1-1-2010

A Location Aware P2P Voice Communication Protocol for Networked Virtual Environments

Gabor Papp
University of Denver

A Location Aware P2P Voice Communication Protocol for Networked Virtual Environments

---

A Dissertation
Presented to
the Faculty of Engineering and Computer Science
University of Denver

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

---

by
Gabor Papp
August 2010
Advisor: Dr. Christopher GauthierDickey

Author: Gabor Papp
Title: A Location Aware P2P Voice Communication Protocol
for Networked Virtual Environments
Advisor: Dr. Christopher GauthierDickey
Degree Date: August 2010

Abstract

Multiparty voice communication, where multiple people can communicate in a group, is an important component of networked virtual environments (NVEs), especially in many types of online games. While most research has been conducted on one-to-one communication, we focus on group communication.

In this dissertation, we present the first measurement study on the characteristics of multiparty voice communications and develop a model of the talking and silence periods observed during multiparty communication. Over a total of 5 months, we measured over 11,000 sessions on an active multi-party voice communication server to quantify the characteristics of communication generated by game players, including group sizes, packet distributions, user and session frequencies, and speaking (and silence) durations. Further, we develop a model for multiparty voice communication that can be used for future research, simulation, network engineering, and game development work.

Next, we propose a peer-to-peer protocol that uses Gabriel graphs, a subgraph of Delaunay-triangulations, to provide scalable multiparty voice communication. In addition, our protocol uses positional information so that voice data can be accurately modeled to listeners to increase the immersiveness of their experience. Our simulations show that the algorithms scale well even in densely populated areas, while prioritizing the sending of voice packets to the closest listeners of a speaker first, thus behaving as users expect. We also develop a security framework that prevents common

attacks. Finally, we implement our protocol and put it through exhaustive validation, where we use the model that we generated using our multiparty voice communication model.

Acknowledgements

The writing of a dissertation can be a lonely and isolating experience, yet it is obviously not possible without the personal and practical support of numerous people. I would like to thank my advisor, Dr. Christopher GauthierDickey, for his constant encouragement, tremendous help and valuable advice.

I am grateful to my committee, Dr. Matthew Rutherford and Dr. Ramakrishna Thurimella, for their guidance during the course of my research. I also wish to thank the chair of my committee, Dr. Alvaro Arias for his availability despite the last minute notice.

My sincere gratitude goes to my parents, Katalin and Marton, and my sister, Szilvia, for their love, support and patience over the last four years.

Finally, I would like to thank my girlfriend, Samantha, for all her support, help and sacrifice. There is no way I could have completed this dissertation had it not been for you.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this dissertation we propose a virtual-location-aware, peer-to-peer voice communication protocol that works well for virtual realities such as massive multiplayer online games. Research to date has only addressed the requirements of this problem separately. In addition, to understand the exact requirements of such a protocol, we conduct a measurement study.

Our work is the first to look at characterizing multiparty voice communication over the Internet, particularly when it is used with multiplayer games. Previous VoIP measurement work has looked at quality of service parameters such as packet loss, packet reordering and its effects on sound quality, or at network characteristics and support of VoIP between two parties. Instead, our research attempts to characterize the traffic, packet arrival rates, group sizes, session frequencies and durations, and speaking and silence periods in order to develop mathematical models for multiparty communication that can be used for simulation and modeling.

Next, we develop a virtual-location-aware, peer-to-peer protocol that satisfies all of the requirements of multiplayer voice communication. Our solution uses a novel mechanism for building and maintaining the connection between players. We also

propose techniques to secure our protocol and to make it cheat-proof. This concept allows our protocol to serve as a solution for voice communication between users of a virtual reality.

1.1 Motivation

Since the advent of the Internet, people have sought to interact over computer networks. In 1995 VocalTec released the first commercial Internet phone software. In 1996 ITU-T began the development of standards for the transmission and signaling of voice communications over Internet Protocol (VoIP) networks with the H.323 standard, but it was only in 1999 when the Session Initiation Protocol - the first community maintained VoIP protocol - appeared. The distributed nature of the Internet allowed several features to be incorporated into VoIP, such as video and conference calls.

During their first appearance, conference calls were limited to small groups because user bandwidth was limited to modem speeds. However, with the increasing penetration of broadband Internet into homes, VoIP allows a massive group of users to be connected and to interact with each other concurrently.

Multiparty voice communication (MVC) is an important application that needs to be studied and researched. In multiplayer computer games, for example, thousands of players can interact and communicate with each other using built-in voice chat systems[1]. Game consoles have also added voice communication support for player interaction as selling features of the hardware. Further, multiparty voice communication is widely used for conference calls and voice chat software; MVC will clearly be part of future online collaborative systems. Thus, research in this area benefits the

[1] `http://www.worldofwarcraft.com`

future design of MVC systems, is relevant and interesting to online virtual realities, and is important to ISP network engineers supporting and hosting voice systems.

Telecommunication has a significant economic impact on modern society. In 2008, estimates placed the telecommunication industry's revenue at $3.85 trillion USD. Since charging and accounting are key elements of a communication systems, they are traditionally designed with a client/server network architecture. This has the advantage that a single authority keeps track of the login and logout times and builds up the connections between the callers and the callees. On the other hand, this architecture has several disadvantages:

1. The computational complexity requires powerful servers.

2. The server is a single point of failure. To avoid downtimes, expensive and complex clusters are required.

3. Having a server introduces an additional step and therefore delay between the users.

To address the limitations of the client/server architecture, researchers have turned to peer-to-peer architectures for IP applications. Peer-to-peer is a distributed solution composed of participants that make a portion of their resources directly available to other network participants without the need for central coordination instances. In these, network peers can communicate directly, which reduces the delay for messages and eliminates congestion.

Perhaps the most well-known peer-to-peer VoIP software is Skype[2], which gained its popularity from being available free of charge. During the peak period it has more than twenty million users logged in concurrently. Although it is free and widely used for long-distance, video and conference calls, it is not popular among game players.

[2] `http://www.skype.com`

TeamSpeak[3] and Ventrilo[4] are the most common choice of multiparty voice communication software for online game players. Although these are client/server solutions, they serve the needs of gamers the best as they have the freedom to join different channels, which mostly represent the teams inside the virtual world. This clearly demonstrates that there is a need for multiplayer voice communication protocols that fit well within virtual realities. The ideal solution would set up the connection between the users that are close to each other inside the virtual space, which would prevent them from having to join the appropriate channel all the time.

However, the design of a peer-to-peer VoIP protocol that takes the virtual locations into account faces a number of challenges. Any location-aware protocol has to maintain consistency between the virtual locations and the communication channels between the users. Since a virtual world is a dynamic environment, the changes have to be reflected with minimal delay in order to make the users satisfied and actively engaged. The protocol also has to maintain a low delay for packet transfers. The human ear is very sensitive, a delay of less than 150ms is typically required to maintain a natural conversation, and a delay of less than 300ms is required to maintain a tolerable conversation [22]. A delay of 450ms or more is simply not acceptable.

An additional challenge is the fact that voice communication can contain sensitive data. On one hand, it has to be secured from people outside of the virtual world. On the other hand, the data also has to be secured from people inside of the virtual world who do not have the credentials to access it. In addition to protecting sensitive data, security can also help prevent cheating.

[3] `http://www.teamspeak.com`
[4] `http://www.ventrilo.com`

1.2 Problem Definition and Research Challenges

1.2.1 Voice over IP

Voice over IP (VoIP) is an IP telephony term for a set of facilities used to manage the delivery of voice information over the Internet. VoIP involves sending voice information in digital form in discrete packets rather than using the traditional circuit-committed protocols of the public switched telephone network (PSTN).

A typical VoIP system includes the following components and technologies:

- *Database:* to locate endpoints in the network.
- *Signaling:* to coordinate the actions of the various networking components.
- *Call connect and disconnect:* mechanism to transport audio content.
- *Coder-decoder:* operations to convert analog waveforms to digital information for transport.

In a scenario where a user initiates a call to another user, first the user's endpoint queries the database using signaling regarding the location of the callee. After receiving the location information, the endpoint builds a connection between itself and the callee's endpoint. Using the channel, the voice communication takes place. During this step the analog voice communication is coded at the caller's endpoint, and decoded at the callee's endpoint with the codec. When the conversation is over, the endpoints disconnect (See Figure 1.1).

In a dynamic environment, this process would not be sufficient. First, the database has to be regularly updated to reflect even the most recent changes of the location of the users in the virtual world. Second, in order to achieve the lowest possible delay,

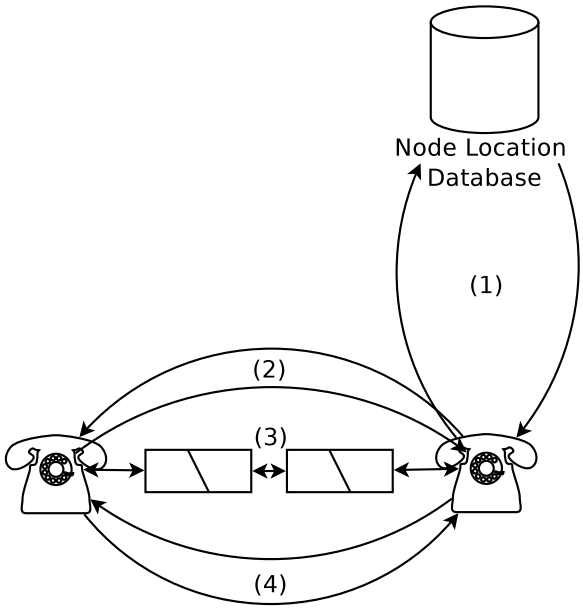


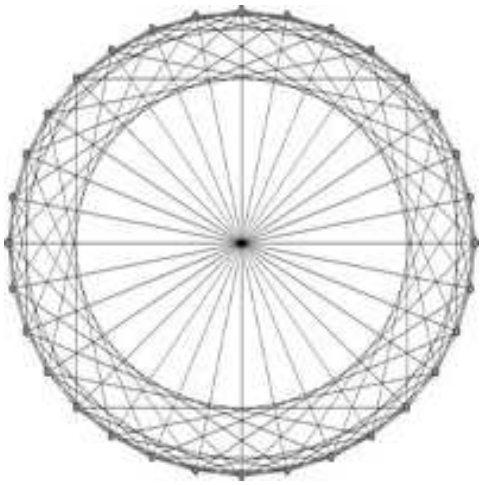
Figure 1.1: *A Basic VoIP Call Architecture:* The basic VoIP call architecture consists of four steps: (1) node location lookup, (2) connection setup, (3) voice transfer and (4) connection termination.

the connection building step has to be eliminated. The simplest way to limit delay is to maintain a connection between all of the users who might talk to each other.

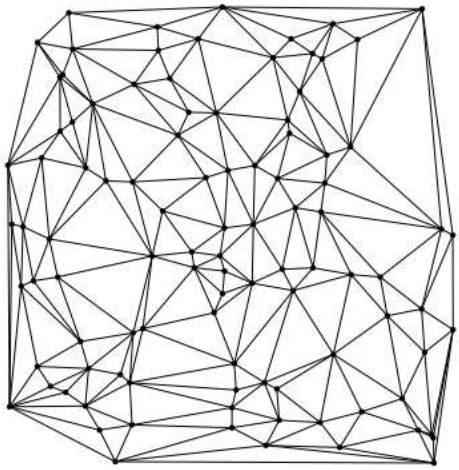
1.2.2 Peer-to-Peer Networks

Peer-to-peer architectures have been used excessively for file sharing for over a decade, but they are not very popular for delay-sensitive, dynamic real-time applications. Structured peer-to-peer architectures are well suited for finding a piece of data in a distributed network since they provide a reliable lookup mechanism. Although this mechanism is also efficient, considering the size of such networks, it is still not fast enough for voice communication.

Unstructured solutions are better suited for such a purpose. They are easily mappable to different layouts and are more suitable for range queries, but they do not provide an efficient lookup mechanism. Figure 1.2 shows the two architectures.



(a) Chord: A Structured P2P Network



(b) Delaunay-Triangulation: An Unstructured Peer-to-Peer Network

Figure 1.2: *Peer-to-Peer Network Architectures:* Structured P2P networks provide a reliable lookup mechanism, whereas unstructured P2P networks are better suited for range queries.

Thus, combining a multiparty voice communication protocol with a peer-to-peer architecture raises several challenges. Node joining and leaving, as well as neighbor maintenance have to be efficient. In addition, the peer-to-peer network has to reflect the connections between the users of the virtual world in order to guarantee low delay.

1.2.3 Virtual Realities

Virtual reality (VR) is a term that applies to computer-simulated environments that can simulate places in the real world as well as in imaginary worlds. Users can interact with a virtual environment through the use of either a standard input device such as a keyboard or mouse, or through a multimodal device such as a wired glove. The simulated environment can be similar to the real world, for example, in simulations for pilot or combat training, or it can differ significantly from reality, as in VR games. Either way, our goal is to enhance user experience in these worlds by offering a virtual-location-aware voice communication solution.

Designing such a protocol raises many challenges. First, the voice sources have to be prioritized. Closer people have to be heard more clearly than people in the background. Second, since people move inside the virtual world, the position of the users has to be updated frequently enough to provide a realistic scenario for the listeners. Third, the protocol has to deliver voice packets only to people that are eligible to receive them. This applies to attackers both inside and outside of the network. While eliminating attackers from the outside requires similar techniques to other VoIP solutions, identifying attackers from the inside is much more complex. Manipulating the virtual location of the players can lead to an unfair advantage and therefore might be used for cheating.

1.3 Proposed Approach and Organization of Dissertation

Our protocol is the first to propose a peer-to-peer, virtual-location-aware voice communication solution for virtual realities. This complex protocol requires the precise identification of the requirements for such systems. Furthermore, it requires the thorough design of the protocol and its security framework. Finally, it has to be validated to make sure it fulfills all the identified requirements. Our approach to the problem consists of three major parts:

1. Requirement identification.

2. Protocol design.

3. Protocol validation.

In Chapter 2 we give the reader the background that is necessary to understand the solved and unresolved challenges that we faced throughout our work. In Chapter 3 we present the measurement study that we conducted over several months to collect

as much data about multiparty voice communication as possible. We also present a way to model such conversations, which we later use in Chapter 5. The protocol design is explained in Chapter 4 along with our security framework. This chapter focuses on the mechanisms of the protocol and also contains simulations and their outcomes which served as an early verification of our solution. Next, we implemented our protocol and put it through exhaustive validation. For this, we used data that we generated using our multiparty voice communication model. We present the results of our validation in Chapter 5. Conclusions are presented in Chapter 6.

Chapter 2

Research Foundation

2.1 Introduction

Our research is related to three areas: voice communication, computer networking and virtual realities. Therefore, a fundamental knowledge in these areas is required to understand our protocol and the choices we have made with regard to our designs. Voice communication is the main topic around which our work is built. In Section 2.2 we discuss some of the existing and most popular voice protocols as well as statistical analysis of voice communications.

Key areas that are closely related to our research within computer networking are peer-to-peer networks and neighbor maintenance. In Section 2.3 we discuss the difference between client-server and peer-to-peer networks. We also compare and contrast structured and unstructured peer-to-peer networks. Furthermore, we analyze different neighbor maintenance solutions and security issues related to these networks.

Since our protocol can be used to communicate in virtual realities and computer games, we briefly discuss modern research related to them, including cheating, in Section 2.4. We focus on the cheating techniques that can take advantage of voice

packet manipulation between players.

2.2 Voice Communication

Over the last 40 years, a vast amount of research has been conducted on voice communication between two parties. The research investigates either the talking patterns between the caller and the callee, or proposes new techniques to establish a connection and transfer the voice packets.

2.2.1 Measurement Studies

Voice patterns consist of *on* and *off* periods (also called talkspurts and silence periods). Research has shown that these *on* periods follow an exponential distribution [6, 17, 41] in traditional telephony. These results are important because they allow designers of hardware, codecs, and network administrators to predict the patterns of speech with mathematical models.

Jiang et al. researched the on-off patterns in VoIP by recording and digitizing conversations and then applying *gap* detectors to determine how long people talked and how long they were silent [21]. Their results show that the length of time that people talk follows an exponential distribution while the gaps between talkspurts deviate significantly from the same distribution.

Markopoulou et al. measured the quality of voice communications over the Internet. They measured delay and loss over wide-area backbone networks and used these results with a voice quality model [27] to determine the efficacy of VoIP over the Internet for voice communication. The authors show that while many Internet backbones have sufficiently low delay, delay variability, and loss, several of them provide poor VoIP quality.

Boutremans et al. examined the impact of network conditions, such as link failure, on the quality of voice communications with VoIP [5]. They show that quality of service at the network level is not needed for VoIP, although link failures cause significant problems for voice communication.

Skype[1] [1], a VoIP application, was measured by Chen et al. in order to determine the level of user satisfaction [9]. By measuring network traffic characteristics, they correlated the amount of jitter and interactivity of a session with the length of the call.

We are more interested in the characteristics of the traffic with multiple parties and less interested in whether voice can be used on the Internet[2]. We set up a TeamSpeak[3] server and measured the traffic directly. Our results differ from previous two-party measurements and show that the on-off patterns of VoIP in multiparty communications follow Weibull distributions more accurately, without significant deviation.

2.2.2 Voice Communication Protocols

In any kind of multimedia streaming the goal is to transfer the data in sequence with minimal jitter. However, minimizing the delay is not always important (e.g. for on-line video streaming, a fairly large delay is tolerable). The same cannot be said about real-time applications such as voice communication. In this section we review solutions that have been proposed in the past to fulfill this special requirement of voice communication.

H.323 is a recommendation from the ITU Telecommunication Standardization Sector that defines the protocols to provide audio-visual communication sessions on any

[1] `http://www.skype.com`

[2] The widespread adoption by gamers and the success of Skype seem to indicate that voice communication over the Internet is both possible and acceptable.

[3] `http://www.teamspeak.com`

packet network. The H.323 standard addresses call signaling and control, multimedia transport and control, and bandwidth control for point-to-point and multi-point conferences.

Yeo et al. proposed a H.323 compilant architecture for supporting voice over IP in a heterogeneous environment [48]. Their system provides IP telephony services to support the two main groups of telecommunication users, namely, the conventional PSTN phone users as well as the Personal Computer users.

The Session Initiation Protocol[4] (SIP) is an IETF-defined signaling protocol that is widely used for controlling multimedia communication sessions such as voice and video calls over the Internet Protocol. The protocol can be used for creating, modifying and terminating two-party or multiparty sessions consisting of one or several media streams. The modification can involve changing addresses or ports, inviting more participants, and adding or deleting media streams. Other feasible application examples include video conferencing, streaming multimedia distribution, instant messaging, presence information, file transfer and online games.

Singh and Schulzrinne proposed a peer-to-peer solution for the SIP protocol [40]. They suggested a two tier hierarchical model, in which the clients simply connect to the supernodes and only the supernodes form a Chord ring. Their solution is more fault tolerant, robust and scalable than the original solution because it does not contain any centralized entity. However, this solution also introduces a higher delay and is not capable of implementing accounting.

Bousteaud et al. performed a comparison of delivery architectures for immersive audio in crowded networked games [4]. In their survey they compare and contrast the client-server, the peer-to-peer and the hybrid methods. Although their conclusion is not straightforward, they find the overall performance of the hybrid method the best

[4] `http://tools.ietf.org/html/rfc3261`

but they also claim that it is the most complicated one.

Bousteaud et al. also developed DICE, a voice communication system that is suitable of handling crowded virtual spaces such as battlefields in on-line games or market places [38]. It is based on a client/server architecture where the clients are not connected directly but via the servers. The protocol is location sensitive and bandwidth efficient. The proposed solution is about 50% more efficient than when the clients are simply connected directly to each other.

PROMISE [18] is a peer-to-peer media streaming system encompassing the key functions of peer lookup, peer-based aggregated streaming, and dynamic adaptations to network and peer conditions by Hefeeda et al. The researchers introduce the CollectCast service, which operates entirely at the application level but infers and exploits properties of the underlying network. CollectCast has a pattern of "one receiver collecting data from multiple senders". The protocol takes into account the network topology to optimize the route of the packets.

Our protocol is similar to PROMISE in that it is a peer-to-peer media streaming solution, in which any node has the ability to receive voice packets from multiple nodes in the system at the same time. Furthermore, it takes the network topology into account. The main difference between our technique and that of Hefeeda is that we focus on the virtual rather then the actual topology of the system. Our aim is not to optimize the network traffic but to provide the most realistic scenario to the players. Therefore, connections between players in our network are not determined based on optimization or personal request, but the virtual location. This results in players who are close in the virtual world having direct connections with each other instead of having direct connections between players who are close in the physical world, but are far in the virtual world.

2.3 Computer Networking

A computer network is a collection of computers and other computational devices connected by communication channels. These networks may be classified according to a wide variety of characteristics. We distinguish client/server and peer-to-peer networks based upon the functional relationship among the elements of the network. A peer-to-peer network is any distributed architecture composed of participants that make a portion of their resources (such as processing power, disk storage or network bandwidth) directly available to other network participants, without the need for central coordination instances (such as servers or stable hosts). Peers are both suppliers and consumers of resources, in contrast to the traditional client-server model where only servers supply, and clients consume. With further classification of peer-to-peer networks, we can distinguish unstructured and structured solutions.

2.3.1 Structured Peer-to-Peer Networks

Structured peer-to-peer networks employ a globally consistent protocol to ensure that any node can efficiently route a search to some peer. Such a guarantee necessitates a more structured pattern of overlay links. By far the most common type of structured peer-to-peer networks is the distributed hash table (DHT) in which a variant of consistent hashing is used to assign ownership of each file to a particular peer, in a way analogous to a traditional hash table's assignment of each key to a particular array slot.

Stoica et al. proposed Chord, a scalable peer-to-peer lookup service [44]. Chord provides support for just one operation: given a key, it maps the key onto a node. Depending upon the application using Chord, that node might be responsible for storing a value associated with the key. Chord uses a variant of consistent hashing

in order to ensure that the load is balanced in the system since each node receives roughly the same number of keys. It is highly probable that when an N^{th} node joins/leaves the network, only an $O(1/n)$ fraction of the keys are moved to a different location.

A similar solution is presented by Rowstron et al. in the design of Pastry [37], a scalable architecture for object location and routing of messages in wide-area peer-to-peer applications. Pastry maps keys to nodes with IDs numerically close to the keys. Every node is assigned a numeric ID, and every value is assigned a numeric key. Unlike similar systems, Pastry maps each key to k nodes numerically nearest to the key itself instead of mapping to just one node, which provides more availability and better accessibility. Since the node IDs are randomly assigned, the k nodes will be geographically diverse, which leaves room to explore locality. When Pastry routes messages to nodes it takes into account the locality of the nodes. Messages reach nearest local nodes before they can be forwarded to far nodes.

Ratnasamy et al. proposed a conceptually different solution for the same problem [35], which uses a d-dimensional Cartesian coordinate space on a d-torus to map the keys and nodeIds. The space is dynamically partitioned among all the nodes in the system. To store a (key, value) pair, the key is deterministically mapped onto a point in this coordinate space using a uniform hash function. The corresponding pair is then stored at the node that owns this zone.

While structured peer-to-peer networks provide an efficient lookup, they do not reflect network topology. Furthermore, they are not efficient for range-based queries: therefore it is hard to apply them for virtual-location-aware voice communication.

2.3.2 Unstructured Peer-to-Peer Networks

Unstructured peer-to-peer networks are formed when the overlay links are established arbitrarily. Such networks can be easily constructed: a new peer that wants to join the network can copy existing links of another node and then form its own links over time. If a peer wants to find a node in an unstructured peer-to-peer network, the query has to be flooded through the network. The main disadvantage with such networks is that flooding causes a high amount of signaling traffic. On the other hand, this kind of network is well-suited for range-queries. A special kind of unstructured peer-to-peer networks is one that supports greedy-routing. Incorporating this technique helps to reduce the number of queried nodes and thus suits virtual-location-aware applications. Figure 1.2(b) shows an example of an unstructured peer-to-peer network.

Neighbor maintenance

For neighbor maintenance, research has recently centered around Delaunay triangulation to reduce the cost of peer-to-peer communication. For example, Steiner and Biersack developed a P2P network using a 3D Delaunay triangulation where edges are flipped to form the triangulation and sets of nodes are locked on the joining and leaving of nodes to maintain consistency [43].

Ohnishi et al. created a P2P network using the 2D coordinates of players in an NVE and an incremental Delaunay-triangulation [31]. Voronoi diagrams, the dual of Delaunay triangulations, are used to designate areas for control by super-nodes.

Varvello et al. used dynamic clustering in a Delaunay-based P2P network in which peers monitor the maintenance traffic for the network and spawn clusters when the traffic exceeds a given threshold [46]. The clusters help reduce maintenance traffic

within the network, which makes Delaunay-triangulation more scalable.

Shun-Yun et al. proposed the Voronoi-based Overlay Network (VON), an efficient design that maintains the peer-to-peer topology in a fully-distributed, low-latency, and message-efficient manner [20].

Ghaffari, Hariri and Shirmohammadi proposed a new distributed architecture for update message exchange in massively multi-user virtual environments [15]. The protocol is based on the Delaunay-triangulation architecture and supports churn and user mobility.

While unstructured peer-to-peer networks provide the kind of neighbor maintenance we need for our protocol, they mostly focus on the Delaunay-triangulation. The Delaunay-triangulation on average has a node degree of six. For voice communication this is too high, since the human ear cannot distinguish this many voice streams. Furthermore, Delaunay-triangulation does not model reality for voice spreading. Our research focuses on these issues and our Gabriel graph based voice communication protocol addresses them. However, we have to note that we use the Delaunay-triangulation excessively for neighbor maintenance: therefore it is an important part of our work.

Security Issues

Peer-to-peer networks have many advantages over the traditional client-server architecture, such as the lack of a single point of failure and better scalability. However, since they are distributed systems, additional security issues arise.

Ten years ago, Lundberg's research showed that ad-hoc protocols did not accommodate any security and were highly vulnerable to attacks [26]. Interestingly, as the similarities between ad-hoc and peer-to-peer networks are fairly high, the same problems can be identified here as well.

Castro et al. presented a secure routing protocol for structured peer-to-peer net-

works [7]. These networks are highly resilient; they can route messages correctly even when a large fraction of the nodes crash or the network partitions. Thus, with the appropriate security they are suitable for large-scale decentralized applications such as storage, content distribution and even group communication. Their research studies attacks aimed at preventing correct message delivery and presents defenses to these attacks. Three main techniques are used: secure *nodeid* assignment, secure routing table maintenance and secure message forwarding. The authors focus only on structured overlays, therefore some of their solutions are specific for only this type of network. However, the three identified areas that have to be secured apply to other kinds of peer-to-peer, such as ours.

Fessi et al. researched an interesting combination of real-time media and peer-to-peer systems [14]. They claim that VoIP and distributed systems have both become very popular, and have received a large amount of attention from the research community in recent years. As a result, a peer-to-peer-based SIP was developed which solves the scalability issues and is able to provide a communication service independently of other network components such as DNS. However, with this solution new security issues arise, such as the Sybil attack, or Spam over IP telephony. Therefore, the paper presents a hybrid solution for telecommunication networks that fills the gap between centralized and purely peer-to-peer networks.

Srivatsa and Liu focus on a higher level of peer-to-peer protocols. Their goal is to secure publish-subscribe overlays against confidentiality, integrity, authentication and DoS attacks. Their solution, EventGuard [42], is implemented using three building blocks: tokens, keys, and signatures. A token is associated to every topic so that it is easier to handle the topic names. Also, every topic has a key to achieve confidentiality and integrity. The third building block is the signature, which is used for message authentication. Six main events are identified in the paper: subscribe, advertise,

publish, unsubscribe, unadvertise and routing. A guard is implemented for each of these in order to protect the system. This system is well-suited for non-delay-sensitive applications, but because of the use of signatures it is not compatible with interactive traffic.

Mittal and Borisov investigate the information leaks in structured peer-to-peer anonymous communication systems [29]. They claim that defending the system against the most commonly feared attack, i.e. targeting the lookup mechanism, results in higher security against active attacks, but lowers the security against the passive attacks. Therefore, there is always a tradeoff between robustness and passive attacks.

2.4 Virtual Realities

In addition to studying research in prior voice communications, it is also important for us to understand the unique features and properties of virtual realities and games.

2.4.1 Game Traffic

Understanding game traffic is important for our measurement study because it helps us understand the context of the source of our voice communication traffic. Borella analyzed game traffic from a popular online game server on a LAN and modeled the inter-packet arrival time and packet sizes with extreme distributions [2]. Their model was validated using the λ^2 test, which we also use to validate our models.

Henderson and Bhatti modeled network traffic of an online game over the Internet [19]. This work is important because the traffic was measured over the Internet instead of over a LAN, providing a more realistic model. Their work shows a daily and weekly traffic pattern similar to what we measured with voice traffic: evenings

have peak traffic while early mornings have the lowest traffic.

Pittman et al. and Svoboda et al. had similar diurnal patterns in their measurement work on large-scale multiplayer games [34, 45]. In addition, other researchers modeled game traffic (packet sizes, arrival times, sessions) with similar results [11, 47].

2.4.2 Cheating

Cheating is the act of circumventing rules in order to gain an unfair advantage, and as such, game players are often willing to cheat in order to win in some way. Our protocol is an application layer protocol, that is built on top of another network layer protocol, such as the Delaunay-triangulation. We assume that the base protocol is capable of handling the network layer cheats and we only have to focus on application and game level cheats.

Chapter 3

Measurement Study

3.1 Introduction

This work is the first scientific measurement study to be conducted on multiparty voice communications for games. *The main contribution of this chapter is a characterization of traffic patterns, group patterns, and voice patterns through measurements. In particular, we model the talking patterns mathematically, based on the measured multiparty voice communication sessions.* Further, the characterization of voice patterns has typically been done on small sets of data; our study measures patterns from thousands of hours of voice data with thousands of unique sessions. Thus, future research in MVC systems will be able to use our models to drive experimental simulations, game developers can use these models to understand the impact of adding voice communications on network traffic already generated by their games, and ISPs can use this information for provisioning servers for hosting MVC systems. We personally use our results to build a peer-to-peer voice communication protocol that suits virtual realities, with taking the virtual location of the players in account and thus providing a more realistic scenario.

To conduct our measurement study, we set up a TeamSpeak server which allows clients to join the server, set up channels, and communicate with other clients on the same channel. TeamSpeak[1] is well known in the gaming community and widely used due to the fact that it can be used for free for non-profit uses (i.e., typical game playing usage or measurement studies). We then recorded traffic over a three month period and analyzed the resulting data.

After analyzing the data and creating an initial model, we set up the server several months later and began collecting data a second time for a two month period. Using the newly collected data, we compared our model from the first data set to understand how the system had changed. This second set of data was used to validate our model and we detail in this chapter the similarities and differences between these two distinct measurement periods.

3.2 Trace Collection

In this section, we describe the architecture of our network, the content of the server log file, the fundamentals of the Speex voice codec[2], the packet format of the TeamSpeak protocol, the collection of the VoIP sessions and the procedure that we used to clean the collected data.

3.2.1 TeamSpeak Communication Architecture

TeamSpeak is a group voice communication server that allows multiple people to connect using a TeamSpeak client, join *channels*, and talk simultaneously with other people in the same channel. In this client/server architecture, the clients encapsulate voice packets using one of many codecs, and send those packets to the server using

[1] `http://www.teamspeak.com`
[2] `http://www.speex.org`

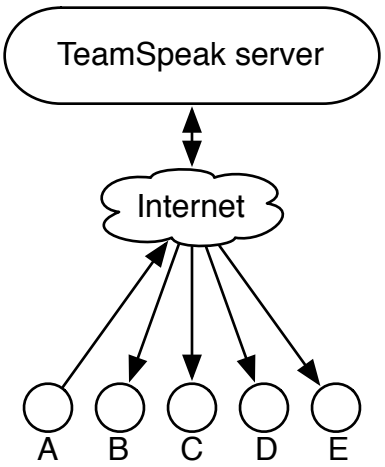


Figure 3.1: *TeamSpeak Architecture:* The TeamSpeak architecture is a client/server architecture which unicasts voice packets to all people in the same *channel*. In this figure, client A sends a voice packet to the server, which in turn replicates and unicasts the packet to B-E.

unicast. The server then unicasts the packets to the other $n-1$ clients connected on the same channel. Note that the server does *not* multiplex the voice packets. Figure 3.1 illustrates this concept.

To measure multiparty voice communications, we set up a TeamSpeak server and advertised it to game players as a free server. The server was set up and advertised starting in November of 2006. We have continued to operate the server collecting data for the long term study of multiparty voice communications.

Once the TeamSpeak server was set up, we began logging all traffic on port 8767 to the server using *tcpdump*. Although TeamSpeak generates a server log file, the data contained in this file (even with maximum verbosity) is minimal and contains data only about logins, logouts, channel switching, and administrative operations. Thus, we needed to use *tcpdump* to record all packet information generated with regards to the TeamSpeak server. We also discovered that when we compared data from the server log and the trace files, the server log was not always accurate. For example,

the server log would often show a player logging in multiple times without logging out. This was probably due to the fact that the player's connection died, but the server had not discovered it before the player re-logged in. On the other hand, from our packet traces we could determine a session length by looking at the time that a player logged into the server to the last time they sent or received a voice packet from any player. Table 3.1 lists the configuration parameters of the TeamSpeak server.

Table 3.1: *Configuration Parameters for TeamSpeak*

Parameter	**Value**
server	`hermes.cs.du.edu`
port	8767
protocol	UDP
codec	12.3 and 16.3 kbps Speex

3.2.2 The Speex Codec

Speex is an Open Source/Free Software patent-free audio compression format designed for speech and is available under the revised BSD licence[3]. It is based on the Code Excited Linear Prediction (CELP) algorithm and is designed to compress voice at bitrates ranging from 2 – 44kbps. However, our TeamSpeak server was set up to only accept 12.3kbps and 16.3kbps encodings on the channels, depending on the voice quality end-users desired.

Regardless of the bit-rate, the Speex codec uses a 20ms frame for encoding audio. This means that the frequency of the arrival of packets is not affected by the bit-rate since one frame is used per 20ms of audio time while making it easy to identify voice packets in our log files since they must be some multiple of the bit-rate, frame size and overhead.

[3] `http://www.xiph.org/licenses/bsd/speex`

3.2.3 The TeamSpeak Voice Packet Format

TeamSpeak only uses UDP packets to communicate between the client and server, including logging in, creating channels, switching channels, and talking to other people. As we needed only the voice packets for analyzing the characteristics of the behavior of the users, in this section we are focusing only on the format of those packets.

Each of the packets contains the regular IP and UDP headers. These are the first 28 bytes of the packets and these are necessary overheads. We discovered that most of the logged packets are 183, 189, 233 and 239 bytes therefore the actual payloads are 155, 161, 205 and 211 bytes. The average inter arrival time of these packets is 100 ms. The bandwidth can be calculated using the following formula:

$$BandWidth = \frac{Payload(bits)}{Time(sec)}.$$

In our case this results in:

$$BW = \frac{[155, 161, 205, 211] \times 8}{0.1} = 12.4, 12.88, 16.4, 16.88 kbps,$$

which roughly match with the Speex codec bitrate specification. Because of this, we claimed that these are the voice packets.

The packets with 155 and 205 bytes payload were all incoming packets to the server. The other two sizes were the sizes of outgoing packets. Further analysis of the data helped us to identify the codec ID in the packets as well as the sender ID and a sequence number. The main difference between the incoming and outgoing packets is that the incoming packets contain only the sender ID and sequence number whereas the outgoing packets contain both the sender ID and sequence number and

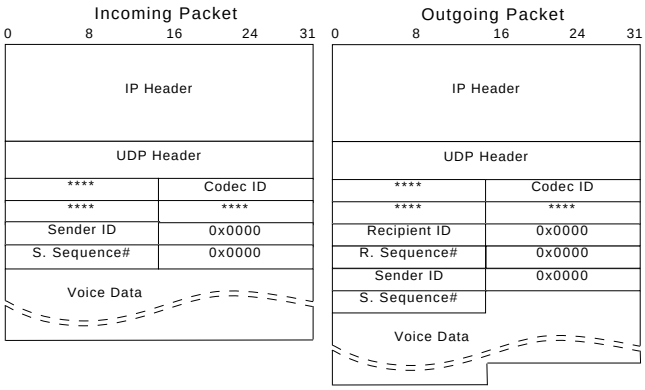


Figure 3.2: *TeamSpeak Voice Packet format:* The TeamSpeak protocol uses UDP to transfer the voice packets. During the replication the server inserts the recipient information in front of the sender information.

the recipient ID and sequence number. This explains the size difference between the incoming and outgoing packets. Figure 3.2 illustrates the TeamSpeak voice packet format.

Knowing the codec ID in the packets we could identify the used codecs. For the smaller packets it is the 12.3 kbps and for the larger ones it is the 16.3 kbps Speex codec, which is in accordance with our previous observation. However, unlike the Speex specification, the mean inter-arrival time for packets was measured to be 100ms (based on data from several million packets) and not 20ms as one would expect. Thus, we claim that TeamSpeak groups five Speex frames before they are sent out. While sampling audio for 100ms introduces a higher delay, it reduces the bandwidth overhead caused by the IP, UDP, and TeamSpeak headers.

Using the formula introduced above, we calculate the total bandwidth required by sampling 100ms of audio vs. only 20ms of audio in the Speex specification.

$$BW_{TS} = \frac{([155,161,205,211]+28)\times 8}{0.1} =$$
$$= 14.64, 15.12, 18.64, 19.12kbps,$$

$$BW_{Speex} = \frac{([28,33,38,43]+28)\times 8}{0.02} =$$
$$= 22.4, 24.4, 26.4, 28.4kbps.$$

BW_{TS} is the amount of bandwidth used by TeamSpeak with 100ms frames while BW_{Speex} is the bandwidth used by the Speex codec with 20ms frames. It can be seen that in case of the 12.3 kbps codec the difference in overhead would be more than 50%.

3.2.4 Filtering Voice Packets

After understanding the TeamSpeak packet format we could filter out the non-voice packets from the trace files. In the first step we simply had to filter for the size of the packets. This helped us to identify the voice packets with an accuracy of more than 99%. In the second step we refined the filtering by identifying the codec ID. We believe that after this step all our packets were only voice packets.

3.2.5 Data Cleaning

As we began the analysis of our data, we discovered that some of the data points were extremely different from the rest. These included excessively long talk sessions as well as silent periods. For example, the extreme outliers of the talkspurts were data points where voice packets were delivered for close to an hour, which would be fairly difficult to accomplish when you consider that we can detect silence gaps as small as 100ms! We reckon that these are rare occasions resulting from something such as loud background music or human behavior such as forgetting to log off while

leaving the computer for several hours. Therefore, we treated these data points as outliers and did not include them in our final results.

While removing extreme outliers can be controversial, we justify our actions by noting that our method removed few or no data points and that the methods used for curve fitting often pick the first and last end-points of the data to begin and end the curve, and then adjust values to force the rest of the graph to fit. Thus, the extreme outliers can cause a curve to not fit the data well, whereas by removing the outliers and fitting the curve allows one to obtain a better fit, according to various metrics. In Section 3.3.6, we detail the effects of our data cleaning.

To remove the extreme outliers, we first analyzed the linearity of the data. Prior research shows that talkspurt and silence periods often follow an exponential distribution [6, 17, 21, 41]. We also plotted preliminary graphs to get an idea of the general trend of the data. Note that linearization for the purpose of cleaning does not need to be perfect (e.g., we used an exponential form, though our data turned out to be Weibull). The purpose of this process is to remove extreme outliers and, given the large number of data points, removing only a small percentage of data points is acceptable. When the data was not linear, we saw that it followed the following form in every case:

$$y = ae^{bx}.$$

In these cases, to linearize the data we took the logarithm of both sides, which resulted in the following equation:

$$\ln(y) = \ln(a) + bx.$$

Once the data was linearized, we identified the first and third quartiles. As we tried

to delete as little data as we could we decided to delete only the extreme outliers. These are the data points that are beyond the outer fences:

$$< Q_1 - 3IQR,$$

or

$$> Q_3 + 3IQR.$$

Here, Q_1 is the first quartile, Q_3 is the third quartile and IQR means the inter-quartile range $(Q_3 - Q_1)$. Thus, only data that was beyond 3 times the inter-quartile range less than the first quartile or greater than the 3rd quartile was removed. While methods that remove all the outliers and not just the extreme ones use $1.5IQR$ we decided to use $3IQR$ and only remove a very small amount of data, which we felt was sufficient for our purposes.

3.3 Measurements

Our first measurement covers a 3 month period from December 2006 to February 2007. During this time, we measured over 7000 sessions from over 800 IP addresses dispersed geographically for an average of 1.46 GB/day in traffic. Five month after the start of our first measurement we set up another one from May 2007 to Jun 2007. Here, we measured another 4000 sessions from 440 IP addresses. The average traffic increased to 1.72GB/day.

3.3.1 User Geographical Distribution

In order to ensure that our data was not biased due to the geographical location of clients connecting to the server, we took advantage of the fact that all the client IP

addresses were obtainable from our log files. Thus, we could estimate the locations of the clients and ensure that they were not all from the same place.

Using the free MaxMind tool, GeoLite Country[4], we determined the latitude, longitude, country and state where applicable of each IP address. This free version of the software claims to have over 98% accuracy.

After processing our data we found the following:

- More than 84% of our users were from North America.
- More than 60% of our users were from the United States.
- Each of the 7 most populous states is responsible for more than 2.5% of the US users and they together are responsible for more than 45% of the US users.
- None of the remaining states reaches the 2.5% limit.

Table 3.2 shows that the majority of our users are from the United States and Canada. This means that our results are not biased because of the different time zones. In the last two rows of the table we listed the ranking of the countries in terms of the measurement periods. The first period includes December, January and February whereas the second includes May and June. It can be seen there are no significant differences. The only two countries - the USA and Canada - that could score more than 10% were the first and the second in both cases. The countries that fall in the 1-10% range also remained the 3rd, 4th and 5th.

Table 3.3 contains the seven most popular states in the United States and the amount of players that they are responsible for. This shows that the player distribution inside the United States is slightly biased towards the east cost, which is in accordance to the natural US population distribution. We can also see that the first

[4] `http://www.maxmind.com/app/geolitecountry`

Table 3.2: *Geographical User Distribution:* Heaviest user distributions by country with rankings of each country for December, January and February (DJF) and May, June (MJ).

Country	**Player Distribution %**	**DJF**	**MJ**
United States	60.82	1	1
Canada	23.54	2	2
Singapore	7.02	3	5
Australia	4.50	4	3
New Zealand	3.48	5	4
Germany	0.16	7	6
Japan	0.16	6	N/A
Bulgaria	0.08	7	N/A
Poland	0.08	N/A	6
Puerto Rico	0.08	7	N/A
Republic of Korea	0.08	7	N/A

Table 3.3: *Geographical User Distribution:* Heaviest user distribution by state with overall state ranking in December, January, February (DJF) and May, June (MJ).

State	**Player Distribution %**	**DJF**	**MJ**
Pennsylvania	13.51	1	3
Texas	7.79	6	1
New York	7.40	2	4
California	6.23	3	5
Florida	4.94	5	2
Colorado	3.90	4	11
New Jersey	2.73	8	8

five states were relatively popular during both measurement periods. *We conclude that the primary result of our server location being in the MST time-zone is simply that most users are from the US and Canada.* Generally, server location affects the user locations due to latency issues, but given the wide-spread locations of users within the continental US and Canada, our data is not biased towards a particular area within these two countries, except to follow natural populations.

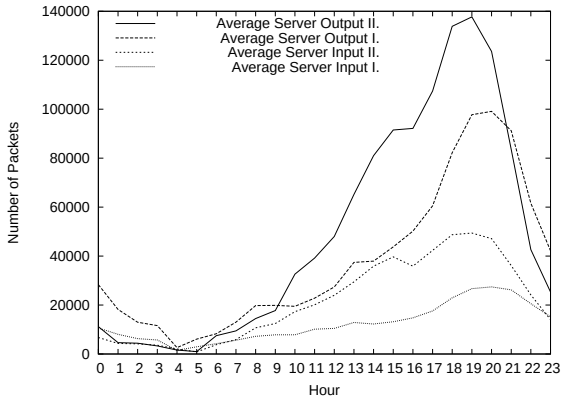


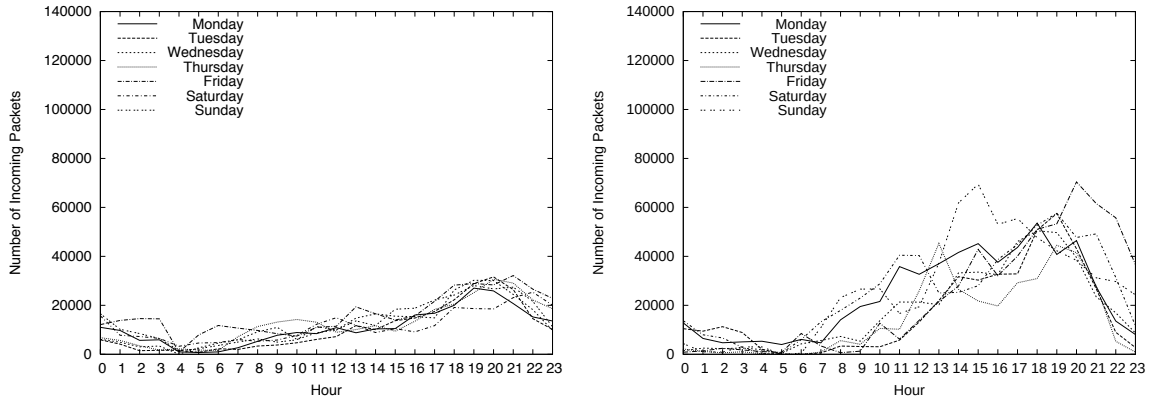
Figure 3.3: *Server Traffic:* Average voice server traffic over a 24 hour period (times shown are MST). The sizes of input and output voice packets are 155 or 205 and 161 or 211 bytes respectively. This result shows that server input doubled while server output quintupled during evening hours. The peak is around 6pm-9pm whereas the most quite period is around 4am-5am.

3.3.2 Overall Server Traffic

The first set of measurements we present are the overall traffic seen by the server during an average day. Figure 3.3 shows the averages, averaged per hour on the x-axis and the number of packets sent and received on the y-axis for the first and second measurement periods. Thus, this figure is an indication of the volume of traffic seen by the TeamSpeak server. Incoming voice packets are always 155 or 205 bytes while outgoing voice packets are always 161 or 211 bytes respectively.

This result shows that server input doubled and server output increased by an approximate factor of 5 during the evenings (approximately 6pm-9pm MST[5]). This indicates that more users are online using multiparty voice communication during the evenings. Note that the local minimum is at 4am MST during the first measurement and 5am MST during the second. Although the peak also changed it shifted one hour earlier and not later, from 7pm-9pm to 6pm-8pm. This is probably due to the fact

[5]Throughout this paper, times are listed as MST, but this is only as a convenience indicating the time-zone the server is located in and has no bearing on the measurements or results.



(a) December 2006 - February 2007

(b) May, June 2007

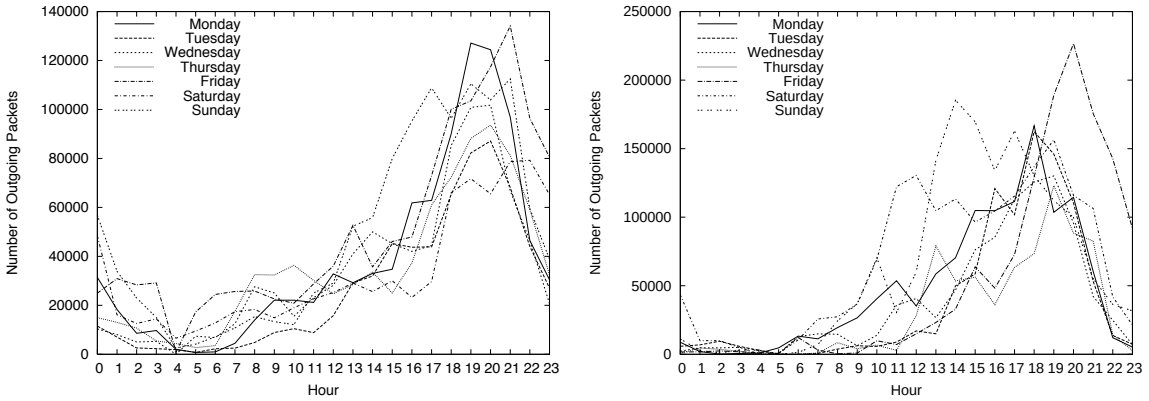
Figure 3.4: *Server Input:* Average voice server input traffic over a 24 hour period (times shown are MST). The packet size of an incoming voice packet is 155 or 205 bytes. This result shows that server input is similar on all days, with a peak during evening hours.

that the second measurement was conducted during late spring and early summer and therefore our players had more free time to play.

We also observed that the peak periods (7pm-9pm and 6pm-8pm) are not only the peaks of our traffic but the traffic rate is also almost constant here. In other words, the number of sessions started is the same as the number of sessions finished during these periods and thus resembles a balanced birth-death process.

We hypothesized that traffic was actually higher on weekends, and therefore we divided our averages into individual days so that we averaged all Mondays separately, all Tuesdays separately, etc. Figure 3.4 shows the inbound server traffic from users. From this figure, we see that most days are very similar with a small amount of variance, though in terms of total input, Fridays and Sundays have the highest amount of inbound traffic.

In Figure 3.5, we can more clearly see that the server output has more traffic on weekends than on weekdays. In addition, Sunday traffic increases earlier than on any



(a) December 2006 - February 2007

(b) May, June 2007

Figure 3.5: *Server Output:* Average voice server output traffic over a 24 hour period (times shown are MST). The packet size of an outgoing voice packet is 161 or 211 bytes. This result demonstrates that server output is higher and extends over more hours during the weekends, and in particular on Sundays.

other day, starting at 1pm MST while the peak of traffic is highest late on Friday evenings at approximately 130k packets/hour in the first and 230k packets/hour in the second measurement. Please note that, while we tried to keep the scaling consistent here we had to rescale the y axis of the second figure in order to make the data fit. Interestingly, Saturdays have a lower peak traffic than Sundays or Fridays, but have a higher average traffic during the early hours of the day. This difference is most likely due to people who are on late Friday continuing to use TeamSpeak into the early hours of Saturday morning (and then probably sleep late that day).

A final interesting trend is that Monday also seems to have a high outbound traffic peak that is similar to Friday. The primary difference appears to be that the traffic is shifted about two hours earlier, probably because game players start earlier so that they can go to bed earlier.

Given the TeamSpeak architecture, which unicasts packets to other players in the same channel, these results provide an insight into the size of the group that is talking

to each other in the same channel. First, on Fridays, the output is approximately 4 to 5 times the size of the input. This implies that for each voice packet that is input, TeamSpeak is replicating it 4 to 5 times, for a group size of 5 to 6. On a day such as Tuesday, the traffic is 2 to 3 times that of the input, indicating group sizes on average of 3 to 4. We conjecture that on Fridays and Sundays, game players are more likely to use multiparty communication to converse with a larger group of other people than on other days. Most likely this is because players have more free time on those days and are able to coordinate getting together online with other players more readily.

When we look at this data in conjunction with the general server traffic, we see an interesting trend. Even though group sizes may increase, the amount of incoming traffic does not increase at the same rate as the outgoing traffic. Given these traffic patterns we believe that while many people may be able to talk at the same time in a large group, human protocols prevent this from occurring. Typically, only one person can talk at a time and they take turns during the sessions. In essence, if more than one person begins talking, the speakers stop to allow only one person to talk so that the conversation can be understood.

Although we are unaware of any previous work that investigates multiparty voice communication, we expect these results to be similar to traffic patterns seen on game servers. Indeed, similar diurnal patterns and weekly patterns have been observed in related game traffic measurement work [19, 34, 45, 47]. There is clearly a peak, a local minimum each day, a strong correlation between days and a higher load on the weekends.

3.3.3 Inter-Packet Arrival Time at the Server

We measured the inter-packet arrival time at the server and plotted the ratio of packets to the inter-packet arrival time in Figure 3.6. Note that this includes *all* voice

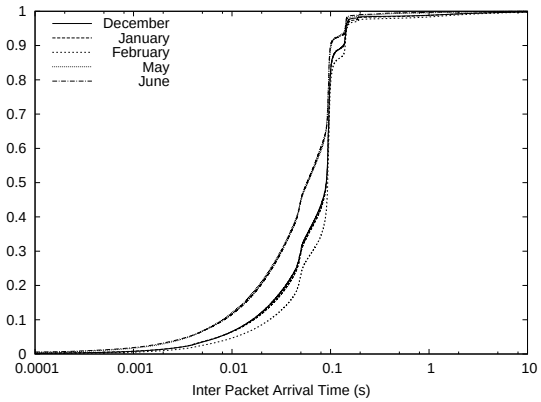


Figure 3.6: *Inter-Packet Arrival Rate:* The duration between the majority (80%) of packets is between .01s and .1s. This indicates a rate of packets between 10 to 100 per second depending on traffic.

packets received by the server during the measurement periods. Due to the streaming nature of the data, since encoded packets are immediately sent out from the client to the server, we see a packet rate between 5 to 100 per second at the server (depending on the number of online users). The larger gaps between packets seen on the server are most likely due to silence during the conversations when few people are logged onto the server.

The inter-packet arrival times show a heavy-tail that is 2 to 3 orders of magnitude larger than the average inter-packet arrival times. Note that this tail is only a small portion of total packet arrival times because we have a client/server architecture. Given a peer-to-peer architecture, we would expect larger periods of silence on any given peer in the system.

3.3.4 Group Sizes

We next examine group sizes to gain an insight into the size of a group that is typical in multiparty voice communication when used with games. As we noted previously, the ratio between the inbound and outbound traffic is an indicator of the average

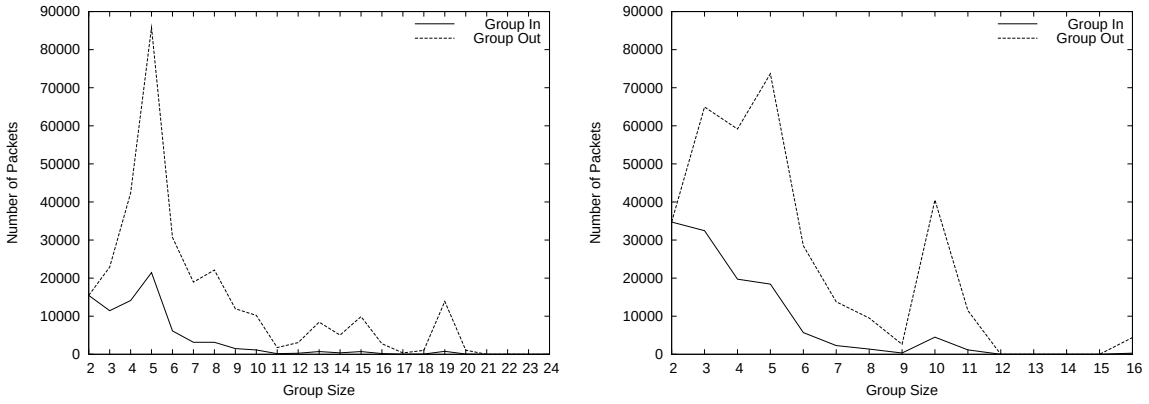
group size.

To perform this measurement, we looked at the trace logs and determined the sender ID for all the outgoing packets. The number of outgoing packets with the same ID and the same sequence number is one less than the actual group size. In processing the group sizes, we found that the server periodically sent duplicate messages for no particular reason. These were filtered out and are probably due to a TeamSpeak bug.

We binned all data according to how many people it was duplicated to, allowing us to examine the data based on the size of the group. Thus, we can determine the effect of the groups with different sizes on both the incoming and outgoing traffic on the server. Note that, neither the server log file nor the TeamSpeak packet format provides information about the used channel and thus it is impossible to identify the actual groups based on the packet alone (this control information must be shared between server and client in non-voice packets). The results are seen in Figure 3.7. Each of the graphs contains two curves. The solid line indicates the incoming traffic and the dashed line indicates the outgoing traffic that is simply the incoming traffic multiplied by the number of players that the particular packet is replicated.

During the first measurement period the results on group sizes show that the most active groups are the ones that are formed by 5 people. It can also be concluded that these groups generate the most outgoing traffic among all the groups. It is also worth mentioning that the amount of incoming packets from groups formed by pairs is almost as high as it is by these groups, but because of the replication the outgoing traffic is much less affected. Similarly, the counter effect can be seen in case of groups that contain 19 people. Although the amount of incoming traffic is low, the amount of outgoing traffic is high due to the large amount of replication necessary. The largest group we observed was 24 people.

During the second measurement period the results are slightly different. Here, the



(a) December 2006 - February 2007

(b) May, June 2007

Figure 3.7: *Group Traffic:* Incoming and outgoing traffic, categorized by group size over the measurement period. We see that, on average, the most groups are between 2 to 8 people talking, with a maximum of 24 people in a group.

most active groups are the ones that are formed by two people. However, the groups that affect the outgoing traffic the most are the ones that contain 5 people. The largest group size is also slightly smaller, 16.

We believe that a correlation between using TeamSpeak and the game being played exists. Currently, one of the most popular online games being played is World of Warcraft. In this game, players are often limited to 5 people in special areas, biasing the data towards a small group of people talking and playing the game together. On the other hand, a large class of multiplayer games, called *first-person shooters*, tend to group players into two groups, each between 8 and 16 players. Multi-party voice communication has also become very important for this class of games. If our TeamSpeak server was used by players of these kinds of games, we would expect the group sizes to correlate. Thus, we concluded that our server was mostly used by players on games which promoted small groups. However, because determining the game being played is impossible from our logs, and because the server was advertised to a wide variety of sources, we believe our results are general enough to at least

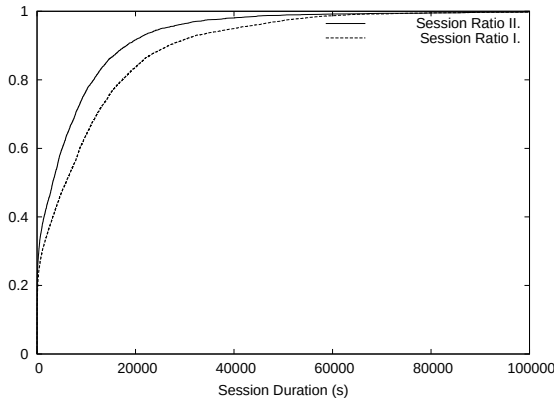


Figure 3.8: *Sessions Length CDF:* We see that of the 7,749 sessions we recorded, half of these sessions were less than 5000 seconds (1.3 hours). A small fraction of these (a few hundred) were over 30,000 seconds (8 hours).

apply to MVC for games in general.

3.3.5 Sessions Characteristics

Over the measurement periods, we recorded 7,749 and 3632 sessions respectively, including the packets that were sent to and from the server and how long users were logged into TeamSpeak. *On average, we observed 75.9 logins per day from 1266 individual users.* To understand this data further, we calculated the session times and generated a CDF as shown in Figure 3.8.

Our calculations show that the shortest sessions were less than one second while the longest session was over 69 hours! However, as Figure 3.8 shows, for 20% of the sessions, users stayed less than 1/2 hour. In addition, 20% of the sessions, users stayed for more than 5 hours. Thus, 60% of the sessions fell somewhere between 1/2 hour and 5 hours.For the small fraction of sessions that were greater than 8 hours, we hypothesize that users simply did not log out of the TeamSpeak server when they were done.

The characteristic of our curve is similar to both that can be found in [19] and in

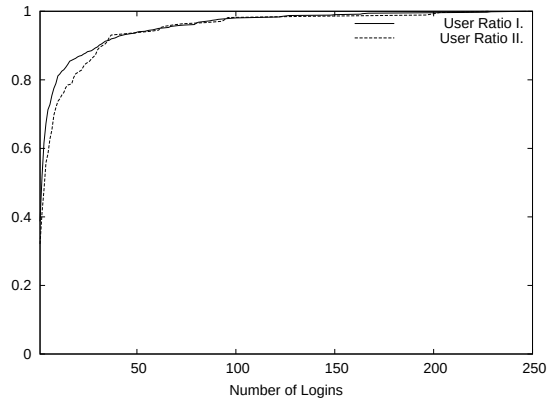


Figure 3.9: *Login Count CDF:* 40 % of all the IP addresses that logged to our server were unique. This is probably due to the fact that DHCP was used to assign their addresses. 17% appeared to log into the server at least once a week on average.

[45]. However, both of these papers analyze on-line games, one of them focuses on a First Person Shooter (FPS) game whereas the other one focuses on a Massively Multiplayer Online Game (MMOG). On the one hand this fact validates our results but on the other it shows that it is nearly impossible to conclude what type of game is played by the users analyzing only the characteristics of the data and not the content of it.

In the next measurements, we matched IP addresses with sessions to determine how many unique IP addresses logged into the system. In essence, we would like to determine how frequently a user logs into and uses the TeamSpeak server. We calculated the CDF of the ratio of logins versus the number of logins as illustrated in Figure 3.9.

Our results indicate that 40% of the users logged into the TeamSpeak server only once, while only 17% logged into it regularly (i. e. at least once every week on average). However, this result is most likely biased due to the fact that some users may be using DHCP to receive their IP addresses when they use the Internet. Thus, multiple IP addresses may refer to the same user and the total number of users we

saw may be fewer.

3.3.6 Measured Voice Patterns

Voice patterns in multiparty voice communication consist of talkspurts (on periods) and silence (off periods). We measured these and the inter-talkspurt arrival time to characterize voice patterns. TeamSpeak uses 100ms long frames, therefore the shortest talkspurt in our case is 100ms. To be consistent, the smallest measureable silence period must also be 100ms. The inter-talkspurt arrival time is measured as the time between any two in-sequence talkspurts observed by the server (see Figure 3.10). Since the smallest talkspurt is 100ms and the smallest silence period is 100ms, then any inter-talkspurt arrival time that is at least 200ms is interpreted as silence. Note that, if we have multiple users using the server at the same time the talkspurts can overlap and thus the inter-talkspurt arrival time can be shorter than the talkspurt itself.

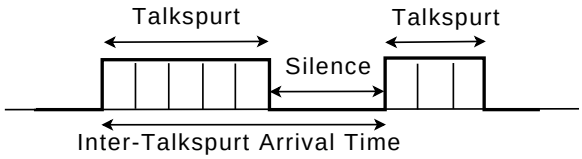


Figure 3.10: *Talkspurts, Silence Periods and the Inter-Talkspurt Arrival Time.*

In order to measure the voice patterns, we captured the voice packets during the peak periods (7pm–9pm and 6pm–8pm). After sorting and analyzing the data we realized that our data points did not fit on a linear curve. Therefore, we identified the extreme outliers using the method described in Section 3.2.5 and removed them from our data set. When we applied the cleaning procedure to the inter-talkspurt arrival times, we deleted 341 and 217 data points. Table 4.1 shows the results of cleaning the inter-talkspurt arrival times.

Table 3.4: *Cleaning the Data Sets:* The effect of removing extreme outliers with the cleaning procedure on inter-talkspurt, talkspurt, and silence periods data sets.

(a) December 2006 – February 2007

	Inter-talkspurt arrival	**Talkspurt**	**Silence**
Original data size	188,225 (100%)	188,313 (100%)	186,626 (100%)
Filtered data size	187,884 (99.82%)	188,158 (99.92%)	186,626 (100%)
Deleted data size	341 (0.18%)	155 (0.08%)	0 (0.00%)

(b) May, June 2007

	Inter-talkspurt arrival	**Talkspurt**	**Silence**
Original data size	225974 (100%)	228560 (100%)	227577 (100%)
Filtered data size	225757 (99.90%)	228559 (100%)	227577 (100%)
Deleted data size	217 (0.10%)	1 (0.00%)	0 (0.00%)

Figure 3.11 plots the inter-talkspurt arrival times seen at the server during the peak periods. The majority (90%) of the inter-talkspurt arrival times is less than 7.65 sec and 4.89 respectively. However, the remaining 10% of the data forms a tail which stretches to 536.83 seconds. Note that we only include the first 100 seconds in the graph so that the CDF can be seen more clearly.

We collected the talkspurts and silence periods for each of the users during the peak periods. We then merged these sets into a single data set and found that the data was non-linear. We transformed it and deleted the extreme outliers with the results listed in Table 4.1.

In Figures 3.12 and 3.13, we plot the CDFs of the talkspurts and silence periods, respectively. Both CDFs appear to follow an exponential distribution, which we explore further in Section 3.4. However, the expected value of the talkspurts is much lower than the expected value of the silence periods. 90% of the talkspurts are shorter than 5.40 sec and 5.07 sec, whereas the same measure for the silence periods is 70.11 sec and 27.88 sec, which is around an order of magnitude higher. This implies that

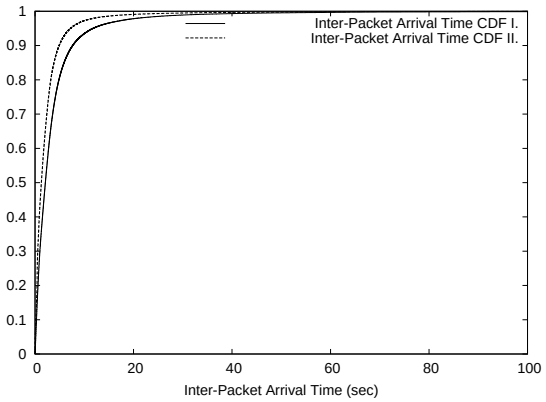


Figure 3.11: *Inter-Talkspurt Arrival Time:* The majority (90%) of the inter-talkspurt arrival times are less than 7.65 sec and 4.89 sec respectively. Although, we can see a change the exponential characteristic of the curve results in a long tail in both cases.

the users tend to listen more than to talk. After the filtering process, our lowest talkspurt value was .1 sec and our highest value was 96.46 sec. This can be seen in Figure 3.12. It is interesting that the length of silence periods dropped dramatically. However, this clarifies why the inter-talkspurt arrival times dropped too.

When we analyzed the silence periods, the filtering process did not effect our data set (see Table 4.1). This is due to the fact that the expected value of our exponential-like curve was higher and thus the IQR was broader. In addition, because our measurements were only performed during the peak periods, the silence periods have an upper bound of 3 hours (or 10,800 secs). The silence period data set ranged from .1 sec (the minimum possible silence period) to 7036.95 sec (almost 2 hours). However, in order to examine the curve of the CDF better, we only include the first 1000 seconds in Figure 3.13.

In Section 3.4, we model the data sets mathematically and discover that both the talkspurt and silence periods are better modeled by Weibull distributions. This makes sense when one considers that exponential distributions are a special case of Weibull distributions where the shape parameter is set to 1.

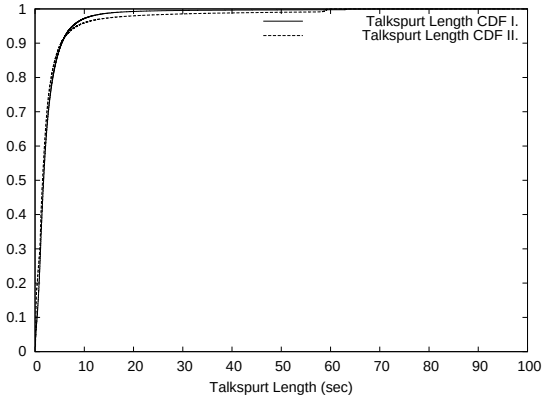


Figure 3.12: *Talkspurts:* The majority (90%) of the talkspurts are less than 5.40 sec and 5.07 sec respectively. The CDF appears to follow an exponential distribution. During the second measurement period the majority of the packets remained to fit in the same range.

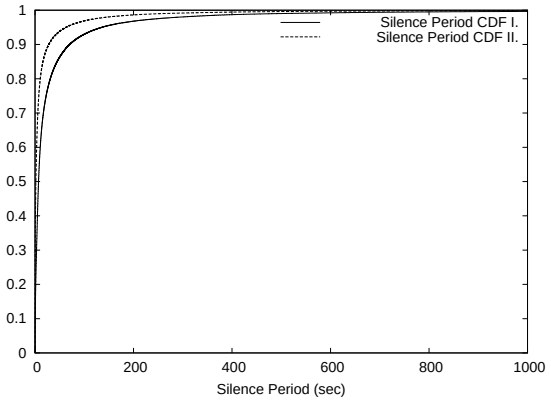


Figure 3.13: *Silence Periods:* The majority (90%) of the silence periods are less than 70.11 sec and 27.88 sec respectively. The CDF appears to follow an exponential distribution. However, the expected value of the silence periods is much higher than the talkspurts.

3.4 Modeling Multiparty Voice Communication

We now turn to the modeling of multiparty voice communication. We have three primary factors that we need to model mathematically: talkspurts, silence and group sizes. With these three models, we can simulate and predict the characteristics of multiparty voice communication, regardless of whether a client/server, peer-to-peer or hybrid architecture is used.

3.4.1 Methodology

Initially, we thought that the data appeared to follow some kind of exponential distribution, but as we analyzed the data further, we discovered that it fits a Weibull distribution better. Note that this differs from previous research in classical telephony and VoIP conversations which showed that the data followed an exponential distribution.

In order to model the conversations, we first estimated the parameters of the exponential and Weibull distributions. We looked at other distributions, but found that these two distributions had the best fit with our data. We then validated our estimation by calculating the mean and standard deviation of the residuals and by using the λ^2 test.

3.4.2 Parameter Estimation

We used two parameter estimation techniques. For the exponential distribution, we used the Maximum Likelihood estimation:

$$L(\lambda) = \prod_{i=1}^{n} \lambda e^{-\lambda x_i} = \lambda^n e^{-\lambda n \overline{x}}$$

where $(x_i, ..., x_n)$ are our data points and $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$, which is the average of our data points. The derivative of the likelihood function's logarithm is:

$$\frac{d}{d\lambda} \ln L(\lambda) = \frac{n}{\lambda} - n\overline{x}$$

and therefore our estimated rate parameter for the exponential distribution is:

$$\hat{\lambda} = \frac{1}{\overline{x}} = \frac{n}{\sum_{i=1}^{n} x_i}$$

For the Weibull distribution, our parameter estimation is based on the least-squares method which minimizes $\mathcal{S}$, the square of the sum of the residuals:

$$\mathcal{S} = \sum_{i=1}^{n} (y_i - f(\vec{x_i}, \vec{a}))^2$$

The data sets consist of the points $\vec{x_i} = (x_1, ..., x_n)$ and the function we are testing is of the form $y_i = f(\vec{x}, \vec{a})$, where $\vec{a}$ is the set of parameters that we are estimating and $\vec{x}$ acts as the independent values.

3.4.3 Error Calculation

In order to justify the correctness of our estimation, one could perform a goodness-of-fit test. However, traditional tests, such as Chi-square (χ^2) and Kolmogorov-Smirnov (KS) are not suitable for data from Internet traffic [32]. Moreover, these tests are biased against large data sets [16], such as the ones that we have.

We use two methods to determine if the data fits a particular distribution. After we have used the maximum likelihood or least squares method to estimate the parameters for a distribution, we plot the residuals and examine their mean and standard deviation. These values give us an idea of how well our model predicts the data. In

addition, we use the λ^2 method as a discrepancy tool [33]. We describe how we used the λ^2 method and how we binned our data in Subsection 3.4.4. With the λ^2 method, we can compare the fit between two possible distributions. Further, if $\lambda^2 > 1.0$, we can reject the distribution as a possible fit.

3.4.4 Using λ^2 for network model evaluation

The quantity λ^2 is the discrepancy between an actual and an assumed statistical model, which is the measure of the goodness-of-fit of the estimated curve. However, this method can be applied to data in different ways. Here, we present the details of how we applied it.

The λ^2 metric is defined as follows:

$$\lambda^2 = \frac{\chi^2 - K - df}{n - 1}$$

where n is the total number of datapoints and df is the number of degrees of freedom of the test.

$$\chi^2 = \sum_i \frac{(O_i - E_i)^2}{E_i}$$

and

$$K = \sum_i \frac{|O_i - E_i|}{E_i}$$

Since this discrepancy is based on Pearson's χ^2 test, it requires the binning of the data. Here O_i is the observed number of datapoints in bin i and E_i is the estimated number of datapoints in bin i. Please note that not just χ^2 but K is also dependent on the number of bins and therefore determining this parameter can be crucial. Choosing a too large parameter causes a too rough estimate; on the other hand if the parameter is too small than the distribution of the datapoints will be too smooth, equivalent

statistically to imprecise estimation.

In our paper we used $1 + 2 \times 2 \times \log_{10} n$ equiprobable bins [24]. If this was not a whole number we took the floor of it. This method ensures that if we have at least one datapoint the nominator of neither χ^2 nor K can be zero. Our experience is that these parameters were accurate and worked well with our data because they were in accordance with our visual based expectations.

3.4.5 Modeling Talkspurts and Silence

To model the talkspurts and silence periods, we looked at the packets sent and received during the peak periods on the server (from 7pm to 9pm and from 6pm to 8pm). We focus on these periods because the model needs to be able to predict the behavior under peak loads. After looking at the data, graphed in Figure 3.12, we hypothesized that the data followed some kind of exponential distribution.

Our first attempt at modeling the talkspurt and silence periods was to try an exponential distribution. Recall that the CDF of the exponential distribution is $1 - e^{-\lambda x}$, where the mean of the distribution is $1/\lambda$. Table 3.5 lists the means and parameters we estimated for the exponential distribution.

It can be seen that the λ^2 test failed for the silence periods during the second measurement, since its value was greater than 1. Thus, we decided to try the Weibull CDF, which is defined as $1 - e^{-(x/\lambda)^k}$. The Weibull CDF is related to the exponential CDF in that when its shape parameter, k, is 1, both CDFs are equivalent.

Using the least-squares method, we estimated the parameters for the Weibull distribution for both the talkspurts and silence periods. The results are shown in Table 3.5. We then plotted the talkspurt and silence data sets along with the Weibull CDFs and their estimated parameters. The talkspurt graph with its model can be seen in Figure 3.14. Visually, the Weibull CDF appears to be a good fit for the talkspurt data

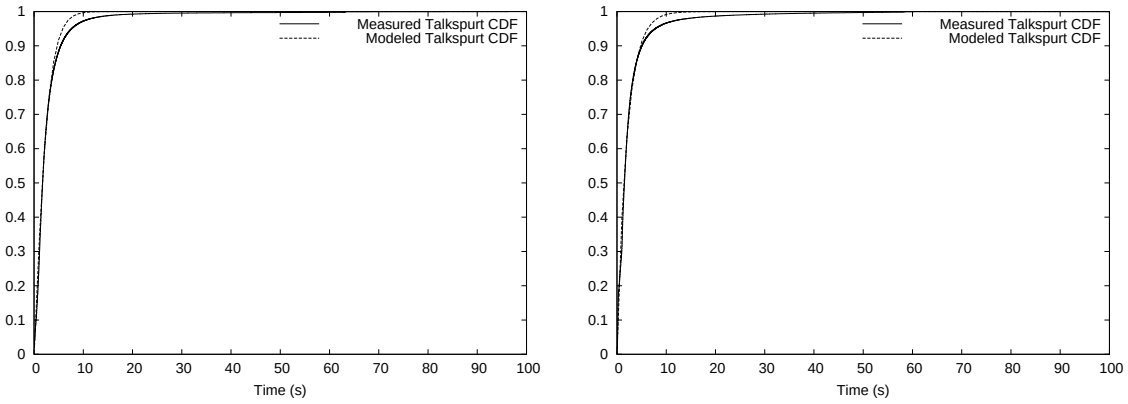
Table 3.5: *Experimental Values:* The Mean, Min and Max are calculated from the data sets. Using our parameter estimation methods, we calculated the parameters for the CDFs of the exponential and Weibull distributions. The λ^2 values are the results of using the λ^2 test to determine the accuracy of our fit (smaller is better). For both the talkspurt and silence data sets, the Weibull distribution is a better fit. Note that the exponential distribution is *not* a fit for the silence periods during the second measurement.

(a) December 2006 – February 2007

	Talkspurt	**Silence**
Mean	2.74s	35.90s
Min	0.1s	0.1s
Max	96.46s	7036.95s
Exponential estimated parameters	$\lambda = 0.3650$	$\lambda = 0.0279$
Weibull estimated parameters	$\lambda = 2.3002$ $k = 1.1846$	$\lambda = 13.5275$ $k = 0.6168$
λ^2-test for exponential	0.0993	0.7779
λ^2-test for Weibull	0.0769	0.0636

(b) May, June 2007

	Talkspurt	**Silence**
Mean	2.61s	16.60s
Min	0.1s	0.1s
Max	119.92s	9635.56s
Exponential estimated parameters	$\lambda = 0.3865$	$\lambda = 0.06068$
Weibull estimated parameters	$\lambda = 2.0183$ $k = 0.9746$	$\lambda = 3.7547$ $k = 0.4731$
λ^2-test for exponential	0.1716	1.2937
λ^2-test for Weibull	0.1694	0.5741



(a) December 2006 - February 2007: The Weibull CDF is plotted with ($k = 1.1846, \lambda = 2.3002$) as its parameters.

(b) May, June 2007: The Weibull CDF is plotted with ($k = 0.9746, \lambda = 2.0183$) as its parameters.

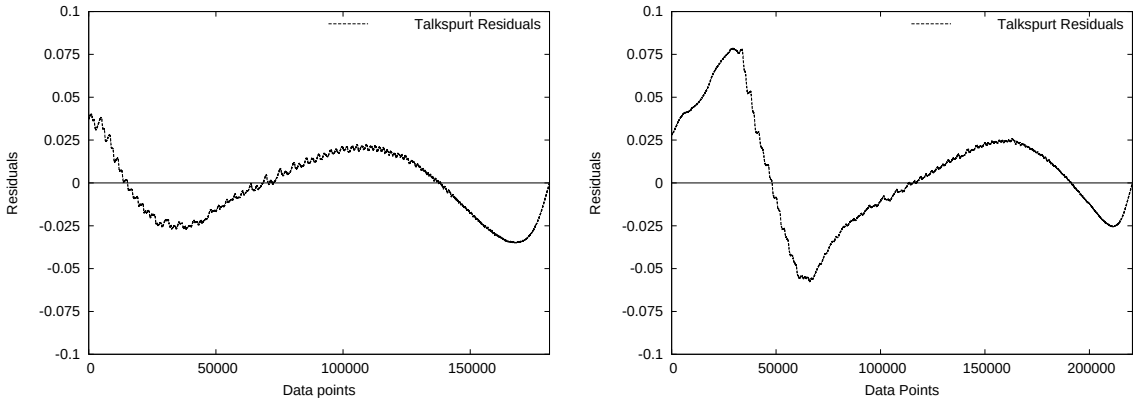
Figure 3.14: *Modeling Talkspurts:* Visually, we see that the Weibull distribution slightly overestimates the number of short talkspurts around the 10s range but otherwise it is a good fit in both cases.

Table 3.6: *Residuals from Model:* The max, min, and standard deviation of the residuals from the talkspurts and silence periods.

	Talkspurt I.	**Talkspurt II.**	**Silence I.**	**Silence II.**
Max	0.0401	0.0784	0.0269	0.0429
Min	-0.0350	-0.0576	-0.0382	-0.0721
Std.Dev.	0.0190	0.0321	0.0180	0.0266

set. Next, we plotted the residuals to examine the mean and standard deviation of the residuals. These graphs can be seen in Figure 3.15. Table 3.6 summarizes the results from our residuals.

We then used the λ^2 test on the CDF and discovered that the Weibull CDF fits better than the exponential as shown in Table 3.5. However, the shape parameter of the Weibull distribution is close to 1 in both cases, indicating that it is only slightly different from being an exponential distribution. Thus, *unlike prior results which showed that an exponential distribution better modeled talkspurts, we found that the Weibull CDF more accurately models the talkspurts of multiparty voice communication.*

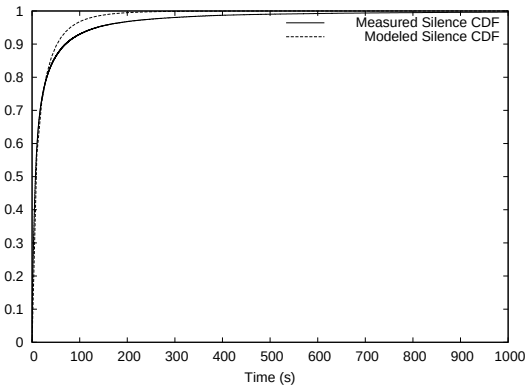


(a) December 2006 - February 2007 (b) May, June 2007

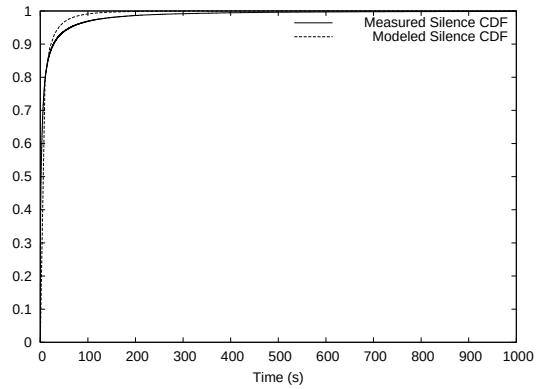
Figure 3.15: *Residuals of Talkspurts:* The residuals, which are the difference between the modeled and measured data, can give us an estimate of how far off any predicated values will be from the measured values. The talkspurt residuals remain within ±7.5% of the actual data set.

For the silence periods, we repeated our method of plotting the data set with the Weibull CDF and its estimated parameters, as shown in Figure 3.16. From this figure, we see that unlike the talkspurts the silence periods changed significantly between the measurement periods. This is an important observation because this implies that the talkspurts are independent from the type of game played and therefore can be used to describe any kind of traffic. On the other hand the silence periods can differ. In the following sections we are investigating if a relationship exists between the group size and the silence periods.

To further validate the results, we plotted the residuals, which are the differences between the predicted values and observed values. Plotting the residuals shows us that the model is off by at most 7.2%, with a standard deviation less than .03, as shown in Figure 3.17 and Table 3.6. Using the λ^2 test, we see that our estimated Weibull CDF is indeed a better fit than the exponential distribution (Table 3.5). Thus, *the silence periods are more accurately modeled with Weibull CDF for multiparty voice*



(a) December 2006 - February 2007: The Weibull CDF is plotted with ($k = 0.6168, \lambda = 13.5275$) as its parameters.



(b) May, June 2007: The Weibull CDF is plotted with ($k = 0.4731, \lambda = 3.7547$) as its parameters.

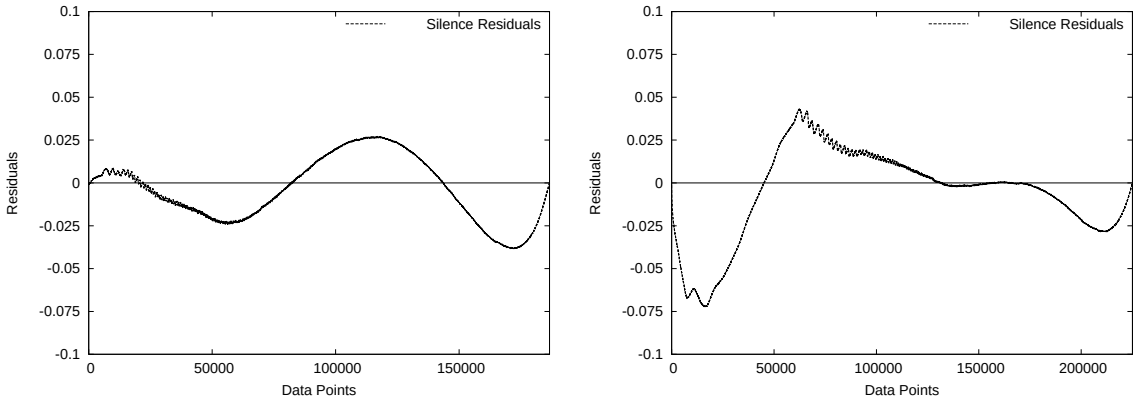
Figure 3.16: *Modeling Silence:* We see that unlike the talkspurts the silence periods changed significantly between the two measurement periods.

communications.

3.4.6 Modelling the Groups

In order to develop a complete model for multiparty voice communication, we also need to understand how talkspurts and silence change with group sizes. We hypothesized that as the number of people in a group increased, the mean talking time decreased while the mean silence time increased. To study this, we plotted the mean talkspurt and silence times versus the group sizes observed during our measurement period.

As TeamSpeak does not use a group identifier in the messages, it is impossible to identify the groups with 100% accuracy. However, for modeling the behavior of the groups with different sizes it is not essential to associate the messages to a particular group. Simply knowing the size of the group that a message was sent to would be sufficient if this method was also capable of grouping the silent periods based on the group size. Thus, we counted the number of replications for each of the incoming



(a) December 2006 - February 2007 (b) May, June 2007

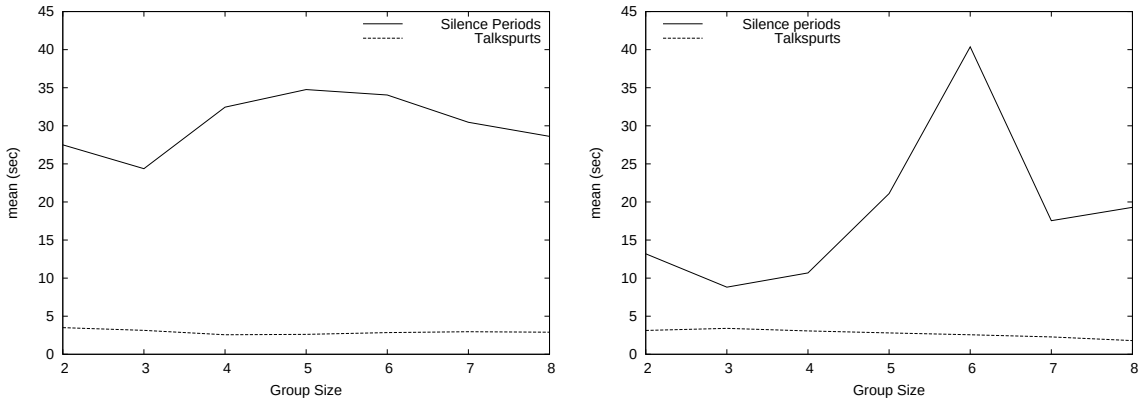
Figure 3.17: *Residuals of Silence:* The residuals, which are the difference between the modeled and measured data, can give us an estimate of how far off any predicated values will be from the measured values. As with the talkspurts, our residuals are typically $\pm 7.5\%$ for the silence periods.

messages from a given user. Next, we used this group size to determine the group size for the following silence period. This way we could associate a group size to both the talkspurts and the silence periods. The only time when our method fails is when a player leaves or joins a group during a silence period. However, this event is very unlikely and therefore our solution is capable of providing quite an accurate result.

Figure 3.18 shows the mean talkspurt and silence times versus the group size. We only show groups of up to 8 people due to the fact that while we did observe groups with up to 24 people, the number of data points in these larger groups were too few to be statistically meaningful.

Looking at this graph, we see that the mean talking time does not change significantly, regardless of the group size, contradicting our hypothesis. On the other hand, silence periods do have an upwards increase until around 6 people, at which point it decreases.

To investigate this unexpected result, we ran a script which looked at the number



(a) December 2006 - February 2007

(b) May, June 2007

Figure 3.18: *Talkspurts and Silence Periods Among Groups:* Note that the mean talkspurt time is fairly constant while the mean silence time fluctuates without a discernible pattern.

of people talking in a group and found that as the group size increases, the number of completely silent people increases (e.g., they only have headphones, but not a mic to speak on). Thus, more people may talk in a larger group, but they still follow the same talkspurt patterns we have seen independent of the group size. In essence, *the same amount of conversation appears to be carried on the same channel, regardless of group size.* Conversations may be dominated by a few talkative people, forcing others to remain silent until they can get a chance to speak. In terms of computer games there are probably friends who tend to talk to each other during the game and the others just try to collect information that is beneficial for them. Thus, *talkspurt periods appear to be independent of the group size, while overall silence periods increase as the group size increases.*

3.5 Conclusion

We have presented a work that examines the characteristics of multiparty communication for games. Our results show familiar and new trends. First, as we modeled

the talkspurts and silence periods, we found that both types of data fit a Weibull CDF, which differs from previous work on traditional telephony and VoIP that shows talkspurts following exponential distribution. Moreover, we showed that the length of the talkspurts are always the same regardless either of the game played or the group size. On the other hand, the distribution of our daily traffic was similar to other works in both games and VoIP. Their results are similar to what we saw, server usage peaked during the evening hours and on weekends.

Finally, human protocols seem to be at work here as our measurements indicate. The increase in group sizes does not increase the amount of input traffic linearly, though output traffic is necessarily linear in the number of packets received. This is simply due to the fact that humans best process voice information when only one person is talking at the same time. Thus, if more than one person starts talking, other speakers naturally back-off and wait for their turn.

Chapter 4

Protocol

4.1 Introduction

To date, most multiparty voice communication software uses a client/server architecture. This architecture is useful because it provides a centralized point for authentication, administration, and security. On the other hand, it requires a large amount of bandwidth to host, it is a single-point of failure, and it requires significant configuration by end-users if they are required to download and host a voice server for games they are playing.

We present a peer-to-peer architecture for multiparty voice communication that scales well with the number of participants and uses information from the virtual environment to determine how to connect nodes. This allows our protocol to send voice packets first to those who are closest to us in the virtual environment and to only cluster avatars who are in each other's area of interest (AOI).

Unlike most MVC software, our protocol uses the virtual locations of avatars to help form its distribution graph. This allows positional audio to be modeled accurately to the listeners and provides an increased level of immersion. Our protocol allows anyone

in the virtual world to talk to anyone else as long as their AOIs intersect. This differs from current games which limit talking to a special group, such as a party or team, or requiring them to log onto voice servers and all join the same channel. We note that clearly separate channels are trivially supported in our protocol and the NVE interface can easily allow voice messages to be blocked. On the other hand, because our protocols can determine who should hear a voice packet, more realistic environments can be created and virtual meeting areas can be more accurately modeled.

Prior work in peer-to-peer multiparty voice communication has used various graph building techniques for communication, such as Delaunay triangulations or Voronoi diagrams (which are related to Delaunay triangulations) [10, 31, 43, 46]. Distributed hash tables (DHTs) have also been used for streaming applications using publish/subscribe mechanisms [8, 36]. While they both work, they do not reflect reality. Delaunay triangulation has a high node-degree, which makes focusing on a given voice stream difficult. DHTs tend to organize nodes on the key-space to maintain logarithmic routing over the entire set of nodes. Therefore two close avatars in the virtual world may be several hops away from each other in the key space, causing added latency when they are expecting little latency because of their close proximity.

Our protocol relies on *Gabriel* graphs, which are subgraphs of a Delaunay triangulation of the entire graph. Gabriel graphs have the important property that they can be calculated locally [28] and that any two closest neighbors are guaranteed to be connected–thus voice packets which should go to neighbors will be sent to the closest neighbors first and then possibly relayed to further nodes in the network. This reduces latency between neighbors since they exchange packets with each other in a single overlay hop. They further have the advantage of having a low average number of neighbors, which helps in understanding concurrent voice streams.

On one hand, this type of connectivity suits interactive systems well. On the other

hand, it poses a major challenge: how can the system securely maintain interactive connectivity between participants while keeping the state in the environment consistent? Although prior research addresses a similar security issue [7, 23, 29, 30, 42], none of these solutions is efficient enough to handle interactive traffic; therefore they cannot be used for our protocol.

In the second half of this chapter we present a framework that is specifically designed for interactive, multimedia traffic, which uses the Delaunay graph as the underlying architecture. We demonstrate how it can prevent Denial of Service attacks, Black Hole attacks, and Eavesdropping, while maintaining the interactive connections between the nodes. Furthermore, we show that the features we use in our framework can be used on top of any unstructured peer-to-peer protocol that uses greedy routing and through simulations we show that our protocol performs well under a wide variety of virtual population distributions and node movements. We also show that our protocol introduces minimal overhead in the system and is resilient to both passive and active attacks.

4.2 P2P Voice Communication

In our protocol, we assume that each node has a position in a 2 dimensional space (though we can extend this to 3 dimensions) and an *Area of Interest*, or AOI, that indicates the farthest distance centered at the avatar's position that voice can be heard from. The protocol works by computing a Gabriel graph (described in Section 4.2.1) for nodes in the system in a completely distributed fashion. When an avatar talks, the voice packets are sent via an AOI limited broadcast from the talking node to its neighbors. Neighbors continue to forward the messages as long as their neighbors fall within the AOI of the talking node. Unlike previous protocols where two neighbors

in the graph may be connected because they are close by a metric such as delay, messages in our protocol only travel to nodes that are possibly interested in them (i.e., they are within the AOI of the sender). This reduces overall traffic and prevents nodes from acting purely as relays.

To handle joining and leaving the network and to assist in calculating the Gabriel graph, nodes also maintain a Delaunay triangulation (which can be done in a distributed fashion [25]). Note that if we combine our protocol with a client/server based virtual environment, maintaining the Delaunay triangulation is no longer necessary because the server can calculate neighbor sets on the server and inform each avatar which Delaunay neighbors they have. In this setup, peers would then perform a distributed calculation of the Gabriel graph and communicate between themselves without needing to further involve the server.

Throughout the chapter, we use the following notation:

- n : the number of nodes in the network
- $v_0, \ldots, v_{n-1}$: the nodes in the network (or vertices of the constructed graph)
- $\overline{v_i v_j}$: an edge between v_i and v_j
- $AOI(v_i)$: indicating the Area of Interest of node v_i. The $AOI(v_i)$ a scalar value that indicates the radius of a circle centered at the position of v_i.

4.2.1 The Gabriel Graph and Its Properties

A Gabriel graph is a type of graph that connects a set of vertices in the Euclidean plane under the following rule: two vertices v_i and v_j are connected by an edge whenever the disc with the line segment $\overline{v_i v_j}$ as its diameter contains no other points from the given point set. Figure 4.1 illustrates the Gabriel graph of three nodes. Both

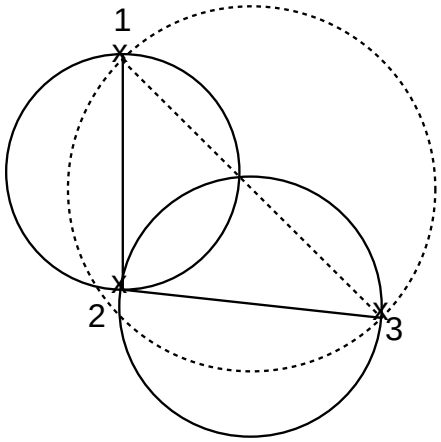


Figure 4.1: *Gabriel Graph Example:* Gabriel graph of three nodes in the plane. The graph is computed by adding an edge between two vertices if a disc which uses the edge as its diameter does not contain any other vertices, thus an edge is set between vertices 1, 2 and 2, 3. However, an edge is not set between vertices 1 and 3 because the disc formed by that edge contains vertex 2.

$\overline{v_1v_2}$ and $\overline{v_2v_3}$ are edges of the graph. However, $\overline{v_1v_3}$ is not an edge because the disc encircling this edge contains the vertex v_2.

Gabriel graphs are related to Delaunay triangulations in that a Gabriel graph is completely contained within a Delaunay triangulation and can be derived from it in $O(n)$ steps, where n is the number of vertices in the Delaunay triangulation.

In addition, Gabriel graphs contain both the Euclidean minimum spanning tree (MST) and the nearest neighbor graph. The MST ensures that the fewest edges are used when broadcasting from a speaking node to its listeners. The nearest neighbor graph is a graph such that for any pair of vertices, (v_i, v_j), a directed edge exists between v_i and v_j if and only if v_j is closer to v_i than any other vertex.

We have chosen to use Gabriel graphs because these properties give us a close approximation to real voice communication:

1. It always contains the nearest neighbor, which means any two avatars that are inside of each others' hearing range and are the closest to each other will always

be directly connected (note that the Gabriel graph may contain cycles).

2. It also contains the minimum spanning tree which ensures that voice packets take as few additional paths as possible, reducing the overall traffic on the network.

3. Being a subgraph of the Delaunay triangulation means that the minimum angle between edges is maximized. Avoiding narrow triangles allows one to create a more realistic simulation since the human voice spreads at a wide angle naturally.

Note that with a Delaunay triangulation, each vertex has on average six neighbors. For a Gabriel graph, the average number of neighbors is 4. This means that we have to replicate packets on average fewer times than with a Delaunay triangulation. The tradeoff is that with fewer edges, the diameter of the graph will be larger—but this should only be a factor in very dense graphs where we must reach a large number of listeners within an AOI. We believe this is an important tradeoff because bandwidth becomes an issue as we increase the quality of the voice packets (and therefore their sizes). In fact, listeners at the end of a long path will necessarily have many other listeners *in front* of them in the virtual space, or they would have had a shorter path since the graph is based on positions in the virtual world. Thus, they would realistically find it difficult to hear someone speaking in a crowd of people. With Gabriel graphs, these packets would be more delayed, but could also be dampened to simulate crowd effects.

4.2.2 Greedy Routing on the Gabriel Graph

We now show that we can route a message to the closest peer to a location in the network using a greedy algorithm. Assume we have nodes in a 2D plane. All nodes

have a pair of coordinates defining their positions. Define $M_{(x,y)}$ as the message being routed to location (x, y). Let $N(v_i)$ be the set of neighbors in the Gabriel graph of node v_i (recall that a neighbor is a node with an edge from itself to v_i in this case).

Theorem 3.1 *Routing a message $M_{(x,y)}$ over a connected Gabriel graph using the following greedy algorithm will always find a path to the node closest to (x, y). This greedy algorithm is defined as: When a node v_i receives the message from v_j, forward the message to the node from the set $N(v_i)\backslash v_j$ which has the closest Euclidean distance to (x, y).*

Proof. Assume the Gabriel graph is connected. Because the graph is connected a path must exist between any two nodes. We hypothesize that we can find a path from v_i to v_j by greedily choosing the neighbor v_k of v_i who is closest to v_j.

Construct a disc such that v_i and v_j lie in the disc and its diameter d is the distance between v_i and v_j. Let D_{ij} be the set of vertices that lie in the disc. We have to distinguish between two cases:

1. D_{ij} is the empty set: a direct edge exists between v_i and v_j. $\Rightarrow$ We traverse through the edge and reach our destination.

2. D_{ij} is not empty: there is no direct edge between v_i and v_j. $\Rightarrow$ Choose node $v_k \in D_{ij}$ such that $v_k \in N(v_i)$ and $|\overline{v_k v_j}|$ is minimal. Since $v_k \in D_{ij}$, the next statement also holds: $|\overline{v_k v_j}| < |\overline{v_i v_j}|$. Next, repeat the algorithm with v_k and v_j.

As we have a finite number of nodes, we get closer to the destination with every step, and eventually we get to the destination itself. □

Neighbor Sets

To maintain the necessary graphs in our protocol, each node maintains two neighbor sets:

- $D(v_i)$: the set of nodes in the network that are Delaunay neighbors, which we call the *Delaunay set*.
- $I(v_i)$: the set of nodes that are Gabriel neighbors of v_i from the Gabriel graph constructed using $D(v_i)$ and $AOI(v_i)$, which we call the *Interest set*.

The Delaunay set is used to maintain the Gabriel neighbors and handle joining and leaving. Note that the Delaunay set can be maintained via distributed Delaunay triangulation protocols [25]. The Gabriel neighbors can be then calculcated from the Delaunay set. Each node looks at its set of Delaunay neighbors and applies the Gabriel graph algorithm to them (decribed in Section 4.2.1), adding those nodes with an edge in the resulting Gabriel graph to the Interest set.

For the voice protocol, we are required to know which nodes are inside the AOI of a given node because we only need to route voice packets to nodes within the AOI. One concern with a Gabriel graph is that a path may exit the AOI of a given node only to re-enter at a later point. For example, assume that we have a similar layout to Figure 4.2. The solid line represents $AOI(v_1)$, and the dashed lines represent the diameters and the corresponding discs. Although the disc with diameter $\overline{v_1v_3}$ does not contain a third node, v_3 is not in $I(v_1)$ because v_3 is outside of $AOI(v_1)$. This illustrates that any node outside of the $AOI(v_1)$ will never have a voice packet routed to it directly from v_1, even though it may have an edge in the corresponding Gabriel graph.

On the other hand, $\overline{v_1v_2}$ is a valid edge only when its corresponding disc does not contain another node. Any point that is inside $AOI(v_1)$ defines a disc that is also

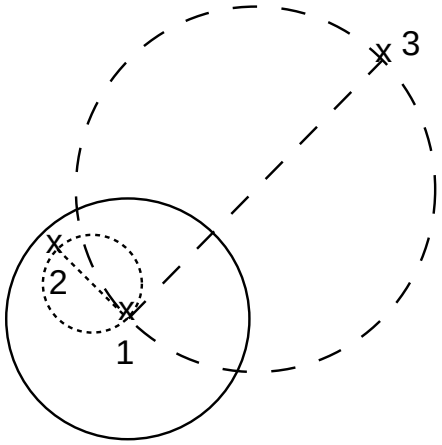


Figure 4.2: *AOI-Limited Broadcast:* In the full Gabriel graph of the network, an edge from vertex 1 to vertex 3 exists. However, because each node has an AOI, this edge can be ignored since the node at vertex 3 would be unable to hear anything said by vertex 1.

completely inside $AOI(v_1)$. As such, we only have to check the validity of edges as possible broadcast paths for the nodes in $D(v_1)$. Since these nodes are maintained by v_1 via Delaunay triangulation algorithms, routing over the Gabriel graph from v_1 only requires knowing a nodes AOI and broadcasting only to those neighbors who fall within the AOI of v_1.

The Voice Packet Graph and Protocol

The voice packet graph is a subgraph of the Gabriel graph of the whole network. It contains only those edges that are not longer than the radius of the AOI and may therefore be disjoint. Given the definition of a Gabriel graph and given all the nodes in the network, a connected graph would be generated. However, with our protocol, we throw out edges between nodes whose are not inside of each other's AOI since an avatar cannot hear beyond its AOI.

The transmission of the voice packets is done only along the edges of the voice packet graph. Every node that generates a voice packet attaches its coordinates and

orientation to the outgoing voice packets and then sends this packet out to all the nodes that are one hop from itself. These nodes then check the coordinates of the sender node and decide which of their neighbors they have to forward the packet to. Since the voice packet graph contains all the nodes that are in each other's hearing range, delivery is guaranteed to all neighbors who may need to receive the packet. Nodes outside of the AOIs of the senders will not be part of the same partition in the voice packet graph. Packets which need to be relayed arrive later than those which are only a single hop away, ensuring that avatars closest together receive communication between each other first.

Figure 4.3 shows a snapshot of the voice communication graph with 50 nodes and 0.15 radius. From this Figure, we can see that nodes close to each other are connected but the entire graph is not necessarily connected. On the other hand, long chains of nodes can be seen (e.g., a path from node 0 to node 11), but because voice packets include positional information, they are only forwarded to neighbors within the radius of the original speaker.

4.2.3 Building and Maintaining the Delaunay Triangulation

Every node in the network maintains its Delaunay neighbors. Using this set the nodes can calculate which other nodes are inside their AOI and which of these nodes are Gabriel neighbors. These Gabriel neighbors are then used to forward the voice packets in the network. We refer to this network as the *control network*.

The Delaunay Triangulation

A Delaunay triangulation is a triangulation of a set of points in a graph such that no point is inside the circumcircle of any triangle formed by any nodes in the network and can be computed in $O(n \log n)$ time for a 2D space. The Delaunay triangulation

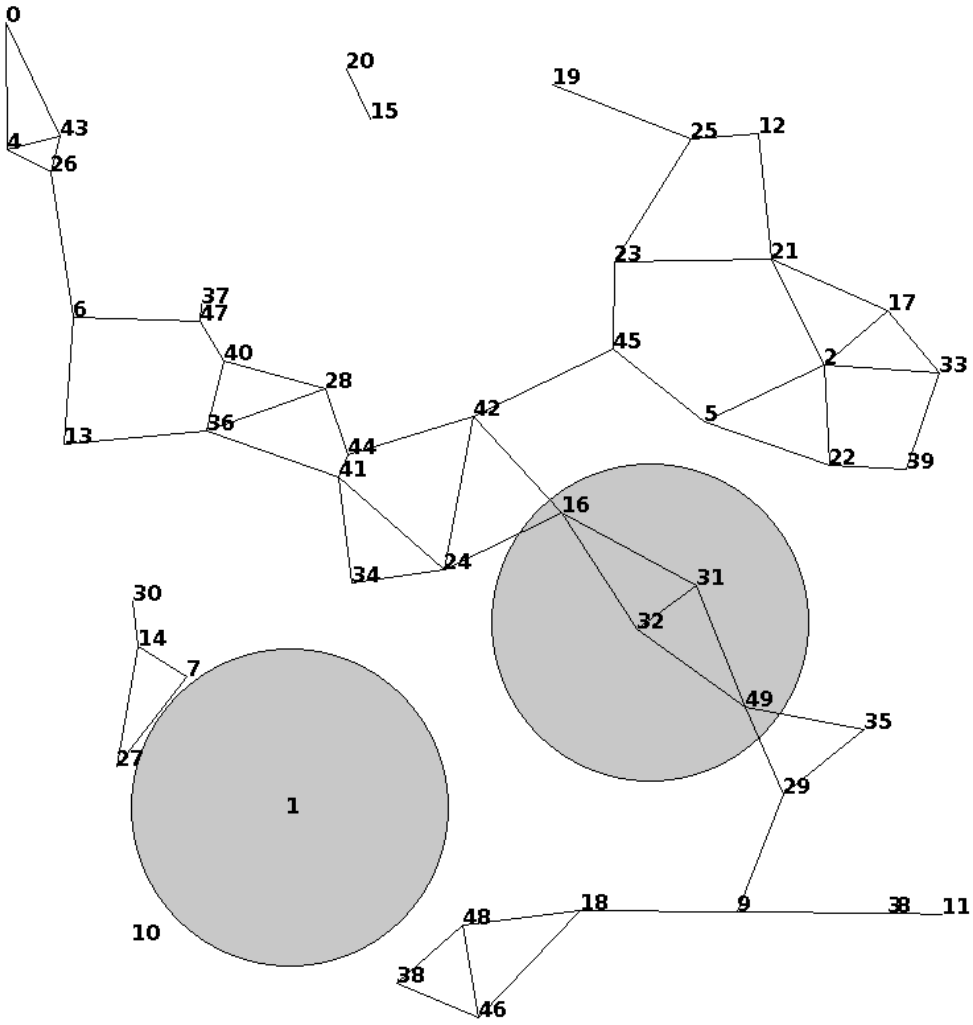


Figure 4.3: *Voice Packet Graph Example:* Note that some nodes are fully disconnected from the graph because their AOIs do not intersect with any other nodes. Nodes that are closest always have an edge between them. Further, some partitions in the graph have large diameters, but because voice packets include positional information, they are only forwarded to nodes within the AOI of the originating speaker. For better understanding we present the discs of node 1 and 32.

has been widely used to keep track of nodes in a network in a distributed manner. Although there are several ways to calculate the Delaunay triangulation, such as the flip, the incremental, or the sweepline algorithms, none of these are distributed.

Lee and Lam focus on the design of the *join*, *leave*, and *maintenance* protocols to construct and maintain a distributed Delaunay triangulation dynamically [25]. The *join* protocol assumes the knowledge of at least one node in the system so that it may *bootstrap* into the system. This node is then able to route the *joining* node to its closest neighbor using and appropriate routing algorithm. Next, a complete neighbor list exchange is performed recursively until no new neighbor is found. The *leave* protocol is not necessary because the *maintenance* protocol itself is sufficient enough to keep the system in a consistent state, but it can speed up this process. These protocols together are able to provide an underlying layer that keeps track of all the Delaunay neighbors of all the nodes in the system in a distributed way.

Bose and Morin investigate the different kinds of routing algorithms for triangulations [3]. They present a greedy routing algorithm, which simply forwards the packets from a node to the neighbor which is the closest to the destination. This algorithm always guarantees the delivery of a packet inside a Delaunay graph along some path to its destination. They also present the compass and randomized compass algorithms, which require less steps on average, but still has an O(n) worst case performance. To eliminate this issue, two more sophisticated algorithms are presented and described in their work.

The above mentioned methods are sufficient enough to build and maintain a Delaunay triangulation even for highly dynamic networks, such as peer-to-peer online games. Thus, we assume that such an underlying network exists that we can use for neighbor maintenance, and later for the Gabriel graph construction.

4.3 Protocol Simulation

In this section we evaluate our protocol by simulation. Initially the nodes in our network are distributed in a 1×1 square. We run each of the simulations for 60 seconds based on two different mobility models. In our simulations we sample the network in every 100 milliseconds, for a total of 600 times during the simulation, which is representative of the rate of voice packets typical in voice communication protocols (See Section 3.2.3). The radius of the AOI ranges from .1 to 1.6 in 6 steps, giving an effective diameter of up to 3.2, which is equivalent to not having any limit on the range of hearing in the virtual world. We simulate from 4 to 1024 total nodes. In our results, we present the average of all the samples collected, and detailed results for the most typical cases.

4.3.1 Mobility Models

During the simulations, the nodes move inside the square based on the random waypoint mobility model. The speed of the nodes is simulated with the Normal distribution, where the mean of the random variable is 1 and so is the standard deviation.

The destination of the nodes is chosen based on two different distributions:

1. Uniform: The x and y coordinates are chosen uniformly and independently from each other. This simulates the traditional random waypoint mobility model.

2. Exponential: The x and y coordinates are chosen together using a three-dimensional distribution that is a superposition of two exponential distributions (Figure 4.4). This models a virtual reality where the nodes tend to gather around two hotspots. The secondary hotspot has a peak that is 75% of the primary hotspot. This way the nodes still move from one peak to the other, but the nodes are closer

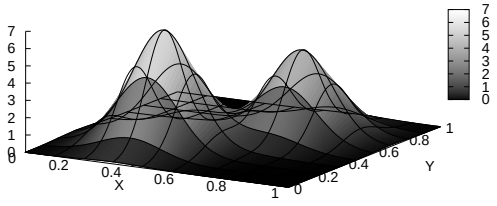


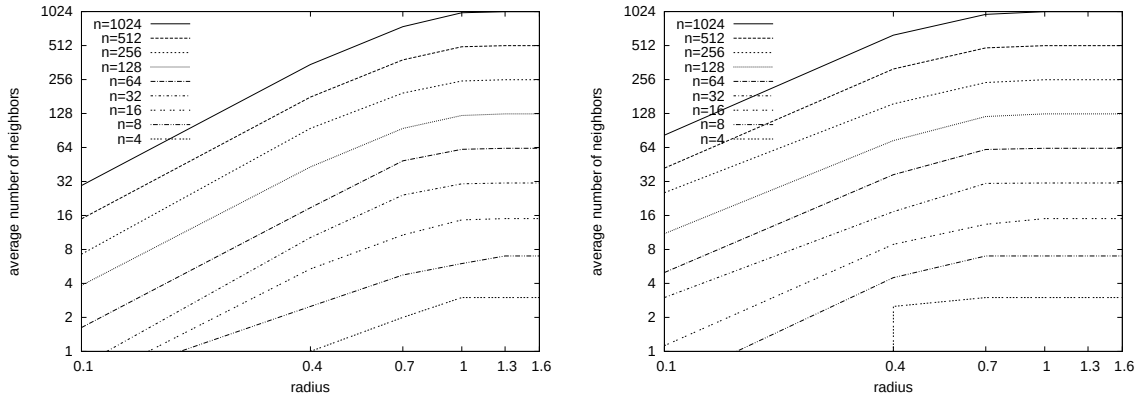
Figure 4.4: *Three-Dimensional Distribution to Choose Destination:* Having two hotspots as destination for the mobility model allows us to investigate the effect of clusters.

to each other and therefore form a cluster. Note that exponential distributions of players have been measured in large-scale, multiplayer games [34], leading us to this mobility model.

When a node reaches its destination, a new location and speed is calculated based on the distributions used.

4.3.2 Theoretical Boundary

The performance of our protocol depends in part on the number of nodes that are inside of the AOI of a given node since this determines the minimum bandwidth used for voice transmission. Figure 4.5 shows the average number of nodes that are inside of the AOI of a given node. We ran multiple simulations, and we varied both the number of nodes participating in the network and the radius of the AOI. As expected, the average number of nodes within the AOI is proportional to the radius of the AOI. Additionally, when the exponential mobility model is used, we see a twofold increase in the number of peers within the AOI.



(a) Uniform distribution

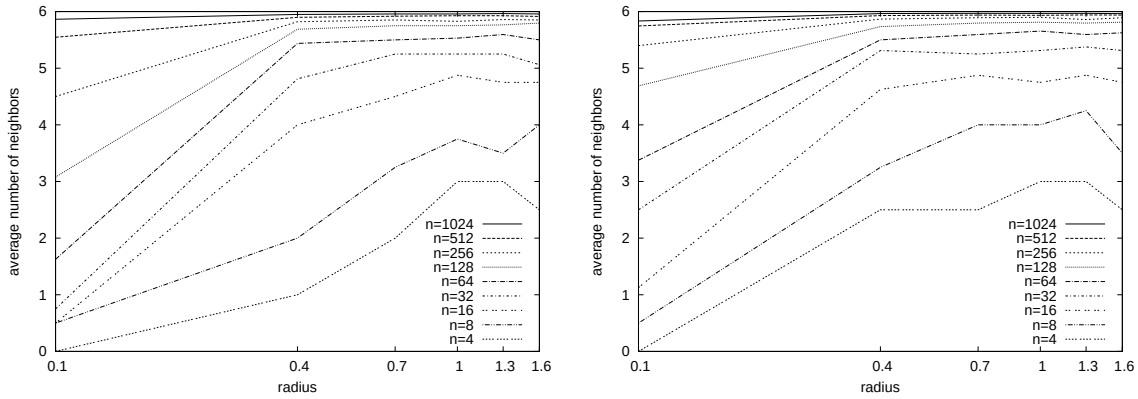
(b) Exponential distribution

Figure 4.5: *Average Number of Nodes Within the AOI:* The number of nodes within an AOI indicates how many peers a protocol will have to handle effectively on average. As the figure shows, the number of nodes inside the AOI increases proportionally with its radius. Note that the number of nodes in the network with the exponential distribution has almost twice as many neighbors, on average, as the uniform distribution.

4.3.3 Load Balance and Scalability

In these experiments, we measured load balancing and scalability of our protocol. Our metric for load balancing is the average degree of a node. While one possibility is to simply maintain the k-closest neighbors to communicate with, this could result in disjoint graphs. While Delaunay and Gabriel graphs will always be connected, they do not guarantee a low node degree and we therefore ran simulations to determine if these graphs have similar properties.

Our results show that both the Delaunay and the Gabriel graphs maintain a low node degree on average with a low standard deviation. Figure 4.6 shows that the maximum average degree of a node in the Delaunay graph is six, which is in accordance to the theoretical average. Thus, the Delaunay triangulation not only guarantees that all of the nodes are connected and therefore any avatar is reachable by any other avatar but it also creates a graph where the average node degree is low and therefore has a



(a) Uniform distribution

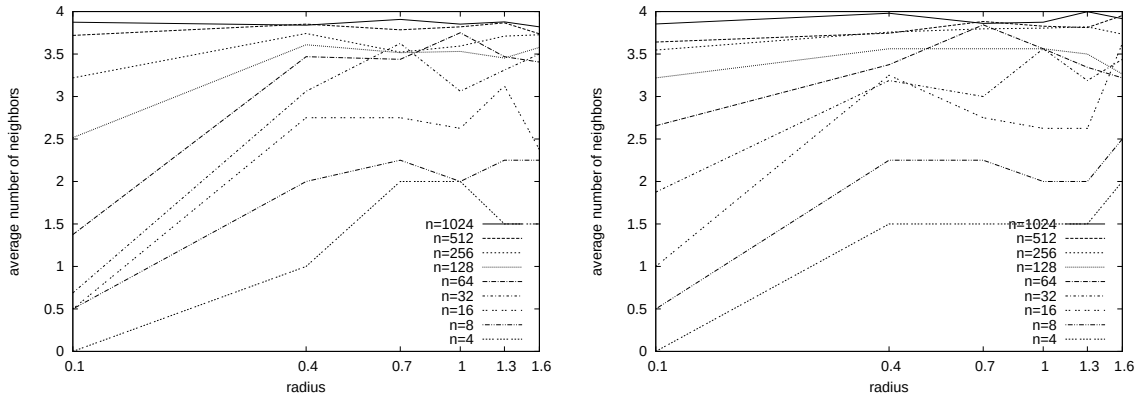
(b) Exponential distribution

Figure 4.6: *Average Node Degree for the Delaunay Triangulation:* as the radius of the AOI and node density increases, the average number of neighbors approaches the theoretical average of 6 neighbors per node. The average neighbor count is a measure of how many times a packet would need to be replicated to reach its listeners. Both exponential and uniform distributions showed similar results, indicating the efficacy of using Delaunay triangulations.

low bandwidth requirement for neighbor maintenance.

Our Gabriel graph protocol shows similar patterns (see Figure 4.7). The results show that the average number of Gabriel neighbors, ranges from 0 to 4. Note that even at its most loaded setup, where the radius of the AOI was 1.6, each node had on average only 4 Gabriel neighbors. *In other words, as the population density increases, the average node degree, and therefore bandwidth requirements, increase very slowly.*

To further understand our results we examined the generated data in detail. We are particularly interested in the worse-case scenarios, so we used only the exponentially-distributed mobility model. Figure 4.8 presents the detailed results of the 0.4 radius simulation run for both the Delaunay and Gabriel graphs with 16, 64, 256 and 1024 nodes respectively. We plotted the minimum, the average and the maximum number of neighbors that a node had to maintain. Note that in this stacked chart, the difference between the minimum, average, and maximum values is presented so that



(a) Uniform distribution

(b) Exponential distribution

Figure 4.7: *Average Node Degree for the Gabriel Graph:* as the radius of the AOI and the node density increases, the bandwidth requirements for our protocol increases. Our results show that the average neighbor count approaches its theoretical maximum of 4, indicating that the Gabriel graph scales well because fewer packets would be replicated over multiple unicast streams. As with Delaunay triangulations, Gabriel graphs were effective in both uniform and exponentially distributed populations.

the height of each bar represents its value on the y-axis correctly.

Figure 4.8 shows that the maximum number of neighbors is never more than twice the average number of neighbors, while the minimum number of neighbors can be quite small because some nodes are isolated. In addition, we see that the maximum Gabriel node degree is consistently smaller than the maximum Delaunay node degree. While we only show the values for the radius of 0.4, we had similar results for the other radii and node densities.

As described in Section 4.2.2, voice packets from v_i are transmitted to all AOI neighbors since they should be able to hear the avatar speaking. However, since we have built a Gabriel graph between every node and its AOI neighbors, the transmission is done in a hop-by-hop fashion. This design not only helps balance the traffic, but generates a realistic scenario where listeners closer to the speaker hear the voice packets before farther listeners.

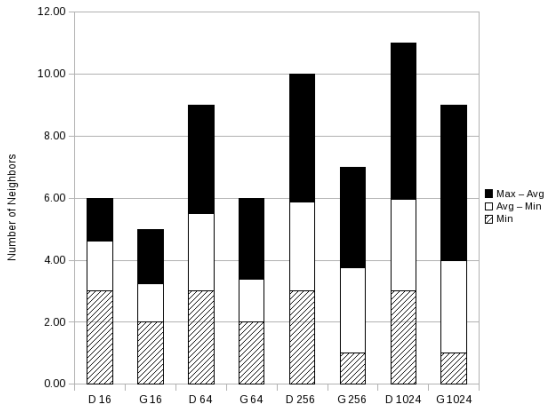
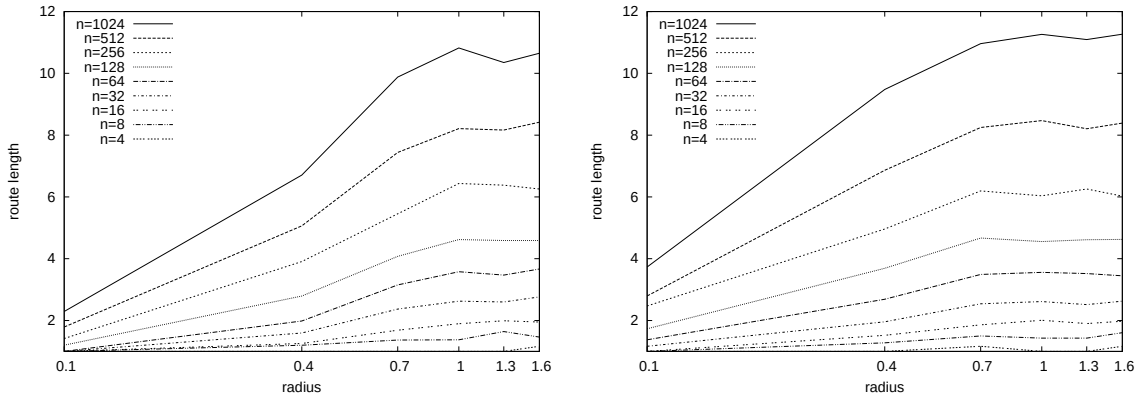


Figure 4.8: *Minimum, Average, and Maximum Neighbors:* This figure illustrates the range of neighbors for a radius of 0.4 and compares both Delaunay and Gabriel graphs of increasing density (D16 is a Delaunay graph with 16 nodes while G16 is a Gabriel graph with 16 nodes). The results illustrate that the average, minimum, and maximum node degrees of the Gabriel graph are consistently smaller than the maximum Delaunay graph.

Figures 4.9 and 4.10 show the average number of hops for a packet to get from its source to a node in its AOI using the Delaunay and Gabriel graphs, respectively. In the worst case, the average number of hops is approximately 11 for Delaunay triangulations and 17 for our Gabriel graph protocol. However, this setup illustrates an extreme case where all 1024 nodes are within each others' hearing range and this value is an order of magnitude larger than a zone contains in World of Warcraft [34], for example. In a more realistic scenario where the radius of the AOI is 0.4 the average number of hops is only 9 and 11 for Delaunay and Gabriel graphs respectively.

On the other hand, even in this extreme case, the bandwidth required by each node would still be low, though the delay would be high for far away listeners due to the number of hops to forward packets to them. It can also be observed that the lower number of Gabriel neighbors results in a longer route length versus routes in the Delaunay graph. However, the Gabriel graph is a closer approximation of reality in that closer listeners to a given speaker in the virtual world will receive



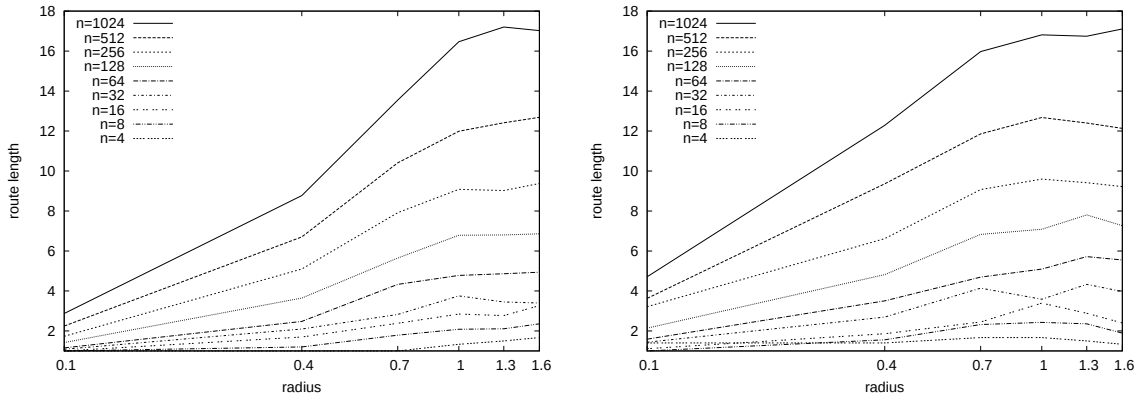
(a) Uniform distribution

(b) Exponential distribution

Figure 4.9: *Average Route Length in the Delaunay Graph:* this figure illustrates that the route length from the source to its destinations increases as the radius and node density increases. In particular, the Delaunay graph increases its route length logarithmically as the density and radius of the AOI increases.

their voice packets before those that are farther, more accurately modeling the way that sound travels in the real world. Further, since the bandwidth required by voice communication will compete with bandwidth for the rest of the NVE, the Gabriel graph has the advantage of fewer neighbors to replicate packets to.

Again, we examined our data in detail to determine if in the worst case scenario the route length was significantly higher than it was on average. We present the minimum, the average and maximum route length for both the Delaunay and the Gabriel graphs. For this simulation we used 16, 64, 256 and 1024 nodes, our second mobility model, and we always used a radius that has a length 0.4 (see Figure 4.11). We found that the route length in the worst case scenario is only about 50% more than it is on average. Thus, our solution is well balanced and does not overload any of the nodes.



(a) Uniform distribution

(b) Exponential distribution

Figure 4.10: *Average Route Length in the Gabriel Graph:* This figure illustrates the route length, in hops, from a speaking avatar to any other avatars within the speaker's AOI. In high density situations, such as when the radius is 1 (and encompasses almost the entire playing field), the number of hops appears to grow logarithmically with the number of nodes in the AOI.

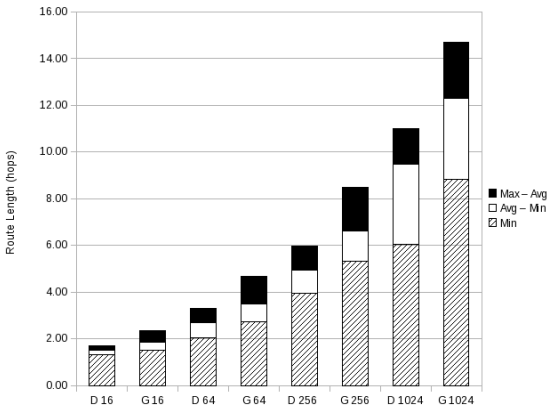


Figure 4.11: *Minimum, Average, and Maximum Route Lengths:* This figure illustrates the range of route lengths for a radius of 0.4 in our simulations and compares both Delaunay and Gabriel graphs of increasing density. The results illustrate that the minimum, average, and maximum route length of the Gabriel graph is longer than that of the Delaunay graph.

4.4 Adding Social Structures

Although our protocol focuses on building and maintaining a graph for location aware voice communication, it can be easily extended to maintain additional connections between the players to accommodate social structures such as *guilds* and *friends lists*. In these cases, a player desires to communicate with other players in the social structures which are far away in the virtual space, but using the Gabriel graph as we have designed could cause significant delay due to the number of hops it would need to take.

To handle these social structures, we propose extending the protocol to add additional edges to the graph so that players are directly connected to the other players in their social group. Fortunately, the extra edges do not effect the routing mechanism because they do not result in dead ends, or local maximums, and our original method guarantees that there is a route between any two nodes that are a part of the Gabriel graph. The drawback of this approach is the additional number of edges that a peer must support. Therefore, an alternative is to construct additional graphs for social structures such that only those members of the social structure are connected in the graph and the AOI of each member is sufficiently large to cover the entire membership. Given these parameters, a Gabriel graph will be constructed between members of the social structure allowing efficient communication between them.

4.5 Security

In this section we briefly overview the different kinds of attacks. Next, we identify the possible attacks with which our network would have to cope. After analyzing each of the attacks, we present a possible solution.

Table 4.1: *Attacks in NVEs:* Categorized as active or passive, we list common, well-known attacks against P2P networks for interactive systems and possible defenses against the attacks.

Active Attacks	
Attack	**Defence**
Denial of Service (DoS) Attack.	Captchas and challenges.
Black Hole Attack.	Periodic message update between the node and its neighbors.
Incorrect Forwarding.	Hiding identity.
Passive Attacks	
Attack	**Defence**
Eavesdropping.	Encryption with symmetrical keys.
Traffic Analyzing.	Hiding identity and encrypting communication channels.

4.5.1 Categories of Attacks

One possible way to categorize attacks in a computer network is based on their nature (Table 4.1). If the attacker cannot change anything in the system, i.e. she is not able to interact with any of the parties involved or modify any of the messages in the system, we call it a passive attack. This kind of attack is solely based on observing the data. The most common attacks in this category are traffic analysis and eavesdropping.

Active attacks require the attacker to be able to transmit data to one or more of the parties, or modify the data stream between them. Alternatively, the attacker can simply drop the data packets without making any changes to them.

4.5.2 Active Attacks

DoS Attacks

We would like to have as few malicious nodes in our network as possible. To achieve this, we have to make sure the nodes that enter the system during the joining process are not malicious. Since our system is purely peer-to-peer, we cannot rely on a centralized authority that controls access to the network. Thus, the validation process that decides whether a node is malicious or not has to be carried out using the nodes that are already inside the system.

Our assumption is that every node that is not controlled by a human is malicious. To be able to distinguish between humans and computer agents, the system has to provide a challenge. We propose using captchas to slow down the joining process. To further slow down the process, challenges could be used. Generating challenges that are easily producible but are hard to solve is not trivial. Network puzzles fit our criteria as they are easy to produce, hard to solve and fast to validate [12, 13].

Once the puzzle is generated, sent to the client and solved by it, the puzzle is validated by nodes nearby the joining node along with randomly chosen nodes from other locations. After the joining node is admitted into the system, a session key is provided to the candidate and stored in the network via a technique such as Shamir's solution [39].

Black Hole Attack

The most basic attack is the *black hole attack*, where a node tries to erase messages in the network typically by dropping packets which should be forwarded. The attacker's goal is to increase the number of messages that she has to forward. She does this by convincing as many nodes as possible that they should route messages through her.

If routing tables are based on gossiping, she can advertise herself as a node which is a direct neighbor of all the other nodes in the system.

To address this attack, we propose using periodic updates between neighbors *and* with nodes two or more hops away in order to detect inconsistencies. Note that this method can only find out if there is an incorrect entry in the system, and does not locate the malicious node itself. The latter can be done with voting.

Incorrect Forwarding

The *incorrect forwarding attack*, where a malicious node purposely routes messages incorrectly, is harder to detect than the black hole attack since we would have to decide if the packet took the correct route. In addition, if the malicious nodes cooperate, the packet might still reach its destination but with a higher delay.

A possible solution to the problem is to hide the identity of the nodes. Since interactive media traffic is often used for multiplayer games and virtual reality systems, the main reason for an attack is cheating, which requires the attacker to be able to identify the sender of a packet. Therefore, hiding the identity is a sufficient defense.

We propose routing messages via an approximate virtual location and not including the source identifier in the messages. In this scenario, routing between areas in the virtual space can be implemented accurately, but malicious nodes would not be able to accurately determine the source of the packets. The drawback is that more nodes would receive the packets due to the higher radius of distribution required by approximating the locations.

4.5.3 Passive Attacks

Eavesdropping

The most common passive attack in a virtual system is eavesdropping. The attacker might use the data for cheating or data mining. To prevent this from happening, we have to make sure that no unathorized party either inside or outside of the system is able to listen to conversations. A possible solution for this problem is to encrypt the data flows with symmetrical keys that can be set up between the nodes during the neighbor maintenance process.

Traffic Analysis

While analyzing the traffic the attacker can gain insight with either identifying the senders, receivers, the content of the data flow or the amount of it. Since we use an encryption we hide the identity of the nodes and we do not send to a particular receiver but to a region, this problem does not affect our users.

4.5.4 Putting Things Together

When a user wishes to join the system, she has to bootstrap into the network and locate an initial node to initiate the joining process with. First, the bootstrap node sends out a captcha. If the user answers the captcha correctly, a challenge is generated by the system. This challenge then has to be solved by the user's computer and the result has to be sent back to the system. The verification of the result is conducted by multiple nodes in the system. Most of the verifying nodes are direct or close neighbors of the node that handles the joining, but some of them are simply randomly chosen. If the verification is successful, the user becomes a part of the system. However, to protect the user from both active and passive attacks, a secure

communication channel has to be established between the node and its neighbors. During this process a symmetrical key is generated that is later used for communication. Since the neighbor maintenance is based on the delaunay triangulation, a new key pair has to be generated after every neighbor change. If the node initiates any traffic, the packets are signed by the location of one of its neighbors. This way, the forwarders cannot identify the original sender.

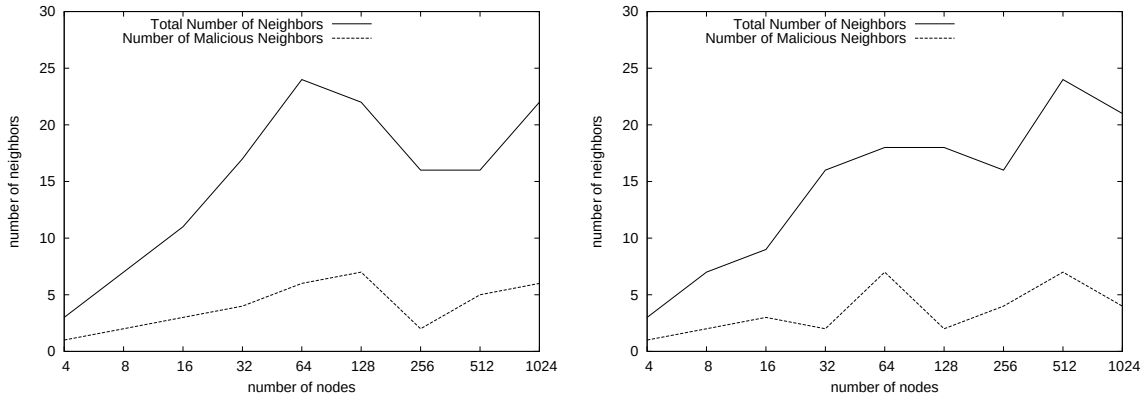
From the user's point of view, the process is almost fully transparent. If a captcha is required, that has to be presented to the user and solved by her. Also, a slight delay is experienced while the computer answers the challenge. Other than these two steps, everything is done in the background and therefore should be completely hidden from the user.

4.6 Security Simulation

In this section we evaluate our protocol by simulation. Initially the nodes in our network are distributed in a 1×1 square. We run each of the simulations for 60 seconds based on two different mobility models. In our simulations we sample the network every 100 milliseconds, for a total of 600 times during the simulation, which is representative of the rate of real-time multimedia traffic. We simulate from 4 to 1024 total nodes.

4.6.1 Puzzle Validation

When a node joins the system, it is a potential threat for all the other nodes. In order to ensure that the nodes that become a part of the network are not malicious, an authentication has to take place. Since the system does not have any previous information about the nodes the only thing that can be validated is if the joining



(a) Uniform distribution

(b) Exponential distribution

Figure 4.12: *Average Ratio of Malicious Neighbors:* The number of neighbors and the number of malicious neighbors indicate the ratio of malicious neighbors. These results show that the ratio of malicious neighbors is independent from the distribution of the nodes in the system and correlates with the ratio of the total number malicious nodes.

node is welcome by the majority or not. Here we present our simulation results about this validation process.

The joining node is challenged to solve a puzzle. If the node is willing to take the time that is required to find the solution, the nodes that are already inside of the system mark the candidate as trusted and let it join the system. To determine how effective this method is, we ran simulations with both the uniform and exponential mobility models and varied the location of the malicious nodes.

Figure 4.12 shows the results of the run, where 25% of the nodes were malicious and were placed randomly in the system and the closest and second closest neighbors were used to validate the secret of the candidate node. The X-axis shows the total number of nodes in the system and the Y-axis shows the number of validating and malicious nodes. Without collusion, we expected that malicious nodes would be unable to easily admit additional malicious nodes into the system–which these results demonstrate. Thus, after the runs, we still saw an approximately 25% ratio of malicious to regular

nodes.

We next investigated the case where malicious nodes clustered to admit new nodes into the system without the proper puzzle solving. In this scenario, we expected malicious nodes to be capable of dominating the system by admitting other malicious nodes into the system over time.

Figure 4.13 is the result of our clustered simulation using the uniform mobility model. Since the network is dynamic and uses the delaunay triangulation for neighbor maintenance, it is almost impossible to tell where a certain node would have to position itself to ensure that it will be a part of the group that validates the solution of the candidate. In the optimal case, all the malicious nodes should cooperate. After trying several scenarios, we found that simply gathering around the node that is conducting the joining process is sufficient enough to admit more malicious nodes into the system. For example, with at least 128 nodes in the system, almost all of the neighbors are malicious (hence the two graph lines join). With fewer than 128 nodes, the ratio of malicious nodes in the system is fewer than the average number of neighbors that a node has, and hence the joining node cannot be surrounded by malicious neighbors. Thus, if the malicious nodes conduct an organized attack against the system, a simple majority based decision fails. Note that, we omit the exponential distribution mobility model results since it performed similarly.

As we explained before, there is a possible solution for this problem. If the decision is not based on the vote of the neighbors but some randomly chosen nodes, the malicious nodes cannot take advantage of their location. The drawback of this solution is the increased overhead. In Figure 4.14, we show the ratio of malicious nodes with and without the random selection of verifying nodes. These results show that without random selection, malicious nodes can dominate the network with as few as 128 nodes in the system. However, when we add random selection, the ratio of malicious nodes

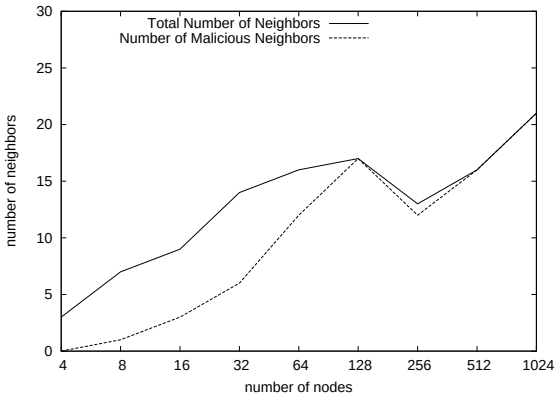


Figure 4.13: *Average Ratio of Malicious Neighbors While Clustering:* The ratio of malicious neighbors increases over time when they collude, eventually overwhelming the system with as few as 128 nodes in the system.

stays constant throughout the set of simulations.

In choosing random verification nodes, we queried the same number of random nodes that were in the 2-hop neighborhood of a node. However, this technique would result in a larger overhead, since the route length to the randomly chosen nodes is longer on average than it is to the 2-hop neighbors. To investigate the number of nodes that we could randomly query without increasing the overhead, we ran multiple simulations. After analyzing these simulations, we concluded that choosing 5 random nodes in the system results in slightly higher message forwarding then querying the 2-hop neighbors. Thus, we decided to query 4 random nodes. Figure 4.15 shows the overhead generated by the above mentioned two scenarios.

Although a possible solution would be to replace our solution with one that is completely random it would result in a much less accurate scenario. Assume that 25% of the nodes are malicious and we use only 4 nodes to validate the node that tries to enter the system. Furthermore, assume that we require at least 3 out of the 4 nodes to authenticate the candidate. In this case only one node can be non-malicious. However, even in this case there is a 5% chance of that a malicious node would get

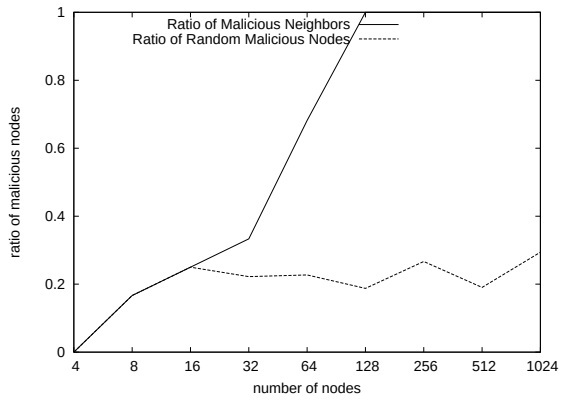


Figure 4.14: *Neighbor Validation vs. Random Validation:* The ratio of malicious nodes among the neighbors is clearly higher when the malicious nodes cooperate, whereas the ratio of randomly chosen nodes correlates to the ratio of malicious nodes in the entire system, indicating that randomly choosing verification nodes alleviates collusion in the system.

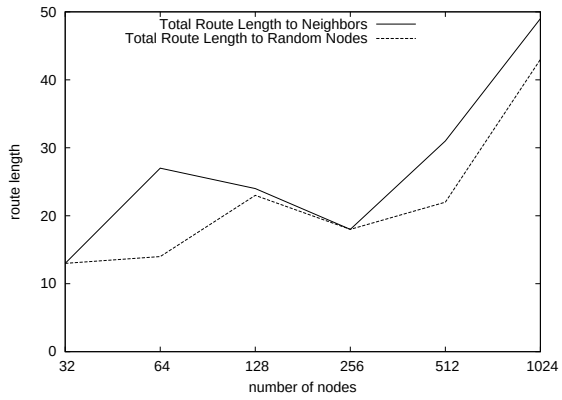


Figure 4.15: *Total Route Length:* The total route length is the sum of all the route lengths that the messages travel. In case of the close neighbors, the routes are shorter but the number of nodes is higher, whereas in case of the randomly chosen nodes we only have 4 nodes, but they can be located anywhere in the system.

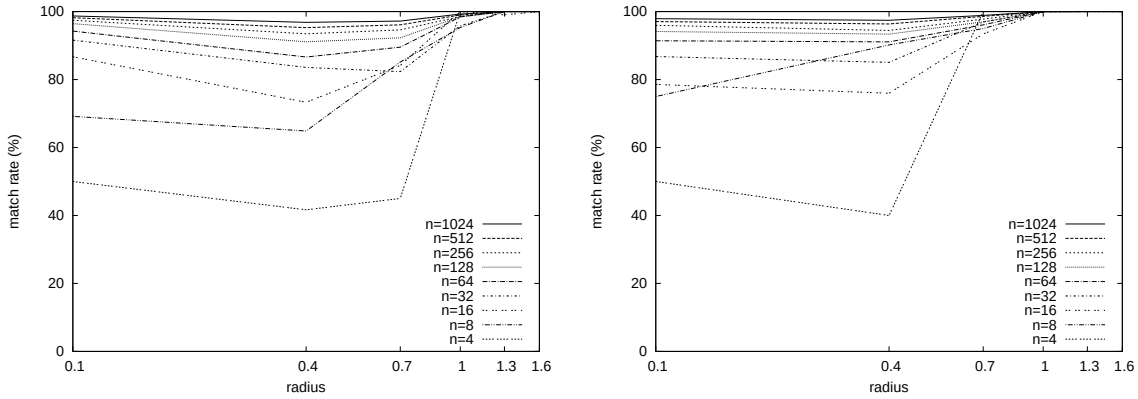
permission to enter the system.

Instead, we propose a hybrid scenario. Since the nodes that enter the system enter it close to the node that gives them permission, we can identify the malicious nodes. We query both the 2-hop neighbors and 4 random nodes. If these two groups make the same decision (i.e. they both agree or disagree about giving permission to the candidate), we consider it a final decision. However, if there is a disagreement between these two groups, we repeat the admission process, but only with the randomly chosen nodes. If their decision is unchanged we mark those nodes that voted against this decision. If a node is marked more than a certain threshold, we eliminate it. This way the number of malicious nodes in the system can be lowered throughout time.

4.6.2 Hiding Identity

In this experiment we measure the accuracy of our solution for hiding the identity of the nodes (Figure 4.16). Recall that to hide the identity, we use a slightly different location than the actual location of the sender node. As a side effect, we deliver packets to nodes who are outside of the original area of interest, and ignore some nodes that are inside. To determine the accuracy, we run simulations with different node densities and Area of Interest (AoI) radii. On the Y axis we present the accuracy of the method in percentages. The number shows what ratio of those nodes that are inside the original AoI, actually receive the packet. The X axis is the radius of the area of interest. Since we use a 1x1 square for our simulations, the 1.6 radius guarantees a total coverage.

Instead of using the location of the sender node, we use the locations of one of its neighbors. As our results show, when we have very few nodes in the system, the accuracy of the protocol is only around 50%. This is due to the fact that the density of nodes is fairly low and therefore using the coordinates of a neighbor node shifts the



(a) Uniform distribution

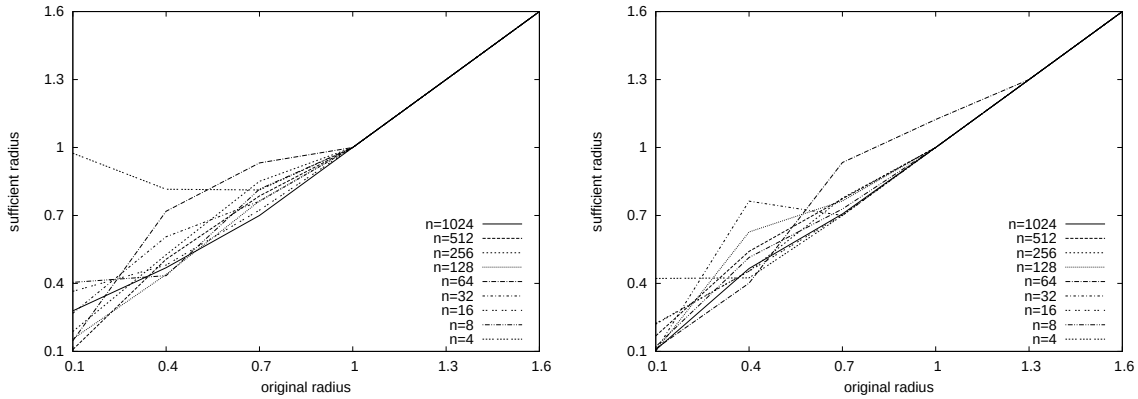
(b) Exponential distribution

Figure 4.16: *Average Match Rate While Hiding Identity:* Using the location of a neighbor instead of the real location results in including unwanted and excluding wanted nodes. As the results point out, there is no real difference between the two mobility models and the accuracy is already acceptable if there are at least 64 nodes in the system.

center of the AoI significantly. As the node density increases, the results get much better. With having only 64 nodes in the system, the accuracy is already around 90%.

We reran the simulations with our exponential mobility model (Figure 4.16). The results are similar to the results that we saw with the uniform model. This is due to the characteristics of the random waypoint mobility model as it tends to cover the extreme cases even when a uniform distribution is used and as such, using a different distribution does not affect the results. However, running the simulation with two different distributions ensured us about the robustness of our solution.

Next, we ran simulations to determine the size of the radius that would be sufficient enough to cover all the nodes that are inside of the original AoI. As we use a slightly modified location, not the exact location of the sender node, we might leave out nodes that are officially inside of the AoI. Therefore, the radius that guarantees that all these nodes are covered, cannot be shorter than the original radius.



(a) Uniform distribution

(b) Exponential distribution

Figure 4.17: *Sufficient Radius for Delivery Guarantee:* With the increase of the size of the radius of the AoI, it can be guaranteed that everyone who has to receive the packet will actually receive it. It can be seen that with the increase of the number of nodes the delta between the original and required radii decreases. Having 128 nodes results in less then 10% difference.

Figure 4.17 shows our results. As it can be seen, if the radius is at least 1 unit long, our system does not show any inaccuracies. If the radius is shorter than that, our solution results only in slight inaccuracies but it can be concluded that it still does better as the node density increases. If we have at least 32 nodes in the system, the radius that guarantees 100% coverage is only 50% longer than the original radius in worst case.

4.7 Conclusion

We have presented a new protocol for multiparty voice communication based on Gabriel graphs and positional information. Our protocol has five interesting properties: 1) positional information allows the voice packets to be mixed into a 2 or 3 dimensional space accurately; 2) voice packets are sent to the closest listeners first, since we have a direct link with them, who then forward them to farther listeners; 3) the average degree of any node in the system is smaller than 6 because Gabriel

graphs are a subset of Delaunay triangulations; 4) the average number of hops in the system also appears to remain low, but depends on the density of players (though high density areas will mainly cause delayed voice data and not overwhelming voice data); and 5) the protocol can be used in both distributed and client/server NVEs.

When forwarding packets in an AOI-limited broadcast, our results show that they would need to be replicated fewer times when using the Gabriel Graph. Next, we have presented a framework for securing our virtual-location-aware peer-to-peer voice communication protocol. Our framework shows how to handle passive and active attacks on the network from malicious nodes joining the system. In addition, we show that by using a combination of random verification and hiding the virtual source locations of the packets in the network, we can prevent malicious nodes from rapidly joining the system and actively attacking the network by dropping or incorrectly routing packets.

Chapter 5

Implementation

5.1 Introduction

In this chapter we describe the details of our implementation and the results of the real-world simulations that we run to validate both our protocol and the conclusions of our measurement study.

5.2 Program Architecture

The architecture of our protocol contains two building blocks. One of them is the bootstrap server. The bootstrap server is not only used to provide information to the nodes that try to join the network, but it also provides location information to the current members. This way, the simulations can run without having actual players. The other building block is the servlet. This is the program that is run by all the nodes and serves both as a client and a server. It is responsible for maintaining the connection with all the other players' servlets as well as to provide a graphical interface to the user and transfer her/his voice packets.

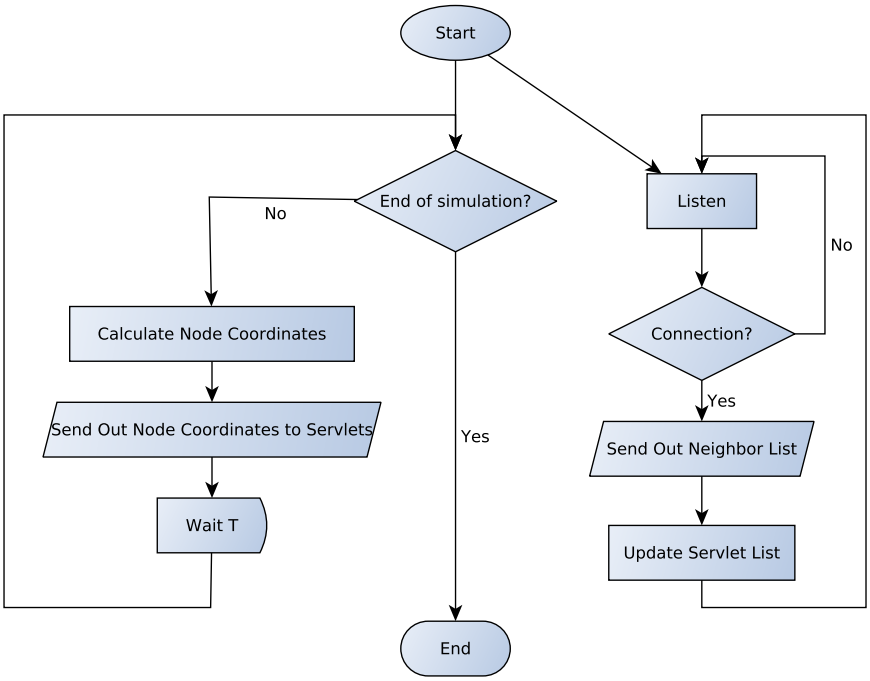


Figure 5.1: *Flowchart of the Bootstrap Server:* The bootstrap server maintains two threads concurrently. One calculates the node locations while the other maintains the servlet list and provides information for servlets that want to join the network.

5.2.1 The Bootstrap Server

The flowchart of the bootstrap server can be seen in Figure 5.1. Although, this only contains the main methods of the actual program, we thoroughly discuss the details in this section.

The bootstrap server initiates two threads upon startup. One of the threads is used to listen to requests from the servlets and updates the servlet list accordingly. The other thread is used only for simulation purposes and would not be required in real-world deployment. We chose this solution for protocol validation due to the lack of available players.

The Location Thread

Upon startup the server requires a parameter that specifies the length of the simulation. This parameter is given in rounds, where a round is 100 ms. We chose this parameter because this is equivalent to the length of a voice packet in a typical voice communication software. Although we ran multiple simulations with 100 ms long voice packets, we experimented with other values as well. The thread is responsible for checking whether there are any more rounds left. If there are not, the simulation ends.

If the server happens to pass the "End of Simulation?" stage, it calculates the node coordinates. In the first run the nodes are simply placed in the virtual reality based upon some command line specified parameters. Next, the node locations are calculated based on the node locations in the previous run. Once the calculation has finished, the server sends out the information to the servlets. The servlets are then responsible for updating their current location, and to discovering their neighbors based on this information.

Before entering the decision making stage again, a delay is incorporated. The total delay is the amount of time that the calculation and the artificial delay take together. However, the time that the coordinate calculation takes is minimal; therefore does not influence this step significantly. Moreover, this step does not have to be synchronized to the voice packet sending interval, as it would not be if the avatars were player controlled.

Out of the two threads that the server runs, the location thread is the only one that can terminate the program. Although we did not mark any connections between the two threads that the server runs, in case of a termination both of them are signaled, which allows them to close open sockets and to free the allocated memory segments.

The Servlet List Thread

The servlet list thread is an infinite loop that is used to do two things: first, it provides information to servlets that want to join the network. Second, it maintains a list of servlets that are in the network.

The server binds to a given port, and then listens to requests from servlets. It remains in the listening stage as long as it does not receive a request. If a request arrives, it checks whether it is a request from a servlet. To do this we use a particular request ID that serves to identify malicious activities. If the request ID is a match, a connection is built between the servlet and the server, and the request is processed.

Since we use the server to calculate the virtual location of the players, the server provides a neighbor list to the servlet. Under normal conditions this would be impossible; the server would have to send out some of the addresses of the servlets in its servlet list instead. Since we are more interested in the behavior of the protocol in the stable stage, we did not implement this feature. However, the constant updating of the location of the nodes in the virtual world can ensure us that the neighbor maintenance protocol is correct, which would be used during the joining stage as well.

The last step of the loop is the servlet list maintenance. Since we know from the beginning how many servlets we have during the actual run, we simply maintain an array with the node ID and the corresponding status of the node: online or offline. Under normal circumstances, the servlet list would have to be maintained as a linked list. The servlet that joined last should be added to the end of the list, and the last k servlets should be pulled from the list as an answer to a query. However, this solution introduces a potential security loophole. A DoS attack against the server could easily be conducted. A malicious machine could register several non-existing nodes in the list, making it practically useless. To avoid this, the joining process has

to be a two-way communication between the server and the servlets. The joining node is not registered until it actually enters the system. To ensure this, the node is registered by the node that is already in the system and initiates the joining process of the candidate. Moreover, the registration only happens if the joining is successful. Another potential security problem is that the servlet list ages and could contain nodes that are no longer in the system. To avoid this problem, after a successful join, the node can report the nodes that it could not find in the system while trying to join. The server then replaces these nodes in the linked list at a lower position. The result is a more up-to-date list that does not delete entries accidentally.

5.2.2 The Servlet

The flowchart of a servlet can be seen in Figure 5.2. Similar to the flowchart of the bootstrap server, this contains only the main methods of the actual program.

After a servlet starts, it logs into the bootstrap server. During this step, it first sends a request to acquire the list of other servlets that are already in the system. Next, it enters the waiting stage. It remains in this step until it receives a reply or the waiting times out. If a reply is received, the servlet checks it to make sure it is a valid reply. If the reply is not valid, the program terminates. Otherwise it joins the network. This step is rather complicated. The candidate servlet has to be in continuous communication with the other servlets that are already in the system. During the joining, the servlet has to find its actual neighbors before it can start sending packets. It also has to inform its neighbors so that they can register the candidate node in their neighbor list. It is only after this step that the servlet is considered a fully-functional part of the network.

Next, a servlet initiates three threads. Unlike the bootstrap server, all three of these threads are loops, and none of them can terminate the program. Under normal

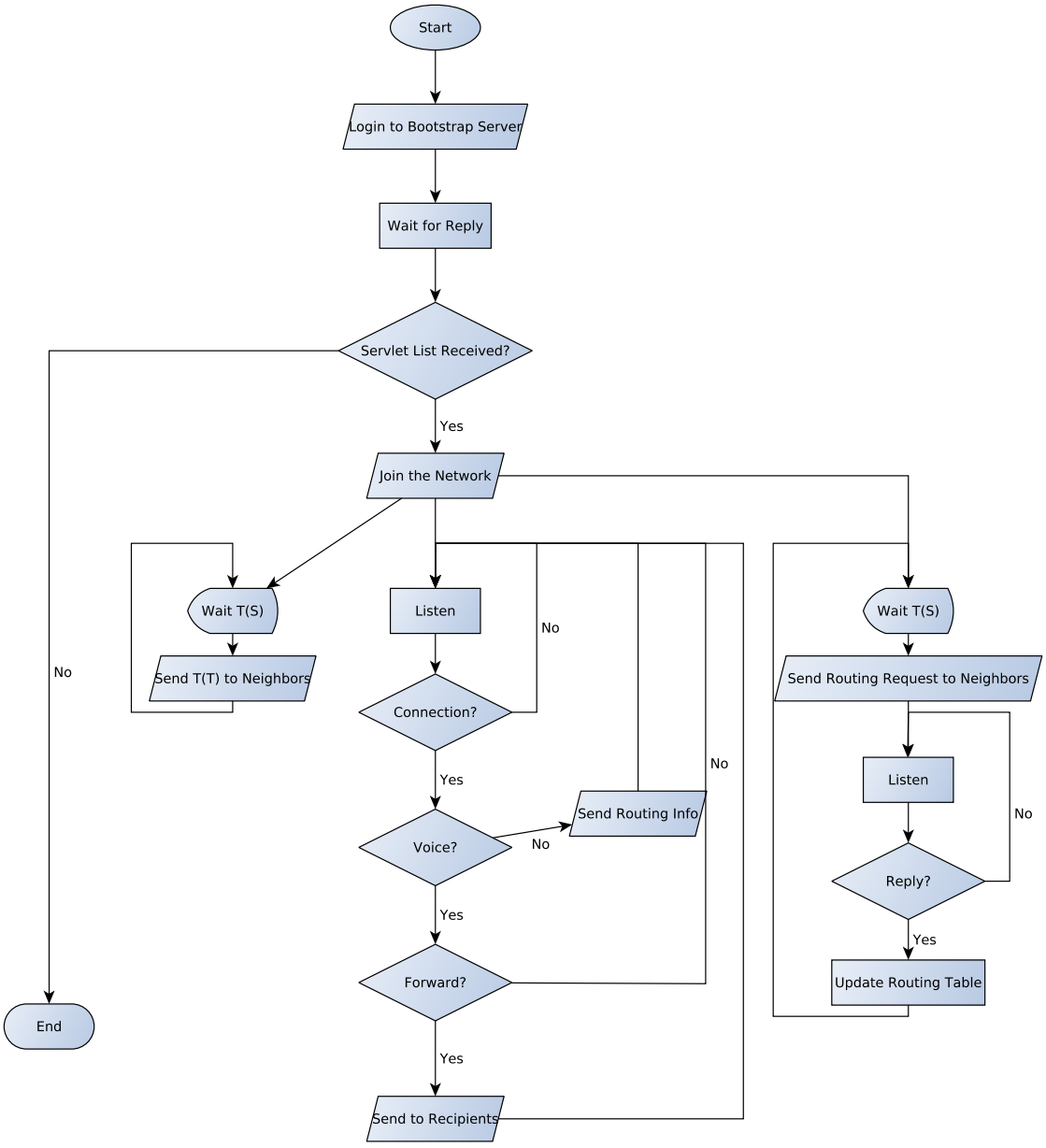


Figure 5.2: *Flowchart of a Servlet:* A servlet maintains three threads concurrently. One is responsible for the neighbor maintenance, while the other two handle the user generated and the incoming voice packets.

circumstances, in which a user controls the avatar, a disconnect option would be available; for the validation process this is not necessary.

One of these three threads is responsible for the neighbor maintenance, while the other two threads are responsible for voice packet handling: one handles the incoming, while the other handles the outgoing voice packets.

The Neighbor Maintenance Thread

The neighbor maintenance is performed by exchanging routing messages with the direct neighbors, but the routing messages might contain information about non-direct neighbors as well. The direct neighbors might change, thus the list of direct neighbors is constantly updated.

After a node successfully joins the network, it performs a periodic neighbor update. The time between these updates is determined based on the node density in the virtual world and the speed of the nodes. Higher density, as well as faster movements, require more frequent updates.

The first step of this process is to send out the routing requests. Next, the servlet enters the listening stage where it waits for the replies. If it receives a reply, it has to check it to make sure the reply is a valid routing table entry. If it is, it updates the corresponding entries in its own routing table. Otherwise it goes back to listening mode and waits for a valid reply.

Since normally a node has 6 neighbors on average, during the routing update multiple replies can be received (we did not indicate this in our flowchart). Thus, the loop does not terminate after receiving only one reply, but rather after a certain threshold, and collects all the replies during this time.

The Outgoing Voice Packet Thread

The outgoing voice packet thread is quite different from what it would be if a user controlled the avatar. In that case we would have to code the user's voice into digital voice packets and send them out. Instead, here we have to simulate this behavior.

Using our voice model, we can randomly generate the silence periods and the talkspurts. After every silence period comes a talkspurt, and after every talkspurt comes a silence period. Thus, the loop has only two stages.

During the first stage the silence period is generated, which is essentially equivalent to a delay. In this stage the loop is idle. Next, it enters the sending stage. Here another time is calculated as the talkspurt time and voice packets are sent out to all the neighbors during the talkspurt stage.

The Incoming Voice Packet Thread

Although we use the incoming voice packet thread to handle both incoming voice packets and routing requests, we named it after its main functionality. In order to make the setup more intuitive to the users, we decided to open only one port for incoming messages and distinguish between them with a packet ID, instead of opening multiple ports for the different functionalities.

If the servlet receives an incoming packet, it first validates that the packet falls into one of the two categories mentioned above. If the packet happens to be a routing table request, a reply which contains the relevant routing information is sent out and the servlet returns to the listening stage. If the incoming packet is a voice packet, the servlet has to decide whether this is the final destination of the packet or if it has to be forwarded to other nodes that are inside of the originator's AoI. This decision is made based on the originator's location that is coded in the packet. Please note

that the voice packet might be forwarded to multiple recipients, who will also have to analyze and possibly forward it.

5.3 Testbed

In this section we desribe how we test the functionality of our program. We focus on two aspects: the neighbor maintenance and the voice packet delivery. The neighbor maintenance is an essential part of the protocol; without it the voice packet delivery is not possible. Therefore, we first test the neighbor maintenance and then we test the voice packet delivery.

5.3.1 General setup

During our validation we use two different scenarios: a static scenario and a dynamic scenario. This helps us validate the different components separately as well as together. Table 5.1 shows the validation matrix. We performed the steps from left to right and from top to bottom. Sections 5.3.2 and 5.3.3 explain the details of each of these steps.

Table 5.1: *Protocol Validation*: The four different steps of validation.

	Static	**Dynamic**
Neighbor maintenance	Validates the initial joining process mechanism.	Validates the neighbor maintenance mechanism.
Vocie Packet Delivery	Validates the voice packet delivery mechanism.	Validates both the neighbor maintenance and the voice packet delivery. ⇒ **Validates the protocol.**

Static Scenario

During our static validation, we do not move the nodes. We simply set up groups with different sizes based on our measurement (See Section 3.3.4). This step helps us validate the basic functions of our protocol and debug any unexpected problems.

Dynamic Scenario

The dynamic scenario was designed to validate our protocol under similar conditions that we would have in a real-world deployment. Here we move the nodes similar to how we did during our simulations. The main difference is that we set up groups that move together. We experimented with using a designated leader in the group that every other node would follow, but this resulted in forming a line by the other players that were following the leader. Therefore, we decided to move the players separately but towards the same target. Unfortunately, after a group reached its first destination we faced a similar problem as before. In the end we used a mobility model in which we chose a destination for the group, and then we added random x and y values to this destination for each of the members of the group. We set the x and y parameters as a uniformly chosen random number between 0 and 10% of the radius of the AoI. We set a new destination for a group when all the members of the group reached its target. This indicates that our group stayed together but this was not always the case. The speed parameter sometimes resulted in a big enough difference that some of the members of the group could not keep up with the rest, or moved fast enough to get out of the AoI of some members of the group. Also, since we had multiple groups in the virtual world moving at the same time, they crossed each others' route on several occasions. This resulted in voice transfer among groups, similar to how it would under real-world use.

5.3.2 Neighbor maintenance

The neighbor maintenance is an essential part of our protocol, and without it the voice packet delivery would not be possible. Using both a static and a dynamic scenario helps us to validate this functionality.

Although the static test cannot be monitored for a long period of time, it is important to verify that it is correct. During this step the candidate joins the network and essentially sets up the connections with its neighbors. After both the neighbor discovery and registration are complete, there should not be any more changes. The static test not only verifies that the routing table of the candidate node is correct, but it also verifies that the neighbors adjust their routing table: they include the new node and delete those old nodes that are no longer their neighbors.

Unlike the static scenario, during the dynamic scenario a continuous change is possible. Therefore, here we cannot set a breakpoint after which we can state that the algorithm works correctly. Rather, the longer the validation is, the more probable the neighbor maintenance is bug free. Again, the correct functionality of this mechanism of the implementation is extremely important, since it is a requirement of the voice delivery. During this validation, we monitored whether the routing table of each node of the network matches the topology of the network. Since we use the server to tell the nodes their destination, we can compare the logs of the server to the logs of the individual nodes. To do this, we maintain two different log files: one for the server and one that is shared by the nodes. The server log file contains the coordinates of each of the nodes every 100 ms and the shared log file of the nodes contains the coordinates of the logs as well as their routing table. By using the server log file, the desired links can be calculated after the run which helps us save resources. After the desired links are calculated, they can be compared to the routing tables.Although

the optimal result would be a perfect match, we encountered slight differences due to the fact that a change in the system might require a few rounds to spread across the nodes.

5.3.3 Voice Packet Delivery

Once our system was validated for neighbor maintenance, we looked at the voice packet delivery. Similar to our previous validations, we started with the static scenario and then continued with the dynamic one.

The static scenario helped us make sure the voice packets were delivered correctly. First, we had to make sure the nodes generated the talkspurts and silence periods correctly. To be able to validate this, we maintained a detailed log file for the nodes. A big advantage of running all the nodes on the same machine, is that clocks do not have to be synchronized. In the log files we included the routing table entries and a delivery time and destination for all the packets. The next step that we performed was the validation of the receiving process. We wanted to make sure that all the nodes received all the packets and that there was no rejection in the system. The last step for which we used the static scenario was the validation of the forwarding process. We checked whether all the sent packets reached all the nodes in the AoI of the sender, and if the route was the desired route.

The dynamic voice packet delivery essentially combines all the previous steps, and therefore provides a detailed picture of the implementation: it not only validates the functionality of the program, but it also provides some performance numbers that we describe in detail in the following sections.

5.4 Program Validation

In this section we present results that we got when we ran our program, using the model that we presented in Section 4.3. Initially the nodes in our network are distributed in a 1×1 square. We run each of the simulations for 60 minutes and log every traffic. The radius of the AoI is 0.1. In our dynamic runs we use 256 nodes, and the exponential mobility model from Section 4.3.1. In our static runs we set up individual groups with different sizes.

5.4.1 Log Files

During our validation we maintain two different log files. One is used to log the activities of the server and one is used to log the activity of the nodes in the network. In reality, we opened an individual file for each of the nodes. In the end we merged them together and analyzed the logs as one file, which reflects all the activities in the network.

The Server Log File

The server is responsible for determining the location of every node in the system and for providing information to all the joining nodes about their neighbors. Thus, we introduce an *ActionID* to be able to distinguish between two events: when we log a login and when we log a network topology change. Also, just in case, we logged the event when the server sends out new location information to the nodes (this event should not have to be logged). Since we run all the nodes on the same computer, there is no reason for any packet loss or delay. Instead, we logged this event to make sure our server functions properly.

Figure 5.3 shows a portion of a typical server log file. The *ActionId* specifies the

ActionID	From	To	Timestamp
Add	25	17	Sec: 1276914302 uSec: 759102
Del	115	68	Sec: 1276910104 uSec: 983161
Rec	72	S	Sec: 1276918282 uSec: 232761
Rep	S	8	Sec: 1276910195 uSec: 53251
Sen	S	91	Sec: 1276915828 uSec: 282800

Figure 5.3: *Server Log Entries:* We maintain 4 fields to keep track of the actions of the server. We provide an example of each of the events.

type of the event logged. It has one of the five following values:

- *Add*: a voice packet graph edge is added between two nodes.
- *Del*: a voice packet graph edge is deleted between two nodes.
- *Rec*: a join request received.
- *Rep*: a join request replied.
- *Sen*: a location update sent.

The *From* and *To* fields represent the sender and the receiver of the packet. They have either the number of a node in the network or the server itself. The *Timestamp* is the time when the event was logged in Unix time. A typical log file starts with several *Add* entries. This is when the server writes out the connections in the network after initialization. Next there are several *Rec* and *Rep* messages. This is the period when the nodes join the system. The *Del* and *Sen* entries are usually evenly distributed after the initialization period. *Del* messages do not appear during static validations. Although in this case we do not need *Sen* messages either, we did not disable them. After we ensured that the joining process is bug free, we focused on the stable state of the network, thus we started to analyze it a few seconds after the last logged *Rep* entry.

The Node Log File

The node log file is more complicated then the server log file. First, it has an additional field, the *PacketID*. This is to be able to identify which packet the node received from which node. During a talkspurt period, several packets are sent out closely to each other. To be able to validate, if these messages arrived in sequence, and with how much delay, we need to keep track of the exact message. A *PacketID* is a 12 byte long string. The first byte is always *N*. This symbolizes that the packet is from a node. If the packet arrives from the server, we simply put an *S* in this field, and nothing else. The following three bytes are the number of the node. In our case this ranges from 000 to 127. Next, an *S* is injected. This symbolizes that the remaining seven digits are the sequence number, which starts from 0000000 and ranges to 9999999. It is important to understand that, this field does not make the *From* field unnecessary. In case of a forwarding, the originator of a packet is not equivalent to the *From* node. Second, the *ActionId* field has more values then before. These are the following:

- *Add*: a voice packet graph edge is added between the node itself and a new neighbor.
- *Del*: a voice packet graph edge is deleted between the node itself and a previous neighbor.
- *RqS*: a join request is sent to the server.
- *RqR*: a join request reply received from the server.
- *Sen*: a voice packet is sent.
- *Rec*: a voice packet is received.
- *Fwd*: a voice packet is forwarded.

ActionID	PacketID	From	To	Timestamp
Add	N/A	25	9	Sec: 1276914302 uSec: 759102
Del	N/A	25	118	Sec: 1276911104 uSec: 983161
RqS	N/A	25	S	Sec: 1276910282 uSec: 53251
RqR	N/A	S	25	Sec: 1276910283 uSec: 232761
Sen	N025S0002834	25	69	Sec: 1276915828 uSec: 282800
Rec	N056S0003451	56	25	Sec: 1276915612 uSec: 712783
Fwd	N105S0000984	25	91	Sec: 1276917260 uSec: 202774
RRc	N043S0001374	43	25	Sec: 1276919073 uSec: 961395
RRp	N025S0000894	25	38	Sec: 1276917007 uSec: 421253
RSe	N025S0002843	25	101	Sec: 1276919645 uSec: 771166
RRR	N094S0001849	94	25	Sec: 1276914621 uSec: 181401

Figure 5.4: *Node Log Entries:* We maintain 5 fields to keep track of the actions of a node. We provide an example of each of the events. All of the entries belong to node 25.

- *RRc*: a routing request is received.
- *RRp*: a routing request reply is sent.
- *RSe*: a routing request is sent out.
- *RRR*: a routing request reply received.

The *From*, *To* and *Timestamp* fields serve a similar purpose as before. Figure 5.4 shows an example of a node log file. Since all the entries belong to node 25, the *From* field matches this for all the outgoing messages. However, the *PacketID* is different in the case of a forwarded packet.

5.4.2 Static Neighbor Maintenance

During this step of validation, first, we looked for *Del* lines in the server log file. If we did not find any, we knew that the movement function was disabled. Next, we filtered the same file for *Add* entries. These entries contain all the links between the servlets and have to match with all the links in the servlet log file (also *Add* entries).

We recorded every link twice in the server log file, making it easier to compare it to the servlet log file. If we had a connection between node A and B, we had two entries. One *From* node A *To* node B and one *From* node B *To* node A. Please note that the order of these entries does not have to match in the two log files.

Since our measurement study showed that, the most active groups have between 2 and 8 players, we used these numbers for our validation. We set up 7 scripts. Each of these scripts corresponds to one of the cases, and launches the node connections in sequence. Although, for voice delivery we tried both the UDP and TCP protocols, for neighbor maintenance we decided to use only TCP. Since, we ran all of our programs on the same machine, the joining process ran incredibly fast. Launching the server, the servlets and setting up the connections took only a few seconds.

5.4.3 Dynamic Neighbor Maintenance

This step is different in two ways from the previous one. First, we had to deal with *Del* entries. Second, we had to run the simulation for an entire hour. In fact, we ran it for a little more, cut off the beginning of the log files, and analyzed the stable state of the network. Table 5.2 shows the parameters and results of our run. We can see that, the server log file did not match the servlet log file with full accuracy. There were both *Add* and *Del* lines that were missing from the servlet log file. The information contained in the log files did not give us enough detail to identify why this happens. So, we ended up extending our server log file with additional information. After every *Add* and *Del* entry, we inserted an additional field to store the distance between the nodes. After reanalyzing the log files, we found that, in every case when the servlet log file missed an entry, the distance between the nodes was very close to 0.1. We suspect that, the cases that we missed are cases, in which two nodes walk very close to each other, but essentially walk by. The server can discover these cases, but the

Table 5.2: *Parameters of the Dynamic Neighbor Mainteance*: While using 128 nodes with 0.1 AoI, the accuracy was more then 99%.

Number of Nodes	**AoI**	**Accuracy**
128	0.1	99.24%

nodes cannot, since they send out update requests on a periodic basis, and therefore might miss some of these cases. To support our theory, we looked at the missing entries again. We found that, the server added and almost immediately deleted these entries. Thus, we believe that our theory is true.

5.4.4 Static Voice Packet Delivery

Here, we analyzed, if all of our voice packets were delivered. We set up groups with 2 to 8 people. Since we worked with separate groups, we did not have any forwarded messages, or routing entry changes, after we reached the stable state. In fact, this is how we found the beginning of that portion of the servlet log file that we analyzed. Unlike our previous cases, we used both UDP and TCP connections. We kept using only TCP for neighbor maintenance, but for voice delivery we tried both.

Table 5.3 shows our results. Since we run all of our servlets on the same machine, the delay was extremely small. However, although it was small, it was significantly different in the two cases. The average delay of TCP delivery was 46% longer then that of the UDP delivery. Please note that, we do not imply that this ratio would hold on the Internet, but we believe that the average delivery time of UPD is shorter, since it is a connectionless protocol. Furthermore, we have to mention that, TCP has other advantages over UDP as well. Since it is the most common, and almost the only protocol used today for data delivery on the Internet, the number of restrictions that system administrators set up against it is much less. Therefore, using TCP might be a better choice to pass firewalls.

Table 5.3: *Delay of the Static Voice Packet Delivery*: Using UDP for voice packet delivery resulted in shorter delay, but this solution has several disadvantages compared to TCP.

Protocol Used for Voice Packet delivery	**Average Delay**
UDP	0.166 ms
TCP	0.243 ms

5.4.5 Dynamic Voice Packet Delivery

The dynamic voice packet delivery validation gave us an opportunity to analyze the details of our implementation. Although we focused on the stable state, we logged several changes during the one hour long run. We call the state stable, if no node joins or leaves the network, but we can still have several neighbor changes.

Table 5.4 shows the results of our runs. We added two new columns to the table. The one hop delay is the same as the delay was before. Since we do not have separate groups, with a direct connection between every node, we have to distinguish between one hop delays, and delays in the network. The one hop delay is the difference between the time a voice packet leaves a node and the time when the next node receives it. The delay is the difference between the time the voice packet leaves the original sender and the time when a node receives it. Thus, this contains the delay of a forwarded packet between the sender and the first receiver, and also between the sender and the second receiver. In theory, this time should be less than the product of the one hop delay and the average route length, since that only shows the delay between a sender and the last nodes that receive the message, but not the intermediate ones.

Interestingly, the average one hop delay was longer then it was before. This is probably because we ran several programs on the same machine at the same time, and therefore the processing time increased. Even more interestingly, the average delay was longer then the product of the average one hop delay and the average route

Table 5.4: *Results of the Dynamic Voice Packet Delivery*: Using UDP for voice packet delivery resulted in shorter delay, but this solution has several disadvantages compared to TCP.

Protocol Used for Voice Packet delivery	**Average One Hop Delay**	**Average Delay**	**Average Route Length**
UDP	0.173 ms	0.348 ms	1.51
TCP	0.254 ms	0.521 ms	1.47

length. Our detailed servlet log file helped us to understand the cause. In addition to the message sending process time we also have to add the processing time. Every node has to look it up in its own routing table, if the voice packet has to be forwarded or not. If yes, it also has to determine the recipients. These additional steps were responsible for the extra delay. Please note that, since the routing table is fairly small, this process adds only a fraction of a ms extra delay, and the only reason why we discovered this is because we run all of our programs on the same machine. In real world deployment, where a few ms long delays are normal, we would not be able to point out this extra time. The difference between the delay of the UDP delivery method and the TCP delivery method remained about the same as it was between the static runs.

5.5 Conclusion

We presented a possible implementation of each part of our protocol and proved that, they can work together, and the solution is ready for real world deployment. We also compared and contrasted the two most common delivery methods used today on the Internet. We found that using UDP resulted in lower delays. However, our testbed consisted of only one machine and therefore we did not have the opportunity to see how each of these protocols would react if there were lost packets or longer delays in the system.

Chapter 6

Conclusion

The task of defining a virtual-location-aware, peer-to-peer protocol for muliparty voice communication includes several challenges and thus it is impossible to find an all-around solution. As the Internet continues to grow and change, new techniques appear to handle the ever increasing traffic. New devices are added and new requirements show up every day.

To better understand the requirements of a multiparty voice communication software that is adequate for players of massive multiplayer virtual environments, we conducted a measurement study. Throughout several months we collected more than 100 GB of voice communication data. Using our data, we identified typical talking patterns, such as the length of talkspurts and silence periods. We also characterized group sizes, and looked into the talking behavior of speakers in groups with different sizes. Using our results we presented a model that can help generate synthetic data for testing.

After understanding the specific properties of online multiparty voice communication, we defined the characteristics of an ideal protocol: it has to reflect the positions of the users in the virtual world to increase user satisfaction and it also has to have

low bandwidth requirements. To achieve this, delay can be sacrificed as long as it does not hurt voice communication with close neighbors in the virtual world. To fulfill such requirements, we looked at different peer-to-peer techniques, such as unstructured and structured solutions. Since our goal is to identify users of the virtual world that are inside of the area of interest of a speaker, we decided to use an unstructured solution. We also had to find a technique that would set up connections between users similarly as voice would spread in the real world. Gabriel graphs proved to be a good match. They are also a subgraph of the Delaunay-triangulation, and such they are easy to maintain and calculate. After finding the right structure, we worked out the detail of the protocol and tested it against several scenario. To make it suitable for everyday use, we added security against both passive and active attacks. We then tested the effectiveness of these solutions too.

The last step of our work was to try out, if our protocol would work in the real world. We implemented two programs in the C programming language, that could go to real world deployment with minor changes. We then validated that our protocol would be able to deliver in practice, what it delivered in theory. Our results show that it matched our expectations.

As future work, we would like to make our implementation public and downloadable for free of charge. We are looking forward to seeing how successful and popular it will be, and what additional features people are going to request or add to it.

Bibliography

[1] S. Baset and H. Schulzrinne. An analysis of the Skype peer-to-peer internet telephony protocol. In *Proceedings of IEEE Infocom*, pages 1–11, April 2006.

[2] M. Borella. Source models of network game traffic. *Computer Communications*, 23(4):403–410, February 2000.

[3] P. Bose and P. Morin. Online routing in triangulations. In *ISAAC '99: Proceedings of the 10th International Symposium on Algorithms and Computation*, pages 113–122, London, UK, 1999. Springer-Verlag.

[4] P. Boustead and F. Safaei. Comparison of delivery architectures for immersive audio in crowded networked games. In *NOSSDAV '04: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, pages 22–27, New York, NY, USA, 2004. ACM.

[5] C. Boutremans, G. Iannaccone, and C. Diot. Impact of link failures on VoIP performance. In *Proceedings of ACM NOSSDAV*, pages 63–71, 2002.

[6] P. Brady. A statistical analysis of on-off patterns in 16 conversations. *Bell Systems Technical Journal*, 47(1):73–91, January 1968.

[7] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, 2002.

[8] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC)*, 2002.

[9] K.-T. Chen, C.-Y. Huang, P. Huang, and C.-L. Lei. Quantifying Skype user satisfaction. In *Proceedings of ACM SIGCOMM*, pages 399–410, 2006.

[10] M. Dowlatshahi and F. Safaei. Multipoint interactive communication for peer-to-peer environments. In *Proceedings of IEEE International Conference on Com-*

munication, June 2006.

[11] J. Färber. Network game traffic modelling. In *Proceedings of the 1st workshop on Network and system support for games*, pages 53–57, 2002.

[12] W. Feng, E. Kaiser, and A. Luu. Design and implementation of network puzzles. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 4, pages 2372–2382 vol. 4, March 2005.

[13] W.-c. Feng. The case for tcp/ip puzzles. *SIGCOMM Comput. Commun. Rev.*, 33(4):322–327, 2003.

[14] A. Fessi, H. Niedermayer, H. Kinkelin, and G. Carle. A cooperative sip infrastructure for highly reliable telecommunication services. In *IPTComm '07: Proceedings of the 1st international conference on Principles, systems and applications of IP telecommunications*, pages 29–38, New York, NY, USA, 2007. ACM.

[15] M. Ghaffari, B. Hariri, and S. Shirmohammadi. A delaunay triangulation architecture supporting churn and user mobility in mmves. In *NOSSDAV '09: Proceedings of the 18th international workshop on Network and operating systems support for digital audio and video*, pages 61–66, New York, NY, USA, 2009. ACM.

[16] L. J. Gleser and D. S. Moore. The effect of dependence on chi-squared and empiric distribution tests of fit. *Annals of Statistics*, 11(4):1100–1108, 1983.

[17] J. Gruber. A comparison of measured and calculated speech temporal parameters relevant to speech activity detection. *IEEE Transactions on Communications*, pages 739–750, April 1982.

[18] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. Promise: peer-to-peer media streaming using collectcast. In *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*, pages 45–54, New York, NY, USA, 2003. ACM.

[19] T. Henderson and S. Bhatti. Modelling user behaviour in networked games. In *MULTIMEDIA '01: Proceedings of the Ninth ACM International Conference on Multimedia*, pages 212–220, New York, NY, USA, 2001. ACM Press.

[20] 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, july-aug. 2006.

[21] W. Jian and H. Schulzrinne. Analysis of on-off patterns in VoIP and their effect on voice traffic aggregation. In *Proceedings of Computer Communications and Networks*, pages 82–87, Oct. 2000.

[22] M. J. Karam and F. A. Tobagi. Analysis of delay and delay jitter of voice traffic in the internet. *Comput. Netw.*, 40(6):711–726, 2002.

[23] S. Kraxberger and U. Payer. Security concept for peer-to-peer systems. In *IWCMC '09: Proceedings of the 2009 International Conference on Wireless Communications and Mobile Computing*, pages 931–936, New York, NY, USA, 2009. ACM.

[24] H. J. Larson. *Statistics: An introduction*. Wiley, New York, NY, USA, 1975.

[25] D.-Y. Lee and S. S. Lam. Protocol design for dynamic delaunay triangulation. In *ICDCS '07: Proceedings of the 27th International Conference on Distributed Computing Systems*, page 26, Washington, DC, USA, 2007. IEEE Computer Society.

[26] J. Lundberg. Routing security in ad hoc networks, 2000.

[27] A. P. Markopoulou, F. A. Tobagi, and M. J. Karam. Assessing the quality of voice communications over internet backbones. *IEEE/ACM Trans. Netw.*, 11(5):747–760, 2003.

[28] D. W. Matula and R. R. Sokal. Properties of gabriel graphs relevant to geographic variation research and clustering of points in the plane. *Geographical Analysis*, 1980.

[29] P. Mittal and N. Borisov. Information leaks in structured peer-to-peer anonymous communication systems. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 267–278, New York, NY, USA, 2008. ACM.

[30] K. Needels and M. Kwon. Secure routing in peer-to-peer distributed hash tables. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 54–58, New York, NY, USA, 2009. ACM.

[31] M. Ohnishi, R. Nishide, and S. Ueshima. Incremental construction of delaunay overlaid network for virtual collaborative space. In *Proceedings of IEEE Conference on Creating, Connecting and Collaborating through Computing*, 2005.

[32] V. Paxson. End-to-end routing behavior in the internet. *IEEE/ACM Trans. Netw.*, 5(5):601–615, 1997.

[33] S. P. Pederson and M. E. Johnson. Estimating model discrepancy. *Technometrics*, 32(3):305–314, 1990.

[34] D. Pittman and C. GauthierDickey. A measurement study of virtual populations in massively multiplayer online games. In *Proceedings of ACM NetGames*, September 2007.

[35] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, pages 161–172, 2001.

[36] S. Ratnasamy, M. Handley, R. Karp, and S. Shenkar. Application-level multicast using content addressable networks. In *Proceedings of Network Group Communications*, November 2001.

[37] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–350. Springer-Verlag, 2001.

[38] F. Safaei. Dice: Internet delivery of immersive voice communication for crowded virtual spaces. In *VR '05: Proceedings of the 2005 IEEE Conference 2005 on Virtual Reality*, pages 35–41, Washington, DC, USA, 2005. IEEE Computer Society.

[39] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[40] K. Singh and H. Schulzrinne. Peer-to-peer internet telephony using sip. In *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pages 63–68, New York, NY, USA, 2005. ACM.

[41] K. Sriram and W. Whitt. Characterizing superposition arrival processes in packet multiplexers for voice and data. *IEEE Selected Areas in Communications*, 4(6):833–846, 1986.

[42] M. Srivatsa and L. Liu. Securing publish-subscribe overlay services with eventguard. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, pages 289–298, New York, NY, USA, 2005. ACM.

[43] M. Steiner and E. W. Biersack. A fully distributed peer to peer structure based on 3d delaunay triangulation. In *Proceedings of Algotel*, May 2005.

[44] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord:

A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, pages 149–160, 2001.

[45] P. Svoboda, W. Karner, and M. Rupp. Traffic analysis and modeling for world of warcraft. *2007. ICC '07. IEEE International Conference on Communications*, pages 1612–1617, 24-28 June 2007.

[46] M. Varvello, E. Biersack, and C. Diot. Dynamic clustering in delaunay-based p2p networked virtual environments. In *Proceedings of ACM NetGames*, September 2007.

[47] Wu-chang Feng, F. Chang, Wu-chi Feng, and J. Walpole. Provisioning on-line games: A traffic analysis of a busy counter-strike server. In *Internet Measurement Workshop*, 2002.

[48] C. K. Yeo, S. C. Hui, I. Y. Soon, and L. M. Ang. H.323 compliant voice over ip system. *Int. J. Comput. Appl. Technol.*, 16(4):143–153, 2003.