Advancing Multimedia: Application Sharing, Latency Measurements and User-Created Services

Omer Boyaci

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

COLUMBIA UNIVERSITY

2012
ABSTRACT

Advancing Multimedia: Application Sharing, Latency Measurements and User-Created Services

Omer Boyaci

Online collaboration tools exist and have been used since the early days of the Internet. Asynchronous tools such as wikis and discussion boards and real-time tools such as instant messaging and voice conferencing have been the only viable collaboration solutions up until recently, due to the low bandwidth between participants. With the increasing bandwidth in computer networks, multimedia collaboration such as application sharing and video conferencing have become feasible. Application and desktop sharing allows sharing of any application with one or more people over the Internet. The participants receive the screen-view of the shared application from the server. Their mouse and keyboard events are delivered and regenerated at the server. Application and desktop sharing enables collaborative work, software tutoring, and e-learning over the Internet. I have developed a high performance application and desktop sharing system called BASS which is efficient, reliable, independent of the operating system, scales well via heterogeneous multicast, supports all applications, and features true application sharing.

Most of the time an application sharing session requires audio and video conferencing to be more useful. High quality video conferencing requires a fair amount of bandwidth and unfortunately Internet bandwidth of home users is still limited and shared by more than one application and user. Therefore, I measured the performance of popular video conferencing applications under congestion to understand whether they are flexible enough to adapt to fluctuating and limited bandwidth conditions. In particular, I analyzed how Skype, Windows Live Messenger, Eyebeam and X-Lite react to changes in available bandwidth, presence of HTTP and BitTorrent traffic and wireless packet losses. To perform these measurements more effectively, I have also developed vDelay, a novel tool for measuring the
capture-to-display latency (CDL) and frame rate of real-time video conferencing sessions. vDelay enables developers and testers to measure the CDL and frame rate of any video conferencing application without modifying the source code. Further, it does not require any specialized hardware. I have used vDelay to measure the CDL and frame rate of popular video chat applications including Skype, Windows Live Messenger, and GMail video chat. vDelay can also be used to measure the CDL and frame rate of these applications in the presence of bandwidth variations.

The results from the performance study showed that existing products, such as Skype, adapt to bandwidth fluctuations fairly well and can differentiate wireless and congestion-based packet losses. Therefore, rather than trying to improve video conferencing tools, I changed my focus to end-user created communication-related services to increase the utility of existing stand alone Internet services, devices in the physical world, communication and online social networks. I have developed SECE (Sense Everything, Control Everything), a new language and its supporting software infrastructure for user created services. SECE allows non-technical end-users to create services that combine communication, social networks, presence, calendaring, location and devices in the physical world. SECE is an event-driven system that uses a natural-English-like language to trigger action scripts. Users associate actions with events and when an event happens its associated action is executed. Presence updates, social network updates, incoming calls, email, calendar and time events, sensor inputs and location updates can trigger rules. SECE retrieves all this information from multiple sources to personalize services and to adapt them to changes in the users context and preferences. Actions can control the delivery of email, change the handling of phone calls, update social network status and set the state of actuators such as lights, thermostats and electrical appliances.
# Table of Contents

1 Introduction ........................................ 1

2 BASS Application Sharing System .......................... 6
   2.1 Introduction ........................................ 6
   2.2 Related work ......................................... 9
   2.3 BASS system architecture ............................... 19
   2.4 The BASS protocol .................................... 21
       2.4.1 Introduction .................................... 21
       2.4.2 Overview .......................................... 21
       2.4.3 Remoting Protocol ................................. 28
       2.4.4 Human Interface Protocol (HIP) ....................... 34
       2.4.5 Using BFCP for application and desktop sharing ...... 37
   2.5 Client architecture .................................... 38
   2.6 Windows XP server architecture ........................ 38
       2.6.1 Mirror driver .................................... 39
       2.6.2 Server architecture ................................. 39
       2.6.3 Serving window updates to users with different bandwidths 40
       2.6.4 Encodings ......................................... 42
       2.6.5 Minimizing the effect of packet loss for UDP clients 43
       2.6.6 Reliable multicast ................................. 43
       2.6.7 Streaming video support ............................ 44
   2.7 Performance results ................................... 45
5.3.5 Context-based rules ........................................... 100
5.3.6 States vs. events .............................................. 101
5.4 The software architecture of SECP .......................... 102

5.4.1 The software components of SECP ......................... 104
5.5 Sensors and actuators ............................................ 106
5.6 Location .......................................................... 109
5.7 Presence and Instant Messaging (IM) ...................... 111

5.7.1 Architecture to support several presence and IM networks .......... 112
5.7.2 Integration of SIMPLE .................................... 117
5.7.3 Integration of XMPP ....................................... 117
5.8 Integration with external services ............................ 118

5.8.1 Overview ...................................................... 118
5.8.2 OAuth (Open Authentication) authentication mechanism .......... 118
5.8.3 Social networks ............................................. 120
5.8.4 VoIP systems ............................................... 124
5.8.5 Translation .................................................. 125
5.8.6 Calendar services ......................................... 126
5.8.7 Online photo sharing services ............................. 127
5.8.8 Google Voice ............................................... 128
5.8.9 Address book .............................................. 128
5.9 Adding new event types and action commands to SECP ........ 129

5.9.1 Overview ...................................................... 129
5.9.2 Adding a new action command to the SECP ................. 129
5.10 A graphical user interface for SECP .......................... 133
5.11 Summary and evaluation of the SECP ...................... 142

6 Conclusions .......................................................... 144

Bibliography .......................................................... 145
List of Figures

2.1 Application sharing system architecture .............................................. 8
2.2 Desktop with overlapping windows ....................................................... 10
2.3 BASS client view .................................................................................. 11
2.4 MAST client view .................................................................................. 12
2.5 UltraVNC client view ............................................................................ 13
2.6 MetaVNC sharing area selection ............................................................ 14
2.7 MetaVNC client view ............................................................................ 15
2.8 Google+ Hangouts with extras sender side menu selection .................. 16
2.9 Google+ Hangouts with extras sender side child window ...................... 16
2.10 Google+ Hangouts with extras receiver side ........................................ 17
2.11 An AH shares three windows ............................................................... 22
2.12 Participant 2 displays the shared windows in shifted coordinates .......... 23
2.13 Participant 3 displays the shared windows in completely different coordinates 23
2.14 Application sharing protocol message structure ................................... 26
2.15 Common remoting/HIP header ............................................................ 28
2.16 A Window Record ............................................................................. 29
2.17 An example WindowManagerInfo message with three Window Records .... 30
2.18 Common remoting/HIP header for RegionUpdate messages ................. 31
2.19 An example non fragmented RegionUpdate message ......................... 32
2.20 Message specific payload for move rectangle ...................................... 32
2.21 Message specific payload for mouse pressed, released and moved messages 35
2.22 Message specific payload for mouse wheel moved .............................. 36
4.5 (a) Capture-to-delay latency (CDL) (b) Standard deviation of capture-to-display latency (c) First read rate (FRR) (d) Frames per second (fps) (e) Bitrate (f) CPU utilization for video encoding

4.6 CDL, fps, and FRR for Skype as a function of time when the available bandwidth is adjusted as a step function

4.7 Comparison of video chat applications in terms of latency

4.8 Comparison of video chat applications in terms of fps

5.1 Timelines web aggregation service

5.2 User information registry (partial)

5.3 The overall software architecture of SECE

5.4 The software components of SECE

5.5 The software stack of SECE

5.6 Phidget experimental setup

5.7 The architecture of sensors and actuators gateway

5.8 SECE’s polygon editing interface

5.9 Twitter’s OAuth authentication flow

5.10 OAuth authentication flow

5.11 SECE generated photo uploaded to Flickr

5.12 SECE homepage

5.13 GUI architecture

5.14 Rule editing interface

5.15 Example rule assistance feature

5.16 Action commands assistance feature

5.17 Lego Mindstorms software

5.18 Registration of third-party services to SECE
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Comparison of related work</td>
<td>18</td>
</tr>
<tr>
<td>2.2</td>
<td>Remoting protocol message types</td>
<td>29</td>
</tr>
<tr>
<td>2.3</td>
<td>Marker and FirstPacket bits carry fragmentation info</td>
<td>31</td>
</tr>
<tr>
<td>2.4</td>
<td>HIP message types</td>
<td>34</td>
</tr>
<tr>
<td>2.5</td>
<td>HID status values</td>
<td>37</td>
</tr>
<tr>
<td>4.1</td>
<td>The size of timestamp encoded as an EAN-8 barcode w.r.t resolution of captured video</td>
<td>78</td>
</tr>
<tr>
<td>4.2</td>
<td>Comparison of video chat applications. The results are sorted by capture-to-display latency (CDL). The ‘G’ in the Resolution column is our best guess of the video resolution. LL, HL and HQ are abbreviations for low latency, high latency, and high quality</td>
<td>83</td>
</tr>
<tr>
<td>5.1</td>
<td>SECE makes it easy to integrate external knowledge seamlessly</td>
<td>94</td>
</tr>
<tr>
<td>5.2</td>
<td>Summary of SECE events and actions</td>
<td>119</td>
</tr>
</tbody>
</table>
Acknowledgments

First and foremost, I want to thank my adviser Professor Henning Schulzrinne. Since the beginning of my study, he led me to the road to successful research. Instead of giving me direct answers, encouraged me to do more research, thinking and evaluation on my side. Not only he taught me how to find a real research problem, but also how to solve it in more reasonable and elegant way. It is a fact that without his ideas, suggestions, insights and funding this thesis would not have been possible.

Secondly, I would like to express my sincere gratitude to my former wife. During my long master and doctoral study, she was with me and made this foreign land a home for me. The true value of her support and accompaniment was priceless.

I would also like to thank my father Mustafa, my mother Ayse and my brother Mahmut. I always felt their support and guidance near me, although, they were thousands of miles away from me. They not only supported me financially but also morally and spiritually.

My sincere thanks are due to the thesis committee members, Prof. Gail Kaiser, Prof. Junfeng Yang, Prof. Jason Nieh and Dr. Anjur Sundaresan Krishnakumar. Their comments and suggestions definitely improved the quality of my research and thesis.

I want extend my thanks to my colleagues Salman Abdul Baset, Andrea Forte and Victoria Beltran Martinez whom I worked very closely and published papers together. I wish to thank my office mates Wonsang Song and Jong Yul Kim for making our office like a family environment. I am grateful to Jan Janak for reviewing my thesis, working together on the SECE project and giving insights on several different topics. I also wish to thank all my lab mates Jae Woo Lee, Kumiko Ono, Suman Srinivasan, Se Gi Hong, and Kyung Hwa Kim.

I owe my deepest gratitude to my friends Enver Cavus and Sedat Bitik for their support, suggestions, reviews, encouragement, prayers and visits to New York.
To my former wife
Chapter 1

Introduction

There is no need to mention the benefits of collaborative working on a document, drawing or presentation. Although the Internet enabled real-time multimedia collaboration such as instant messaging, audio and video conferencing, and application sharing, the limited bandwidths of participants and inadequate collaboration tools slowed down the adoption of video conferencing and application sharing applications. Improvements in Internet bandwidth of home users do not solve this problem completely because uplink bandwidth which is used by collaboration tools is still limited and shared with other applications and users. Most residential Internet connections are asymmetric\(^1\) which means bandwidth from the user to the Internet (uplink) is less than the bandwidth from the Internet to the user (downlink). The general assumption behind this asymmetry is that most users download a lot more content than they upload. However, real-time multimedia collaboration tools such as video conferencing and application sharing not just use the downlink but also the uplink. For example, Skype recommends a sustained 1 Mb/s symmetrical bandwidth or higher to make an HD video call\(^2\). Similarly, The situation is similar for application sharing; a screenshot of an 1280x800 resolution desktop is around 200-800 KB (depending on the content) using lossless compression and will take 2-8 seconds to transmit over a 1 Mb/s connection. Ac-

\(^{1}\)http://bernoullinetworks.com/node/28

According to the Federal Communication Commission (FCC) report \(^3\), at mid-year 2010, 63% of reportable connections were slower than 768 kbps in the upstream direction, 18% were at least 768 kb/s in the upstream direction but slower than 1.5 Mb/s, and 19% were at least 1.5 Mb/s in the upstream direction. Sharing the limited uplink with other applications such as BitTorrent running on the same computer or on the other users’ computers (sharing the same connection) makes things a lot harder.

In the first part of this dissertation, firstly, I present a high-performance application and desktop sharing system which takes limited and shared uplink bandwidths into account. Secondly, I present the performance measurement results of popular video conferencing applications such as Skype running on congested links such as residential uplinks. Finally, I present a novel tool to measure the capture-to-delay latency and frame rate of a video conferencing session in real-time.

On the basis of our measurement results, we believe that our proposed application sharing tool and current video conferencing applications such as Skype do a good job under limited and fluctuating bandwidth conditions. Realizing more work on real-time multimedia collaboration will only bring incremental improvement, I changed my focus to end-user created communication-related services. The second part of the dissertation presents our proposed solution, the SECE (Sense Everything, Control Everything), which enables end users to create services which takes actions (e.g. sends an SMS reminder, turns on the air conditioner or sprinklers, updates the user’s social network status, and tweets) triggered by events (e.g., a change in the user’s location, calendar or availability, an incoming call, or an update on weather or stock prices).

The first contribution of this dissertation is a real-time multimedia collaboration tool for application and desktop sharing: BASS (BASS Application and Sharing System) [Omer Boyaci and Henning Schulzrinne, 2008]. Application and desktop sharing allows two or more people to collaborate on a single document, drawing or project in real-time. Some applications like Netbeans and Google Docs are collaboration-aware and allow more than one person to work on the same document at the same time [net, 2011a; goo, 2011]. However,
most applications are not collaboration-aware. Fortunately, collaboration features can be added to existing single-user applications transparently, without changing their source code. There are two models for collaboration-transparent application sharing: application-specific and generic. An application-specific solution allows to share a specific application such as Microsoft Word [Sun et al., 2006] or Autodesk Maya [Agustina et al., 2008], while a generic one allows to share any application. Application-specific solutions are expensive in terms of engineering cost and they may not allow to use all features of the application [Sun et al., 2006; Agustina et al., 2008]. Also, to participate in a sharing session, all participants must have a copy of the shared application. In the generic model, the application can be anything such as a word processor, CAD/CAM, presentation software or movie editor. Also, the participants do not need to install the application. One disadvantage of generic application sharing is that its generic nature makes it less efficient as compared to the application-specific model in certain scenarios. I developed an application and desktop sharing system, BASS, based on the generic model.

Application sharing differs from desktop sharing. In desktop sharing, a server distributes all screen updates for the whole screen or a defined rectangle. In application sharing, the server distributes screen updates if and only if they belong to the shared application’s windows to preserve privacy. The main challenges of application and desktop sharing are participant scalability, reliability, privacy, operating system independence, and performance. BASS scales quite well via reliable multicast as discussed in Section 2.6.6. The sharing system should be efficient in the sense that it should transmit only the changed parts of the screen, and it should not consume all the bandwidth and CPU resources while doing this. BASS uses the most efficient technique, a mirror driver, to detect changed regions of the screen. Animations and videos become an important component of multimedia applications and they require more bandwidth for collaboration than text and images. BASS uses different encodings for different regions of the screen which yields high performance for videos and animations in terms of bandwidth and frame rate.

The second contribution of this thesis is the performance measurement results of popular video conferencing applications under congestion [Omer Boyaci et al., 2009a]. As we mentioned before, regular home users with DSL or cable connections may not have enough
bandwidth all the time to experience high quality video communication. Therefore, video chat applications have to deal with changing and limited bandwidth while trying to maximize the video quality. To determine the performance of video chat applications under congestion, I measured how Skype, Windows Live Messenger, Eyebeam and X-Lite react to changes in available bandwidth, presence of HTTP and BitTorrent traffic and wireless packet losses. The performance measurement results indicated that some of the current solutions such as Skype seem to adequately address the problems of differentiating wireless losses from congestion losses and quickly adapting fluctuating and limited bandwidths.

Differentiation of losses and adaptation to bandwidth fluctuations are not enough to experience a great video conference because frame rate and latency are also very important for real-time video communication. During the measurement study mentioned above, we realized the necessity of such a tool to precisely measure the end-to-end delay and the frame rate of a real-time video stream.

The third contribution is \textit{vDelay} [Omer Boyaci \textit{et al.}, 2009b], a novel tool to measure the capture-to-display latency (CDL) and frame rate of a real-time video stream. Real-time video chat applications have three key software components: a video encoder that compresses the video captured from the camera, a video decoder that decompresses the video received over the network, and a playout buffer that smooths the playout of received video due to network variations. These software components impact CDL and frame rate of the real-time video played at a receiver application. Capture-to-display latency is the total time to encode and decode a video frame, playout buffer time, and latency of the network path. Along with bitrate, these two metrics provide quick insights into the performance of a real-time video application. Developers and testers can use these metrics to determine whether the measured performance of a video chat or conferencing application meets the expectations of users. Moreover, since numerous video chat applications are available, testers can publish the CDL and frame rate metrics to guide regular user’s selection of a video chat application.

\textit{vDelay} has three important properties. First, it does not require any change in the source code or executable of a real-time video application. Thus, it can be used to measure the CDL and frame rate of closed source video applications. Second, \textit{vDelay} does not require
any specialized hardware. Third, it is written in Java so it is platform independent and can be used to measure CDL and frame rate of a real-time interactive video application on any operating system.

We have used vDelay to measure the CDL and frame rate of the popular video chat applications Skype, Windows Live Messenger, and GMail video chat. Combined results from these two measurements have highlighted the performance of video conferencing tools in terms of CDL, frame rate, adaptation to bandwidth fluctuations and capability to differentiate congestion losses from wireless losses.

The final contribution is the development of the SECE (Sense Everything, Control Everything) [Omer Bovaci et al., 2010], a new language and supporting infrastructure which allows end-users with limited technical skills to create services that combine communication, presence, social networks, calendaring, location and devices in the physical world. Although several Internet services improve our daily life, they are not integrated and programmable by end-users, decreasing their utility. SECE is a context-aware platform that connects services that until now were isolated, leading to new, more useful and user-personalized, composite services. These services do not require user interaction; they are automated.

SECE takes actions automatically on behalf of the users depending on the monitored information and triggered events. In order to build such a system, the user has to define event-action rules. SECE uses a natural-English-like language to define event-action rules. The syntax and semantics of SECE language are designed such that it is easily understandable and usable by non-technical end-users. An example script which turns the homes lights on every sunset shows the end-user friendliness of SECE: "every sunset homelights on; ". The main challenges were to keep the language user-friendly while not decreasing its power and to develop the software which and has to integrate and communicate with several Internet services such as email, IM, phone, SMS, location, calendar, presence and translation services, and social networks.

I have developed a multi-user SECE server prototype which allow users to edit, compile, and deploy SECE scripts. It has a web-based graphical user interface where users can manage their rules and third-party service subscriptions. The web-based user interface communicates with a back-end server which stores and executes user rules.
Chapter 2

BASS Application Sharing System

2.1 Introduction

People who want to work collaboratively over the Internet on a document, drawing, or presentation in real-time need an application and desktop sharing tool. During the sharing session, this tool will distribute the screen-view of the shared application to the participants and receives and regenerates mouse and keyboard events coming from the participants.

I developed an application and desktop sharing tool called BASS which is CPU and bandwidth efficient, reliable in the face of packet losses, adapts to changing bandwidth conditions, is independent of the operating system, scales well in terms of number of participants via heterogeneous multicast, supports all applications, and features true application sharing. Any application can be shared, including word processors, browsers, presentation software or video players. Also, the participants do not need to install the application.

Application sharing differs from screen sharing. In screen sharing, there is no notion of a shared application; the server distributes whether a particular region or all of the screen. The background image and all the windows in this shared region will be send to participants. However, in application sharing, the server distributes screen updates if and only if they belong to the shared application’s windows. Applications often consist of a changing set of related windows which serve the same task and are usually associated with the same process on the host computer. Implementing screen sharing is easier than application sharing, however it does not provide security and privacy. Screen sharing may expose non-shared window contents to participants and may allow unauthorized access to
non-shared applications. Application sharing requires tracking the windows of the shared application. Tracking only the boundary of the shared application’s main window is not enough. Windows of non-shared applications may cover the shared windows or shared application may open new child windows such as those for selecting options or fonts. A true application sharing system must block all the nonshared windows and must transfer all the child windows of the shared application. Several related sharing systems claim application sharing, however they just track the coordinates of the shared application’s main window and they transmit any window whether shared or non-shared from this region.

Remote access to graphical applications and desktops has two important characteristics. First, the access protocol is unaware of any semantic characteristics of the applications being shared; it only transmits the visual characteristics of the windows. This is different, therefore, from shared-drawing or shared-editing tools that allow distributed modification of documents. Secondly, the protocol is designed to work with applications which were not written to be used remotely, by intercepting or simulating their connections to their native window systems. That distinguishes it from systems such as the X Window System [Balde-schwieler et al., 1993] which allow natively-written applications to be displayed on remote viewers.

We note that remote access to an application (“remote desktop”) and multiple users sharing an application within a collaboration setting such as a multimedia call or multiparty conference are very similar. Therefore, the BASS implementation and the protocols defined in this chapter support both.

Most video encodings have been designed for photographic video input which makes them unsuitable for application sharing. In particular, screen encoding may need to be lossless and typically operates on artificial rather than natural (photographic) video input. The video input is characterized by large areas of the screen that remain unchanged for long periods of time, while others change rapidly. (However, rendering the output of a modern computer-generated animation application such as video games blurs the distinction between traditional motion video output and screen sharing.)

Section 2.2 discusses related work. The system architecture is discussed in Section 2.3 and the details of the BASS protocol are discussed in Section 2.4. Client and server architectures are explained in Sections 2.5 and 2.6, respectively. Finally, Section 2.7 compares the performance of BASS to other systems in terms of bandwidth and frame rate.
**Definitions**  An *application host* (AH) is the computer which runs the shared application, distributes the screen updates to the participants, and regenerates human interface events received from participants.

*Participant* is the computer which receives screen updates from AH and sends human interface events back to the AH. Participants do not need to store or run the shared application. More than one participant may connect to a single AH.

The *remoting protocol* messages allows the AH to distribute windowing information and screen updates to participants.

The *human interface protocol* (HIP) allows participants to send human interface device (HID) events to AH. HIDs generates mouse or keyboard events such as a key press, key release, mouse move, and mouse click.

We distinguish between the AH user and participants. The AH user interacts with the application using normal operating system mechanisms. Participants interact via the delivery protocols described here. An application and desktop sharing system adheres to a client-server architecture (Figure 2.1).

---

**Figure 2.1: Application sharing system architecture**
2.2 Related work

Several application sharing systems were proposed since the invention of computer networks and network-aware windowing systems. Rapport [Ahuja et al., 1988] was one of the earliest. It was based on the the X Window System [Jones, 1989] which is a network-transparent window system. Several X protocol multiplexors such as SharedX [Daniel Garfinkel and Yip, April 1994], DMX [dmx, 2004], XMX [xmx, 1999], Xmux [McFarlane, 1991] and CCFX [Krantz et al., 1998] have been developed [Baldeschwieler et al., 1993] later. These multiplexors fail to support heterogeneous X servers. Also, it is not possible to support late joiners. A comparison of early application sharing systems are discussed in Ahuja et al. [Ahuja et al., 1990].

Microsoft provides Windows Meeting Space for Windows Vista and Netmeeting for Windows XP. Netmeeting was released in 1999 for Windows 98; in my tests, it failed to display pop-ups and menus. Windows Vista introduces application sharing feature as part of Windows Meeting Space [wms, 2011], but all the attendees must use Windows Vista. In 2008 Microsoft released SharedView [sha, 2011] which works in all Window versions and supports application sharing.

VNC [Richardson et al., 1998] is a cross-platform open source desktop sharing system but it supports only screen sharing. VNC uses a client-pull based transmission mechanism which performs poorly compared with server-push based transmissions under high round-trip time (RTT).

SharedAppVnc [Wallace and Li, 2007] supports true application sharing, but the delay is on the order of seconds. It uses a lossy codec and does not support multicast.

Some sharing systems such as UltraVNC [uvn, 2011], MetaVNC [met, 2010] and Multicast Application Sharing Tool (MAST) [Lewis et al., 2006] claim application sharing support. However, they only track the screen rectangle of the shared application’s main window and they distribute any window within this screen rectangle whether they are shared or non-shared. This may expose information from other non-shared applications. Shared application may open new child windows such as those for selecting options or fonts. UltraVNC, MetaVNC and MAST failed to share child windows. A true application sharing system must block all the non-shared windows and must transfer all the child windows of
the shared application. For example, if a user wants to share only the “Internet Explorer” application, which has the title “Windows Live Hotmail - Windows Internet Explorer”, from the desktop seen in Figure 2.2, then the participants should only see the main and the “Internet Options” windows. BASS (Figure 2.3) and Microsoft SharedView display only these two windows with a correct size while blocking the desktop background and the non-shared windows. In my test MAST, discussed in the following section, did not display the shared application in correct size and did not block the non-shared application and desktop background (Figure 2.4). Similarly, UltraVNC version 1.0.5 failed in my tests due to following problems: the cursor position did not match, windows belonging to unshared applications are shared, new windows belonging to same application are not included and long menus are not shown properly (Figure 2.5). MetaVNC allows user to select which window to share
Figure 2.3: BASS client view
not which application to share (Figure 2.6), therefore it has the same problems: unshared
application windows are shared and child windows are not shared (Figure 2.7).

Current sharing solutions perform poorly if the user wants to share photos or movies. They use the same encoding for text, computer-generated images, movies, and photographic images. Lossless encodings give poor performance for movies and photographic images. While lossy encodings generate visual artifacts around texts and computer-generated images such as straight lines. THINC [Baratto et al., 2005] and RDP [rdp, 2011] can play full motion movies if the bandwidth between the user and participant is tens of Mb/s. Due to their high bandwidth requirements, they do not scale well for large number of participants, and they do not perform well for realistic bandwidth conditions. BASS is the only system which uses different encodings for different regions of the screen. BASS uses the Theora [the, 2011] video codec to stream movies, JPEG [jpe, 1992] to transmit images, and PNG [png, 2011] for the rest.

VNC, Netmeeting and Windows Meeting Space all rely on unicast only, so they do not scale well to larger groups. Sharing an application via unicast increases the bandwidth usage linearly with the number of participants. For instance, Microsoft suggests Windows Meeting Space and SharedView for groups of no more than 15 users. The TeleTeachingTool [Ziewer and Seidl, 2002] and MAST use multicast to overcome this limitation. The TeleTeachingTool adds multicast support to VNC servers; however, it is developed just for online teaching, so it does not allow participants to control the shared desktop. Also, it does not support
Figure 2.5: UltraVNC client view
true application sharing due to its underlying VNC system. MAST allows remote users to participate via their keyboard and mouse, but its screen capture model is based on polling which is very primitive and not comparable to current state-of-the-art capturing methods like mirror drivers, discussed in Section 2.6.1.

Although both TeleTeachingTool and MAST use multicast for scalability, they do not address the unreliable nature of UDP transmissions. Even if the packets are delivered, they may be out of order. In order to compensate for packet loss, the TeleTeachingTool and MAST periodically transmit the whole screen which increases the bandwidth and CPU usage. Table 2.1 compares the sharing systems discussed so far. Nieh et al compared sharing systems in detail [Lai and Nieh, 2006].
Figure 2.7: MetaVNC client view
Figure 2.8: Google+ Hangouts with extras sender side menu selection

Figure 2.9: Google+ Hangouts with extras sender side child window
Figure 2.10: Google+ Hangouts with extras receiver side
BASS application sharing was developed between December 2005 and December 2008. Several other solutions are released to market after 2008. Microsoft released Shared-View [sha, 2011] in 2008 which works in all Window versions and supports true application sharing. TeamViewer [tea, 2011a], a VNC based application sharing solution, supports true application sharing since December 2009 [tea, 2011b]. Google released Google+ [han, 2011a] to public on September 2011. It features "Hangout with extras" [han, 2011b] which allows to share your screen or a particular window with the participants. As of December 2011, Hangouts did not transfer menus (Figure 2.8) and child windows (Figure 2.9) of a shared application (Figure 2.10), although it blocked unshared windows. Also, the image quality of Hangouts can be low due to its lossy codec. Google Hangouts does not allow participants to remotely control the shared application, it just transmits the screen capture of the shared window.

<table>
<thead>
<tr>
<th></th>
<th>Scalable</th>
<th>App Sharing</th>
<th>Remote Control</th>
<th>Recording</th>
</tr>
</thead>
<tbody>
<tr>
<td>UltraVNC</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Windows Meeting Space</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Microsoft SharedView</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>TeleTeachingTool</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Mast</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Teamviewer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Google+ Hangouts</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>BASS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 2.1: Comparison of related work
2.3 BASS system architecture

BASS is based on a client-server architecture. The server is the computer which runs the shared application. The participants use a lightweight Java \cite{jav, 2011} client application for connecting to the server, and they do not need the shared application. Clients receive screen updates from the server and send keyboard and mouse events to the server.

The Java client works in every operating system. The server could not be written in Java because Java does not have OS level windowing information and cannot learn screen updates from the OS. Therefore, there should be a server for each operating system and I have developed a Windows XP server and mirror driver. A mirror driver is the best known technique for capturing screen update events and will be discussed in detail in Section \ref{mirror}. The mirror driver runs in kernel mode and notifies the user mode server when it detects changes in the GUI of the shared application. The server then prepares an RTP packet containing encoded image of the updated region. RTP allows the clients to re-order the packets, recognize missing packets and synchronize application sharing with other media types like audio and video. The screen updates can be encoded with PNG \cite{png, 2011}, JPEG \cite{jpe, 1992} or Theora \cite{the, 2011} according to their characteristics. PNG is an open image format which uses a lossless compression algorithm \cite{Deutsch and Gailly, 1996; Deutsch, 1996} and more suitable for computer-generated images. JPEG is lossy, but more suitable for photographic images. Theora is an open source video codec comparable to H.264 and suitable for movie encoding.

The server supports both multicast and unicast transmissions. For unicast connections, either UDP or TCP can be used. Since TCP provides reliable communication and flow control, it is more suitable for unicast sessions than UDP. Multiple TCP clients sharing a single application may have different bandwidths, so I have developed an algorithm which sends the updates at the link speed of each client. For UDP clients, the server controls the transmission rate because UDP does not provide flow and congestion control. Several simultaneous multicast sessions with different transmission rates can be created at the server. The server can share an application to TCP clients, UDP clients, and several multicast addresses in the same sharing session.

Participants can join a sharing session anytime, and they need the full screen buffer
before receiving partial updates. Therefore, they send a RCTP-based feedback message, Full Intra-frame Request (FIR) [Ott et al., 2006], after joining the session. The server prepares and transmits the image of the whole shared region after receiving an FIR message. Preparing a full screen update is costly in terms of CPU, so the sharing server stores the generated image for some time.

Although multiple users can receive screen updates simultaneously, clearly only one of them can manipulate the application via keyboard and mouse events. BASS uses the Binary Floor Control Protocol (BFCP) [Camarillo et al., 2006] to restrict the control of the application to a single user. BFCP receives floor request and floor release messages from clients, and then it grants the floor to the appropriate client for a period of time while keeping the requests from other clients in a FIFO queue. All BFCP messages, keyboard and mouse events are transmitted directly to the server using TCP. Java’s key-codes are used for mouse and keyboard events because these events are captured from a Java client and regenerated by a Java component at the server. These key-codes are publicly available [jav, 2011b].

I have also added a recording feature to BASS. Participants can record the sharing session to a file. This file may be used for watching the session locally or streaming to multiple receivers simultaneously. This feature is very useful for preparing lectures or software tutorials for future use.
2.4 The BASS protocol

The protocol used between BASS clients and servers are published in Internet drafts [Boyaci and Schulzrinne, 2008a; Boyaci and Schulzrinne, 2008b]. This section gives a brief overview of the protocol; the Internet drafts covers the details.

2.4.1 Introduction

Application sharing can be integrated into the existing IETF session model, encompassing session descriptions using the Session Description Protocol (SDP) [Handley and Jacobson, 1998] or successors and the Session Initiation Protocol (SIP) [Rosenberg et al., 2002]. Application sharing needs many of the same control functions as other multimedia sessions, such as address binding and media negotiation.

The application sharing problem can be divided into four components: (1) setting up a session to the AH, (2) transporting human interface events from the participants to the AH, (3) delivering screen output from the AH to the participants, (4) moderating access to shared human interface devices such as pointing devices (e.g., mouse, joystick, trackball) and text input (keyboard). We refer to component (2) as the "human interface protocol (HIP)" and component (3) as the "remoting protocol". In our architecture, remote user input access is moderated by the Binary Floor Control Protocol (BFCP) [Camarillo et al., 2006].

Session negotiation and description can be provided by existing session setup protocols. Thus, they are beyond the scope of this document.

The rest of this chapter is laid out as follows. Section 2.4.2 gives an overview of the protocol’s architecture and components. The remoting protocol and HIP are described in Section 2.4.3 and 2.4.4, respectively.

2.4.2 Overview

2.4.2.1 Coordinate system

The origin (0,0) of the coordinate system is the upper left corner. All coordinates carried in the protocol messages are absolute and measured in pixels.
Figure 2.11 shows an example scenario where three windows are shared. All coordinates are absolute. A participant can display the windows in their original coordinates or it can display them in different coordinates. Participant 1 displays the windows in their original coordinates (Figure 2.11). Participant 2 shifts all the windows 220 pixels left and 150 pixels up (Figure 2.12). Participant 2 preserves the relations between windows, while participant 3 combines all the windows in order to fit them to its small screen (Figure 2.13). In this example scenario, all participants preserve the z-order of windows. The AH informs the participants about windows’ positions and sizes, z-order, and their groupings. The AH may assign same group identifier to the windows which belongs to the same process. Grouping information may be used by the participant while relocating the windows or enforcing the z-order. A participant may allow changing the z-order (i.e., stacking order) of windows locally,
Figure 2.12: Participant 2 displays the shared windows in shifted coordinates.

Figure 2.13: Participant 3 displays the shared windows in completely different coordinates.
CHAPTER 2. BASS APPLICATION SHARING SYSTEM

without changing the z-order in the AH. Several remoting/HIP message types carries left, top, width and height fields. The units of these fields are in pixels and they are unsigned.

The AH must only accept legitimate HIP events by checking whether the requested coordinates are inside the shared windows.

2.4.2.2 Operation

Detecting a change in the GUI of the shared application, the AH prepares an RTP [Schulzrinne et al., 2003] packet containing an encoded image of the updated region. RTP allows the participants to re-order the packets, recognize missing packets and synchronize application sharing with other media types like audio and video. The screen updates can be encoded with PNG [Duce, 2003], JPEG [Berc et al., 1998], JPEG 2000 [Futemma et al., 2008], Theora [the, 2011] or other media types like H.264 [Wenger et al., 2005], according to their characteristics. PNG is an open image format which uses a lossless compression algorithm and more suitable for computer generated images. JPEG is lossy, but more suitable for photographic images. JPEG 2000 supports both lossless and lossy compression, therefore suitable for both computer generated images and movies. Theora is an open source video codec comparable to H.264 and suitable for movie encoding.

Although multiple users could receive the screen updates simultaneously, clearly only one of them can manipulate the application via keyboard and mouse events. The Binary Floor Control Protocol (BFCP) [Camarillo et al., 2006] may be used to restrict the control of the application to a single user. BFCP receives floor request and floor release messages from participants; and then it grants the floor to the appropriate participant for a period of time while keeping the requests from other participants in a FIFO queue. The details of utilizing BFCP in the context of application and desktop sharing are given in Section 2.4.3.

The protocol supports two different mouse pointer models. Mouse pointer images can be transmitted as RegionUpdate messages or they may be transmitted separately as MousePointerInfo messages. The AH decides which mouse model to use. The participants must support both mouse models.

HIP supports both UTF-8 encoded Unicode characters and other keyboard keys which are not defined in Unicode such as function and control keys. For keyboard events publicly
available Java virtual key codes \cite{jav,2011b} are used.

The application and desktop sharing models defined in this section can be integrated into the IETF conferencing model. The Session Initiation Protocol (SIP) \cite{Rosenberg et al., 2002} can be used to initiate and control remote access. This allows the use of existing SIP mechanisms for confidentiality, authentication and authorization, user location, conferencing.

Additional, optional mechanisms can enhance application and desktop sharing. Audio streams can be associated with a desktop or application; participant-side scaling can be used to optimize transmission of data to participants with a small screen; and it is often useful to allow copy-and-paste between applications running on a participant and those running on an AH. This document does not define any such extensions.

The AH can support both multicast and unicast transmissions. For unicast connections, either UDP or TCP can be used. The AH can share an application to TCP participants, UDP participants, and several multicast addresses in the same sharing session.

2.4.2.3 Operation details for UDP-based participants

The AH controls the transmission rate for participants using UDP, because UDP itself does not provide flow and congestion control. Several simultaneous multicast sessions with different transmission rates can be created at the AH.

Participants can join a sharing session anytime, and they need the shared windows’ information and full screen buffer before receiving partial updates. Therefore, participants using UDP send an RCTP-based feedback message, Picture Loss Indication (PLI) \cite{Ott et al., 2006}, after joining the session. The AH prepares and transmits the windows’ state information and image of the whole shared region after receiving a PLI message.

2.4.2.4 Operation details for TCP-based participants

Since TCP provides reliable communication and flow control, it is more suitable for unicast sessions. TCP participants may have different bandwidths, so an algorithm which sends the updates at the link speed of each participant is needed.
Neither TCP nor RTP declares the length of an RTP packet. Therefore, RTP framing [Lazzaro, 2006] is used to split RTP packets within the TCP byte stream.

The AH prepares and transmits the windows’ state information and image of the whole shared region to the new participant, right after the TCP connection establishment.

2.4.2.5 Protocol overview of BASS

Application and desktop sharing protocol consists of two subprotocols: remoting and human interface protocol (HIP). Remoting messages transmit screen updates from AH to participants. HIP messages transmit mouse and keyboard events from the participant to the AH. Remoting and HIP messages are RTP messages. They consist of an RTP header, common remoting/HIP header, message-type specific header, and message payload (Figure 2.14).

The HIP messages have a different payload type than the remoting messages.

Remoting Protocol Overview  The remoting protocol consists of four messages from the AH to the participant and two control messages from participant to AH. The AH-to-participant messages are WindowStateInfo, RegionUpdate, MoveRectangle, and MousePointerInfo. The RTCP messages from UDP-based participant to AH are "Picture loss
indication (PLI)” and ”NACK request”.

The WindowManagerInfo message informs the participants about the windowIDs of the windows, their positions and sizes, z-order, and their groupings. All remoting messages carry the windowID to identify the target of message. For TCP participants, the AH transmits WindowManagerInfo message right after establishing a connection. UDP participants send a ”Picture loss indication (PLI)” to the AH as soon as they join the session. Receiving this PLI message, the AH transmits WindowManagerInfo message. The AH transmits RegionUpdate messages for updated regions. Whenever the shared window resizes or relocates, the AH sends a WindowManagerInfo message. Similarly, if the z-order of windows changes, the AH send a WindowManagerInfo message. MoveRectangle instructs the participant to move a region from one place to another, which is efficient for some drawing operations like scrolls. The MousePointerInfo message transmits the position and icon of the mouse pointer. Some AHs may transmit pointer images inside the RegionUpdate messages, so they may not need MousePointerInfo message.

”Picture loss indication (PLI)” and ”NACK request” are control messages and they are transmitted as RTCP messages. The ”NACK request” is used only by UDP participants to request retransmission of missing packets from the AH. AHs may support retransmissions. PLI can be used by both UDP and TCP participants to request a full screen refresh.

**Human Interface Protocol (HIP) Overview** HIP consist of seven messages: namely, MousePressed, MouseReleased, MouseMoved, MouseWheelMoved, KeyPressed, KeyReleased and KeyTyped. These messages are all from AH to participant and carried as RTP messages. However, these HIP messages have different payload type than the remoting messages.
2.4.3 Remoting Protocol

2.4.3.1 Payload format

RTP Header Usage The marker bit indicates the last packet of a multi-packet RegionUpdate (Section 2.4.3.2) message. The marker bit allows the receiver to finish decoding the picture, without waiting for the next packet with a new timestamp. Unless defined otherwise, all other message types must set this bit to zero.

The RTP timestamp indicates the time instance the remoting message has been created at the AH. The RTP timestamp is based on a 90-kHz clock. If a RegionUpdate message occupies more than one packet, the timestamp SHALL be the same for all of those packets. Furthermore, the initial value of the timestamp must be random (unpredictable) to make known-plaintext attacks more difficult [Schulzrinne et al., 2003].

The remaining RTP header fields are used as specified in RFC 3550.

Common remoting/HIP header All remoting protocol messages carry a common remoting/HIP header (Figure 2.15) which follows the RTP header. Message type and parameter fields are 8 bit identifiers, whereas the windowID is a 16-bit identifier. The windowID field is unsigned and has a range of 0-65535.

![Figure 2.15: Common remoting/HIP header](image)

Table 2 enumerates the message types defined in this document. Participants must implement all of them.

2.4.3.2 AH-to-participant messages

WindowManagerInfo The WindowManagerInfo message informs the participants about windows, their positions and sizes, z-order, and their groupings. This message transfers the complete window manager state to the participants. Each shared window resize and
Table 2.2: Remoting protocol message types

<table>
<thead>
<tr>
<th>Value</th>
<th>Message Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WindowManagerInfo</td>
</tr>
<tr>
<td>2</td>
<td>RegionUpdate</td>
</tr>
<tr>
<td>3</td>
<td>MoveRectangle</td>
</tr>
<tr>
<td>4</td>
<td>MousePointerInfo</td>
</tr>
</tbody>
</table>

relocation in any coordinate triggers a WindowManagerInfo message. Parameter and WindowID fields of common remoting/HIP header must be ignored. This message carries a message specific payload. One or more window records follow the common remoting/HIP header (Figure 2.16).

<table>
<thead>
<tr>
<th>WindowID</th>
<th>GroupID</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Left</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Top</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Width</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Height</td>
</tr>
</tbody>
</table>

Figure 2.16: A Window Record

Each window record is 20-bytes. The z-order information is given implicitly to the participants. The first record describes the window at the bottom of the stacking order, the last record the one on top. The "left" and "Top" fields carries the upper-left coordinate of the window. The "Width" and "Height" fields carries the width and the height of the window, respectively. Each window is assigned a WindowID. The participant must create a window for each new WindowID and must close this window after receiving a WindowManagerInfo message which does not contain this WindowID. GroupID fields informs the participant about grouping of windows. The AH may assign same GroupID to the
windows which belong to the same process. Grouping information may be used by the participant while relocating the windows or enforcing the z-order. The value of "0" for GroupID field is reserved and represents no grouping for given window.

Figure 2.17 is an example `WindowManagerInfo` message for the three shared windows in Figure 2.11. The participant must keep the existing window image after a resize and relocation.

<table>
<thead>
<tr>
<th>(Msg Type = 1)</th>
<th>Parameter = 0</th>
<th>WindowID = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>WindowID = 1</td>
<td>GroupID = 1</td>
<td>Reserved = 0</td>
</tr>
<tr>
<td>Left = 220</td>
<td>Top = 150</td>
<td>Width = 350</td>
</tr>
<tr>
<td>Height = 450</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WindowID = 2</td>
<td>GroupID = 2</td>
<td>Reserved = 0</td>
</tr>
<tr>
<td>Left = 850</td>
<td>Top = 320</td>
<td>Width = 160</td>
</tr>
<tr>
<td>Height = 150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WindowID = 3</td>
<td>GroupID = 1</td>
<td>Reserved = 0</td>
</tr>
<tr>
<td>Left = 450</td>
<td>Top = 400</td>
<td>Width = 350</td>
</tr>
<tr>
<td>Height = 300</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.17: An example `WindowManagerInfo` message with three Window Records
**RegionUpdate**  The RegionUpdate message instructs the participant to update the specified region of a window with new content. This message carries a message-type specific header and payload. This protocol supports all media types which have an RTP payload specification. It is possible that AH or participant may support only some media types. Therefore, they should negotiate supported media types during the session establishment. The 8 bit "parameter" field of the common remoting/HIP header will carry both the First-Packet bit and the actual payload type of the content. The 7 bit PT field carries the actual payload type of the content which can be PNG, JPEG, Theora, or any other media type which has an RTP payload specification. All AH and participant software implementations must support PNG images. The message-type specific header follows common remoting/HIP header. Message-type specific header consists of two 32 bit parameters, left and top. These two parameters informs the participants about the left-top coordinate of the RegionUpdate. The width and height of the RegionUpdate is not transmitted explicitly by this protocol.

![Figure 2.18: Common remoting/HIP header for RegionUpdate messages](chart)

If the content of the update does not fit into a single RTP message, it will be carried in several RTP payloads. All the payloads will carry the 32 bit common remoting/HIP header, while left and top fields are carried only in the first RTP payload. The marker bit and FirstPacket bit informs the participants about the fragmentation (Table 2.3).

**Table 2.3: Marker and FirstPacket bits carry fragmentation info**

<table>
<thead>
<tr>
<th>Marker bit</th>
<th>FirstPacket bit</th>
<th>Fragment Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Not Fragmented</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Start Fragment</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Continuation Fragment</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>End Fragment</td>
</tr>
</tbody>
</table>
Figure 2.19 displays an example RegionUpdate message. The RTP header is omitted in Figure 2.19. The RegionUpdate is non fragmented, therefore both the marker bit in the RTP header and FirstPacket bit in the payload header is set to 1.

MoveRectangle

The MoveRectangle message instructs the participant to move the specified region of a window to a new position. This message carries a message-type specific header. The AH informs the participants about the source rectangle via source left, source top, width and height parameters. Participants learns the destination coordinates from destination left and top parameters. Source and destination rectangles may overlap.

Figure 2.19: An example non fragmented RegionUpdate message

<table>
<thead>
<tr>
<th>Source Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Top</td>
</tr>
<tr>
<td>Width</td>
</tr>
<tr>
<td>Height</td>
</tr>
<tr>
<td>Destination Left</td>
</tr>
<tr>
<td>Destination Top</td>
</tr>
</tbody>
</table>

Figure 2.20: Message specific payload for move rectangle
MousePointerInfo  Some AHs may include the mouse pointer image inside the RegionUpdate messages. However, some AHs may choose to inform the participant about the mouse position and icon explicitly. If the RegionUpdate messages contain the mouse pointer icon, then the MousePointerInfo message is unnecessary. When receiving this message, the participant should draw the mouse pointer to the given position. This message carries a message specific payload. The format of this message is same as RegionUpdate message (Section 2.4.3.2) except they have different message types. The payload of MousePointerInfo message can be only the left and top coordinates. In this case, the participant must move the existing pointer image to the given coordinates. Payload may carry both the left and top coordinates and the new image of the mouse pointer. The participant must store and use this image until a new image arrives from the AH. If the AH uses MousePointerInfo messages, it must inform the late joiners about the current position and image of mouse pointer.

2.4.3.3 Participant-to-AH messages

Participants using UDP can send two RTCP messages to the AH. Late-joiners may inform the AH using the "Picture loss indication (PLI)" message in order to receive a full screen update. For the missing packets, UDP participants may send a "NACK Request".

Picture Loss Indication (PLI)  The "Picture Loss Indication (PLI)" message instructs the AH to generate a full screen update of the shared region. Before the full screen update, the AH will send a WindowManagerInfo message to inform the new participant about windows. Both TCP and UDP participants may transmit this message. The message format conforms to the "Picture Loss Indication (PLI)" section 6.3.1 of Ott et al., 2006.

NACK Request  The "NACK Request" message informs the AH about missing RTP packets. The message format conforms to the "Generic NACK" Section 6.2.1 of Ott et al., 2006. Multicast participants and AHs may take necessary precautions to prevent NACK storms such as waiting random amount of time before sending a "NACK Request" message.
2.4.4 Human Interface Protocol (HIP)

Participants may send human interface events to the AH in order to interact with the shared application.

2.4.4.1 Payload format

**RTP Header Usage** The marker bit must be set to zero by the participant and ignored by the AH.

The RTP timestamp indicates when the keyboard or mouse event occurred at the participant. The RTP timestamp of HIP messages is based on a 90-kHz clock. The initial value of the timestamp must be random (unpredictable) to make known-plaintext attacks on encryption more difficult; see RTP [Schulzrinne et al., 2003].

The remaining RTP header fields are used as specified in RFC 3550.

**Common remoting/HIP header** All HIP messages carry the same common remoting/HIP header shown in Figure 2.15 and discussed in Section 2.4.3.1. The WindowID parameter indicates the window where the keyboard or mouse event took place, i.e., the window that had keyboard or mouse focus.

The following HIP message types are defined:

<table>
<thead>
<tr>
<th>Value</th>
<th>Message Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>121</td>
<td>MousePressed</td>
</tr>
<tr>
<td>122</td>
<td>MouseReleased</td>
</tr>
<tr>
<td>123</td>
<td>MouseMoved</td>
</tr>
<tr>
<td>124</td>
<td>MouseWheelMoved</td>
</tr>
<tr>
<td>125</td>
<td>KeyPressed</td>
</tr>
<tr>
<td>126</td>
<td>KeyReleased</td>
</tr>
<tr>
<td>127</td>
<td>KeyTyped</td>
</tr>
</tbody>
</table>
2.4.4.2 **MousePressed** message

The **MousePressed** message instructs the AH to generate a mouse pressed event at the given coordinates of the screen. This message carries a message-type specific header. The "parameter" field of the common remoting/HIP header carries the mouse button information. The values of 1, 2 and 3 are defined for left, right, and middle button, respectively. The AH and participant may negotiate additional values for other mouse buttons. The AH may ignore unrecognized values. Message-type specific header for **MousePressed**, **MouseReleased**, and **MouseMoved** messages is same and illustrated in Figure 2.21.

![Figure 2.21: Message specific payload for mouse pressed, released and moved messages](image)

2.4.4.3 **MouseReleased** message

The **MouseReleased** message instructs the AH to generate a mouse released event at the given coordinates of the screen. This message carries a message-type specific header. The "parameter" field of the common remoting/HIP header carries the mouse button information. The values of 1, 2 and 3 are defined for left, right, and middle button, respectively. Other values may be defined for other mouse buttons. The AH may ignore unrecognized values.

2.4.4.4 **MouseMoved** message

The **MouseMoved** message instructs the AH to move the mouse pointer to the coordinates provided. This message carries a message-type specific header.
2.4.4.5 **MouseWheelMoved** message

The **MouseWheelMoved** message instructs the AH to generate a mouse wheel moved event at given coordinates of the screen. This message carries a message-type specific header.

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

*Figure 2.22: Message specific payload for mouse wheel moved*
```

The "distance" field carries the wheel rotation amount as "120 * (number of notches)". A mouse wheel has discrete, evenly spaced notches. When user rotates the wheel, a wheel message is sent to OS as each notch is encountered. The "distance" field does not carry number of notches in order to support a smooth-scrolling wheel. Instead, "distance" field carries each notch as 120. Some mice may only send multiples of 120, while a smooth-scrolling mouse may send any values. A positive value indicates that the wheel was rotated forward, away from the user; a negative value indicates that the wheel was rotated backward, toward the user. The negative values are transmitted using 2's complement method.

2.4.4.6 **KeyPressed** message

The **KeyPressed** message instructs the AH to generate a "key pressed" event. This message carries a message-type specific header which consists of a 32 bit KeyCode. Java virtual keycodes are used and they are publicly available on the openJDK website \[jav, 2011b\]. The actual values are inside the KeyEvent.java file. For example, F1 key is defined as "int VK_F1 = 0x70;" in KeyEvent.java.

2.4.4.7 **KeyReleased** message

The **KeyReleased** message instructs the AH to generate a "key released" event. This message carries a message-type specific header which consists of a 32 bit KeyCode. Java keycodes are used and they are publicly available at openJDK website \[jav, 2011b\]. The actual values
are inside the KeyEvent.java file. For example, F1 key is defined as "int VK_F1 = 0x70;" in KeyEvent.java. A KeyReleased event for a key without a prior KeyPressed event for this key is acceptable.

2.4.4.8 KeyTyped message

KeyTyped message instructs the AH to inject some number of UTF-8 encoded characters into the operating systems input queue. This message carries a message specific payload. There is no padding for the UTF-8 string. The participant must send more than one KeyTyped message if the string does not fit into a single KeyTyped packet.

2.4.5 Using BFCP for application and desktop sharing

Application and desktop sharing tools may utilize Binary Floor Control Protocol (BFCP) [Cammarillo et al., 2006] for managing the ownership of AH’s human interface devices (HID).

BFCP defines several messages, but only five of them is a must for Application and Desktop Sharing, namely "Floor Request", "Floor Release", "Floor Granted", "Floor Released" and "Floor Request Queued".

In Application and Desktop Sharing context, the floor is the AH’s HIDs. In this context, it is possible that the AH may temporarily block HID events without revoking the floor control. For example, the AH may temporarily block HID events if the shared application loses the focus or is covered by a non-shared application. The AH informs the current floor holder about the status of HIDs via STATUS-INFO attribute of ”Floor Granted” messages. The participant may receive several ”Floor Granted” messages with different ”HID Status” values. Participant applications may inform the user about current ”HID Status”. HID Status values are 16-bit unsigned values and are defined as:

Table 2.5: HID status values

<table>
<thead>
<tr>
<th>Value</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>STATE_NOT_ALLOWED</td>
</tr>
<tr>
<td>1</td>
<td>STATE_KEYBOARD_ALLOWED</td>
</tr>
<tr>
<td>2</td>
<td>STATE_MOUSE_ALLOWED</td>
</tr>
<tr>
<td>3</td>
<td>STATE_ALL_ALLOWED</td>
</tr>
</tbody>
</table>
2.5 Client architecture

The BASS client is very simple and lightweight compared with the server. It receives screen updates from the server and displays them to the participant. It is completely stateless in the sense that it can disconnect and reconnect to the server. Due to its simplicity, clients for different platforms can be easily developed. A Java BASS client has been developed in our lab. BASS clients can listen for or initiate connections. Participants of a sharing session can be view-only or they can request input control from the server. To request input control from server, a user presses the “control” button in the GUI of the client application (Figure 2.3). The client sends a “noor request message, and then the server responds with a ”granted” or ”request queued” message. The server grants the floor immediately if nobody else currently controls the floor. Otherwise, the request will be queued in a FIFO queue, and the floor will be granted to the requesters one-by-one automatically. Users can release the floor by pressing the “control” button again. The floor is automatically released after a period of inactivity or after the client holding the floor leaves the session. After the server grants the floor, the client captures all keyboard and mouse events locally and transmits them to the server via RTP messages.

2.6 Windows XP server architecture

The Windows server allows Windows XP users to share an application with other participants. The Windows server has two main components, a kernel-mode mirror driver and user-mode sharing server. The mirror driver tracks the updated regions of the screen, and notifies the user-mode sharing server about these updates through a shared memory region. The user-mode sharing server learns the updated regions, prepares region updates, and transmits these updates to the participants (Figure 2.23). The sharing server also receives and regenerates mouse and keyboard events from participants. These two components are examined in detail in the following sections.


2.6.1 Mirror driver

A mirror driver is a display driver that mirrors the drawing operations of a physical display driver. The Windows OS calls the physical and mirror display drivers with the same GDI (graphics device interface) commands. It is the most efficient way of learning screen updates because the operating system provides the exact coordinates of screen updates. I had to develop my own mirror driver for the sharing server because there is no free and open source mirror driver. Remote Desktop Connection and VNC also use their own mirror drivers to efficiently learn the screen updates. Stability and correctness are very important for kernel-mode components because they may easily cause restart or blue screen. My mirror driver is completely stable such that I have not observed any crashes at all for the last two years.

I have used shared memory to establish a communication channel between the mirror driver and the sharing server. Both mirror driver and the sharing server map the same region of the memory to their own address spaces. The shared memory consists of a frame buffer to keep the screen state and a ring buffer which is used by the mirror driver to insert update commands and coordinates. There are two types of commands, BitBlt and MoveRect. The Windows OS notifies the mirror driver for an update, and then mirror driver inserts this update to the ring buffer with command type and the coordinates of the region. In case of application sharing, the BASS server computes the bounding rectangle of the shared application windows and informs the mirror driver about the tracking region. The mirror driver only tracks this specific region instead of the whole desktop, decreasing CPU overhead.

2.6.2 Server architecture

The Windows XP sharing server is a user-mode process which complements the mirror driver. While the mirror driver keeps track of the frame buffer and the list of updated regions, the sharing server handles connection establishment, process keyboard and mouse events, and optimizes, compresses and transmits screen updates (Figure 2.23). The multi-threaded sharing server can serve multiple clients simultaneously. The server can wait for incoming connections and it can also connect to clients directly if instructed by the user. The sharing server has been designed considering the following challenges. Participants may
have different bandwidths and they can join in anytime. UDP-based multicast and unicast sessions should be reliable even though UDP does not provide reliability. Some regions or windows may require different encoding for better performance.

The sharing server has one main thread, one manager thread, and a number of client threads. The main thread periodically checks for updated regions and prepares encoded images of these regions. These images are inserted to the image ring buffer which stores them until they are transmitted by client threads.

2.6.3 Serving window updates to users with different bandwidths

Encoding screen updates is a CPU-intensive operation, so there is no point in generating lots of updates if the clients do not have enough bandwidth to display them. The manager thread observes each client’s bandwidth and throttles the main thread according to the highest bandwidth client. This technique can be easily explained with a movie playing example. Assume that the user shares a movie with remote participants. The same region of the screen is updated 24-30 times per second. Initially, the server tries to generate as many updates per second as possible. If at least one of the clients has enough bandwidth to receive these updates, the manager thread allows the main thread to continue this pace. But if none of the clients has enough bandwidth to deal with that update rate, then the manager thread slows the main thread by forcing it to sleep between update generations. Therefore, the main thread will generate a single update by combining several updates into one. The manager thread tries to equalize the update generation rate to the fastest client’s bandwidth speed. This technique prevents unnecessary CPU usage in the sharing server.

The effective bandwidths of clients may change during the sharing session. This is properly handled by the manager thread because it checks clients’ effective bandwidths by periodically asking how many bytes they transferred during this period. Each thread keeps track of the sent bytes to its client. A similar technique is utilized by the low bandwidth client threads. The fastest client thread transmits all the generated updates, however other client threads may not transmit all the generated updates due to their low bandwidths. These low bandwidth client threads skip some of the region updates if there are newer updates for these regions. Going back to video player example, the fastest client thread
Figure 2.23: Windows XP server architecture
may transmit 12 frames per second, whereas the other client threads may transmit some of these generated frames permitted by their bandwidths.

2.6.4 Encodings

The updates are distributed as PNG images except for movies or photos. PNG is very suitable and efficient for computer-generated images. However, its lossless nature results in a large increase in the compressed size of photographic images with negligible gain in quality, compared with JPEG and Theora which are specifically designed for photographic images. But the server does not know whether an updated region contains photographic or computer-generated content, because the mirror driver runs at the frame buffer level and at that level, there are only pixels and no metadata. Fortunately, detecting movie playing is very easy due to its specific characteristics. Different from other applications, movies generate 24-30 updates per second in a specific region of the screen. Benefiting from this characteristic, I have developed an algorithm to detect movie playing in order to use JPEG or Theora encoding for this region. Consecutive updates to a specific region trigger the detection. The Detection algorithm encodes the region using JPEG and compares the compressed image size between JPEG and PNG. If the JPEG size is less than a quarter of the PNG size, the server switches the default algorithm for this region to Theora and stores this result in a lookup table for subsequent updates. Theora is the default encoding for movies because it is four times more bandwidth efficient than JPEG. However, encoding Theora is costlier than JPEG, especially for high resolution movies. JPEG uses approximately four times more bandwidth than Theora but can generate 1.5 times more frames. The user can switch the server’s default encoding for movies from Theora to JPEG if all the participants have enough bandwidth. If the compression ratio of JPEG or Theora regions drops below 12:1 compared to raw bits during the session, the server deletes the stored encoding information for this region from the lookup table. Similarly, regions which are marked as PNG regions are periodically rechecked.
2.6.5 Minimizing the effect of packet loss for UDP clients

With UDP, packets can get lost. Region updates may require several kilobytes or even megabytes. Unless designed carefully, a single packet loss may destroy the region update completely. In order to minimize the effect of packet loss, I have developed an algorithm which generates several small PNG images for a given update region. Blindly generating a PNG image for each scan line may increase the bandwidth usage because a new zlib compressor object should be created for each new PNG. Creating a new zlib compressor decreases the compression ratio, so the bandwidth usage increases. My algorithm tries to maximize the number of scan lines included in a single UDP packet while trying to keep the packet size below 1500 bytes, which the MTU for Ethernet. Due to its adaptive nature, it may feed tens of lines for a text whereas it may feed only a single line for a photographic image. I have observed an increase of approximately twenty percent in bandwidth due to small PNGs.

Transmitting self-contained UDP packets minimizes the effect of packet loss. Instead of losing the complete region update, participants may lose only a few scan lines in case of a packet loss. They may end up with imperfect frame buffer due to packet losses, but they can continue to participate to the session. The retransmission mechanism discussed in the next section helps to restore the frame buffer state.

2.6.6 Reliable multicast

In the previous section, I described my algorithm which minimizes the effect of packet loss. In this section, I explain another supplementary technique which retransmits missing packets [Adamson et al., 2004] to deal with packet losses. When a packet loss occurs, the participant views the rest of the image except missing scan-lines. The client application sends a negative acknowledgment (NACK) for this missing packet [Ott et al., 2006]. Receiving this NACK, the server retransmits the requested packet. I have modified the oRTP library [ort, 2011] which is used in the server side and extended my own Java RTP library on the client side to support this feature. The client/server applications do not deal with retransmission and buffering of packets, as these are handled by the RTP libraries.

The retransmission mechanism accommodates malicious or corrupted client behavior.
and NACK storms. In order to protect itself from malicious clients, the server does not retransmit a packet more than three times. If a packet failed to reach to several multicast clients, they all send a NACK back to the server causing a NACK storm. Instead of sending a NACK right after detecting a packet loss, clients wait for a random amount of time (0-100 ms). If a client observes a multicast NACK from another client while waiting, it suppresses its own multicast NACK request.

2.6.7 Streaming video support

BASS is able to detect the regions of screen where a video is playing due to the fact that videos generate 24-30 updates per second in a specific region of the screen different from other applications. Benefiting from this characteristic, I have developed an algorithm to detect video playing in order to use JPEG or Theora encoding for this region. However, encoding in real-time is computationally expensive. Although a Pentium 4 can encode 426x320 movie in full motion, it can only encode 6-10 frames per second for a 852x480 movie. Therefore, I implemented another feature into BASS which enables to stream full motion movies to participants regardless of the resolution. The user copies the movie file into a BASS’s specific directory. BASS automatically detects the movie and displays it in the GUI. If the movie is not encoded in Theora, BASS transcodes the movie into Theora using ffmpeg2theora [ffm, 2011]. This preprocessing takes 30 seconds for a 20 seconds 852x480 MPEG-4 movie on a Pentium 4 3 GHz. After the transcoding user can stream the movie to participants using negligible CPU power. This feature is useful if the AH has the videos to be shared before the sharing session. For example, an instructor or presenter who will share a set of videos may transcode them beforehand.
2.7 Performance results

I compared the bandwidth usage of sharing systems for web browsing. I also compared them for playing movies in terms of both bandwidth usage and frame rate. All sharing systems use 24 bits per pixel except RDP, which uses 16 bits per pixel. If RDP used 24 bits, it would consume fifty percent more bandwidth. The tests were conducted in January 2008 and I used a Pentium 4 3 GHz CPU and 1 GB memory as the server and a Athlon XP 2600+ CPU and 1 GB memory as the client. Server and client are connected over a 100 Mb/s LAN. For the movie playing comparison over a low bandwidth measurement, I restricted the bandwidth of the client to 3 Mb/s using NetLimiter \[\text{net, 2011b}\]. To count the frame rate, I used a Canopus TwinPact100 scan-line converter. This box takes the RGB output of the client as input, and it outputs a digital movie stream via Firewire cable. I recorded this movie stream using the iMovie application of a Macbook pro. I then counted the individual frames to find the actual frame rate.

![Figure 2.24: Web browsing performance](image)

Figure 2.24 compares RDP, VNC and BASS for web browsing. During the measurement, the server automatically visited the home pages of the twenty most popular webpages according to alexa.com. I developed and used a Java-based automated testing application, available at \[\text{web, 2011}\], which visits each website for ten seconds. Some of these websites have animations and advertisements. Therefore, the bandwidth usage depends on how ea-
gerly a particular sharing system transmits updates. I can say that all systems consume almost the same bandwidth, around 1 Mb/s.

Although VNC and BASS use similar compression techniques, VNC consumes less bandwidth because it uses a single compressor during the session, while BASS uses a separate compressor for each update. Using a new a compressor for each update allows BASS to compress each update only once regardless of the number of participants. However, VNC has to compress the same update for each participant because each participant has a different compressor. In case of more than one participant, VNC consumes more CPU, while the CPU usage of BASS remains constant.

Another benefit of using a new compressor for each update is that packet losses causes limited problems. Using a single compressor for the whole session requires retransmission for any packet loss, otherwise rest of the stream can not be decoded by the receiver. However, in our implementation a packet loss makes only a single update unusable. This is particularly important during multicast sessions where packet losses can happen and retransmissions may not be feasible due to large number of participants. VNC itself does not support multicast which means with each new participant both the CPU and bandwidth consumption will increase. In the BASS multicast sessions the CPU and bandwidth consumption will remain steady regardless of number of users. TeleTeachingTool modifies VNC to support multicast; however, it is forced to use hextile encoding due to VNC’s single compressor for the whole session issue. Hextile encoding uses RRE (rise-and-run-length encoding) which is essentially a two-dimensional analogue of run-length encoding. Hextile consumes more bandwidth than zlib based solutions\(^1\). To address this problem, Google Summer of Code 2010/VNC\(^2\) project developed a new encoding Tight PNG which is basically what we implemented in BASS.

\(^1\)From Google Summer of Code 2010/VNC project website: “Hextile only uses tile-based compression. Other encodings are more popular such as Tight and ZRLE that use zlib-based compression. These encodings significantly reduce the bandwidth required by VNC.”

\(^2\)http://wiki.qemu.org/Google_Summer_of_Code_2010/VNC
I measured the multimedia performances of sharing systems by playing a movie over both an unlimited bandwidth link and a 3 Mb/s bandwidth link. The movie is a 20 seconds soundless 852x480 24 fps MPEG-4 encoded trailer of Warren Miller’s *Higher Ground*. The BASS server can be configured by the user to use JPEG or Theora for movies. BASS-T and BASS-J represent BASS systems which use Theora and JPEG for movies, respectively. BASS-M represents BASS’s Theora streaming feature, discussed in Section 2.6.7, instead of playing them in default media player.

Figure 2.25 compares sharing systems over an unlimited bandwidth link. BASS-M and THINC are able to play the movie in full motion, however THINC consumes 112 Mb/s, while BASS-M consumes only 1.6 Mb/s. RDP gives the second highest frame rate, but consumes 45 Mb/s. BASS-J gives 9 fps consuming just 2 Mb/s, and BASS-T gives 6 fps consuming less than 1 Mb/s. VNC is the worst performer in terms of frame rate. In conclusion, BASS gives acceptable frame rate while using less than 2 Mb/s.

![Comparison of sharing systems in terms of movie performance (unlimited bandwidth)](image_url)
Comparing sharing systems in unlimited bandwidth environments is not very realistic because some participants may have low bandwidths. I repeated the same experiments over a 3 Mb/s link (Figure 2.26). Frame rates of all sharing systems dropped less than a frame per second except BASS whose frame rate remained the same. BASS-M is able to play full motion movies over an 1.6 Mb/s link. In conclusion, over low bandwidth links, all three BASS configurations yield a frame rate that is at least six times than the other sharing systems.

Figure 2.26: Movie performance comparison of sharing systems (network is limited to 3 Mb/s)
2.8 Summary

Application and desktop sharing systems are an important part of real-time multimedia collaboration. They allow working on the same document, drawing, or presentation from different locations regardless of the operation system of participants. I have developed an application and desktop sharing system, BASS, which is highly scalable in terms of number of participants, CPU and bandwidth efficient, and independent of the operating system.

BASS supports all applications due to its generic model, and transmits only the shared application and its child windows.

BASS uses very little CPU cycles thanks to its mirror driver which tells the exact location of screen updates to the BASS. Also, its CPU usage stays steady regardless of number of participants thanks to compressing each update only once.

BASS, VNC and RDP consume roughly the same bandwidth for sharing a web browser. However, BASS uses several times less bandwidth than the others for playing videos owing to its algorithm which detects video activity independent of the video player in use. This unique feature of using different encodings for different parts of the screen differentiates the BASS from other sharing solutions.

BASS is specifically designed to support multiple participants. It compresses each update only once. Also, it supports multicasting with retransmissions and selective negative acknowledgments. Thanks to the multicasting, the bandwidth usage does not increase with new participants.

We have used industry standards like RTP, BFCP, and PNG while developing BASS. Others can develop clients and servers compatible with BASS by implementing its open protocol.

I have developed the BASS server on Windows XP. Recent versions of Windows, Mac OS X and Linux switched to modern GPU-enhanced compositing window managers. They are called Desktop Window Manager with the Windows Aero theme on Windows, Quartz Compositor with Aqua theme on Mac OS X, and XGL/Compiz on Linux. GPU-enhanced window managers brings new challenges to application and screen sharing. The frame buffer is no longer directly accessible by CPU, but resides inside the GPU. According to
Microsoft support ³: “Mirror drivers are based on the Windows XP display driver model. The new Desktop Window Manager in Windows 7 uses Windows Display Driver Model (WDDM) with a DirectX 9 class graphics processor to support Aero. Therefore, while a mirror driver is active, Windows disables Desktop Window Manager and Windows Aero.” I experienced a similar problem with vDelay receiver which needs to take a partial screenshot in every 5-10ms. Taking a screenshot takes more time when Aero is enabled than when it is disabled. The current solution to both problems is disabling the Aero until a better solution is developed by OS vendors.

Chapter 3

Performance of video-chat applications under congestion

3.1 Introduction

The application and desktop sharing system discussed in the previous chapter enhances real-time multimedia collaboration; however, application sharing should be accompanied by video and audio conferencing to have a better experience. We decided to analyze the current commercial video conferencing tools to understand whether they are good enough or there is a necessity to design new protocols and applications. We only focused on video conferencing due to its higher bandwidth requirements than audio conferencing. We come up with a set of requirements discussed below to properly evaluate existing video conferencing tools.

Adaptation to fluctuating and limited bandwidth conditions When thinking about high-speed Internet, we have to consider many different technologies such as ADSL, cable and satellite. Usually, for all of them the speed of the downlink and the speed of the uplink differ. For example, for ADSL it is common to have downlink speeds of 3 Mb/s and uplink speeds of 1 Mb/s with the uplink speeds always significantly lower than the downlink speeds. For common web applications such as web browsing and video streaming, this does not cause problems as the amount of information sent on the uplink is considerably lower
than the one received on the downlink. Things however are different for applications such as video conferencing, where a client needs not only to receive video but also to send it. In such case, the uplink becomes the bottleneck of the system. To make things worse, other applications running on the same computer or other computers sharing the same link generate cross traffic. The presence of cross traffic creates congestion on both links, with the video stream on the uplink suffering more, given the much lower available bandwidth on the uplink. In other words, when congestion happens, the video stream on the uplink will suffer first, thus determining the quality for the whole video chat. Because of this, in the rest of the chapter we focus our attention on the uplink only.

Changes in bandwidth can significantly affect video and audio quality, with video suffering the most given its higher bandwidth requirements. Usually, congestion happens because bandwidth has to be shared among multiple competing flows. The application has to decrease its transmission rate in order not to create a new congestion or to get rid of an existing congestion completely. Also, it has to increase its transmission speed in order to benefit from available bandwidth when there is no sign of congestion. A video chat application must have this adaptation behavior not only to provide an acceptable video conferencing experience to the users but also to maintain a fair share of bandwidth among flows. To measure their adaptation performance, we changed the available bandwidth programmatically to generate reproducible results. To analyze their fairness to other flows, we introduced HTTP and BitTorrent traffic competing with video and audio traffic.

**Differentiation of congestion losses from wireless losses** The video conferencing application needs to find a way to detect congestion. Usually, a way to do so is through measuring the packet loss and another way is through measuring round-trip time (RTT). When congestion happens, the queues in the routers fill up and at some point overflow. This causes packets to be dropped and lost. A recent article [Gettys, 2011](#) explains that excessive buffering inside the network causes high latency and jitter. Therefore, it is not sufficient to rely on packet loss, at the same time RTT has to be taken into account. The buffers have to be overflow to observe a packet loss while changes in RTT can be detected way earlier. More will be discussed in Section 3.3.
Latency  The end-to-end latency both for audio and video has to be as low as possible to experience a decent video conferencing session. High latency degrades the interactivity of real-time communication [Liang et al., 2003]. ITU-T Recommendation G.114 recommends to keep the mount-to-ear delay for voice over IP less than 150 ms [ITU, 1996].

Frame rate  In order to have a smoother video conferencing experience the frame rate has to be high. For streaming video, participants of a study found the quality acceptable 80% of the time at the frame rate of 6 fps [McCarthy et al., 2004]. Similarly, participants of a video conferencing session found 5 fps usable [Tang, John C., 1992]. Thus, we can conclude that at least 5 frames per second is required for an acceptable video conferencing session.

Video quality  High quality video conferencing is always preferred by users. Quality of a video stream depends on several factors such as image quality, and the video codec in use.

This chapter presents the performance measurement results which address the first two requirements: bandwidth adaptation and loss differentiation. Frame rate and latency measurements will be discussed in the next chapter. We did not measure the video quality of these tools explicitly because it depends on the codecs they have employed. Generally speaking, H.264-based [Wiegand et al., 2003] codecs give better image quality for the same bandwidth than H.263-based ones. The performance measurement results of the first four metrics represents the quality of a video conferencing tool.

During our measurements we tested Skype 4.0.0.215 [bib, 2011b], Windows Live Messenger 14.0.8064.206 [bib, 2009c], Eyebeam version 1.5.19.5.52345 [bib, 2009a] and X-Lite 3.0.47546 [bib, 2009b] due to their popularity and availability.

Among the clients we tested, Skype behaved the best by adapting its codec parameters based not only on packet loss but also on RTT and jitter. This allowed Skype to closely follow the changes in bandwidth without causing any packet loss. Eyebeam performed the worst with high fluctuations in the transmission speed of its video traffic and with poor adaptation to bandwidth fluctuations.

The rest of the chapter is organized as follows. In Section 3.2 we give an overview on the state of the art, Section 3.3 describes the difference between wireless losses and losses due to congestion. In Section 3.4 we present our experimental results and finally Section 3.5 concludes this chapter.
CHAPTER 3. PERFORMANCE OF VIDEO-CHAT APPLICATIONS UNDER CONGESTION

3.2 Related work

Real-time video chat has stricter requirements than streaming video. Popular streaming video websites like Youtube, Hulu, Netflix and Joost use TCP, which has flow and congestion control mechanisms. They buffer the content in the client before playing. The content is stored at more than one bitrate and the most appropriate one is used. Netflix determines the available bandwidth itself whereas Hulu and Youtube allow users to switch to high quality. Several studies propose how to stream video for heterogeneous environments [Ho et al., 2007; Muntean et al., 2007]. Real-time video chat, however, has very strict delay requirements and the retransmission mechanism of TCP does not fit into this model. Because of this, all the measured video chat clients stream over UDP.

In order to avoid congestion or under-utilization of the link the sender needs to adjust its transmission rate. Under-utilization may cause low quality because uplink of the residential area networks are limited. Over-utilization causes unfairness to other traffic as well as packet loss, hence degrading video quality.

A protocol for real-time video conferencing called VTP (Video Transport Protocol) is proposed in Yang et al. [Yang et al., 2006]. Their protocol has a unique end-to-end rate control mechanism utilizing an achieved rate estimation scheme that aims to avoid drastic rate fluctuations while maintaining friendliness to legacy protocols. VTP utilizes a variant of Spike [Tobe et al., 2000a] as its loss discrimination algorithm. We could not measure the performance of the VTP because it requires a complex hybrid testbed consists of both simulated models and real implementations.

The performance of audio chat applications has been studied extensively compared to video chat. Baset and Schulzrinne compared Skype, MSN, Yahoo and Gtalk in terms of audio quality and mouth-to-ear latency [Baset and Schulzrinne, 2006]. Hofeld and Binzenhöfer measured Skype quality and bandwidth adaptation in UMTS [Hofeld and Binzenhöfer, 2008]. Although their work studies performance of Skype under congestion, it only covers audio calls whereas we focus on video calls.

The piece of work closest to ours is [De Cicco et al., 2008; De Cicco et al., 2011]. Here, however, the authors only measure Skype’s video adaptation to bandwidth variations. On the other hand, we cover Skype, Windows Live Messenger, Eyebeam, a commercial SIP-based client and X-Lite, a free SIP-based client. We use Skype 4.0 for Windows which supports high-quality video chat whereas they used Skype 2.0 for Linux.
3.3 Wireless losses vs. congestion losses

Generally, applications consider packet loss a sign of congestion. This is usually true since during congestion queues in the routers fill up and packets get dropped. There are situations, however, in which packet loss is not a sign of congestion. This is true, for example, in a wireless environment. The wireless medium introduces by its own nature losses due to many factors such as signal fading, obstacles and co-channel interference [Forte et al., 2006]. Because of this, an application needs to distinguish between the two kinds of losses. If an application is not capable to distinguish between the two kinds of losses, when a wireless loss occurs, the application will think that the medium is congested and therefore will try to back-off by lowering its sending rate. This, however, will only be counterproductive since there is no congestion and yet the application will experience lower quality due to its lower sending rate. An algorithm that helps in distinguishing between the two types of losses is Spike [Tobe et al., 2000b]. Spike is an end-to-end loss differentiation algorithm (LDA) which is based on relative one-way trip time (ROTT). Spike classifies a loss as congestion related if the loss happens when the ROTT presents a spike.

Figure 3.1: Experimental testbed
3.4 Measurements

3.4.1 Experimental setup

We deployed a small testbed consisting of a desktop PC running FreeBSD 7.1 and two Lenovo Thinkpad X63 laptops running Windows Vista. In order to adjust the available bandwidth we used the desktop PC as a gateway by installing two ethernet cards and by running the dummynet application \cite{Rizzo2010} to emulate a cross country Internet link. By using the dummynet application we were able to adjust many different network parameters such as queue sizes, RTT, maximum bandwidth and random packet loss. Figure 3.1 shows the setup for the experiments. All PCs were connected to the internet and all traffic between sender and receiver was going through the PC running dummynet. The two laptops were used as IM clients, that is were running a video chat. One desktop PC running FreeBSD and Dummynet was used as gateway. All machines used as IM clients were also running Wireshark \cite{bib2011d} in order to collect and later analyze packet flows.

3.4.2 Results

We wanted to emulate a video-chat session between two ADSL users located on either coast of the United States. To emulate this type of network we set the total RTT value to 114 ms, queue size to 60 kB \cite{Dischinger2007} and the maximum available bandwidth to 3 Mb/s for the downlink and 1 Mb/s for the uplink.

We performed three sets of experiments. In the first set, we analyzed how video conferencing applications adapt to changes in bandwidth. In the second set, we measured the impact of cross traffic, either HTTP or BitTorrent. At last, we observed the impact of wireless losses.
3.4.2.1 Changes in bandwidth

Available bandwidth of a video conferencing application fluctuates during the session due to traffic from other applications both running on the same computer and on other computers sharing the same link. For mobile users, the available bandwidth may change depending on the wireless network signal strength. To cover all these different situations, in our experiments we changed the available bandwidth programmatically to generate reproducible results. We also changed the available bandwidth introducing cross traffic from network applications by sharing a file over peer-to-peer networks or uploading a file to a hosting service. In the first set of experiments, we modify the available bandwidth by following a step function. We consider two step functions. The first step function decreases and increases the available bandwidth of 80 kb every 10 seconds, while the second one has decreases and increases of 400 kb every 10 seconds.

Figures 3.2, 3.3, 3.4 and 3.5 show the measurement results for Skype, Live Messenger, Eyebeam and X-Lite, respectively, when the first step function is used. We also show what different video and audio codecs were used and how the applications changed codecs depending on the congestion level.

In the following we describe in more detail the behavior of each IM client.

X-Lite does not support H.264 [Wiegand et al., 2003] for video, but it rather uses H.263 [h26, 1998] which has poorer quality compared to H.264. From our measurements, we have seen that it has a minimum bitrate of 180 kb/s as it does not go below such value even when it experiences 100% loss. When congestion happens, even though it experiences 100% packet loss, it does not stop the video. It tries to recover from a congestion situation by using Forward Error Correction (FEC) for audio. This however, does not help much as it contributes to increasing the level of congestion by increasing the packet size. Finally, X-Lite does not drop the call even though the audio quality is very poor.

Generally speaking, a good video-chat application, when in a congested state, should drop the video stream in order to preserve the audio stream as much as possible. Furthermore, if congestion is so high that even the audio stream is severely affected, then it should drop the call. This would help in lowering the overall level of congestion and it would not represent a big penalty for the user since the quality of the call was extremely poor.
CHAPTER 3. PERFORMANCE OF VIDEO-CHAT APPLICATIONS UNDER CONGESTION

Figure 3.2: Skype with 10-10 step function

Figure 3.3: Windows Live Messenger with 10-10 step function
CHAPTER 3. PERFORMANCE OF VIDEO-CHAT APPLICATIONS UNDER CONGESTION

Figure 3.4: Eyebeam with 10-10 step function

Figure 3.5: X-Lite with 10-10 step function
In terms of bandwidth, X-Lite decreases its transmission speed gradually. Unfortunately, once congestions stops and more bandwidth becomes available, X-Lite does not increase its transmission speed.

Eyebeam uses the H.264 video codec. Similarly to X-Lite, it tries to use FEC when it detects high congestion while still trying to keep both video and audio streams. In other words, it does not try to disable video in order to keep the audio quality to an acceptable level. Furthermore, it seems to support only two different bitrates for video. This is insufficient to support the different levels of congestion. Eyebeam presents much higher fluctuations in transmission speed than X-Lite, due perhaps to the implementation of the H.264 codec. Such fluctuations translate in higher losses when the available bandwidth starts decreasing since the peaks of such fluctuations exceed the maximum available bandwidth. Furthermore, once the available bandwidth starts increasing again, similarly to X-Lite, the transmission speed does not increase, staying steady at the lower speed, thus never reaching the original level.

Figure 3.6: Windows Live Messenger behavior when decreasing/increasing the available bandwidth according to 10-50 step function
Skype behaves differently than the other IM clients. In particular, it promptly adapts its transmission rate to changes in bandwidth, thus preventing packet loss until the minimum bitrate is reached at which point it drops the call. Skype has this behavior because it uses other metrics on top of packet loss in order to detect congestion. It seems that parameters such as RTT and jitter are taken into account. Cicco et al observed that Skype decreases its sending rate when congestion is present on the reverse path [De Cicco et al., 2011]. They believed that reduction in sending speed seems to be triggered by the increased RTT on the reverse path. In particular, we can see from Figure 3.2 that as the available bandwidth goes down, the transmission speed follows it closely, avoiding packet loss. On the other hand, for the other IM clients packet loss starts much earlier since in order to detect congestion they need to “see” some packet loss. Also, the way other IM clients lower their transmission rate is much more aggressive.

While Skype reacts to congestion by trying to closely match the available bandwidth, Windows Live Messenger drops its transmission speed drastically when it detects congestion. In particular, it drops the video transmission rate and then slowly tries to increase it again. No action is taken on the audio flow. When it reaches very low bitrates, it completely disables the video and it adds FEC to the audio trying to preserve the audio stream as much as possible. A disadvantage of Live Messenger compared to Skype is that its minimum audio bitrate is 50 kb/s which prevents it from operating at very low bitrates. Skype audio codec, on the other hand, can operate at bitrates as low as 16 kb/s. In terms of bandwidth, Live Messenger decreases its transmission rate with the available bandwidth. As the available bandwidth increases, the transmission speed for Live Messenger increases very slowly taking up to 9 minutes to reach the original value.

According to our subjective observations, Skype and MSN when in congested state decrease video frame rate and quality, showing an almost-still image with few artifacts as there is no or little packet loss. On the other hand X-Lite and Eyebeam try to keep their frame rate and quality high, showing a smoother video but with lots of artifacts due to higher packet loss. We believe that in terms of end-user experience, the first approach is better. Low-quality and low frame rate video without artifacts gives a better user experience than high frame rate video with lots of artifacts.
Lastly, video chat applications should be able to lower their video bitrate to very low levels in order to keep the audio at an acceptable level. Video codecs should be able to adapt to changes in bandwidth by supporting any requested bitrate. Such behavior we have seen it only in Skype while the other video-chat applications support only a few fixed bitrate levels.

When the second step function is used, all IM clients behave similarly to the case of the first step function. In this case, however, Live Messenger seems to be performing best by quickly adapting to the sudden change in bandwidth (see Figure 3.6). In particular, MSN monitors the packet loss ratio and if it sees a very high packet loss then it drastically drops its transmission rate and when more bandwidth becomes available, it increases its transmission rate very slowly. However, if Live Messenger sees low packet loss, it still drops its transmission rate to a very low level but this time it tries to increase it back to its original value in a very short time.

Eyebeam performs worst as it lowers its transmission speed only after the bandwidth has increased back to its original value. Still, both Eyebeam and X-Lite do not increase back their transmission speeds once the available bandwidth is back to its original value.
CHAPTER 3. PERFORMANCE OF VIDEO-CHAT APPLICATIONS UNDER CONGESTION

3.4.2.2 Measuring TCP friendliness by introducing HTTP as cross-traffic

We introduced HTTP cross-traffic to measure how TCP friendly these four video conferencing applications to other TCP applications. We use Dummynet to restrict the uplink bandwidth to 1 Mb/s. For generating HTTP traffic on the uplink, we uploaded a 9 MB file to a web-hosting service called Media Fire [bib, 2011a].

Figures 3.7, 3.8, 3.9 and 3.10 show how the bandwidth is shared between the audio-video traffic and the HTTP traffic for Skype, Live Messenger, Eyebeam and X-Lite, respectively.

Eyebeam and X-Lite show a similar behavior; this is not surprising given that they are both products of the same company. In particular, both do not adjust their transmission rate at all, keeping it steady at the same value it had before the competing traffic was introduced. As a consequence, bandwidth is shared in a more or less fair way between audio-video traffic and HTTP traffic. Packet loss is higher for Eyebeam than for X-Lite because Eyebeam’s transmission rate fluctuates. As mentioned earlier, such fluctuations are due to the video codec Eyebeam uses and its implementation. Such heavy fluctuations cause spikes in transmission rate which translate to spikes in used bandwidth, that is spikes in packet loss. It is curious to notice how the free version of Eyebeam, that is X-Lite, does not present such spikes. This is due to the fact that X-Lite is using a different video codec, H.263.

Skype adapts to the presence of other traffic by lowering its transmission rate. As we can see from Figure 3.7, the very good adaptability of Skype allows it to generate very low packet loss. Unfortunately, in doing so, Skype will always be penalized as the bandwidth that is not used by Skype is consumed by the HTTP traffic.

Live Messenger, similar to Eyebeam and X-Lite, does not lower its transmission rate, keeping it steady. On one hand this prevents HTTP from using most of the available bandwidth, on the other hand it causes higher packet loss.
CHAPTER 3. PERFORMANCE OF VIDEO-CHAT APPLICATIONS UNDER CONGESTION

Figure 3.7: Skype bandwidth, delay and loss in the presence of concurrent HTTP traffic

Figure 3.8: Windows Live Messenger behavior in the presence of concurrent HTTP traffic
CHAPTER 3. PERFORMANCE OF VIDEO-CHAT APPLICATIONS UNDER CONGESTION

Figure 3.9: Eyebeam behavior in the presence of concurrent HTTP traffic

Figure 3.10: X-Lite behavior in the presence of concurrent HTTP traffic
3.4.2.3 BitTorrent as cross-traffic

In this section we discuss our experimental results when introducing BitTorrent traffic during a video chat. For the BitTorrent traffic, we used the Vuze [bib, 2011c] BitTorrent client and did not limit its maximum upload speed.

Figures 3.11, 3.12, 3.13 and 3.14 show how the IM clients adapt to the presence of BitTorrent traffic. Eyebeam and X-Lite react in the same way as in the HTTP case. They both keep the same transmission rate with and without BitTorrent traffic. The fluctuations in transmission rate in Eyebeam, cause a higher loss rate than in X-Lite. It is interesting to note that since X-Lite and Eyebeam do not lower their transmission rate, they will prevent BitTorrent traffic from consuming more bandwidth.

On the other hand, Skype lowers its transmission rate as soon as BitTorrent traffic is introduced (see Figure 3.11). This limits the losses of video and audio traffic, however BitTorrent traffic will take most of the available bandwidth. Once the BitTorrent traffic stops, Skype goes back to its initial transmission speed fairly quick. The reason why Skype lowers its transmission rate considerably more than with HTTP traffic, is because the BitTorrent client opens several concurrent TCP connections taking in our experiments about 85% of the available bandwidth. We observed more than 20 concurrent TCP connections.

Live Messenger behaves differently than in the HTTP case. When BitTorrent traffic is introduced, Live Messenger lowers its transmission rate significantly, leaving almost all the available bandwidth to the BitTorrent traffic. This is because in this case the amount of cross traffic is much higher than in the HTTP case, therefore the amount of packet loss is also higher. This causes Live Messenger to lower its transmission rate considerably. In the HTTP case, the amount of cross traffic and therefore packet loss was considerably smaller, thus leaving its transmission rate the same. As mentioned before, once the cross traffic is removed, Live Messenger takes a long time to go back to its original transmission rate.

Ideally, the desired behavior would be an equal share of bandwidth between different video chat and other application with small packet loss. This makes Eyebeam, X-Lite and Windows Live Messenger all better than Skype from this point of view. By lowering its transmission speed, Skype just frees bandwidth which is then taken by other flows, leaving the level of congestion unchanged.
CHAPTER 3. PERFORMANCE OF VIDEO-CHAT APPLICATIONS UNDER CONGESTION

Figure 3.11: Skype behavior in the presence of concurrent BitTorrent traffic

Figure 3.12: Windows Live Messenger behavior in the presence of concurrent BitTorrent traffic
CHAPTER 3. PERFORMANCE OF VIDEO-CHAT APPLICATIONS UNDER CONGESTION

Figure 3.13: Eyebeam behavior in the presence of concurrent BitTorrent traffic

Figure 3.14: X-Lite behavior in the presence of concurrent BitTorrent traffic
CHAPTER 3. PERFORMANCE OF VIDEO-CHAT APPLICATIONS UNDER CONGESTION

3.4.2.4 Wireless losses

Packet losses degrade the quality of a video chat significantly. This is especially true with modern codecs like H.264 as there is a high correlation between frames. Therefore, a bandwidth adaption algorithm should try to eliminate packet losses by decreasing its transmission rate in case of congestion. However, not all losses are due to congestion, wireless networks introduce wireless losses due to signal fading, interference and channel quality. Decreasing transmission rate will not help in case of non-congestion related losses. Some other techniques like FEC and retransmissions can be utilized. However, in order to respond to losses an application should differentiate congestion losses from random ones.

In these measurements we wanted to see how the various IM clients behaved to wireless losses and in particular, to see if they could differentiate wireless losses from congestion losses.

Figure 3.15: Skype behavior in the presence of wireless losses
We introduce 1% random packet loss using Dummynet and consider two scenarios. In one we introduce packet loss throughout the video-chat session, in the other one we introduce packet loss in the middle of the video-chat session. Figure 3.15 shows our results for Skype in these two scenarios. The top graph refers to wireless losses introduced in the middle of the video-chat session while the graph on the bottom refers to wireless losses introduced throughout the video-chat session.

In both scenarios Eyebeam and X-Lite do not change their rate.

Skype on the other hand behaves differently. When all the losses are introduced in the middle of the chat session (see Figure 3.15), Skype reacts increasing its transmission bitrate by about 20%. This is different from the case when losses are due to congestion as in that case Skype decreases its transmission rate. We believe such increase in transmission rate is not due to retransmissions triggered by the losses as the increase in transmission rate would then be on the same order of the wireless losses, that is 1%. It seems that Skype can distinguish between congestion losses and wireless losses by monitoring packets delay. In case of wireless losses packet delay does not change while in case of congestion losses packet delay spikes \cite{Tobe et al., 2000b}. Due to closeness of Skype protocol we really don’t know what is happening, but we guess that Skype adds FEC in case of wireless losses.

When wireless losses are introduced throughout the video chat session (see Figure 3.15), Skype increases its transmission speed gradually. This is different from its usual behavior of reaching full transmission speed almost immediately.
3.5 Conclusions

We built a testbed in order to analyze the behavior of four popular video IM clients, focusing on the video-chat feature and on how such clients react to changes in bandwidth due to congestion. We analyzed the behavior of Skype, Live Messenger, X-Lite and Eyebeam. As competing traffic, we considered both HTTP traffic and BitTorrent traffic.

We found that Skype adapts gradually to changes in bandwidth, reacting to both increases and decreases in bandwidth. Because Skype appears to monitor also RTT and jitter on top of packet loss, usually it can adapt its transmission speed before packet loss occurs. Live Messenger drops its transmission rate drastically when packet loss is detected and increases its transmission rate very slowly when there is available bandwidth. Because of this, Live Messenger performs best when drastic drops in available bandwidth happen. On the other hand, however, it does take an extremely long time to raise its transmission rate back to its pre-congestion rate.

X-Lite and Eyebeam do not change their transmission speed when cross traffic is present, which makes them less sensitive to the presence of BitTorrent traffic. When the available bandwidth decreases, they decrease their transmission speed. Unfortunately, once more bandwidth becomes available, both X-Lite and Eyebeam do not increase their transmission rate. Finally, Eyebeam experiences strong fluctuations in transmission rate due to the codec used and its implementation. These fluctuations are not present in X-Lite and cause higher packet loss when spikes in transmission speed occur.

Due to limited upstream bandwidth, video clients must have bandwidth adaptation mechanisms and must be able to differentiate between wireless losses and congestion losses. On the basis of our measurement results we believe that Skype meets these two requirements.

One question to answer is whether there is an optimal adaptation behavior and if there is one is it possible to implement it in practice? Presumably, this would mean tracking available fair-share bandwidth instantaneously. In our cross-traffic measurements, video chat applications compete with a single HTTP stream or with a bunch of TCP streams originating from BitTorrent client over a 1Mb/s shared link. According to max-min fairness [Jaffe, 1981] metric, video chat application should not lower its transmission rate encountering with a HTTP or BitTorrent cross-traffic as long as they consume less than 500kb/s. If
video chat application uses more than 500kb/s as in the case of Microsoft Live Messenger, the video chat application should lower its rate to equal-share (500kb/s). We assume application level equality not a single TCP stream level equality. Otherwise, BitTorrent would not leave any bandwidth to other applications due to its 20 concurrent TCP connections.

According to mentioned guidelines, Skype did not utilize its fair-share bandwidth in case of HTTP cross-traffic. Figure 3.7 shows that Skype utilized 20% of the available bandwidth (200kb/s) while the HTTP stream consumed the rest (80%). Although Windows Live Messenger initially decreased its rate to 150kb/s encountering with HTTP traffic, later it increased its transmission rate to its fair-share bandwidth (Figure 3.8). By not changing their transmission rate at all (~400kb/s) Eyebeam and X-Lite utilized their max-min fair-share bandwidth in case of a 1Mb/s available bandwidth. However, if the available bandwidth was 400kb/s they will fail to utilize their max-min fair-share bandwidth. Similarly, both Skype and Windows Live Messenger failed to utilize their max-min fair-share in case of BitTorrent cross-traffic by lowering their bandwidth usage to %15 and leaving the rest %85 to BitTorrent.

Due to excessive buffering inside the network taking only packet losses into account is not sufficient, delay-based flow control mechanisms which take RTT into account should be considered, too.
Chapter 4

vDelay: a novel tool to measure capture-to-display latency and frame rate

4.1 Introduction

Performance measurement results of popular video conferencing applications under congestion are discussed in the previous chapter. During the measurements, we realized the necessity of a tool to measure the capture-to-display latency and frame rate of a real-time video conferencing session. These two metrics show the quality of the video conferencing session. A fine-grade video communication requires low latency and high frame rates. The latency and frame rate results measured by our novel tool combined with the adaptation to bandwidth changes performance and loss differentiation capability results from previous chapter will reveal the overall quality of a video conferencing application. Thus, this work complements our previous measurement study.

Real-time video chat applications augment the communication experience of participants by allowing them to see other participants in addition to having an audio conversation. These applications have three key software components: a video encoder that compresses the video captured from the camera, a video decoder that decompresses the video received
over the network, and a playout buffer that smooths the playout of received video due to network delay variations. These software components impact capture-to-display latency (CDL) and frame rate of the real-time video played at a receiver application. Capture-to-display latency is the total time to encode and decode a video frame, playout buffer time, and latency of the network path. Along with bitrate, these two metrics provide quick insights into the performance of a real-time video application. Developers and testers can use these metrics to determine whether the experimental performance of a video chat or conferencing application meets the expected performance. Moreover, since numerous video chat applications are available today, users can use the CDL and frame rate metrics to guide their selection of a video chat application.

I developed vDelay, a Java-based operating system independent tool to measure the capture-to-display latency (CDL) and frame rate of a video stream. The video stream is captured at the caller user agent and displayed at a callee user agent. Both caller and callee user agents run the same video application. vDelay does not require any change in the source code or executable of a real-time video application. Thus, it can be used to measure the CDL and frame rate of closed source video applications. Also, vDelay does not require any specialized hardware.

This chapter is organized as follows. Section 4.2 discusses issues involved in measuring CDL and frame rate. Section 4.3 presents the architecture of vDelay. Section 4.4 describes the experimental setup and Section 4.5 presents CDL and frame rate results for video chat applications. Section 4.7 discusses related work.

4.2 Measuring CDL and frame rate

For the rest of this chapter, we assume that a video session is established between two participants. We designate one machine running the video chat application as a caller user agent and the other as a callee user agent. For simplicity, we refer to these machines as caller and callee, respectively.

The key to measuring capture-to-display latency (CDL) and frame rate lies in embedding a timestamp in the caller's video, and retrieving that timestamp at the callee. The
timestamp is the current system time at the machine running the caller user agent. Assuming that the machines running the caller and callee user agents are time synchronized within an acceptable error, the capture-to-display latency is the difference between the timestamp retrieved from the caller’s video and current system time at the machine running the callee user agent. This difference can also be used to calculate the inter-frame display time at the callee user agent. Further, since every new frame must have an increasing value of a timestamp, the number of frames within a time period can be used to calculate the frame rate of the received video.

We use a trick to embed the timestamp in the caller’s video that does not require any change to the video chat application. The timestamp, i.e., the current system time at the machine running the caller user agent, is displayed at the monitor of the machine running the caller user agent every $t$ time units. In our case, the monitor is a liquid crystal display (LCD) device. A webcam is attached to the machine running the caller user agent and faces the LCD monitor. Thus, it captures the current image on the LCD monitor which includes the timestamp. The caller user agent then encodes this captured frame including the timestamp, and sends it over the network to the callee user agent which decodes the frame and displays it on its attached LCD monitor. An application running on the same machine as the callee user agent grabs the timestamp from the received frame, and calculates the time difference between the timestamp grabbed from the received frame and local system time. The timestamps are processed to calculate CDL and frame rate. Figure 4.1 shows the setup for measuring CDL and frame rate. The novelty of this approach lies in the fact that no additional hardware is needed and no modification to the software of any real-time video application is required.

Next, we discuss how to display and retrieve a timestamp at/from a LCD, and the factors that impact the latency and accuracy of the received timestamp.
4.2.1 How to display and capture a timestamp?

We considered three approaches for displaying the timestamp at the caller user agent. These approaches display the timestamp as (1) an EAN-8 barcode \cite{ean, 2011}, (2) numeric characters, and (3) a progress bar. From experimentation, we found that displaying timestamp as a barcode was the most attractive option for two reasons: (1) barcodes such as EAN-8 and EAN-13 have a built in checksum mechanism and (2) barcode reading is very fast. The checksum is necessary because a barcode image can get distorted due to bandwidth variations and lossy encoding of video codecs. Without a checksum, it is difficult to ascertain whether the timestamp grabbed from the frame is the same as the one displayed at the caller user agent. No built in checksums exist for timestamps displayed as numeric characters and as a progress bar. Although it is conceivable to design checksums for both numeric characters and a progress bar, we did not feel a need since the reading accuracy of the timestamp encoded as a barcode was above 94\% for a range of video chat applications (see Section 4.5).

During our experiments we found that at the callee, barcode image could be read in less than a millisecond, facilitating the online calculation of capture-to-display latency and frame rate, whereas it took a few seconds to recognize via OCR the timestamp displayed as numeric characters. For these reasons, we have used barcodes to encode and retrieve the timestamp from a video frame.
4.2.2 Factors impacting the embedding and retrieval of timestamp

Below, we discuss the factors that impact the embedding and retrieval of a timestamp. These are LCD refresh rate and response time, camera aperture, shutter speed, and timestamp area in the captured video. The first three impact the capture-to-display latency and frame rate calculations whereas the last factor impacts the success rate of retrieving barcodes at the machine running the callee user agent. For the rest of this chapter, our use of the timestamp means the current system time displayed as a barcode on the LCD monitor of the machine running the caller user agent. In Section 4.5, we describe how a timestamp is encoded as a barcode.

4.2.2.1 Refresh rate and response time of a LCD

The refresh rate determines the speed at which an image is displayed on the LCD monitor and typically starts at 60Hz on modern LCD monitors. This means that it may take up to 16.6 ms for a timestamp to appear on a LCD monitor. The response time is the amount of time it takes a pixel to refresh itself, i.e., ready itself for displaying the new pixel. The response time on modern LCD monitors is typically 5 ms. Together, refresh rate and response time can delay the displaying of timestamp by a few milliseconds. There is no way to measure the response time of an LCD programmatically. LCD testers use physical detectors to measure this value, therefore it is not possible to subtract this value to correct the reported CDL result.

To eliminate the impact of refresh rate and response time, a virtual webcam driver can be designed which grabs the frame from the frame buffer and passes it to the caller user agent for encoding. However, the design of such a virtual webcam driver is tightly coupled with the underlying operating system. Thus, we have not explored this approach.

4.2.2.2 Aperture

Aperture is a hole in the camera through which light enters the camera. If the hole is narrow, less light enters the camera and the captured image containing the barcode is likely to be dark. Consequently, it may be necessary to keep the camera shutter open for a longer period to capture the barcode being displayed on the LCD monitor. Keeping the shutter open for
a long period adds delay to the capture-to-display latency. Therefore, it is necessary to set the camera aperture to its highest value to minimize the length of the period the shutter is open.

4.2.2.3 Shutter speed

Shutter speed is the duration of time for which the shutter of a camera is open. If a shutter remains open when multiple timestamps are being displayed on a LCD monitor, the camera will capture all of these timestamps, and it may be difficult to retrieve them at the receiver. Thus, shutter speed must be greater than the refresh rate of a LCD monitor. Further, a low shutter speed can impact the frame rate of the received video. Therefore, it should be set to a value that adds the least delay to the capture-to-display latency and maximizes the achievable frame rate.

4.2.2.4 Timestamp area in the captured video

Video chat applications capture the video at different resolutions such as 640x480 and 320x240. The timestamp should occupy a sufficient area in the captured frame to maximize the successful reading of barcodes at the callee machine. Through trial and error, we determined the minimum area that a timestamp encoded as an EAN-8 barcode should occupy in the captured video frame. Table 4.1 shows these values measured as a ratio of the video stream. These results may be adjustable depending on the quality of a webcam, and the barcode reader. Nevertheless, they provide a useful guideline for conducting similar experiments.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Size of timestamp w.r.t resolution of video stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>320x240</td>
<td>1/4</td>
</tr>
<tr>
<td>640x480</td>
<td>1/16</td>
</tr>
<tr>
<td>800x600</td>
<td>1/24</td>
</tr>
<tr>
<td>1024x768</td>
<td>1/40</td>
</tr>
</tbody>
</table>

Table 4.1: The size of timestamp encoded as an EAN-8 barcode w.r.t resolution of captured video.
CHAPTER 4. VDELAY: A NOVEL TOOL TO MEASURE CAPTURE-TO-DISPLAY LATENCY AND FRAME RATE

4.3 vDelay architecture

The vDelay tool consists of a vDelay-S and vDelay-R Java application that run on the caller user agent machine and callee user agent machine, respectively. The vDelay-S application displays the system time as an EAN-8 barcode on the LCD monitor. There are three related issues. First, the system time must be displayed to the resolution of a millisecond to accurately measure capture-to-display latency. Since an EAN-8 barcode can only represent a maximum of eight digits, an EAN-8 barcode can at most capture the eight least significant digits of the system time measured in milliseconds. Second, through experimentation, we found that generating barcodes every millisecond was not computationally efficient, so we generated the barcode images in advance. Based on the least significant digits of the system time, the vDelay-S application selects the appropriate barcode image and displays it on the screen. Lastly, an EAN-8 barcode image can represent a numeric range between zero and 10 million, so it might be necessary to generate these many barcode images. However, we only generated 10,000 EAN-8 barcode images using barcode4j [bar, 2011], an open source barcode generator, that represent the numeric range [0, 9999]. Depending on the last four digits of the system time measured in milliseconds, the vDelay-S application displays the barcode image that represents those digits and displays them on the LCD monitor. This numeric range implies that after 10 seconds, the same barcode image is displayed on the screen. As discussed in Section 4.2.2.1, the displaying of barcodes on the LCD monitor is delayed by few milliseconds depending on the refresh rate and response time of a LCD monitor. Figure 4.3 presents the architecture of vDelay.

![Figure 4.2: vDelay-R architecture.](image)

Figure 4.2 shows the block diagram of vDelay-R application. To facilitate grabbing the barcode from the received frame, the vDelay-R application lets user select the (top, left) and (bottom, right) screen coordinates of the barcode image with mouse clicks. The vDelay-R
application then grabs the barcode image from the frame-buffer every five milliseconds, and passes it to a barcode reader. The time to grab the barcode image from the frame is less than a millisecond. We have used zing [zxi, 2011], an open-source barcode reader. The vDelay-R application reads the barcode, retrieves the timestamp, and computes the difference between the local system time and the timestamp retrieved from the frame. It then computes the capture-to-display latency and frame rate and outputs them to the LCD monitor and writes them to a file. It also computes the first read rate (FRR) of barcodes and writes it to the display and a file. Figure 4.4 shows a screen shot of vDelay-R application.

Kato et al. [Kato and Tan, 2007] define FRR as:

$$\text{FRR} = \frac{\text{Number of successful first reads}}{\text{Number of attempted first reads}}$$

A timestamp from a single frame can be grabbed multiple times depending on the the instant at which the frame is grabbed from the screen. However, the vDelay-R application reports the difference between the earliest grabbed timestamp from a frame and the current system time.
CHAPTER 4. VDELAY: A NOVEL TOOL TO MEASURE CAPTURE-TO-DISPLAY LATENCY AND FRAME RATE

Figure 4.4: Screen shot of vDelay-R application. FPS, CDL, and FRR statistics are shown at the top of the image. The barcode received from the caller user agent is also visible.

4.4 Clock synchronization

vDelay tool assumes that clocks are synchronized between the machines running the video chat applications. With a minor adjustment, vDelay tool can be used to calculate the capture-to-display latency when clock synchronization may not be possible. The idea is that the callee user agent reflects the video containing the timestamp back to the caller user agent. To reflect the video without requiring any change to the video chat application, the webcam attached to the machine running the callee user agent points towards the LCD monitor of the callee machine which displays the video received from the caller that also contains the timestamp. The vDelay-R application is run at the caller user agent which retrieves the timestamp from the frame received from callee. vDelay-R then compares the timestamp with its current time to calculate the time elapsed since the frame was sent from caller to callee user agent. This elapsed time includes the round-trip network delay, and video encoding, decoding, and playout time at the caller and callee user agent. Assuming the round-trip network delay is negligible as it would typically be in a laboratory LAN, one half of this elapsed time approximately gives the capture-to-display latency.
CHAPTER 4. VDELAY: A NOVEL TOOL TO MEASURE CAPTURE-TO-DISPLAY LATENCY AND FRAME RATE

4.5 Experimental setup

Figure 4.1 shows the experimental setup we used for our measurements. It consists of two machines, each with an Intel Core Duo processor and a Dell 1909W flat panel display [del, 2011]. The brightness on the LCD monitors is set to its maximum value. Both machines run the Windows Vista operating system and are connected to the same subnet (RTT < 1 ms). The time on both machines is synchronized through NTP and the NTP server query interval was 10 seconds. Each machine runs a video chat application. A Logitech Quickcam Pro 9000 webcam [log, 2011] is attached to the machine running the caller user agent and points towards the LCD monitor displaying the timestamp. A video session is established between two user agents. The caller user agent sends the captured video including the timestamp encoded as barcode over the network to the network to the callee user agent.

The webcam attached to the caller user agent captures the images on the LCD monitor. These images include icons and desktop applications along with the timestamp. It can be argued that video chat applications are optimized for human images and not the computer displays, and thus the statistics obtained for CDL and frame rate may not reflect the common use case for these applications. To address this, we prerecorded a video of a human user sitting in front of a webcam and run it on the machine running the caller user agent. The vDelay-S application displays the timestamp at a corner of this prerecorded video of a human user. The webcam connected to the machine running the caller user agent sender points to the monitor and captures the prerecorded video of a human user and the timestamp, thereby mimicking a realistic video chat session.

At the machine running the callee user agent, we run the vDelay-R Java application after the video call has been established and the caller’s video along with the barcode is visible at the LCD monitor. We run the video session for 10 minutes and report the CDL and frame rate results for this duration.

It is possible that the only webcam available is the one attached to the display of a machine (such as LCD of a laptop) and cannot be detached. Thus, it cannot be pointed towards LCD monitor displaying the timestamp. To resolve this, a mirror can be placed in front of the LCD monitor of the machine running the caller user agent. The camera attached to the top of the LCD monitor can capture the timestamp being displayed in the mirror.
4.6 Results

We used vDelay tool to measure capture-to-display latency and frame rate of Skype [sky, 2011], Windows Live Messenger [bib, 2009c], Yahoo Messenger [yah, 2011a], GMail video chat [gvi, 2011], AOL Instant Messenger (AIM) [aol, 2011], X-Lite [bib, 2009b], eyeBeam [bib, 2009a], and Tokbox [tok, 2011] applications. We used the setup described in Section 4.5. For all the video chat applications, we ran the experiment for ten minutes and repeated it twice. The Tokbox application completely runs in browser and only depends on the availability of an Adobe Flash player. In Tokbox, the caller user agent sends packet over TCP to a Flash server maintained by Tokbox which forwards these packets to the callee user agent over TCP and vice versa. With the exception of Tokbox, the caller user agent sends packets directly to the callee user agent. Besides Tokbox and Yahoo Messenger, all the video applications send packets over UDP. For Skype, the video session was of high quality (HQ) as indicated by an icon in the received video.

<table>
<thead>
<tr>
<th>Chat application</th>
<th>Version</th>
<th>Video codec</th>
<th>Resolution</th>
<th>Bitrate (kb/s)</th>
<th>Fps</th>
<th>CDL (ms)</th>
<th>Std. dev (ms)</th>
<th>Encoding CPU (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live Messenger</td>
<td>14.0.8064.206</td>
<td>H.264</td>
<td>640x480</td>
<td>600</td>
<td>23</td>
<td>69</td>
<td>16</td>
<td>28</td>
</tr>
<tr>
<td>Gtalk</td>
<td>v1.0.8.0</td>
<td>H.264</td>
<td>512x300</td>
<td>1000</td>
<td>27</td>
<td>99</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>X-Lite</td>
<td>3.0.47546</td>
<td>H.263+</td>
<td>320x240</td>
<td>400</td>
<td>27</td>
<td>102</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>Yahoo</td>
<td>9.0.0.2152</td>
<td>Unknown</td>
<td>320x240</td>
<td>72</td>
<td>3</td>
<td>113</td>
<td>23</td>
<td>1</td>
</tr>
<tr>
<td>eyeBeam</td>
<td>1.5.19.5.52345</td>
<td>H.264</td>
<td>640x480</td>
<td>400</td>
<td>27</td>
<td>129</td>
<td>16</td>
<td>25</td>
</tr>
<tr>
<td>AIM</td>
<td>6.8.14.6</td>
<td>Unknown</td>
<td>240x180</td>
<td>120</td>
<td>9</td>
<td>147</td>
<td>57</td>
<td>20</td>
</tr>
<tr>
<td>Tokbox (LL)</td>
<td>2.01.2351d05</td>
<td>Unknown</td>
<td>270x200</td>
<td>320</td>
<td>24</td>
<td>148</td>
<td>72</td>
<td>25</td>
</tr>
<tr>
<td>Skype (HQ)</td>
<td>4.0.0.215</td>
<td>VP7</td>
<td>640x480</td>
<td>560</td>
<td>20</td>
<td>238</td>
<td>22</td>
<td>44</td>
</tr>
<tr>
<td>Tokbox (HL)</td>
<td>2.01.2351d05</td>
<td>Unknown</td>
<td>270x200</td>
<td>320</td>
<td>23</td>
<td>342</td>
<td>69</td>
<td>25</td>
</tr>
</tbody>
</table>

Table 4.2: Comparison of video chat applications. The results are sorted by capture-to-display latency (CDL). The ‘G’ in the Resolution column is our best guess of the video resolution. LL, HL and HQ are abbreviations for low latency, high latency, and high quality.

Table 4.2 shows the results of these video applications. The reported results include capture-to-display latency (CDL), standard deviation of CDL, frame rate, and first read rate (FRR) measured using vDelay, and bitrate and CPU utilization of the caller user agent that encodes the video. The results are sorted by capture-to-display latency. For ease of
CHAPTER 4. VDELAY: A NOVEL TOOL TO MEASURE CAPTURE-TO-DISPLAY LATENCY AND FRAME RATE

Figure 4.5: (a) Capture-to-delay latency (CDL) (b) Standard deviation of capture-to-display latency (c) First read rate (FRR) (d) Frames per second (fps) (e) Bitrate (f) CPU utilization for video encoding.

comparison, we also graphically show these results in Figure 4.5. As mentioned before, Tokbox forwards packets from a caller user agent to a callee user agent through servers which are based in different geographical locations. The use of a server in different location impacts the CDL. Therefore, we report the minimum and maximum observed CDL for Tokbox which are abbreviated as LL (low latency) and HL (high latency) in Table 4.2.

Our results indicate that amongst all video chat applications, Windows Live Messenger has the best CDL value. For Tokbox (LL), Tokbox (HL), and AIM, the standard deviation of CDL is more than 50 ms. We conjecture that Tokbox has a high standard deviation for CDL due to the packet scheduling at the server relaying media packets. For AIM, we attribute the high standard deviation to the video encoding function. Figure 4.7 compares video chat applications in terms of CDL.

X-Lite and eyeBeam have the highest achieved frame rate per second (fps). Except for Yahoo Messenger and AIM, the frame rate of all video chat applications is above 20 frames per second. Figure 4.8 compares video chat applications in terms of FPS. As for the CPU utilization of the machine running the caller user agent, we measured that Skype uses
vDelay can be used to measure CDL and frame rate of a video chat application under controlled network conditions. Such use provides a powerful testing mechanism for application developers. One instance is shown in Figure 4.6, which shows the performance of Skype when the available bandwidth of a video session is adjusted as a step function. The figure shows that Skype suffers from a high jitter in frame rate as the available bandwidth is gradually decreased. With the decrease in available bandwidth, CDL starts to increase indicating the impact of network queuing and playout buffer adjustments. The CDL graph shows large spikes when available bandwidth is below 400 kb/s. However, the CDL of Skype is close to its operating mean when the available bandwidth is above 400 kb/s.
Figure 4.7: Comparison of video chat applications in terms of latency.

Figure 4.8: Comparison of video chat applications in terms of fps.
4.7 Related work

Existing video latency measurement tools involve the use of a specialized hardware. OmniView [omm, 2011] is a tool that uses a specialized PCI card. Our goal is to measure video latency without the use of any specialized hardware.

Yoshimura et al. [Yoshimura and Masugi, 2004] designed a module for a video streaming application that for each frame measures the deviation from the playout time. Their approach does not calculate capture-to-display latency or frame rate, and requires changing the video application.

adelay [ade, 2011] is a tool that can be used to measure mouth-to-ear latency for audio.
Chapter 5

SECE: Sense Everything, Control Everything

5.1 Introduction

Real-time multimedia collaboration tools which I have presented in previous chapters enable users to communicate. In this chapter I present SECE (Sense Everything, Control Everything) which enables non-technical end-users to create communication-related services. SECE will assist users by automatically managing their communication activities such as messaging and calls, their physical devices such as sensors and actuators, and their social network activities such as tweeting and updating status according to the rules created by the user.

Communication is not limited to telephony anymore, as millions of people use IM, SMS, email, Twitter, and Facebook everyday. These stand alone Internet services are not automated and programmable by end-users, decreasing their utility. For example, it is not easy to create a service which forwards incoming calls to voice mail while the user is in a meeting or turns on the air conditioner while the user approaches his home. Moreover, although these services handle very similar information (e.g., calendar, buddies’ status, presence, messages and user history), they do not work together. Such a lack of service cooperation and automation forces users to check services one after another and manually copy data or configure services based on other services. Unfortunately, there is currently no
easy way to create new services which integrate location, presence, calendar, address book, IM, SMS, calls, email, Facebook and Twitter. Networked sensors and actuators for lights, temperature, humidity, smoke, and motion are also becoming popular both in residential and commercial environments. Sensors can be used as an information source in user-created services and actuators can be controlled by these services.

We developed SECE, a new language and supporting infrastructure which enables end-users to create services which integrates location, presence, calendar, address book, IM, SMS, calls, email, Facebook, Twitter, and physical devices such as sensors and actuators. SECE is a context-aware platform that connects services that until now were isolated, leading to new, more useful and user-personalized, composite services. These services do not require user interaction; they are automated and embedded into users’ life. SECE does not require user interaction except during the service creation phase due to its event-driven operation. Incoming and outgoing phone calls, IM or email messages, presence status updates, sensor inputs, location updates, social network activities such as incoming wall messages or tweets, changes in stock prices or weather are all possible SECE events. Whenever an event occurs it triggers one or more user-created SECE services which eventually handles the event. SECE converges fixed and mobile services by integrating the Internet, cellular and sensor networks. This integration requires interacting with Internet servers, web services, home gateways, and wireless and fixed user devices. SECE has to both sense and control because sensing without controlling is not very useful.

SECE takes actions automatically on behalf of users depending on the monitored information and triggered events. In order to build such a system, the user has to define event-action rules. There are several ways to allow users to define these rules such as using XML, forms or scripts. We choose to develop SECE using a natural-English-like formal language because it is more powerful and easy-to-use than XML and form-based solutions.
An example script which turns the home’s lights on every sunset shows the end-user friendliness of SECE:

```
Listing 5.1: An example SECE script which turns the home’s lights on every sunset

every sunset {
    homelights on;
}
```

SECE has two fully-integrated components, the language itself and its supporting software architecture. IETF standard protocols are used to interconnect networked components.

This chapter is organized as follows. Section 5.2 discusses related work. The SECE language is described in Section 5.3, and the architecture of SECE is presented in Section 5.4. Integration of sensors and actuators is discussed in Section 5.5, and Section 5.6 presents how several location services are integrated into SECE. Integration of presence and instant messaging is explained in Section 5.7. Section 5.8 explains the integration of external web services into SECE. Graphical user interface (GUI) of SECE is presented in Section 5.10.

### 5.2 Related work

<table>
<thead>
<tr>
<th>Systems</th>
<th>User rules</th>
<th>User actions</th>
<th>Communications</th>
<th>Time</th>
<th>Location</th>
<th>Presence</th>
<th>Sensors</th>
<th>Web services</th>
<th>Actuators</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECE</td>
<td>NL-like rules</td>
<td>Tcl scripts</td>
<td>Call, email, IM</td>
<td>✓</td>
<td>User &amp; buddies</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CPL</td>
<td>XML tree</td>
<td>Fixed XML actions</td>
<td>Call</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>LESS</td>
<td>XML tree</td>
<td>XML actions</td>
<td>Call</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>X10, vcr</td>
</tr>
<tr>
<td>SPL</td>
<td>script</td>
<td>Signaling actions</td>
<td>Call</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>VisuCom</td>
<td>Graphical UI</td>
<td>Signaling actions</td>
<td>Call</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>CybreMinder</td>
<td>Form based</td>
<td>Reminder</td>
<td></td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Task.fm</td>
<td>Time rule</td>
<td>Reminder</td>
<td></td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>DiaSpec</td>
<td>Java</td>
<td>Java</td>
<td>✓✓</td>
<td>✓✓</td>
<td>✓✓</td>
<td>✓✓</td>
<td>✓✓</td>
<td>✓✓</td>
<td>✓✓</td>
</tr>
</tbody>
</table>

Table 5.1: Comparison to related work
Several solutions for user created communication-related services have been proposed; some of these solutions are compared in Table 5.1. CPL \cite{Rosenberg99}, LESS \cite{Wu03}, SPL \cite{Burgy06}, VisuCom \cite{Latry07} and DiaSpec \cite{Jouve08} are attempts to allow end users to create services, but they are all limited to controlling call routing. Also, CPL and LESS use XML and, hence, even simple services require long programs. Moreover, XML-based languages are difficult to read and write for non-technical end-users. DiaSpec is a very low level domain-specific design language similar to Java. Writing a specification in DiaSpec and then developing a service using the generated Java framework is definitely not suitable for non-technical end users. The authors of DiaSpec extended \cite{Cassou09} their initial work to support services beyond telephony, which include sensors and actuators. However, it is still only suitable for advanced developers. SPL is a scripting language which is suitable for end-users but only for telephony events such as forwarding or rejecting incoming calls. VisuCom has the same functionality as SPL, but allows users to create services visually via GUI components. Although visual interface of VisuCom is suitable for end-users, its services are limited to telephony events.

CybreMinder \cite{Dey00} is a context-aware tool which allows users to setup email, SMS, print out and on-screen reminders based not only on time but also location and presence status of other users. It uses local sensors such as active badges and floor-embedded pressure sensors to detect a user’s location. It does not take any actions, but rather displays reminders to the end user. Also it is not as powerful as scripting-based systems due to its form-based nature. Task.fm \cite{tas11} is a similar SMS and email remainder system which uses natural language to describe time instants when email or SMS reminders will be sent. However, Task.fm only supports time-based rules and does not include information from sensors. This tool does not take actions other than reminding users via SMS, email or phone call.

Although service composition is being of great interest in the research community, most of the proposed solutions are only theoretical and do not provide any implementation. Yahoo Pipes \cite{yah11b} is a graphical tool for web service composition, but it only generates web mashups from public web feeds and public webpages. Yahoo pipes is restricted to
public knowledge which means it could not even generate web mashups from users’ private information such as their Facebook wall messages or emails.

There are also some web aggregation services like Timelines [tim, 2010] and netvibes [net, 2010] but they just combine all the news and events from users’ social networking, webmail and news sites into one simple page (Figure 5.1). Ping.fm [pin, 2010] allows updating several social network statuses from a single webpage, but it does not support actions triggered by events.

The scripting languages shown in Table 5.1 are neither suitable for non-technical users and only support a limited set of context information.

Figure 5.1: Timelines web aggregation service
5.3 The SECE language

A SECE rule has two parts, the event description and the actions. The event description defines the conditions that need to be satisfied to execute the actions. We have designed the SECE language a formal language but similar to natural English, making it easy to remember and use. A very simple but illustrative example which sends an SMS to the user when Bob’s presence status changes to available is given below:

```
Listing 5.2: An example script which SMS the user when Bob’s presence status changes to available

If Bob’s status is available {
    sms me "Bob is available now.";
}
```

The SECE language is only intended to define events, while rule actions are written in the Tcl language \cite{Ousterhout and Jones, 2009}. We chose Tcl due to its extensibility that makes it simple to add new commands. Thus, SECE users can describe events in a user-friendly and natural way while taking advantage of the expressive power of Tcl to define actions. Moreover, Tcl’s syntax is simple if no complex control statements and structures are considered. This can be seen in the rule examples given in the following subsections. (We may add support for other scripting languages like Ruby \cite{rub, 2011} or Python \cite{py, 2011} in the future.) Another promising although challenging future step would be to extend the SECE language to define rule actions.

The SECE language supports five types of events: time, calendar, context, location and communication. The following subsections explain each of these rules. As a formal language, SECE states the valid combinations of keywords and variables for each kind of event. In all the rule examples, the variables have been highlighted in bold to expose the structure of the language. SECE provides a set of new Tcl commands, such as “sms”, “im”, “email”, “tweet” or “call”. Some commands are specific for particular events, as for example the “accept” and “reject” commands can only be used in communication-based rules. SECE tries to make it easy to integrate external knowledge and uses context such as addresses, phone numbers, weather, and stock prices seamlessly without having to explicitly invoke
External knowledge | Example language constructs
--- | ---
access to the database of personal information | My mobile, Bob’s address
access to contextual information | me.location, bob.activity, bob.presence
access to events-specific (e.g. call, email) information | inside an incoming call rule [incoming origin], reject
address book and IM/presence names | Bob’s calendar events, including public holidays | Thanksgiving, Bob’s birthday
daily times | sunset, sunrise, dawn, dusk, twilight
usage of geocoding and gazettes to look up landmark names | “Columbia University”

Table 5.1: SECE makes it easy to integrate external knowledge seamlessly

libraries or functions. The current status of this integration can be seen from Table 5.1.

5.3.1 Time-based rules

Time-based rules support single and recurring events. We base our time sublanguage design on the iCal specification (RFC5545 [Desruisseaux, 2009]). Ical can express single and recurring events but it is designed to be processed by computers, not users. We designed the SECE’s time sublanguage to be easy to write while maintaining the full expressive power of the iCal specification. Single events start with an on keyword, while recurring events start with an every keyword.
CHAPTER 5. SECE: SENSE EVERYTHING, CONTROL EVERYTHING

An example of SECE time event which will trigger every noon till next April and its equivalent iCal definition for a recurring event is given below.

**Listing 5.3: An example recurring time event which will trigger every noon till next April**

<table>
<thead>
<tr>
<th>SECE: every day at 12:00 until April</th>
</tr>
</thead>
<tbody>
<tr>
<td>iCal: BEGIN:VCALENDAR</td>
</tr>
<tr>
<td>BEGIN:VEVENT</td>
</tr>
<tr>
<td>DTSTART;TZID=America/New_York:20100101T120000</td>
</tr>
<tr>
<td>RRULE:FREQ=DAILY;BYHOUR=12;UNTIL=20100401T120000</td>
</tr>
<tr>
<td>END:VEVENT</td>
</tr>
<tr>
<td>END:VCALENDAR</td>
</tr>
</tbody>
</table>

The recurrence can be defined by the second, minute, hour, day, week, month or year. How long the recurrence takes is determined by the from, until, during or for parameters. A recurrence will repeat indefinitely if no until, during, or for parameters are indicated. The time sublanguage supports natural language constructs like Thanksgiving, Tom’s birthday, sunset, sunrise, lunch break, and tomorrow. In the case of Tom’s birthday, future versions of SECE may try to find the birthdate of Bob from available services like the user’s calendar, Facebook or contacts. Similar lookup operations can be performed for sunset, sunrise, and lunch break. Some expressions like sunset and sunrise can be computed programmatically whereas others like lunch break have to be defined by the user via SECE’s web-based user interface (Section 5.10). Some example time-based rules are given below.

**Listing 5.4: Sends an SMS to Anne on her birthday**

```plaintext
on Anne’s birthday, 2010 at 12:00 in Europe/Zurich {
  sms Anne "Happy Birthday!!!kisses. John";
}
```

**Listing 5.5: Calls Bob at a particular date and time**

```plaintext
on July 16, 2011 at 10:00 am in bob@example.com.location {
  call bob;
}
```
Listing 5.6: Backups the systems every day afternoon except August

```java
every day at last working hour except August {
    backup;
}
```

Listing 5.7: Reminds the user to check the students’ reports and tweets a message every last day of the months

```java
every last monthly day {
    email me "Reminder" "Check the students’ monthly report";
    tweet "one more month is finished.";
}
```

Listing 5.8: Sends a reminder email to a list about weekly meeting

```java
every week on WE at 6:00 PM from 1/1/10 until May 10, 2010 except 3th WE of Feb including first day of June, 2010 {
    email irt-list "Meeting Reminder" "weekly meeting today at 6:00 PM";
}
```

5.3.2 Calendar-based rules

Calendar-based rules specify events that are defined in the user’s calendar. They can be triggered some time before or after an event occurs, as well as when an event begins or finishes. SECE is integrated with Google Calendar [gca, 2010a] and can download all the events from user’s Google account using the Google Calendar API [gca, 2010b]. We implemented the event command to get a Calendar event’s information (title, description, location, duration, start time, end time and participants). Calendar-based rules can be useful to create user-personalized reminders, as in the first example below, but also for other services, as the second example which updates the user’s presence status to busy and reminds the participants to turn their phones off when the weekly meeting begins. When a calendar-based rule is created, SECE checks all of the user calendars about the event using Google Calendar API and, if it is found, determines when the rule should be triggered based on the rule’s conditions and the event’s starting and end times.
CHAPTER 5. SECE: SENSE EVERYTHING, CONTROL EVERYTHING

Listing 5.9: Reminds the participants half an hour before the weekly meeting begins and if the user is not within the three miles of campus emails Bob to prepare everything

```plaintext
when 30 minutes before "weekly meeting" {
    email [event participants] "Reminder" "The weekly meeting will start in 30 minutes";
    if {me not within 3 miles of campus} {
        email [status bob.email] "I’m away" "Please, head the conference room and prepare everything for the weekly meeting. Not sure if I will be on time.";
    }
}
```

Listing 5.10: Changes the user's status to busy and reminds the participants to turn their phones off when the weekly meeting begins

```plaintext
when "weekly meeting" begins {
    status activity busy;
    sms [event participants] "Please, switch your cell phone off or set silent mode."
}
```

5.3.3 Location-based rules

The SECE’s location sublanguage supports five types of location information that are commonly used: geospatial coordinates (longitude/latitude), civic information (street addresses), well-known places, user-specific places and the location of other users. Well-known places are unique and widely-known landmarks such as “Columbia University” or “Rockefeller Center”. User-specific locations are places that are of interest for the local user and therefore are defined by the user in the system, such as office, home and university. The system resolves these constants via the user’s address book, but also allows the user to define custom terms, such as “clubhouse” in the list below. The supported location operators are near [landmark], within [distance] of [landmark], in [landmark] and outside of [landmark]. All these operators can be combined with the “a” and “an” indefinite articles to express generic locations (e.g., ‘a postal office’). Some examples of location events are given below.
5.3.4 Communication-based rules

Communication-based rules specify the action to execute in response to (1) incoming calls, IMs, emails, SMSs or voicemails, (2) outgoing calls or IMs, and (3) missed calls. While an incoming or outgoing call is always a SIP call in our implementation, a missed call could be also a phone call. All these events can be filtered by the user destination and origin, using the `from` and `to` parameters, respectively.

The Tcl environment of SECE is context aware. Properties of an incoming events can be accessed via `incoming` command. This command takes a parameter and returns the requested information about the incoming event. The supported parameters are `origin`, `destination`, `content`, `timestamp`, and `subject`. Depending on the incoming event type this command may return different results. For example `incoming content` may return the message text for an IM event or the email body for an email event. There are other commands like `accept`, `reject`, and `forward`; these will only be available if the context is right. Some communication-based rules are given below.

```
Listing 5.12: Examples of location-based rules

incoming call from a workmate {
    if {[my activity is "on the phone"]} { forward sip:bob@example.com; }
}
```
### Listing 5.13: Examples of location-based rules

```plaintext
missed call {
    if { [my activity is meeting] }
        sms [incoming origin] "Sorry, I am in a meeting but will call you back asap.";
}
```

### Listing 5.14: Examples of location-based rules

```plaintext
incoming call to me.phone.work {
    if { [my location is not office] }
        autoanswer audio noffice.au;
        email me "[incoming origin] tried to reach you on your work phone at [incoming timestamp]";
}
```

### Listing 5.15: Examples of location-based rules

```plaintext
incoming email from my boss {
    if { my activity is not working }
        sms me "New email from the boss at [incoming timestamp]. Subject: [incoming subject]";
}
```

### Listing 5.16: Examples of location-based rules

```plaintext
incoming im {
    if { [my status is away] }
        sms me "[incoming origin] sent this IM: [incoming content]";
}
```
5.3.5 Context-based rules

Context-based rules specify the action to execute when context information changes, such as presence, call, weather, stock prices, sensor states. To be more extensible, SECE keeps all the context information in a registry tree (Figure 5.2). Each user has a separate and isolated tree in the current implementation, but future versions of SECE may enable users to share parts of their tree with other users. Any existing or newly introduced component of SECE can read and write to the registry tree. Registry trees are automatically stored in a database table and their state are restored in case the SECE server reboots. The contextual information (e.g., activity, status and stock.google in the below example rules) can be any hierarchical variable in the form of x.y.z.t, such as phone.office, activity and office.temperature. The following listings present example context-based rules.

<table>
<thead>
<tr>
<th>Listing 5.17: Automatically publishes activities to user’s calendar as soon as user activity changes</th>
</tr>
</thead>
</table>
| if my activity changed {
  publish "activity: [status activity]" to calendar;
} |

<table>
<thead>
<tr>
<th>Listing 5.18: Notifies the user via instant messaging as soon as Bob becomes available</th>
</tr>
</thead>
</table>
| if bob@example.com’s status is available {
  im me "Bob is available."
} |

<table>
<thead>
<tr>
<th>Listing 5.19: Notifies the user via SMS if Google’s stock prices passes $580</th>
</tr>
</thead>
</table>
| if stock.google > 580 {
  sms me "google stock: [stock google]"
} |

The context’s subject is given by the my and ’s operators (e.g., “bob’s phone.office” and “my activity”). Shortcuts can be used instead of these operators, so that for example bob.device.mobility is equal to bob’s device.mobility. The relational operators can be expressed as symbols or text (e.g., the equal relation can be given by “=”,” “is” or “equal”). Information derived from sensors, such as smoke, light, humidity, motion and temperature
sensors can be also used in context-based rules. Naming of sensors is an open problem that, for now, is beyond our scope. We have adopted a simple solution that consists of a translation table from internal, machine-friendly names (e.g., 00-0C-F1-56-98-AD) to more user-friendly identifiers (e.g., office.smoke).

```plaintext
Listing 5.20: Notifies the user via SMS if motion detector in warehouse detects motion

if my warehouse.motion equals true {
    sms me "person in the warehouse."
}
```

```plaintext
Listing 5.21: Notifies the user via SMS and fire department via text-to-speech if office smoke detector detects a fire

if my office.smoke equals true {
    sms me "fire in the office"
    calltts firedepartment "fire in [status office.address]"
}
```

### 5.3.6 States vs. events

SECE is designed for handling events, i.e., state transitions, that trigger a set of actions. This works well for discrete events, such as calls and calendar entries, but is somewhat more awkward for expressing behavior that combines a set of variables to define the state of another variable. For example, to manage the home heating systems, events would have to be defined for people entering and leaving the house, along with temperature and time-of-day conditions. It is much easier to write such cases as predicates, such as “turn on the air conditioner if the indoor temperature is higher than 80 F and I am at home”. One possible syntax for such conditions is shown in the example below.

```plaintext
Listing 5.22: Possible syntax for defining the state of an air conditioner

ac := temperature > 80 and me in home;
```

Only one predicate can exist for a variable and, hence, rule conflicts on actuators are avoided. We are currently exploring the applicability of predicate and event-based systems, and whether it makes sense to integrate them or keep them separate.
5.4 The software architecture of SECE

Due to its integrative nature, SECE has to communicate with several third party applications, hardware, and APIs like Google services (e.g., GMail, Google Contacts and Google Calendar), Facebook, Twitter, maps, VoIP proxy servers, presence servers, sensors and actuators (see Section 5.5 and 5.8). SECE considers not only the user’s context but also information about external entities other than sensors, such as his or her buddies.

![Figure 5.2: User information registry (partial)](image)

SECE keeps the user information in a Document Object Model (DOM) [dom, 2004] tree registry (Figure 5.2). The user information is not restricted to personal information like phone numbers but also includes contextual information from sensors and Internet services. Context-based rules associate events with the nodes of the registry. A rule does not have to be associated with a leaf node; it can be associated with any node. The benefit of associating rules with top-level nodes is to write generic rules like “if Bob changes {...}” to allow monitoring any activity related to a subtree.

As Figure 5.3 depicts, the Presence Server (PS) plays a key role in collecting contextual information from different sources. SECE relies on SIMPLE [sim, 2010], the Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions, which is an instant messaging (IM) and presence protocol suite based on Session Initiation Protocol
According to the SIMPLE architecture, the PS receives presence publications from the context sources that contain the most recent information and, in turn, it notifies SECE of the context changes. In the SECE framework, context sources include user devices’ presence applications and gateways that control sensor networks, energy consumption and user location via RFID. Currently, we are using the Mobicents Presence Server \cite{mob2011}.

Another external server that plays a key role in our SECE implementation is the SIP Express Router (SER) \cite{ser2011}, which handles SIP communications. SER will inform SECE whenever an incoming or outgoing communication, such as a call or IM, takes place. Then, if a communication rule is triggered, a rule action could forward, reject, or modify the call. Details about SER integration will be covered in Section 5.8.4.

Integration of sensors and actuators will be discussed in Section 5.5 in detail. Similarly, Section 5.8 discusses how third party web services are integrated into SECE.
5.4.1 The software components of SECE

The software components of SECE are illustrated in Figure 5.4 and Figure 5.5. We are developing SECE in Java due to its extensive libraries and support for all operating systems.

Figure 5.4: The software components of SECE

Figure 5.5 only shows some relevant Java libraries such as ANTLR, which is used by the language compiler, JACL [Lam and Smith, 1997] that is a Tcl implementation in Java, JAIN-SIP [jai, 2011] for SIP signaling and GDATA [gda, 2010] to access the Google web services.

The agent layer contains the agents that communicate with external services. Agents can generate events (e.g., the Mobicents agent creates presence events), provide some useful functions (e.g., the GMaps agent provides direct and reverse geo-coding) or take some action (e.g., the Gmail agent can send emails).
The rules layer contains the rule implementations. These implementations utilize the service API layer to subscribe to interesting events, to check rules’ conditions and to execute rules’ actions if necessary. The context database contains all the users’ and their buddies’ context, including presence, location, preferences, configuration data and sensor information. Rules only can modify or read this database through the APIs in the service API layer.
5.5 Sensors and actuators

Sensors and actuators are an important part of our daily life and we thus made them part of SECE. Electrical appliances can be controlled automatically depending on the information coming from sensors, from other web services such as weather service and from time of day. Presence sensors may update users availability. Combined with SECE’s knowledge on the user’s presence, location, and availability; actuators may take actions on behalf of the user. For example, when the user is approaching his office, SECE may turn on the AC depending on the temperature information coming from office temperature sensor. Or SECE may turn off the lights if there is no motion sensor activity in the user’s home or office.

![Phidget experimental setup](image)

SECE’s sensors and actuators support is platform independent. Currently, we are experimenting with ZigBee [zig, 2010] and Insteon [ins, 2010] wireless device control modules and Phidgets [phi, 2010] USB sensors and actuators. Figure 5.6 presents our USB-based phidget.
experimental setup where motion\(^1\), temperature\(^2\) and light\(^3\) sensors and an LED light\(^4\) are attached to a controller board\(^5\). Controller board, which is attached to a computer running gateway software via USB, sends sensors updates to the computer and retrieves actuators states from it.

The communication between SECE and gateway can be carried both via REST-style (Representational State Transfer) requests or via the SIMPLE event notifications. We have implemented the REST-style architecture in our SECE implementation and future versions of SECE may implement the SIMPLE based solution.

Figure 5.7: The architecture of sensors and actuators gateway

---

2. [http://www.phidgets.com/documentation/Phidgets/1124.pdf](http://www.phidgets.com/documentation/Phidgets/1124.pdf)
The message format for sensor information and action requests can be the same in both architectures. Sensor information can be carried in RDF documents which makes the protocol sensor network agnostic. Actions on actuators can also described in RDF documents.

The gateway is split into two layers: a device-independent layer and a protocol layer. The former maintains an RDF database that represents the conceptual sensor model, while the latter carries out the necessary translations between the RDF model and the device- and network-dependent information and actions.

In our REST-style implementation HTTP POST requests are used to transfer sensor updates to SECE and action requests from SECE to gateway (see Figure 5.7).

SECE may automatically create Tcl commands for each actuator after being notified of the RDF model. But in our current implementation it just updates the related registry entry such as office.motion. Similarly if the user updates a actuator related registry entry SECE sends an action request to gateway such as “office.led = true”.

Future versions of SECE may use Constrained Application Protocol (CoAP), a specialized web transfer protocol for use with constrained networks and nodes for machine-to-machine applications such as smart energy and building automation. CoAP provides a method/response interaction model between application end-points, supports built-in resource discovery, and includes key web concepts such as URIs and content-types. CoAP easily translates to HTTP for integration with the web while meeting specialized requirements such as multicast support, asynchronous message exchanges, very low overhead and simplicity for constrained environments.

Google previewed an initiative called Android @ Home during Google I/O conference 2011, which allows Android apps to discover, connect and communicate with appliances and devices in your home. SECE may control and sense these devices using the same APIs available to Android applications.

\footnote{http://www.google.com/events/io/2011/}
5.6 Location

SECE learns the user’s current location from Google Latitude [gla, 2010a] service. We choose Google Latitude since it has clients for almost all mobile platforms. The user’s mobile device uploads his location to Google Latitude servers periodically. SECE retrieves the user’s current location from Latitude servers using the Google Latitude API [gla, 2010b].

In order to support location-based rules, learning the user’s current location is not enough. SECE computes the distance between the user’s location and an address and supports not only point-based locations but also polygon based ones. SECE uses online map APIs for geo-coding and supports polygon definitions both in the location sublanguage and also in the GUI. Future versions of SECE may learn the generic places like museums around a specific location, which is required by some rule types, by consulting LoST (Location-to-Service Translation Protocol) [Hardie et al., 2008] servers.

As we noted earlier in Section 5.3.3, the SECE’s location sublanguage supports five types of location information that are commonly used: geospatial coordinates (longitude/latitude), civic information (street addresses), well-known places, user-specific places and other users. SECE uses Google Maps Data API for Java [gma, 2010] to learn the geospatial coordinates of places which are used in near [landmark] or within [distance] of [landmark] rules. User-specific locations are places that are of interest for the local user and therefore are defined by the user in the system, such as office, home and school. The system resolves these constants via the user’s address book or polygon list (which is discussed in the next paragraph), but also allows the user to define custom terms, such as “clubhouse” in the list below.

Geospatial point coordinates are not enough for in [landmark] and outside of [landmark] rules; polygon-based location definitions are needed. We implemented a polygon editing page (Figure 5.8) in SECE’s web based GUI. Users may draw and name their own private polygons or they can use existing public polygons while they are preparing their in [landmark] and outside of [landmark] rules. Some example rules which require polygon-based locations are given below.
Bob in “Columbia University” { ... }
me in a theatre { ... }
Alice outside of clubhouse { ... }

To sum up, the supported location operators are near [landmark], within [distance] of [landmark], in [landmark] and outside of [landmark]. All these operators can be combined with the “a” and “an” indefinite articles to express generic locations (e.g., ‘a postal office’). SECE will query a LoST server to retrieve the generic places around a specific location. This part is not implemented yet.

Figure 5.8: SECE’s polygon editing interface
5.7 Presence and Instant Messaging (IM)

Integration of Presence and Instant Messaging (IM) networks to SECE allows users (1) to monitor their friends presence states, (2) to send instant messages to their friends programmatically, (3) to update their presence states using other information sources such as their current location, activity (e.g. on the phone call) or calendar (e.g. in a meeting), (4) to process incoming IM programmatically. SECE not only can monitor incoming instant messages but can also relay them via other communication methods like SMS in case users are away from their computer.

Unfortunately, there is no single IM network which is used by everyone. Instead, there are several IM networks and most popular ones are proprietary. We tried to integrate Skype and Windows Live Messenger networks but could not succeed due to closed protocols and nonexistent libraries. Skype Public API 7 allows connecting hardware and software accessories to Skype desktop clients. This API is not useful to SECE unless the SECE server and the Skype is running on the same desktop computer which is not true most of the case where SECE is running on a remote server.

Fortunately, IETF has standardized two presence and IM protocols namely SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) [Rosenberg, 2004a; Rosenberg, 2004b; Rosenberg, 2004c; Peterson, 2004a; Peterson, 2004b; Peterson, 2004c; Klyne and Atkins, 2004; Sugano et al., 2004] and The Extensible Messaging and Presence Protocol (XMPP) [Saint-Andre, 2004a; Saint-Andre, 2004b; Saint-Andre, 2004c]. We integrated both of them into SECE.

Current presence and IM networks support Multiple Point of Presence (MPOP) [mpop] which allows the user to log in to IM network from more than one places or devices and to switch between them seamlessly without relogging in. Incoming instant messages are also delivered to all logged in endpoints. SECE registers to presence and IM networks on behalf of the SECE user using MPOP feature. Also, SECE can send and receive IM messages, and can set and get presence state on behalf of the SECE user.

Listing 5.23 illustrates a rule which sends an IM to the caller if the user is not available.

---

7http://developer.skype.com/
CHAPTER 5. SECE: SENSE EVERYTHING, CONTROL EVERYTHING

Listing 5.23: Sends an IM to caller if the user is not available

```java
incoming call {
    if { [my status is unavailable] } {
        im [incoming origin] "I am not available right now and will call you back
        as soon as possible.";
    }
}
```

5.7.1 Architecture to support several presence and IM networks

Each presence and IM network has different APIs and libraries. In order to support several current and possible future networks we have designed two Java interfaces. These interfaces allow interacting with different presence and IM networks in the same way. Also, new networks can be added easily without any modifications on the SECE side.

The SECE server implements the `SECEListener` interface (Listing 5.24) which consists of several callback methods. SECE passes a reference to itself while creating presence and IM service objects. These service objects notify SECE when a presence or IM activity happens via callback methods. The functionalities of these callback methods are explained in their respective comment blocks in listing 5.24.

Listing 5.24: Sends an IM to caller if the user is not available

```java
package edu.columbia.cs.sece.presenceAgent;
import java.util.Collection;

public interface SECEListener {

    /**
     * This callback method notifies the SECE for an incoming IM.
     * The contents of the IM and the sender are contained in the
     * MessageEvent object. Implemented for both XMPP
     * and SIMPLE clients.
     * @author Jaya Allamsetty
     */
    public void processMessageEvent(MessageEvent event);
```
The presence and IM service objects implement the IPresenceIM interface. SECE can interact with presence and IM networks using this standard interface. For example, in order to send an IM message to an IM address, SECE calls the sendMessage() method of the appropriate service object. These methods are not used by the end-user but SECE developers. The end-users register their IM accounts to SECE via web user interface. They can send an IM to their friends via im action command. Internally im action command creates an IPresenceIM object using the user’s IM account parameters such as username and password, which are stored in the registry database. Then SECE sends the requested IM using the IPresenceIM object’s sendMessage() method.

End-users can (1) send an IM to their friends using the im action command, (2) change their presence states using presence action command, (3) create rules to process incoming IMs using the incoming im communication-based rule, (4) create rules to process their friends status updates using context-based rules such as “if Bob is available”. In the current version of SECE only im command is implemented.
Listing 5.25: Sends an IM to caller if the user is not available

```java
package edu.columbia.cs.sece.presenceAgent;
import java.util.Collection;
import org.jivesoftware.smack.packet.Presence;

public interface IPresenceIM {

    /**
     * This method attaches the specified seceListeners to the client. The appropriate methods implemented by the listener are invoked by the client if it is in the listening mode. Implemented for both XMPP and SIMPLE clients.
     * @author Jaya Allamsetty
     */
    public void addSECEListener(SECEListener seceListener)
            throws ConnectionIssueException;

    /**
     * This method detaches the specified seceListener from the client. Implemented for both XMPP and SIMPLE clients.
     */
    public void removeSECEListener(SECEListener seceListener)
            throws ConnectionIssueException;

    /**
     * This method changes the mode of the client to Listening. The appropriate methods implemented by the attached listeners are invoked by the client upon receipt of interesting events from the server or from other users. Implemented for both XMPP and SIMPLE clients.
     */
    public void startListening();

    /**
     * This method changes the mode of the client to not listening. Methods of the attached listeners are not invoked upon receipt of interesting events. This is the default mode of operation for both the XMPP and
     */
```
* SIMPLE clients.
* Implemented for both XMPP and SIMPLE clients
*/
public void stopListening();

/**
 * This method sends an IM to the destination user. A single IM is sent
 * depending on the implementation of the user agent client Implemented for
 * both XMPP and SIMPLE clients
 */
public void sendMessage(String username, String message)
    throws ConnectionIssueException;

/**
 * This method sends an IM to a group of users This method is currently
 * implemented for XMPP clients only. Implemented for only XMPP clients
 */
public void sendMessage(Collection<String> group, String message)
    throws MethodNotImplementedException, ConnectionIssueException;

/**
 * This method publishes the presence information to the server that the
 * user is registered to. for XMPP - user status and mode is published to
 * the users in its Roster or contact list for SIMPLE - user information is
 * sent via a PUBLISH to the presence server Implemented for both XMPP and
 * SIMPLE clients
 */
public void setPresence(Presence presence)
    throws ConnectionIssueException, MethodNotImplementedException;

/**
 * This method publishes the Rich Presence information to the presence
 * server using PUBLISH This method is currently implemented for SIMPLE
 * client only Implemented only for SIMPLE clients
 */
public void setPresence(RichPresence presence)
    throws ConnectionIssueException, MethodNotImplementedException;
/*
 * This method retrieves the presence information of a particular user if
 * the target user is in the contact list of the user on whom this method
 * is invoked. This is currently implemented for XMPP clients only
 */

public Presence getPresence(String username)
    throws MethodNotImplementedException, ConnectionIssueException;

/**
 * This method adds the contact to the specified group For XMPP - adds an
 * entry in the Roster of the user For SIMPLE - this method creates a
 * ResourceList with the user as the entry, uploads the list to the XDM
 * server and sends a Subscribe to the Presence Server so that the user is
 * notified of any changes of state Implemented for both XMPP and SIMPLE
 * clients
 */

public void addContact(String username, String group)
    throws ConnectionIssueException;

/**
 * This method adds a contact to the user For XMPP - adds an entry in the
 * roster associated with the user
 * For SIMPLE - creates a ResourceList with
 * the default name, uploads the list to the XDM server and sends a
 * Subscribe to the presence server so that the user is notified of changes
 * of state Implemented for both XMPP and SIMPLE clients
 */

public void addContact(String username)
    throws MethodNotImplementedException, ConnectionIssueException;

/**
 * This method retrieves the list of contacts in the roster of the user. It
 * is currently implemented only for XMPP
 */

public Collection<String[]> getContacts()
    throws MethodNotImplementedException;
public void subscribe(String username) throws ConnectionIssueException, MethodNotImplementedException;

public void subscribe(Collection<String> group) throws ConnectionIssueException, MethodNotImplementedException;

5.7.2 Integration of SIMPLE

SIMPLE [sim, 2010], the Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions, is a SIP-based solution. We have used JAIN-SIP [jai, 2011] library to receive and send SIP messages. We tested the integration using Mobicents 8, the open source Service Logic Execution Environment (SLEE) and SIP server, but our solution should work with other SIP based presence servers. For SIMPLE client we used Jitsi 9.

5.7.3 Integration of XMPP

We have used the Smack [sma, 2011] open source XMPP (Jabber) client library to receive and send XMPP messages. The Smack library does not implement our two interfaces, but we created a class which implements these two interfaces. We tested our implementation using Google Talk [gta, 2011b] which is an XMPP client.

8http://www.mobicents.org/
9http://www.jitsi.org/
5.8 Integration with external services

5.8.1 Overview

SECE is integrated with several different online services such as social networks, VoIP systems, presence servers, online maps, location, photo sharing and calendaring services. Table 5.2 summarizes action commands and events supported by SECE.

In order to integrate with online services, SECE has to take actions on behalf of the user; such actions usually require authentication. Asking the user for his username and password is a possible but problematic solution. Users do not want to give their username and passwords to other services to prevent account theft and unauthorized activities. Fortunately, the IETF OAuth \[Hammer-Lahav, 2010\] standard solves these problems by authenticating SECE to these online services without requiring the user’s username and password. We describe it briefly below.

5.8.2 OAuth (Open Authentication) authentication mechanism

OAuth (Open Authentication) \[Hammer-Lahav, 2010\] is an IETF standard which enables users to grant third-party services to access their resources without revealing their username and passwords. The OAuth Guide \[bau, 2010\] describes OAuth as

Many luxury cars come with a valet key. It is a special key you give the parking attendant and unlike your regular key, will only allow the car to be driven a short distance while blocking access to the trunk and the onboard cell phone. Regardless of the restrictions the valet key imposes, the idea is very clever. You give someone limited access to your car with a special key, while using another key to unlock everything else.

As the web grows, more and more sites rely on distributed services and cloud computing: a photo lab printing your Flickr photos, a social network using your Google address book to look for friends, or a third-party application utilizing APIs from multiple services.

The problem is, in order for these applications to access user data on other sites, they ask for usernames and passwords. Not only does this require exposing
<table>
<thead>
<tr>
<th>Context</th>
<th>Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facebook</td>
<td>incoming wallmessage</td>
<td>facebook</td>
</tr>
<tr>
<td></td>
<td>incoming newsmessage</td>
<td></td>
</tr>
<tr>
<td></td>
<td>incoming direct</td>
<td></td>
</tr>
<tr>
<td>Twitter</td>
<td>incoming twitter direct</td>
<td>tweet</td>
</tr>
<tr>
<td></td>
<td>incoming twitter wallmessage</td>
<td></td>
</tr>
<tr>
<td>Phone calls</td>
<td>incoming call</td>
<td>call</td>
</tr>
<tr>
<td></td>
<td>incoming voicemail</td>
<td>calltts</td>
</tr>
<tr>
<td></td>
<td>missed call</td>
<td>accept</td>
</tr>
<tr>
<td></td>
<td>outgoing call</td>
<td>reject</td>
</tr>
<tr>
<td></td>
<td>forward</td>
<td></td>
</tr>
<tr>
<td>SMS</td>
<td>incoming SMS</td>
<td>sms</td>
</tr>
<tr>
<td>IM</td>
<td>incoming im</td>
<td>im</td>
</tr>
<tr>
<td></td>
<td>outgoing im</td>
<td></td>
</tr>
<tr>
<td>Email</td>
<td>incoming email</td>
<td>email</td>
</tr>
<tr>
<td>Presence</td>
<td>if Bob is available</td>
<td>presence</td>
</tr>
<tr>
<td>Calendar</td>
<td>when [time] before [meeting]</td>
<td>schedule</td>
</tr>
<tr>
<td></td>
<td>when [meeting] begins</td>
<td></td>
</tr>
<tr>
<td>Flickr</td>
<td></td>
<td>flickr</td>
</tr>
<tr>
<td>Translate</td>
<td></td>
<td>to_en, to_tr, ...</td>
</tr>
<tr>
<td>Location</td>
<td>near [landmark]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>within [dist] of [landmark]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>in [landmark]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>outside of [landmark]</td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>on [time]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>every [time]</td>
<td></td>
</tr>
<tr>
<td>Contextual</td>
<td>if [variable] [operator]</td>
<td>status [variable] [value]</td>
</tr>
<tr>
<td>Sensors</td>
<td>if office.motion equals true</td>
<td></td>
</tr>
<tr>
<td></td>
<td>if office.temperature &gt; 250</td>
<td></td>
</tr>
<tr>
<td>Actuators</td>
<td></td>
<td>status office.light true</td>
</tr>
</tbody>
</table>

Table 5.2: Summary of SECE events and actions
user passwords to someone else — often the same passwords used for online banking and other sites — it also provides these application unlimited access to do as they wish. They can do anything, including changing the passwords and lock users out.

OAuth provides a method for users to grant third-party access to their resources without sharing their passwords. It also provides a way to grant limited access (in scope, duration, etc.).

For example, a web user (resource owner) can grant a printing service (client) access to her private photos stored at a photo sharing service (server), without sharing her username and password with the printing service. Instead, she authenticates directly with the photo sharing service which issues the printing service delegation-specific credentials.

Several service providers such as Yahoo, Facebook, Twitter, Flickr and Google support OAuth and even some of them require third-party services to use OAuth. A complete list of service providers which support OAuth can be retrieved from OAuth wiki [oau, 2010a].

The OAuth authentication flow can be seen from Figure 5.10. In our case, SECE becomes the consumer and web services like Facebook, Google and Yahoo become the service provider. SECE web interface has a page (Figure 5.18) where user can register his other accounts with SECE. Figure 5.8 displays the process which user should take in order to register his account with SECE. After clicking “Register my Twitter Account”, the user is taken to a Twitter webpage where user authorizes SECE to access user’s Twitter account.

5.8.3 Social networks

SECE can receive direct\(^\text{11}\) and wall\(^\text{12}\) messages from Twitter and Facebook. It can also post tweets to Twitter and change Facebook status. Other social networks can be added in

\(^{10}\text{http://dev.twitter.com/pages/sign_in_with_twitter}\)

\(^{11}\text{A private message exchanged directly between two users.}\)

\(^{12}\text{The Wall is a space on each user’s profile page that allows friends to post messages for the other users to see. Different users’ wall posts show up in an individual’s news feed.}\)
future. It is also possible to retrieve other information such as friends, contacts and social events from these social networks.

**Twitter** SECE supports the following two rules to receive incoming tweets and direct messages from Twitter.

<table>
<thead>
<tr>
<th>Listing 5.26: Communication-based rule to process incoming private messages from Twitter network</th>
</tr>
</thead>
<tbody>
<tr>
<td>incoming twitter direct</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing 5.27: Communication-based rule to process incoming tweets from the followed users</th>
</tr>
</thead>
<tbody>
<tr>
<td>incoming twitter wallmessage</td>
</tr>
</tbody>
</table>
OAuth Authentication Flow

Person Using Web Browser
or Manual Entry Consumer/Service Provider

Consumer

Service Provider

A
Request Request Token

Grant Request Token

B
Direct User to Service Provider

Obtain Unauthorized Request Token

C
Obtain User Authorization

Direct User to Consumer

D
Request Access Token

Grant Access Token

E
Access Protected Resources

F
Exchange Request Token for Access Token

G

Figure 5.10: OAuth authentication flow
SECE also supports tweeting via command

Listing 5.28: Action command to tweet

```plaintext
tweet "I am available";
```

SECE will ignore an incoming tweet if it is posted by SECE to prevent a cycle.

**Facebook** Facebook has a “wall feed” which are the messages posted to user’s own wall and another “news feed”\(^\text{13}\) which are the messages posted to user’s friends’ walls [fac, 2011]. SECE supports the following three rules to receive incoming messages from Facebook

Listing 5.29: Communication-based rule to process incoming private messages from user’s friends

```plaintext
incoming facebook direct
```

Listing 5.30: Communication-based rule to process wall messages posted to user’s own wall

```plaintext
incoming facebook wallmessage
```

Listing 5.31: Communication-based rule to process wall messages posted to user’s friends walls (news feed aggregates wall messages posted to user’s friends’ walls)

```plaintext
incoming facebook newsmessage
```

SECE also supports updating the user’s Facebook status via command

Listing 5.32: Action command to update user’s Facebook status

```plaintext
facebook "I am working on SECE-Facebook integration";
```

Twitter4J [twi, 2010] and RestFB [res, 2010] Java libraries are used to integrate Twitter and Facebook, respectively.

---

\(^{13}\)News feed displays constantly updated list of user’s friends’ Facebook activity. News Feed highlights information that includes profile changes, upcoming events, and birthdays, among other updates. News Feed also shows conversations taking place between the walls of a user’s friends.
SECE is integrated with the SIP Express Router (SER) [ser, 2011]. There is a TCP-based communication link between SER and SECE. Using this link, SER and SECE can exchange messages to initiate new calls or to handle an incoming or outgoing call. New Tcl commands `reject`, `forward`, and `accept` are added to handle an incoming or outgoing call. SECE can initiate SIP calls using the new Tcl action command `call`. Another action command `calltts` is introduced to allow reading the given command parameters to the dialed number with the help of text-to-speech technology. The current SER implementation does not support `call` and `calltts` commands yet. Due to the delay-sensitiveness of VoIP calls, SER will not wait more than a few hundred milliseconds for the SECE’s response for incoming or outgoing calls. Some example rules which demonstrate the interaction between SECE and SER are given below.

**Listing 5.33:** Rejects all incoming calls from 1800JUSTADS

```
incoming call from 1800JUSTADS {
    reject;
}
```

**Listing 5.34:** Forwards incoming calls from workmates to Bob if the user is on the phone

```
incoming call from a workmate {
    if { ![my activity is "on the phone"] } {
        forward sip:bob@example.com;
    }
}
```

**Listing 5.35:** Informs the caller about the user's unavailability via SMS

```
incoming call {
    if { [my activity is meeting] } {
        sms [incoming origin] "Sorry, I am in a meeting but will call you back asap."
    }
}
```
CHAPTER 5. SECE: SENSE EVERYTHING, CONTROL EVERYTHING

Listing 5.36: If the user is not in his office when a call comes SECE plays a message to the caller and notifies the user via email

```plaintext
incoming call to me.phone.work {
    if { [my location is not office] } {
        autoanswer audio nooffice.au;
        email me "missed call" ":[incoming origin] tried to reach you on your work phone at [incoming timestamp]";
    }
}
```

Listing 5.37: Calls the fire department and speaks the address of the office via text-to-speech in case of a fire detection by smoke sensors

```plaintext
if my office.smoke equals true {
    calltts firedepartment "fire in [status office.address]";
}
```

5.8.5 Translation

SECE support translation of text using the Google Translation API [gta, 2011a]. The language of the original text is auto-detected. The following rule sends an email message for every incoming Facebook news message. The body of the email contains Turkish-translated version the original message.

Listing 5.38: Sends an email message which contains Turkish-translated version of the original Facebook news message

```plaintext
incoming facebook newsmesage{
    email [my email.home] "facebook message" [to_tr [incoming content]];
}
```

to_xx commands translate given texts to a particular language. Users replace xx with the requested language letters, such as to_en, to_tr. Google Translate and SECE currently supports 57 languages.
5.8.6 Calendar services

SECE is integrated with Google Calendar \cite{gca,2010a} and can download all the events from a user’s Google account using Google Calendar API \cite{gca,2010b}. We implemented the `event` command to get a Calendar event’s information (title, description, location, duration, start time, end time and participants). An example meeting remainder rule using `event` command is given below.

```
Listing 5.39: Reminds the participants 30 minutes before the meeting via SMS and email

when 30 minutes before any meeting {
  set content "The event [event title] will start in 30 minutes and will last [event duration] minutes. Description: [event description]. Start time: [event start]. End time: [event end]. Location: [event location]. Participants: [event participants]";
  set subject "Calendar: [event title]";
  email [my email.home] $subject $content;
  sms [my email.home] $subject $content;
}
```

The schedule command is also implemented to publish an event to the user’s calendar.

```
Listing 5.40: Publishes an call received event to the user’s calendar for each incoming call

incoming call {
  schedule "call received from [incoming origin]";
}
```

This command’s parameter is the event’s title (“What” in Google Calendar) and the time is the current time. Another function which updates your calendar with more complete calendar events, which can include participants, start and end times, location and description is implemented too.
5.8.7 Online photo sharing services

SECE can post a computer-generated photo to the user’s Flickr [fi, 2010] account. What is more interesting than posting a regular photo is a computer-generated photo from given text messages. A new command 
\texttt{flickr} added to Tcl which takes any number of text arguments, converts these arguments into a single photo, and uploads this photo to Flickr. This feature can be used to display a single image on a Flickr-connected digital photo frame sitting on the user’s desk or attached to the user’s office door. Your desk frame may display recent events such as incoming SMS, missed calls or incoming emails. Whereas the user’s office door frame may display his schedule or a message about your availability. Remember that SECE knows the user’s location, presence status, whether the user is on a phone call, and his calendar. All this knowledge can be used to post status update messages to social networks and digital photo frames. Other online photo sharing services like Photobucket [pho, 2010] and SmugMug [smu, 2010] may be integrated to SECE later. Figure 5.11 shows the result of following command.

\begin{verbatim}
Listing 5.41: Generates and publishes a computer-generated image from given text messages to Flickr
flickr "Hello IRT members how are you?" "testing flickr integration"
"my testing is almost done";
\end{verbatim}

Figure 5.11: SECE generated photo uploaded to Flickr
5.8.8 Google Voice

Google Voice \([gv, 2010]\) is a service which allows user to send and receive SMS messages, initiate voice calls, and ring all of their phones for incoming calls. We have used the Java API for Google Voice \([gva, 2010]\) for this integration. New Tcl commands \texttt{sms} and \texttt{call} are added to send SMS and to initiate a new call, respectively. Communication-based rules \texttt{incoming sms}, \texttt{incoming voicemail}, and \texttt{missed call} are also wired to user's Google Voice account.

SECE periodically checks the user’s Google Voice account for SMS, voicemails, and missed calls. The checking period can be automatically adjusted depending on the SECE server’s load, but currently it is set to 5 seconds.

5.8.9 Address book

SECE downloads user’s contacts and groups from a user’s Google account using Google Contacts Data API \([gco, 2010]\). Downloaded information can be used in rules such as Bob’s email, irt\_members, Alice’s phone.
5.9 Adding new event types and action commands to SECE

5.9.1 Overview

In the previous sections I presented the action commands and event types we implemented on the current version of SECE. In this section I discuss the extensibility of SECE and how to add new event types and action commands. While designing the SECE we specifically considered the fact that SECE has to support new event types and action commands due to its integrative nature. Adding a new action command is pretty easy and will be demonstrated below.

On the other hand, adding a new event type requires more work. First of all, the new rule header for the event has to be added to the SECE grammar which is stored in a single file with over 2,000 lines of grammar definitions. The steps after grammar file modification are easier because all events are tracked by service classes and all service classes extends a base SECEService class. The developers has to extend this base class to suit their needs. The base class handles starting, restarting and stopping of the service. In future implementations, the grammar file can be modularized into several files to allow easy additions to the SECE grammar.

5.9.2 Adding a new action command to the SECE

In order to present adding new action commands, first I will explain how SECE runs user’s Tcl scripts. Listing 5.42 displays the executeCode() method which runs the user’s Tcl scripts. SECE’s action commands such as “tweet” are added to the Tcl interpreter object before executing the user’s script. SECE registers an object for each new action command. For example, SECE creates an EmailCmd object and registers this object for “email” action commands.

```
public boolean executeCode(Service service, String code) {
    //Creates a new Tcl interpreter
    Interp interp = new Interp();
```

Listing 5.42: Displays how SECE executes user’s Tcl scripts
try {
    // Add new actions commands to the Tcl interpreter
    interp.createCommand("email", new EmailCmd(emailEventProducer));
    interp.createCommand("status", new StatusCmd(this));
    interp.createCommand("tweet", new TweetCmd(this));
    interp.createCommand("flickr", new FlickrCmd(this));
    interp.createCommand("facebook", new FacebookCmd(this));
    interp.createCommand("sms", new SMSCmd(googleVoice, service));
    interp.createCommand("im", new ImCmd(this));
    interp.createCommand("call", new CallCmd(null, googleVoice, service));
    interp.createCommand("incoming", new IncomingCmd(service));
    interp.createCommand("my", new MyCmd(this));
    interp.createCommand("accept", new AcceptCmd(service));
    interp.createCommand("reject", new RejectCmd(service));
    interp.createCommand("event", new EventCmd(service));
    interp.createCommand("schedule", new ScheduleCmd(this.
       googleCalendarHandler));
    TranslatorCmd st = new TranslatorCmd();
    for (final Language language : Language.values()) {
        interp.createCommand("to_"+language.toString(), st);
    }
    // runs the user’s Tcl script
    interp.eval(code);
} catch (Exception ex) {
    return false;
} finally {
    interp.dispose();
}
return true;

The Tcl interpreter calls the cmdProc method of the EmailCmd object encountering an “email” action command in the user’s script. All registered objects have the cmdProc method because they all extend Tcl’s Command base class which has this method. The signature of the cmdProc method is given below in listing 5.43. The arguments to action
commands are passed to registered objects in an array.

Listing 5.43: The signature of cmdProc method

```java
public void cmdProc(
    Interp interp,      // Current interpreter.
    TclObject objv[])   // Arguments to "email" action command.
    throws TclException {
}
```

Listing 5.44 displays the source code of FacebookCmd class. When the Tcl interpreter encounters a line like `facebook "Updating my status via SECE."` in the user’s script, it calls the `cmdProc` method of registered `FacebookCmd` object. The method updates the user’s Facebook status using a client library. This Facebook client library requires an OAuth token to authorize the Facebook user. `FacebookCmd` object reads this OAuth token from the user’s registry database. This token is inserted into the user’s registry database while the user registers his Facebook account with SECE using the web interface (Figure 5.18).

Listing 5.44: The source code of FacebookCmd class which updates the user’s Facebook status

```java
package edu.columbia.lucs.tcl;

import com.restfb.DefaultFacebookClient;
import com.restfb.FacebookClient;
import com.restfb.Parameter;
import com.restfb.exception.FacebookException;
import com.restfb.types.FacebookType;
import edu.columbia.lucs.Manager;
import java.util.logging.Level;
import java.util.logging.Logger;
import tcl.lang.);

public class FacebookCmd implements Command {
    Manager man;

    public FacebookCmd(Manager man) {
        this.man = man;
```
New action commands can be added to SECE by just creating a new class which implements the Tcl’s `Command` interface. Currently, new action commands have to added to the interpreter manually using sentences like `interp.createCommand("facebook", new FacebookCmd(this));`. This procedure requires compiling the SECE source code again and then restarting the server. Future implementations of SECE can automate this stage easily by scanning the classes in action commands directory and adding all of them to the interpreter automatically. This way, no compilation or restart will be required.
5.10 A graphical user interface for SECE

SECE has a web interface to manage rules, registry, logs, location polygons and third party service subscriptions. Initially, I had developed a Java-based GUI, but then we decided to design a web interface which allows to access SECE from mobile devices. Currently, we are testing SECE in our test server located in Columbia University\(^\text{14}\).

We used Google Web Toolkit (GWT) \(^\text{[gwt, 2010]}\) to implement the web interface. GWT allows developers to design and implement their GUIs in Java, and then it compiles them into a web application based on Asynchronous JavaScript and XML (AJAX) \(^\text{[van Kesteren, 2007]}\). Users of a SECE server first create an account, then login. The homepage of the web interface lists all the rules (Figure 5.12). Clicking on a rule header reveals the rule body. For each rule, there are delete and modify buttons. In the bottom of the page, there is a button which allow users to add new rules.

The architecture of the GUI component is shown in Figure 5.13. The web application communicates with the GWT server via asynchronous HTTP XML requests. The web application does not need to send a request for each user interaction because several operations like displaying the rule body in the homepage are handled by Javascripts. After a successful login, the GWT web application sends an AJAX request to retrieve all the rules. The GWT server retrieves the rules directly from the database and returns them to the web application. For all retrieval operations, the GWT server queries the database and returns the result without disturbing the SECE server. But for updates or insertions of a rule or registry entry, the GWT server forwards the request to the SECE server. SECE server restarts the rule or updates the registry entry and reflects these changes to the database.

There is another web page to edit existing rules or add new rules (Figure 5.14). The user writes the rule header to the textbox titled “Rule Header” and the rule body to the textbox titled “Rule Body”. There is an option to execute the Tcl code to detect problems. If this option is checked, the SECE server will trigger rule as if an event had occurred. If there is a problem in the syntax of rule body, rule header or Tcl part, the web interface will inform the end-user and will not save or run the rule until the problem is fixed.

\(^{14}\)http://lagrange.cs.columbia.edu/
Figure 5.12: SECE homepage
The rule editor helps users to prepare their rules via several assistive features. Although rule header and body looks like natural language, we don’t expect users to remember all rule header types and their formats. Also, informing the user about all supported rule types, rule formats, action commands and their syntax helps them to prepare their rules easily.

The first assistive feature is the combobox with several example rules, as shown in Figure 5.13. We provide an example rule for each rule type. When user selects one of these rules the rule header and body parts display the example rule. Then, users can modify this rule to suit their needs. The second assistive feature is the action icons panel. These icons are positioned over the rule body textbox. When user clicks one of these icons, the editor inserts action command to the current cursor position and updates the action command info boxes which are located under the rule body textbox.

The last assistive feature is the panel with the action command information boxes. Under the rule body textbox there are three components which give information about action commands: a combobox, a label, and a multiline textbox.

The combobox lists all possible action commands (Figure 5.14). This action command list display all possible commands, while the action command icons display only most commonly used action commands.

The label next to the combobox displays the syntax of the selected action command. For example, if the user selects “email - sends an email” from the combobox this syntax label
CHAPTER 5. SECE: SENSE EVERYTHING, CONTROL EVERYTHING

will display “email email_address subject body” which means the “email” action command takes three parameters the email address of the receiver, the subject of the email and then the text of the email message.

The multiline textbox under the action command list displays detailed information about the functionality of the selected action command. The detailed information is brief for some commands such as “email” and comprehensive for other commands like “incoming” (Listing 5.45).

Listing 5.45: The detailed information displayed in multiline textbox on how-to use the incoming action command

<table>
<thead>
<tr>
<th>Field names are</th>
<th>origin</th>
<th>destination</th>
<th>content</th>
<th>timestamp</th>
<th>subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieves information about incoming call/sms/im/voicemail/email.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The editor also warns the user about incompatibilities. For example, in Figure 5.16, the user selects “incoming” action command while editing a time-based rule; the editor displays a warning message “incoming is not available for this rule type.” in red color to warn to the user.

The rule editor interface can be improved later without affecting the other parts of the system. Future generations of the SECE system most probably will have a more advanced easy-to-use interface. One possible interface may look like Lego Mindstorms software [min 2011]. In this interface (Figure 5.17) instead of writing rules, user express their rules connecting and configuring action and event components.

Under the configuration tab, there are registry, log and accounts web pages (Figure 5.18). Registry page allows to monitor and modify registry entries. The “Logs” page displays the log activity of the SECE server. The “Accounts” page allows users to register their third-party web services accounts with SECE using OAuth [Hammer-Lahav, 2010] mechanism.
Figure 5.14: Rule editing interface
Figure 5.15: Example rule assistance feature
Figure 5.16: Action commands assistance feature
Figure 5.17: Lego Mindstorms software
Figure 5.18: Registration of third-party services to SECE
5.11 Summary and evaluation of the SECE

SECE consists of almost twenty thousand lines of Java source code and two thousand lines of grammar definition to describe the sublanguages. The ANTLR-generated parser and lexer consist of fifteen thousand lines of Java source code. Fifty different third-party Java libraries are used.

We have tried to design the software architecture of SECE to be extensible using our fifteen years of programming experience. Due to its extensibility, we were able to easily add new event types and action commands on top the core SECE components. We tried to select most current technologies to build SECE. For example, for the user interface we have picked the GWT (Google Web Toolkit) which enables any devices regardless of its form factor to connect and use the SECE.

The OAuth authentication protocol has enabled us to build the SECE without collecting user’s credentials. This kind of authentication protocol is a must to build a system which aggregates third-party services. Beside OAuth, we used several Java APIs to communicate with third-party services such as Facebook, Twitter, Gmail, and Google Maps.

The main strengths of the SECE are: (1) enables non-technical end-users to create services using a natural-English-like syntax, (2) enables users to use information coming from online social networks, communication networks, sensor networks, and presence and IM networks while creating their rules, (3) enables users to forward, reject or accept incoming calls using the other information mentioned above, (4) enables to create rules which sends reminders via email, SMS or IM, (5) assists users to create and update rules easily using the web-based rule editor, (6) adding new event types and action commands are easy due to its extensible architecture, (7) runs on all operating systems due to its development programming language Java.

The main weaknesses of the SECE: (1) Tcl, SECE’s scripting language, may not be suitable for end users with very limited technical skills, (2) the rule editor may not be easy to use for everybody, (3) the sublanguages grammar file is not modularized which makes it hard to modify and extent. To address these weaknesses, future implementations of SECE may provide a visual user interface in addition to a textual one and may divide the grammar file into more managable pieces.
While selecting which third-party services to include we have focused on online social networks, communication networks such as email, SMS and phone calls, presence and IM networks, location and calendaring services. The reason behind this selection is that these services are part of most people's daily life and their integration to the SECE allows users to control and manage them better. We have tried to cover some of the most popular services from each domain. For instance, we have integrated Facebook and Twitter but left out Linkedin due to time constraints.

In conclusion, I have designed and implemented the SECE system which enables non-technical end-users to create communication-related services. Our implementation of the SECE supports five different event types based on time, location, calendar, communication and context events. Fifteen action commands are introduced which allows the user to change his social network status, to send an IM, email or SMS, to tweet, to initiate, forward, reject or accept a phone call, to translate any text to any language, to generate and publish an image to Flickr from sentences, to schedule an event, and to access information about a calendar event with a single line of statement like `sms Bob "meeting in 5 minutes."`.

SECE offers one of the most extensive exploration of how to enable end users with very limited technical skills to create communication-related services.

\[^{15}\text{http://www.linkedin.com/}\]
Chapter 6

Conclusions

High performance real-time communication tools I have developed enhance users’ communication experience. Using BASS, they can share an application with their peers without fear of privacy and security risks. Using SECE, they can control their real-time communication services in a context-aware, human-oriented, integrative and proactive way.

Real-time communication has specific requirements; low latency and robustness to wireless or congestion-based packet losses. Insights from the measurements I have conducted highlights the importance of bandwidth adaption and loss differentiation for real-time video conferencing applications. Without these two mechanisms, user experience can suffer from network congestion or wireless packet losses.

vDelay can be used to measure the real-time performance of existing or newly-introduced video conferencing tools by reviewers and tester. Together with bitrate, the application designer can use CDL and frame-rate statistics gathered using vDelay to determine if the measured performance of the video chat application meets the expected performance under various network conditions. vDelay can be improved to assess the quality of video image beside measuring CDL and frame rate. CDL and frame rate of video calls on mobile devices can be measured using vDelay.

Due to time constraints I could not analyze the effect of mobility for real-time communications. Video calls over IP networks on mobile devices are becoming popular. Low CPU power and low bandwidth may result high latency due to video encoding/decoding and network propagation times. Performance of popular mobile video chat applications Skype,
Fring and Facetime can be measured in a future study. Similarly, application sharing can suffer from limited CPU power, low screen resolution and low bandwidth on mobile devices. Due to their low resolution screen, mobile devices have to scale down the received image. Pre-scaling the screen update on the desktop computer may increase the battery life and performance of mobile devices.

User experience is not only important for real-time communication services but also for controlling them via SECE. A user evaluation study has to be made to assess how easy to use SECE by end users with limited technical skills. According to the results of this study a GUI-based rule editor can be developed.
Bibliography


[phi, 2010] Phidgets Inc. - Unique and Easy to Use USB Interfaces. [Online; accessed 18-April-2011].


