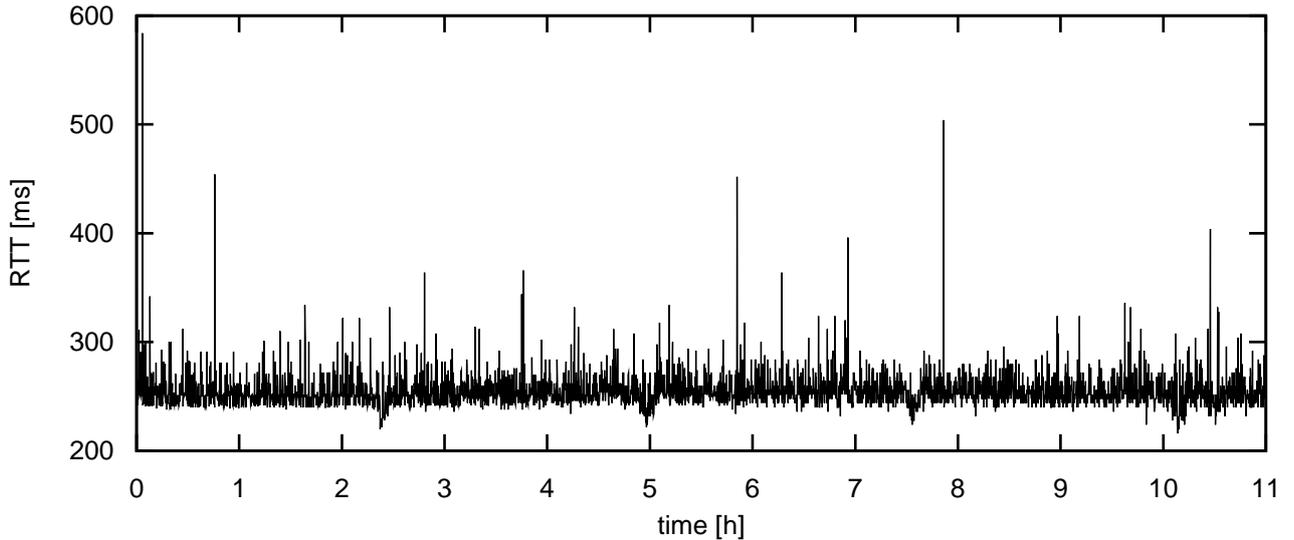
Figure 3: Measured round trip times between Mannheim (Germany) and Penticton (Canada) over the course of eleven hours
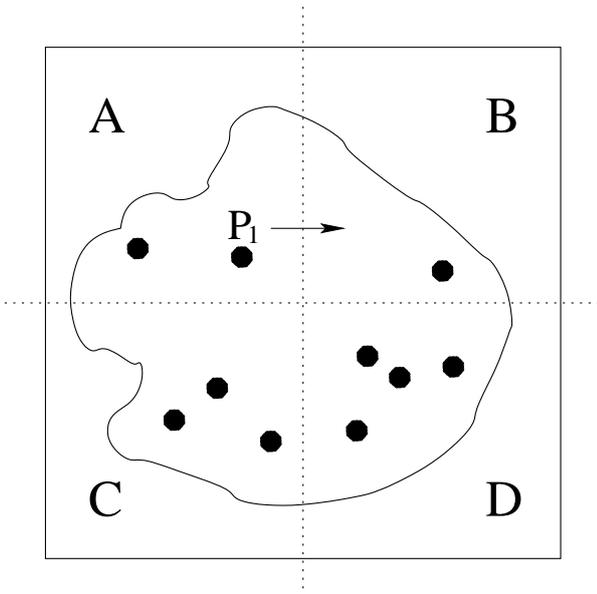

Figure 2: The game world is divided into four parts. Player P_1 is initially located in cell A and moves to cell B.

directly visible state information to players outside cell A is not necessary as they are out of sight anyway. When P_1 moves over to cell B, these constraints should now hold for the new cell and the players in it. Since P_1 did not communicate with any of the nodes in section B before, it might not even know of their existence. The problem of keeping a list of active players and their location up to date without using a central server is not trivial.

Fortunately, this is a well known issue in peer-to-peer applications, and the Skype software provides functionalities to handle similar situations. We found a way to utilise the Skype functionality. In version 7 of the API, *public chats* were introduced. A public chat can be created without any additional members. Its globally unique chat ID can then be passed to other users to allow them to join the conversation at a later time.

For our game application, this means that the first player to join a game creates a number of hidden *public chats* - one for each cell in the game. They are hidden in that they are not directly visible to the players, and they are not meant for text or voice communication. The IDs of these chats have to be stored in a table on the player's computer. When new players join a running game, they have to know at least one active player to receive all chat IDs from.

If P_1 in our example above now wants to move from cell A to cell B, the game application issues a Skype command to leave A's public chat channel and join B's instead. It is now possible to query the user handles of all players in B simply by querying the active members of its chat room. It is also imaginable to create and delete chat rooms on demand, when the first player enters a cell or the last player leaves respectively. The newly created and deleted IDs have to be broadcast to all players. We didn't include this in our implementation.

4.4 Implementation

We implemented our online game in C++. So far we have only compiled it for Windows using the compiler included with Visual C++ 8.0. We're completely relying on the Irrlicht API to create GUI windows and handle GUI events. Our entire network communication is handled by Skype. Since both are also available for Linux and Mac OS, porting our application to these other operating systems shouldn't present us with too much trouble.

5. PERFORMANCE EVALUATION

In order to judge the usability of Skype for our network

game, we conducted round trip time (RTT) measurements of data packets sent over the Skype network. We sent packets with a payload of 64 bytes over the same streams as we use for sending game data. The RTT is being measured once every ten seconds and over a total duration of eleven hours. For our tests we ran our game on four PCs, two of which were located in Mannheim (Germany), one in Sophia Antipolis (France), and one in Penticton (Canada). One of the two PCs in Mannheim served as a central point by sending out data packets to the other three machines and measuring their RTT. The graph obtained from the measurements between Germany and Canada is shown in Figure 3. The test with Canada yielded a mean RTT of $255ms$ with a standard deviation of $16.3ms$. For France, we obtained a mean RTT of $67ms$ with a standard deviation of $8.1ms$. Finally the test in Germany resulted in a mean of $31ms$ and a standard deviation of $1.9ms$.

[7] states that for FPS delays of up to $300ms$ are being accepted by most players. From this statement and our results we can conclude that our game in combination with the Skype network is able to achieve delays low enough to allow acceptable online gaming. Only the RTTs to Canada were close to our desired upper delay boundary, which doesn't leave any buffer for cases when the network is partially congested.

6. RELATED WORK

Creating scalable online games is a major research area. The work related to this paper can be divided into two categories: *Scalability and Latency*, where scalability exhibits the main issue.

Scalability

In order to obtain scalability different approaches have been evolved. Most of them are based on two strategies: utilising the resources of joining nodes and restricting the communication to the relevant nodes by using a AOI-concept. The different strategies can be categorised into two classes:

- adapting standard peer-to-peer algorithms (DHT) to the issues of online games [10, 5].

- using neighbour discovering algorithms e.g. a Voronoi-based Overlay Network [8].

Our approach is to outsource the problem by using the skype functionality.

Latency

Many popular online games are well-investigated in order to research latency issues (Half-Life [7], Quake 3 [1], Warcraft III [13], Unreal Tournament 2003 [3]). But all these measurements are based on a client-server architecture. The reason therefore is the absence of commercial peer-to-peer games and thus latency in peer-to-peer games is still a research topic.

7. CONCLUSIONS AND FUTURE WORK

We analysed current online games and derived the requirements for our distributed multi-player game. We then described the architecture and implementation of our game and measured the Skype network delays. In our approach,

we pushed the responsibility for distributed state maintenance towards Skype. Our performance measurements indicated that the architecture is sufficient for fast online games, even over long distances.

It can be concluded that our architecture is suitable for implementing a distributed interactive application that in principle scales well. Further work includes analysation of the game behaviour when played by a larger number of players simultaneously. It would be interesting to see whether our game achieves similar latencies in the presence of many participants. As a next step, we will research into the security aspects of a peer-to-peer game using Skype.

8. REFERENCES

[1] G. Armitage. An experimental estimation of latency sensitivity in multiplayer quake 3. In : *Proceedings of the 11th IEEE International Conference on Networks (ICON '03)*, pages 137–141, September 2003.

[2] S. Baset and H. Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. Technical Report CUCS-039-04, Computer Science Department, Columbia University, New York, NY, September 2004.

[3] 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 (NetGames '04)*, pages 144–151, Portland, Oregon, USA, 2004.

[4] Blizzard Entertainment. Press release. `http://www.blizzard.com/press/070111.shtml`.

[5] 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 (NetGames '06)*, number 44, New York, NY, USA, 2006. ACM Press.

[6] T. Fritsch, B. Voigt, and J. Schiller. Distribution of online hardcore player behavior (how hardcore are you?). In : *Proceedings of the 5th ACM SIGCOMM workshop on Network and system support for games(NetGames '06)*, number 16, New York, NY, USA, 2006. ACM Press.

[7] T. Henderson. Latency and user behaviour on a multiplayer game server. In : *Proceedings of the third International COST264 Workshop on Networked Group Communication (NGC '01)*, pages 1–13, London, UK, 2001. Springer-Verlag.

[8] S.-Y. Hu, J.-F. Chen, and T.-H. Chen. VON: a scalable peer-to-peer network for virtual environments. *Network, IEEE*, 20(4):22–31, 2006.

[9] Irrlicht. A free open source 3d engine. `http://irrlicht.sourceforge.net/`.

[10] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In : *Proceedings of the Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies (INFOCOM'04)*, volume 1, pages 96–107, March 2004.

[11] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg. Local-lag and Timewarp: Providing Consistency for Replicated Continuous Applications. *IEEE Transactions on Multimedia*, pages 47–57, February 2004.

[12] D. Mills. Internet time synchronization: The network time protocol. In : *Proceedings of the IEEE Transactions on Communications (1991)*, volume 39, pages 1482–1493, 1991.

[13] 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 (NetGames '03)*, pages 3–14, New York, NY, USA, 2003. ACM Press.

[14] Skype. A peer-to-peer internet telephony software. Webpage: `www.skype.com`.

[15] J. Smed, T. Kaukoranta, and H. Hakonen. Aspects of networking in multiplayer computer games. In : *Proceedings of the International Conference on Application and Development of Computer Games in the 21st Century*, pages 74–81, Hong Kong SAR, China, November 2001.

[16] J. Vogel and M. Mauve. Consistency Control for Distributed Interactive Media. In : Proceedings of the *9th ACM Multimedia, ACM MM 2001*, pages 221 – 230, Ottawa, Canada, September 2001.

[17] J. Vogel, M. Mauve, V. Hilt, and W. Effelsberg. Late Join Algorithms for Distributed Interactive Applications. *ACM/Springer Multimedia Systems*, pages 327–336, October 2003.

[18] A. P. Yu and S. T. Vuong. Mopar: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games. In : *Proceedings of the international workshop on Network and operating systems support for digital audio and video (NOSSDAV '05)*, pages 99–104, 2005.