# Protocols and System Design, Reliability, and Energy Efficiency in Peer-to-Peer Communication Systems

**Salman Abdul Baset**

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

**COLUMBIA UNIVERSITY**

2011

# ABSTRACT

**Protocols and System Design, Reliability, and Energy Efficiency in Peer-to-Peer Communication Systems**

**Salman Abdul Baset**

Modern Voice-over-IP (VoIP) communication systems provide a bundle of services to their users. These services range from the most basic voice-based services such as voice calls and voicemail to more advanced ones such as conferencing, voicemail-to-text, and online address books. Besides voice, modern VoIP systems provide video calls and video conferencing, presence, instant messaging (IM), and even desktop sharing services. These systems also let their users establish a voice, video, or a text session with devices in cellular, public switched telephone network (PSTN), or other VoIP networks.

The peer-to-peer (p2p) paradigm for building VoIP systems involves minimal or no use of managed servers and is therefore attractive from an administrative and economic perspective. However, the benefits of using p2p paradigm in VoIP systems are not without their challenges. First, p2p communication (VoIP) systems can be deployed in environments with varying requirements of scalability, connectivity, security, interoperability, and performance. These requirements bring forth the question of designing open and standardized protocols for diverse deployments. Second, the presence of restrictive network address translators (NATs) and firewalls prevents machines from directly exchanging packets and is problematic from the perspective of establishing direct media sessions. The p2p communication systems address this problem by using an intermediate peer with unrestricted connectivity to relay the session or by preferring the use of TCP. This technique for addressing connectivity problems raises questions about the reliability and session quality of p2p communication systems compared with the traditional client-server VoIP systems. Third, while administrative overheads are likely to be lower in running p2p communication sys-

tems as compared to client-server, can the same be said about the energy efficiency? Fourth, what type of techniques can be used to gain insights into the performance of a deployed p2p VoIP system like Skype?

The thesis addresses the challenges in designing, building, and analyzing peer-to-peer communication systems. The thesis presents Peer-to-Peer Protocol (P2PP), an open protocol for building p2p communication systems with varying operational requirements. P2PP is now part of the IETF's P2PSIP protocol and is on track to become an RFC. The thesis describes the design and implementation of OpenVoIP, a proof-of-concept p2p communication system to demonstrate the feasibility of P2PP and to explore issues in building p2p communication systems. The thesis introduces a simple and novel analytical model for analyzing the reliability of peer-to-peer communication systems and analyzes the feasibility of TCP for sending real-time traffic. The thesis then analyzes the energy efficiency of peer-to-peer and client-server VoIP systems and shows that p2p VoIP systems are less energy efficient than client-server even if the peers consume a small amount of energy for running the p2p network. Finally, the thesis presents an analysis of the Skype protocol which indicates that Skype is free-riding on the network bandwidth of universities.

# Table of Contents

# List of Figures

# List of Tables

# Acknowledgments

I want to express my deepest, sincerest, and utmost gratitude for my PhD advisor, Prof. Henning Schulzrinne. Henning has been a research leader in IP communications and networking, and has significantly impacted the VoIP industry through the protocols he designed which are now Internet standards. I was very fortunate to work with him as a PhD student and have greatly benefited from his insights in research and protocol standardization. When I was a Masters student at Columbia, Henning suggested to me to investigate the workings of the Skype application. It was an equally challenging and fun project and greatly motivated me to pursue a PhD in the area of peer-to-peer communication systems. Henning is very clear, to the point, and always offers a unique perspective which makes you wonder "Why didn't I think of this?". He has this gifted ability of clearly laying out the pros and cons of different solutions for addressing a problem which helps his audience, be it a reader of his research papers, his students, peers, or folks in the standardization committees, thoroughly understand the relevant issues to arrive at the best solution. In my fledgling research career, I have tried to emulate this unique characteristic of his. Henning introduced me to the Internet Engineering Task Force (IETF) where I have been fortunate to work on the standardization of peer-to-peer Session Initiation Protocol. I am forever indebted to him for his mentoring and support and will continue to benefit from it in my research career.

I am very grateful to my PhD committee members, Prof. Vishal Misra, Prof. Dan Rubenstein, Prof. Keith Ross, and Prof. Richard Yang for providing valuable and critical feedback on my dissertation. The quality of my work has been significantly enhanced by their comments.

I am thankful to the Columbia faculty with whom I took courses or was a teaching assistant for their classes. In particular, I want to thank Prof. Vishal Misra, Prof. Angelos

D. Keromytis, Prof. Alfred Aho, Prof. Clifford Stein, Prof. Mark Brown, Prof. Yechiam Yemini, Prof. Angelos D. Keromytis, Prof. Kathy McKeown and Prof. Steven M. Bellovin. I vividly recall the fun and learning I had in designing and implementing a language for controlling the desktop windows behavior as part of the the programming languages and translators course taught by ProfȦho. I was also fortunate to work as a teaching assistant coordinator in the computer science department under Prof. Elizabeth Sklar and Prof. Tal Malkin, and am grateful for their mentoring and support. I also want to sincerely thank my undergraduate thesis advisor Prof. Syed Afaq Hussain and other faculty who imparted me with knowledge and guided me as an undergraduate student. I also want to express my deepest gratitude for Sylvia Ratnasamy, Gianluca Iannaccone, Venkatesh Krishnaswamy, Kishore Dhara, and Wassim Matragi with whom I worked as a summer intern.

I was fortunate to be surrounded with talented colleagues at my research lab, the computer science department at Columbia, and beyond with whom I was able to bounce research ideas, and weed out the silly ideas from the promising ones. I want to express my sincerest thanks to Kundan Singh, Eli Brosh, Jan Janak, Joshua Reich, Jae Woo Lee, Kumiko Ono, Wonsang Song, Jong Yul Kim, Andrea Forte, Omer Boyaci, Jonathan Lennox, Saikat Guha, Knarig Arabshian, Alex Sherman, Oren Ladaan, Malek Salem, Ang Cui, Stelios Sidiroglou-Douskos, Angelos Stavrou, Abhinav Verma, Rishi Talreja, Ilias Diakonikolas, Abhinav Kamra, Matthew Burnside, and Faisal Ghias Mir. I am also grateful to my office mates at Columbia, Wisam Dakka, Sameer Maskey, Arezu Moghadam, and Kumiko Ono for making the work environment a fun and lively place.

I am grateful and appreciative of the work done by Gaurav Gupta, Dhruv Chopra, Wookyun Kho, Max Czapanskiy, Jiaje Chen and Rebecca Tong, who were undergraduate or master students at Columbia, and worked with me on my research projects. In Spring 2008, I had an opportunity to teach the Networking Laboratory course at Columbia and had fun interacting and working with students.

My stay in the computer science department was facilitated by the wonderful and ever pleasant administrative, computing, and facilities staff. I am thankful to Daisy Nguyen, Paul Blaer, Shlomo Hershkop, Patricia Hervey, Rosemary Addarich, Lily Bao, Elias Tesfaye, Twinkle Edwards, Cindy Walters, Evelyn Guzman, Alice Cueba, and Jonathan Stark for

their help and support.

I have learned a lot in the process of standardizing Peer-to-Peer Session Initiation Protocol (P2PSIP) at IETF. I want to sincerely thank folks at the P2PSIP working group IETF and especially Cullen Jennings, Eric Rescorla, Bruce Lowekamp, Jonathan Rosenberg, Gonzalo Camarillo, Marcin Matuszewski, and Henry Sinnerich for the many technical discussions.

I want to express my most profound and deepest gratitude to my parents who always loved me and gave me the best possible education. I will not be what I am without them. The untimely death of my mother in 2006 left my family and myself in a great shock. I am very grateful to my extended family, especially my mother's sisters and the mother of my late friend Rao Salman Aziz, for helping my family and myself navigate through the emotionally difficult time.

It is not possible to list all the individuals who have helped me in my professional and educational life. I want to express my sincerest thanks to everyone who have imparted me with any knowledge and groomed me as a student and as an individual. I also acknowledge the funding organizations that made my research possible.

To my parents.

# Chapter 1

# Introduction

Modern Voice-over-IP (VoIP) communication systems provide a bundle of services to their users. These services range from the most basic voice-based services such as establishing voice calls and storing and retrieving voicemail to the more advanced such as conferencing, voicemail-to-text, and online address books. Besides voice, modern VoIP systems provide video calls and video conferencing, presence, instant messaging (IM), and even desktop sharing services. These systems also let their users establish a voice, video, or a text session with devices in cellular, public switched telephone network (PSTN), or other VoIP networks.

Client-server (c/s) is one way to architect modern VoIP systems [Rosenberg *et al.*, 2002]. The idea is that clients (or user agents) can establish a media session[1] with the assistance of managed servers. In these systems, the user agents store their reachable network address with a managed registrar using a session establishment protocol such as Session Initiation Protocol (SIP) [Rosenberg *et al.*, 2002]. The user agents typically run on hardphones, mobile devices, or desktop machines. When a user desires to establish a media session with another user, its user agent sends the signaling (call establishment) message to the proxy server which is co-located with the registrar. The proxy server locates the network address of the callee user agent and forwards the call establishment message to the callee user agent. The user agents then directly exchange the media traffic such as voice, video, or IM. However, the presence of middle boxes such as restrictive network address translators

[1] we refer to voice, video, or IM session as a media session.

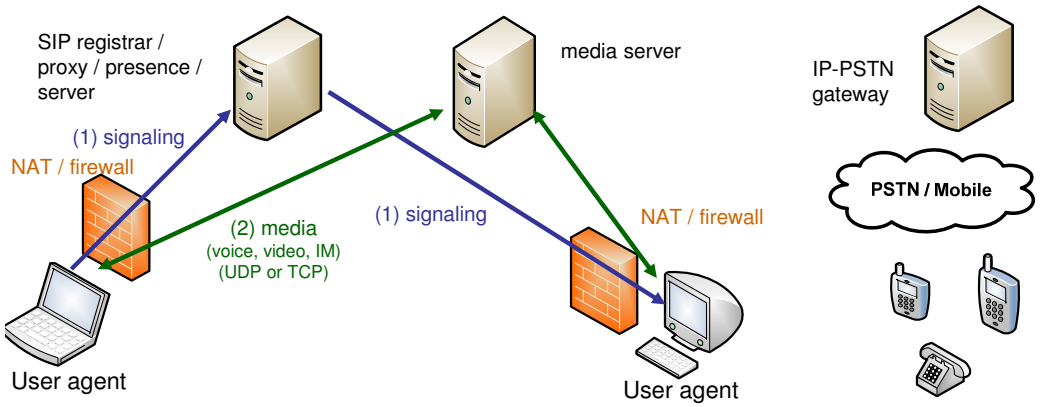



Figure 1.1: A client-server communication system.

(NATs) [Audet and Jennings, 2007] and firewalls may prevent them from directly exchanging packets. Therefore, in addition to the proxy server for signaling, the user agents also have to rely on a managed relay server that relays the media traffic between them. Moreover, the user agents may also use managed servers for establishing an audio or a video conference since they may not have sufficient uplink or downlink network capacity to establish a full-mesh conference [Lennox and Schulzrinne, 2003]. Further, if a user agent desires to establish a media session with a device outside the client-server system such as a PSTN or mobile phone, it must do so through a designated server or a gateway.

Using designated servers to provide directory service, media relaying due to NATs and firewalls, conferencing, and presence creates a network and administrative overhead for the provider. Such overheads have associated economic costs which grow with the number of users. Figure 1.1 shows an instance of a client-server communication system where two user agents behind a restrictive NAT and firewall exchange media traffic through a managed server. The figure also shows an IP-PSTN gateway that enables user agents in the client-server system to establish media sessions with devices in PSTN and cellular networks.

The peer-to-peer (p2p) paradigm is another way to architect modern VoIP systems and has been popularized by Skype [Skype, 2010a]. The idea is that the user agents or nodes cooperate to provide the directory service (registrar), call establishment, and media relaying services which are provided by managed servers in the client-server VoIP systems. The user agents do so by running an **overlay protocol** which distributes the functionality

of directory service, call establishment, and media relaying among the user agents. At the heart of the overlay protocol lies the **overlay algorithm**[2], which determines how a user agent forwards messages to the intended user agent in a distributed manner. In the research literature, user agents providing the distributed routing and storage services are known as **super nodes** [Liang *et al.*, 2004; Baset and Schulzrinne, 2006] or **peers** [Bryan *et al.*, 2010]. Similar to client-server systems, the user agents run on hard phones, mobile devices and desktop machines with heterogeneous network connectivity. Due to their hardware and network capabilities, some user agents may not be able to provide registrar or media relaying services. Such user agents may not run the overlay protocol, and instead connect to one or more user agents running the overlay protocol. In the research literature, such user agents are known as **ordinary nodes** [Liang *et al.*, 2004], **ordinary hosts** [Baset and Schulzrinne, 2006] or **clients** [Bryan *et al.*, 2010].

To establish a media session in a p2p communication system, the caller user agent launches a distributed search using the overlay protocol to discover the network address of the callee user agent and then directly exchanges signaling and media traffic with the callee user agent. However, since restrictive NATs and firewalls may prevent some user agents from directly exchanging packets, the user agents involved in setting up a media session rely on user agents with unrestricted connectivity to relay signaling and media traffic between them. Typically, the user agents send real-time traffic such as voice and video over unreliable protocol such as UDP, but may be forced to use TCP if the NATs or firewalls block UDP. Since user agents may not have sufficient uplink and downlink network capacity to establish an audio or a video conference, they may use other user agents as conferencing relays. Further, user agents may also cooperate to provide voicemail, address book, and presence service to other user agents. However, like the client-server communication systems, the user agents must contact a designated server to establish a media session with devices in cellular, PSTN, or other communication networks.

Figure 1.2 shows an instance of a peer-to-peer communication system. The nodes A-E run the overlay protocol and provide distributed directory and media relaying services, whereas nodes F and G are behind a restrictive NAT or a firewall device and connect to

[2]Chord [Stoica *et al.*, 2003] is an example of an overlay algorithm.

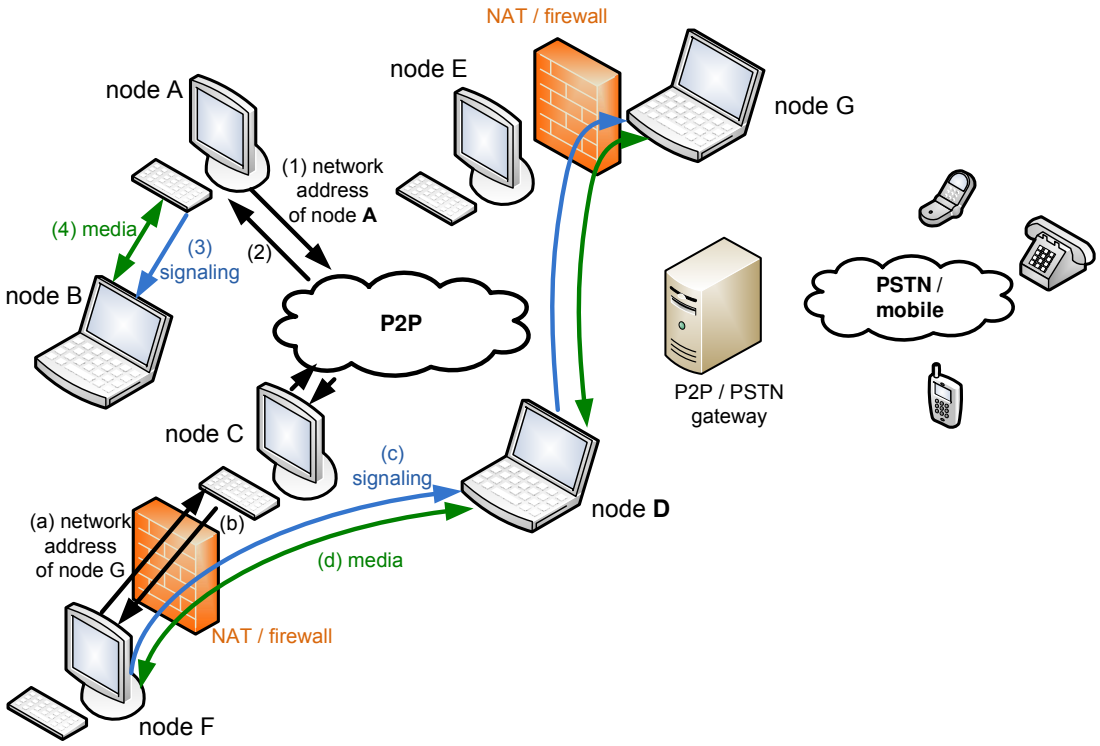



Figure 1.2: A peer-to-peer communication system.

nodes C and E, respectively, to establish a media session. The nodes A and B directly exchange media traffic whereas nodes F and G use the node D for exchanging media traffic.

## 1.1 Problems and Challenges

The above characteristics of a p2p communication system give rise to the following unique problems and challenges for designing, building, and analyzing these systems.

**Protocol and System Design:** How should open, standardized, and interoperable protocols for building peer-to-peer communication systems be designed so that these systems can be deployed in ad hoc, enterprise, and Internet environments? How media sessions can be established in the presence of restrictive NATs and firewalls and how to locate user agents that can relay media sessions for user agents behind restrictive NATs and firewalls?

**Conferencing:** How can audio and video conferences be established and scaled without the use of managed servers since the network capacity of the participants is finite and they may also be behind NATs and firewalls? How can non-participant peers help scale the conference?

**Reliability:** What factors impact the reliability of a p2p communication system and how do we analyze its reliability? What techniques can be used to improve the reliability of such a system, especially the relayed media sessions?

**Interference with the User Application:** A user agent relaying a voice or a video session shares network resources with other user applications. The relaying of a media session can interfere with other user applications that require the network bandwidth, and impair their performance. How can we characterize and minimize this interference, or provide incentives to the user for relaying media sessions?

**Session Quality:** The user agents that are unable to exchange UDP packets due to restrictive NATs and firewalls may be able to use TCP for establishing a media session. What are the conditions under which sending real-time traffic such as voice and video over TCP is feasible?

**Energy Efficiency:** Are p2p communication systems more energy efficient than c/s VoIP systems? What is the total energy consumption of a VoIP system and what are the sources of inefficiency in such a system? How can we alleviate those inefficiencies?

**Measurement:** How can the performance of peer-to-peer communication systems such as Skype be measured? Is Skype free-riding on the network bandwidth of universities?

## 1.2 Overview and Contributions of the Thesis

This thesis focuses on the above mentioned problems and challenges in designing, building, and analyzing peer-to-peer communication systems. The thesis is divided into three parts. Part I (Chapter 3-5) presents protocols and systems for designing and building peer-to-peer

communication systems, and for enabling video conferencing without managed servers. Part II (Chapter 6-8) presents analysis of reliability, session quality, energy efficiency in these systems. Part III (Chapter 9) focuses on devising measurement techniques to gain insights into the workings of peer-to-peer communication systems, such as Skype. A brief overview of the chapters and their contributions is as follows.

Chapter 3 discusses the requirements for designing a peer-to-peer communication protocol. It then presents Peer-to-Peer Protocol (P2PP), an open and interoperable protocol for building p2p communication systems [Baset *et al.*, 2007]. The protocol has been incorporated in the RELOAD protocol [Jennings *et al.*, 2010] which is being standardized in the Peer-to-Peer Session Initiation Protocol (P2PSIP) working group of the Internet Engineering Task Force (IETF).

Chapter 4 presents OpenVoIP, a proof-of-concept system that demonstrates the feasibility of P2PP and explores issues in building p2p communication systems [Baset and Schulzrinne, 2008].

Chapter 5 presents a theoretical framework for peer assisted audio and video conferencing. The framework assumes that peers have finite uplink capacities, analyzes how to add new participants without disrupting the existing participants, and the reliability of p2p video conferences.

Chapter 6 formalizes the notion of reliability in peer-to-peer communication systems, presents a simple model to analyze the reliability of relayed media sessions, and describes techniques to improve the reliability of such media sessions [Baset and Schulzrinne, 2010]. Then, it discusses distributed techniques to find a user agent willing to relay media session in $O(1)$ hops such that the latency of a relayed call and its interference with the user applications is minimized.

Chapter 7 characterizes the working region under which sending real-time traffic may be feasible over TCP. It explores the impact of packet size and TCP protocol settings on the TCP induced delay, provides guidance on setting playout buffers for TCP, and introduces techniques to minimize the impact of delay variations due to the AIMD nature of TCP [Brosh *et al.*, 2008].

Chapter 8 presents a framework to compare the energy efficiency of p2p and c/s com-

munication systems. It identify sources of energy inefficiency in these systems [Baset *et al.*, 2010], presents the economic cost per user per month of running such a system, and provides recommendations to alleviate the inefficiencies.

Chapter 9 details measurement techniques for analyzing peer-to-peer communication systems that use proprietary protocols. Skype is presented as the case study [Baset and Schulzrinne, 2006]. The measurements indicate that Skype is free riding on the network bandwidth of the universities [Kho *et al.*, 2008].

## 1.3 Related Work

Singh [Singh, 2006] and Bryan [Bryan *et al.*, 2005] proposed architectures for building peer-to-peer Session Initiation Protocol (SIP) systems. Their work focused on the architecture level issues and demonstrated the feasibility of distributing the registrar and proxy servers defined by the SIP protocol to the user agents. This thesis focuses on designing, building, and analyzing peer-to-peer communication systems and comprehensively explores issues such as protocol and system design, reliability, relay selection, session quality, conferencing, energy efficiency, and measurement. Related work relevant to these issues is discussed in each chapter.

# Chapter 2

# Definitions and Background

This chapter provides the definitions and background that are applicable throughout the thesis. As mentioned in Chapter 1, network address translators (NATs) may prevent hosts from directly exchanging packets. We provide an overview of different types of NATs, the problems they cause, and the protocols for exchanging packets in the presence of NATs directly or through an intermediary (Section 2.2). We then comment on the percentage of VoIP calls in the Internet that may need an intermediary due to restrictive NATs and firewalls (Section 2.3). The nodes or user agents can potentially use any structured or unstructured overlay algorithm to form a p2p communication network. We provide a brief overview of structured and unstructured overlay algorithms in Section 2.4.

## 2.1 Definitions

In this section, we define the terminology that is applicable throughout the thesis.

**Peer-to-Peer or Overlay communication network** is an overlay network that entities (nodes) form by running a p2p/overlay protocol in order to establish media sessions with minimal or no use of managed servers.

**P2P or Overlay communication protocol** defines the messages that nodes must exchange for the operation of the p2p network, includes mechanisms such as NAT traversal and security, and allows devices with heterogeneous resources and capabilities to

participate in the p2p network.

**Overlay algorithm** determines the next node in the overlay where a node should send the message. Distributed hash tables and Gnutella [Chawathe *et al.*, 2003] are examples of structured and unstructured overlay algorithms.

**Node** is an entity that either runs the p2p protocol (peer) or connects to one or more peers to use the services provided by the overlay (client). Depending on the context, it is also referred to as an endpoint or user agent in the thesis.

**Peer** is node that fully participates in the p2p network, runs the p2p protocol, and provides message routing and media relaying services.

**Client** is a node that connects to one or more peers to use the services provided by the p2p network.

**DHT** is a form of structured overlay algorithm, that provides a lookup and storage service similar to a hash table. The responsibility of locating and storing the key/value pairs is distributed among the nodes.

**Overlay operator or provider** is an entity that enables node to run a p2p communication system.

## 2.2 Network Address Translators (NATs)

NAT devices [Egevang and Francis, 1994] are typically deployed at the edge of the network, commonly referred to as an internal network, and present it as a single IP address to the external network. The NAT devices have at least two IP addresses, an internal IP address and an external IP address. The later is also referred to as the server-reflexive address [Rosenberg, 2010]. When a device in the internal network needs to exchange packets with a device in the external network, the NAT device assigns an external IP address and port number for this exchange, so that the packets from the device in the external network can be routed back to the NAT and onwards to the device in the internal network. The NAT devices may also filter the packets that arrive at the external IP address and port number

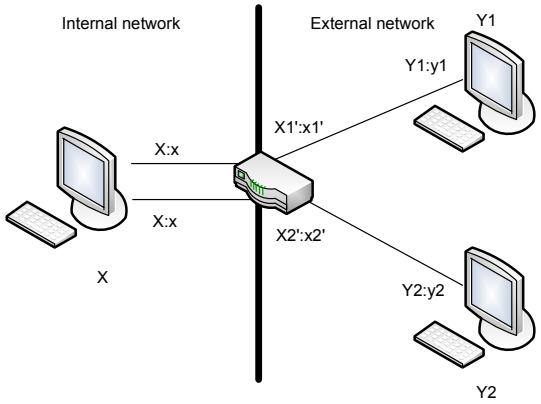



Figure 2.1: Machine X in the internal network exchanging packets with machines Y1 and Y2 in the external network through a NAT device.

allocated by the NAT device. RFC 5128 [Srisuresh *et al.*, 2008] refers to these behaviors as the mapping and filtering behaviors of NATs. Together, these behaviors determine if machines behind two different NAT devices can directly exchange packets or may require the use of an intermediary. Next, we briefly elaborate the mapping and filtering behavior of NAT devices. For a detailed description of these behaviors, we refer the reader to [Audet and Jennings, 2007; Ford *et al.*, 2005].

Consider Figure 2.1 where a machine with an IP address and port number X:x behind a NAT device needs to exchange packets with machines Y1 and Y2 in the external network, having IP addresses and port numbers Y1:y1 and Y2:y2, respectively. Assume that for the packet exchange between X:x and Y1:y1 and X:x and Y2:y2, the NAT device allocates an external IP address and port number X1':x1' and X2':x2', respectively.

### 2.2.1 Mapping Behavior of NATs

The mapping is known as **endpoint-independent mapping**, if the NAT reuses the mapping for subsequent packets sent from the same internal IP address and port X:x to any external IP address and port such as Y1:y1 or Y2:y2. The mapping is known as **address-dependent mapping** if the NAT reuses the mapping for subsequent packets sent from the same internal IP address and port X:x to the same external IP address, regardless of the

external port. That is, packets sent from X:x to any port on Y1 will use the same mapping X1':x1'. However, any packets sent by X:x to any port on Y2 will use a different mapping X2':x2'. The mapping is known as **address and port-dependent mapping** if the NAT assigns a new mapping for packets sent from the same internal IP address and port X:x to any external IP address and port number.

If the NAT behavior is not endpoint-independent, then it requires the hosts behind two different NAT devices to correctly guess the mapping allocated by the NAT device in order to directly exchange packets. An incorrect guess means that hosts behind two different NATs must exchange packets through an intermediary or a relay.

### 2.2.2 Filtering Behavior of NATs

When a host in an internal network exchanges packets with a host in the external network, the NAT stores a mapping between the internal host and the external host and assigns a filtering rule. The rule is known as **endpoint-independent filtering** if the NAT device forwards packets received from any host on this mapping. The rule is known as **address-dependent filtering** if the NAT device only forwards packets from the external host for which the mapping is maintained, regardless of the source port of the packet received from the external host. The rule is known as **address and port-dependent filtering** if the NAT device only forwards packets from the port of the external host with which the internal host has exchanged packets.

The NATs with endpoint-independent mapping and filtering behavior are the least constrained types. The devices behind these NAT devices are able to establish a media session without an intermediary. The NATs with address and port-dependent mapping and filtering are the most constrained type.

### 2.2.3 Typical NAT Behavior

A survey of 1,787 unique NAT devices indicates that only 11% have an endpoint independent mapping and filtering behavior [Müller and Klenk, 2010]. The rest of the NAT devices have an endpoint dependent mapping and filtering behavior, which makes it difficult for hosts behind these devices to directly exchange packets.

### 2.2.4 Protocols for Traversing NATs

Session traversal utilities for NAT (STUN) [Rosenberg *et al.*, 2008] is a protocol that can be used by an endpoint to determine the IP address and port allocated to it by a NAT. The protocol can also be used to check the type of NAT an endpoint is behind [MacDonald and Lowekamp, 2010] and to check direct connectivity between endpoints that may be behind different NAT devices. Traversal using relays around NAT (TURN) [Mahy *et al.*, 2010] is an extension of the STUN protocol that allows the endpoints behind two different NATs or firewalls to exchange packets through an intermediary. Interactive connectivity establishment (ICE) [Rosenberg, 2010] is a protocol that makes use of the STUN and TURN protocol to establish connectivity between two endpoints directly or through an intermediary.

STUN, TURN, and ICE do not require any special behavior from the NAT devices. Therefore, we use these protocols in designing p2p communication protocols. Protocols such as UPnP [UPnP Forum, 2010] require explicit signaling between applications and NAT devices and will not work with the installed base of NATs that does not support these protocols.

## 2.3 Percentage of VoIP Calls in the Internet Requiring a Relay

The percentage of VoIP calls in the Internet that require a relay depends on the call arrival patterns through restrictive NATs. The precise determination of this number is impossible without access to call logs of the client-server or peer-to-peer VoIP providers which they are reluctant to share. However, our conversation with people in the VoIP industry suggest that the ball park range for the percentage of calls needing a relay is between 15-30%. A recent survey of 1,787 NAT devices indicates that hosts behind approximately 30% of these devices cannot traverse the NATs using UDP or TCP [Müller and Klenk, 2010] implying that hosts behind two different such devices are not likely to directly exchange packets without an intermediary. If the VoIP call arrival distribution follows this NAT distribution, the percentage of calls requiring a relay falls within the range.

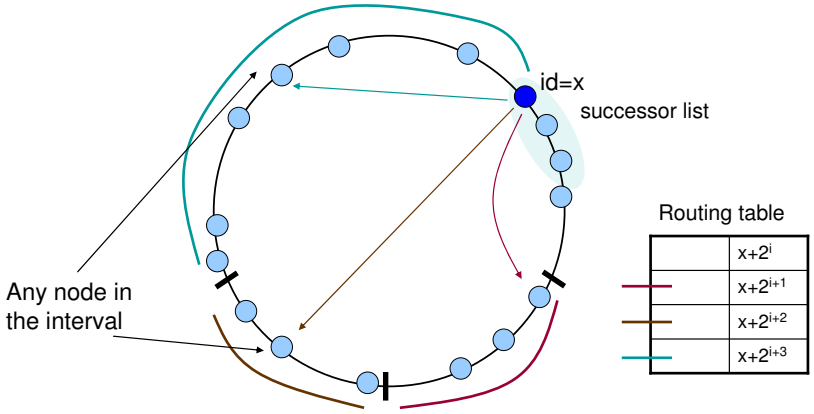



Figure 2.2: Chord DHT.

## 2.4 Overlay Algorithms

The overlay algorithms determine how nodes store and locate data in the overlay network. There are two types of overlay algorithms, structured and unstructured. Structured overlay algorithms guarantee a deterministic way of storing and locating data. They are suitable for finding a 'needle' in a hay-stack. On the other hand, the unstructured overlay algorithms do not guarantee a deterministic storage and lookup mechanism. Consequently, they are more suitable for grabbing hay from a hay stack, i.e., finding copies of a popular file in a p2p file sharing network.

Next, we briefly describe distributed hash tables (DHTs) which are a type of structured overlay algorithms. DHTs allow nodes to form an overlay network and to provide a distributed lookup and storage service similar to hash tables.

### 2.4.1 Distributed Hash Tables (DHTs)

We use Chord [Stoica *et al.*, 2003] to explain the underlying concepts of distributed hash tables. For a detailed survey of different overlay algorithms including DHTs, we refer the reader to RFC 4981 [Risson and Moors, 2007]. Figure 2.2 illustrates a Chord network. Each node in the Chord network has a fixed length identifier which is randomly chosen by a node or by a central authority. Similar to a hash table, the key space in Chord ranges from zero to the maximum value of the fixed length identifier. A node maintains two tables for maintaining connectivity to the Chord network, the **successor list**, and the **routing**

**table**. The successor list of a node $X$ contains the IP addresses and port numbers of nodes running the Chord protocol, that have numerically closest ID's to X. We refer to the successor list as the **neighbor table** of a node. The routing table of a node $X$ contains the IP addresses of $\log N$ Chord nodes, assuming there are $N$ nodes in the system. A node selects a node for its $i^{th}$ routing table row with an ID that lies between $[X+2^i, X+2^{i+1})$. A node uses the successor list to maintain a consistent view of the overlay whereas it uses the routing table to quickly send messages to other nodes in the Chord network. A message can traverse $\log N$ hops on average. The nodes in the Chord network have to periodically check the liveness of the nodes in their successor (neighbor) and routing tables. A node stores any key value pairs whose keys are between its predecessor's identifier and itself. When a node joins the Chord network, it takes ownership of a portion of the data that its successor stores. Similarly, when it leaves the Chord network, its successor must take ownership of the data stored by this departing node.

# Part I

# Protocol and System Design

# Chapter 3

# Protocols for Building Peer-to-Peer Communication Systems

## 3.1 Introduction

The research in peer-to-peer systems has focused on the design of structured or unstructured protocols [Stoica *et al.*, 2003; Ratnasamy *et al.*, 2001; Rowstron and Druschel, 2001a; Rhea *et al.*, 2004; Maymounkov and Mazieres, 2002; Chawathe *et al.*, 2003], file sharing [Rowstron and Druschel, 2001b; Rhea *et al.*, 2003], and streaming [Zhang *et al.*, 2005] that distribute the functionality of servers to nodes. The designers of these protocols, more often than not, need to reinvent mechanisms for data model, message reliability, security, and NAT and firewall traversal. Such reinvention increases the time to build and deploy a p2p system. Further, many of the above referenced protocols ignore the issue of NATs and firewalls altogether which is central to the deployment of a p2p communication system.

Skype [Skype, 2010a] is the first peer-to-peer VoIP application that enables user agents to establish media sessions with minimal use of managed servers. It does so by distributing the directory service, proxy server, and media relaying functions to the Skype user agents. Specifically, the Skype user agents cooperate to provide a distributed directory service for locating the network address of other Skype user agents, and for exchanging signaling messages to establish a media session. They can then directly exchange media traffic such as voice, video, or IM. However, restrictive NATs and firewalls may prevent them from directly

exchanging packets. The Skype network enables media session establishment between these user agents by using other Skype nodes with unrestricted Internet connectivity to relay the signaling and media traffic between these user agents.

Although Skype works, it uses a proprietary and encrypted protocol and requires Internet access to connect to the Skype's p2p network. Consequently, it cannot work in environments with no Internet connectivity. Further, our research has shown that the success rate of Skype media sessions depends on the characteristics of a network connection such as a NAT, and the selection of a Skype user agent to relay media traffic is suboptimal [Kho *et al.*, 2008]. Moreover, the relaying of media session consumes network bandwidth on the machine running the Skype application. Our conversations with Skype users suggest that at times they have become annoyed with the use of their machine's resources by the Skype application and decided to terminate it. This action can result in the failure of the relayed media session.

The thesis devises an open, standardized, and interoperable protocol for building peer-to-peer communication systems that is motivated by the desire to prevent the reinvention of mechanisms for data model, message reliability, security, and NAT and firewall traversal while at the same time, keeping the protocol extensible and flexible for non-VoIP uses. The protocol facilitates the design and development of VoIP systems that require minimal or no use of managed infrastructure and doing so under many different and often conflicting requirements of network connectivity, scalability, resource and service discovery, reliability, monitoring and diagnostics, and security. The protocol is extensible, allows incorporating a p2p protocol (which we refer to as overlay algorithm (Section 3.2)) for file-sharing, streaming, or VoIP, and reuses the same protocol machinery for data model, message reliability, security, and NAT and firewall traversal for different overlay algorithms, thereby preventing reinvention of this machinery for each overlay algorithm.

The rest of this chapter is organized as follows. In Section 3.2, we describe the requirements of designing such a protocol. In Section 3.3-3.6, we present Peer-to-Peer Protocol (P2PP) that we have designed. The protocol can be used to build p2p communication systems for ad hoc, enterprise, and Internet scale environments. In Chapter 4, we present OpenVoIP, a p2p communication system that we have built using P2PP that implements

three different DHTs. As shown in Section 4.4, 85% of the total lines of code (approximately 16,000) are independent of the overlay algorithm implemented using P2PP. This result confirms our assertion of keeping the same protocol machinery for data model, message reliability, security, and NAT and firewall traversal for different overlay algorithms.

## 3.2 Requirements

The goal of a peer-to-peer communication protocol is to enable media session establishment with minimal or no use of managed servers. Below, we describe the main requirements for designing an open, standardized, and interoperable p2p communication protocol for building p2p communication systems that can be deployed in ad hoc, enteprise, and Internet scale environments.

1. **Incorporating Different Overlay Algorithms** The p2p communication protocol should allow to incorporate different overlay algorithms.

   **Description** The p2p communication system can be deployed in enterprise and home networks (SOHO), emergency and ad hoc situations, mobile devices, and over the Internet. In the emergency and ad hoc situation, the quick establishment of the communication is paramount; in mobile devices, the conservation of devices resources such as battery is important; whereas in the Internet deployments, scaling to millions of nodes is necessary. The diverse deployment and scalability requirements imply that one overlay algorithm may not be suitable for all deployments. On the other hand, many deployments of the p2p communication protocol need mechanisms for NAT and firewall traversal, security, and connectivity. Further, we desire to keep the protocol extensible for non-VoIP usages. Therefore, the p2p communication protocol should have a mechanism to incorporate an appropriate overlay algorithm to meet the diverse scalability requirements. At the same time, it should provide mechanisms for NAT traversal, security, and connectivity that are independent of different overlay algorithms.

2. **NAT and Firewall Traversal** The p2p communication protocol should distribute the functionality of NAT and firewall traversal servers to the endpoints or peers. Further, the protocol should facilitate the discovery of peers providing the NAT and firewall traversal service.

   **Description** This requirement is one of the main reasons for designing a peer-to-peer communication protocol for VoIP. A VoIP provider that uses managed servers to relay media sessions between user agents that are behind restrictive NATs and firewalls incurs an economic and administrative overhead. A p2p communication protocol and system can reduce this overhead by distributing the functionality of these servers to the endpoints.

   A peer with NAT and firewall traversal capabilities should be selected such that it does not increase the delay of the media session.

3. **Resilience** A p2p communication system must continue to function as peers arrive, depart, and fail. The design of the protocol should not make any assumptions about the uptime of the peers.

   **Description** The peers will run on the machines of end users. These users can turn off their machines or p2p application at anytime. The protocol should provide mechanisms for dealing with the departing peers so that the functions performed by them can be transferred to other peers in a seamless way.

4. **Security** The protocol should provide mechanisms for preventing fake identity creation of users and nodes, and for message confidentiality and data integrity.

   **Description** When running p2p communication protocol in untrusted environments, rogue users can prevent legitimate nodes from fully participating in the overlay by creating fake node and user identities. Such an attack is known as Sybil attack [Douceur, 2002]. The protocol should provide mechanisms to protect against Sybil attacks. Further, to prevent the nodes from snooping in the traffic or modifying the stored data in

the overlay, the protocol should provide mechanisms for message confidentiality and data integrity. These mechanisms should be independent of the overlay algorithm.

5. **Service Model** The protocol should support the addressing, storage, and discovery of heterogeneous data (resource) in the overlay. The overlay treats the stored data as opaque; only the appropriate p2p application can interpret it. Further, the protocol should be flexible to allow new resources to be stored in an existing p2p network.

   **Description** NAT and firewall traversal servers, voicemail, address book, and configuration storage are examples of heterogeneous resources and services which peers can provide. However, even in a p2p communication system, managed servers can provide these services. The protocol should provide mechanisms for addressing, storing, and discovering heterogeneous resources and services that are either provided by peers or by managed servers.

6. **Namespace** The protocol should allow centralized and decentralized naming authorities.

   **Description** A centralized naming authority is needed to guarantee a unique namespace for users, nodes, resources, or services. In the absence of any naming authority, the protocol and the system should be able to determine if the identifiers of users, nodes, resources, or services collide and how to resolve collisions.

7. **Reusing Existing Protocols** The p2p communication protocol should reuse existing protocols where possible.

   **Description** Session Initiation Protocol (SIP) [Rosenberg *et al.*, 2002] is a protocol for establishing a session between user agents. TLS [Dierks and Rescorla, 2008] and DTLS [Rescorla and Modadugu, 2006] protocols can provide message confidentiality over a reliable and unreliable transport. STUN [Rosenberg *et al.*, 2008], TURN [Mahy *et al.*, 2010], and ICE [Rosenberg, 2010] are protocols for enabling connectivity between endpoints that may be behind NATs. The p2p communication protocol should

reuse these protocols because they are well defined for accomplishing their respective tasks and partially or completely reinventing them will result in a complex protocol that is difficult to implement.

8. **Protocol Overhead** The byte and message overhead of the protocol should be minimal.

   **Description** The protocol is envisioned to be run on low capability mobile devices such as WiFi phones and wireless enabled cameras as well as more resourceful devices such as desktop PC's. A protocol which has a large byte and message overhead can impact the device resources. Consequently, the user will be less inclined to run applications based on such a protocol.

9. **Interconnect with Other Communication Networks** The p2p communication protocol and system should facilitate interconnection with other p2p and client-server VoIP systems, circuit switched PSTN, and cellular networks, and facilitate their discovery in a distributed or a centralized manner.

   **Description** A user of a p2p communication system may need to establish media sessions with users in other communication networks such as PSTN. The protocol should facilitate this interconnection.

10. **Non-VoIP Usages** The p2p communication protocol should be reasonably flexible so that it may be used to implement other p2p systems such as content distribution or streaming.

    **Description** Any peer-to-peer protocol and system such as file sharing and streaming needs mechanisms for data model, message reliability and confidentiality, data integrity, and NAT and firewall traversal. If a p2p communication protocol provides a flexible data model, and provides generalized mechanisms for message reliability and confidentiality, data integrity, and NAT and firewall traversal, it can then be extended to support non-VoIP usages such as file sharing and streaming.

## 3.3 Peer-to-Peer Protocol (P2PP)

The thesis presents Peer-to-Peer Protocol (P2PP) [Baset *et al.*, 2007], an application layer request-response binary protocol that allows participating nodes to form an overlay using any structured or unstructured overlay algorithms. The design of P2PP exploits commonalities in the existing peer-to-peer protocols (which we refer to as overlay algorithms (Section 2.1)) such as Bamboo [Rhea *et al.*, 2004], Chord [Stoica *et al.*, 2003], CAN [Ratnasamy *et al.*, 2001], Pastry [Rowstron and Druschel, 2001a], Kademlia [Maymounkov and Mazieres, 2002], and Gia [Chawathe *et al.*, 2003] thereby defining a protocol that does not include details specific to any of these protocols and has mechanisms to incorporate a protocol-specific feature. P2PP defines mechanisms for NAT and firewall traversal, uses secure transport, and has mechanisms for exchanging node capabilities and diagnostic information that are independent of any overlay algorithm. P2PP allows non-participant nodes (clients) to use overlay services through participant nodes (peers).

In Section 3.4, we give a design overview of P2PP. In Section 3.4.3, we describe the key components of a P2PP message. We then describe the P2PP operations in Section 3.6.

## 3.4 Design Overview of P2PP

This section describes the entities defined by P2PP (node model), how the nodes store data (data model), the types of messages and their reliable delivery (message and reliability model), the security mechanisms of P2PP (security model), and NAT and firewall traversal mechanisms.

### 3.4.1 Node Model

P2PP defines two types of nodes, namely, **peers** and **clients**. A peer is a node participating in an overlay that provides storage and routing services to other nodes in the overlay. A client is a node that does not provide any storage or routing services. Instead, it communicates with one or more peers to use the storage and routing services provided by them. A peer can participate in more than one overlay and a client can also communicate with peers in different overlays. Each overlay is identified by a fixed length identifier which we

refer to as **overlay-id** (see Appendix A.15 for bit-level details).

P2PP does not define a priori which nodes can act as peers or clients and leaves the decision to the designer of the p2p communication system. The rationale is that P2PP can be used to form an overlay of participating nodes in many different network environments such as Internet, enterprise, and mobile networks. The nodes which are suitable candidates for becoming a peer in one environment may not necessarily be suitable in the other environment. The CPU and memory usage, uptime, and network connectivity of the machine running the p2p application are some of the metrics which the designer of the p2p application can use to determine whether a node should be a peer or a client. The classification of a node as a peer or a client is not permanent; a client can join the overlay as a peer and later be promoted to a peer after it has been online for sometime, or a peer can invite a client to join the overlay. Similarly, a peer can leave the overlay and join as a client.

In P2PP, peers and clients are identified by a **node-id** (Appendix A.1). The length of the node-id is fixed per overlay instance. The node-id for peers and clients is chosen from the same identifier space. To some extent, the assignment of a node-id to a node depends on the underlying overlay algorithm. For structured overlay algorithms such as DHTs, a node-id is a randomly chosen identifier and its length depends on the hash function used in the DHT. For unstructured protocols, the identifier can be chosen in any appropriate way. P2PP allows a central naming authority to select node-id's for peers and clients. Alternatively, peers and clients can choose their own node-id's; however, the possibility of collision arises. Further, the lack of a central authority for selecting a node-id opens the system to Sybil attack [Douceur, 2002]. To address these issues, P2PP defines an optional entity called an **enrollment and authentication (E&A) server**. The E&A server ensures that node identifiers are unique and verifiable, i.e., E&A issues a digital certificate for the node-id. In addition, the E&A server can also authenticate the users running the p2p application. A peer (typically, the first peer in the overlay) can also act as an E&A server.

Bootstrapping is a fundamental issue in the overlay as the nodes must discover the network address of other peers in order to join the overlay. There are many different mechanisms for discovering a peer already in the overlay [Cooper *et al.*, 2007]. P2PP defines an optional entity called the **bootstrap server** which provides the joining peers

and clients with the IP address of peers already in the p2p network. The bootstrap server may be co-located with the E&A server. A peer can also act as a bootstrap server. P2PP leaves it to the designer of the p2p communication system to decide how a bootstrap server should maintain a [sub]set of online peers in the overlay. In Section 4.3.1, we describe the mechanism that we have used in our OpenVoIP system.

For diagnostic purposes, an overlay operator may like to construct a 'map' of nodes. Such a map allows the overlay operator to identify problematic hot spots in the overlay and take remedial actions. P2PP defines an optional entity called the **diagnostic server**, which can query diagnostic information from nodes in the overlay.

Purists can argue that the use of enrollment and authentication server, bootstrap server, and diagnostic server violates the distributed nature of the overlay. While technically correct, the practical issues involved in running an overlay such as authenticating users, preventing Sybil attacks, bootstrapping in a timely manner, and finding problematic hotspots necessitate their usage.

#### 3.4.1.1 P2PP Node Stack

The protocol stack of a P2PP node has three conceptual layers, namely, usage, overlay, and transport layers. Figure 3.1 shows these three layers.

- **Application Layer** defines an asynchronous application-level API. An application can use this API to perform overlay operations such as joining or leaving the overlay, and storing and retrieving a resource-object.

- **Overlay Layer** is the 'brain' of P2PP protocol stack. It contains mechanisms for message routing, overlay maintenance, NAT traversal, storage management, and replication. The message routing component in a peer includes the appropriate overlay algorithm which determines how and where a peer should send the new or incoming messages in the overlay.

  The overlay layer receives API requests from the application layer and accomplishes the requested function. The routing mechanism routes the requests in a recursive, iterative, or parallel manner. The overlay maintenance mechanism strives to preserve

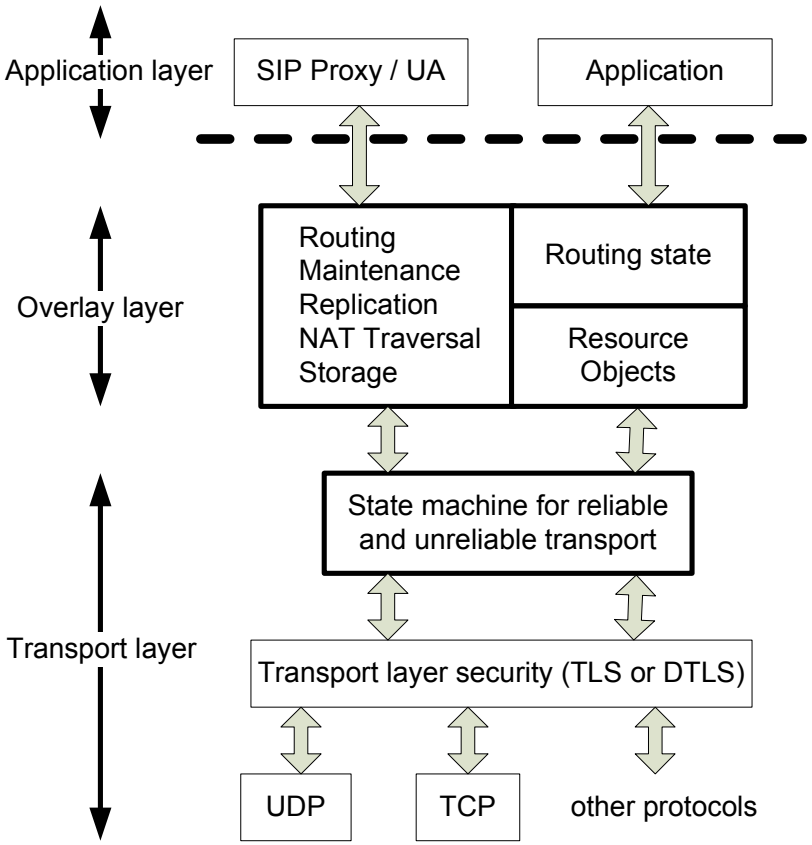



Figure 3.1: Protocol stack of a P2PP peer.

the routing correctness and connectivity in the presence of churn. The replication mechanism maintains the availability of resource-objects in the overlay. The NAT traversal mechanism uses STUN [Rosenberg *et al.*, 2008] and ICE [Rosenberg, 2010] to traverse NATs and firewalls. A client does not provide routing, storage, or replication services.

- **Transport Layer** is used by a P2PP node to send messages over an unreliable or a reliable transport. A node prefers the use of reliable transport such as TCP. For unreliable transports such as UDP, P2PP provides an ACK-based hop-by-hop reliability mechanism. For reliable transports, reliability is provided by the underlying transport protocol.

  P2PP uses Transport Layer Security (TLS) [Dierks and Rescorla, 2008] and Datagram-TLS (DTLS) [Rescorla and Modadugu, 2006] to provide message confidentiality.

**P2PP Node State:** A P2PP node maintains the following state:

**Overlay algorithm** and its relevant parameters for the overlay to which a node is connected.

**Overlay-ID** is the identifier of the overlay this node is part of.

**Routing table** contains the node-info objects (Section 3.5.3) of a subset of peers. It is only maintained by a peer. Each node-info object in the routing table contains node-id and host, server-reflexive, and relay IP addresses and port numbers of the other peer. In addition, a peer may also store information about the round-trip time (RTT) and uptime of peers in its routing table. A peer uses the routing table to quickly forward the message to the peer responsible for handling a message.

**Neighbor table** contains a set of node-info objects for peers having an identifier that are closest to the peer's node-id in structured overlay algorithms. Like routing table, it is also only maintained by a peer. This table ensures that a peer has a consistent view of the overlay, and guarantees that the messages will be forwarded to their correct destination. The neighbor table is known as successor list in Chord [Stoica *et al.*, 2003] and leaf set in Pastry [Rowstron and Druschel, 2001a].

**Publish table** contains the resource-objects (Section 3.4.2) that this node has published in the overlay. Each resource-object has an expiration time. A peer must republish the resource-object before its expiration time; otherwise, the peer storing the object will remove it.

**Resource table** includes the resource-objects that a peer stores on behalf of other peers to provide the distributed storage service. The clients do not store this table. A peer should expunge the resource-objects from this table that an expired time.

**Transaction table** keeps track of in progress requests, responses, and indications (Section 3.4.3). The retransmission of requests, responses, and indications is governed by a state machine defined in Appendix B.

**Number of clients** connected to a peer.

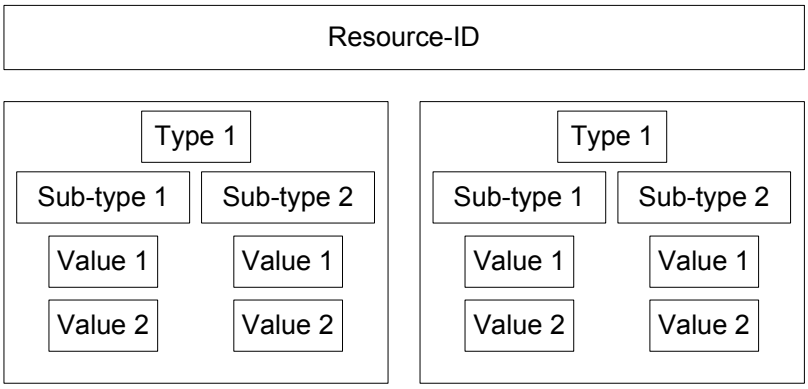



Figure 3.2: Resource-object.

**Media relay flows** include the number and bandwidth of media flows being relayed through this peer.

### 3.4.2 Data Model

In P2PP, the data to be stored is defined as a **resource-object** (see Appendix A.6 for bit-level details). A resource-object is identified by a **resource-id**, the type and sub-type of the value being stored, and the value itself. The resource-id is an identifier for the object, and the type and sub-type define a namespace, i.e., resource-objects having same resource-id but a different type and sub-type belong to a different namespace. The value of the object being stored depends on the type and sub-type. Examples of resource objects in a p2p communication system include the network addresses where a user is reachable, contact lists, and services such as NAT and firewall traversal. Among these examples, the reachable address of a user can have the same resource-id (such as email address) but different types, e.g., desktop phone, office phone. Figure 3.2 shows the relationship between the resource-id, type, sub-type, and value.

A resource-object also includes an **owner** object (Appendix A.10) to correctly identify the owner of the object (not shown in Figure 3.2). This object contains the node-id of the node publishing the object. Lastly, each resource-object also includes a cryptographic signature of the object to protect it against tampering. The mechanism for computing the signature is defined in Appendix A.14. P2PP allows peers to store and search resource-objects having the same resource-id, type and sub-type but different owners.

### 3.4.3 Message Model

All P2PP messages begin with a common header followed by a sequence of type-length-value (TLV) objects[1]. P2PP defines three types of messages, namely, request, response, and indications. The requests always generate a response whereas indications do not require any. Indications are useful in scenarios such as leaving the overlay or informing other peers about a routing table change. The response messages contain a response code which indicates if the request was successfully processed. The values of the response code are inspired by the HTTP and SIP response codes.

P2PP allows nodes to forward the requests in a recursive or an iterative manner. In recursive routing, a request is forwarded from one peer to the other until it reaches the peer responsible for handling the request as determined by the overlay algorithm. The response is then sent along the same path on which the request was received. If the request originator desires to directly receive the response from the node generating the response, it can set the **D** flag in the request-options object (Appendix A.16).

In iterative routing, if a peer determines that it is not the destination of the message, it sends in its response the IP address and port number of the next hop. The request originator then sends the request to the next hop. Peers never forward the indications. Figure 3.3 show a conceptual diagram of request routing in a recursive and iterative manner. P2PP does not mandate the use of recursive or iterative routing and leaves that decision to the designer of the p2p communication application. The designer can choose to use either recursive or iterative routing on a per overlay or a per message basis.

Note that the use of iterative routing in conjunction with a secure transport such as TLS and DTLS can be expensive as the request originator may have to establish a TLS or a DTLS connection with every next hop. The establishment of a TLS or a DTLS connection incurs a round-trip time (RTT) overhead of 1.5-2 RTT. Also, if the nodes are behind a NAT, then it may be necessary to perform connectivity checks before iterative routing, which also have an overhead between 2-4 RTT's. For these reasons, the nodes may prefer the use of recursive routing because it minimizes the RTT overhead associated with connection establishment and NAT traversal. On the other hand, the overlay can be constructed in a

[1]For simplicity, we refer to TLV objects as 'objects'

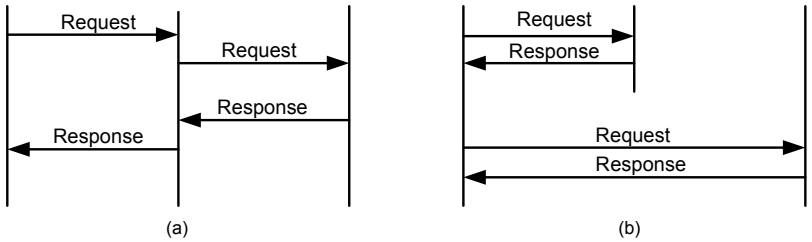



Figure 3.3: A conceptual diagram showing request routing in (a) recursive (b) iterative manner.

way so that only nodes that are not behind NATs are selected as peers. In this way, the peers do not have to perform the connectivity check before sending the message to other peers, although, they still incur the DTLS/TLS connection establishment overhead.

### 3.4.4 Reliability Model

P2PP defines a hop-by-hop reliability model, i.e., nodes are only responsible for ensuring the reliable delivery of the request, response, or indication to the next hop. In addition, the request or response originators need to ensure that the messages are delivered in a reliable manner. The reason for not using an end-to-end reliability model is that the intermediate nodes may go offline at any instant which may impact the timely delivery of the message since the request originator will only find it through retransmission timeouts. P2PP allows nodes to send messages over a reliable and unreliable transport. For unreliable transports, P2PP provides an acknowledgement (ACK) based mechanism for reliable message delivery (see Appendix B).

P2PP does not support message fragmentation and instead makes use of the underlying reliable transport such as TCP for sending messages larger than MTU. For recursive routing, the hop-by-hop reliability model allows the routing path to be a mix of reliable and unreliable transports.

Since each request generates a response, P2PP allows the acknowledgements and responses to be combined when using unreliable transports. Figure 3.4 shows the unoptimized and optimized iterative request routing and response generation over an unreliable transport.

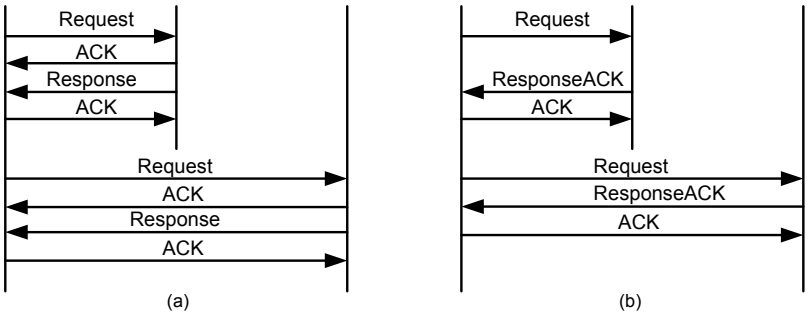



Figure 3.4: Request routing in an iterative manner over unreliable transport. (a) unoptimized forwarding (b) optimized forwarding.

### 3.4.5 Security Model

P2PP allows to create verifiable identities for nodes and users, and ensures message confidentiality by using secure transport protocols. A verifiable identity contains an identifier that is cryptographically signed by a trusted party. Without verifiable identities, the nodes may launch a Sybil attack [Douceur, 2002]. To prevent this attack, P2PP allows using an enrollment and authentication (E&A) server (Section 3.4.1) that assigns a unique and verifiable node-id to each node participating in the overlay and to the human user of the p2p communication system. It does so by issuing a certificate that binds the user and node-id to the public key of the user. P2PP also allows nodes to operate without a managed enrollment and authentication server. In such a scenario, nodes use self-signed certificates and generate their own identifiers.

P2PP provides a hop-by-hop security model. The nodes in P2PP establish an unreliable or reliable secure channel with other nodes using TLS [Dierks and Rescorla, 2008] or DTLS [Rescorla and Modadugu, 2006].

### 3.4.6 NAT and Firewall Traversal

P2PP enables any node with unrestricted connectivity to assist nodes behind restrictive NATs and firewalls to use overlay services and to establish media sessions. It does so by allowing nodes (peers and clients) to run STUN [Rosenberg *et al.*, 2008], TURN [Mahy *et al.*, 2010], and ICE [Rosenberg, 2010] protocols (see Section 2.2.4). Unlike protocols such as UPnP [UPnP Forum, 2010] which require explicit support from NATs, the STUN, TURN,

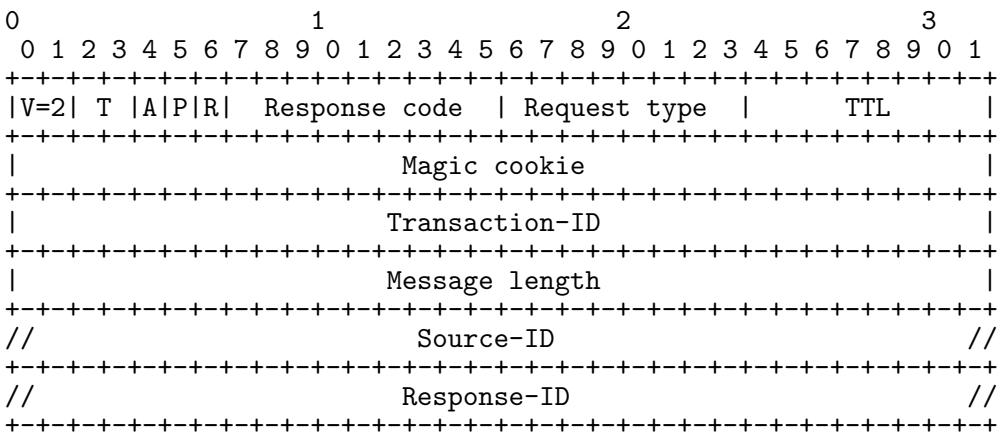



Figure 3.5: Common header of a P2PP message.

and ICE protocols can function without such support. P2PP does not restrict which nodes can provide the STUN and TURN service and leaves the decision to the system designer. Further, P2PP also leaves it to the system designer to devise a suitable mechanism for discovering nodes that provide the STUN and TURN service. As an example, a system designer may only allow nodes that are not behind NATs to provide the NAT and firewall traversal service.

## 3.5 Key Components of a P2PP Message

All P2PP messages begin with a common header, followed by a sequence of type-length-value (TLV) objects. This section describes the common header of a P2PP message, the general TLV object format, the node-info and the resource-object.

### 3.5.1 Common Header

The common header is a fixed length header which is part of every P2PP message. Figure 3.5 shows the bit fields of the common header. They are:

- **V – Version (2 bits)**: A field specifying the version of the protocol.
- **T – Message type (2 bits)**: A field which specifies the type of the message: a request (00), a response (01), or an indication (10).

- **A – Acknowledgement flag (1 bit)**: If set (A=1), the message is an acknowledgement to a request or a response.
- **P – Peer or client (1 bit)**: A flag set by the originator of the message to indicate whether it is a peer (P=1) or a client (P=0).
- **R – Recursive or iterative (1 bit)**: A flag set by the message originator indicating whether the message should be routed in a recursive or an iterative manner.
- **Response code (9 bits)**: The response code of the message. It is set to zero in requests and indications.
- **Request type (8 bits)**: The type of the request or indication such as Join or Leave.
- **TTL (time-to-live) (8 bits)**: The number of peers the request can traverse. It is set by the request originator and decremented by hops in the path of the request.
- **Magic cookie (32 bits)**: A field with a fixed value (0x596ABF0D) to differentiate P2PP messages from other protocol messages such as STUN [Rosenberg *et al.*, 2008].
- **Transaction-ID (32 bits)**: A unique number set by the request originator to match the responses with the originated requests. Together, with the source-ID, the tuple can uniquely identify a message in the system.
- **Message length (32 bits)**: The byte length of the message after the common header.
- **Source-ID (variable)**: The node-id object of the peer or client sending the request. The default length of this field is 160 bits. The overlays may use a different length node-id. The length of this field is determined when a node sends an Enroll or Bootstrap request. Since node-ids must be unique and nodes must choose a locally unique transaction-ID, the combination of source-ID and transaction-ID can uniquely identify a message.
- **Response-ID (variable))**: The node-id object of the peer or client sending the response or an acknowledgement.

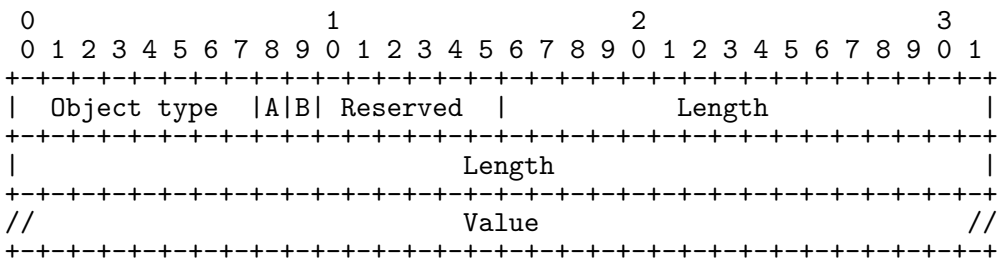



Figure 3.6: Common header of a TLV object.

### 3.5.2 General Object Format

Each P2PP object begins with a fixed six-byte header which identifies the object type and the length of the object. The header is followed by the variable length value of the object. The bit-fields of the object header are defined as follows:

- **Object type (8 bits)**: An identifier for the type of the object.
- **AB**: These bits are set by the message originator to specify how a node should process the objects.

  AB=00 "(Mandatory)": If the object is not understood, the entire message containing the object must be rejected with a 420 (Object Type Error) response.

  AB=01 "(Ignore)": If the object is not understood, it must be deleted and the rest of the message processed as usual.

  The combination AB=10 and AB=11 are reserved.
- **Length (32 bits)**: The byte length of the object.

### 3.5.3 Node-Info

Each node sending the request, response, or an acknowledgement includes a node-info object in the message (see Appendix A.2 for bit-level details). The node-info object contains a fixed length *node-id* object (Appendix A.1) and a set of reachable IP address and port numbers gathered using ICE [Rosenberg, 2010], which are encoded in the address-info object (Appendix A.3). A node may include its uptime information and self-signed or signed certificate objects (Appendix A.11) in the node-info object. The certificates can be used by

other nodes to establish a TLS or a DTLS connection with this node. Additionally, a node may also include information about the machine resources being consumed.

We use the following notation to define a TLV object and a P2PP method. The identifier on the left of '=' sign is the name of the TLV object or the P2PP method. The list of identifiers on the right of '=' sign are the concrete types and TLV objects contained in the object or the message. The objects within square brackets imply that they are optional. The 'ext' at the end of the object or the message implies that the object or message is extensible. We use lower case to refer to the objects in the text.

```
Node-Info = Node-ID
            Address-Info
            [Uptime]
            [Certificate]
            [Node-Resource-Utilization]
            [ext]
```

### 3.5.4 Resource-Object

A resource-object is a P2PP TLV object containing the information that the nodes store in the overlay. As discussed in Section 3.4.2, a resource-object has a resource-id, content-type, sub-type, and a value. The resource-object may have additional fields such as *owner*, the node which published the object; *expires*, the time after which the node storing the object can safely remove it; *signature*, the cryptographic hash over computed all the fields in the resource-object and signed by the publisher of the object. The owner, expires, and signature fields are defined in Appendix A.10, A.7, and A.14, respectively.

A resource-object is defined as follows:

```
Resource-Object = Cont-type
                  Sub-type
                  Resource-ID
                  Expires
                  Value
```

```
[Owner]
[ext]
Signature
```

## 3.6 Overview of P2PP Methods

P2PP methods (or operations) are grouped into five categories. They are (1) enrollment and bootstrap, (2) overlay maintenance, (3) data storage, (4) connection management, (5) monitoring and bandwidth measurement.

The enrollment and bootstrap category includes methods for obtaining a signed certificate from a central authority and to discover the network address of peers already in the overlay. The overlay maintenance category includes methods for joining and leaving the overlay, and for maintaining connections to other nodes in the overlay. The data storage category includes methods for storing and retrieving resource-objects in the overlay, and for replicating and transferring the objects to the responsible nodes. The connection management includes a method for setting up a connection for exchanging messages of application layer protocols such as SIP. The bandwidth measurement describes a method for helping peers determine their uplink and downlink network capacity by running a bandwidth test with other peers. The measurement of link capacity is useful in relay selection.

Each request, response, indication or acknowledgement message contains a node-info object which contains the node-id and address-info objects of the node sending the message.

### 3.6.1 Enrollment and Bootstrap

This section defines an Enroll method which a node uses to obtain a signed certificate and a Bootstrap method to discover the network address of a peer already in the overlay.

#### 3.6.1.1 Enroll

In P2PP, each node has a unique identifier known as node-id. Further, each user running the p2p communication application also has a human readable user identifier such as an email address, user name, or a SIP address-of-record (AoR) [Rosenberg *et al.*, 2002]. The

goal of the Enroll method is to have the user identifier, node-id, and the public key of the node cryptographically signed by a trusted central authority. The trusted authority can be an Enrollment and Authentication (E&A) server (Section 3.4.1); however, the first peer in the overlay or any other peer can also act as this trusted authority.

To enroll in an overlay, a user chooses an identifier and a password. The node or user agent then generates a public/private key pair, and embeds the user name in the distinguished name of a certificate signing request (CSR) [Nystrom and Kaliski, 2000]. The node must securely store the private key. The CSR is encoded using PKCS#10 [Nystrom and Kaliski, 2000]. The node then establishes a TLS connection with the E&A server (without the client certificate authentication) and sends the CSR and the hash of the password to the enrollment server in an Enroll request.

The node sending the Enroll request may not be aware of the length of the node-id or the overlay algorithm. Since every request and response must contain the source-ID, response-ID, and node-id field in the node-info object, P2PP specifies a default node-id of length 160 bits. When a node sends the Enroll request, it randomly chooses a 160 bit node-id and sets it in the source-ID field of the common header and node-id field of the node-info object.

If the request is authenticated, the enrollment server creates a random node-id and a signed X.509 [International Telecommunication Union (ITU), 2006] certificate encoded using distinguished encoding rules (DER) [International Telecommunication Union (ITU), 1997]. It embeds the node-id in the common name (CN) of the certificate along with the user identifier and sends this certificate in a 200 (Ok) response to the node that sent the Enroll request. As discussed in Sections 3.4.2 and 3.4.5, the main reason for the server to choose a node-id instead of the node randomly generating its own node-id is that it prevents the nodes from launching a chosen location attack. The nodes can launch the chosen location attack in structured overlays because the node-id determines the position in the overlay where a node joins. If the request fails, the E&A server replies with a 450 (Request Unauthorized) response.

A user may join the overlay from multiple devices such as a hardphone, a desktop phone, and a mobile device. P2PP places no restriction on whether a user can enroll from multiple devices. Therefore, for each device, the user agent generates a separate Enroll request,

and obtains a new certificate binding the user identifier to the node-id of that device. Conceptually, there is a one-to-many relationship between user identifiers and node-id's.

The objects contained in the Enroll request and its response are given below.

```
Enroll = Common-header
         [Request-Options]
         Node-Info
         Certificate-Sign-Request
         Password


Enroll (Resp) = Common Header
         Node-Info
         Certificate
         [ext]
```

#### 3.6.1.2 Bootstrap

To join an overlay, a **joining node (JN)** must determine the overlay algorithm used to form the overlay, and must discover a peer that is already part of the overlay, thereafter referred to as an **admitting peer (AP)**. There are different mechanisms to discover a node already in the overlay as specified by Cooper [Cooper *et al.*, 2007]. Any node in the overlay can provide the bootstrap functionality. Alternatively, the overlay provider may maintain a bootstrap server which keeps track of a subset of peers already in the overlay, and configures the nodes with the network address of the bootstrap server. As discussed in Section 3.4.1, P2PP defines an optional bootstrap server that a joining node can use to discover any node in the overlay. The bootstrap server can be co-located with the enrollment and authentication server or it can be any peer in the overlay.

To discover the overlay algorithm and peers already in the overlay, P2PP defines a Bootstrap method. The nodes may be preconfigured with the network address of the bootstrap server. Alternatively, a JN may send a Bootstrap request to a multicast address. Upon receiving a Bootstrap request, the bootstrap server or a node replies with a list of node-info objects representing online peers in the overlay, and a p2p-options object (Appendix A.15).

The p2p-options object contains information about the overlay algorithm being used. The JN then sends the Join request to one of the peers in the list. If the list is empty, then the joining peer is the first peer in the overlay.

If a JN has not sent the Enroll request before sending the bootstrap request, it may not be aware of the length of node identifier (node-id object) used by peers in the overlay. This situation arises when the nodes use the self-signed certificate security model. When sending a bootstrap request, a JN randomly selects a 160-bit node-id and includes it in the source-ID field of the common header and node-id of the node-info object. After it receives the response to the Bootstrap request, it can determine the length of node-id used by peers in the overlay. In the subsequent messages, the JN can randomly pick the same length node-id.

The TLV objects contained in the Bootstrap request and its response are given below.

```
Bootstrap = Common-header
            Node-Info

Bootstrap (Resp) = Common Header
            Node-Info
            P2P-Options
            [Node-Info]*
            [ext]

P2P-Options = Node-ID length
            Overlay-ID length
            Hash algorithm
            Base
            Overlay-ID
```

### 3.6.2 Overlay Maintenance

This section defines methods for joining and leaving the overlay (Join and Leave), for exchanging the routing and neighbor tables with other nodes (ExchangeTable), for exploring

the overlay to find a routing table candidate (LookupPeer), and to maintain liveness with other nodes (KeepAlive).

#### 3.6.2.1 Join

A JN discovers the overlay protocol and existing peers in the overlay (thereafter referred to as admitting peers (APs)) using the bootstrap mechanism described in Section 3.6.1. It then sends a Join request to the APs in the overlay. Before sending the Join request, a node may also send a LookupObject request (Section 3.6.3.2) to the APs to determine the STUN [Rosenberg *et al.*, 2008] and TURN [Mahy *et al.*, 2010] port of AP. The JN can then determine its server-reflexive and relay addresses by running connectivity checks using STUN protocol with one of the APs. Conceptually, the overlay nodes are connected in a graph, and a JN adds a new vertex to this graph and forms direct links with its immediate neighbors[2]. The JN must correctly inform its neighbor peers that it is about to join, and take ownership of any resource-objects from them. Overall, a node will likely follow the steps below to join the overlay:

1. Optionally determine its reachable address (such as server-reflexive) by discovering peers that provide the STUN [Rosenberg *et al.*, 2008] and TURN [Mahy *et al.*, 2010] service and performing connectivity checks with them.

2. Discover the peer(s) where the node will join by sending the Join request to the admitting peers.

3. Inform the neighbor peer(s) that JN will be their new neighbor or will join as a client.

4. Build the routing and neighbor tables if joining as a peer.

5. Obtain the resource-objects from its neighbor peers.

6. Publish any resource-objects in the overlay.

A JN can request the admitting peer to forward the Join request in a recursive or an iterative manner. It does so by setting the appropriate value of the R flag in the common

[2] we refer to immediate neighbors as **neighbor peers**.

header. A peer receiving the Join request with the recursive routing flag set (R=1) forwards it to the peer where JN will join (neighbor peer) or which might be closest to the peer where JN will join. A peer receiving the Join request with the iterative routing flag set (R=0) replies with a 302 (Next Hop) response if the JN will not be its immediate neighbor and includes a node-info object of the next hop peer.

If a peer receiving the request determines that the JN will be its neighbor, it sends a 200 (Ok) response to the node from which it received the request. The JN can then send an ExchangeTable request to the neighbor peer to retrieve its neighbor table. The neighbor table is a list of node-info objects and is only defined for structured overlays. As discussed in Chapter 2, we refer to Chord's successor list [Stoica *et al.*, 2003] or Pastry's leaf set [Rowstron and Druschel, 2001a] as a neighbor table.

Once the JN receives the neighbor table, it sends the Join request with the S flag set in the request-options object (Appendix A.16) to the peers in the received neighbor table. The Join request with the S flag set indicates to the overlay peers that the JN will be their new neighbor. It is up to the peer generating the 200 (Ok) response to decide, based on the overlay algorithm being used, which neighbors a joining peer may contact to insert itself appropriately in the overlay. The neighbors of this newly joining peer should update their neighbor and routing tables appropriately. If the overlay protocol does not require a neighbor table, then the JN does not send the Join request with the S flag set.

A JN may request to receive a copy of the routing table and neighbor table of the peers that receive the Join request. It does so by setting the R, N, and E flags in the request-options object (Appendix A.16). The nodes receiving the Join request with these flags set include their routing-table (Appendix A.19) and neighbor table (Appendix A.20) objects in the response. By receiving the routing or neighbor tables of the peers in the path of the Join request, the JN can quickly build its view of the overlay by filling its routing and neighbor tables. Once the JN has successfully joined the overlay as a peer, its neighbor peers transfer it the ownership of appropriate resource-objects, using a TransferObject request (Section 3.6.3.3).

**Rejecting or Deferring a Join Request:** The peers in the overlay may decide to reject a Join request with a 406 (Request Rejected) response because it does not satisfy the overlay policy for peers. The network connectivity (such as NAT) and node resources (such as CPU utilization) are examples of these policies. P2PP does not specify any policy for peer admission and defers this decision to the overlay operator.

The peer(s) in the overlay may not be willing to admit a JN with no history about its uptime. If so, they reply with a 407 (Join Request Deferred) response and an expires object. The expires object contains the time in seconds after which the peer can resend the Join request. The JN receiving this response should join the overlay as a client.

After the passage of time in the expires object, a client may attempt to join again as a peer. Also, the AP or other peers may explicitly invite a client to join the overlay before the passage of time in the expires object by sending an Invite request (Section 3.6.2.6).

**Client Join:** A client enrolls and authenticates itself and discovers a peer already in the overlay by using mechanisms defined in Section 3.6.1. It gathers its reachable addresses by discovering a peer providing the STUN and TURN service, and sends a Join request without setting the P (peer) flag in the common header. The peer receiving the Join request may immediately reply with a 200 (Ok) response if it can support a new client. It is up to the overlay implementer to decide how many clients may connect to a peer. If an AP determines that it already has a certain number of clients, it replies with a 302 (Next Hop) response containing a list of peers to which a client may send the Join request.

After receiving a 200 response to its Join request, the client may establish a TCP connection with the peer and periodically exchange KeepAlive messages (Section 3.6.2.5) to check the liveness of its connected peer.

The objects contained in the Join request and its response are given below.

```
Join = Common-header
       [Request-Options]
       Node-Info

Join (Resp) = Common Header
```

```
        Node-Info
        [Node-Info]  //the next hop object
        [Expires]
        [Routing-table]
        [Neighbor-table]
        [ext]

Routing-table = Num-entries
        [Node-Info]+

Neighbor-table = Num-entries
        [Node-Info]+
```

#### 3.6.2.2 Leave

A peer sends a Leave indication to gracefully inform its routing or neighbor peers about its departure. It includes in the Leave indication a list of resource-objects that its neighbor nodes should take over. The peers receiving a Leave indication must update their routing or neighbor tables appropriately.

```
Leave = Common-header
          Node-Info
          [Resource-object]*
```

#### 3.6.2.3 ExchangeTable

A peer sends the ExchangeTable request to retrieve the routing or neighbor table of another peer. The peer receiving the request sends its routing table or neighbor table as routing-table and neighbor-table objects in a 200 (Ok) response. Each table object contains a list of node-info objects.

```
ExchangeTable = Common-Header
                Node-Info
```

```
                Request-Options

ExchangeTable (resp) = Common-Header
                Node-Info
                [Routing-table]
                [Neighbor-table]
```

#### 3.6.2.4 LookupPeer

As discussed in Chapter 2, a peer maintains a routing and neighbor table which contains the network addresses of other peers in the overlay. A peer selects the peers for its routing and neighbor tables on the basis of the overlay algorithm being used. As an example, in Chord [Stoica *et al.*, 2003], a peer with an id X selects a peer for its $i^{th}$ routing table row such that the id of the selected peer lies between $[X + 2^i, X + 2^{(i+1)})$. For unstructured overlays, a peer may select peers for its routing and neighbor tables based on the capabilities of the machines running those peers and their network connectivity [Chawathe *et al.*, 2003].

P2PP defines a LookupPeer request which allows a peer to discover the correct peers (based on the overlay algorithm) to fill its routing or neighbor table. The LookupPeer request contains the plookup object (Appendix A.17), which specifies the criteria according to which a peer searches for other peers. The LookupPeer method and the plookup object is customized for every overlay algorithm.

A peer can issue a LookupPeer request to locate a single peer, multiple peers or range of peers. The peer sending a 200 (Ok) response for the LookupPeer request includes a list of node-info objects. The number of node-info objects are controlled by the 'num' field in the plookup object. The R flag in the plookup object specifies that a peer is searching for node(s) with node-id's that lies within a range of ID's. The range search is used for structured overlays.

```
LookupPeer = Common-Header
             [Request-Options]
             Node-Info
             PLookup
```

```
LookupPeer (resp) = Common-Header
             Node-Info
             [Node-Info]*
             [ext]
```

#### 3.6.2.5 KeepAlive

P2PP defines a KeepAlive method which nodes can use to check the liveness of other nodes. The KeepAlive, ExchangeTable and LookupPeer methods must be specified for any overlay algorithm being used. By defining objects for these methods relevant to any DHT or unstructured protocol, P2PP can potentially be used to implement them.

```
KeepAlive = Common-Header
            [Request-Options]
            Node-Info
            [ext]

KeepAlive  (resp) = Common-Header
            Node-Info
            [ext]
```

#### 3.6.2.6 Invite

A peer in the overlay can send an Invite request to any client, requesting it to become a peer. A peer may choose a client for sending the Invite request based on its uptime and its network connectivity information.

```
Invite = Common-header
         Node-Info

Invite (Resp) = Common-header
         Node-Info
```

### 3.6.3 Data Storage

P2PP defines PublishObject and LookupObject methods to store and retrieve data from the nodes in the overlay. It also defines the TransferObject and ReplicateObject methods to transfer the ownership of resource-objects to a newly joined peer, and to replicate the resource-objects stored by a peer.

#### 3.6.3.1 PublishObject

A node sends a PublishObject request to a peer already in the overlay to publish a new resource-object or update an existing resource-object. A node typically sends a PublishObject request in response to a application layer API call. The resource-object has a content-type and a sub-type. The publish operation involves locating the peer responsible for the resource-object and then storing the object on that peer.

The publisher of the resource-object must also include information about its owner in the request. It is possible for multiple owners to publish or update a resource-object with the same resource-id, and peer may be interested in retrieving the resource-object that is published or updated by a certain owner. For example, a user logged in from different phone devices (e.g., hardphone, desktop phone) may publish the network address of the devices with the same resource-id, which is the user name. Moreover, other users may only be interested in calling, say, the hardphone, of the user, and thus may only retrieve the resource-object published by the user's hardphone.

A publisher must ensure that the integrity of the resource-object is preserved. It does so by computing a digital signature over all fields of the resource-object (except for the length field in the object header), and attaches the computed signature and relevant certificates as a signature object (Appendix A.14) at the end of the resource-object.

The peer responsible for storing the resource-object replies with a 200 (Ok) response and includes an expires object in the response. The expires object indicates the time before which the publisher must re-publish the object.

```
PublishObject = Common-header
            [Request-Options]
```

```
            Node-Info
            Resource-Object

PublishObject (Resp) = Common-header
            Node-Info
            [Expires]
```

#### 3.6.3.2 LookupObject

P2PP defines a LookupObject method to retrieve a resource-object from the overlay. A node issuing the LookupObject request specifies the resource-id of the object that it is searching for, and the type and sub-type of the objects in the rlookup object (Appendix A.18). Additionally, the node may specify the owner of the object by setting the owner object in rlookup object.

If the peer receiving the request stores the object being searched, it replies with a 200 (Ok) response which contains the resource-object. If multiple owners had published data under the resource-id being searched, and if the owner is not specified, the peer storing the resource-object includes all objects in its response.

The resource-object size may exceed the MTU if the request was sent over an unreliable transport protocol. The node responsible for the resource-object replies with a 480 (Alternate Service) response. On receiving this response, the request originator sends the LookupObject request over TCP to the peer responsible for the resource-object.

```
LookupObject = Common-header
               [Request-Options]
               Node-Info
               RLookup

LookupObject (resp) = Common-header
               Node-Info
               Resource-Object
```

#### 3.6.3.3 TransferObject

A peer sends a TransferObject request to a newly joined peer to transfer the ownership of the appropriate resource-objects to that peer. The resource-objects considered for transfer are determined according to the overlay algorithm being used. For each resource-object being transferred, the peer includes an elapsed object (Appendix A.8) at the end of the resource-object after the signature object. The object indicates the time elapsed since the node was storing the object. Together with the expires object in the resource-object, a node receiving the resource-objects can determine how long to keep the objects before safely removing them.

```
TransferObject = Common-Header
                 [Request-Options]
                 [Resource-object]+


TransferObject (resp) = Common-Header
                 Node-Info
```

#### 3.6.3.4 ReplicateObject

A peer storing the resource-objects may decide to replicate them in case it may go offline. P2PP defines a ReplicateObject method to replicate the objects on near-by nodes. P2PP does not specify any replication policy and leaves the decision to the designer of the p2p communication application. The reason is that it is impossible to determine a priori a suitable replication for an environment in which P2PP will run. However, in practice, simple replication strategies such as replicating the object on the adjacent neighbor of the node storing the object may be sufficient.

```
ReplicateObject = Common-Header
                  [Request-Options]
                  Resource-Object


ReplicateObject (resp) = Common-Header
```

```
            Node-Info
```

### 3.6.4 Connection Management

#### 3.6.4.1 Tunnel

In addition to exchanging P2PP messages, a node may desire to exchange other application layer protocol messages with the overlay nodes. Since the other nodes may be behind NATs and firewall, a node gathers its address candidates where it will receive the protocol messages, encodes them in an address-info object, and exchanges them in a Tunnel method with the destination node. The destination node also replies with its address candidates. Then, the two nodes perform ICE [Mahy *et al.*, 2010] connectivity checks and can exchange the application layer protocol messages.

```
Tunnel = Common-header
         [Request-Options]
         Node-Info
         Address-Info


Tunnel (Resp) = Common-header
         Node-Info
         Address-Info
```

### 3.6.5 Monitoring and Bandwidth Measurement

This section defines GetDiagnostics method for obtaining the diagnostic information from a node and a MeasureBandwidth method which nodes can use to run bandwidth tests.

#### 3.6.5.1 GetDiagnostics

The overlay operator may need to obtain diagnostic information from nodes, to construct a geographical or logical map, and to identify problematic hotspots in the overlay. P2PP defines a GetDiagnostic method which allows an overlay operator to query diagnostic information from nodes. The object-req (Appendix A.21) lists the type of the objects that a

node should return. The node then returns the requested TLV objects.

```
GetDiagnostics = Common-header
          [Request-Options]
          Node-Info
          [Object-Req]+


GetDiagnostics (Resp) = Common-header
          Node-Info
          [Objects]+
```

#### 3.6.5.2 MeasureBandwidth

To determine if it can relay a voice or video session, a peer needs to be able to measure its uplink and downlink network bandwidth. The idea is that peers help other peers measure their uplink and downlink capacity in addition to providing the routing, storage, and media relaying services. If a peer wants to measure its uplink and downlink capacity, it sends a MeasureBandwidth request to another peer. The request includes the bwtest object (Appendix A.22), which describes the type of the test, the direction of the test (uplink, downlink, both), the duration of the test, and the IP address and port numbers on which this node listens for the test. P2PP currently defines two capacity measurement tests, namely, TCP throughput and PathChar [Jacobson, 2010]. Other network capacity measurement tests can be easily incorporated.

If the peer receiving the request is able to run the test, it replies with a 200 (Ok) response, and includes the network address and port number on which it will run the test. The peers can then run the test for the specified duration and measure their network capacity.

```
MeasureBandwidth = Common-header
          [Request-Options]
          Node-Info
          BWTest
          [ext]
```

```
MeasureBandwidthTunnel (Resp) = Common-header
         Node-Info
         BWTest
         [ext]

BWTest = Type
         Direction
         Duration
         Address-Info
```

### 3.6.6 Response and Error Codes

P2PP defines three types of response codes. They are grouped as follows:

- 2xx (Success) response indicates that the request has been successfully processed.
- 3xx (Redirect) responses indicate that the request should be redirected to another peer.
- 4xx (Request Failure) responses indicate that the request has failed.

#### 3.6.6.1 2xx (Successful) Responses

- 200 (Ok) is a successful answer to the request.

#### 3.6.6.2 3xx (Redirect) Responses

- 302 (Next Hop) response is only generated for iterative requests if the peer receiving the request is not the final destination for the request.

#### 3.6.6.3 4xx (Failure) Responses

- 400 (Bad Request) indicates that there was an error parsing the request.

- 404 (Not Found) response is generated for a LookupObject request if the resource-object being searched for is not found.
- 405 (Error Storing Object) specifies that there was an error storing the resource-object.
- 406 (Request Rejected) indicates that the request was understood but rejected by the peer.
- 407 (Join Request Deferred) signals that the peer sending the Join request should retry after the elapsed time.
- 410 (TTL Hops) specifies that the number of TTL hops traversed by the request have exceeded the specified TTL value.
- 413 (Message Too Large) indicates that the response message size was too large. This response is typically generated for unreliable transports.
- 418 (Timeout) signals that the request timed out. This response is generated by a peer when request was forwarded in a recursive manner over UDP and no response was received from the downstream peers.
- 420 (Object Type Error) response is generated when the node processing the message cannot understand any of the TLV objects in the request.
- 450 (Request Unauthorized) signals that the enrollment and authentication server cannot verify the Enroll request.
- 460 (Busy) indicates that the peer is busy and cannot provide the service.
- 480 (Alternate Service) specifies that the peer sending the message should try an alternate transport service such as TCP.

## 3.7 Related Work

The research in p2p protocols has focused on designing structured and unstructured protocols [Stoica *et al.*, 2003; Ratnasamy *et al.*, 2001; Rowstron and Druschel, 2001a; Maymounkov and Mazieres, 2002; Chawathe *et al.*, 2003; Rhea, 2005], file sharing [Rowstron

and Druschel, 2001b; Rhea *et al.*, 2003; BitTorrent, 2010], and streaming [Zhang *et al.*, 2005]. A common issue facing these protocols is that they have to provide a data model, and mechanisms for data integrity, message reliability and confidentiality, and NAT and firewall traversal which they unfortunately have to reinvent. Moreover, the issue of NAT and firewall traversal, which is central to the VoIP call establishment, is altogether ignored by these protocols. As an example, OpenDHT [Rhea, 2005] only lets nodes with a public IP address to be part of the overlay. The filesharing applications such as Bittorrent [Dessent, 2010] address the NAT and firewall traversal problem by reducing the download rate of files for nodes that are behind NATs and firewalls and by instructing the user to enable port-forwarding on NAT devices. Reducing the rate is not an option for VoIP systems because it can impact the quality of the calls and it is unrealistic to expect ordinary users to configure NAT devices for port-forwarding.

Dabek *et al.* [Dabek *et al.*, 2003] were the first to propose a common API for structured overlays which exploited the commonalities in DHTs. The core function of their proposed API is a route() function which is responsible for delivering a message to the next hop. They outlined how different DHT operations such as put() and get() can be implemented using their API. However, they did not define a message format, nor did they specify any mechanisms for data integrity, message reliability and confidentiality, and NAT and firewall traversal.

Singh [Singh, 2006] and Bryan *et al.* [Bryan *et al.*, 2005] proposed a peer-to-peer version of the SIP protocol that uses SIP messages to implement a DHT. However, their approach is inherently tied to SIP and is not easily extensible for non-SIP applications.

P2PP is the first protocol to exploit commonalities in existing structured and unstructured p2p protocols, and it not only provides an API similar to Dabek's, but also provides mechanisms for data model and integrity, message reliability and confidentiality, and NAT and firewall traversal that are independent of any structured or unstructured protocol. Thus, it can lower the barrier to design and implement a new p2p protocol and prevent the reinvention of the above mechanisms in a new p2p protocol. This feature can enable P2PP to be used for building p2p systems such as streaming and video-on-demand. P2PP is now part of the RELOAD [Jennings *et al.*, 2010] protocol which is being standardized in the

IETF.

## 3.8 Conclusion

We have defined and described Peer-to-Peer (P2PP) protocol, an application layer protocol for building peer-to-peer communication systems. P2PP allows to incorporate any structured or unstructured protocol as an overlay algorithm by defining a small set of methods that implement the non-common aspects of these protocols. By explicitly defining a notion of peers (nodes that fully participate in the overlay) and clients (that connect to one or more peers), P2PP allows a system designer to build an arbitrary hierarchy of nodes. Peers can provide the storage, routing, NAT traversal, and media relaying service. NAT traversal is a fundamental feature of the protocol.

P2PP uses a hop-by-hop reliability model. It makes use of the TLS and DTLS as the secure transport and provides integrity of the data being stored. These features are independent of any structured or unstructured protocol. We believe that P2PP can not only be used to build p2p communication systems for diverse deployment environments (such as ad hoc or Internet scale), but can also be extended for building file-sharing, streaming or video-on-demand systems.

# Chapter 4

# OpenVoIP – A Peer-to-Peer Communication System

## 4.1 Introduction

The thesis presents OpenVoIP [Baset and Schulzrinne, 2008], an open source peer-to-peer communication system that has been deployed on the PlanetLab [PlanetLab, 2010]. The goals of building this system are (1) to demonstrate the feasibility of using Peer-to-Peer Protocol [Baset *et al.*, 2007] for building p2p communication systems (2) to show that a common protocol can be used to implement different overlay algorithms (3) to demonstrate an Internet scale p2p communication system, (4) to establish media and IM sessions between nodes in the presence of NATs and firewalls by using other peers as a relay, (5) to show a diagnostic and monitoring mechanism for p2p communication systems.

The rest of the chapter is organized as follows. Section 4.2 describes the system architecture of OpenVoIP and the resource-objects it defines for the system function. Section 4.3 discusses the various functions performed by nodes in the OpenVoIP system, such as bootstrap, call establishment, relay search, and monitoring and diagnostics. Section 4.5 discusses related work.

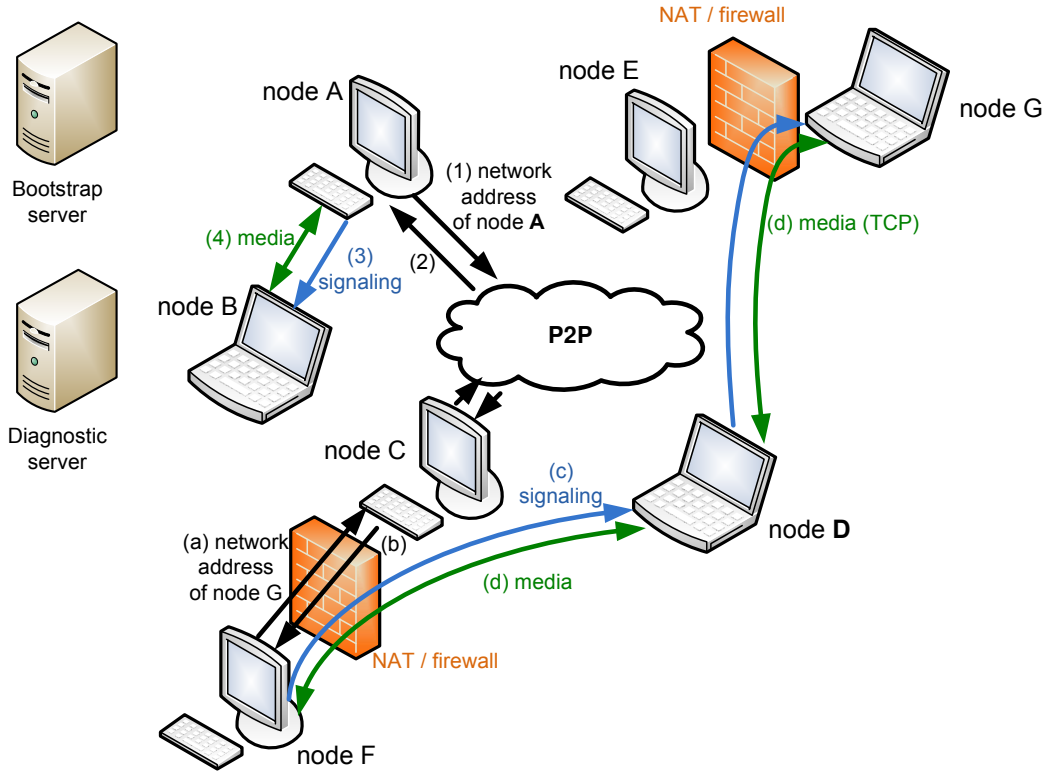



Figure 4.1: The OpenVoIP system.

## 4.2 System Architecture

OpenVoIP has four types of entities; a *peer*, which participates in the overlay, stores records, routes requests, relays signaling and media traffic, and helps other nodes determine their network capacity by running a bandwidth test; a *client*, which is connected to one or more peers, does not route requests or store records, and may require a relay to establish a media session; a *bootstrap* server, which stores and returns the IP addresses and port numbers of a subset of peers in the overlay to the joining peers and clients; and a *diagnostic* server, which monitors the status of the nodes in the system. The nodes behind NAT and firewalls join the overlay as clients; otherwise, the nodes join the overlay as peers. The joining nodes perform connectivity checks with the peers in the overlay to determine if they are behind a NAT. Peers, client, bootstrap server, and diagnostic server run the Peer-to-Peer Protocol defined in Chapter 3. Figure 4.1 shows the architecture of the OpenVoIP system. Node A discovers the network address of node B by performing a search in the overlay for the user

name of node B using the LookupObject request and then directly exchanges signaling and media traffic (messages labeled (1)-(4)). The nodes F and G are behind a NAT/firewall device and connect to the nodes C and E, respectively, which act as peers in the system. The nodes C and E use node D (peer) for exchanging signaling and media traffic (messages (a)-(d)). The peers and clients run as a stand alone executable on the PlanetLab machines. Unlike OpenDHT [Rhea, 2005], where only PlanetLab nodes fully participate in the overlay by providing the routing and storage services, any node running P2PP can fully participate in the OpenVoIP as long as they are not behind a NAT. Note that P2PP does not place any restriction on which nodes can be peers or clients. The choice of using only non-NAT nodes as peers is a design decision of OpenVoIP and is motivated by avoiding the complexity of having nodes behind NATs fully participate in the overlay.

We have also integrated the P2PP library with OpenWengo [OpenWengo, 2010], an open source SIP phone (now known as Qutecom [Qutecom, 2010]). We refer to this application as the Wengo-P2PP phone. Any user can run the Wengo-P2PP phone, which, depending on the absence or presence of a NAT device, can join the OpenVoIP as a peer or a client, respectively. The Wengo-P2PP phone uses the Session Initiation Protocol (SIP) [Rosenberg *et al.*, 2002] for session establishment and Real-time Transport Protocol (RTP) [Schulzrinne *et al.*, 2003] for exchanging media traffic. Figure 4.2 shows a snapshot of the Wengo-P2PP phone. The user running the application, *bob2*, has established a voice session with user *alice*. The voice packets go through a media relay (indicated by the '(relay)' in the figure), since their respective user agents are unable to directly exchange voice packets due to NAT connectivity issues.

### 4.2.1 Resource-Objects for Operation

OpenVoIP defines two resource-objects (Section 3.5.4) for its operation, namely, SIP-CONTACT and STUN-TURN. The SIP-CONTACT object contains the SIP address-of-record (AoR) of a user (e.g., sip:bob@example.com) and the set of network addresses (host, server reflexive, and relay) where a user can receive the signaling messages. The STUN-TURN object contains the network address and port number where a peer listens for the STUN [Rosenberg *et al.*, 2008] requests. Next, we discuss these resource-objects in more detail.

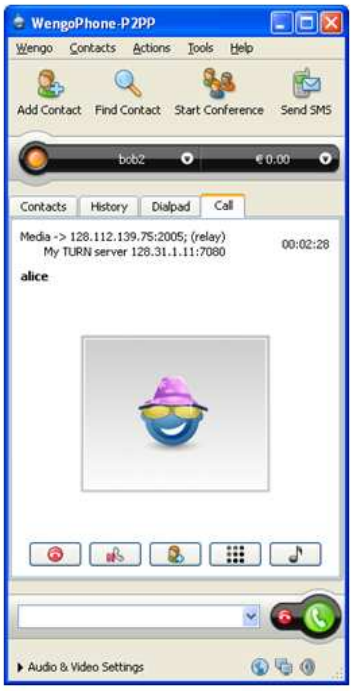



Figure 4.2: Wengo-P2PP phone relaying a media session through an OpenVoIP peer.

#### 4.2.1.1 SIP-CONTACT

When starting Wengo-P2PP phone for the first time, the user chooses a SIP address-of-record as its user identifier. The Wengo-P2PP phone then publishes the user identifier and reachable network address of the phone in the overlay, so that other phones can locate it and establish a media session. OpenVoIP defines a SIP-CONTACT resource-object, which contains the SIP address-of-record of the user (e.g., sip:bob@example.com) as the resource-id and the reachable network addresses gathered using ICE [Rosenberg, 2010] encoded in an address-info object as the value of the resource-object. The content-type of the object is SIP AoR. Other nodes can search for the user identifier using a LookupObject request, and discover the resource-object containing the reachable address in order to establish a media session using SIP.

#### 4.2.1.2 STUN-TURN

Each peer in the OpenVoIP system provides the NAT traversal and relaying service for signaling and media traffic. The peers do so by running the STUN [Rosenberg *et al.*, 2008] and TURN [Mahy *et al.*, 2010] protocols. An OpenVoIP node listens on the same port for the incoming STUN and TURN control messages. Other nodes can discover the network address and port number of the STUN and TURN service that a peer provides by sending a LookupObject request. The resource-id object in rlookup object (Appendix A.18) contained in the request is set to the node-id of the peer receiving the request and the content-type is STUN-TURN. The node receiving the LookupObject request replies with a STUN-TURN resource-object. The resource-id of this object is set to the node-id of the object sending the response, the content type is STUN-TURN, and the value is an address-info object containing the network address and port number on which the node listens for the STUN and TURN requests.

## 4.3 System Functionality

We describe how a node discovers the network addresses of peers already in the overlay (Section 4.3.1), the process it follows to join the overlay as a peer or as a client (Section 4.3.2), the mechanism for call establishment (Section 4.3.3), the technique for finding a suitable relay for establishing a media session (Section 4.3.4), and the mechanism for monitoring and diagnostics (Section 4.3.5).

### 4.3.1 Bootstrap

To join OpenVoIP, a node must discover the network address of the peers already in the system. OpenVoIP is designed to run on the public Internet where mechanisms such as IP multicast for discovering a peer already in the overlay may not always work. Therefore, we run a bootstrap server which maintains a list of the network addresses of a subset of peers already in the overlay. In our implementation, the bootstrap server maintains a list of the network addresses of ten peers, but the number is configurable. When a Bootstrap request arrives at the bootstrap server, it selects those network addresses in a random order and

returns them to the node that sent the Bootstrap request. The node can then send the Join request to the first node in the list. The bootstrap server returns the peers in a random order to equally distribute the load of Join requests over these nodes.

A key question is how does a bootstrap server determines which peers to maintain in its list. Initially, we programmed the bootstrap server to maintain a list of most recent ten nodes that had joined the overlay. However, this technique suffered from three problems. First, in OpenVoIP, nodes behind NAT and firewall can only join as clients. However, whether a node joins the system as a client or a peer is only determined after a node runs connectivity checks with peers already in the overlay. Although the nodes can run connectivity checks with the bootstrap server, we wanted to keep the functionality at the server to a minimum. Second, since we have made available the source code and executables online, potentially anyone can join OpenVoIP, thereby generating a Bootstrap request at the server. During our operational experience, we observed that some times people tried running our executables multiple times to connect to OpenVoIP and then went offline for extended periods. Since the bootstrap server maintains a finite list of peers, all the peers in the list can potentially be offline. Third, although it is possible for a bootstrap server to periodically receive a KeepAlive request from the peers and then select suitable peers, such mechanism requires additional functionality at the bootstrap server.

In our implementation, we have used a rather simple approach. The bootstrap server maintains a list of ten peers referred to as bootstrap peers, and we ensure their availability. These peers participate in the overlay in the same manner as other peers and run the same protocol as other peers. By ensuring their availability, we avoid the problem that each peer must inform the bootstrap server about its presence and also avoid the issue that the bootstrap server may store a list of offline peers. The load on the bootstrap nodes is distributed equally since the bootstrap server returns the list of bootstrap peers in a random order to the joining node. Although not implemented in OpenVoIP, the load on the bootstrap nodes can further be reduced if the joining nodes cache the peers they were connected to in the last run and try sending them the Join request before contacting the bootstrap server.

### 4.3.2 Joining the Overlay

After discovering the reachable address of a peer already in the overlay thereafter referred to as the admitting peer (AP), the joining node (JN) sends it a LookupObject request to discover the STUN [Rosenberg *et al.*, 2008] port this peer is listening on. The peer replies with a 200 (Ok) response containing the STUN-TURN object (Section 4.2.1.2). On receiving the 200 (Ok) response, the JN runs the ICE [Rosenberg, 2010] connectivity checks with this peer to determine if it is behind a NAT. As mentioned in Section 4.2, all peers in the OpenVoIP system run on machines with a public IP address and are not behind a NAT. If the JN determines that it is not behind a NAT, it sends a Join request to the admitting peer with the P flag set in the common header, indicating that it desires to join as a peer. Otherwise, the JN sends the Join request without the P flag set in the common header, indicating that it is joining as a client. The node sending the Join request can set the recursive or iterative routing flag (R flag in the common header) based on the configuration file; the default is iterative. When a peer in the overlay (neighbor peer) responds with a 200 (Ok) response to the Join request, the JN takes an appropriate action depending on whether it joins as a peer or a client.

#### 4.3.2.1 Peer Join

If a node is joining as a peer, it sends an ExchangeTable request to the neighbor peer, requesting for its neighbor and routing table. In structured overlays, the peers use the neighbor table to maintain a consistent view of the overlay such as successor list in Chord [Stoica *et al.*, 2003] or leaf-set in Pastry [Rowstron and Druschel, 2001a]. The joining peer sends the Join request with the S flag set in request-options object to indicate to the nodes received in the neighbor table that it will be their new neighbor. The neighbors of this peer then transfer any resource-objects to this peer using the TransferObject request (Section 3.6.3.3). The JN then builds its routing table from the routing table it received in response to the ExchangeTable request. Before inserting a peer in its routing table, JN checks its liveness by sending a KeepAlive request. A peer then opens a listening port for the STUN protocol and publishes a SIP-CONTACT object in the overlay using the PublishObject method.

Figure 4.3 shows the message flow between a node joining the OpenVoIP system as

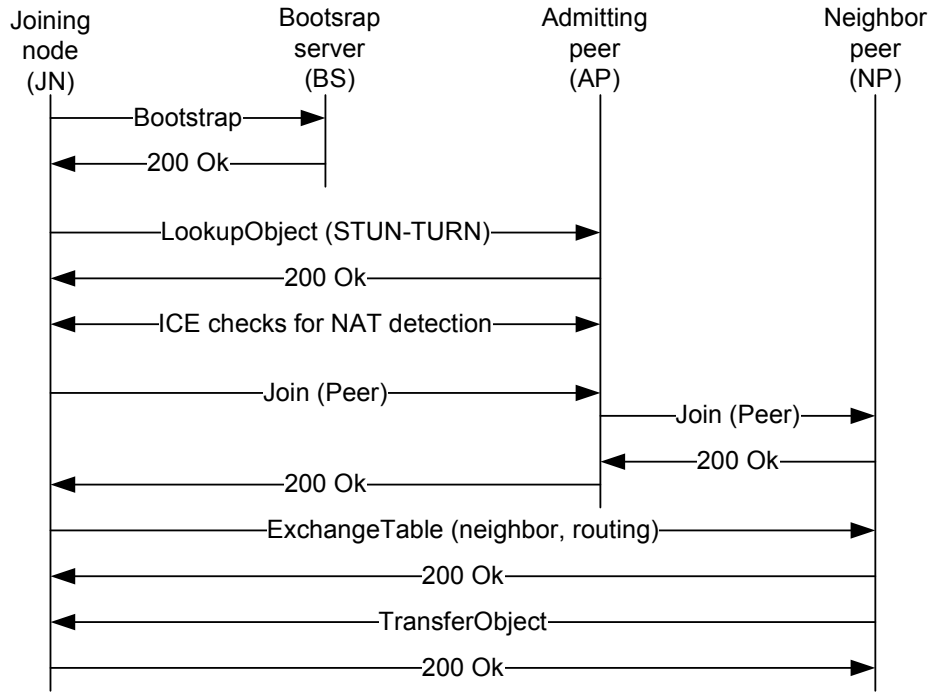



Figure 4.3: The message flow between a node joining as a peer, the bootstrap server, and peers in the OpenVoIP system.

a peer, the bootstrap server, and peers in the OpenVoIP system. The messages are not strictly ordered by time. The KeepAlive and PublishObject methods are not shown.

#### 4.3.2.2 Client Join

When a node joining as a client receives the 200 Ok response from the responsible peer, thereafter referred to as the connected peer, it connects to that peer. It then sends an ExchangeTable request to this peer to retrieve its neighbor and routing table. The purpose of retrieving these tables is twofold. First, they allow a client to build a list of backup peers in case its connected peer goes offline; second, a node keeps track of the peers with the smallest network latency from itself, in anticipation that it may use them or their close-by neighbors (in terms of network latency) as relays for exchanging the signaling and media traffic. The client publishes a SIP-CONTACT object in the overlay. For a client, the SIP-CONTACT object contains the host, server-reflexive, and relay addresses where it can receive the SIP signaling messages.

### 4.3.3 Call Establishment

To establish a call, the Wengo-P2PP phone sends a LookupObject request to search for the SIP-CONTACT object of the callee phone. If the caller node is a client, its connected peer performs the search on its behalf. After discovering the SIP-CONTACT object of the callee, the caller node performs connectivity checks with host, server-reflexive, and relay addresses of the callee to determine the reachable addresses over which it can exchange the SIP signaling traffic. The caller prefers the use of host and server-reflexive addresses for SIP. Then, the caller Wengo-P2PP phone sends the SIP INVITE request to callee through the network address determined by the connectivity checks. The INVITE request contains the host, server-reflexive, and relay candidates on which a caller can receive callee's media traffic.

After the callee picks up the phone, its user agent replies with a 200 (Ok) SIP response, and a media session is established. The nodes then perform connectivity checks to determine if they can directly exchange the media traffic using a host or server-reflexive address, or if they require the use of a relay.

### 4.3.4 Distributed Relay Search

OpenVoIP is a two level hierarchical system where peers provide the message routing, storage, media relaying and bandwidth test measurements, and clients connect to one or more peers. Each peer maintains a routing table containing the subset of peers in the overlay that are selected according to the overlay algorithm being used. We used the above two facts in designing a relay selection scheme which we refer to as the *Threshold* scheme. Section 6.5.3 presents a detailed description of this scheme and evaluates its performance using simulations. A brief description of this scheme is as follows.

Each peer maintains RTT information with peers in its routing table as part of liveness checks. In addition, a peer also exchanges the uptime and spare network capacity information with other peers in its routing table. A peer can determine its spare capacity by running a bandwidth test with other peers as described in Section 4.3.4.1. Since relaying a call utilizes the network link of a machine, it can interfere with other network centric applications running on the relay machine. It is important to select a relay that minimizes

this interference. We refer to this interference as user annoyance (Section 6.5.2). Since relay calls consume network bandwidth, the spare network capacity of a machine is a useful metric to quantify the impact of such interference. Section 6.5.2.1 discusses issues in estimating network link capacity.

The goal of the relay selection is to quickly find a relay with acceptable network latency and minimal user annoyance. Each client maintains a list of close-by peers in terms of network latency. When a client needs a relay, it requests its close-by peer for a set of relay candidates. If the peer cannot fulfill the relay request itself, it searches its routing table for peers that have a network latency of less than 150 ms from this peer, and have the maximum spare network capacity. If the peer cannot find any relay that meets both criteria, it randomly selects a peer providing the media relay service from its routing table. The idea behind requesting a relay candidate from a close-by peer is that a client may not have to perform latency measurements with relay candidates returned by a close-by peer as the worst case latency (from a client to a close-by peer, and from the close-by peer to peers in its routing table) may still be within acceptable latency bounds. Another benefit of this scheme is that a client can find a relay in $O(1)$ hops if relays are plenty. However, the performance of this scheme starts to degrade as the percentage of relay requests reaches more than 70% of the total relaying capacity of all peers (see Section 6.5.1).

#### 4.3.4.1 Link Capacity Measurements

The peers periodically perform a bandwidth measurement test with other peers to determine their uplink and downlink capacity. In our implementation, a peer performs the bandwidth test every 15-30 minutes, with the precise time selected on a random basis within this interval. The bandwidth test comprises of sending dummy bulk data over a TCP connection for a period of ten seconds. The peers do so by using the MeasureBandwith (Section 3.6.5.2) method of P2PP. Using TCP throughput measurements is likely going to provide a lower (or conservative) estimate of the network capacity of the node due to the AIMD nature of TCP. However, such lower estimates are still useful from a user annoyance perspective. The peers also monitor their network usage, and whether they are relaying any media sessions. The peers then exchange the spare network capacity as part of KeepAlive messages.


Figure 4.4: The OpenVoIP peers running on PlanetLab.

### 4.3.5 Monitoring and Diagnostics

OpenVoIP is an experimental peer-to-peer communication system running on PlanetLab. Like many other distributed systems, it is difficult to get a quick feel of the system without an appropriate user interface. The importance of a user interface for distributed systems is more pronounced for the experimental systems, as a researcher will like to quickly gain insights into the system operation and demonstrate the feasibility of the system to non-technical users.

We have developed a Google map interface for OpenVoIP that displays the online and offline peers running on a geographical map and allows a convenient way to interact with the nodes running in the system. This geographical interface displays the peers as markers at their geographic locations on the map. We use the MaxMind tool [Maxmind, 2010] to translate the IP addresses of peer into latitude and longitude. We run a diagnostic server (see Section 3.4.1) that periodically checks the liveness of peers in OpenVoIP. Due to scalability reasons, we only check the liveness of peers that we run on the PlanetLab and ignore any peers that may be run by other users. Figure 4.4 shows the OpenVoIP peers running on PlanetLab as green and red markers, indicating if those peers are online or offline.

The user interface also allows a user with a web browser to query the state of a node and store and retrieve the information it stores. Under the hood, it uses the GetDiagnostics

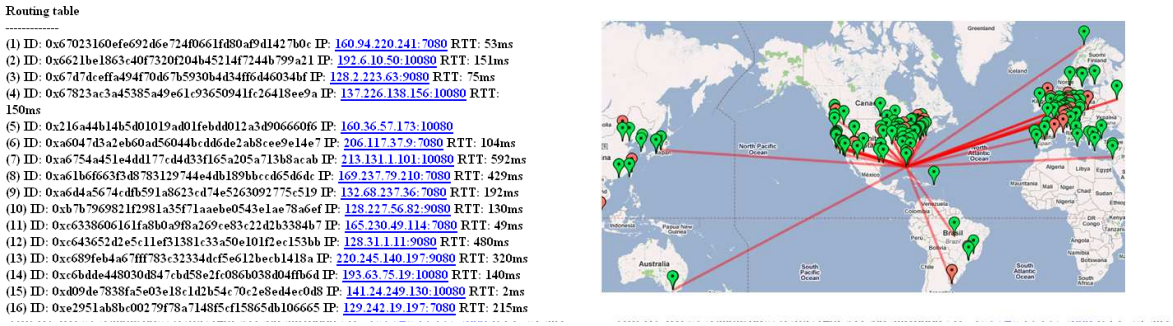



Figure 4.5: The routing table of a peer on PlanetLab.


Figure 4.6: Tracing a LookupObject request on the OpenVoIP's Google maps visual interface.

(Section 3.6.5.1) method of P2PP to query the node state, its routing, and resource table. Figure 4.5 shows the routing table of a peer in OpenVoIP system. The figure also shows the round-trip time to the peers in the routing table of this peer. The interface also allows a user to retrieve and display the SIP-CONTACT object of a user. Figure 4.6 illustrates the progress of a LookupObject request for the SIP-CONTACT resource object.

## 4.4 Lessons Learned

We have implemented three DHTs based on P2PP protocol for our OpenVoIP system. These DHTs are Bamboo [Rhea *et al.*, 2004], Chord [Stoica *et al.*, 2003], and Kademlia [Maymounkov and Mazieres, 2002]. We found that DHT-specific portions in our implementations comprised 10-15% of the total lines of code, which were approximately sixteen thousand. Our implementation of the P2PP library includes mechanisms for data model, message reliability, and NAT and firewall traversal. Overall, the lines of DHT specific code will likely

comprise a smaller chunk (< 10%) since we did not implement the message confidentiality and data integrity mechanisms. This outcome in terms of a smaller percentage of lines of code for the DHT-specific part is a vindication of P2PP design decision which provided common mechanisms for data model, data integrity, message reliability and confidentiality, and NAT and firewall traversal.

## 4.5 Related Work

OpenDHT [Rhea, 2005] is an open source system that uses the Pastry DHT to build a distributed storage system and has been deployed on PlanetLab. However, it has the following limitations. First, it uses a custom protocol for building the distributed storage system. While the custom protocols are useful in building experimental systems, they are not appropriate for running interoperable systems. Second, OpenDHT as-is does not allow nodes that do not run on PlanetLab to fully participate in the OpenDHT as a full peer. Moreover, the system does not facilitate nodes behind NATs and firewalls to participate in the overlay, nor does it allow OpenDHT nodes to provide the media relaying service. Further, the system uses only one DHT (Pastry) and does not allow to incorporate any other overlay algorithms.

Skype [Skype, 2010a] is a peer-to-peer communication system that enables establishing media sessions between Skype nodes. However, Skype uses a proprietary protocol. Further, a Skype application requires Internet access to connect to the Skype network. Consequently, it cannot work in environments where access to the Internet may not be available.

## 4.6 Conclusion

We have designed and implemented OpenVoIP, an open source peer-to-peer communication system and have deployed it on the PlanetLab. OpenVoIP demonstrates the feasibility of using P2PP for building p2p communication systems. The system allows the nodes behind NAT to join the overlay as clients; the rest of the nodes join as peers. We have also designed and implemented a browser-based monitoring and diagnostic user interface for OpenVoIP. The interface facilitates gaining quick insights into the workings of the system. We have

also explored the practical feasibility of a relay selection scheme that minimizes the latency of a relayed call and the interference of the call with other applications running on the relay machine.

# Chapter 5

# Peer-to-Peer Multiparty Video Conferencing

## 5.1 Introduction

To establish a multiparty video conference in a peer-to-peer communication system, the participants' user agents can form a full mesh network [Lennox and Schulzrinne, 2003], where each user agent sends its video stream to all the participant user agents and receives the video stream from each participant user agent. However, two factors may not allow user agents to do so. First, the uplink capacity of participants is typically smaller than the downlink capacity and varies across user agents, thereby limiting the number of participants that can be accommodated at a desired bit rate. Second, some of the participants may be behind restrictive NATs and firewalls and unable to directly exchange packets with other participants. Thus, the limited uplink capacity and the presence of NATs and firewalls makes it challenging for the user agents to establish a full-mesh video conference without helpers and intermediaries. In client-server communication systems, the helpers are managed servers whereas in p2p communication systems, the helpers are non-participant altruistic peers. Each participant user agent can use these helpers to construct an application layer multicast tree for conferencing (ALMC) to deliver its video stream to all other conference participants.

Video conferencing using altruistic peers (or helpers) is non-trivial because, like confer-

ence participants, the helpers also have limited uplink bandwidth. Thus, multiple helpers may be needed to forward a participant's video stream to all other participants. Moreover, these helpers can leave the p2p network at any time, thereby impacting the reliability of p2p video conferences. In addition, the latency of a video stream passing through multiple helpers may exceed the tight playout requirements for video conferencing. Also, the naive forwarding of the video streams using helpers can significantly impair the network performance of other applications running on the helper machines. To prevent such an impairment while maintaining a desired bit rate for conferencing, more helpers are needed.

This chapter systematically explores the practical issues in constructing helper-assisted ALMC trees to deliver a participant's video stream to all other participants and outlines a protocol for enabling peer-to-peer video conferencing. Throughout the chapter, we refer to altruistic peers as helpers. As of June 2010, there were no commercial peer-to-peer video conferencing applications. The popular p2p VoIP and IM application, Skype [Skype, 2010a], uses managed servers for video conferences involving three to five participants. For two party video calls, the Skype application may use a helper if the user agents cannot directly exchange packets. In Section 5.2, we provide the results of measuring the bit rate of a helper-assisted two party video call in Skype, and contrast it with the bit rate of helper-assisted video call in ooVoo [ooVoo, 2010], a client-server video conferencing application. In Section 5.3, we describe two approaches for constructing helper-assisted ALMC trees, namely *tree* and *split*, and discuss their impact on the number of helpers, reliability of delivery, helper state, and latency. We then discuss if minimizing the number of helpers is always a good idea from the perspective of the latency of a stream, whether helpers should be selected close to the source or the recipient, if the helpers should also transcode the video stream besides forwarding it, and if the participants can be used as helpers. In Section 5.4, we outline a protocol for helper-assisted video conferences. Our hope is that this protocol can be useful for the designers of p2p video conferencing applications and for the standardization of a p2p video conferencing protocol.

## 5.2 Helper Bandwidth Usage in Skype and ooVoo

As of June 2010, Skype, a popular peer-to-peer VoIP application, supports five party video conferencing through managed servers but may use another Skype application as helper for two party video calls. To check the former, we established ten video conference calls between four Skype user agents running in June 2010. The user agents were running on machines in our lab and were connected through 1 Gb/s Ethernet. The traffic was not blocked between those machines. We found that each Skype user agent sent its video to the servers managed by Skype, which forwarded the stream to all the other user agents. We also found either two or three managed servers were used to forward the video stream of a user agent to all the other user agents in the video conference. We observed the maximum upload bandwidth from a Skype user agent to the managed server to be approximately 360 kb/s. Surprisingly, the user agents did not exchange packets directly even though they were on the same LAN.

We have discovered that Skype uses other Skype peers as helpers for two party video calls if the traffic between the two Skype user agents may be blocked. We measured the bit rate of the calls involving a Skype peer as a relay. Our goal is to understand if Skype limits the bit rate of a relayed video call. The video call was established between two Skype applications (version 4.1.0.79) running on machines in our lab in February 2010. Skype was forced to select a UDP relay (or a helper) for the video call because we blocked direct traffic between the machines. We found that the bit rate of a video call through a relay never exceeded 110 kb/s (or approximately 13 kB/s), whereas the bit rate of a call involving no relay was 570 kb/s. These measurements suggest that Skype bounds the bit rate of a relayed video call.

ooVoo [ooVoo, 2010] is a client-server video conferencing service and uses managed helpers to relay video calls. We measured the bit rate of a relayed two party ooVoo video call by using the same setup as the one used for Skype experiments. Unlike Skype, the relay used was a machine managed by ooVoo and was not an altruistic peer. The bit rate of a relayed video call through a ooVoo managed helper was as high as 320 kb/s, or three times the bit rate of a two party relayed Skype video call. The difference between the two bit rates highlights the tradeoff between receiving a high quality (high bit rate) stream through an altruistic peer and the interference of the video stream with the applications running

| | Participant od=3 | | | | Participant od=1 | | | |
|---|---|---|---|---|---|---|---|---|
| NP/HO | 2 | 3 | 4 | 5 | 2 | 3 | 4 | 5 |
| 3 | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 3 |
| 4 | 0 | 0 | 0 | 0 | 8 | 4 | 4 | 4 |
| 5 | 5 | 5 | 5 | 5 | 15 | 10 | 5 | 5 |
| 6 | 12 | 6 | 6 | 6 | 24 | 12 | 12 | 6 |
| 7 | 21 | 14 | 7 | 7 | 35 | 21 | 14 | 14 |
| 8 | 32 | 16 | 16 | 8 | 48 | 24 | 16 | 16 |
| 9 | 45 | 27 | 18 | 18 | 63 | 36 | 27 | 18 |
| 10 | 60 | 30 | 20 | 20 | 80 | 40 | 30 | 20 |

Table 5.1: Number of helpers needed for the *tree* approach as a function of number of participants (NP) (first column) and helper outdegree (HO) (second row). The number of helpers are calculated for participant outdegree (od) of 3 and 1, respectively.

on that peer. For these reasons, Skype limits the bit rate of a relayed video call. Besides interference, relaying a high bit rate video call has an economic cost for users running these altruistic peers if their ISP's Internet plan either caps the network traffic or if the users have to pay for every downloaded bit.

## 5.3 Practical Issues in Application Layer Multicast Tree (ALMC) Construction

Two approaches for ALMC tree construction have been proposed in the application layer multicast and video conferencing literature, namely, a *tree* (such as [Castro *et al.*, 2003]) and a *split* approach (such as [Chen *et al.*, 2008]). In the tree approach, each participant of a video conference constructs a balanced tree of helpers to deliver its video stream to all the conference participants. In the split approach, the participant splits its video stream among helpers, and each helper directly forwards the split stream to all the conference participants. Figure 5.1 shows an example of a nine-party video conference using both approaches. The outdegree of each helper and participant in the figure is two and one, respectively.

Next, we discuss the practical issues for these schemes.

| | Participant od=3 | | | | Participant od=1 | | | |
|---|---|---|---|---|---|---|---|---|
| NP/HO | 2 | 3 | 4 | 5 | 2 | 3 | 4 | 5 |
| 3 | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 3 |
| 4 | 0 | 0 | 0 | 0 | 8 | 4 | 4 | 4 |
| 5 | 5 | 5 | 5 | 5 | 10 | 10 | 5 | 5 |
| 6 | 12 | 6 | 6 | 6 | 18 | 12 | 12 | 6 |
| 7 | 14 | 14 | 7 | 7 | 21 | 14 | 14 | 14 |
| 8 | 24 | 16 | 16 | 8 | 32 | 24 | 16 | 16 |
| 9 | 27 | 18 | 18 | 18 | 36 | 27 | 18 | 18 |
| 10 | 40 | 30 | 20 | 20 | 50 | 30 | 30 | 20 |

Table 5.2: Number of helpers needed for the *split* approach as a function of number of participants (NP) (first column) and helper outdegree (HO) (second row). The number of helpers are calculated for participant outdegree (od) of 3 and 1, respectively.

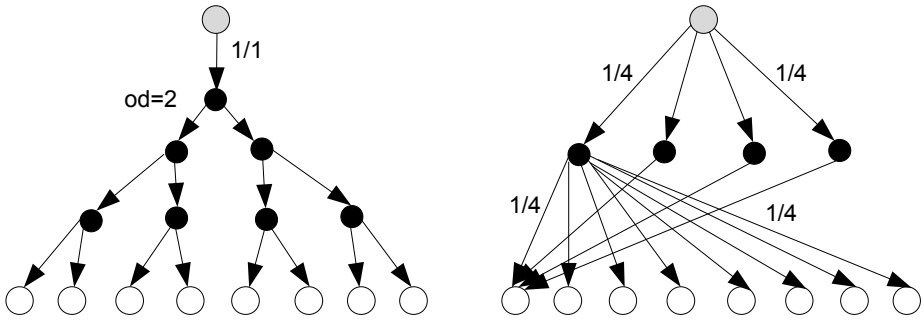



Figure 5.1: Tree (left) and split (right) approach for video conferencing. The black nodes are helpers. For the split approach, not all flows are shown.

### 5.3.1 How Many Helpers?

A question arises how many helpers are needed to scale a video conference as a function of the number of participants $n$, for the *tree* and the *split* approach. For simplicity of analysis, assume that each helper has an outdegree, $k$ ($k \geq 2$), and each participant has an outdegree of one. The degree corresponds to a reasonable video stream bit rate such as 100 kb/s. The upper bound on the number of helpers in a complete $k$-ary tree, assuming helpers are not shared among different trees, is given by Equation (5.1):

$$\lceil \frac{k^{log_k(n-1)} - 1}{k - 1} \rceil \quad (n > 2) \tag{5.1}$$

The upper bound on the number of helpers using the split approach when each participant has an outdegree of one is given by Equation (5.2):

$$\lceil \frac{(n-1)}{k} \rceil \quad (n > 2) \tag{5.2}$$

Table 5.1 and Table 5.2 show the total number of helpers needed for the tree and the split approach as a function of number of participants and helper and participant outdegree, assuming there is no limit on the indegree of participants. The tables illustrate that as the number of participants increase, the tree approach requires significantly more helpers than the split approach. As an example, for 10 participants and helper outdegree of two, the total number of helpers for the tree and the split approach is 80 and 50, respectively, or eight and five helpers per ALMC tree rooted at every participant.

The split approach performs better than the tree approach in terms of number of helpers; however, it still requires 50 helpers (or five per tree) for a 10 party conference when helper outdegree is two. It is non-trivial to search for such a large number of helpers as these helpers are altruistic peers. Although the number of helpers can be reduced if the sender reduces its bit rate, this compromises the quality of the video stream. Moreover, with a large number of helpers, the helper churn becomes a significant problem. Using a naive approach for protecting against helper churn, in which each participant maintains a backup for every helper becomes very costly in the number of backup helpers as the number of primary helpers increases. The relatively large number of helpers highlights the limits of scaling video conferences using altruistic peers as helpers.

In addition to restricted uplink capacity, participants also have finite downlink capacity. The downlink capacity of the participants can become saturated as the number of participants increase, since a participant must receive the video stream from all other participants. Although the participants may choose to receive the video of only currently active participant, this solutions compromises the interactivity of the video conference. When the downlink capacity of all participants is saturated and a new participant is added to the conference, each participant must request the other participants to reduce their sending rate in order to receive the video stream of the new participant. This reducing of rate

frees capacity on existing helpers, which can be used to serve new participants without adding any helpers. As an example, refer to the split approach shown in Figure 5.1 where each helper has an outdegree of two for a complete video stream, where a complete video stream implies a specific bit rate (e.g., 100 kb/s). Now consider that each participant has an indegree of eight. Without an indegree limit, a conference of 17 participants will need eight helpers using the split approach, since each participant must receive the video stream from 16 sources. However, with an indegree limit of eight, each source must reduce its rate by half, so that each participant can receive the video from 16 other participants. Each of the four helpers which previously received a quarter of the video stream now receives one-eighth of the original stream and can forward it to sixteen participants (assuming negligible packet header overhead). This example shows an important practical consideration: when a new participant joins, the old participants may not need to add additional helpers if the downlink capacity of the existing participants is saturated.

### 5.3.2 Reliability

When a helper goes offline in the tree approach, the number of affected participants depend on the level of the failed helper, whereas in the split approach, all participants are affected unless error correction or erasure codes are used. We are interested in calculating the reliability of the helper-assisted video conferences as a function of number of helpers, helper lifetime, and video conference duration for the tree and split schemes. Specifically, what is the probability that the video stream a participant receives from another participant will not be disrupted due to helper churn? Let $X_i$ denote the lifetime of a helper, $R_i$ be its residual lifetime, and $X_i$ and $R_i$ are i.i.d. The probability that a participant receiving the video stream from another participant's ALMC tree will not be disrupted due to helper churn is given by Equation (5.3).

$$P(t) = \prod_{i=1}^{r} P(R_i > t) \tag{5.3}$$

where $r$ is the number of helpers a video stream passes through or is split across, before a participant receives it.

The number of helpers along the path of a participant using the tree approach is given by $\lceil \log_k(n-1) \rceil$ and is always less than or equal to the number of helpers using the split approach, assuming both schemes employ helpers with the same outdegree. In Figure 5.1, the number of helpers along the path of the tree are three, whereas the number of helpers in the split approach are four. As mentioned before, a video stream is disrupted whenever a helper in the path of a video stream goes offline. If the lifetimes are pareto distributed, the residual lifetime distribution is given by $F(x) = 1 - (1 + \frac{x}{b})^{(1-a)}$, where $a$ is the shape parameter and $b$ is the scale parameter [Leonard *et al.*, 2005]. Figure 5.2 (left) shows the probability that a video stream a participant receives from *another* participant is not disrupted for various conference durations and number of helpers. The parameters $a$ and $b$ were selected so that mean helper lifetime was 5 hours, a choice motivated by Skype node lifetimes [Guha *et al.*, 2006; Kho *et al.*, 2008]. For the scenario shown in Figure 5.1, the probability that the video stream at any participant is not disrupted using the tree and the split approach, which use three and four relays, respectively, is 58% and 48%, which is extremely low from a reliability perspective. The low probabilities suggest that for video conferences involving a large number of helpers, it may be necessary to employ a combination of reliability improvement and video coding techniques, explained below, to minimize the video disruption due to helper churn.

Recently, Chen *et al.* [Chen *et al.*, 2008] advocated the use of layered coding with the split approach. However, the performance of layered coding using a split approach is poor from a reliability perspective, as the failure of a helper forwarding the $i^{th}$ layer can render the $(i+1)^{th}$ and higher layers useless. The alternative to a layered coding or a blind splitting of a single description video stream is to use multiple description coding (MDC) [Goyal, 2001]. Although the bit rate of MDC is typically higher than layered coding, it is more suitable for dealing with the helper churn in a multi-helper split scheme.

Besides helper churn, the video stream can also be momentarily disrupted when a participant joins the conference. In the tree approach, when a new participant joins the conference, it may momentarily disrupt the video of at most one existing participant if a new helper must be added to the tree to serve the new participant. In contrast, in the split approach, a new participant join may momentarily disrupt the video at all other participants. This

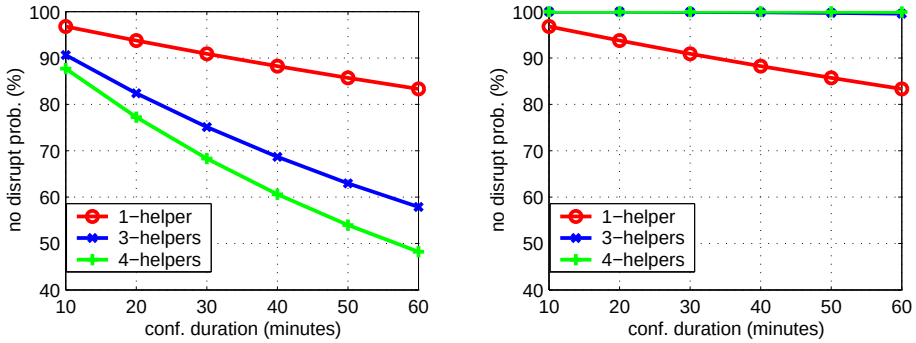



Figure 5.2: (Left) Probability that a video stream is not disrupted at a receiving participant when it passes through or is split across 1, 3, or 4 helpers. (Right) Probability of video disruption in a split scheme when multiple description coding (MDC) is used for 1, 3, or 4 helpers. Helper lifetime is pareto distributed with a mean of five hours.

happens if a video stream source adds a new helper to serve the new participant and thus has to redistribute its video stream over this new set of helpers.

### 5.3.3 Helper and Participant State, and Latency of a Video Stream

In the tree approach, the state maintained by a helper is proportional to its outdegree (assuming a specific bit rate of the stream), whereas in the split approach the helper state grows with the number of participants. However, the video conference size is likely to be small (10-15 participants) and can also be bounded by the application (five participants in Skype, six participants in ooVoo [ooVoo, 2010]). As such, the increased state per helper for small conference sizes does not present a problem.

In the tree approach, a source sends the video stream to a helper which forwards it to the next helper and so on until it is forwarded to the recipient. The latency of the video stream in the tree approach is the sum of these overlay hops. In the split approach, portions of the split stream are received through multiple helpers, and the latency of the complete stream received at a participant is the maximum of the latencies of the split streams through one-hop helpers.

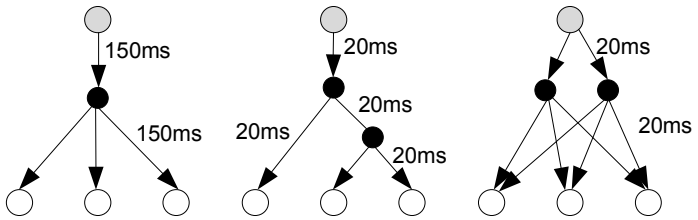



Figure 5.3: One helper (left) tree approach (center), split approach (right). The tree and split approach optimize for latency whereas one helper optimizes for number of helpers. The black nodes are helpers.

### 5.3.4 Always Minimize Helpers?

Figure 5.3 shows the tradeoff between latency and number of helpers in ALMC tree construction. The left tree uses only one helper but the latency from the source to the recipients is 300 ms. The latency is relatively small for the center and right trees that use the tree and split approach, respectively, but they use two helpers. This situation arises because some helpers can forward more streams than other helpers and illustrates that for these scenarios, minimizing helpers may not always result in a minimum latency tree.

### 5.3.5 Select Helper Close to the Source or Recipient?

Figure 5.4 shows the tradeoff between selecting a helper close to the source or one of the recipient. Selecting helpers close to one of the recipient has the problem that the helper may not be close to other participants in terms of network latency. As a consequence, other participants receiving the video stream of a source from this helper may suffer from increased latency. On the other hand, selecting helpers as close to the source as possible is less likely to put any recipient at a latency disadvantage.

### 5.3.6 Transcoding a Video Stream?

So far we have assumed that helpers only forward the video stream to participants or other helpers. In addition to this forwarding of a stream, a helper can also transcode a stream to adjust the bit rate of the forwarded stream according to the downlink capacity of the participants. Transcoding is likely needed in a single tree scheme to handle the

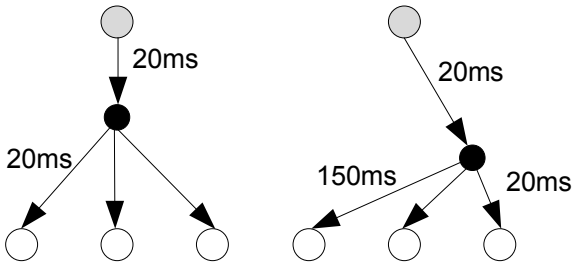



Figure 5.4: Selecting a helper close to the source or one of the recipients.

heterogeneous downlink bandwidth of the participants; otherwise, the bit rate of the video stream received by all the participants is constrained by the smallest downlink capacity amongst the receiving participants. However, transcoding a video stream puts a processing burden on the helper. If the helper is managed, this burden may be acceptable. However, the users of peers providing the helper service may not tolerate the processing burden in addition to the use of their network connection without proper incentives.

### 5.3.7 Sharing Helpers Across ALMC Trees?

In theory, helpers that have sufficient spare bandwidth can forward the video of several participants to all other participants, i.e., a helper can be shared across several ALMC trees of the same or different conference. However, as mentioned in Section 5.2, the designer of a commercial p2p video conferencing system is likely to limit the bit rate of a video stream through a helper (e.g., to 100 kb/s) to minimize the interference of the p2p helper application with network centric applications running on the helper machine. Such cap on the bit rate practically limits the use of helpers across different ALMC trees.

### 5.3.8 Participants as Helpers?

The participants can also be used as helpers if they have sufficient uplink bandwidth, as proposed by [Chen *et al.*, 2008]. However, Dischinger *et al.* [Dischinger *et al.*, 2007] reported in their measurement study of broadband hosts that the uplink network capacity of majority of broadband hosts was less than 500 kb/s. For such uplink bandwidths, a participant may only be used as a helper for small conference sizes (e.g., 3 or 4) assuming no other

application requiring network bandwidth is running on the participant's machine. However, for moderately sized conferences (such as six) where some participants may require a helper, it is practically difficult for a participant to be used as a helper. Also, the video stream of a participant that uses other participants as helpers can be disrupted when the participant acting as a helper leave or drop out of the conference.

### 5.3.9 Managed Helpers

A p2p communication system provider like Skype can scale video conferences by using managed helpers in addition to altruistic peers. Using managed helpers has several advantages. First, unlike altruistic peers, managed helpers are likely to have a reasonable uptime guarantee, so helper churn becomes less of an issue. When churn is less of a concern, participants can use single description or layered coding instead of MDC. Moreover, helpers can also combine the video stream from several sources and transcode the aggregated video stream to accurately match the downlink bandwidth of a participant. However, the benefits of using managed helpers have an economic cost. As an example, Nationalnet dedicated hosting service [NationalNet, 2010] used by ooVoo costs $559 per month for a 100 Mb/s dedicated server (in February 2010). For a six party video conference, where each participant has an uplink and downlink network capacity of 100 kb/s and sends its stream at 100 kb/s to the server, the uplink and downlink bandwidth usage on the server is 3,000 kb/s and 500 kb/s, respectively, assuming the server is able to encode video from all participants at 100 kb/s, and distribute this encoded stream to each participant.

This analysis suggests that a server can at most support 33 simultaneous 6-party video conferences for the scenario described. The cost per minute of using the server, assuming 50% utilization for the month, is approximately three cents per minute ($\frac{\$559\,\text{per month}}{12\,\text{hours}*60}$). For million minutes per month of video conferencing, the cost is $30,000 per month.

## 5.4 Putting it All Together

We outline the salient features of a protocol for accomplishing p2p video conferencing that strives to achieve the best quality for the video conference in terms of bit rate and latency,

while at the same time minimizing the impact of helper churn.

### 5.4.1 Use Split Approach with Multiple Description Coding

When a conference participant needs more than one helper to distribute its video stream to other helpers, it should use the split approach as it uses less helpers than the tree approach and is likely to achieve lower latency for all the participants. Further, when using multiple helpers in the split approach, a source should use multiple description coding (MDC) to split the video across multiple helpers so that the failure of a helper does not completely disrupt the video stream at the receiving participants.

### 5.4.2 Select Helpers Close to the Video Source

Each conference participant (or source of the video stream) should select helpers as close to itself as possible (e.g., within a 30 ms threshold or so) rather than finding a helper that optimizes latency across all recipients. This heuristic is likely to work best when participants are geographically scattered and helpers are plenty.

### 5.4.3 Prioritize Voice over Video

While each participant needs to receive video from all other participants for the best interactive experience, it only needs to receive the voice of the active participant. A video conferencing protocol should prioritize the transmission of voice over video. Further, the helpers for voice should be distinct from those of video, and where possible, voice should be delivered in a full-mesh manner.

### 5.4.4 Limit Bandwidth Usage at Helpers

To minimize interference with the network-centric applications running on helpers, a video conferencing protocol should limit the bandwidth usage at a helper. Limiting the bandwidth usage at a helper is also necessary from the perspective of the users of a p2p communication application, who may desire to know about the maximum bandwidth usage on their machine. A video conferencing protocol can also bound the number of helpers in use, or limit the number of participants in a conference based on the availability of helpers in the system.

### 5.4.5 Mitigate Helper Churn

As mentioned in Section 5.3.1, maintaining a backup for every helper may not be feasible as the number of helpers per ALMC tree grows. Instead, we recommend that the conference participants should maintain a common pool of helpers. The number of helpers in this pool can be a function of number of participants and may be capped at an upper limit (such as four). Each participant maintains liveness checks with all the helpers in this pool. When a helper churns, a participant can select a helper from this pool that has the lowest latency from itself and meets the desired bandwidth. The participating user agents should collaborate to replenish the pool of helpers when the helpers fail.

### 5.4.6 Use IP Multicast Where Possible

Although IP multicast is not widely available for end hosts, the p2p video conferencing protocol should prefer the use of IP multicast where ever possible because it can help minimize the use of helpers. Conference participants can check the support for IP multicast when they join a conference, similar to NAT detection tests [MacDonald and Lowekamp, 2010] performed by the p2p applications when they join the overlay [Baset and Schulzrinne, 2006]. Further, for optimal performance, video conferencing protocol should perform connectivity checks to determine if other participants are within the same LAN.

## 5.5 Related Work

Narada [Chu *et al.*, 2000] and NICE [Banerjee *et al.*, 2002] were among the early ALM protocols that focused on streaming from one source to many recipients without using any non-participant helpers. Narada did not address the issue of participant churn whereas NICE did not incorporate participant bandwidth for ALM tree construction. In a followup work on Narada, Yang [Chu *et al.*, 2001] focused on conferencing with only participant nodes but did not address the issue of multiple sources sharing the same overlay mesh and participant churn. To scale the video conferences, Luo *et al.* [Luo *et al.*, 2007] used IP multicast where possible in addition to the ALM tree, but such scaling is limited without the use of non-participant helpers since IP multicast is not widely available. Further,

they did not address the issue of participant churn. In an ALM tree, some receivers may act as helpers which places an undue load on them as compared to receivers which only receive the stream. SplitStream [Castro *et al.*, 2003] addressed this problem by striping the stream using multiple description coding (MDC) [Goyal, 2001] over multiple trees so that each receiver forwards one stripe. Although their work can be extended for video conferencing and to incorporate helpers, they use a Pastry DHT to construct the ALM trees for forwarding stripes. The use of a DHT to construct ALMC trees may lead to unnecessary tree transformations when participants join or leave the conference.

Shi *et al.* [Shi and Turner, 2002] were the first to consider the use of helpers (multicast service nodes (MSNs)) for streaming and focused on optimizing their network utilization (or residual degree), while keeping the tree diameter low. However, they assumed that helpers were managed servers and thus did not consider the issue of dynamic memberships. OMNI [Banerjee *et al.*, 2006] assumed a client population attached to helpers and focused on minimizing latency of the ALM tree based on client population. In contrast, in p2p video conferencing, the clients can potentially be served by any helper, subject to latency constraints. Recently, Wang *et al.* [Wang and Ramchandran, 2008] considered using helpers for p2p live multicast for reducing load on the server but did not optimize the latency of the ALM tree and did not effectively mitigate the impact of churn.

The work most related to ours is by Chen *et al.* [Chen *et al.*, 2008], who considered multi-party video conferencing at a single rate and extended their work to use scalable video codecs [Ponec *et al.*, 2009]. Their idea is to split a single or layered stream from a conference participant across one or two-hop trees. The two-hop trees are constructed using other participants or non-participants as helpers. They showed that two-hop delay tree construction achieves the same rate as intra-session network coding. However, participants usually do not have sufficient bandwidth and scaling the video conference is likely to have to rely on non-participant helpers. They did not consider the issues in managing the reliability of helper-assisted video conferences. Further, they did not consider the practical issues of helper selection due to restrictive NATs and firewalls. Nevertheless, our work builds on theirs and provides practical insights on constructing a holistic solution for p2p video conferencing.

Different vendors [Polycom Conferencing Infrastructure., 2010; Tandberg, 2010] provide hardware based multipoint control units (MCUs) where the participants send the video stream to a centrally located MCU which then transmits the streams to each participant according to their downlink bandwidth. Although this solution works, the bandwidth requirements at the server are proportional to the number of participants. Further, it may not be economically feasible for a small organization to invest in the server bandwidth or purchase costly MCU hardware. Luo *et al.* [Luo *et al.*, 2007] developed a PCI card to replicate the functionality of a centralized MCU but their approach supports only four participants.

Peer-to-peer video conferencing remains challenging as VoIP applications such as Skype [Skype, 2010a], Windows Live Messenger [Microsoft, 2010], and ooVoo [ooVoo, 2010] either support video conferencing through managed servers or only support two party video calls.

## 5.6 Conclusion

We have systematically explored the practical issues involved in scaling peer-to-peer video conferences using altruistic peers as helpers. Our analysis indicates that it is challenging to scale video conferences using altruistic peers for moderately large conference sizes (e.g., 10 participants or more) without drafting a large number of altruistic peers or compromising the quality of the video stream. We have outlined a p2p video conferencing protocol for constructing helper-assisted application layer multicast tree for conferencing to deliver a participant's video stream to all other participants. The salient features of this protocol are helper selection, helper churn mitigation, and bandwidth guarantees at the helper.

# Part II

# Analysis

# Chapter 6

# Reliability and Relay Selection

## 6.1 Introduction

Restrictive network address translators (NATs) and firewalls prevent hosts from directly exchanging packets. A survey of 1,787 NAT devices indicates that hosts behind approximately 30% of these devices cannot traverse the NATs using UDP or TCP [Müller and Klenk, 2010] implying that hosts behind two different such devices are not likely to directly exchange packets without an intermediary. Moreover, corporations are increasingly deploying firewalls to protect their networks from malicious traffic that originates both outside and inside their networks. The restrictive NATs and firewalls pose a problem for IP communication systems because they prevent the user agents from directly exchanging signaling and media traffic.

In a client-server (c/s) communication system, the caller user agent discovers the network address of a callee user agent through a managed server and exchanges signaling information with the callee user agent to establish a media session. The media traffic flows directly between the user agents. To address the connectivity constraints due to restrictive NATs and firewalls, c/s systems such as Vonage [Vonage, 2010] use managed servers for relaying the media traffic between user agents with restrictive connectivity. In contrast, peer-to-peer (p2p) communication systems try to minimize the number of servers. In these systems, the user agents collaborate to discover the network address of the callee user agent. The caller and callee user agents then directly exchange signaling and media traffic to establish a media

session. When the caller or callee user agents are behind restrictive NATs and firewalls and cannot directly exchange packets, they rely on user agents (or peers) with unrestricted connectivity for exchanging signaling and media traffic. Skype is an example of a peer-to-peer communication system that uses this technique [Baset and Schulzrinne, 2006]. Suh *et al.* [Suh *et al.*, 2006] observed that hundreds of calls were being relayed by a single Skype relay.

The above characteristics of a p2p communication system pose unique challenges for a system designer. First, the lookup performance in p2p systems must at least be as effective as the lookup performance of client-server systems. Additionally, a media session may be prematurely terminated because a relay peer goes offline. This issue motivates a formal analysis of the reliability of p2p communication systems and devising of techniques to prevent dropped sessions. Moreover, since media sessions such as voice and video have a tight playout requirement, the network latency of a media session involving a relay peer should not exceed these tight requirements. Further, the relaying of a media session may interfere with other user applications and impair their performance. A system designer must either provide incentives for users to run relay peers or design techniques that minimize the interference of relayed session with other user applications.

In this chapter, we present a framework to analyze the reliability of peer-to-peer communication systems (Section 6.3). We then devise a simple analytical model that predicts the smallest number of relays needed to achieve the desired reliability for relayed media sessions (Section 6.4.1) and evaluate it on exponential, pareto, and observed Skype node lifetimes. For a given node lifetime and call duration distribution, the model allows determining the minimum number of relays so that the percentage of successful relayed calls does not fall below a desired threshold (e.g., 99.9%). Such an analysis can help characterize the resources (relays) needed for improving the reliability of relayed calls. We then devise two techniques to prevent dropped sessions, selecting $k$ relay peers at the beginning of a call with no-replacement and with-replacement and predict their reliability improvement using reliability theory in Section 6.4.2 and 6.4.3. In Section 6.4.4, we analyze the reliability improvement scheme used by Skype. Section 6.4.5 presents the experimental evaluation of the model and discussion.

In Section 6.5, we present a distributed technique to find a relay peer in $O(1)$ hops and compare the performance of this technique to a relay selection scheme that has global knowledge of all the relays in the p2p network. Instead of designing incentives for users to allow relaying of media sessions through their user agents, we aim to minimize the interference of relayed session with the user applications. To capture the impact of the relayed media sessions on the user applications, we introduce the notion of *user annoyance* (Section 6.5.2). We augment our distributed search technique to select a relay that minimizes delay, user annoyance, or both within a threshold. To the best of our knowledge, we are the first to address the reliability issues in p2p communication systems, and to devise techniques for finding a relay that optimizes the latency of a relayed call and user annoyance. Our analysis and results are also applicable to media translation and conferencing in p2p communication systems (see Chapter 5).

## 6.2 Problem Setting

We consider a peer-to-peer communication system that has $N$ participating nodes. A node is a machine with CPU, memory and disk and is connected to the Internet through a dial-up, DSL, cable, fiber, or a wireless connection. Typically, a human user is associated with each node or a machine and runs a peer-to-peer communication application(s) (also referred to as user agents) and other applications. The p2p applications use any peer-to-peer protocol to form a p2p network. There are two types of nodes in a peer-to-peer communication network, peers and free-riders. In the literature, they are also referred to as super nodes and ordinary nodes [Liang *et al.*, 2004; Baset and Schulzrinne, 2006] or peers and clients [Bryan *et al.*, 2010]. A peer fully participates in the p2p network, collaborates with other peers to discover the reachable network address of the callee user agent, and can relay one or more media sessions. A free-rider does not collaborate in the discovery of the callee user agent and does not relay any media sessions. However, this collaboration is not always purposefully avoided. The presence of restrictive NATs and firewalls may hinder the participation of a node in the overlay, thereby forcing it to act as a free-rider. The need for relaying media sessions between caller and callee user agents arises precisely due to this

reason. For ease of exposition, we refer to the caller and callee user agent as caller and callee, relay peer as relay, and voice session as a call. Unless stated otherwise, we refer to the p2p communication application as a p2p application.

## 6.3 Reliability of a P2P Communication System

Availability is the classical metric for modeling the reliability of a communication system and is typically expressed by the number of nines after a decimal point. For example, a "3 nines" (99.9%) reliability means that the system is down only 0.1% of the time. In a p2p communication system, availability implies the ability of the system to find the network address of the callee, and also to find a relay for establishing the relayed call. However, this notion does not fully capture the reliability of relayed calls because in addition to relay search failure, calls can also fail due to relay churn since there is no guarantee about the uptime of relays. Thus, a more accurate metric to capture the reliability of calls in a p2p communication system is the number of successfully completed calls.

$$P_{succ} = P_{ss}F_{norelay} + P_{ss}\overline{F}_{norelay}P_{rs}P(R > D) \tag{6.1}$$

Equation (6.1) formalizes the notion of reliability or percentage of successful calls in a p2p communication system. The term to the immediate left of plus sign is the probability of successfully finding the network address of the callee user agent, $P_{ss}$, times the proportion of calls that do not need a relay, $F_{norelay}$. The term to the immediate right of plus sign is the probability of successfully finding the relay, $P_{rs}$, times the proportion of calls that need a relay, $\overline{F}_{norelay}$, times the probability that the residual lifetime of a relay, $R$, is greater than the call duration distribution $D$. This equation indicates that the proportion of successful calls can be increased by enhancing the performance of lookup schemes using techniques similar to [Rhea *et al.*, 2004], by designing schemes that establish a media session between user agents in the presence of NATs and firewalls without requiring a relay [Ford *et al.*, 2005], and by improving the success rate of distributed relay search and relay calls. We focus our attention on analyzing the reliability of relayed calls and relay search since other areas have seen related work [Rosenberg, 2010; Rhea, 2005].

## 6.4 Modeling the Reliability of Relayed Calls

We present a simple model to calculate the minimum number of relays per call, $k$, so that the success rate of relayed calls is above a desired reliability criteria such as 99.9% (Section 6.4.1), analyze two reliability improvement schemes, namely, *no-replacement* (Section 6.4.2) and *with-replacement* (Section 6.4.3), and present an evaluation of the model and reliability improvement schemes (Section 6.4.5). Our analysis assume that the nodes that need a relay to establish a call (ordinary nodes) can randomly select it from the set of all relays, that relays are plenty, and the system has reached stationarity. In Section 6.5.1, we discuss a distributed scheme to find a relay.

### 6.4.1 Number of Relays

Let $X_i$ be a random variable (r.v) that denotes the lifetime of relay $i$, $F_{X_i}$ be its CDF, and $X_i$ be independent and identically distributed (i.i.d). Let $R_i$ be a random variable that denotes the residual lifetime of relay $i$ when it starts relaying the call and $D$ denote the distribution of call duration. When a relay fails, the call it is relaying is immediately switched to a new relay $j$, having residual lifetime $R_j$. Since the new relay is selected immediately when the old relay fails, the residual lifetime of the relays used are also i.i.d. For simplicity, we assume that calls are not dropped during switch over to a new relay. Leonard *et al.* [Leonard *et al.*, 2005] note that if the system has reached stationarity, the CDF of residual lifetimes is given as:

$$F_R(x) = P(R_i < x) = \frac{1}{E[X_i]} \int_0^x (1 - F(z))dz \tag{6.2}$$

We are interested in determining the minimum relays per call $k$, so that the number of successfully completed relayed calls is above a desired criteria such as 99.9%, i.e.,

$$\text{Desired reliability} \leq P(\sum_{i=1}^{k} R_i > D) \tag{6.3}$$

**Lemma 1** *When $X$ and $D$ are exponentially distributed with parameters $\lambda$ and $\nu$, the r.h.s*

*of Equation* (6.3) *has a closed form solution:*

$$P(\sum_{i=1}^{k} R_i > D) = 1 - (\frac{\lambda}{\lambda+\nu})^k \tag{6.4}$$

*Proof:*

For exponential distribution, Equation (6.2) can be solved to obtain $F_R(x)$ and its probability distribution function (pdf) $f_R(x)$, which are $1 - e^{-\lambda x}$ and $\lambda e^{-\lambda x}$, respectively. Using conditioning:

$$\begin{aligned}
P(D < \sum_{i=1}^{k} R_i) &= \int_0^\infty F(D < m) \times f(\sum_{i=1}^{k} R_i = m)dm \\
&f(\sum_{i=1}^{k} R_i = m) \text{ is a k-fold convolution of exponential r.v's} \\
&\text{which have a gamma pdf.} \\
&= \int_0^\infty (1 - e^{\nu m}) \times \frac{\lambda e^{-\lambda m}(\lambda m)^{k-1}}{(k-1)!}dm \\
&= \int_0^\infty \frac{\lambda e^{-\lambda m}(\lambda m)^{k-1}}{(k-1)!} - \frac{\lambda e^{-(\lambda+\nu)m}(\lambda m)^{k-1}}{(k-1)!}dm \qquad (6.5) \\
&\text{The left term of Equation (6.5) is 1 since it is an integral of gamma} \\
&\text{pdf. Multiple and divide the right term by } (\lambda+\nu)^k \text{ and} \\
&\text{using } \Gamma(n) = \int_0^\infty e^{-x}x^{k-1}dx = (k-1)! \\
&= 1 - (\frac{\lambda}{\lambda+\nu})^k \qquad (6.6)
\end{aligned}$$

For arbitrary lifetime and call distribution, the r.h.s of Equation (6.3) is difficult to solve as convolution of $k$ i.i.d random variables is non-trivial. Instead, we use the following approximation which replaces the sum of $k$ r.v's with their maximum.

**Lemma 2** *The sum of* k *i.i.d r.v's* $R_i$ *being greater than another r.v* $D$ *is greater than or equal to one minus the* $k^{th}$ *exponentiation of the probability of* $R$ *being less than* $D$*.*

$$P(\sum_{i=1}^{k} R_i > D) \geq 1 - P(R < D)^k \quad (R_i \text{ are i.i.d}) \tag{6.7}$$

| | a=2,b=5 (mean lifetime=5 hours) | | | | a=3,b=2 (mean lifetime=1 hour) | | | |
|---|---|---|---|---|---|---|---|---|
| | k=2 | | k=4 | | k=2 | | k=4 | |
| call duration | sim (%) | rel-e (%) | sim (%) | rel-e (%) | sim (%) | rel-e (%) | sim (%) | rel-e (%) |
| 2.5 | 0.0074 | 8.5755 | 0 | 0.00 | 0.1544 | 0.2171 | 0.0003 | 21.3205 |
| 5 | 0.0251 | 3.6121 | 0 | 0.00 | 0.5517 | 0.2131 | 0.0027 | 6.9055 |
| 10 | 0.0961 | 1.7193 | 8e-5 | 20.0925 | 1.8179 | 0.1980 | 0.0319 | 3.8110 |
| 20 | 0.3553 | 1.3791 | 0.0011 | 14.7570 | 5.2869 | 0.1456 | 0.2772 | 0.2958 |
| 30 | 0.7171 | 0.4476 | 0.0053 | 1.9231 | 9.0853 | 0.0737 | 0.8292 | 0.2894 |
| 40 | 1.1567 | 0.4465 | 0.0137 | 1.7594 | 12.867 | 0.0233 | 1.6608 | 0.2589 |
| 50 | 1.6537 | 0.4349 | 0.0265 | 1.1979 | 16.464 | 0.0061 | 2.7106 | 0.0885 |
| 60 | 2.1895 | 0.1096 | 0.0482 | 0.8299 | 19.836 | 0.0303 | 3.9368 | 0.0585 |

Table 6.1: Simulated values of $P(\sum_{i=1}^{k=2} R_i < D)$ and $P(\sum_{i=1}^{k=4} R_i < D)$ for pareto lifetimes are shown in the 'sim' column. The values indicate the percentage of dropped relay calls in $10^7$ runs. The relative error of the approximation $P(R < D)^{k=2}$ and $P(R < D)^{k=4}$ with respect to the simulated values is shown in the 'rel-e' column. Call duration is exponentially distributed.

*Proof:*

$$P(\sum_{i=1}^{k} R_i < D) \leq P(\max(R_1, \ldots, R_k) < D)$$

$$P(\max R_i < D) = P(R_1 < D, \ldots, R_k < D)$$

$$= P(R < D)^k \quad \text{since } R_i \text{ are i.i.d}$$

$$P(\sum_{i=1}^{k} R_i > D) \geq 1 - P(R < D)^k$$

Observe that if node lifetimes are exponentially distributed, the equality holds in Equation (6.7) holds and Equation (6.4) is obtained. For non-exponential node lifetimes, the $k^{th}$ exponentiation decreases much faster than the sum and intuitively, the bound is loose for large values of $k$. However, the relative error of the bound depends on the lifetime and call duration distributions. Next, we examine the relative error of Equation (6.7) for pareto distribution since the measurement studies of Skype node lifetimes suggest using heavy tailed distributions as an approximation [Guha *et al.*, 2006] and pareto is the most natural choice for such an approximation.

#### 6.4.1.1 Pareto Node Lifetimes

The CDF of pareto lifetimes is $F(x) = 1-(\frac{x}{b})^{-a}$, where $a$ is the shape parameter and $b$ is the scale parameter. For our analysis, we use the shifted pareto distribution $F(x) = 1-(1+\frac{x}{b})^{-a}$ with mean $\frac{b}{a-1}$ [Leonard *et al.*, 2005], because without the shift, a node is guaranteed to be up for $b$ units of time. Clearly, the mean of this distribution is only defined for $a > 1$ whereas variance is only defined for $a > 2$ which prevents the calculation of an an exact analytical formula for sum of $k$ pareto i.i.d r.v's. Zaliapin *et al.* [Zaliapin *et al.*, 2005] describe methods for approximating the upper quantile (0.98), lower quantile (0.02), and median of sum of $k$ pareto i.i.d r.v's. Their results indicate that although replacing the sum with the maximum can reasonably approximate the quantiles around median, such an approximation is poor for the lower and upper quantiles and for large values of $k$ (e.g., $> 10$). The CDF of residuals of pareto lifetimes is $F(x) = 1 - (1 + \frac{x}{b})^{1-a}$ [Leonard *et al.*, 2005]. Although, the approximation results by Zaliapin can be extended to the sum of pareto residuals for arbitrary values of $a$, $b$, and $k$, such an effort is beyond the scope of this thesis. Further, the utility of precise approximation may be limited due to the difficulty in estimating the pareto parameters. Also, real node lifetimes do not follow a strict pareto distribution and incorporate effects such as diurnal variations [Guha *et al.*, 2006; Kho *et al.*, 2008]. Therefore, to obtain a bound on the minimum number of relays to achieve desired reliability, we approximate the sum of $k$ pareto residuals with their maximum, but note that in doing so, it is necessary to get an estimate of the relative error of such an approximation to determine its usefulness.

In Table 6.1, we show the simulated values of the sum of two and four pareto residual $R_i$ being less than exponentially distributed call holding times $D$ and the relative error of the approximation (maximum of $R_i$ being less than $D$) with respect to the simulated values. The simulated results are an average over $10^7$ runs. The parameters $a$ and $b$ were chosen so that the mean of the distribution was five and one hour, respectively. The choice of mean uptime of five hours approximately reflects the median of the observed Skype node lifetimes [Guha *et al.*, 2006; Kho *et al.*, 2008], whereas mean node lifetime of one hour is for a relatively less stable system. The top two values in the fourth column are zero because sum of four $R_i$ was never observed to be smaller than $D$ (with mean of 2.5 and 5 minutes)

in $10^7$ runs. For relayed calls, these values are interpreted as observing no call failure in $10^7$ runs. Observe that the relative error is low ($< 0.2\%$) when the value of the simulated sum of $R_i$ r.v's being less than $D$ is above 2% whereas the relative error increases for simulated values smaller than 2% and the increase in number of summands from two to four. This result is consistent with [Zaliapin *et al.*, 2005] which notes that using the maximum of $k$ pareto r.v's instead of their sum is not a good approximation for lower quantiles ($< 0.02$). However, note that although the relative error increases as call holding times, $D$, decrease relative to node lifetimes and the number of summands $k$ increase, we are only interested in the smallest value of $k$ for which the call success rate is just above the desired reliability such as 99.9% and not an arbitrary large value of $k$. In general, the approximation can be applied to determine the smallest value of $k$ that meets the desired reliability criteria, as long as the relative error remains low (e.g., $< 1\%$).

Next, we present two schemes for preventing the failure of relayed media sessions due to relay churn.

### 6.4.2 No-replacement Scheme

In the no-replacement scheme, $k$ relays are selected at the beginning of the call with one relay acting as primary and $k-1$ acting as backup. If the primary relay fails, the call is switched to a backup relay. We assume that calls are not dropped during switch over. A call fails when all $k$ relays fail. Let $R_i$ be a random variable that denotes the residual lifetime of the relay $i$ when it is drafted as a relay and $D$ be a random variable that denotes call duration. We are interested in the probability that at least one of the relay, that were selected when call was established, is online before the call completes:

$$\begin{aligned} & P(\max(R_1, \ldots, R_k) > D) \\ & = 1 - \int_0^\infty P(R < z)^k P(D = z) dz \quad (R_i \text{ are i.i.d}) \end{aligned} \tag{6.8}$$

We solved Equation (6.8) to determine the proportion of successful relay calls using two or three relays when node lifetimes are exponentially distributed, and the corresponding expressions are $1 - \frac{2\nu}{\lambda+\nu} + \frac{\nu}{2\lambda+\nu}$ and $(\frac{1}{2\lambda+\nu} - \frac{1}{3\lambda+\nu})\frac{6\lambda^2}{\lambda+\nu}$, respectively. For pareto node lifetimes,

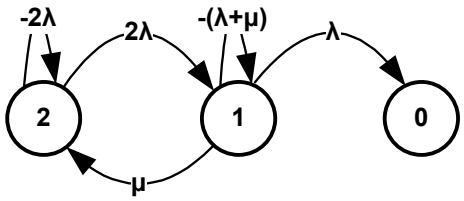



Figure 6.1: Markov chain for a 2-relay with-replacement scheme.

we numerically solved Equation (6.8) to obtain the proportion of successful calls using two or three relays.

**How many relays?** Does adding more relays at the beginning of a relayed call result in significant reliability improvement? Unfortunately, the proportional increase in the reliability diminishes with selecting more relays at the start of the call. For example, when node lifetimes are exponentially distributed, the MTTF of a 2-relay, 3-relay, and 4-relay schemes are $\frac{3}{2\lambda}$, $\frac{11}{6\lambda}$, and $\frac{25}{12\lambda}$, respectively. The proportional increase in the MTTF for these schemes is 50%, 22%, and 13%, respectively. Clearly, this is a case of diminishing returns. Further, maintaining numerous backup relays exclusively for every call when relays are not plentiful is likely to result in a poor performance from the perspective of successful call establishment for relayed calls. Moreover, nodes in a media session also incur the overhead of sending keep-alive traffic to many relays.

### 6.4.3 With-replacement Scheme

This scheme is similar to the no-replacement scheme in that $k$ relays are selected at the beginning of a call, and a call is switched to a backup relay if the primary relay fails. However, when a caller or callee detects that one of the $k$ relays has failed, it launches a search to replace the failed relay. Suppose it takes $\mu$ time units to detect that a relay has failed and find a new relay. If node lifetime and search time are exponentially distributed, a Markov chain can be used to evaluate the reliability of this scheme [Birolini, 2004]. For a single backup relay, the Markov chain is shown in Figure 6.1. In the reliability literature, this scheme is referred to as 1-out-of-2 active redundancy with constant failure rate $\lambda$ and constant repair rate $\mu$ [Birolini, 2004]. This chain can be solved to obtain MTTF, i.e., the time it spends in states (2) and (1), when two and one relays are operational. The failure

rate is the reciprocal of MTTF, i.e.,

$$\frac{1}{\lambda_{WR}} = MTTF = \frac{3\lambda + \mu}{2\lambda^2} \tag{6.9}$$

The subscript WR denotes with-replacement. For $\lambda << \mu$, this scheme approximately behaves like a one relay scheme with constant failure rate $\lambda_{WR}$ ([Birolini, 2004, page 190]). Let $R_{WR}$ be a random variable that denotes the reliability of this scheme. Since its failure rate is constant, its CDF is $R_{WR}(t) = e^{-\lambda_{WR}(t)}$. When call duration is exponentially distributed with parameter $\nu$, the probability that a call completes before the two relays fail and a search for the replacement relay also fails is:

$$P(R_{WR} > D) = \frac{\nu}{\nu + \lambda_{WR}} \tag{6.10}$$

When the node failure rates are not constant, either non-homogeneous poisson processes may be used to model the reliability of this scheme or node lifetime can be split into periods where failure rate is constant. However, the difficulty in using such analysis lies in the fact that for heavy tailed distributions, the shape parameter $a$ is often not accurately known. Therefore, we leave such analysis for future work.

### 6.4.4 Reliability of Relayed Calls in Skype

We performed experiments to determine if the Skype application employs a no-replacement or a with-replacement scheme. We blocked direct traffic between two machines running in our lab using NetPeeker [Net Peeker, 2010] and then ran Skype applications on them and established a call. Since the traffic was blocked between the machines, the Skype applications were forced to use a relay to exchange signaling and media traffic. Using NetPeeker [Net Peeker, 2010], we blocked the media traffic between caller machine and the relay, which is similar to emulating a relay failure. Within 2-4 seconds, the Skype applications chose a new media relay. We then immediately blocked traffic between this new relay and the caller Skype application which resulted into the call getting disconnected. The experiment shows that when a call is established that requires a relay, the Skype application chooses a backup relay at the start of the call. When both relays fail simultaneously, the

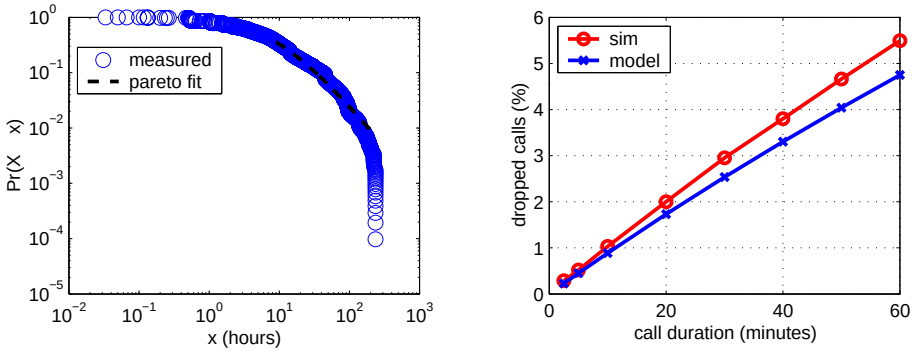



Figure 6.2: (Left) CCDF of the node lifetimes and the pareto fit for Skype data set (right) percentage of dropped calls through simulations on the Skype data set and using a pareto model when only one relay is used.

call is disconnected.

To determine if a Skype application searches for a new relay when the primary relay fails and the call is shifted to the backup relay, we gradually increased the time between primary and backup relay failure from 30 s to two minutes. Our experiments indicated that a Skype application approximately waits for a minute before searching for a new relay. Thus, it employs a 'periodic-recovery' scheme for replacing a failed relay instead of a 'reactive-recovery' scheme. We periodically failed the primary relay every 90 s for a call lasting 15 minutes and found that the Skype application was able to find a backup relay and the call did not get disconnected.

All the experiments were performed during the first week of December 2009 and more than 70 calls were established over a period of seven days.

### 6.4.5 Evaluation and Discussion

We evaluated the analytical model for the number of relays, and reliability improving techniques using simulations. We wrote an event driven simulator in which nodes form an overlay network using Chord. All nodes running the Chord overlay can potentially provide the relay service. The request for relaying a call randomly arrives on any of the nodes in the Chord overlay. We used a relay selector which randomly selected a relay from the pool of all online relays having sufficient network capacity. The inter-arrival time between

requests for relayed calls was exponentially distributed and its mean was adjusted over the course of the simulation so that the cumulative network load of relayed calls did not exceed a target aggregate network utilization of the peers. In the results presented in this section, the aggregate uplink network utilization of all the peers never exceeded 40%. Thus, in our simulations, the relayed calls only failed due to relay failure and not due to the scarcity of relays. We ran the simulation for 10 days of simulated time and repeated the experiments until $10^7$ call attempts had been made. The warm up period is excluded from the reported results.

We used three node lifetime data sets. The first two data sets contained synthetically generated exponential and pareto node uptime and downtime with a mean of 300 minutes. The pareto parameters $a$ and $b$ were chosen as 2 and 5, respectively. The third data set contained the uptime and downtime of 4,000 Skype super nodes measured for 25 days by Guha [Guha *et al.*, 2006]. The uptime of Skype nodes was measured by sending a specially crafted Skype message to these nodes every 30 minutes. We randomly selected 1,740 nodes from this data set of 4,000 nodes because this is the maximum number of nodes for which all pair ping latency data is available [Gummadi *et al.*, 2002]. In Section 6.5.3, we use this data for designing a distributed relay search mechanism that minimizes the latency of relayed calls.

At any instant during simulation, the online nodes amongst the 1,740 nodes ran the Chord overlay and could potentially provide the relay service. The median and mean uptime of these 1,740 nodes was 256 and 711 minutes, respectively. The pareto parameters, $a$ and $b$, computed using the method of maximum likelihood and Kolmogorov-Smirnov statistic, were 1.4916 and 8.9833, respectively. Figure 6.2 (left) shows the CCDF of Skype node lifetimes and the pareto fit indicated by a dashed straight line. Towards the end of the tail, the measured lifetimes exhibit a knee of the curve. This happens because the node lifetimes did not strictly exhibit a pareto behavior and the measurement is stopped after $T$ time units. For the Skype data set, Figure 6.2 (right) shows the percentage of dropped calls when a call is assigned to one relay, through simulations and those predicted by the model $P(R < D)$. The relative error with respect to simulations was less than 15%. Wang [Wang *et al.*, 2009] suggested that there is an inherent inaccuracy in computing the exact

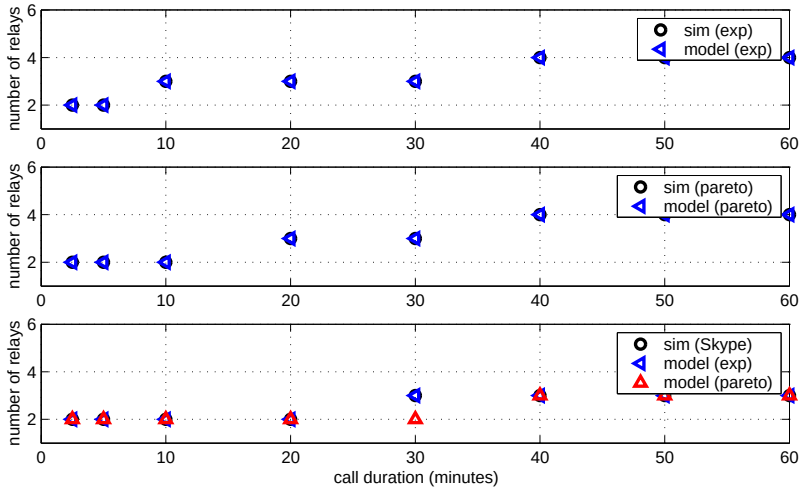



Figure 6.3: Number of relays needed to maintain a 99.9% call success rate when node lifetimes are distributed as exponential (top), pareto (middle), and Skype (bottom). The mean node lifetime for exponential and pareto distributions was 300 minutes. The mean and median node lifetime in the Skype data set was 711 and 256 minutes, respectively.

parameters of the node lifetime distribution when they are sampled every $T$ time units. We note that such a bias depends on the ratio of the mean node lifetime and the sampling interval: the higher the ratio, the lower the inaccuracy and vice versa. Nevertheless, we note that when the real lifetime data is used for churn simulations, such a bias will always be present.

A key consideration is to realistically set the upload and download bandwidth of a relay peer since it cannot relay an arbitrary number of calls. Dischinger *et al.* [Dischinger *et al.*, 2007] have measured the upload and download bandwidth for a range of broadband hosts and we set the relay bandwidths according to their reported distribution. We assume that a relay call needs an uplink and downlink bandwidth of 128 kb/s (using the G.711 codec). Modern codecs such as SILK [Skype, 2010c] which has a bit rate between 4-40 kb/s can bring down the required bandwidth at a relay to 8-80 kb/s. To simulate the effect of network traffic belonging to other applications, we randomly set the uplink network utilization of a node between 10-30% of its uplink capacity at the start of the simulation. Depending on its spare capacity, a relay peer can relay more than one call.

Figure 6.3 shows the number of relays for exponential, pareto, and Skype node life-

times for a range of exponentially distributed call holding times. Guha [Guha *et al.*, 2006] showed that 95% of Skype relayed calls last less than an hour. The approximation from Equation (6.7) is used to calculate the number of relays when pareto distribution is used to model node lifetimes. For pareto node lifetimes (second graph in the figure from the top) and call duration of 60 minutes, the relative error of the approximation was less than 1%. The results from the simulation show that for the Skype data set and for call durations of 60 minutes or less, three relays are sufficient to achieve a call success rate of 99.9%. Observe that modeling the Skype node lifetimes as exponential and pareto resulted in a minimum relay prediction of three relays which matches the simulations. For call duration of 30 minutes, the pareto model under predicts the number of relays. However, this is expected as Skype node lifetimes do not exactly follow the pareto model (Figure 6.2). Also, for the results shown, note that although only three or four relays or less are needed to achieve call drop rate of 0.1% or less for call duration of 60 minutes, the number can be higher when node lifetimes are smaller. As an example, when the node lifetimes are exponential with a mean of one hour, at least ten relays per call are needed to achieve a success rate of 99.9% for mean call duration of 60 minutes.

Figure 6.4 shows the reliability of a 2-relay and 3-relay no-replacement scheme for exponential, pareto, and Skype node lifetime data sets. As expected, there is a good match between analytically computed (using Equation (6.8)) and simulated call success rates for exponential and pareto node lifetimes. For the Skype data set, the simulations show that a 2-relay scheme achieves a 99.9% success rate for call durations of 10 minutes or less whereas for call duration of 60 minutes, the success rate is 99.25%. For 2-relay no-replacement scheme, using exponential and pareto node lifetimes to model Skype node lifetimes results in over predicting and under predicting the number of dropped calls by approximately a factor of two, respectively.

Figure 6.5 shows the reliability of a 2-relay with-replacement scheme for exponential, pareto, and Skype node lifetime data sets. The time to detect if a relay has failed and consequently to search a new relay is exponentially distributed with a mean of 60 s. As expected, the Markov model accurately predicts the call drop rate when node lifetimes are exponential. The results also indicate that the Markov model may be a reasonable

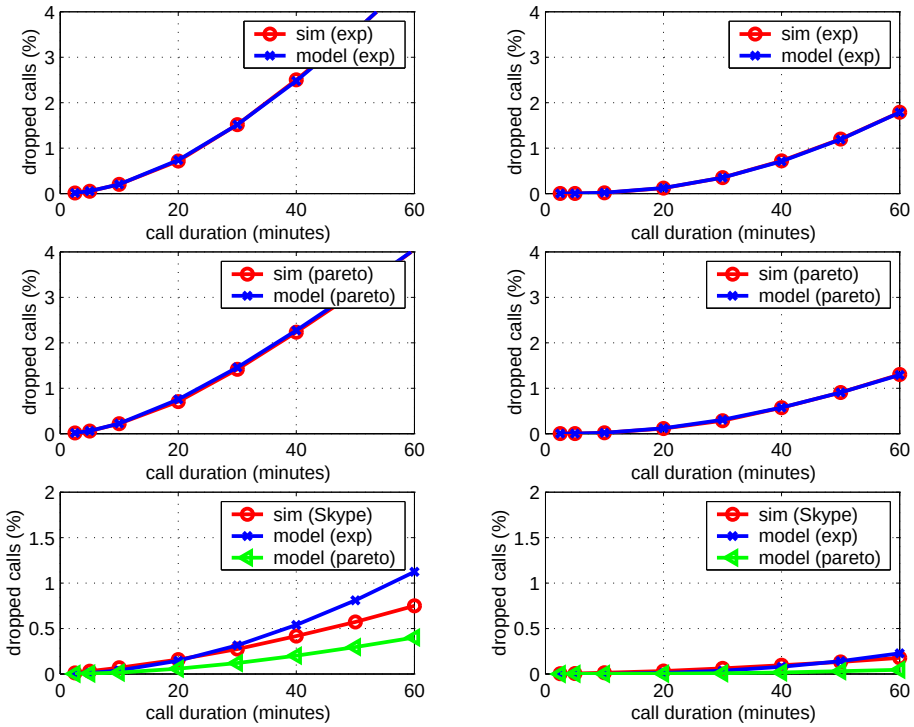



Figure 6.4: Proportional of failed calls using simulations and model for exponential (top), pareto (middle), and Skype (bottom) node lifetimes. The figures on the left and right are for a 2-relay and 3-relay no-replacement scheme, respectively.

approximation for pareto node lifetimes. For Skype data set and for call duration of 60 minutes, this scheme achieves a call success rate of 99.65%, an improvement of 0.3% over a 2-relay no-replacement scheme. The improvement is small because node lifetimes have a large mean (711 minutes). When node lifetimes have a small mean, it may be necessary to incorporate a with-replacement scheme to avoid dropped calls. Since Skype employs a 2-relay with-replacement scheme having a relay search time of approximately 60 s, the results from our simulations indicate that the drop rate of relayed calls is likely to be small. However, Skype's relay mechanism is not completely random and is biased towards low latency and high bandwidth relays. Such a bias may result in higher drop rates for relayed calls [Godfrey *et al.*, 2006]. Nevertheless, an implication of the results is that for Skype node lifetimes, simple schemes for reliability improvement such as two relay no-replacement and with-replacement give reasonable reliability performance thereby obviating the need for

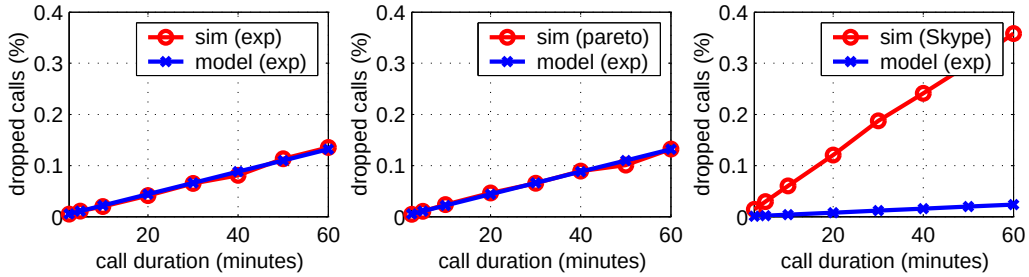



Figure 6.5: Proportion of failed calls using simulations and Markov model for 2-relay with-replacement scheme for exponential (left), pareto (middle), and Skype (right) node lifetimes.

a more sophisticated reliability improvement scheme.

#### 6.4.5.1 Practical Implications of Reliability Improvement Schemes

In a $k$-relay no-replacement scheme, both caller and callee exchange information about $k$ relays at the time of call establishment. After a call has been established, they must periodically check the liveness of all $k$ relays. The liveness period should be adjusted so that when the primary relay fails, there is a high likelihood that the new relay to be incorporated is alive. However, the reliability returns of maintaining a large number of backup relays at the start of the call are diminishing, especially under high churn. For this reason, a with-replacement scheme is attractive. Such a scheme can potentially start with 2 or 3 relays, and find a replacement for a failed relay. However, the caller and callee must exchange information about the new relay in subsequent signaling messages. For a no-replacement scheme, no such exchange is required.

#### 6.4.5.2 Other Reasons for Call Failure

Relay failure is not the only reason why relayed calls may fail. Such calls can also fail during call switching. Also, since nodes may use silence suppression, it may take more time to correctly distinguish between silence periods and a failed relay because the frequency of heart-beat messages is likely to be lower than real-time voice or video packets. If a search for a relay is launched at the time when all relays have failed, the caller and callee will perceive a silence gap in the conversation. If the duration of the perceived gap is long, the

call participants may simply terminate the call.

## 6.5 Relay Selection

In this section, we devise distributed techniques to find a relay that address several practical issues. The first issue is that the distributed relay search must find a relay in a timely manner to minimize the call establishment time and to quickly recover from relay churn. Also, the relaying of media session can interfere with the user applications and impair their performance. It is important to select relays in a way that minimize this interference. Besides minimizing interference, latency and increasing reliability are key objectives for relayed calls. Addressing all these factors is a multi-objective optimization problem which is NP-hard [Zaroliagis, 2005].

In Section 6.5.1, we devise a distributed relay selection technique that can find a relay in $O(1)$ hops and compare its performance to a scheme that randomly selects a relay from the global pool of all relays. Section 6.5.2 introduces the notion of user annoyance. In Section 6.5.3, we augment the distributed relay selection scheme to devise heuristics for finding a relay that, for a relayed call, minimizes user annoyance or latency or both, and evaluate their performance.

### 6.5.1 Distributed Relay Selection

We devise a relay selection scheme where a node requesting a relay can find a relay in $O(1)$ hops. As mentioned earlier, quickly finding a relay is necessary to reduce call establishment time and to recover from relay churn. The key idea to accomplish this goal is to construct a two tier peer-to-peer network. All peers in the top tier provide routing services and can also potentially provide relay services. The peers form the top tier network using any structured or unstructured p2p protocols. Each peer maintains a data structure called a routing table to maintain connectivity with other peers in the overlay. Each entry in this table contains the network address and round-trip time of a reachable peer in the overlay. As part of keep-alive messages to check the liveness of entries in its routing table, a peer also exchanges information with its routing table entries on how many relay calls they can

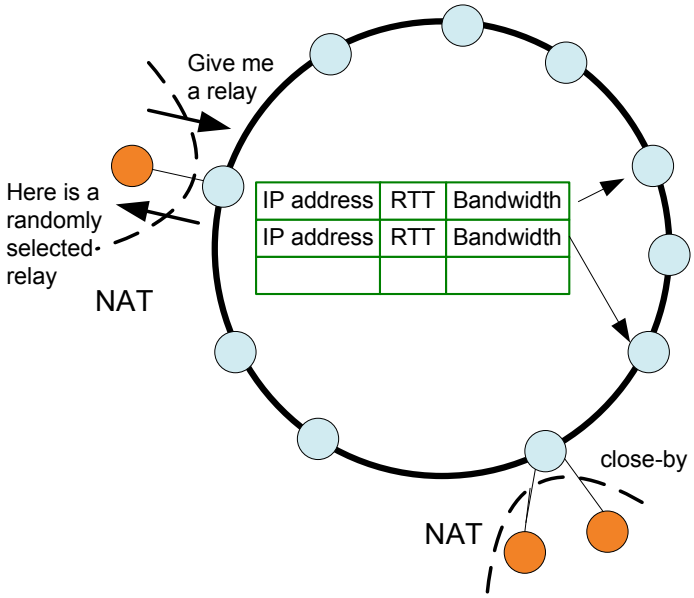



Figure 6.6: Two-tiered overlay implementing a *local-random* scheme for relay selection. The nodes shown in light blue color and connected by a circle form the top tier, whereas the nodes in the lower tier, as shown by the orange color, connect to one of the nodes in the top tier.

support, their uptime and the time for last user keyboard or mouse activity.

The nodes in the lower tier are typically behind a restrictive NAT or a firewall and are connected to close-by peer(s) in the top tier in terms of network latency. These nodes may need a relay need a relay peer for establishing a media session. Such nodes send a request to their connected peer which consults its routing table and returns to the requesting node a set of available relays. If none of the peers in the routing table can fulfill the relay request, the peer forwards the request to a randomly selected peer in its routing table, which in turn consults its routing table for available relay peers. The number of forwarding hops is bounded by a constant such as four. As an example, if on average 30% of the nodes in a peer's routing table are busy routing a call, then the probability of not finding an available relay after traversing four randomly selected hops is less than one percent (0.81%).

If the number of relay requests is low and uniformly distributed across all peers, this scheme is likely to find a relay in $O(1)$ hops. We refer to this scheme as *local-random* scheme because it selects a relay by leveraging the local overlay view of a peer. This scheme is in

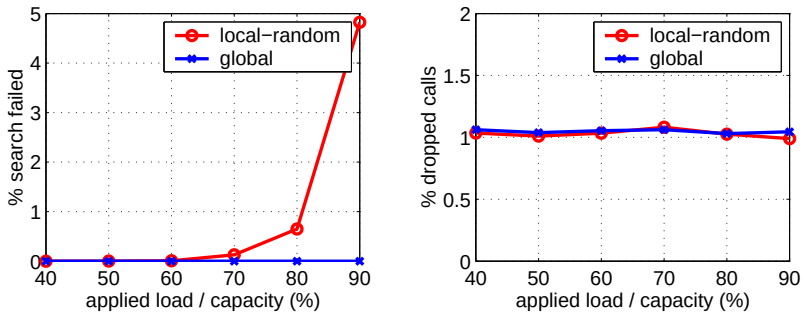



Figure 6.7: Performance of local-random scheme vs. global scheme as a function of system load (left graph). Percentage of dropped calls when one relays fails (right graph).

contrast to a global random scheme, which has knowledge of all relays in the system and randomly picks a relay from this global pool. Figure 6.6 shows an illustration of this scheme.

We evaluate the performance of this scheme through simulations. We use Chord [Stoica *et al.*, 2003] as the overlay protocol. Each Chord peer maintains a randomized routing table [Godfrey *et al.*, 2006] instead of a deterministic table, i.e., for the $i^{th}$ routing table row, it picks any node with an ID in the interval $[ID + 2^i, ID + 2^{i+1})$. This is so because Godfrey *et al.* [Godfrey *et al.*, 2006] showed that the randomized scheme for populating routing tables has a better performance under churn. The Chord network is run on 1,740 nodes that follow the Skype node lifetime distribution as discussed in Section 6.4.5. The requests for relaying a media session arrive at any relay and are uniformly distributed across relays. Intuitively, the local-random scheme may have poor performance when relay requests are concentrated on few peers. However, this issue is easily addressed if a peer unable to fulfill the relay request forwards it to a randomly selected peer in its routing table.

The metric for evaluating the performance of this scheme is its ability to find a relay compared to a scheme with global knowledge of all relays for an increasing number of relay requests. The inability to find a relay impacts the success rate of relayed calls (Equation (6.1)). The relay search is likely to fail when the number of relay requests is close to or exceeds the network capacity of the peers. If the percentage of relay peers that are relaying calls is low, then local-random scheme is likely to find a relay. However, this may not be the case when the number of relay requests is close to the capacity of the system. Figure 6.7 plots the percentage of calls that fail to find a single relay. For the results shown, the

local-random scheme did not forward the relay request to any peers. The $x$-axis is the ratio of the applied load to the total relay capacity of all relays. The figure shows local-random scheme is unable to find relays when there are few relays available. However, this scheme gives comparable performance in terms of percentage of dropped calls due to relay failure even under heavy relay request load because local relay selection is still random similar to the global random selection scheme.

### 6.5.2 User Annoyance

A key difference between p2p file-sharing and communication systems is in their approach to free-riders. The tit-for-tat mechanism for sharing file chunks in BitTorrent-like fileshar-ing systems aims to minimize the impact of free-riders that are not willing to share files or are behind restrictive NAT and firewalls. Such nodes can only download files at a re-duced rate [Dessent, 2010]. Reducing rate may not be an option in p2p communication networks because it can affect the quality of audio, video, and conference calls. Thus, in contrast to a p2p file-sharing system, a p2p communication system must provide accept-able service to nodes behind restrictive NATs and firewalls. This key requirement together with the distributed nature of a p2p communication network implies that nodes in the p2p network with unrestricted connectivity must relay calls for nodes with restrictive network connectivity. Consequently, the relayed calls may interfere with user applications running on these altruistic peers especially those that require network connectivity. We refer to any interference (e.g., network, CPU, memory) with applications running on the relay machine as 'user annoyance'.

We focus on characterizing the user annoyance and augmenting the relay selection scheme to minimize user annoyance. User annoyance for relayed calls can also be reduced by providing incentives. However, in a system where proportion of relayed calls is much smaller than the number of available relays, it may be possible to avoid peers where a relay call is likely to cause a high interference with the user applications, and thus bypassing the issue of providing incentives.

The question is how to measure user annoyance. Since relayed calls are network centric and since it is difficult to accurately estimate the perceived impact of the relayed calls on

user applications, we use a simple measure, i.e., the spare network capacity of the network connection of the user's machine to estimate the user annoyance. The higher the spare network capacity, the smaller the likelihood of annoyance of a user whose machine would be used as a relay. This simplistic measure may not accurately measure user annoyance; however, it is more practical than the approaches which require instrumenting the user applications to determine the precise impact of relayed calls. A peer can periodically measure its uplink and downlink capacity, say, every 30 minutes and by determining the current network usage, gauge its spare network capacity which it can then advertise to peers in its routing table. We use this technique in our PlanetLab implementation (Section 6.5.4).

#### 6.5.2.1 Estimating Spare Network Capacity

Measuring user annoyance requires estimating of the capacity of the network link. Unlike CPU, memory, and disk, it is non-trivial to estimate the network capacity. To an extent, this depends on the type of network link. On point-to-point dial-up connections, the maximum link speed is typically determined by the speed of the modem. As DSL and cable Internet penetrates homes and the use of WiFi routers at home becomes common, a device no longer directly connects to the ISP in a way similar to dial-up; rather, a device connects to a WiFi router which connects to the DSL or cable modem, which in turn is connected to the ISP. Using the link speed of the connected WiFi link will overestimate the machine-to-ISP link capacity by a large margin.

We suggest three approaches for determining the machine-to-ISP link capacity in the presence of intermediate devices such as WiFi routers, and cable or DSL modems. The first approach uses the fact that link capacity is agreed upon between ISP and customer when the latter purchases a broadband plan. The idea is to design protocols which allows ISP to pass this link capacity to the cable or DSL modem which in turn passes this information to downstream devices such as WiFi routers or laptops. This idea can be implemented as a DHCP option, for example. The problem with this approach is that ISPs typically perform statistical multiplexing on multiple flows, and the instantaneous capacity of the link may be less than the purchased capacity. Also, this technique requires changing the already deployed cable/DSL modems and WiFi routers, which is a non-trivial task. Nevertheless,

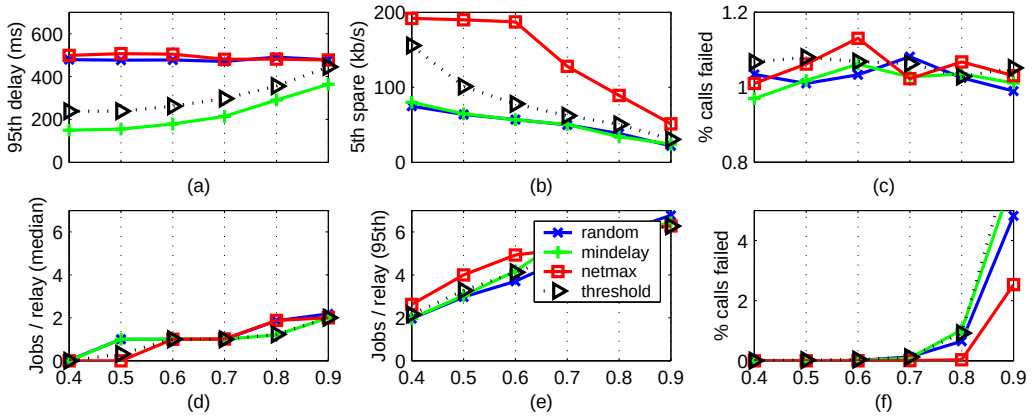



Figure 6.8: The x-axis represents the ratio of bandwidth consumption of total number of calls in the system to the total network capacity of all nodes. (a) 95th delay (ms) of completed calls (b) 5th percentile of spare network capacity (c) percentage of failed calls due to relay churn (d)(e) median and 95th percentile of number of jobs per relay (f) percentage of calls that fail to find a relay.

it is a solution that does not require p2p applications to perform any network capacity measurements. In the second approach, a p2p application can perform measurements to estimate the link capacity by sending a train of packets to other peers in the p2p network using tools such as LinkWidth [Chakravarty *et al.*, 2008] or Pathchar [Jacobson, 2010]. Third, an operating system or the p2p application can keep track of the maximum data rate seen on the machine's network link within a recent time window and use it as an estimate of link capacity. However, when multiple machines are connected to the same network device such as WiFi router, using the capacity usage of a machine's network link will not reflect the total traffic flowing through the router. Therefore, we use the second approach in our PlanetLab implementation.

### 6.5.3 Heuristics

Besides minimizing user annoyance, it is necessary to minimize the delay of a relayed call and increase its reliability. In essence, this is a multi-objective optimization problem. We devise heuristics to optimize these metrics and evaluate their performance.

In Section 6.5.1, we constructed a two tier overlay network and peers in the top tier

maintain information about the round-trip time, spare network capacity, and uptime of the nodes in their routing table. Peers can periodically exchange this information, perhaps as part of keep-alive messages. A node searching for a relay then sends a request to its connected peer which applies the heuristics and returns a set of candidate nodes.

Below, we discuss the heuristics for selecting a relay peer from a candidate set returned by the local-random scheme.

- *Random:* Select a node randomly from the candidate set.
- *NetMax:* Select the node with the maximum spare network capacity.
- *MinDelay:* Select the node that has the smallest RTT.
- *Threshold:* Select the node that does not add more than 200 ms of delay on top of the direct network latency between caller and callee, and has maximum spare bandwidth. If no candidate meets the criteria, randomly select any node.

Figure 6.8 shows results for these heuristics. The results were obtained through simulations on a 1,740 node Chord network, with node lifetimes taken from the Skype data set as described in Section 6.4.5. We assume that the network latency between the clients and their connected peers is very small (close to zero). This assumption is reasonable because clients will likely connect to minimum latency peers to use overlay services. The heuristics are evaluated according to several metrics. The first metric is the 95th percentile of the total delay of a relayed call minus the direct network latency between session peers. A large value of this latency indicates that relay selection is poor from the perspective of delay. The second metric is the 5th percentile of the absolute spare capacity on relay nodes. A larger spare capacity signifies that high capacity relays are selected. The third metric is the percentage of calls that fail due to relay failure. The fourth metric is the median and 95th percentile of the number of relayed calls per relay. If the relay selection is biased towards relays with large spare network capacity, the percentage of relays with large spare network capacity are few, and relayed calls are not plenty, then the relay jobs will be concentrated on these high capacity peers with the result that the median number of jobs per relay is

likely zero whereas the 95th percentile of the number of jobs is significantly greater than one. The last metric is the percentage of calls that cannot find a relay.

The results shows that *MinDelay* heuristic gives the best delay performance (Figure 6.8(a)). *NetMax* heuristic ensures that relays with large spare network capacity are preferred over relays with small spare capacity and achieves the best performance for user annoyance as indicated by a large value of 5th spare network capacity in Figure 6.8(b). However, this scheme has a consequence that more calls are assigned to few high capacity nodes, making these calls more vulnerable to node failure (Figure 6.8(c)). The *Threshold* approach gives the best performance in terms of minimizing latency and user annoyance. The *Threshold* scheme has a slightly high call drop rate due to failed relays but this can be improved by biasing relay selection towards idle nodes, e.g., machines with no keyboard or mouse activity within a time period. All heuristics have similar performance in terms of their ability to find a relay under increasing load.

As mentioned in Section 6.5.2, spare network capacity is a simplistic measure to estimate user annoyance. In addition to spare network capacity, machine idle time is a useful measure for relay selection. The idea is to select a relay with spare capacity that has been idle for sometime. The use of idle time as a relay selection metric is motivated by the SETI@home project [SETI@home, 2010]. SETI@home runs compute jobs as a screen saver on idle machines that are distributed around the world. When this approach is used in a p2p communication network, peers participating in the top-level hierarchy inform peers in their routing table how long they have been idle and whether they are in the screen saver mode. A node in need of a relay then selects a peer that meets the delay constraint, has been idle, and has the maximum spare capacity.

Figure 6.7 showed that search for relays starts to fail when the requests for relay calls are close to or exceed the total network capacity of the relay nodes. This is unacceptable for an overlay provider like Skype. The only solution for the overlay provider is to provision the p2p applications with on-demand media relay servers that are hosted within a data center. When nodes establishing a media session fail to find a relay peer, they send a request to the media relay server to relay the media session. Such a hybrid solution is necessary for a commercial p2p VoIP provider, if it needs to guarantee call establishment when there are

not enough relays in the system.

### 6.5.4 PlanetLab Deployment

To examine the feasibility of relay selection schemes, we implemented the *Random* and *Threshold* scheme in our OpenVoIP system Chapter 4). OpenVoIP is a two-level hierarchical overlay network deployed on PlanetLab that uses the Kademlia DHT [Risson and Moors, 2007]. We successfully scaled the top-level network to 1,000 peers that run on 500 PlanetLab machines. Each peer in the top-level network fully participates in the overlay and can act as a relay peer using the TURN protocol [Mahy *et al.*, 2010]. Further, each peer periodically performs uplink and downlink TCP throughput measurements and shares this information with its routing table nodes. Using TCP throughput provides a more conservative estimate of the link capacity than tools such as LinkWidth [Chakravarty *et al.*, 2008] or Pathchar [Jacobson, 2010]. In addition to sharing its uplink and downlink capacity measurements, a peer also shares its uptime with its routing table nodes. We have integrated p2p functionality with Wengo Phone, an open source SIP phone [OpenWengo, 2010] (now known as Qutecom [Qutecom, 2010]). This P2PSIP phone fully participates in the overlay if it is not behind a NAT or a firewall. Otherwise, it participates as a client. When two P2PSIP phones behind a restrictive NAT cannot establish a media session directly, they use a peer in the top-level hierarchy to relay the media session.

We have implemented the *Random* and *Threshold* scheme for relay selection. Our implementation of the *Threshold* scheme uses delay and spare network capacity metric. We do not use a SETI@home like technique for determining whether a machine is idle as the PlanetLab machines are not user desktop machines. The results for the *Threshold* scheme indicate that relay selection is biased towards nodes with maximum spare network capacity and low latency. We note that these relayed calls are real voice calls between two SIP user agents and are not emulated.

## 6.6 Related Work

There has been extensive research on constructing proximity aware DHTs [Rhea, 2005] and to minimize the impact of churn on DHT routing [Godfrey *et al.*, 2006]. Ren *et al.* [Ren *et al.*, 2006] showed through measurements that many relay peer selections in Skype are sub optimal, waiting time to select a peer can be quite long, and there are a large number of unnecessary probes. They designed an autonomous system aware p2p protocol (ASAP), which considers autonomous systems into peer relay selection. Their approach suffers from three limitations. First, when using DHTs, the network address of all relay peers within the same AS can get stored on a single node, creating a single point of failure. Second, their techniques do not incorporate interference of a relay session with the user applications. This is critical because users will not altruistically run a p2p application if it actively interferes with their applications. Finally, they provide no guidance on how many relay peers are needed to achieve desired reliability. Leonard *et al.* [Leonard *et al.*, 2005] analyzed the time to node isolation in DHTs for exponential and pareto residual lifetimes. However, our focus is on characterizing the reliability of relayed calls. Godfrey *et al.* [Godfrey *et al.*, 2006] analyzed the impact of churn on the DHT routing performance and suggested techniques to minimize such impact. Our relay selection techniques uses their random selection approach. However, it is imperative to explicitly devise schemes to prevent dropped calls. Tan *et al.* [Tan and Jarvis, 2007] presented an analysis to improve the reliability of DHT-based multicast by improving its delivery ratio. However, delivery ratio is not an appropriate metric to for analyzing reliability in peer-to-peer communication systems.

Connectivity issues due to NAT and firewalls also arise in p2p file sharing networks such as Kazaa [Liang *et al.*, 2004] and BitTorrent [BitTorrent, 2010]. BitTorrent allows nodes behind restrictive NAT and firewalls to download file chunks, albeit at a lower rate. To improve the download rate, the BitTorrent FAQ recommends that users configure the 'port forwarding' feature of NATs [Dessent, 2010]. Lowering rate is not an option in p2p communication networks because it can impact the quality of a call. Further, a user of the p2p communication may find it difficult to configure the NAT device and may abandon the p2p application in favor of a configuration-less communication application.

## 6.7 Conclusion

We have formalized the notion of reliability in peer-to-peer communication systems and designed a simple analytical model that predicts the reliability of relayed calls as a function of node lifetime and call duration distributions. Our analysis shows that for Skype node lifetimes and for call durations of 60 minutes or less, at least 2-3 relays are needed to achieve a 99.9% call success rate. We have presented two techniques for relay selection, namely, no-replacement and with-replacement, and used reliability theory to analyze them. We have observed that Skype follows a 2-relay with-replacement scheme, and it uses periodic recovery to replace a failed relay, and the search period is more than a minute. Our results indicate that exponential distribution, despite its limitations, is useful in analyzing the reliability of relayed calls.

We also introduced the notion of user annoyance which measures the interference of a p2p communication application relaying a call with other applications running on a machine. We have devised a distributed technique to find a relay in $O(1)$ hops. We augment this technique to find a relay that minimizes the latency of a relayed call and user annoyance. Finally, we have explored the feasibility of our relay selection schemes on a 1,000 node peer-to-peer communication system deployed on PlanetLab.

# Chapter 7

# Understanding TCP Behavior for Real-time Traffic

## 7.1 Introduction

The popularity of real-time applications, such as VoIP and video streaming has grown rapidly in recent years. The conventional wisdom is that TCP is inappropriate for such applications because its congestion controlled reliable delivery may lead to excessive end-to-end delays that violate the real-time requirements of these applications. This has led to the design of alternative unreliable transport protocols [Handley *et al.*, 2003; Kohler *et al.*, 2006] that favor timely data delivery over reliability while still providing mechanisms for congestion control.

Despite the perceived shortcomings of TCP, it has been reported that more than 50% of the commercial streaming traffic is carried over TCP [Guo *et al.*, 2006]. Popular media applications such as Skype [Baset and Schulzrinne, 2006] and Windows Media Services [Guo *et al.*, 2006] use TCP due to the deployment of NATs and firewalls that often block UDP traffic. Further, TCP is by definition TCP-friendly [Handley *et al.*, 2003] and is a mature and widely-tested protocol whose performance can be fine tuned.

The gap between the perceived shortcomings of TCP and its wide adoption in real-world implementations motivated us to investigate the delay performance of TCP. We seek to address the following questions: (1) Under what conditions can TCP satisfy the delay

requirements of real-time applications? (2) Can the performance of these applications be enhanced using simple application-layer techniques? We address these questions in the context of two real-time media applications that are characterized by timely and continuous data delivery: VoIP and live video streaming.

We used a test-bed with configured drop-rates and a drop-tail router to study the impact of TCP delay on the performance of real-time applications. We analyzed how the delay depends on the congestion control and reliable delivery mechanisms of TCP. The results obtained yield guidelines for delay-friendly TCP settings and may further be used to compare the performance of TCP with alternative protocols [Handley *et al.*, 2003; Kohler *et al.*, 2006] and experimental real-time enhancements for TCP [Goel *et al.*, 2002; Mukherjee and Brecht, 2000; McCreary *et al.*, 2005]. We analyzed two application-level schemes, namely, packet splitting and parallel connections, which we found to significantly reduce the delay of live video streaming flows.

Our research reveals that real-time application performance over TCP may not be as delay-unfriendly as is commonly believed. One reason is that the congestion control mechanism used by TCP regulates rate as a function of the number of packets sent by the application. Such a packet-based congestion control mechanism results in a significant performance bias *in favor* of flows with small packet sizes, such as VoIP. Second, due to implementation artifacts, the average congestion window size can overestimate the actual load of a rate-limited flow. This overestimation reduces the likelihood of timeouts and consequently the resulting TCP delay.

The rest of this chapter is organized as follows. Section 7.2 describes the application setting. Section 7.3 describes the experimental setup. Section 7.4 discusses the working region of VoIP and live video streaming flows, the impact of packet size and byte-counting on the delay performance, and playout buffer setting. Section 7.5 presents two techniques for improving the delay performance of real-time traffic, namely, packet splitting and parallel connections. Section 7.6 lists guidelines for setting the TCP, OS, and application parameters for real-time traffic.

## 7.2 Application Setting

We study a general real-time media application, with a Constant Bit Rate (CBR) source, that sends data across the network using TCP. CBR is the most basic and dominant encoding for media flows in the Internet [Wang *et al.*, 2007]. Although our analysis is general, we focus on CBR sources corresponding to VoIP and live video streaming, as detailed in Section 7.3. The VoIP and live video streaming flows are application-limited, i.e., their sending rate is a function of media encoding and not the underlying network. This is in contrast to greedy flows, such as FTP, which are network-limited.

Throughout this section, we refer to the transmission unit of TCP as a segment and to the TCP payload (i.e., the application-layer data unit) as a packet. The maximum segment size, MSS, is determined by the maximum transmission unit of the network path [Stevens, 1994]. A common characteristic of real-time applications is their sensitivity to the end-to-end delay which may vary from application to application. For live video streaming, there is usually minimal interactivity involved, so the application can afford a startup delay in the order of seconds [Guo *et al.*, 2006]. For VoIP, the delay must not exceed 400 ms in order to maintain acceptable interactivity [Goel *et al.*, 2002]. To reduce end-to-end delays, VoIP often uses small payloads (e.g., 160 byte packets) that correspond to 20 ms or 30 ms of audio. Thus, in the context of this study, the difference between VoIP and live video streaming flows is their packet sizes and their tolerance of delay.

### 7.2.1 TCP Interaction with VoIP-like Flows

The performance of real-time applications that use small packets (e.g., VoIP) is directly affected by whether the congestion control mechanism in TCP is byte or packet-based. According to [Allman *et al.*, 2009], there are two different issues at work. First, a TCP sender can track the congestion control state in terms of outstanding bytes or outstanding packets. Second, a TCP sender can update the congestion control state based on how many bytes are acknowledged, a mechanism known as byte-counting, or by some constant for each ACK arrival, a mechanism known as ACK-counting. We compare the performance of ACK and byte-counting mechanisms (Section 7.4.3) and focus on the former due to its

wide-deployment [Medina *et al.*, 2005], as also verified by our measurements.

## 7.3 Experimental Setup

We studied the delay performance of TCP both in a controlled network environment and via Internet experiments using PlanetLab and residential machines. We define TCP delay as the time it takes the application to get a packet from source to destination through a TCP connection. We use "CBR-TCP" to denote a TCP connection with a CBR source, "FTP" for a TCP connection with bulk data transfer, and "web" for a TCP connection with HTTP traffic.

We wrote a tool that can send and receive bidirectional CBR over TCP flows with different packet sizes and different packetization intervals, i.e., the gap between transmission of real-time packets. We use CBR sources with packet sizes of 174, 724 and 1448 bytes, and packetization intervals of 20 ms and 30 ms, as these choices approximately reflect typical one-way voice [Schulzrinne *et al.*, 2003], low bit rate interactive video [Goel *et al.*, 2002] and live video streaming [Guo *et al.*, 2006]. The size of the packet includes a 12 byte RTP header [Schulzrinne *et al.*, 2003] and two bytes for framing RTP packets over TCP [Lazzaro, 2006]. Hence, excluding the header size, the bit rate of the voice flows is 64 kb/s and 42 kb/s, that of interactive video is 284 kb/s and 187 kb/s, and that of live video streaming is 573 kb/s and 378 kb/s. Unless stated otherwise, we refer to the voice flow with a bit rate of 64 kb/s as 'VoIP' and the live video streaming flow with a bit rate of 573 kb/s as 'video'.

We abuse terminology and refer to the segment loss rate in the network as the packet loss rate. These loss rates may differ for VoIP flows because TCP can assemble several small packets into one segment during network-limited periods.

### 7.3.1 Configured Drop Rates

We measured the TCP delay of CBR flows on a test-bed that emulates a wide range of network settings. The topology of the test-bed is shown in Figure 7.1(a). We consider a single CBR-TCP flow going through a router running NIST Net [NIST Net, 2010], a network emulation program which can introduce constant delay and can drop packets according to

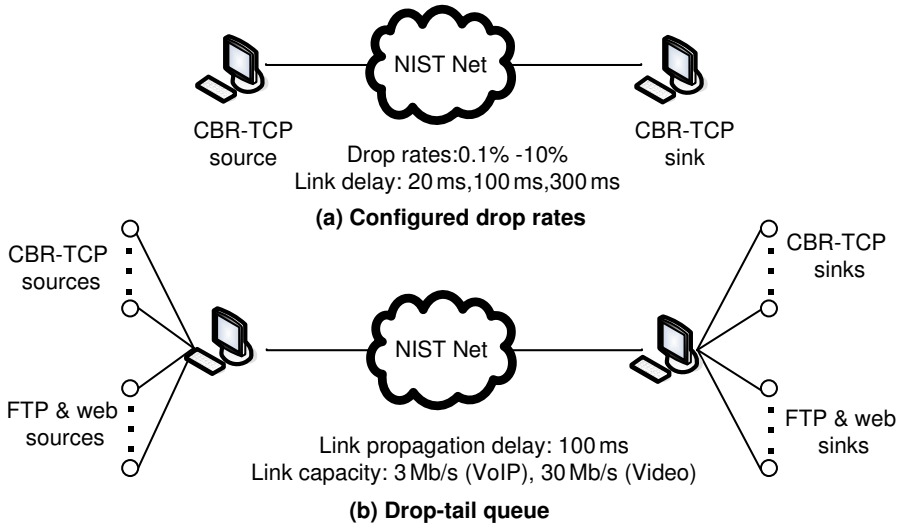



Figure 7.1: Experimental setup for analyzing the delay performance of TCP in a controlled environment.

a configured loss process. We configured NIST Net to drop packets uniformly at random with rates of 0.1%, 0.5%, 1%, 2%, 3%, 5% and 10%, irrespective of their size. We do not consider loss rates greater than 10% because the average TCP throughput (i.e., the available network bandwidth) does not satisfy the rate requirement of the CBR-TCP flow for the RTTs considered. We used fixed propagation delays of 20 ms, 100 ms, and 300 ms.

### 7.3.2 Using Drop-Tail Routers

We used the test-bed from Section 7.3.1 and modified NIST Net to incorporate a drop-tail queue. We devised a multi-flow setting in which five VoIP CBR-TCP flows compete with five long-lived FTP and varying number of web flows. We repeated the experiment for video flows. We used the SRI and ISI traffic generator [McKenney *et al.*, 2010] to generate exponentially distributed web traffic with a mean duration of 50 ms and a constant packet size of 512 bytes. The choice of the number of FTP and web flows, and packet size for web flows was inspired by the configuration used to evaluate the performance of TFRC-small packets [Floyd and Kohler, 2007]. The round-trip propagation delay was set to 100 ms for all experiments. The link capacity was set to 3 Mb/s and 30 Mb/s for voice and video CBR-TCP flows, respectively, so that the ratio of cumulative bit rate of five CBR-TCP

flows to link capacity was one to ten. The drop-tail queue was maintained in packets and configured to hold 100 packets. For each configuration, we ran the experiment for five minutes, repeated it five times, and present the average of the results.

## 7.4 Discussion

In this section, we explore the delay performance of real-time delivery over TCP. We experimentally characterize the working region for VoIP and live video streaming applications with bit rates of 64 kb/s and 573 kb/s, respectively. Then, we study the impact of various mechanisms in TCP on its delay performance.

### 7.4.1 Working Region

Here we characterize the working region for VoIP and live video streaming applications, i.e., the conditions under which the performance of these applications is satisfactory. In general, the user perceived media quality is acceptable when the fraction of packets that arrive beyond their playout time is low and the end-to-end delay is low.

For interactive applications, ITU G.114 recommends that the worst-case one-way delay should be 400 ms [International Telecommunication Union (ITU), 2003]. Studies show that 200 ms is an acceptable one-way delay limit for VoIP applications [Na and Yoo, 2002]. The choice of the delay limit for live video streaming is more flexible because people can usually tolerate a few seconds of startup delay. For the analysis, we consider a 5 s startup delay as suggested by [Guo *et al.*, 2006]. While VoIP can usually tolerate up to 5% of packets that miss their playout deadline without a significant effect on intelligibility [Na and Yoo, 2002], video viewing quality drops rapidly at 0.1% loss [Wang *et al.*, 2004]. We follow these guidelines and define the working region for VoIP and live video streaming as the range of network loss rates and RTTs where the 95th percentile and maximum TCP delay is at most 200 ms and 5 s, respectively. We explore how the performance varies with the delay limit.

Figure 7.2(a) plots the 95th percentile delay for various loss rates from 0.1% to 10% and RTTs of 20 ms, 100 ms, and 300 ms for a VoIP flow with a bit rate of 64 kb/s. The results shown were obtained empirically using the environment described in Section 7.3.1.

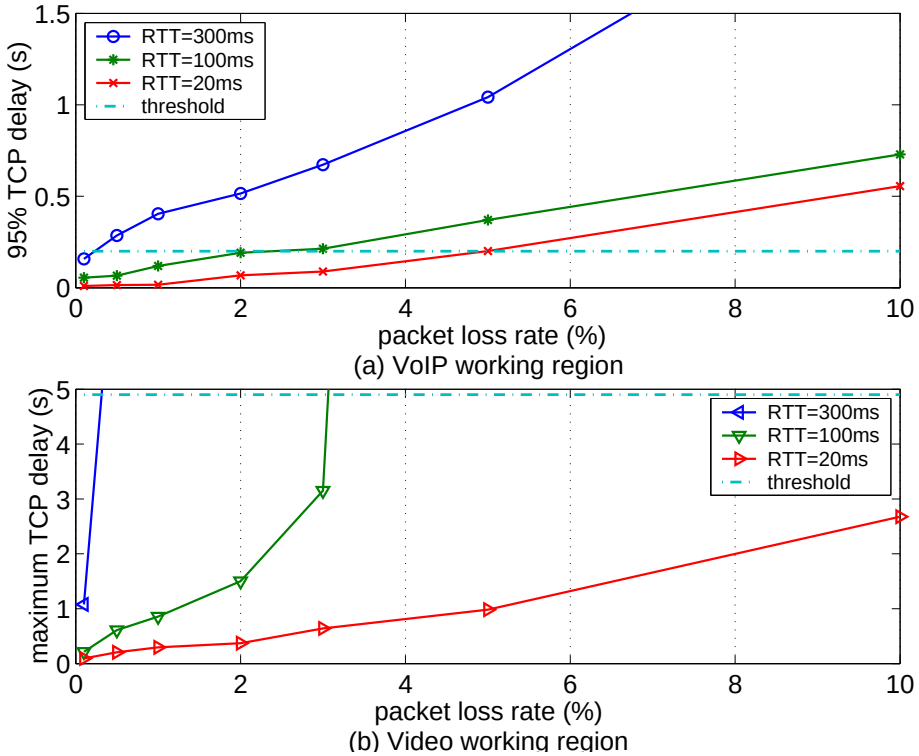



Figure 7.2: Working region for VoIP and video streaming as a function of RTT and packet loss rate.

Observe that when the RTT is 100 ms, the delay tolerance for VoIP is satisfied when the network loss rate is at most 2%. However, when the RTT is only 20 ms, the results indicate a tolerance of up to 5%. At the boundary of the working region, the delay added by TCP causes 5% of the packets to miss their playback deadline. Figure 7.2(b) plots the maximum delay for a live video streaming flow with a bit rate of 573 kb/s. When the RTT is 100 ms, the streaming threshold is satisfied when the loss rate is at most 3%. For RTT of 300 ms, it is satisfied at a network loss rate of 0.1%. The jump in the maximum delay at a network loss rate of 0.5% and RTT of 300 ms occurs because the 5 s startup delay is no longer sufficient to completely mask TCP delays. This knee of the curve typically occurs when the achievable TCP throughput is close to the bit rate of the video flow. The bit rates of 64 kb/s and 573 kb/s are the highest among the bit rates considered in Section 7.3 for VoIP and video flows and therefore, they give the most conservative estimate of the working region. However, the working region can be significantly constrained if the application does not use

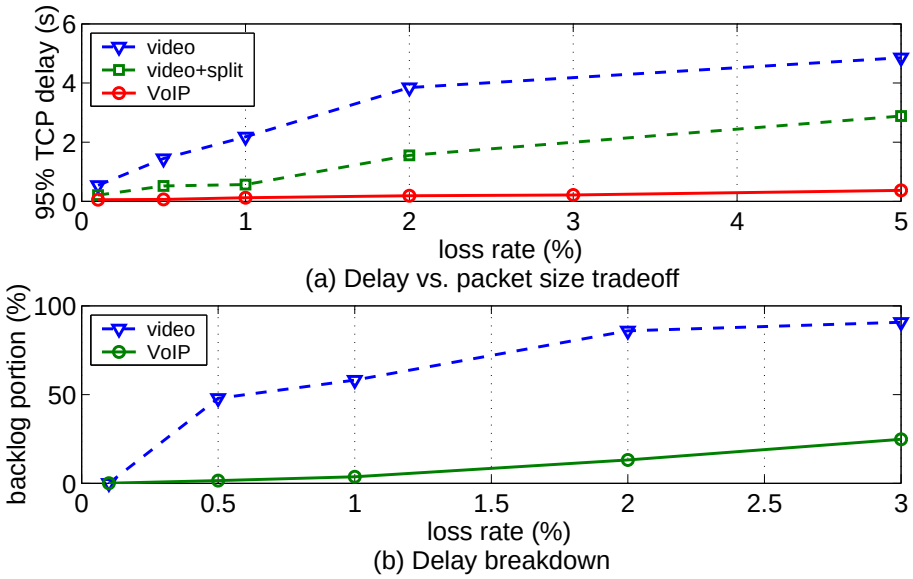



Figure 7.3: (a) The delay performance of two TCP flows having the same load in kb/s but different load in pps, and a VoIP flow. (b) Delay breakdown: the portion of TCP-level delays caused by the congestion control mechanism.

delay-friendly TCP settings.

### 7.4.2 The Effect of Packet Size on Performance

Our experiments indicate that under the same network conditions, VoIP flows perform significantly better than video flows. Figure 7.3(a) plots the 95% delay for VoIP and video flows with the same workload in packet per second (pps) but quite different workload in terms of bits per second (kb/s), i.e., the VoIP flow has a bit rate of 64 kb/s whereas the video flow has a bit rate of 573 kb/s. The figure clearly shows a performance bias towards the VoIP flow. This happens because a video flow has a higher bit rate than a VoIP flow. Hence, during network-limited periods, a TCP sender transmitting a video flow builds up a larger packet backlog and consequently, it requires more time to drain this backlog. For VoIP flows, the TCP sender groups several queued VoIP packets into one transmission packet as permitted by the MSS. This further increases the queue drain rate, thereby reducing the queuing delay at the TCP sender.

An interesting question to ask is that among two flows having the same workload in kb/s, does TCP have a performance bias towards a flow with larger workload in pps? To address

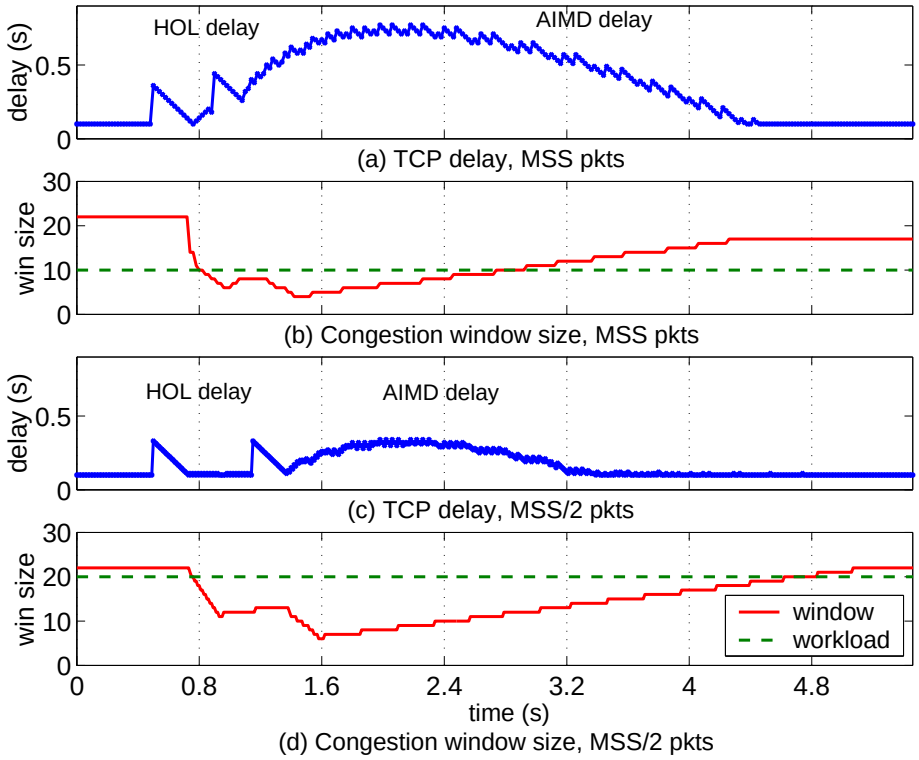



Figure 7.4: TCP delay and congestion window evolution for two flows with the same workload in kb/s but two different packet sizes, i.e., MSS (a-b) and half-MSS (c-d).

this question, we measured the delay performance of two flows having the same workload in kb/s but different workload in pps. The results are shown by the curves labeled video and video+split in Figure 7.3(a). Specifically, the packet rate of video+split flow is twice of the video flow but the application-level workload rate in bytes is the same. Surprisingly, there is a performance bias towards the flow with twice the packet rate of the other flow.

To illustrate the reason for this performance difference, we plot the TCP delay and congestion window size for two flows with the same application-level workload in kb/s in Figure 7.4. The flow in Figures 7.4(a) and (b) corresponds to an application that sends 100 MSS-sized packets per second. The flow in Figures 7.4(c) and (d) corresponds to an application that sends 200 half MSS-sized packets per second. Both flows operate over a symmetric network with 200 ms RTT and experience two close-by losses. Observe that the flow with half MSS-sized packets experiences lower delay than the other one. This happens

because the AIMD mechanism updates the congestion control state as a function of the number of *packets* sent, rather than as a function of the number of *bytes* sent (see Section 7.2.1). Since TCP adapts its congestion control state and hence its throughput based on the number of packets sent, the magnitude of the throughput fluctuations (in bytes) is smaller for the flow with smaller packet size and higher packet rate, resulting in lower delays.

We analyze the breakdown of TCP-level delays by computing the time packets are backlogged at the sender (i.e., the congestion control delay component) and the time it takes the TCP sender to get a packet to the receiving application (i.e., the retransmission and head-of-line delay components). Figure 7.3(b), shows the delay breakdown in terms of these two components for VoIP and video flows. As shown, the delays of a VoIP flow over TCP tend to be dominated by the loss recovery latency, whereas those of a video flow tend to be dominated by the delays caused by the congestion control mechanism. Similar results were obtained for CBR sources with other bit rates.

### 7.4.3 Sensitivity to Byte-counting

In order to provide a measured response to ACKs that cover only small amounts of data, Allman [Allman, 2003] proposes to increase the congestion window based on the number of bytes acknowledged by each incoming ACK rather than on the number of ACKs received. This mechanism is known as byte-counting. Byte-counting is configured on a per-system rather than per-connection basis in Linux, and is disabled by default. It is not implemented in Windows XP. A question arises how does the performance of VoIP flows change when TCP increases its congestion window by the number of bytes sent.

To answer this question, we measured the delay of five VoIP flows competing with five long-lived TCP flows and varying number of web flows in a drop-tail queue environment (see Section 7.3.2). Figure 7.5 shows 95th percentile and maximum delay for a VoIP flow using ACK and byte-counting. It highlights the gain of a VoIP flow when byte-counting is not used. On average, the use of byte-counting increases the TCP delay by 10-20%. The delay increases because TCP with byte-counting increases its sending rate in proportion to the number of bytes sent. Hence, a byte-counting TCP can be viewed as more fair

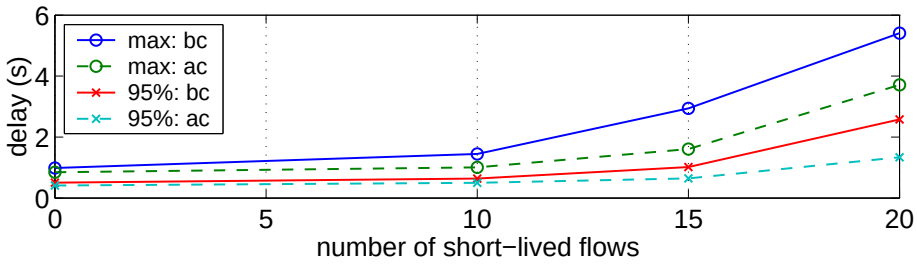



Figure 7.5: 95% and maximum delay for a VoIP flow using ACK and byte-counting.

than ACK-counting TCP with respect to the congestion control behavior. The support for byte-based congestion control mechanism must come from the underlying operating system. However, since Linux and Windows XP use ACK-counting by default, VoIP flows implicitly benefit from it.

### 7.4.4 Playout Buffer Size Setting

Real-time applications use a playout buffer to compensate for variable network delays. The receiving application usually delays the playout of received media packets for some time so that a large fraction of the packets is received before their scheduled playout times. A question of interest is how should an application factor in the TCP-level delays in computing the appropriate playout buffer size.

TCP is a reliable and in-order delivery protocol that deals with packet loss by introducing delay. TCP needs at least $RTT + 3/f$ to detect and recover a lost packet using a fast retransmit, where $1/f$ is the packetization interval. The time TCP needs to detect a lost packet using a retransmission timeout is at least the base timeout value. According to [Handley *et al.*, 2003], this value can be approximated by $4RTT$. Setting the playout delay below the sum of the one-way network delay $L$ and the minimum of these two thresholds will always cause the application to exhaust its playout buffer when a packet loss occurs.

In the considered environment shown in Figure 7.6, TCP can recover from a packet loss using a fast retransmit within $1.5RTT + 60\,\text{ms}$, because the packetization interval is 20 ms and the network is symmetric. As seen in the plot, setting the playout delay to the loss recovery latency of TCP yields up to 5% of late packets for RTTs of up to 100 ms and

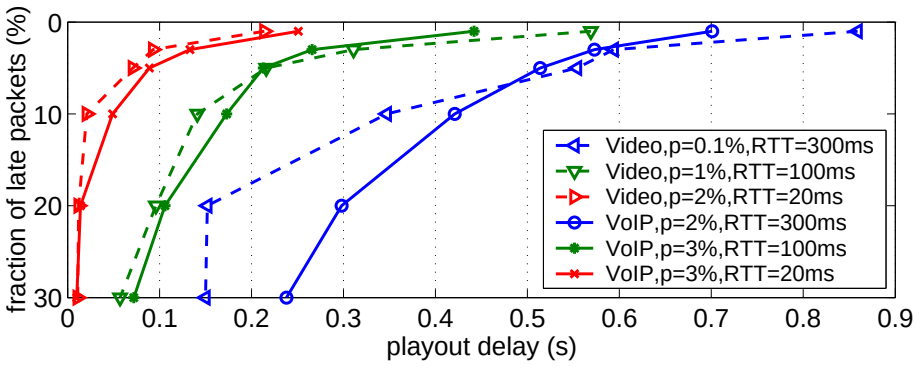



Figure 7.6: The required playout delay for VoIP and video flows and various RTTs and loss rates.

network loss rates of up to 2%. From further experimentation, we find that a buffer with a playout delay of $L + RTT + 3/f$ can mask out TCP delays for a large portion of the VoIP working region. This setting, however, does not mask out TCP delays for the video working region, because the delay of video flows is dominated by the packet backlog rather than by the loss recovery latency.

Figure 7.7 shows the delays masked out by the proposed playout setting for VoIP and video flows in a network with a 100 ms RTT and packet loss rates of 1% and 3%.

## 7.5 Delay Reduction Approaches

In this section we discuss application-level heuristics that can improve the performance of real-time media applications without additional help from the network. We analyze whether the delay reduction comes at the expense of other flows, in particular long-lived FTP flows. In the following, we first discuss a packet splitting scheme and then consider the use of parallel connections. We show that both schemes are effective for video flows but have only a marginal impact on VoIP flows.

### 7.5.1 Packet Splitting

As described in Section 7.4.2, the congestion control mechanism of TCP results in a performance bias in favor of flows with small packets. A question of interest is whether the

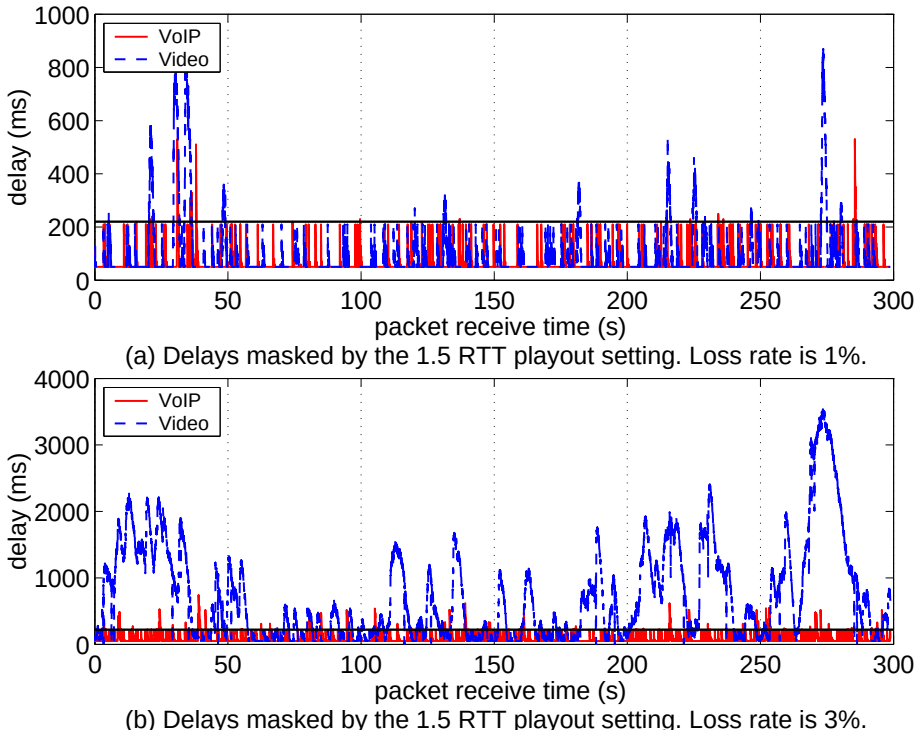



Figure 7.7: The TCP delays masked out by a $1.5RTT + 3/f$ playout delay for a network loss rate of (a) 1% and (b) 3%.

delay performance of real-time applications can be improved by masquerading TCP flows with large packets as flows with small packets. The application can split every large packet into a few smaller ones, while maintaining the same workload in bytes per second. We call this scheme *split-N*, where $N$ is the number of small packets generated. Packet splitting, however, may also backfire: if all CBR-TCP flows started using packet splitting, the network could quickly become congested due to the TCP header overhead. Hence, a wide-scale adoption of such an approach runs the risk of degrading the performance of all flows. Further, reducing the packet size can increase instances of packet reordering [Medina *et al.*, 2005].

To understand the performance of *split-N* within a wide-scale deployment, we measured the delay of a video flow (i.e., a 573 kb/s video source) in an environment with a drop-tail queue, as described in Section 7.3.2. As shown in Figure 7.8(a), the *split-2* scheme reduces TCP delay by up to 30% under low and moderate loss rates, whereas schemes with

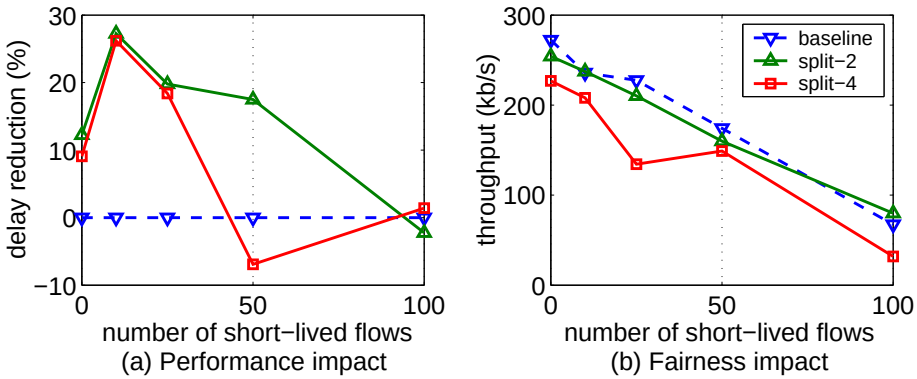



Figure 7.8: (a) The reduction in the 95th delay percentile of a video flow using *split-N* in a drop-tail queue environment (b) Throughput of background FTP flows in the same environment.

higher split factors yield diminishing gains or even perform worse than a no-split scheme. The performance degradation is partially due to the increase in the burstiness of the flow with packet splitting. This burstiness can be reduced to some extent by evenly spacing split-packets over the packetization interval. However, perfect pacing may be difficult to achieve at the application layer due to the small packetization intervals (e.g., 20 ms) used in practice.

During periods of high congestion (100 short-lived flows), a TCP sender using a *split-N* scheme is heavily backlogged and hence is unable to improve performance using the *split-N* scheme. We used the drop-tail queue environment to study the fairness implications of this scheme. In particular, we measured the throughput of long-lived TCP flows that share a congested link with video flows employing packet splitting. As shown in Figure 7.8(b), the *split-N* scheme impacts the throughput of the long-lived TCP flows. For example, the use of *split-4* reduces the throughput of a background TCP flow by 27% on average. From the plot, we observe that the throughput reduces quickly with the split factor.

### 7.5.2 Parallel Connections

A straightforward approach to improve the delay performance of a CBR-TCP flow is to stripe its load across parallel TCP connections. The idea is that several TCP streams are more aggressive than one TCP stream with respect to the congestion control behavior [Wang

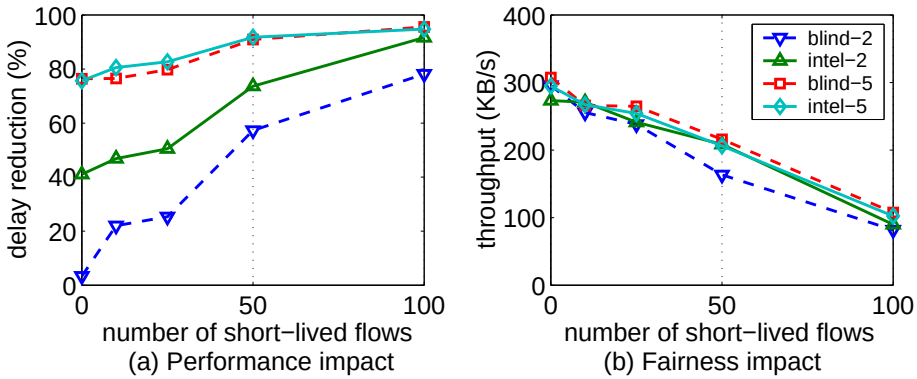



Figure 7.9: (a) The reduction in the 95th delay percentile for a 'blind' and an 'intelligent' scheme with N connections in a drop-tail queue environment (b) Throughput of background FTP flows in the same environment.

*et al.*, 2007], which can result in lower TCP delays. Previous exploration of parallel TCP connections for streaming and data-intensive applications has focused mainly on enhancing the throughput. However, we focus on reducing the delay. Specifically, we provide insights on the delay performance of parallel connection schemes for real-time applications.

Packet striping can be done in a delay-agnostic or delay-aware fashion. The simplest approach is to use a delay-agnostic ('blind') parallel connection scheme that sends packets over parallel TCP connections in a round-robin fashion. We propose a delay-aware ('intelligent') scheme which selects a connection for packet transmission that has the smallest TCP send queue and is not in the timeout state and show the results in Figure 7.9(a). The 'intel-2' scheme outperforms the 'blind-2' scheme because it dynamically avoids connections with large queues and in timeout states. Further, due to its dynamic nature, this scheme copes better with connections with small congestion windows.

Observe that when the number of connections was increased to five, the performance of 'intel-5' and 'blind-5' was comparable. This phenomena occurs because the increase in the total number of TCP connections causes the back log at any connection to be reduced. Similar to the 'blind' scheme, we observe that using more than five parallel connections results in diminishing gains. We note that the parallelization spectrum ranges from a single flow to having as many flows as the packet rate per RTT.

Similar to packet splitting, we study the fairness impact of these schemes on the back-

ground traffic using a drop-tail queue environment. We present the results in Figure 7.9(b). As shown, both 'intelligent' and 'blind' schemes have a negligible impact on the throughput of the background long-lived FTP flows. The impact is negligible because these schemes do not introduce additional traffic besides session setup and teardown. Though parallel TCP streams are more aggressive than a single TCP stream, their aggregated throughput is still limited by the rate of the CBR source.

## 7.6 Delay-Friendly Guidelines

We present delay-friendly guidelines for TCP-based VoIP and live video streaming applications. We categorize them into TCP and OS-level guidelines, and application-level guidelines. In the following, we first provide a comprehensive set of guidelines for setting TCP and OS parameters. While several settings such as disabling Nagle's algorithm and using large receive buffers are common practices in delay-sensitive applications, the impact of others, specifically, window validation, byte counting and limited transmit is less obvious. We then provide guidelines for setting application-level parameters.

### 7.6.1 TCP and OS-level guidelines

- Nagle's algorithm should be disabled as it introduces transmission delays at the TCP sender.
- CBR-TCP applications should set a large receive buffer and operate with non-blocking sockets so that the transmission of TCP is not limited by its flow control mechanism.
- To increase the loss efficiency of TCP, SACK should be enabled [Chen *et al.*, 2006] and limited transmit be used. The latter is primarily applicable for a TCP connection with small windows.
- Congestion window validation during application-limited periods [Handley *et al.*, 2000], and byte-counting should be disabled. These settings are disabled by default on Linux and Windows XP systems.

- The initial window size should be set to four segments as it can remove delays up to three RTTs and a timeout during the initial slow-start period [Allman *et al.*, 2002]. Some operating systems (e.g., Linux) inherit the *ssthresh* value from the last connection for the current connection. This may cause the current connection to enter congestion avoidance earlier even if it does not suffer from any loss, thereby incurring additional start-up delay during initial slow-start. Therefore, we suggest that *ssthresh* inheritance be disabled.

As a general rule, the above TCP-level configurations should be set on a per-connection basis if the underlying operating system supports it. Note that these guidelines do not require any change in the TCP stack.

### 7.6.2 Application-Level guidelines

- **Playout buffer setting.** As discussed in Section 7.4.4, a CBR-TCP application with small packets can statically set the playout delay to the loss recovery latency $L + RTT + 3/f$. This setting masks out TCP delay variations for a wide range of environments, in particular, for a large portion of the VoIP working region.

- **Video flows.** The delay performance of video flows can be improved by parallelizing the flow over multiple connections. If supported by the underlying operating system, an application should preferentially send packets over the connection that has the smallest send buffer size and is not in a timeout state (In Linux, an application can check the send buffer size by invoking `ioctl(,SIOCOUTQ,)` system call). Splitting MSS-sized packets by half reduces delay for video flows if a single connection is preferred.

- **Idle periods.** VoIP over TCP may have somewhat different properties than CBR over TCP. In particular, VoIP with silence suppression has idle periods, requiring TCP to ramp back up to the old sending rate after idle periods. We therefore suggest that an application should continue sending minimal traffic during idle periods (e.g., 3 packets per round-trip time, as suggested by [Mondal and Kuzmanovic, 2007]) to avoid having very small congestion windows. The size of the packets sent during silence periods can

be as low as one-byte if the TCP implementation uses ACK-counting. This is because an ACK-counting TCP increases the congestion window size based on the number of packets sent rather than the number of bytes sent.

- **MSS to packet size ratio.** We recommend that the ratio of MSS to packet size should be an integer. This setting insures that the packet assembly capability of TCP will give the lowest possible sending rate in packets per second when the TCP sender is backlogged. The lower sending rate results in lower delays compared to a flow with an equivalent load in kb/s and a non integer MSS to packet size ratio.

- **Proactive packet drop.** By examining the size of the TCP send buffer, an application can potentially infer the delays a new packet will experience. Thus, it may choose to drop packets at the sender, if the buffer size crosses a certain inferred delay threshold.

- **In-order delivery mechanism.** As described in Section 7.4.2, the delays of TCP flows with small packets tend to be dominated by the in-order delivery mechanism of TCP. That is, packets are held in the receive buffer while waiting for a lost packet(s) to arrive. A potential modification to the TCP operating system API can allow the application to peek into its receive buffer and extract out-of-order packets. A similar approach has been proposed by [McCreary *et al.*, 2005]. Although this modification requires changing the receive API to receive our-of-order packets, it does not change the network semantics of TCP.

## 7.7 Related Work

Goel *et al.* [Goel *et al.*, 2002] present an empirical study of kernel-level TCP enhancements to reduce the delays induced by congestion control for streaming flows. The performance of TCP for real-time flows has also been considered by [Mukherjee and Brecht, 2000; McCreary *et al.*, 2005]. However, unlike our study, these papers propose a modification to the TCP stack, do not provide insights on the delay working region for VoIP flows, and provide no guidelines for the playout buffer setting.

## 7.8 Conclusion

We presented a study of delay incurred by the real-time traffic such as voice and live video streaming when carried over TCP. We characterized the working region of TCP and studied the impact of packet size on the delay performance of TCP. We presented guidelines for setting the playout buffer of real-time traffic when carried over TCP and techniques for improving the delay performance of this traffic. These techniques can be useful to the designers of real-time applications.

# Chapter 8

# Energy Efficiency of VoIP Systems

## 8.1 Introduction

We aim to understand and analyze the energy efficiency of Voice-over-IP (VoIP) systems. The core function of a VoIP system is to provide mechanisms for storing and locating the network addresses of user agents and for establishing voice and video media sessions, often in the presence of restrictive network address translators (NATs) and firewalls. These systems also provide additional functionality such as voicemail, contact lists (address books), conferencing, and calling circuit-switched (PSTN) and mobile phones. From the perspective of energy efficiency, a VoIP system can broadly be classified according to two criteria: whether it is a primary-line phone service replacing PSTN and whether it uses a client-server (c/s) or a peer-to-peer (p2p) architecture. Vonage [Vonage, 2010] and Google Talk [Google, 2010] are examples of c/s architectures, while Skype [Skype, 2010a] is an example of a p2p architecture. Of these, only Vonage is a primary-line phone service replacing PSTN.

We begin the chapter by describing the common configurations of deployed c/s and p2p VoIP systems (Section 8.2). We then devise a simple model for analyzing the energy efficiency of these common configurations (Section 8.3). This model enables a systematic comparison of c/s and p2p configurations of VoIP systems. We then present measurements for the c/s and p2p VoIP components of these systems (Section 8.4) which we apply to the model developed for identifying the sources of energy wastage in these systems and the incurred economic costs (Section 8.5). Based on our analysis, we provide recommendations

to improve the energy efficiency of VoIP systems (Section 8.6). Finally, we present the related work (Section 8.7).

## 8.2 VoIP System Architecture

We briefly explain the main functionalities of VoIP systems and describe how they are typically implemented in c/s and p2p VoIP systems. We then describe in more detail the architecture of a typical Internet telephony service provider (ITSP), an enterprise VoIP system, a softphone based c/s VoIP system, and Skype. The first three are representative of a client-server VoIP architecture, and the latter is representative of a p2p VoIP architecture.

### 8.2.1 Functionalities of a VoIP System

The main functionalities of a VoIP system are:

**Signaling** - storing and locating the reachable address of the user agents, and routing calls between user agents.

**NAT keep-alive** - sending and processing user agent traffic to maintain state at the NAT devices for receiving incoming requests and calls.

**Media relaying** - sending VoIP traffic directly between two user agents or through a relay. Relaying is necessary when one or both of the user agents are behind a restrictive NAT or a firewall which prevents establishment of a direct VoIP connection.

**Authentication, authorization, accounting** - verifying that a user agent is permitted to use the system and tracking usage for billing purposes.

**PSTN and mobile connectivity** - establishing calls between VoIP clients, and PSTN and mobile phones using managed gateways.

**Other services** - such as voicemail, contact list storage, video calls, and multiparty audio and video conferencing.

Of the services listed above, signaling, NAT keep-alive, and media relaying lend themselves most easily to a p2p implementation. Consequently in the VoIP systems (including

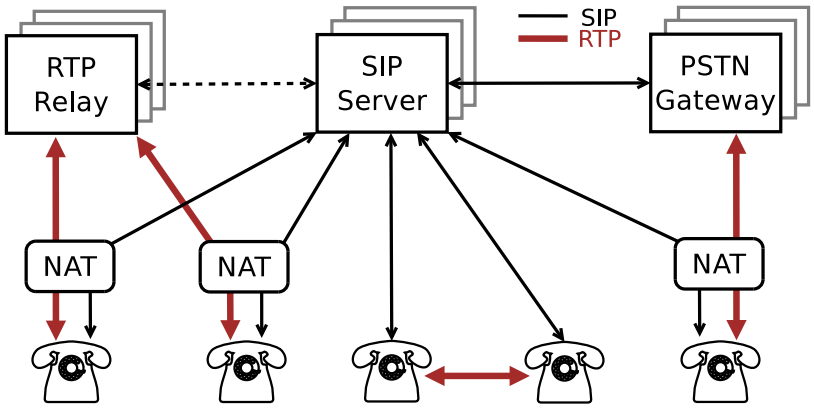



Figure 8.1: Client-server Internet telephony service provider (ITSP) architecture.

Skype) of which we are aware, all but signaling, NAT keep-alive, and media relaying functionality are implemented on centralized servers. As we will see in Section 8.5, the relative energy consumption of c/s and p2p VoIP systems will be determined by the relative efficiency of c/s and p2p implementations of signaling, NAT keep-alive, and media relaying.

### 8.2.2 Client-server VoIP Architecture

We consider three types of client-server VoIP systems. The first type is an Internet telephony service provider (ITSP) that provides telephony service to residential and business customers as their primary voice service. The second type is representative of VoIP system deployment in an enterprise. The third type represents softphone-based VoIP systems like Google Talk.

#### 8.2.2.1 Typical ITSP (T-ITSP)

We surveyed three c/s ITSPs in February 2010 to obtain information about their server systems, subscriber populations and characteristics of the network traffic. Based on this survey, we present an overview of the largest of these whose architecture is typical for an ITSP. We refer to this ITSP as *T-ITSP* in order to preserve its anonymity.

T-ITSP uses an infrastructure based on open protocols, namely SIP [Rosenberg *et al.*, 2002] for signaling and RTP [Schulzrinne *et al.*, 2003] for media. It uses a SIP proxy and registrar implementation based on SIP Express Router (SER) [SIP Router Project, 2010]. The SIP registrar stores the reachable address of user agents (phones), whereas the proxy

server forwards signaling requests between user agents. Users place calls predominantly through hardware SIP phones. Most such phones are audio-capable only, although some also support video. The vast majority of hardphones are connected to the broadband Internet through a broadband modem which in turn is connected to a home or an office router. The router is typically configured to act as a NAT/firewall. Over 90% of SIP signaling is carried over UDP. User agents connect to SIP servers, perform SIP digest authentication, and register their reachable address every 50 minutes to receive incoming calls, a process we refer to as a *registration* event.

Because most existing NAT devices maintain UDP bindings for a short period of time [Ford *et al.*, 2005], hardphones behind NATs need to periodically refresh the binding in order to reliably receive incoming calls. The hardphones achieve this by sending a SIP NOTIFY request [Roach, 2002] every 15 s to the SIP server, which replies with a 200 OK response. While wasteful, this method proved to be the only reliable way of maintaining NAT bindings.

To establish a call, the user agents send the SIP INVITE requests to the SIP proxy servers, which then forward these requests to the destination user agents. The vast majority of hardphones are behind NATs/firewalls and a large proportion of these devices use default settings that prevent them directly exchanging voice packets. Consequently, T-ITSP needs to operate RTP relay servers to relay these calls, thereby consuming additional energy and network bandwidth. T-ITSP also maintains a number of PSTN servers for calling phones in the traditional telephone network. T-ITSP does not encrypt signaling or media traffic. Figure 8.1 illustrates the architecture of T-ITSP.

**Traffic** T-ITSP has a total subscriber base of approximately 100,000 users. The peak call arrival rate is 15 calls per second (CPS) and the systems see no more than 8,000 calls at any instant. Approximately 60% (or 4,800) of the peak calls are to subscribers within the ITSP; the rest are being routed to PSTN/mobile phones. Hardphones register their network address with T-ITSP's SIP registrar every 50 minutes and send a SIP NOTIFY message every 15 s to maintain the NAT binding. For 100 k subscribers, these statistics imply that the SIP registrar needs to process 33 registration events and 6,667 NOTIFY

events per second. In Section 8.4, we extrapolate these peak numbers for a large subscriber base.

#### 8.2.2.2 Enterprise VoIP Systems

The enterprise VoIP system comprises of SIP proxy and registrar servers, hardphones, and enterprise ethernet switches for connecting hardphones to the proxy server. In addition to the VoIP phones, office computers are also connected to the same ethernet switch. In some installations, the enterprise switches also provide power to the hardphones through Power-over-Ethernet (PoE) [Power over Ethernet (PoE), 2010]. The enterprise VoIP system is connected to the other VoIP, PSTN, or mobile telephony systems through gateways. Typically, the IP address space in an enterprise is flat and the NAT devices are sporadic. Consequently, unlike T-ITSP, the hardphones do not need to periodically send SIP NOTIFY messages to keep the NAT bindings. Further, the enterprise VoIP system does not need to maintain media relay servers. When the IP address space is not flat, the VoIP systems in different departments are typically connected via gateways or call managers [Cisco, 2010b].

#### 8.2.2.3 Softphone-based VoIP Systems

The softphone-based client-server VoIP systems such as Google Talk are similar in their functionality to T-ITSP, except that the user agents mostly run as a software application on a desktop or a mobile device. Such systems typically do not replace PSTN as the primary phone service.

### 8.2.3 P2P VoIP Architecture – Skype

We present an overview of Skype [Skype, 2010a] which is representative of a p2p VoIP system. Skype is not advertised as a primary-line phone service. There are two types of nodes in a Skype network, super nodes and ordinary nodes. The super nodes form the Skype overlay network, with ordinary nodes connecting to one or more super nodes. Super nodes, which are chosen for their unrestricted connectivity and high-bandwidth, are responsible for signaling, NAT keep-alive, and media relaying. Skype encrypts signaling and media traffic to prevent super nodes from eavesdropping. Skype managed-servers provide functionality

| Feature | T-ITSP | Enterprise | Google Talk | Skype |
|---|---|---|---|---|
| User agents (UA) | Hardphone | Hardphone / Softphone | Softphone | Softphone |
| UAs always on | Yes | Yes | No | No |
| Signaling | Centralized | Centralized | Centralized | P2P+ Centralized |
| NAT keep-alive | Centralized | None | Centralized | P2P |
| Media relaying | Centralized | None | Centralized | P2P |
| PSTN connectivity | Centralized | Centralized | Centralized | Centralized |
| Voicemail | Centralized | Centralized | Centralized | Centralized |
| Contact list | Centralized | Centralized | Centralized | Centralized |

Table 8.1: Comparison of T-ITSP, enterprise VoIP, Google Talk, and Skype features. The value of 'None' in the Enterprise column indicates that the user agents typically do not send NAT keep-alives, nor do they require media relays for establishing calls with user agents within the same enterprise.

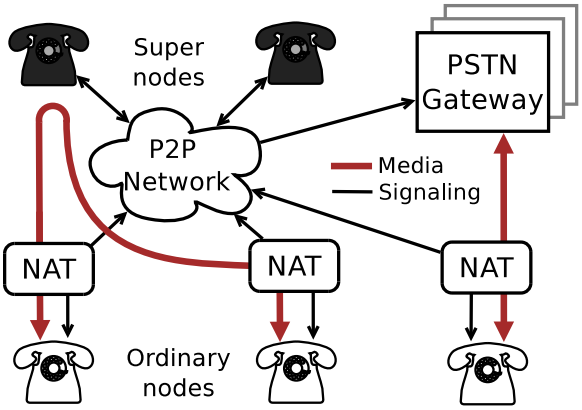



Figure 8.2: P2P VoIP architecture.

for authentication, contact list and voicemail storage, and calling PSTN and mobile phones. Figure 8.2 shows an illustration of a p2p VoIP system. Table 8.1 compares the distributed and centralized features of the T-ITSP, enterprise VoIP systems, Google Talk, and Skype.

## 8.3 Power Consumption Model

We present a model for understanding the power consumption of c/s and p2p VoIP system architectures. We focus on signaling, NAT traversal, and media relaying as they are accomplished using managed servers in the c/s but through super nodes in p2p VoIP systems.

Let $N$ be the total number of online subscribers of a VoIP system and let $\lambda_{INV}$ be the peak rate of calls per second these subscribers make and $d$ be the average call duration. These calls are either to other subscribers of the VoIP provider or to PSTN or mobile phones. Let $p_v$ be the percentage of VoIP calls. Of these, let $p_{relay}$ be the proportion of calls that need a relay.

### 8.3.1 Client-Server

As discussed in Section 8.2.2.1, a c/s VoIP architecture has dedicated servers for handling the signaling, NAT traversal, and media relaying traffic. Signaling traffic includes registration of user agent network addresses with the SIP registrar and call signaling for establishing media sessions. Let $\lambda_{REG}$ and $\lambda_{INV}$ denote the peak number of SIP registration events and calls per seconds, respectively, that $N$ user agents generate. The NAT traversal traffic (SIP NOTIFY in T-ITSP) is sent by the user agents to refresh NAT bindings and ensuring reliable receipt of incoming calls. Let $\lambda_{NAT}$ be the rate of these NAT traversal messages per second. $\lambda_{NAT}$ will be significantly lower for signaling over TCP than over UDP. In most c/s VoIP systems, signaling and NAT traversal are handled on separate servers from those of media-relaying.

Let $S(\lambda_{REG}, \lambda_{INV}, \lambda_{NAT}, PROTO)$ represent the number of signaling servers needed to handle the peak signaling and NAT traversal load under a particular transport protocol $PROTO$. The $PROTO$ may be UDP, TCP, or TLS. An advantage of using permanent TCP connections between user agents and SIP servers is that it reduces the frequency of the traffic to maintain NAT bindings. However, maintaining hundreds of thousands of TCP or TLS connections on a server is costly in terms of the memory needed [Shen *et al.*, 2010]. Let $M(\lambda_{INV}, d, p_v, p_{relay})$ represent the number of media relay servers needed to relay calls. Let $w_s$ and $w_m$ denote respectively the wattage consumed by signaling and media servers at the peak load. Let $c$ be the system's PUE and $r_s$ and $r_m$ be the redundancy factor used for signaling and media servers. Then the power consumed by the signaling and media-relay servers is given as follows:

$$w_{c/s} = (Sw_s r_s + Mw_m r_m)c \tag{8.1}$$

### 8.3.2 Peer-to-Peer

Recall from Section 8.2.3 that there are two types of nodes in a p2p communication system, namely, super nodes that forward signaling and routing traffic from other nodes and relay a call between nodes with restrictive network capacity, and secondly, ordinary nodes that do not participate in the overlay routing and connect to one or more super nodes. Let $N_S$ be the number of super nodes in the p2p system with a total population of $N$ subscribers. In contrast to c/s systems, where it is easy to attribute the energy consumption of signaling, NAT traversal and relaying, it is non-trivial to do so for super nodes in p2p systems. We consider two reasonable accounting strategies which apply as well to energy accounting on phones and network devices:

- **delta** - count only the additional power drawn by the signaling and relaying functions of the super node machine above that of the baseline power consumption of the machine.
- **prop** - in addition to delta, attribute to p2p VoIP a fraction of the system baseline power consumption that is proportional to the time the CPU is woken up to handle signaling, NAT traversal and media relaying traffic.

For simplicity, assume that each super node sends and receives $\lambda_{MAINT}$ messages to maintain the overlay, and receives $\frac{1}{N_S}$ of the total registration, call invites, and NAT traversal. Each super node relays at maximum one call at a time. A node may use a secure transport protocol such as TLS or DTLS for non-media relaying traffic. Let $w_{base}$ denote the baseline wattage drawn by the super node machine. Let $w_\Delta$ denote the wattage drawn by the overlay maintenance, registration, signaling, NAT traversal, and media relaying functionality. Let $p$ be the proportion of time the CPU is woken up to serve super node requests if *prop* accounting policy is chosen or zero for the *delta* policy. Then the power consumed by p2p super nodes is

$$w_{p2p} = (w_\Delta + w_{base}p)N_S \tag{8.2}$$

### 8.3.3 Comparison Issues in C/S and P2P VoIP Systems

In this section, we highlight the broader issues in comparing c/s and p2p VoIP systems.

#### 8.3.3.1 PSTN Replacement

The most important consideration for our comparison is whether the VoIP system replaces the always-on PSTN system (e.g., Vonage). For such a system, the user agents must always be reachable (or powered on) to receive incoming calls. The total energy consumed by such systems is the sum total of the energy consumed by always-on user agents and servers, if any.

In contrast, systems like Google Talk and Skype run as a software application on a desktop, laptop, or a mobile device and do not replace PSTN as the primary-line voice service. Therefore, it is not possible to directly compare them with systems like Vonage. Moreover, Google Talk uses a c/s architecture whereas Skype uses a p2p architecture. When comparing these two architectures, it is important that we examine the power consumed by the machines providing the core functionality (servers in c/s, super nodes in p2p) and not the difference in energy consumed by the user agents.

#### 8.3.3.2 Network Costs

C/S and p2p communication systems have a different network footprint as in the latter, nodes have to exchange data to maintain the p2p network. Edge and core routers likely incur an energy cost for forwarding traffic for p2p and c/s communication systems. However, these costs are harder to quantify as the edge and core routers are always on. Although an analysis similar to [Nedevschi *et al.*, 2008] can be used, we focus on quantifying the energy usage of the system itself and not the network. However, we do incorporate the energy costs of broadband modems and network switches to which VoIP user agents are directly connected and that otherwise cannot be powered down without disconnecting the user agent.

## 8.4 Measurements and Results

In this section, we describe a set of experiments for measuring the power consumption of signaling and media relay servers, broadband modems and home routers, enterprise ethernet switches, user agents (hardphones and softphones), and Skype super nodes. Our

power measurements were taken using a Watts-up .NET power meter [Watts up?, 2010]. The meter provides 0.1 W precision and claims an accuracy of 1.5% of the measured value.

### 8.4.1 Signaling and Media Relay Servers

Based on the architecture and load information of T-ITSP, we set up a test bed consisting of two servers, the first for handling signaling and NAT traversal workload, and the other for handling media relaying. Our goal was to measure the power consumption of these servers under peak load, and extrapolate the number of servers needed and the power consumed based on peak workload, using the model developed in Section 8.3.1. Although, this extrapolation may be considered an over simplification, it still provides useful insights into the energy consumed by large scale c/s VoIP systems.

#### 8.4.1.1 Testbed Overview

In our test bed, the SIP server machine was a Dell PowerEdge 1900 server [Dell, 2010] with two quad-core 2.33 GHz Intel Xeon X5345 processors and 4 GB of memory. It was connected to load-generators with two Intel 82545GM Gigabit Ethernet controllers. The machine had six fans. It ran Debian Squeeze (snapshot from 26th February 2010) with Linux kernel 2.6.32. We installed the latest version of SIP-Router, an open source SIP server [SIP Router Project, 2010] on the machine and configured it with all the features an ITSP operating in the public Internet would need to use. The SIP server was configured to use 2.5 GB of memory and 16 processes (2 per core). We used MySQL 5.1.41-3 (from a Debian package) configured with 2 GB of query cache. We used SIPp [SIPp, 2010] version 3.1.r590-1 to generate SIP traffic according to the model described in Section 8.2.2.1.

For RTP relay tests, we used an IBM HS22 blade server [IBM, 2010] with 5 blades installed. One of the blades was used as an RTP relay server; the remaining four blades and another two desktop-class PCs were used as RTP load generators. Each blade had two Intel Xeon quad-core CPUs running at 2.9 GHz and a 10 GigE Intel NIC with multiple hardware transmission and receive queues and ran a Linux 2.6.31 kernel. We used the latest version of iptrtpproxy [iptrtpproxy, 2010], a kernel-level RTP relay. The software relays RTP packets using iptables rules. We used a modified version of SEMS [iptel.org, 2010] to generate a

large number of simultaneous RTP sessions.

#### 8.4.1.2 SIP Server Measurements

We performed a number of measurements to figure out the maximum number of subscribers that our SIP server can support. We wanted to determine the maximum load on this server in three configurations: (1) signaling and NAT keep-alive (SIP NOTIFY) traffic carried over UDP as described in Section 8.2.2.1; (2) signaling traffic over UDP but without any SIP NOTIFY traffic; (3) signaling traffic over permanent TLS connections. The first configuration allowed us to reason about the maximum ITSP-like workload a server can handle. The second configuration provided insights into peak ITSP-like signaling workload a server can handle, assuming there were no NATs. The third configuration was helpful from the perspective of comparing T-ITSP to Skype, as Skype uses a TLS-like protocol to encrypt signaling and media traffic.

Before running any tests, we provisioned the database of the SIP server with one million unique subscribers. The baseline consumption of the server was 160 W. The machine had 6 fans; each fan consumed 10 W when running at full speed. The power consumption when all fans were removed and the machine was idle was 145 W. To see how CPUs contributed to the overall power consumption of the machine, we ran 8 cpuburn [Redelmeier, 2010] processes (one per core). The machine consumed 332 W when all cores were fully utilized.

For the first configuration, we found out that our server could handle T-ITSP's traffic mix for approximately half a million users. Under this load, the number of calls ($\lambda_{INV}$), registrations ($\lambda_{REG}$), and NAT keep-alives ($\lambda_{NAT}$) events per second were 75 k, 166 k, 33 k, respectively, and the server consumes ($w_s$) 210 W. For the second configuration, in which there was no NAT traversal traffic, we found that our server could handle load for approximately one million subscribers. $w_s$ was 190 W.

For the third configuration (signaling over TLS) there was no need to exchange frequent keep-alive messages over TCP connections to keep NAT bindings open, so $\lambda_{NAT}$ was 0. With SIP over TLS, the SIP server used 61 kB of memory per connection and one connection was needed per user agent. Consequently, memory became our bottleneck and a maximum of 43 k simultaneously connected user agents could be supported on a single SIP server. $w_s$

| Transport | NAT keep-alive | 100 k | 1 M | 10 M | 100 M |
|---|---|---|---|---|---|
| UDP | YES<br>NOTIFY/s | 1 | 2 | 20 | 200 |
| UDP | NO | 1 | 1 | 10 | 100 |
| TLS | NO | 3 | 25 | 250 | 2500 |

Table 8.2: Signaling servers needed by configuration.

| % relayed calls | 100 k | 1 M | 10 M | 100 M |
|---|---|---|---|---|
| 0% | 0 | 0 | 0 | 0 |
| 30% | 1 | 2 | 10 | 96 |
| 100% | 1 | 4 | 32 | 320 |

Table 8.3: Media servers needed when relayed calls are 0%, 30%, and 100% of ITSP-ITSP calls.

was 209 W.

Based on these measurements, we extrapolate the number of servers needed for these configurations in Table 8.2. Compared to the first configuration, observe that eliminating the keep-alive traffic reduces the number of servers by half in the second configuration. Although the number of signaling servers needed for the third configuration increases approximately by a factor of 12 as compared to the first configuration, we believe that such limitation can be addressed by (1) tuning the SSL buffer, (2) increasing memory in our server, and (3) using hardware SSL accelerators.

#### 8.4.1.3 Media Relay Server

We managed to saturate the IBM blade with 15,000 simultaneous calls. Each call had a bit rate of 64 kbit/s for an aggregate bit rate of 960 Mbit/s. At this rate, the resource bottleneck appeared to be a single CPU core overloaded by the ksoftirqd kernel thread. It is likely that even greater call volumes could be relayed by optimizing the multi-core scheduling of this machine using techniques such as [Dobrescu *et al.*, 2009]. At this workload, the media relay server consumed approximately 240 W ($w_m$). In Table 8.3, we extrapolate the number of relay servers needed as a function of user population and the number of calls that need relaying.

### 8.4.2 Broadband Modems, Middleboxes and Ethernet Switches

A typical residential broadband user is connected to the Internet through a home router (ethernet switch + WiFi) which in turn is connected to the broadband modem (cable, DSL, or fiber). Our measurements indicate that the recent models of WiFi routers with four ethernet switches consume, on average, 4-6 W of power. Similarly, a broadband modem also consumes 4-6 W of power. In our calculations, we use 5 W as an estimate for broadband modem and home router power consumption.

In an enterprise, the VoIP hardphones are connected to an ethernet switch which is typically PoE enabled. A 48 port Cisco switch model C2960S-48LPD-L consumes 70 W of power at five percent throughput [Cisco, 2010a] and has 370 W of available PoE power or 7.70 W per port.

### 8.4.3 User Agents

We performed measurements to determine the power consumption for a variety of user agents that included hardware SIP phones and softphones. We also performed power measurements for Skype super nodes.

#### 8.4.3.1 Hardware SIP Phones

For a variety of SIP-based hardphones, we found that phones consume between 3 W to 6 W of power. We also observed that the phone power consumption does not change when the user is in a voice call.

#### 8.4.3.2 Softphones

We used Skype and Google Talk as representative of softphones. For several desktop machines running Windows XP and Windows 7, we did not observe any discernible change in the machine baseline power consumption when Skype and Google Talk were idle. The non-discernible change in the power draw when these softphones are idle is partially attributed to the power meter we used which can only measure power up to tenth of a watt with an accuracy of 1.5%. When placing a voice call, we found that on average Skype and Google Talk consumes between 6 W to 8 W on a Windows XP and Windows 7 desktop machine.

Similarly, for a video call, Skype and Google Talk consumed between 10 W to 20 W. For laptop machines running Windows XP and Max OS X, we found that Skype and Google Talk, on average, consumed between 1-2 W when placing a voice call. As with the desktop machines, Skype and Google Talk did not cause any discernible power increase when idling. We observed similar power draw behavior for other SIP-based software clients.

#### 8.4.3.3 Skype's Energy Consumption as a Super Node

Measuring Skype's energy consumption as a super node is not straightforward. First, we need a machine to transition to super node status. Since the Skype client itself decides whether to become a super node, we can only encourage this decision to be made by ensuring that the node has a public IP address, has sufficient bandwidth, and is lightly loaded which we desired anyway given that we were trying to isolate what we assumed Skype's relatively low power consumption amidst the noise of the machine's hardware and operating system. To this end, we ran a Skype client for a few hours on a machine with a public IP address and good network connectivity. To determine if the Skype is relaying a call, we performed measurements using a traffic sniffer running on another machine which is connected to the same hub as the Skype machine. We assumed a call is being relayed if the bit rate was above a threshold [Suh *et al.*, 2006]. Although, our meter readings indicated that there was a non-zero power increase, the difference measured was smaller than the measurement error reported by the power meter. Determining when a super node is handling signaling traffic is even harder to detect, and the power draw per event lasts for a shorter interval and is likely smaller in magnitude. We did find that the machine can go to sleep when Skype is acting as a super node and relaying the call. The calls were either dropped or transferred to another relay; however, it is impossible for us to ascertain the status of those calls due to the closed nature of the Skype network.

## 8.5 Discussion

Our model and measurements allow us to answer the following questions, i.e., (1) what is the total energy consumed by a VoIP system that may or may not replace PSTN as the

| Users | 10 k | 100 k | 1 M | 10 M | 100 M |
|---|---|---|---|---|---|
| Servers (NATs) | 0.90 | 0.90 | 1.78 | 13.16 | 129.68 |
| Servers (no NATs) | 0.42 | 0.42 | 0.84 | 4.20 | 40.20 |
| Broadband modems | 50 | 500 | 5000 | 50,000 | 500,000 |
| Home routers | 50 | 500 | 5000 | 50,000 | 500,000 |
| Hardphones | 50 | 500 | 5000 | 50,000 | 500,000 |

Table 8.4: T-ITSP energy consumption as a function of number of users. All numbers are in kilowatts. The wattage for servers includes the PUE factor 'c' of two.

primary line phone service, (2) where is energy consumed in such a system, (3) are p2p VoIP systems more energy efficient than c/s?

To answer questions (1) and (2), we consider T-ITSP (Section 8.2.2.1), enterprise (Section 8.2.2.2), and softphone-based c/s VoIP deployments (Section 8.2.2.3). Recall that for the T-ITSP workload that include signaling and NAT keep-alive traffic over UDP, our SIP server can handle this workload for 500 thousand subscribers, and consumes 209 W ($w_S$) under peak load. The RTP relay server under test consumed 240 W ($w_M$) and can relay 15 thousand calls, with each call having a bit rate of 64 kb/s. The number of active calls in the system for 500 k users are 24 k (by extrapolating the number of active calls for 100 k T-ITSP users), requiring two relay servers to handle this load (one server can handle 15 k calls). Depending on the actual deployment, not all calls need relaying. Our conversations with various VoIP system providers suggest that using NAT traversal techniques like ICE [Rosenberg, 2010] will likely bring down the relayed sessions under 30%. When relaying 30% of the 24 k calls, only one relay server is needed. We compute $w_{c/s}$ for both 100% and 30% relaying using our c/s model (Equation (8.1)). We plug $c$ (PUE) as 2, and $r_S = 1$ and $r_M = 1$ in our model. For 100% and 30% relaying, the computed $w_S$ is 1.378 kW and 0.89 kW, respectively. Observe that these numbers are approximate for the peak load and will be higher if the servers are under utilized.

Table 8.4 shows the energy consumed in kilowatts for running the servers, broadband modems, home routers, and hardphones. Based on our measurements, we assign 5 W for running the broadband modem and 5 W for the WiFi router with four ethernet ports. These numbers will be higher for a WiFi router with more than four ports. Nevertheless, the energy consumed by these devices cannot be solely attributed to VoIP because the

both VoIP and non-VoIP traffic share the same router. A reasonable assumption is that on average, such sharing occurs only for 12 hours in a day. The rest of the time, these devices must remain powered on so that a VoIP user can receive incoming calls. Using this conservative assumption, we calculate the approximate power required to run a 100 million VoIP system to be 1000.129 MW. The number is calculated by using plugging 500 MW for phones, 500 MW for broadband modems and home routers (discounted by 50% because of our usage assumption) and 129.68 kW for running servers. The monthly cost of running such a system, at 11 cents per kWh [U.S. Energy Information Administration, Independent Statistics and Analysis, 2010] is 79.2 million dollars or 80 cents per user per month (rounded up). The energy cost per month of running the servers is $10,270 or less than one thousandth of a cent per user per month.

In enterprise VoIP systems, there are typically minimal or no NATs. Consequently, the hardphones do not need to send SIP NOTIFY packets to the SIP proxy server for keeping the NAT bindings alive nor do they will likely require any media relay servers. However, VoIP hardphones must be connected to the ethernet switches. A 48-port PoE enabled ethernet switch when connected with hardphones that require 5 W per phone consumes 310 W. For an organization with 100,000 hardphones, the total number of such switches needed are at least 2,084. If only one half of the ports in each switch are used for VoIP phones and the rest for non-VoIP usages such as Internet, then the number of switches increases to 4,168. Assuming that switches solely serve VoIP traffic for one half of the day (ignoring idle time on weekends and holidays), the monthly power consumption and economic cost of an enterprise system with 100,000 users is approximately 465,033 kWh and $51,153, respectively. The latter number when rounded up is 52 cents per user per month.

These results indicate always on VoIP phones are a major source of energy waste in T-ITSP and enterprise VoIP systems. Further, the always on broadband modems, home routers, and enterprise switches significantly add to the energy bill. In contrast, the servers only consume a tiny fraction (<0.02%) of the total power consumed by a VoIP system replacing PSTN. Table 8.4 also illustrates that restrictive NATs and firewalls are wasteful in terms of server power consumption as they increase the total energy consumption of servers by a factor of two and three for number of users below and above one million,

respectively.

For softphone-based c/s systems such as Google Talk that do not replace PSTN as the primary line phone service, their servers incur the same server energy usage for an equivalent load as for the servers in VoIP systems that replace PSTN. However, the softphone energy consumption is harder to quantify in these systems. This is because the softphones typically run on PC's which are powered on any way. If the softphones consume a small fraction of the power consumed by the PC, it is likely that they will still dominate the total power consumption of such a system; however, the relative power fraction of servers will increase. On the other hand, if the users leave their PC's powered on solely for the purpose of receiving calls (such as magicJack [magicJack, 2010]), then the power consumption of running these softphones will be much higher than hardphones, making such systems highly energy inefficient. As such, a user study is needed to determine how long the users keep their PC's idle but powered on for receiving incoming calls.

To answer the third question whether p2p system is more energy efficient than c/s or vice versa, we note this will only hold if the power consumed by all the super nodes assuming a delta accounting policy is less than the total power consumed by the servers in c/s systems, i.e.,

$$w_{\Delta} N_S < w_{c/s} \tag{8.3}$$

Observe that this equation does not include the power consumed by user agents, broadband modems, home routers, or ethernet switches because we assume that they consume the same amount of power in c/s and p2p VoIP systems. To solve (3) for $w_{\Delta}$, we need to estimate the total number of super nodes in the system that can process signaling, NAT keep-alive and media relaying traffic. We estimate the number of super nodes to be 1% of the total user population, meaning that in a population of 500 k user agents, 5 k are super nodes. This assumption is reasonable since if 30% of the 24 thousand active calls (7,200) need a relay, a super node roughly relays one complete call at any instant. Further, in Skype, a super node does not relay more than one call at any instant. Thus, the power consumption per super node, $w_{\Delta}$, is $\frac{0.89\,k}{5\,k} = 0.178\,W$ in order for c/s and p2p systems to be

equivalent in terms of energy efficiency. When the servers are under utilized, say 50%, $w_\Delta$ is twice its original value (0.356 W). The small value of $w_\Delta$ suggests that if the super nodes were to consume more power than this value in order to handle the signaling, NAT keep-alives, and media relaying workload, a p2p system using super nodes will become energy inefficient as compared to a c/s VoIP system.

Due to the low precision of our power meter, we are not able to ascertain if Skype super node and relaying power consumption is close to $w_\Delta$. However, we speculate that the power consumed by super nodes and relays running on desktop machines may likely be close to the $w_\Delta$ calculated above. The reason is that the CPU of a relatively unloaded machine running a Skype super node or relay may be woken often to service these requests, thus incurring the small power draw to cause it to go above $w_\Delta$. On the contrary, handling an additional job on a loaded server causes almost no additional CPU wakeups.

The analysis reveals that in a VoIP system replacing PSTN, hardphones and switching equipment consume 99.98% of the total energy consumed by the VoIP system. Thus, in order to make VoIP system more energy efficient, we need to take advantage of techniques that allow powering down these devices when idle. In the next section, we discuss the use of these techniques.

## 8.6 Recommendations for Reducing the Power Consumption of VoIP Systems

In this section, we discuss using a number of existing techniques that can potentially reduce the energy consumption of hardphones, switches and middleboxes, and servers when these devices are idle. As a result, the devices in a VoIP system will potentially only draw power when making or receiving VoIP calls. Observe that unlike cloud-based systems where services can be aggregated on a smaller number of servers to improve utilization and reduce energy wastage, it is not possible do so in a VoIP system. The reason is simple: the users want to receive and make calls through their telephones and it is simply not possible to aggregate phones similar to aggregating compute jobs on a server.

Our analysis showed that hardphones, broadband modems, home routers, and enterprise

switches comprise the biggest chunk of the total energy consumed in a VoIP system. To reduce the energy consumption of hardphones, the various components of the phone including LCD display, processor, and ethernet jack should be powered down when not in active use. The former two can be accomplished by turning off the LCD display and and by making use of energy efficient processors, whereas the later can be accomplished using energy efficient ethernet [Christensen *et al.*, 2010]. If the phones were only used for eight hours a day and were powered down or ran on minimal power (<0.1 W) during the remaining 16 hours, it will bring down the per user per month energy bill from 79 cents to 53 cents in T-ITSP like systems, and from 52 cents to 32 cents in enterprise VoIP systems.

In T-ITSP like systems, the hardphones must send keep-alive messages over UDP every 15 s to keep the NAT bindings alive. Such wasteful traffic prevents the phones and home routers from taking advantage of any sleep modes available on the device. To eliminate such wasteful traffic, the phones can establish a permanent TCP connection with the SIP server. Further, the ISP's can setup a SIP phone on the broadband modem which is typically not behind a NAT device. When the SIP user agent on the broadband modem receives an incoming call, it can wake up the home router and the phone using techniques such as Wake-on-LAN [Wake-on-LAN, 2010] to receive incoming call. This technique can further bring down the per user per month energy bill for running VoIP phones.

Our analysis also indicated the number of servers needed to support a large VoIP user base is fairly small; one SIP server can handle registration events and NAT keep-alive traffic for 500 thousand users, and RTP relay server can relay calls for 15 thousand calls. By setting up the VoIP user agents on cable modems, the NAT keep-alive traffic can potentially be eliminated. By using advanced NAT traversal techniques, such as ICE [Rosenberg, 2010] to allow user agents to detect network conditions, the use of RTP relay server can be further minimized. Together, these techniques can significantly reduce the power consumption of VoIP servers.

## 8.7 Related Work

Nedevschi *et al.*[Nedevschi *et al.*, 2008] have developed models describing the relative power efficiency of c/s and p2p architectures for generalized network applications (e.g., file-sharing), and conclude that p2p approaches use system energy more efficiently than the c/s ones. Similarly, Valancius *et al.*[Valancius *et al.*, 2009] argue that building p2p nano-data centers on the Internet gateway devices provides energy savings over traditional centralized data centers. In both papers, the energy savings argument boils down to data center servers (1) needing cooling, network, and other overheads measured by a multiplicative factor called *Power Utilization Efficiency - PUE* and (2) having significant baseline power consumption (i.e., power consumption when idling). Typical data center PUEs range from 1.2–2, while the PUE of a peer is 1 (e.g., home air-conditioning is already running) and peers are on anyway, so processes running on peers escape this baseline cost.

We examined the relative energy efficiency of c/s and p2p VoIP systems, and found, intriguingly, that the energy consumption of a peer does not need to be very large in order for a p2p architecture to be *less* energy efficient than a c/s one.

## 8.8 Conclusion

We identified the key components that are implemented on servers in a c/s VoIP system and by super nodes in a p2p VoIP system. We presented a model for understanding power consumption of c/s and p2p VoIP systems. We performed a number of experiments to determine the power consumption of different components of c/s and p2p VoIP systems. Our model, analysis, and measurements indicate that for VoIP systems used as a replacement for always-on PSTN system, the power consumed by hardphones and connected network devices (broadband modems, home routers, and enterprise switches) overwhelmingly dominate the total power consumed by the VoIP system and the per user per month cost is less than a dollar in such systems. Moreover, when comparing c/s and p2p VoIP systems, our results show that even when super nodes consume relatively small power for system operation, the p2p VoIP system can be less energy efficient than a c/s VoIP system. Further, we demonstrated the presence of NATs as the main obstacle to building energy efficient VoIP

systems.

# Part III

# Measurement

# Chapter 9

# Measurements – Skype

## 9.1 Introduction

Skype [Skype, 2010a] is a peer-to-peer (p2p) VoIP application developed by the organization that created Kazaa [Liang *et al.*, 2004]. Skype allows its users to place voice and video calls, and send instant messages to other Skype users, similar to the MSN and Yahoo IM applications. However, the underlying protocols and techniques it employs are quite different.

Like its file sharing predecessor Kazaa, Skype is an overlay network of users running the Skype application. There are two types of nodes in this network, namely, ordinary nodes (ON) and super nodes (SN). Both ON and SN run the same Skype application. An ordinary node is a Skype node that is behind a restrictive NAT or a firewall device and connects to one or more super nodes. A super node is an ordinary node's end-point on the Skype network. Any node with a public IP address having sufficient CPU, memory, and network bandwidth is a candidate to become a super node. The super nodes store information about online users and provide media relaying services to the Skype nodes that are behind a restrictive NAT or a firewall device. At startup, both ON and SN must authenticate themselves with the Skype login server. Although not a Skype node itself, the Skype login server is an important entity in the Skype network as it stores the user names and passwords of Skype users. This server also ensures that Skype user names are unique across the Skype name space. Starting with the Skype version 1.2, the contact list is also

stored on the login server. Figure 9.1 illustrates the relationship between ordinary nodes, super nodes and the login server.

Apart from the login server, there are SkypeOut [Skype, 2010f] and SkypeIn [Skype, 2010e] servers which provide PC-to-PSTN and PSTN-to-PC bridging. SkypeOut and SkypeIn servers do not play a role in PC-to-PC call establishment and hence we do not consider them to be a part of the Skype peer-to-peer network. Starting with version 5.0 beta, Skype supports five party video conferencing. However, it uses managed servers for a video conference involving more than two participants.

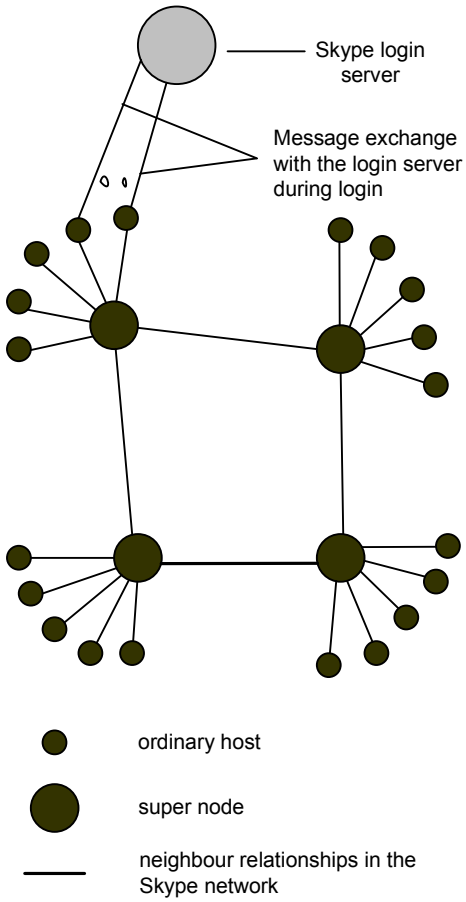



Figure 9.1: Skype Network. There are three main entities: supernodes, ordinary nodes, and the login server.

We present the experimental setup to analyze the Skype functionality and then describe the key functions of Skype in Section 9.2. We then describe our experiments to gain insights into the Skype's relay selection mechanisms in Section 9.3. For brevity, we refer to the Skype

application as SA.

## 9.2 Functionality

We performed experiments for the Windows Skype version 1.4.0.84 and for the Linux Skype version 1.2.0.18 in December 2005. We used traffic analysis, shared library and system call interception techniques to analyze various functions of the Skype protocol such as login, call establishment, search, NAT and firewall traversal, and conferencing.

We devised an experimental setup for analyzing the protocol of the Skype Windows version. In the first setup, both SA's were run on machines with public IP addresses; in the second setup, one SA was behind a NAT with endpoint-independent mapping and address and port dependent filtering (port-restricted NAT); in the third setup, both SA's were behind a port-restricted NAT and UDP-blocking firewall. The NAT was configured using Linux 'iptables'. We used Ethereal [Ethereal, 2010] to monitor the network traffic and used NetPeeker [Net Peeker, 2010] to tune the available bandwidth to allow us to analyze the Skype operation under network congestion.

For the Skype Linux version, we used shared library and system call interception techniques to gain more insights into the Skype protocol. In Linux, when a program starts, it dynamically loads the shared libraries pointed to by the `LD_PRELOAD` environment variable before loading any other shared library including libc [GNU C Library, 2010]. This feature makes it possible to overload a libc function such as `strcpy()` or a system call such as `send()`. When `LD_PRELOAD` is set to a library containing an overloaded `strcpy()` function, and the program which contains the `strcpy()` calls is executed, the overloaded `strcpy()` is called. The parameters passed to this overloaded `strcpy()` function can be displayed or any appropriate action can be taken. Also, the overloaded `strcpy()` function can then call the `strcpy()` function defined in libc. Figure 9.2 shows the sample code for the overridden `strcpy()` call.

Next, we briefly describe the key functions of the Skype protocol. A detailed description of the experiments can be found in [Baset and Schulzrinne, 2006].

```
char* strcpy(char* dest, const char* src) {
   void *handle = NULL;
   double (*mystrcpy)(char* dest, const char* src);
   long temp;

   handle = dlopen("/lib/libc.so.6", RTLD_LAZY);
   mystrcpy = dlsym(handle, "strcpy");
   temp = (*mystrcpy)(dest, src);
   dlclose(handle);

   return dest;
}
```

Figure 9.2: strcpy() overload

### 9.2.1 Search

Skype claims to have implemented a '3G P2P' or 'Global Index' [Skype, 2010b] technology, which is guaranteed to find a user if that user has logged in the Skype network in the last 72 hours. Skype allows wildcard search. Such a search mechanism is costly to implement using DHTs in terms of number of messages as it requires the implementation of inverted index. Therefore, it is unlikely that Skype uses DHTs for its peer-to-peer network, and for searching a user name.

### 9.2.2 NAT Traversal

We believe that each SA uses a variant of the STUN [Rosenberg *et al.*, 2008] protocol to determine the type of NAT and firewall it is behind. We also believe that there is no global NAT and firewall traversal server because if there was one, the SA node would have exchanged traffic with it during the login and call establishment phases in the many experiments we performed.

### 9.2.3 Overlay Connectivity

The Skype network is an overlay network and thus each SA needs to build and refresh a table of reachable nodes. In Skype, this table is called host cache (HC) and it contains IP address and port number of super nodes. Starting with Skype v1.0, the HC is stored in an XML file. In the earlier versions, the host cache was stored in Windows registry.

### 9.2.4 Call Establishment

Each SA uses TCP for signaling, and both UDP and TCP for transporting media traffic. If the caller, callee or both are behind a restrictive NAT and firewall, the SA uses another Skype node for relaying media traffic. Further, the caller and callee use TCP to exchange media packets if firewalls block UDP.

### 9.2.5 Codecs

Skype uses wideband codecs [Skype, 2010c] which allows it to maintain reasonable call quality at bandwidth as low as 8 kb/s.

### 9.2.6 Audio Conferencing

Skype does not do full-mesh audio conferencing [Lennox and Schulzrinne, 2003] and instead uses one of the participants, which is typically the conference initiator, as the audio mixer.

### 9.2.7 Video Conferencing

As discussed in Section 5.2, Skype uses managed servers for video conferencing with three or more participants. For one-to-one video calls, the SA's in the session may use another Skype user as a relay if they cannot directly exchange packets. The Skype application limits the bandwidth of a video call involving another SA as a relay.

## 9.3 Skype Relay Calls

Skype addresses the issue of network connectivity between SA's by using other Skype nodes as relays. We performed experiments to gain insights in the Skype relay selection mechanism, and to determine the characteristics of relay calls and machines relaying the Skype calls, without modifying the Skype application. During a four month period, we attempted over 38,000 Skype calls under three different network setups of which 18,000 were successful calls. Our results show that although Skype exhibits geographical locality in relay selection, the median one way call latency is not insignificant, suggesting the benefit of further tuning of Skype relay selection mechanism. Second, we discuss factors impacting the success rate

of Skype calls. Finally, we find that approximately 46% of the successful calls were through relays run by users in universities. The result suggests that Skype is free-riding on the network bandwidth of universities.

Section 9.3.1 describes the experimental setup and Section 9.3.2 discusses factors affecting success rate of relay calls. In Section 9.3.3, we discuss the geographical, ISP, and autonomous system distribution of relay nodes and relay calls and online presence of relay nodes.

### 9.3.1 Experimental Setup

A key aspect of Skype's robust connectivity is its ability to traverse NAT and firewalls by routing a video or a voice call through one or more Skype relays. We refer to the node relaying a Skype call as relay node (RN) and the call being routed as a relay call (RC). A relay node is a SN that has sufficient bandwidth to relay a voice or a video call. To study Skype's robust connectivity, we devised an experiment in which a call was established between two Skype clients running on machines in our lab at Columbia University. The RTT between the two machines was less than one ms and both machines were connected to the same LAN. The network conditions between the caller and callee machines were configured so that they were forced to use a RN. Specifically, the experiment was performed under three different network setups: unrestricted connectivity (Figure 9.3), caller and callee behind NATs with address and port dependent mapping and filtering (Figure 9.4)[1], and direct-blocked setup, in which the caller and callee, having otherwise unrestricted connectivity, could not send packets to each other (Figure 9.5). We refer to these setups as unrestricted, NAT, and direct-blocked in this chapter. For the NAT and direct-blocked setup, we performed the experiment with and without deleting the host cache. The experiments were performed from March 27 to July 3, 2007. We used Skype v3.2 for Windows XP in our experiments.

We wrote a script using AutoIt [AutoIt, 2010] that used the Skype API [Skype, 2010d] to automate the establishment of Skype calls, checked the call status, and collected the relay data at caller and callee SA. The duration of a call was configured to be one minute and during this time, the script gathered the number of packets sent by the caller and

[1] see Chapter 2 for background on NAT types


Figure 9.3: Unrestricted

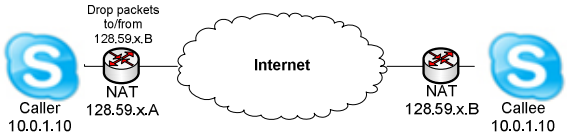



Figure 9.4: NAT: caller and callee behind address and port dependent NAT.

callee SA's to the unique IP address and port number pairs using WinDump [Riverbed Technology, 2010]. With a packetization interval of 30 ms (as used by Skype), the caller machine should approximately send 1,800 packets to the RN within a minute and vice versa. After terminating each call, we parsed the WinDump data and labeled the IP address and port number that most frequently appeared in the WinDump data as the relay node.

We also found that the presence of host cache (a list of online Skype nodes) affected the success rate of RCs. Throughout the experiment, we collected the IP addresses and port numbers of Skype relay nodes and tracked their online status by sending a specially crafted Skype message to them. We also calculated the geographical, ISP, and autonomous system (AS) distribution of RNs and RCs using MaxMind [Maxmind, 2010] and AS number lookup utility [Kondo, 2010] and measured the round trip time from our lab to the RNs. We used the RTT data to gain insights into the efficiency of the Skype relay selection algorithm.

### 9.3.2 Factors Impacting the Success Rate of Skype Relay Calls

We established over 38,000 calls over a three month period for the three different network setups described in Section 9.3.1. For 37,761 calls, the network was configured such that Skype was forced to select a relay. Out of these calls, approximately 18,000 were successful and the success percentage depends on the network setup. Seventeen percent (3,146) of the successful RCs used a different relay from caller to callee and from callee to caller. There was almost no difference between the call success rates of video and voice calls. Table 9.1 shows the detailed call statistics for the three network setups.

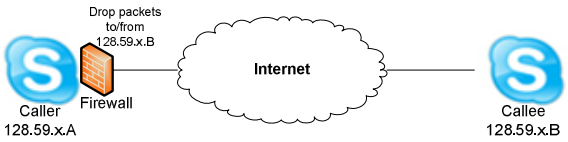



Figure 9.5: Direct-blocked: packets between caller and callee are dropped.

Table 9.1: Statistics for Skype relay call experiments.

| | Experimental setups | | | | | Aggregated |
|---|---|---|---|---|---|---|
| | Unrestricted | NAT | | Direct-blocked | | |
| | | HC deleted | HC not deleted | HC deleted | HC not deleted | |
| Call trials | 867 | 4,649 | 5,658 | 15,774 | 11,680 | 38,628 |
| Successful calls | 867 | 339 | 5,567 | 2,586 | 8,597 | 17,962 |
| Success rate (%) | 100% | 7.3% | 98.4% | 16.4% | 73.6% | 46.5% |
| Relays found | N/A | 379 | 2,843 | 2,718 | 4,317 | 9,584 |
| % of calls through US relays | N/A | 74% | 81.2 | 91.6% | 92.4% | 88.2% |
| % of succ. calls w/two relays | N/A | 28.6% | 17.7% | 31.2% | 14.6% | 17.5% |
| One-way call latency (ms) | N/A | 29.1 | 95.7 | 8.8 | 13.3 | 43.6 |

We posed the following questions: how does the network and retention of host cache from previous runs of Skype application impacts the success rate of relay calls. Observe from Table 9.1 that the success rate of RCs for the direct-blocked setup that retains HC after a call trial is lower than the unrestricted and NAT setups. We have observed that when a SA comes online, it sends a notification to the Skype users in its contact list. Since the direct-blocked setup will drop any packets between caller and callee, this notification is dropped. When a caller initiates a call, it must search for the callee SA in the Skype network. After finding IP address of the callee SA, the caller SA sends the signaling traffic through another SA. Further, the caller and callee must find a Skype node for relaying the media traffic. However, unlike the NAT setup, the relay search is initiated only during call establishment, as both caller and callee had earlier assumed that they had unrestricted connectivity. Thus, for the direct-blocked setup, an attempt to find a signaling and media relay at the time of call establishment decreases the success rate of RCs in the direct-blocked setup as compared to the NAT setup.

To check if the retention of host cache from the previous runs of SA impacts the success

rate of relay calls, we deleted the HC after every trial for the three network setups and measured the call success rate. In the absence of the HC, a SA uses bootstrap nodes maintained by the Skype to join the Skype network. For experiments in the NAT and direct-blocked setup where HC was deleted after every call trial, we observed a strange phenomena with the call success rate. Initially, the call success rate was 100%, but as the time passed, there was a drastic drop in the success rate, and ultimately all calls started to fail. We experimented with different Skype user names for the caller and callee and observed the same phenomena. Interestingly, we did not observe this phenomena for the unrestricted setup. We offer a possible explanation below.

We have studied that the intermediate SNs contacted by the caller during the callee search process cache the results [Baset and Schulzrinne, 2006]. Since the HC is deleted after every call trial, a caller SA must contact the same set of bootstrap peers during login. It takes time, on the order of minutes, for Skype to build a new HC from scratch. Moreover, a Skype callee behind a NAT and direct-blocked setup must publish a reachable IP address, obtained through STUN [Rosenberg *et al.*, 2008] and TURN [Mahy *et al.*, 2010] like mechanisms, in the Skype network. It is likely that for a new call trial, the callee's reachable address is not updated in the SNs cache and the caller SA always reaches the SNs caching the old reachable address of callee.

Thus, we attribute the RC failures to (1) the stale IP address and port number of callee and its SN in the cache of other Skype nodes and (2) the inability of Skype to find a relay at the time of call establishment.

### 9.3.3 Characterization of Skype Relay Nodes

During our experiments, we found 9,584 unique relays. In this section, we classify the IP address of these RNs according to their geographical, ISP, and autonomous-system (AS) distribution, and present RTT and uptime measurements for them. We also present a geographical distribution of relay calls and comment on the efficacy of the Skype relay selection algorithm.

Table 9.2: Top five organizations with relay nodes

| .edu | | | | |
|---|---|---|---|---|
| Organization | % of RNs | % of calls | Median Uptime (hours) | Median RTT (ms) |
| Columbia | 2.7 | 15.1 | 3.3 | 0.3 |
| Yale | 2.1 | 5.1 | 3.9 | 9.8 |
| Georgia Tech. | 1.3 | 5.2 | 4.1 | 30.1 |
| MIT | 1.1 | 0.9 | 6.2 | 0.9 |
| NYU | 1.1 | 2.3 | 5.9 | 2.27 |
| .com | | | | |
| Road Runner | 9.8 | 7.2 | 4.6 | 15.5 |
| AOL | 4.0 | 4.5 | 4.2 | 95.6 |
| Mindspring | 2.6 | 1.6 | 3.4 | 58.6 |
| Rogers | 2.5 | 1.5 | 0.2 | 34.9 |
| Charter | 1.6 | 1.0 | 3.5 | 32.8 |
| .net | | | | |
| Comcast | 18.1 | 12.3 | 3.9 | 29.1 |
| Optimum Online | 9.2 | 6.1 | 3.8 | 14.9 |
| Cox | 2.6 | 1.4 | 2.2 | 81.8 |
| SBC | 2.5 | 1.5 | 6.7 | 38.3 |
| Ameritech | 1.5 | 0.7 | 6.2 | 40.7 |

#### 9.3.3.1 Relay Distribution

We used MaxMind [Maxmind, 2010] to determine the geographical distribution of relay node IP addresses and `nslookup` for reverse DNS lookup. Out of 9,584 RNs, 89.36% were present in North America and 11.5% were in Europe. The US-based RNs comprised 82.64% (7,920) of the total relay nodes and 21.18% of the total RNs were in New York state. We also classify the RNs using their domain names obtained from reverse DNS lookup. Table 9.2 shows five organizations with a .edu, .net, and .com suffix having the most number of unique RNs, the percentage of calls routed, and median RTT. An interesting aspect is that 22.4% (2150) RNs had a .edu suffix, which indicates their affiliation with universities. In Section 9.3.3.3, we present the distribution of relay calls through nodes at educational institutions.

Besides geographical and domain name classification of 9,584 RN IP addresses, we also used the `aslookup` [Kondo, 2010] utility to discover the AS number of the IP addresses of RNs. The tool contacts one or more `whois` servers to obtain the AS number of an IP address. The `aslookup` was successfully able to retrieve AS numbers of 7,954 (83%) IP addresses that belonged to 336 unique autonomous systems. The statistics are summarized

Table 9.3: Top ten AS with the largest number of unique RNs

| Organization | % RNs | AS # | % succ. calls | Median RTT (ms) | Organization | % RNs | AS # | % succ. calls | Median RTT (ms) |
|---|---|---|---|---|---|---|---|---|---|
| Cable Vision | 9.2 | 6128 | 6.1 | 15.3 | AOL | 3.9 | 1668 | 3.4 | 95.6 |
| RR-NYC | 7.3 | 12271 | 5.9 | 16.4 | Comcast | 3.7 | 33287 | 2.8 | 30.1 |
| Rogers | 4.5 | 812 | 2.5 | 52.1 | Columbia Univ. | 3.1 | 14 | 17.4 | 0.28 |
| SBC | 4.4 | 7132 | 2.5 | 46.6 | Cox | 2.9 | 22773 | 1.7 | 65.6 |
| Comcast | 4.4 | 7015 | 3.6 | 16.2 | Comcast | 2.6 | 33657 | 2.0 | 34.5 |

in Table 9.3 and Figure 9.6. Out of 336 ASes, the top ten AS hosted 46% of RN IP addresses while the top twenty percent covered 90% (8,627) of RN IP addresses. Note that New York state had 21% of the total RN IP addresses and observe that a New York city ISP, RR-NYC (Road Runner) and Columbia University in the city of New York have 10.4% of the total RN IP addresses. This result gives an indication that a SA attempts to select a RN that is geographically closer to caller and callee.

Observe from Table 9.3 that a higher percentage of RNs belonging to one organization does not imply that more relay calls are routed through hosts in that organization. Cable Vision, an ISP, has 9.2% of total RNs but they only relay 6.1% of the calls. Columbia University has 3.1% of the total RNs but they relay 17.4% of the total RNs calls. This result gives another indication that Skype is attempting to optimize the RN selection. However, as we will discuss in Section 9.3.3.2, this selection is far from optimal.

Figure 9.7 shows the number of unique RNs found for the complete duration of the experiment, i.e., from March 27 to July 3, 2007, and is a linearly increasing line. This result indicates that the total population of RN candidates is likely much larger and there are undiscovered Skype RNs which could have been found if the experiment was not stopped on July third.

Guha *et al.* [Guha *et al.*, 2006] had mined the HC of Skype to obtain a list of 2,081 Skype SNs. We compared our RN list to Guha's SN list and found that the two lists had only six IP addresses in common. One reason for such a minimal overlap is a time gap of more than one year between the two studies. The other reason is that Guha mined SN list from Skype client's HC. Having a SN listed in the HC does not necessarily imply that those nodes will be selected as a relay. Instead of mining the HC for SNs which may be selected

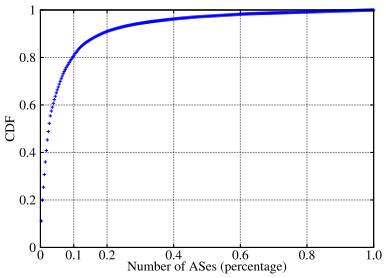



Figure 9.6: Number of relays nodes (RN) per AS.

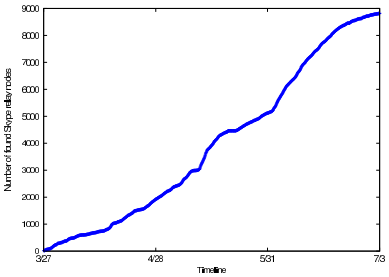



Figure 9.7: Number of unique RNs found.

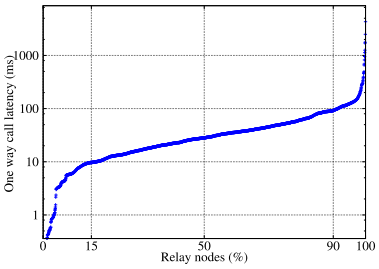



Figure 9.8: CDF of one way call latency from caller to RN.

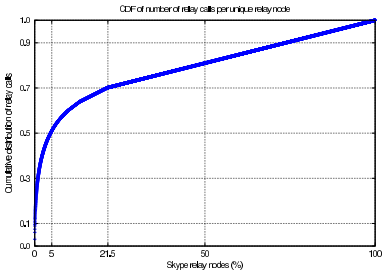



Figure 9.9: CDF of relay calls (RCs) per unique RN.

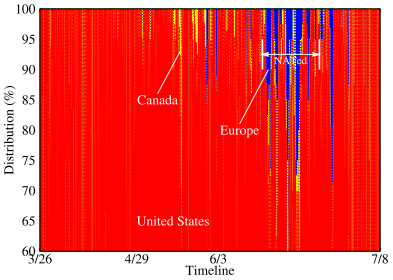



Figure 9.10: Geographical distribution of RCs.

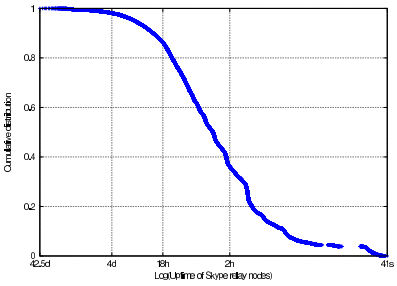


Figure 9.11: CCDF of RNs uptime.

as a relay, we established relayed Skype calls and tracked the selected RNs.

#### 9.3.3.2 Packet Delay of Relay Nodes from Caller

To gain more insights in the efficacy of Skype relay node selection algorithm, we also analyzed the network latency from our lab to the relay nodes. We measured the RTT of RNs by sending especially crafted Skype messages to each RN. Figure 9.8 shows the CDF of one way call latency. The average and median one way latency for RNs measured across all experimental setups are 52.2 ms and 43.6 ms. Since both caller and callee machines were located in our lab and have the same RTT to the RN, one-way median network latency for a call is 43.6 ms. For the NAT setup, one way median network latency between our machines in our lab running the caller and callee SA and RN is 95.6 ms and for direct-blocked setup, it is 13.3 ms. The NAT setup is likely to be a common case on the Internet and a median latency of 95.6 ms between two machines behind NATs, having otherwise a RTT of less than one ms, is not insignificant. Moreover, one-way median latency for the NAT setup is significantly higher than the direct-blocked setup. One could argue that for NAT setup, a

SA will choose a relay at the time of login, whereas in direct-blocked setup a SA chooses a relay at the time of call establishment. Thus, a SA would have more time to optimize relay selection for the NAT setup, and consequently, one-way call latency should be lower for the NAT setup. However, we did not observe low latencies for calls in the NAT setup and the cause remains unclear.

#### 9.3.3.3 Call Distribution per Relay

We characterize the distribution of relay calls per RN and determine whether there is any temporal locality in the selection of a RN, i.e., for how many subsequent calls does a SA uses the same relay.

As listed in Table 9.1, there are 9,584 RNs that relay 17,095 calls so on average each RN relays two calls. Figure 9.9 and Figure 9.10 show the CDF of RCs per unique RN and geographical distribution of RCs. Approximately, 6.3% (603) of the 9,584 nodes belonging to 74 autonomous systems relayed 50% of the total calls. Clearly, a significant portion of relay calls are routed through a small subset of RNs and this refutes any conjectures about Skype relay algorithm selecting a random RN. The result will be clearer when we discuss the uptime of RNs as some nodes are online for many days.

As listed in Table 9.1, 88.2% of successful RCs were routed through relays in US, and 3.7% and 6.93% through relays in Canada and Europe. Observe that for the NAT setup, only 81.2% of the calls were routed through US-based relays as compared to 92.4% for direct-blocked setup which indicates that there is room for improvement in Skype relay selection algorithm. This is also highlighted in Figure 9.10 which shows an increase in the use of European relay nodes during the month of June when NAT setup experiments were performed. It is unclear why Skype relatively uses more non-US relays when both caller and callee are behind address and port dependent NATs.

To understand if Skype is free riding on the network bandwidth of universities, we performed a reverse lookup of RN IP addresses and group the results by edu, net and com suffixes. In Figure 9.12, we plot the percentage of RN's with edu, net, com, and other suffixes that relay the calls. As shown, approximately 46% of the relay calls were through RN's that have a .edu suffix. The result is an indication that universities are implicity

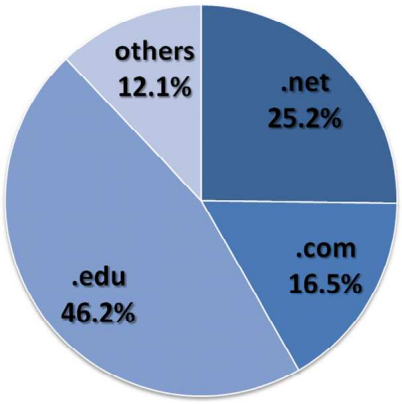



Figure 9.12: Percentage of successful relay calls through machines with a DNS suffix ending in edu, net, com, and other.

helping towards the health of the Skype network. The reason why a significant percentage of relayed calls is through relays in universities is because the universities typically charge a flat rate for bandwidth and electricity provided to student dorms. Consequently, the students running the Skype application in their dorms do not have any incentive to shut down the Skype application.

Figure 9.13 shows the timeline of the number of times the top five RNs were selected as a relay and their uptime. From our results we noted that the maximum number of times a relay node was selected in consecutive trials was seven. These results indicate that Skype is caching the RN lists. Surprisingly, the RN with the largest uptime of 42.5 days was only once selected as a relay during the course of our experiments. It is possible that this node was selected as a relay for calls placed by other users. However, it is difficult to obtain this information due to the closed nature of the Skype protocol.

#### 9.3.3.4 Uptime of Relays

We tracked the presence of RNs discovered in the three experimental setups. Every few minutes, we sent a specially crafted Skype message, thereafter called Skype-ping, to RN IP address and port number to which a Skype client across different versions is known to respond. If there was no reply, we consider the Skype node to be offline. We conducted this experiment from March 27 to July 3, 2007.

Our result shows that the uptime distribution of Skype relay nodes follows a diurnal

pattern. This result is quite similar to the uptime distribution of supernodes found by Guha [Guha *et al.*, 2006]. The likely reason as also mentioned by Guha is that there are more users in the Skype network during the day than during the night. Figure 9.11 shows the CDF of the uptime of RNs. A relay node can be online at different times during the study period. The CDF plot shows all the uptimes of a unique RN and not the cumulative uptime. Over the course of our RN study, the maximum and median observed uptime of RN was 42.5 days and 3.5 hours, respectively. The median uptime for RNs in .edu, .net, and .com domains was 4.5 hours, 3.7 hours, and 2.5 hours, respectively. Note the relatively longer uptime for RNs with a .edu suffix and this again raises the question whether university networks are indirectly supporting the Skype peer-to-peer network.

The median uptime of SNs reported by Guha was 5.5 hours. Perhaps the difference between our RN median uptime and the one reported by Guha is caused by the different duration of the uptime study. We conducted our uptime study over a five month period and discovered 9,584 unique RNs whereas Guha conducted his study over a period of one month for 2,081 unique SNs.

An aspect of Skype which can impact our uptime statistics is that Skype does not have a standardized listening port. SA picks a random port upon installation and additionally, listens for incoming requests at port 80 and 443, the HTTP and HTTPS ports, respectively. There is no guarantee that a SA will always use the same random port picked at installation. Therefore, it is possible that a Skype-ping message may actually never be received by a SA although it may be online. Thus, the uptime results may not accurately represent the uptime of RNs. We tried sending a crafted Skype message to ports 80 and 443 in the hope that a SA will respond since it always listens for incoming requests on these ports; however, in our experiments, we never received a response for crafted Skype messages sent on ports 80 and 443.

#### 9.3.3.5 Summary of Results

82.64% of the RNs were located in US and 6.3% of the total RNs relayed 50% of the calls. This reuse of relay nodes suggests that Skype caches RN information. The median one-way network latency was 43.6 ms and is dependent on the experimental setup. Our

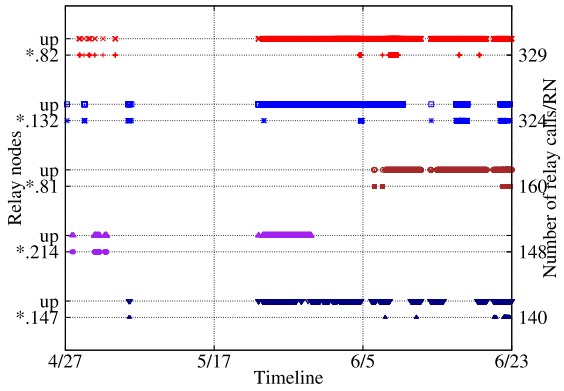



Figure 9.13: Relay selection and uptime duration of top five RNs.

results indicate that the mechanisms for relay selection can further be improved, especially when both caller and callee are behind NAT. The RC failures in the absence of HC can be reduced by quickly updating the SN cache for callee's reachable address. Further, since 50% (approximately 9,000) of the calls are relayed by nodes belonging to 74 autonomous systems, it is possible to use AS number as an approximate metric to search for a relay closer to caller or callee. However, using AS number as a metric to determine node locality depends on the physical geography of AS. Our results also indicate that the machines running Skype applications in universities greatly contribute towards the health of the Skype network.

### 9.3.4 Related Work

The closest study to ours was an experimental study of Skype SNs conducted by Guha et al. in 2005 [Guha *et al.*, 2006]. Guha monitored the HC of a SA and tracked the population and presence of SNs. It was not certain if these SNs were also acting as relay nodes. We, however, discovered the RNs by establishing and monitoring calls that were relayed through these Skype nodes. Further, we studied the factors impacting Skype RCs success rate, the presence information of RNs and their geographical distribution.

## 9.4 Conclusion

In this Chapter, we described the two tier architecture of Skype. We also gave an overview of various functionalities in Skype such as search, call establishment, NAT traversal, and

conferencing. We then performed experiments to gain insights in the Skype relay selection mechanism. Our analysis indicates that a significant portion (approximately 46%) of the Skype relay calls are routed through machines running in universities. We also observed that for approximately 17.5% of successful relay calls, one relay is used for sending media from caller to callee SA and vice versa. Finally, we observed that the success rate of Skype relay calls depends on the network restriction and the host cache.

# Part IV

# Conclusions

# Chapter 10

# Conclusions

The peer-to-peer (p2p) paradigm for building VoIP systems involves minimal or no use of managed servers and is therefore attractive from an administrative and economic perspective. However, the benefits of using p2p paradigm for building VoIP systems are not without their challenges of protocols and system design, reliability, energy efficiency, and measurement.

This thesis describes protocols and systems for building peer-to-peer communication systems. It defines Peer-to-Peer Protocol (P2PP) [Baset *et al.*, 2007], an open and interoperable protocol for building p2p communication systems. The protocol has been incorporated in the RELOAD protocol [Jennings *et al.*, 2010] which is being standardized in the P2PSIP working group of Internet Engineering Task Force (IETF). The thesis presents the design, implementation, and lessons learned from building OpenVoIP [Baset and Schulzrinne, 2008], an open source peer-to-peer communication system. OpenVoIP has been deployed on PlanetLab. The system demonstrates the feasibility of P2PP and helps understand challenges in building p2p communication systems. The thesis then presents a systematic exploration of issues in designing and building peer-to-peer video conferencing.

The thesis presents an in-depth analysis of reliability in p2p communication systems, the use of TCP for real-time traffic, and energy efficiency of p2p and client-server VoIP systems. It presents a framework to understand the reliability of p2p communication systems and a simple model to understand the reliability of media sessions that require a relay [Baset and Schulzrinne, 2010]. The model can be used to answer questions about the reliability of p2p

communication systems like Skype. An insight from the model is that Skype user agents, that require a relay to exchange media traffic, need at least three relays to maintain a call success rate of 99.9%. Due to the presence of restrictive NATs and firewalls, the user agents may be unable to directly exchange UDP traffic but may establish a direct TCP connection. The thesis presents a feasibility study of using TCP for sending real-time traffic such as voice and video, explores where the delays occur in the TCP for real-time flows, and presents techniques to mitigate the delay impact of TCP [Brosh *et al.*, 2008]. A main observation is that when TCP carries VoIP traffic, the delays due to packet retransmission and inorder delivery mechanism of TCP dominate the total incurred delays, whereas in the case of video flows over TCP, the congestion control mechanism of TCP is mainly responsible for the delays incurred.

The thesis describes a model to compare the energy efficiency of peer-to-peer and client-server communication systems and identifies sources of inefficiency in these systems [Baset *et al.*, 2010]. The model and measurements highlight that in VoIP systems replacing PBX or PSTN, VoIP hardphones consume the largest part of the total energy consumed by the VoIP system and p2p communication systems are less energy efficient than client-server even if the peers consume a small amount of energy for the p2p network operation. Finally, the thesis presents a set of techniques to understand the workings of Skype, a popular p2p VoIP application. In particular, the thesis presents measurements which show that Skype is free-riding on the network bandwidth of universities [Baset and Schulzrinne, 2006; Kho *et al.*, 2008].

# Part V

# Appendices

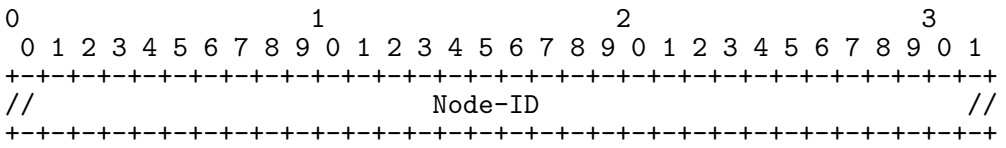
# Appendix A

# P2PP TLV Object Bit Fields

This appendix defines the bit fields for the P2PP TLV objects. The fields which either start or end in // have a variable length. All the other fields have a fixed length.

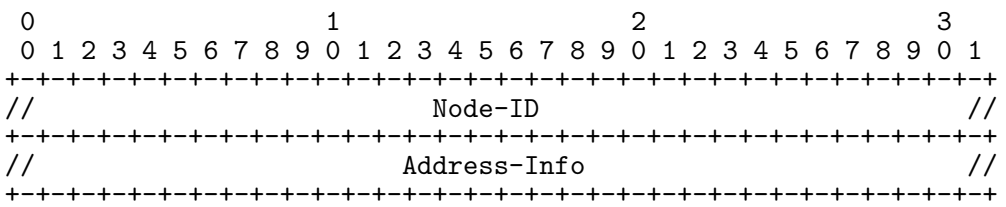
## A.1 Node-ID

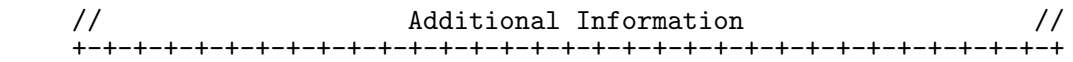
The node-id object contains a fixed length identifier for a node.




## A.2 Node-Info

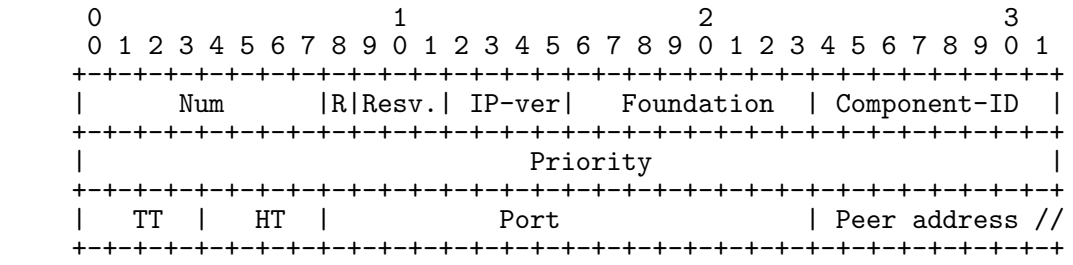
The node-info object contains a node-id (Appendix A.1) and an address-info object (Appendix A.3), and may include other objects such as uptime (Appendix A.9), certificate (Appendix A.11), and node-resource utilization (Appendix A.4).






## A.3 Address-Info

The address-info object contains the host, server-reflexive, and/or relay address of a node. The server-reflexive and relay addresses are gathered using STUN [Rosenberg *et al.*, 2008], TURN [Mahy *et al.*, 2010], and ICE [Rosenberg, 2010].




Num (4 bits): The number of ICE candidates.

R flag (1 bit): If set, rel-addr and rel-port are included as defined in ICE [Rosenberg, 2010].

IP-Ver (4 bits): The IP version number, 4 or 6.

Foundation (8 bits): The foundation field as defined by ICE [Rosenberg, 2010].

Component-ID (8 bits): The component-ID field as defined by ICE. The values for various components are defined in Table A.6.

Priority (32 bits): The priority of the address obtained through ICE.

TT (4 bits): The transport type of the address. One of UDP (0000), or TCP (0001).

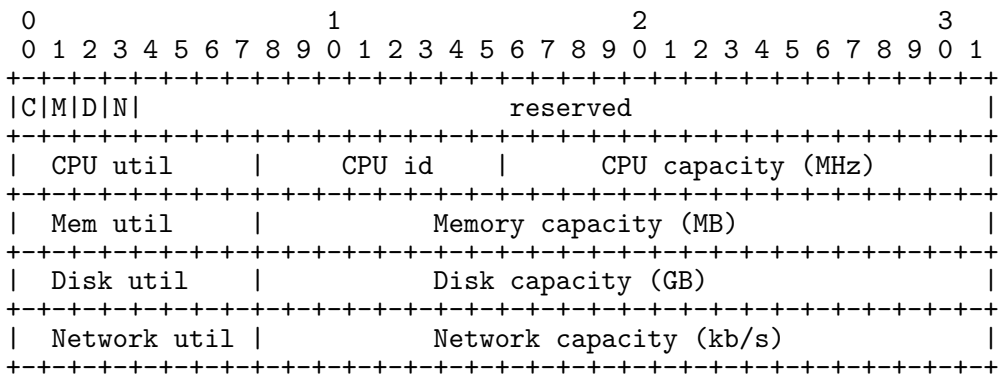
HT (4 bits): The address type of the peer as defined in ICE. One of host (0000), server reflexive (0001), peer reflexive (0010), or relayed candidate (0011).

Port (16 bits): The port on which this node listens for requests.

Peer address (variable): The IP address of the peer. Its length depends on the IP-Ver field.

## A.4 Node-Resource-Utilization

The node-resource-utilization object contains the CPU, memory, disk, and network utilization and total capacity. A flag at the beginning of the object indicates which of these fields are included in the object.




C flag (1 bits): If set, CPU utilization and capacity are included. For multicore processors, this field in conjunction with the CPU id field determines whether to report the average utilization across all cores (CPU id = 0), or of each core starting from the first core (CPU id 1, 2, ...).

M flag (1 bit): If set, memory utilization and memory capacity of the node, measured in megabytes, is included.

D flag (1 bits): If set, the disk utilization and the total disk capacity measured in gigabytes (GB), is included.

N flag (1 bits): If set, the network utilization and the total network capacity, measured in kb/s, is included.

CPU id (8 bits): If the value is zero, then the CPU capacity reports the average utilization across all cores for multiprocessors. Otherwise, it reports utilization and capacity for each core.

CPU capacity (16 bits): The CPU capacity in MHz as determined by the CPU id field.

Mem util (8 bits): The memory utilization of the machine.

Memory capacity (24 bits): The memory capacity of the machine in megabytes.

Disk util (8 bits): The disk utilization of the machine.

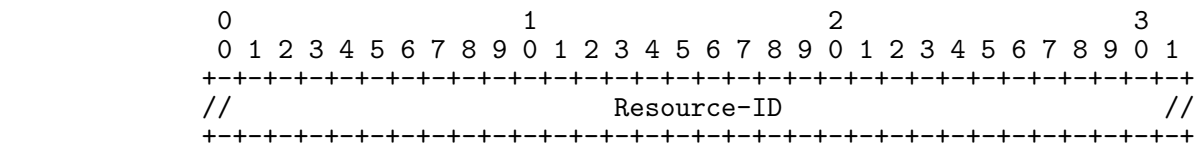
Disk capacity (24 bits): The disk capacity of the machine in gigabytes.

Network util (8 bits): The network utilization of the machine.

Network capacity (24 bits): The network capacity in kb/s.

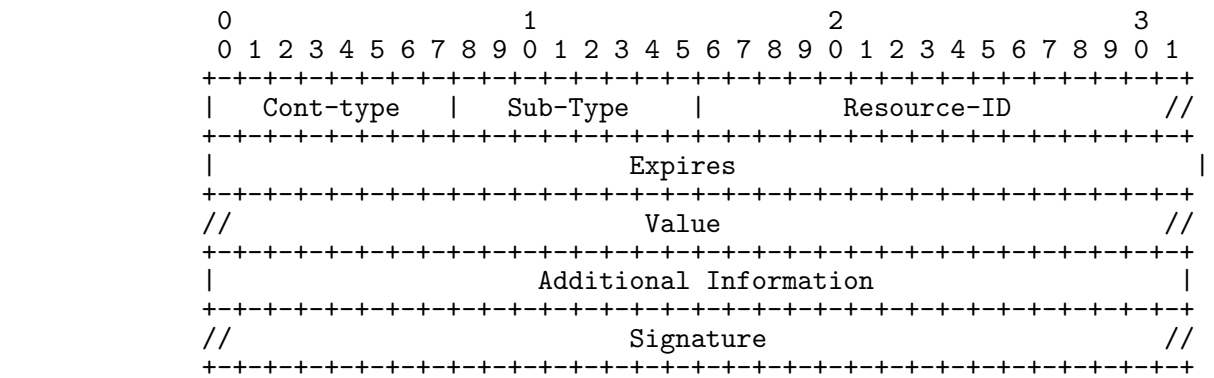
## A.5 Resource-ID

The resource-id is an identifier for a resource-object.




## A.6 Resource-Object

The resource-object is a unit of information stored in the overlay. It is extensible and contains the following fields:



Cont-type (8-bits): An identifier for the type of the content in the resource-object. The values for the existing content types are defined in Table A.5.

Sub-type (8-bits): An identifier which further qualifies the content type as defined by cont-type.

Resource-ID (variable): The identifier of the resource-object.

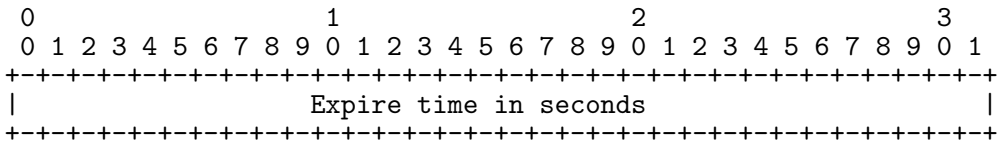
Expires (32-bits): The expires object (Appendix A.7) denotes the timer after which the object being stored can be safely removed.

Value (variable): The value of the object. It depends on the content-type and sub-type.

Signature: The cryptographic signature (Appendix A.14) over all the fields in the resource-object.

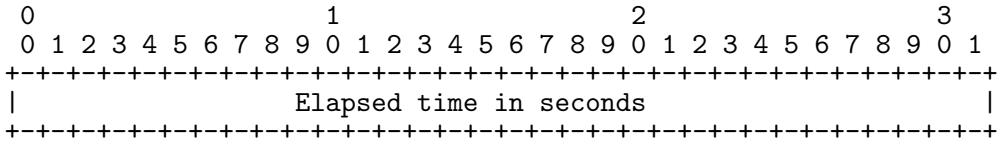
## A.7 Expires

The time in seconds after which the relevant information becomes invalid. It's length is 32-bits.



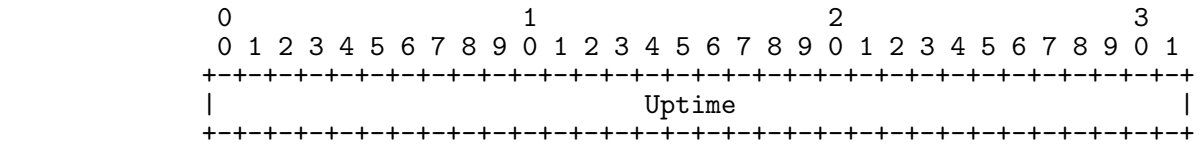

## A.8 Elapsed

The time elapsed since the operation under consideration. It's length is 32-bits.



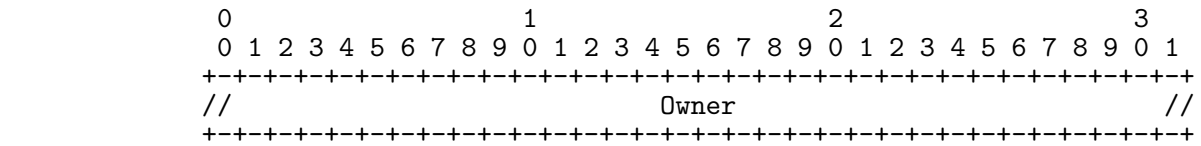
## A.9 Uptime

The time elapsed since the node has been online. It's length is 32-bits.



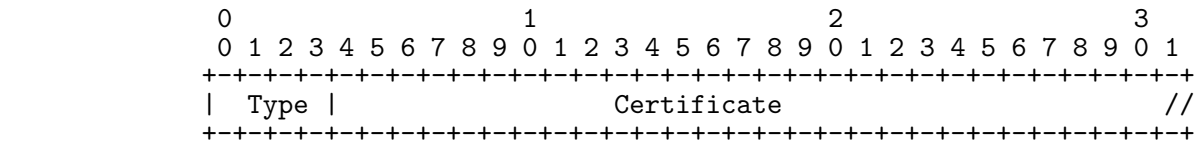

## A.10 Owner

The owner (or publisher) of the resource-object. It is typically the node-id of the node publishing the object but can also be an string identifier for the device publishing the object such as "hardphone" or "desktopphone".




## A.11 Certificate

An X.509 certificate [International Telecommunication Union (ITU), 2006] in DER encoding [International Telecommunication Union (ITU), 1997]. It's length depends on the size of the certificate.

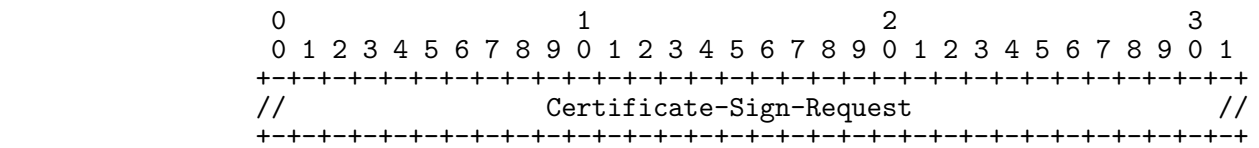



Type (8-bits): Self-Signed (0x00), Server-Signed (0x01).

Certificate: The certificate itself.

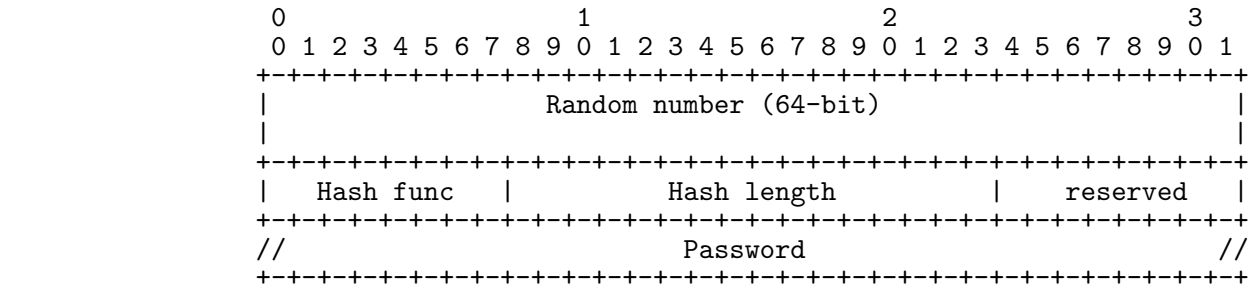
## A.12 Certificate-Sign-Request

A certificate sign request [Nystrom and Kaliski, 2000] encoded in ASN.1. Its length is variable.




## A.13 Password

The cryptographic hash of the password and a 64-bit random number.




Random number (64-bits)): A 64-bit random number.

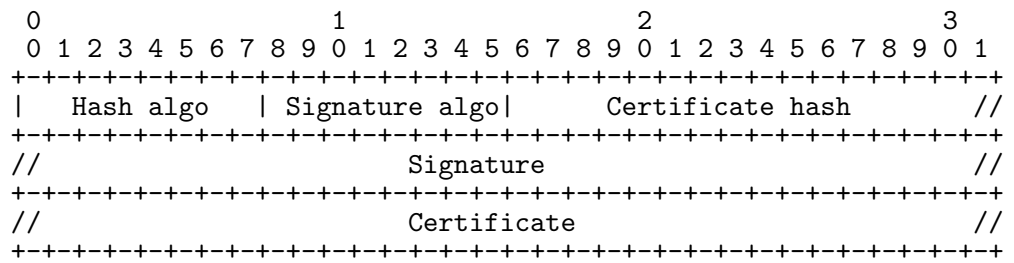
Hash func (8-bits): The hash function used. The codes for the hash functions are defined in Table A.3.

Hash length (16-bits): The length of the hash function output.

Password (variable): The cryptographic hash of the password and 64-bit random number.

## A.14 Signature

The signature computed over the information.




Hash algo (8-bits): The hash function used to compute the CertificateHash of the certificate and to computer the hash over the data. The hash algorithm is defined in IANA TLS HashAlgorithm registry [IANA, 2010]. The input to the hash function includes the data to be signed, Hash Algo, Signature Algo, and Certificate Hash bit-fields.

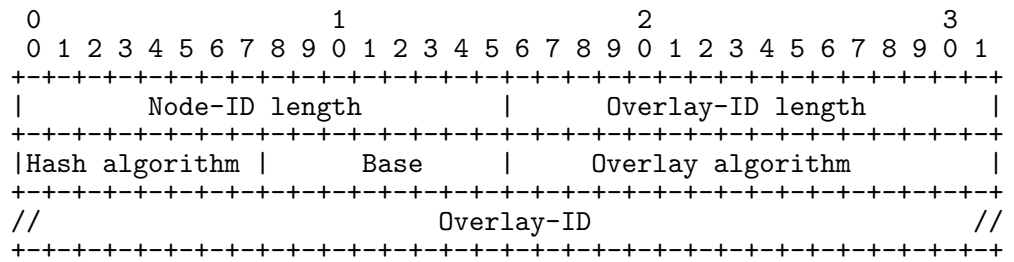
Signature algo (8-bits): The signature algorithm to compute the signature over the hash of data. It is defined in IANA TLS SignatureAlgorithm registry [IANA, 2010].

Certificate hash (variable): The hash of the node certificate as defined in the certificate object.

Signature (variable): The cryptographic signature. It is computed as: Signature(Data, Hash Algo, Signature Algo, H(Certificate))

## A.15 P2P-Options

This TLV object defines the options associated with the overlay, including the overlay identifier and the overlay algorithm being used.




Node-ID length (16 bits): The length of the node-id.

Overlay-ID length (16 bits): The length of the identifier for this overlay.

Hash algorithm (8 bits): An identifier for the hash algorithm being used. The hash algorithm is used in structured overlay algorithms such as DHTs.

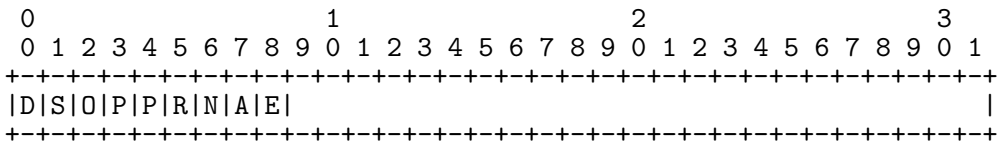
Base (8 bits): The base used in DHTs. It is set to zero for unstructured overlays.

Overlay algorithm (16 bits): An identifier for the overlay algorithm run by peers in this overlay. The list of overlay algorithms is defined in Table A.4.

Overlay-ID (variable): The identifier of the overlay as determined by the overlay operator or by the first user of the overlay. The example of former is "overlay.domain.com"; the latter's example is "bob's overlay".

## A.16 Request-Options

A fixed 32-bit options for identifying the various options in the request.




D (1 bit): If set (D=1), the node sending the request prefers to receive the response directly. This flag is only checked, if the R flag (recursive routing) is set.

S (1 bit): If set (S=1), the request is being sent to the immediate neighbors of the newly joining peer. The request must be a Join request.

O (1 bit): If set (O=1), and if R flag is also set, the nodes in the path of the request should insert their node-info objects in the record-route TLV object.

P (1 bit): If set (P=1), designate one copy as primary for parallel lookups.

R (1 bit) request-routing-table: If set (R=1), send a copy of the routing table to the peer issuing the request either in a response or in a separate ExchangeTable request. The

transmission of the routing-table copy is governed by the in-separate-request (E flag) and the partial-copy (A flag) flags.

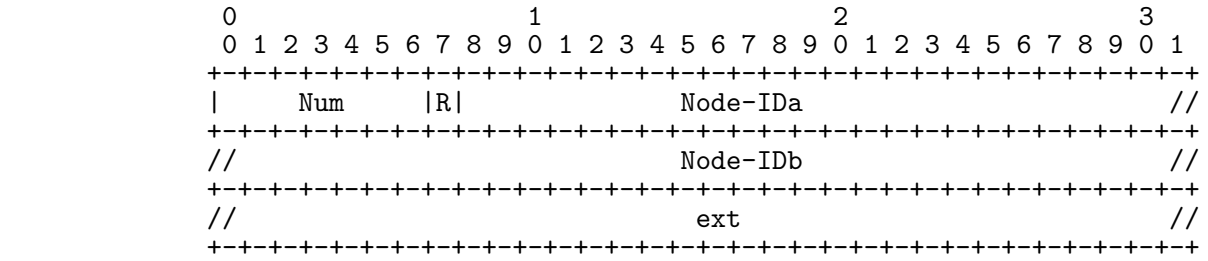
N (1 bit) request-neighbor-table: If set (N=1), send a copy of the neighbor table to the peer issuing the request in a response or ExchangeTable request. The transmission of routing-table copy is governed by the in-separate-request (E flag) and the partial-copy flags.

A (1 bit) partial-reply for routing or neighbor table: If set (A=1), the peer generating the definite response sends a copy of the routing or neighbor table as determined by the P and N flags in its response as permitted by the UDP MTU size. If E (in-separate-request) is also set, the rest of the routing or neighbor table is sent in a separate ExchangeTable request.

E (1 bit) in-separate-request: If set (E=1), the node sending the request desires to receive a copy of the routing table of a node receiving the request in a separate ExchangeTable request.

## A.17 PLookup

The plookup object specifies the peers a node is searching for in order to maintain connectivity with the p2p network. For structured overlays, a node may search for peers with node-IDs that lie within a range. For unstructured overlays, the node-id's can be replaced with any appropriate metric such as free CPU or spare network capacity. Each overlay algorithm can override this object.



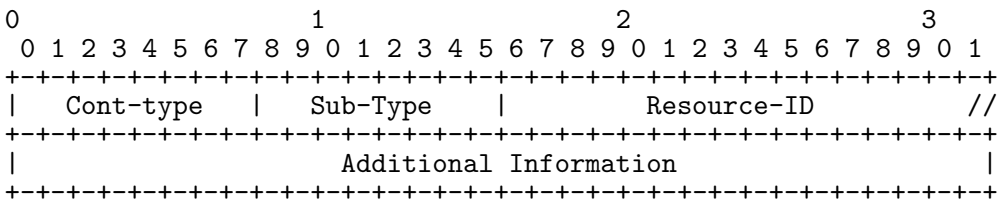

Num (7 bits): Number of peers to look for.

R (1 bit): If set (R=1), then it is a range lookup.

Node-IDa (variable): Node-ID. The node-id specified by the peer sending the LookupPeer request in order to find peers with a node-id that is closest to the specified id. It is applicable only in DHTs.

Node-IDb (variable): Node-ID. The node-id specified by the peer sending the LookupPeer request. It is only specified in structured overlay algorithms when the id of node being searched must lie within a range.

## A.18 RLookup



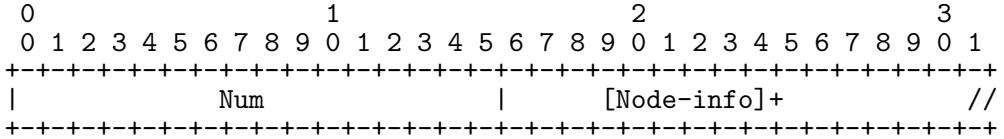

Cont-type (8 bits): An identifier for the type of content contained in this resource-object. Table A.5 specifies the identifiers for the defined content types.

Sub-type (8 bits): An identifier which further classifies the content type as defined by cont-type.

Resource-ID (variable): The resource-id object that identifies the resource being searched for.

## A.19 Routing-table

The routing-table TLV object contains the list of peers in the routing table of a peer. Each peer is encoded as a node-info object.

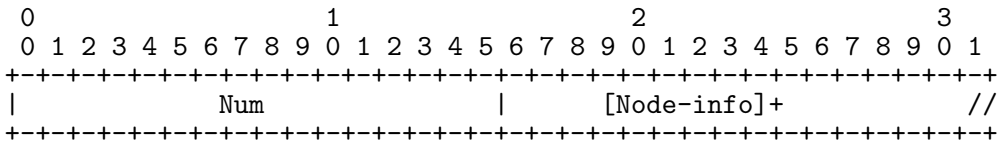


Num (16 bits): The number of node-info objects in the table.

Node-info (variable): One or more node-info objects, each representing an individual entry in the routing table of a peer.

## A.20 Neighbor-table

The Neighbor-table TLV object contains the list of peers in the neighbor table of a peer. Each peer is encoded as a Node-info object.

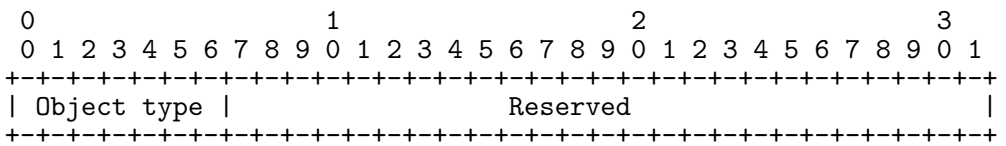



Num (16 bits): The number of node-info objects in the table.

Node-info (variable): One or more node-info objects, each representing an individual entry in the routing table of a peer.

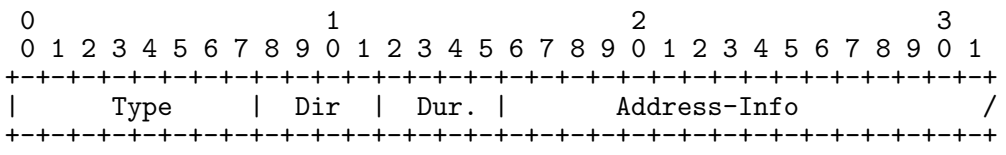
## A.21 Object-Req

The object-req object describes the type of the object that a node should return in response to a GetDiagnostics request (Section 3.6.5.1).




## A.22 BWTest

The bwtest object defines the bandwidth test that nodes run to measure their uplink and downlink capacity.




Type (8-bits): The type of the bandwidth test. A value of zero means that nodes run the TCP throughput measurement test, whereas a value of one means that nodes run the PathChar [Jacobson, 2010] test.

Dir (4-bits): The direction of the test. The node sending the request specifies if it wants to measure its uplink or downlink capacity or both. A value of 1 means uplink only, 2 downlink only, and 3 both uplink and downlink.

Dur (4-bits): The duration of the test.

Address-Info: The address-info TLV object contains the host and server reflexive port numbers on which a node listens for performing a bandwidth test.

## A.23 Message and Object Identifiers

This section defines the identifiers for the messages, TLV objects, and other identifiers defined and used by the Peer-to-Peer Protocol.

| Message | ID | Message | ID |
|---|---|---|---|
| Enroll | 0 | PublishObject | 9 |
| Bootstrap | 2 | LookupObject | 10 |
| Join | 3 | ReplicateObject | 12 |
| Leave | 4 | TransferObject | 13 |
| KeepAlive | 5 | Tunnel | 14 |
| LookupPeer | 6 | MeasureBandwidth | 17 |
| ExchangeTable | 7 | | |

Table A.1: Message identifiers.

| Object Type | ID | Object Type | ID |
|---|---|---|---|
| Node-ID | 0 | Signature | 9 |
| Node-Info | 1 | P2P-Options | 10 |
| Address-Info | 2 | Request-Options | 11 |
| Resource-ID | 3 | PLookup | 12 |
| Resource-Object | 4 | RLookup | 13 |
| Expires | 5 | Routing-table | 14 |
| Owner | 6 | Neighbor-table | 15 |
| Certificate | 7 | BWTest | 16 |
| Certificate-Sign-Request | 8 | | |

Table A.2: Object identifiers.

| Hash algorithm | ID |
|---|---|
| None | 0 |
| SHA1 | 1 |
| SHA-256 | 2 |
| SHA-512 | 3 |
| MD4 | 4 |
| MD5 | 5 |

Table A.3: Hash algorithms

| Overlay algorithm | ID |
|---|---|
| Chord | 0 |
| CAN | 1 |
| Kademlia | 2 |
| Pastry | 3 |
| Bamboo | 4 |

Table A.4: Overlay algorithms

| Content-type | ID |
|---|---|
| SIP-CONTACT | 0 |
| STUN-TURN | 1 |

Table A.5: Content-type of resource-object

| Component-ID | ID |
|---|---|
| RTP | 0 |
| RTCP | 1 |
| SIP | 2 |
| P2PP | 3 |

Table A.6: Component-ID

# Appendix B

# P2PP Transport Layer

This section defines mechanisms for reliably delivering a message to the next hop. The reliable delivery of a message constitutes a **transaction**. For requests, a transaction consists of a single request followed by acknowledgements (ACKs) if any, and a response. For responses, reponseACKs, and indications, a transaction consists of single response, responseACK, or indication, followed by an ACK if an unreliable transport is used. Section 3.4.4 discussed the need for acknowledgements and responseACKs.

A transaction is identified by a source-ID, transaction-ID, and a transaction-type triple. This triple is used to match acknowledgements (ACK) to requests, responses and indications. However, the responses and responseACKs are matched to requests using only source-ID and transaction-ID tuple. A transaction can be of four types, namely, request, response, responseACK, or an indication. The source-ID, transaction-ID, and transaction-type must be preserved in the ACKs.

## B.1 Transaction State Machine

This section defines the transaction state machine for unreliable and reliable transports.

### B.1.1 State Machine for Unreliable Transports

For unreliable transports, the transaction state machine for requests is shown in Figure B.1 and state machine for responses and indications is shown in Figure B.2.

The "Trans_Msg" (abbreviation for transmit message) state is entered when a node sends a request, response, responseACK, or an indication. When entering this state, the transaction should set timer T1 and T2. Timer T1 governs retransmissions and is updated after $i^{th}$ retransmission as follows: $T1 = 2^i * T0$. If the timer T2 fires, the state machine transitions to "Failed" state and is terminated.

If a request was sent and an acknowledgement was received, the state machine transitions to "Wait_Resp" state. When entering this state, the transaction should set timer T3. If timer T3 fires, the state machine transitions to "Failed" state and is terminated. If a responseACK was received, the transaction sends an ACK and is terminated.

If a response, responseACK, or an indication was sent and ACK was received, the state-machine immediately transitions to the "Terminated" state.

### B.1.2 State Machine for Reliable Transports

For reliable transports, the transaction state machine for requests is shown in Figure B.3; the state machine for responses and indications is shown in Figure B.3.

The "Trans_Msg" state is entered when a peer sends or forwards the request, response, or an indication. If the message was a response or an indication and was successfully sent, the state machine transitions to the "Terminated" state. If the message was a request and was successfully sent, the transaction sets Timer T3 and transitions to the "Wait_Resp" state. If a response is received, the transaction is terminated. No ACK is sent for requests, responses, and indications sent over reliable transport. Similarly, no responseACK is generated for a request.

If the request was forwarded in a recursive manner, the application must not terminate the reliable-transport connection. If the request was forwarded in an iterative manner, an application may terminate the reliable transport connection if it does not anticipate its reuse.

### B.1.3 Timers

This section defines timers and their default values for message state machines.

```
Timer    Value
```

```
----------------------
T0        500 ms
T1        T0
T2        5s
T3        5s
```

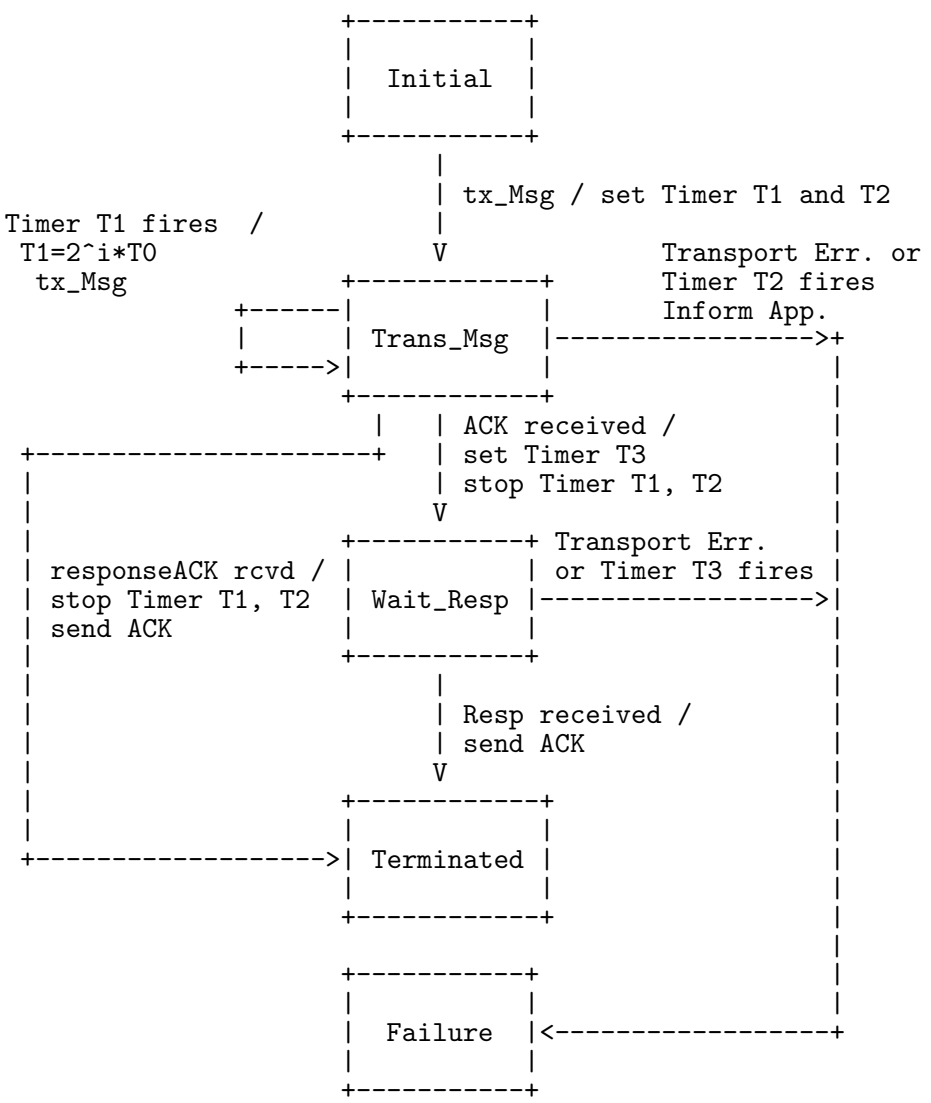



Figure B.1: Transaction state machine for requests sent over an unreliable transport protocol.

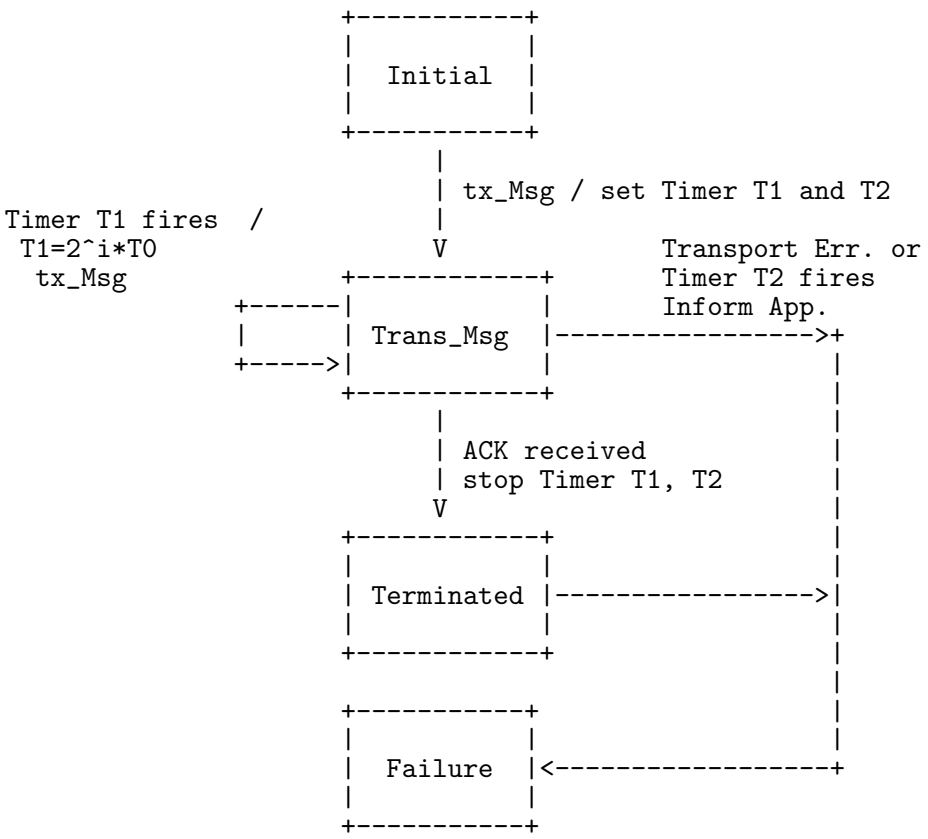



Figure B.2: Transaction state machine for responses and indications sent over an unreliable transport protocol.

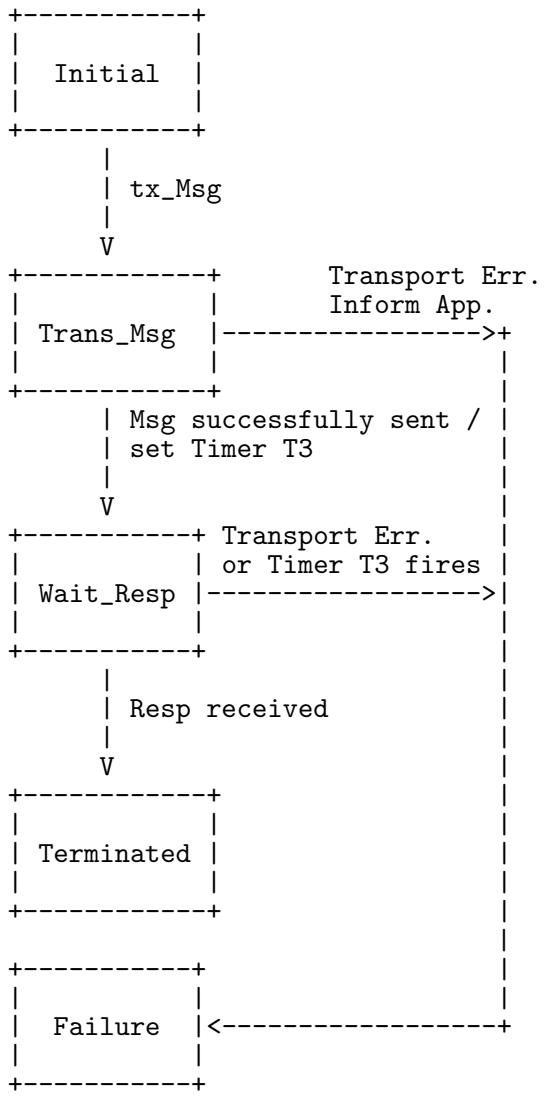



Figure B.3: Transaction state machine for requests sent over a reliable transport protocol.

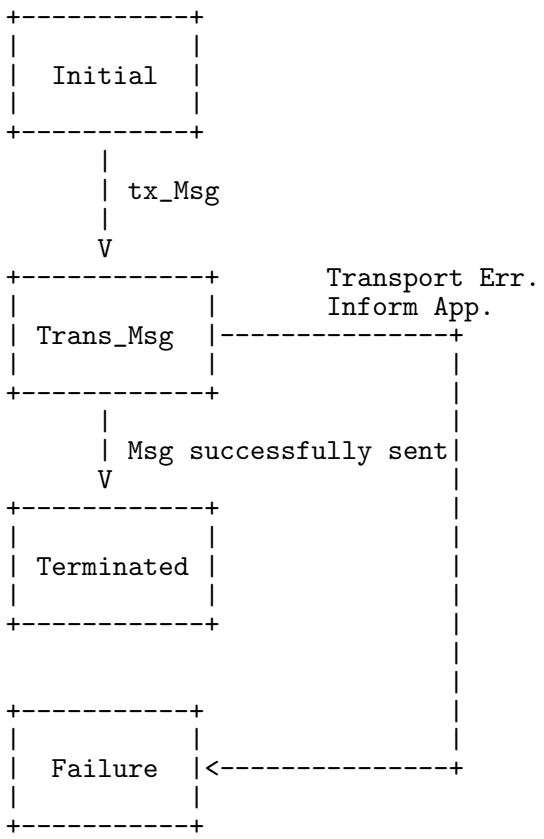



Figure B.4: Transaction state machine for responses and indications sent over for a reliable transport protocol.

# Part VI

# Bibliography

# Bibliography

[Allman *et al.*, 2002] Mark Allman, Sally Floyd, and Craig Patridge. Increasing TCP's Initial Window. RFC 3390, October 2002.

[Allman *et al.*, 2009] Mark Allman, Vern Paxson, and Ethan Blanton. TCP Congestion Control. RFC 5681, September 2009.

[Allman, 2003] Mark Allman. TCP Congestion Control with Appropriate Byte Counting (ABC). RFC 3465, February 2003.

[Audet and Jennings, 2007] Francois Audet and Cullen Jennings. Network Address Translation (NAT) Behavioral Requirements for Unicast UDP. RFC 4787, January 2007.

[AutoIt, 2010] AutoIt. `http://www.autoitscript.com/`, 2010. [Online; accessed June 2010].

[Banerjee *et al.*, 2002] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable Application Layer Multicast. In *Proc. of SIGCOMM*, Pittsburgh, Pennsylvania, USA, August 2002.

[Banerjee *et al.*, 2006] Suman Banerjee, Christopher Kommareddy, Koushik Kar, Bobby Bhattacharjee, and Samir Khuller. OMNI: An Efficient Overlay Multicast Infrastructure for Real-time Applications. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 50(6):826–841, 2006.

[Baset and Schulzrinne, 2006] Salman A. Baset and Henning Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In *Proc. of IEEE INFOCOM*, Barcelona, Spain, April 2006.

[Baset and Schulzrinne, 2008] Salman A. Baset and Henning Schulzrinne. OpenVoIP: An Open Peer-to-Peer VoIP and IM System. In *Proc. of SIGCOMM (demo)*, Seattle, WA, USA, September 2008.

[Baset and Schulzrinne, 2010] Salman A. Baset and Henning Schulzrinne. Reliablity and Relay Selection in Peer-to-Peer Communication Systems. In *Proc. of IPTCOMM*, Munich, Germany, August 2010.

[Baset *et al.*, 2007] Salman A. Baset, Henning Schulzrinne, and Marcin Matuszewski. Peer-to-Peer Protocol (P2PP). Internet-draft (work-in-progress), IETF, November 2007.

[Baset *et al.*, 2010] Salman A. Baset, Joshua Reich, Jan Janak, Paval Kaparek, Vishal Misra, Dan Rubenstein, and Henning Schulzrinne. How Green is IP-Telephony? In *Proc. of SIGCOMM Green Networking Workshop*, New Delhi, India, August 2010.

[Birolini, 2004] Alessandro Birolini. *Reliability Engineering: Theory and Practice*. Springer-Verlag, New York, NY, USA, 2004.

[BitTorrent, 2010] BitTorrent. `http://www.bittorrent.com/`, 2010. [Online; accessed June 2010].

[Brosh *et al.*, 2008] Eli Brosh, Salman A. Baset, Vishal Misra, Dan Rubenstein, and Henning Schulzrinne. The Delay-Friendliness of TCP. In *Proc. of SIGMETRICS*, Annapolis, MD, USA, June 2008.

[Bryan *et al.*, 2005] David A. Bryan, Bruce Lowekamp, and Cullen Jennings. SOSIMPLE: A Serverless, Standards-based, P2P SIP Communication System. In *Proc. of the First International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications*, Orlando, FL, USA, July 2005.

[Bryan *et al.*, 2010] David Bryan, Philip Matthews, Eunsoo Shim, Dean Willis, and Spencer Dawkings. Concepts and Terminology for Peer-to-Peer SIP. Internet draft (work-in-progress), October 2010.

[Castro *et al.*, 2003] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. SplitStream: High-Bandwidth Multicast in Cooperative Environments. In *Proc. of SOSP*, Bolton Landing, NY, USA, October 2003.

[Chakravarty *et al.*, 2008] Sambuddho Chakravarty, Angelos Stavrou, and Angelos Keromytis. LinkWidth: A Method to Measure Link Capacity and Available Bandwidth using Single-End Probes. Technical Report (cucs-002-08), Department of Computer Science, Columbia University, January 2008.

[Chawathe *et al.*, 2003] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making Gnutella-like P2P Systems Scalable. In *Proc. of SIGCOMM*, Karlsruhe, Germany, August 2003.

[Chen *et al.*, 2006] Kuan-Ta Chen, Chun-Ying Huang, Polly Huang, and Chin-Laung Lei. An Empirical Evaluation of TCP Performance in Online Games. In *Proc. of ACM SIGCHI*, Montral, Canada, April 2006.

[Chen *et al.*, 2008] Minghua Chen, Miroslav Ponec, Sudipta Sengupta, Jin Li, and Philip A. Chou. Utility Maximization in Peer-to-Peer Systems. In *Proc. of SIGMETRICS*, Annapolis, Maryland, USA, June 2008.

[Christensen *et al.*, 2010] Ken Christensen, Pedro Reviriego, Bruce Nordman, Michael Bennett, Mehrgan Mostowfi, and Juan Antonio Maestro. IEEE 802.3az: The Road to Energy Efficient Ethernet. *Communications Magazine, IEEE*, 48(11):50 –56, November 2010.

[Chu *et al.*, 2000] Yang-hua Chu, Sanjay G. Rao, and Hui Zhang. A Case for End System Multicast. In *Proc. of SIGMETRICS*, Santa Clara, California, USA, June 2000.

[Chu *et al.*, 2001] Yang Chu, Sanjay Rao, Srinivasan Seshan, and Hui Zhang. Enabling Conferencing Applications on the Internet Using an Overlay Multicast Architecture. In *Proc. of SIGCOMM*, San Diego, CA, USA, August 2001.

[Cisco, 2010a] Cisco. Cisco Catalyst 2960-S and 2960 Series Switches with LAN Base Software. `http://www.cisco.com/en/US/prod/collateral/switches/ps5718/ps6406/product_data_sheet0900aecd80322c0c.html`, 2010. [Online; accessed November 2010].

[Cisco, 2010b] Cisco. Cisco Unified Communications Manager (CallManager). `http://www.cisco.com/en/US/products/sw/voicesw/ps556/index.html`, 2010. [Online; accessed November 2010].

[Cooper *et al.*, 2007] Eric Cooper, Alan Johnston, and Philip Matthews. Bootstrap Mechanisms for P2PSIP. Internet draft (work-in-progress), February 2007.

[Dabek *et al.*, 2003] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *Proc. of IPTPS*, Berkeley, CA, February 2003.

[Dell, 2010] Dell. Dell PowerEdge 1900 Server. `http://www.dell.com/downloads/emea/products/pedge/en/PE1900_Spec_Sheet_Quad.pdf`, 2010. [Online; accessed June 2010].

[Dessent, 2010] Brian Dessent. BitTorrent FAQ and Guide, What ports does BitTorrent use? Will it work with a firewall/NAT? `http://dessent.net/btfaq/#ports`, 2010. [Online; accessed June 2010].

[Dierks and Rescorla, 2008] Tim Dierks and Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008.

[Dischinger *et al.*, 2007] Marcel Dischinger, Andreas Haeberlen, Krishna P. Gummadi, and Stefan Saroiu. Characterizing Residential Broadband Networks. In *Proc. of IMC*, San Diego, CA, USA, October 2007.

[Dobrescu *et al.*, 2009] Mihai Dobrescu, Norbert Egi, Katerina Argyraki, Byung-Gon Chun, Kevin Fall, Gianluca Iannaccone, Allan Knies, Maziar Manesh, and Sylvia Ratnasamy. RouteBricks: Exploiting Parallelism To Scale Software Routers. In *Proc. of SOSP*, Big Sky, MT, USA, October 2009.

[Douceur, 2002] John R. Douceur. The Sybil Attack. In *Proc. of IPTPS*, Cambridge, MA, USA, 2002.

[Egevang and Francis, 1994] Kjeld Borch Egevang and Paul Francis. The IP Network Address Translator (NAT). RFC 1631, May 1994.

[Ethereal, 2010] Ethereal. `http://www.ethereal.com`, 2010. [Online; accessed June 2010].

[Floyd and Kohler, 2007] Sally Floyd and Eddie Kohler. TCP Friendly Rate Control (TFRC): The Small-Packet (SP) Variant. RFC 4828 (Experimental), April 2007.

[Ford *et al.*, 2005] Bryan Ford, Pyda Srisuresh, and Dan Kegel. Peer-to-Peer Communication Across Network Address Translators. In *Proc. of USENIX Annual Technical Conference*, Anaheim, CA, USA, April 2005.

[GNU C Library, 2010] GNU C Library. `http://www.gnu.org/software/libc/`, 2010. [Online; accessed June 2010].

[Godfrey *et al.*, 2006] P. Brighten Godfrey, Scott Shenker, and Ion Stoica. Minimizing Churn in Distributed Systems. In *Proc. of SIGCOMM*, Pisa, Italy, August 2006.

[Goel *et al.*, 2002] Ashvin Goel, Charles Krasic, Kang Li, and Jonathan Walpole. Supporting Low-Latency TCP Based Media Streams. In *Proc. of IWQoS*, Miami, Florida, USA, May 2002.

[Google, 2010] Google. Google Talk. `http://www.google.com/talk/`, 2010. [Online; accessed June 2010].

[Goyal, 2001] V.K. Goyal. Multiple Description Coding: Compression Meets the Network. *IEEE Signal Processing Magazine*, 18(5):74–93, Sep 2001.

[Guha *et al.*, 2006] Saikat Guha, Neil Daswani, and Ravi Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *Proc. of IPTPS*, Santa Barbara, CA, USA, February 2006.

[Gummadi *et al.*, 2002] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: Estimating Latency Between Arbitrary Internet End Hosts. *SIGCOMM Comput. Commun. Rev.*, 32(3):11–11, 2002.

[Guo *et al.*, 2006] Lei Guo, Enhua Tan, Songqing Chen, Zhen Xiao, Oliver Spatscheck, and Xiaodong Zhang. Delving into Internet Streaming Media Delivery: A Quality and Resource Utilization Perspective. In *Proc. of IMC*, Rio de Janeiro, Brazil, October 2006.

[Handley *et al.*, 2000] Mark Handley, Jitendra Padhye, and Sally Floyd. TCP Congestion Window Validation. RFC 2861, June 2000.

[Handley *et al.*, 2003] Mark Handley, Sally Floyd, Jitendra Padhye, and Joerg Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 3448, January 2003.

[IANA, 2010] IANA. Transport Layer Security (TLS) Parameters. `http://www.iana.org/assignments/tls-parameters/tls-parameters.xml`, 2010. [Online; accessed June 2010].

[IBM, 2010] IBM. IBM BladeCenter. `http://ibm.com/systems/bladecenter/`, 2010. [Online; accessed June 2010].

[International Telecommunication Union (ITU), 1997] International Telecommunication Union (ITU). Series X: Data Networks and Open System Communications; Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). Recommendation X.690, Telecommunication Standardization Sector of ITU, December 1997.

[International Telecommunication Union (ITU), 2003] International Telecommunication Union (ITU). Series G: Transmission Systems and Media, General Recommendation on the Transmission Quality for an Entire International Telephone Connection; One-way Transmission Time. Recommendation G.114, Telecommunication Standardization Sector of ITU, May 2003.

[International Telecommunication Union (ITU), 2006] International Telecommunication Union (ITU). Series X: Data Networks, Information Technology Open System Communication and Security; Information Technology - Open Systems Interconnection - The Directory: Public-key and Attribute Certificate Frameworks. Recommendation X.509, Telecommunication Standardization Sector of ITU, May 2006.

[iptel.org, 2010] iptel.org. SIP Express Media Server (SEMS). `http://www.iptel.org/sems`, 2010. [Online; accessed June 2010].

[iptrtpproxy, 2010] iptrtpproxy. `http://www.2p.cz/en/netfilter_rtp_proxy/iptrtpproxy`, 2010. [Online; accessed June 2010].

[Jacobson, 2010] Van Jacobson. PathChar. `http://www.caida.org/tools/utilities/others/pathchar/`, 2010. [Online; accessed June 2010].

[Jennings *et al.*, 2010] Cullen Jennings, Bruce Lowekamp, Eric Rescorla, Salman A. Baset, and Henning Schulzrinne. REsource LOcation And Discovery (RELOAD) Base Protocol. Internet draft (work-in-progress), November 2010.

[Kho *et al.*, 2008] Wookyun Kho, Salman Abdul Baset, and Henning Schulzrinne. Skype Relay Calls: Measurements and Experiments. In *Proc. of IEEE Global Internet Symposium*, Phoenix, AZ, USA, April 2008.

[Kohler *et al.*, 2006] Eddie Kohler, Mark Handley, and Sally Floyd. Designing DCCP: Congestion Control Without Reliability. In *Proc. of SIGCOMM*, Pisa, Italy, September 2006.

[Kondo, 2010] Kuniaki Kondo. AS Number Lookup Utility. `http://aslookup.bgpview.org/index-e.html`, 2010. [Online; accessed June 2010].

[Lazzaro, 2006] John Lazzaro. Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport. RFC 4571, July 2006.

[Lennox and Schulzrinne, 2003] Jonathan Lennox and Henning Schulzrinne. A Protocol for Reliable Decentralized Conferencing. In *Proc. of NOSSDAV*, Monterey, CA, USA, June 2003.

[Leonard *et al.*, 2005] Derek Leonard, Vivek Rai, and Dmitri Loguinov. On Lifetime-based Node Failure and Stochastic Resilience of Decentralized Peer-to-Peer Networks. In *Proc. of SIGMETRICS*, Banf, Alberta, Canada, June 2005.

[Liang *et al.*, 2004] J. Liang, R. Kumar, and K. Ross. Understanding Kazaa, 2004.

[Luo *et al.*, 2007] Chong Luo, Wei Wang, Jian Tang, Jun Sun, and Jiang Li. A Multiparty Videoconferencing System Over an Application-Level Multicast Protocol. *IEEE Transactions on Multimedia*, 9(8):1621–1632, December 2007.

[MacDonald and Lowekamp, 2010] Derek C. MacDonald and Bruce Lowekamp. NAT Behavior Discovery Using STUN. RFC 5780, May 2010.

[magicJack, 2010] magicJack. `http://www.magicjack.com/6/index.asp`, 2010. [Online; accessed November 2010].

[Mahy *et al.*, 2010] Rohan Mahy, Philip Matthews, and Jonathan Rosenberg. Traversal Using Relays around NAT (TURN). RFC 5766, April 2010.

[Maxmind, 2010] Maxmind. `http://www.maxmind.com/`, 2010. [Online; accessed June 2010].

[Maymounkov and Mazieres, 2002] Petar Maymounkov and David Mazieres. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Proc. of IPTPS*, Cambridge, MA, USA, March 2002.

[McCreary *et al.*, 2005] Doug McCreary, Kang Li, Scott A. Watterson, and David K. Lowenthal. TCP-RC: A Receiver-Centered TCP Protocol for Delay-Sensitive Applications. In *Proc. of MMCN*, San Jose, CA, USA, January 2005.

[McKenney *et al.*, 2010] Paul E. McKenney, Dan Y. Lee, and Barbara A. Denny. Traffic Generator Software. `http://www.postel.org/tg/tg.html`, 2010. [Online; accessed June 2010].

[Medina *et al.*, 2005] Alberto Medina, Mark Allman, and Sally Floyd. Measuring the Evolution of Transport Protocols in the Internet. *SIGCOMM Comput. Commun. Rev.*, 35(2):37–52, April 2005.

[Microsoft, 2010] Microsoft. Windows Live Messenger. `http://explore.live.com/windows-live-messenger?os=winxp`, 2010. [Online; accessed June 2010].

[Mondal and Kuzmanovic, 2007] Amit Mondal and Aleksandar Kuzmanovic. When TCP Friendliness Becomes Harmful. In *Proc. of IEEE INFOCOM*, Anchorage, AK, USA, May 2007.

[Mukherjee and Brecht, 2000] Biswaroop Mukherjee and Tim Brecht. Time-lined TCP for the TCP-friendly Delivery of Streaming Media. In *Proc. of IEEE ICNP*, Osaka, Japan, November 2000.

[Müller and Klenk, 2010] Andreas Müller and Andreas Klenk. NAT tester. `http://nattest.net.in.tum.de/`, 2010. [Online; accessed June 2010].

[Na and Yoo, 2002] Songun Na and Seungwha Yoo. Allowable Propagation Delay for VoIP Calls of Acceptable Quality. In *Proc. of AISA*, London, UK, August 2002.

[NationalNet, 2010] NationalNet. Dedicated Hosting. `http://www.nationalnet.com/dedicated_hosting.html`, 2010. [Online; accessed June 2010].

[Nedevschi *et al.*, 2008] Sergiu Nedevschi, Jitendra Padhye, and Sylvia Ratnasamy. Hot Data Centers vs. Cool Peers. In *Proc. of HotPower workshop co-located with USENIX OSDI Symposium*, San Diego, CA, USA, December 2008.

[Net Peeker, 2010] Net Peeker. `http://www.net-peeker.com`, 2010. [Online; accessed June 2010].

[NIST Net, 2010] NIST Net. `http://www-x.antd.nist.gov/nistnet/`, 2010. [Online; accessed June 2010].

[Nystrom and Kaliski, 2000] Magnus Nystrom and Burt Kaliski. PKCS #10: Certification Request Syntax Specification Version 1.7. RFC 2986, November 2000.

[ooVoo, 2010] ooVoo. `http://www.oovoo.com/`, 2010. [Online; accessed June 2010].

[OpenWengo, 2010] OpenWengo. `http://www.openwengo.org/`, 2010. [Online; accessed June 2010].

[PlanetLab, 2010] PlanetLab. `http://www.planet-lab.org/`, 2010. [Online; accessed June 2010].

[Polycom Conferencing Infrastructure., 2010] Polycom Conferencing Infrastructure. `http://www.polycom.com/products/telepresence_video/conferencing_infrastructure/index.html`, 2010. [Online; accessed June 2010].

[Ponec *et al.*, 2009] Miroslav Ponec, Sudipta Sengupta, Minghua Chen, Jin Li, and Philip A. Chou. Multi-rate Peer-to-Peer Video Conferencing: A Distributed Approach using Scalable Coding. In *Proc. of ICME*, New York, NY, USA, June 2009.

[Power over Ethernet (PoE), 2010] Power over Ethernet (PoE). `http://en.wikipedia.org/wiki/Power_over_Ethernet`, 2010. [Online; accessed November 2010].

[Qutecom, 2010] Qutecom. `http://www.qutecom.org/`, 2010. [Online; accessed June 2010].

[Ratnasamy *et al.*, 2001] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A Scalable Content-Addressable Network. In *Proc. of SIGCOMM*, San Diego, CA, USA, August 2001.

[Redelmeier, 2010] Robert Redelmeier. cpuburn. `http://pages.sbcglobal.net/redelm/`, 2010. [Online; accessed June 2010].

[Ren *et al.*, 2006] Shansi Ren, Lei Guo, and Xiaodong Zhang. ASAP: an AS-Aware Peer-Relay Protocol for High Quality VoIP. In *Proc. of ICDCS*, Lisbon, Portugal, July 2006.

[Rescorla and Modadugu, 2006] Eric Rescorla and Nagendra Modadugu. Datagram Transport Layer Security. RFC 4347, April 2006.

[Rhea *et al.*, 2003] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz. Pond: The OceanStore Prototype. In *Proc. of 2nd USENIX Conference on File and Storage Technologies*, San Francisco, CA, 2003.

[Rhea *et al.*, 2004] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling Churn in a DHT. In *Proc. of USENIX Technical Conference*, Anaheim, CA, USA, April 2004.

[Rhea, 2005] Sean Rhea. *OpenDHT: A Public DHT Service*. PhD thesis, University of California at Berkeley, Berkeley, CA, USA, 2005.

[Risson and Moors, 2007] John Risson and Tim Moors. Survey of Research towards Robust Peer-to-Peer Networks: Search Methods. RFC 4981, September 2007.

[Riverbed Technology, 2010] Riverbed Technology. WinDump. `http://www.winpcap.org/windump/install/default.htm`, 2010. [Online; accessed June 2010].

[Roach, 2002] Adam Roach. Session Initiation Protocol (SIP)-Specific Event Notification. RFC 3265, June 2002.

[Rosenberg *et al.*, 2002] Jonathan Rosenberg, Henning Schulzrinne, Gonzalo Camarillo, Alan Johnston, Jon Peterson, Robert Sparks, Mark Handley, and Eve Schooler. SIP: Session Initiation Protocol. RFC 3261, June 2002.

[Rosenberg *et al.*, 2008] Jonathan Rosenberg, Rohan Mahy, Philip Matthews, and Dan Wing. Session Traversal Utilities for NAT (STUN). RFC 5389, October 2008.

[Rosenberg, 2010] Jonathan Rosenberg. Interactive Connectivity Establishment (ICE). RFC 5245, April 2010.

[Rowstron and Druschel, 2001a] Antony Rowstron and Peter Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany, November 2001.

[Rowstron and Druschel, 2001b] Antony Rowstron and Peter Druschel. Storage Management and Caching in PAST, a Large Scale, Persistent Peer-to-Peer Storage Utility. In *Proc. of SOSP*, Chateau Lake Louise, Banff, Canada, October 2001.

[Schulzrinne *et al.*, 2003] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, July 2003.

[SETI@home, 2010] SETI@home. `http://setiathome.ssl.berkeley.edu/`, 2010. [Online; accessed June 2010].

[Shen *et al.*, 2010] Charles Shen, Erich Nahum, Henning Schulzrinne, and C. Wright. The Impact of TLS on SIP Server Performance. In *Proc. of IPTCOMM*, Munich, Germany, August 2010.

[Shi and Turner, 2002] Sherlia Y. Shi and Jonathan S. Turner. Routing in Overlay Multicast Networks. In *Proc. of IEEE INFOCOM*, New York, NY, USA, June 2002.

[Singh, 2006] Kundan Singh. *Reliable, Scalable and Interoperable Internet Telephony.* PhD thesis, Columbia University, New York, NY, USA, 2006.

[SIP Router Project, 2010] SIP Router Project. `http://sip-router.org/`, 2010. [Online; accessed June 2010].

[SIPp, 2010] SIPp. `http://sipp.sourceforge.net/`, 2010. [Online; accessed June 2010].

[Skype, 2010a] Skype. `http://www.skype.com/`, 2010. [Online; accessed June 2010].

[Skype, 2010b] Skype. P2P Telephony Explained – For Geeks Only. `http://www.skype.com/help/guides/p2pexplained/`, 2010. [Online; accessed June 2010].

[Skype, 2010c] Skype. SILK: Super Wideband Audio Codec. `http://developer.skype.com/silk/`, 2010. [Online; accessed June 2010].

[Skype, 2010d] Skype. Skype API. `http://developer.skype.com/accessories`, 2010. [Online; accessed June 2010].

[Skype, 2010e] Skype. SkypeIn. `http://www.skype.com/allfeatures/onlinenumber/`, 2010. [Online; accessed June 2010].

[Skype, 2010f] Skype. SkypeOut. `http://www.skype.com/intl/en-us/features/allfeatures/call-phones-and-mobiles/`, 2010. [Online; accessed June 2010].

[Srisuresh *et al.*, 2008] Pyda Srisuresh, Bryan Ford, and Dan Kegel. State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs). RFC 5128, March 2008.

[Stevens, 1994] W. Richard Stevens. *TCP/IP Illustrated*, volume 1. Addison-Wesley, MA, November 1994.

[Stoica *et al.*, 2003] Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. *IEEE/ACM Transactions on Networking*, 11:17–32, February 2003.

[Suh *et al.*, 2006] K. Suh, D. R. Figueiredo, J. Kurose, and D. Towsley. Characterizing and Detecting Relayed Traffic: A Case Study using Skype. In *Proc. of IEEE INFOCOM*, Barcelona, Spain, April 2006.

[Tan and Jarvis, 2007] Guang Tan and Stephen A. Jarvis. Stochastic Analysis and Improvement of the Reliability of DHT-Based Multicast. In *Proc. of IEEE INFOCOM*, Anchorage, AK, USA, May 2007.

[Tandberg, 2010] Tandberg. TelePresence Conferencing Infrastructure. `http://www.tandberg.com/video-conferencing-multipoint-control.jsp`, 2010. [Online; accessed June 2010].

[UPnP Forum, 2010] UPnP Forum. Internet Gateway Device (IGD) V 2.0. `http://upnp.org/specs/gw/UPnP-gw-InternetGatewayDevice-v2-Device.pdf`, 2010.

[U.S. Energy Information Administration, Independent Statistics and Analysis, 2010] U.S. Energy Information Administration, Independent Statistics and Analysis. Average Retail Price of Electricity to Ultimate Customers by End-Use Sector, by State. `http://www.eia.doe.gov/electricity/epm/table5_6_a.html`, 2010. [Online; accessed November 2010].

[Valancius *et al.*, 2009] Vytautas Valancius, Nikolaos Laoutaris, Laurent Massoulie, Christophe Diot, and Pablo Rodriguez. Greening the Internet with Nano Data Centers. In *Proc. of CoNEXT*, Rome, Italy, December 2009.

[Vonage, 2010] Vonage. `http://www.vonage.com/`, 2010. [Online; accessed June 2010].

[Wake-on-LAN, 2010] Wake-on-LAN. `http://en.wikipedia.org/wiki/Wake-on-LAN`, 2010. [Online; accessed June 2010].

[Wang and Ramchandran, 2008] Jiajun Wang and K. Ramchandran. Enhancing Peer-to-Peer Live Multicast Quality Using Helpers. In *15th IEEE International Conference on Image Processing (ICIP)*, San Diego, CA, USA, October 2008.

[Wang *et al.*, 2004] Bing Wang, Jim Kurose, Prashant Shenoy, and Don Towsley. Multimedia Streaming via TCP: An Analytic Performance Study. In *Proc. of ACM Multimedia*, New York, NY, USA, October 2004.

[Wang *et al.*, 2007] Bing Wang, Wei Wei, and Don Towsley. Multipath Live Streaming via TCP: Scheme, Performance and Benefits. In *Proc. of CoNEXT*, New York, NY, USA, December 2007.

[Wang *et al.*, 2009] Xiaoming Wang, Zhongmei Yao, and Dmitri Loguinov. Residual-Based Estimation of Peer and Link Lifetimes in P2P Networks. *IEEE/ACM Transactions on Networking*, 17(3):726–739, 2009.

[Watts up?, 2010] Watts up? .NET Power Meter. `https://www.wattsupmeters.com/secure/products.php?pn=0&wai=285&spec=2`, 2010. [Online; accessed June 2010].

[Zaliapin *et al.*, 2005] I. V. Zaliapin, Y. Y. Kagan, and F. P. Schoenberg. Approximating the Distribution of Pareto Sums. *Pure and Applied Geophysics*, 162:1187–1228, May 2005.

[Zaroliagis, 2005] Christos Zaroliagis. Recent Advances in Multiobjective Optimization. In *Lecture Notes in Computer Science*, volume 3777, pages 45–47. Springer Berlin / Heidelberg, 2005.

[Zhang *et al.*, 2005] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Tak-Shing Peter Yum. CoolStreaming/DONet: A Data-driven Overlay Network for Peer-to-Peer Live Media Streaming. In *Proc. of IEEE INFOCOM*, Miami, FL, USA, March 2005.