

# **Projects in Computer Science**

## **COMS E6901**

### Final Report on **pTHINC Audio Transport and Latency**

Prepared for Professor Jason Nieh  
February 4, 2009  
John Morales

## Introduction

THINC [1] is a remote display architecture created at Columbia University that is aimed at achieving near-native performance through a thin-client system. Highlights of the architecture include a custom display protocol that enables simple but powerful compression, support for bi-directional audio, support for video playback, and finally encryption over SSL. This advanced functionality is made possible to some extent due to the computational resources available at the client. This leads us to the topic of this paper, pTHINC [2], a thin-client architecture for mobile devices.

The following report will examine and document some of the challenges faced to enable the pTHINC client to support VoIP applications on a device where, compared to the desktop client version of the client, resources are now constrained.

## Primary Objectives

The focus of the research completed during E6901 was on improving the pTHINC thin-client's audio performance, with a special emphasis on VoIP audio quality and mouth-to-ear latency. Previous work [3] on this topic saw some difficulty with respect to these two metrics, and so the focus of this effort was to investigate the following potential causes and possible solutions:

- Latency is a result of audio being transported over TCP.
  - Using UDP for audio transport could alleviate some of the resource overhead and improve mouth-to-ear latency.
- Quality suffers due to a lack of available network bandwidth.
  - Providing compression in either the general form of zlib, or a voice-specific solution such as speex [4], might improve quality by reducing the bandwidth required.
- Quality suffers due to network jitter.
  - By experimenting with the amount of audio buffering on the pTHINC client, one might be able to reduce network jitter's impact on quality.

## Overview of Contributions

The work completed during this project has produced the following results:

- Audio may now (optionally) be transported over UDP instead of TCP.
- When audio is transported over TCP, mouth-to-ear latency is about 321 ms on average. When sent over UDP, the latency improves with an average of about 292 ms. This corresponds to an increase in latency of about 110 ms compared to a native VoIP application running on the device (more on this later).
- pTHINC now provides an acceptable level of audio quality (on either TCP or UDP).

## Audio Performance Considerations

At a high level, the primary bottleneck of bi-directional audio quality is limited CPU resources on the device (with the possible addition of memory bandwidth as well). This was determined after implementing and testing a UDP path for audio transmission and seeing no improvement in quality, which led to much experimentation to find that specific optimizations made all the difference. These optimizations, this will now be discussed in more detail, focused on buffer block sizes and the total amount of buffering space allocated for playback and recording.

Interestingly, finding the optimal buffer configuration was tricky as VoIP audio quality and music streaming quality have, generally speaking, competing and somewhat opposite requirements; VoIP latency benefits the most from shorter buffers, which reduces the latency of sending and receiving audio data, however music streaming prefers larger buffers it allows the device to better handle the byte rate of a song, which is typically much higher than that of a VoIP call.

One primary issue encountered during the project, however, was that Skype did not fit the traditional audio model for VoIP. That is, because Skype is known to carefully select a call's bitrate dynamically based upon what it can discover about the current network conditions [7], a Skype call might specify a very ambitious quality of 48 kHz, 16-bit sampling in stereo. As a result the audio quality of a Skype call over the pTHINC client was more jittery compared to the Ekiga softphone, which samples at a more modest 16 kHz, 16-bit, mono setting. (This behavior also led to the decision to focus on Ekiga for latency results as the call quality and behavior was more stable than Skype.)

The process of deciding the most suitable buffer settings was done by a brute force approach with subjectively measuring the audio quality and latency over range of buffer combinations. The number of playback buffer slots was tested on all power of 2 increments from 4 to 256 and recording buffer slots from 4 to 128. Block sizes were varied from the original 4 kB down to 512 kB. As previously stated, if one were simply looking for the best mp3 streaming quality, one would definitely choose 256 for playback slots. The downside of this of course is that latency values of a VoIP call with 256 slots were observed to be on the order of whole seconds—not very practical.

In the end, the following values were determined to be the best compromise on VoIP quality and latency as well as music streaming quality:

- 64 Playback buffer slots, each sized at 1 kB.
- 16 Recording buffer slots, each sized at 1 kB.
- 1 kB block sizes

Having the amount of buffer space skewed toward playback allowed for the best compromise on VoIP vs. streaming; by reducing the amount of recording buffer, voice

latency from the device was kept acceptable, while music streaming benefits from the increased amount of playback buffer but not so much that it impairs VoIP.

The last challenge of audio quality was the UDP packet payload and audio buffer block sizes. Initially, it seemed that UDP was providing inferior performance to TCP, however it was later discovered that this was actually caused by the UDP packet payload size being too large. Set at 4 kB, this large payload was seemingly overwhelming the device's ability to send out the audio data recorded at the microphone. Reducing this value to 1 kB significantly improved VoIP's audio quality when using UDP, especially in the case of using Skype since, depending on network conditions, Skype specifies a very ambitious recording quality of 48 kHz, 16-bit sampling in stereo. Reducing the client's internal block size from 4 down to 1 kB also helped the quality of communications over TCP as well.

Lastly, it was noticed that increasing the compiler optimization on the pTHINC client, and setting the Windows Mobile CPU throttling to Max Performance, also seemed to help audio quality to a lesser degree.

## **THINC Architecture and Development Environment**

A basic overview of the pTHINC architecture is as follows:

### **THINC Server**

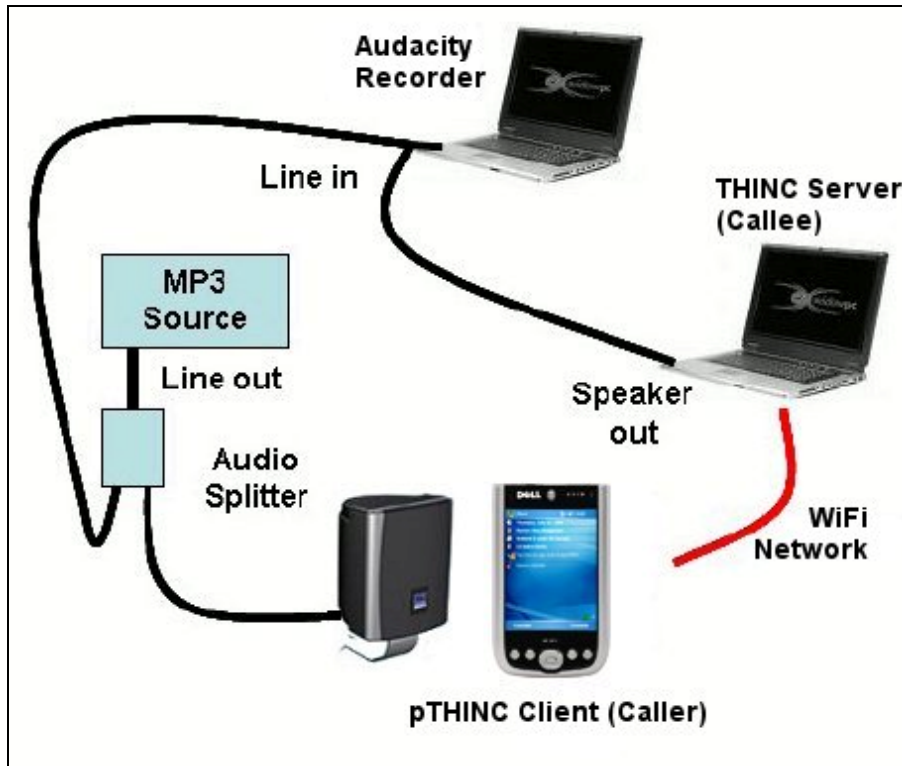
The THINC server and sound daemon (tsd) can be run on any Linux-based commodity PC. The machine used for this project's development and testing is a Dell E1705 laptop with a 2.0 GHz Core 2 Duo and 2 GB RAM. Development took place using Visual Studio 2005 on Windows XP Professional, while the THINC server ran on a virtual instance of Ubuntu Edgy (6.10) OS using VMWare Workstation v6.0.2.

### **pTHINC Client Device**

The thin-client device is a Dell Axim X51v running Windows Mobile 5.0. The device features a 624 MHz ARM processor, 64 MB of RAM, 802.11 b/g WiFi and Bluetooth.

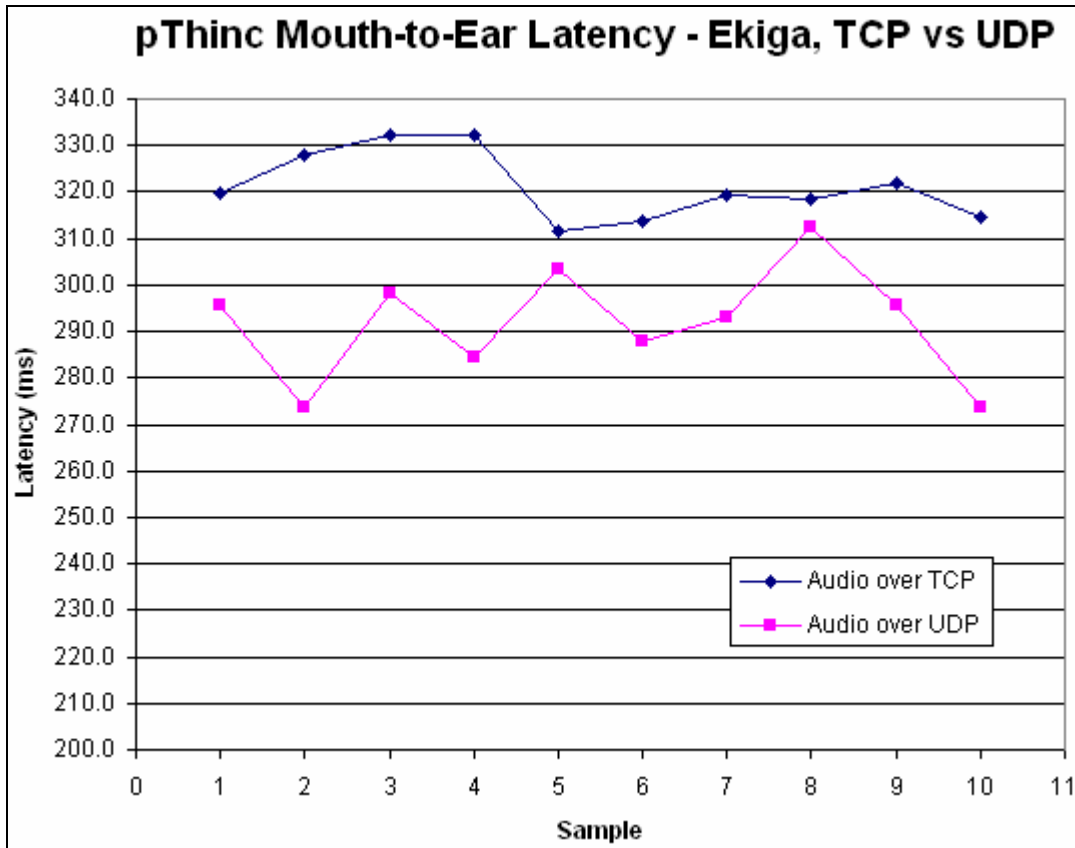
## **Mouth-to-Ear Latency Measurements**

VoIP call latency measurements were carried out with the help of the Audacity tool according to the following diagram:



As the diagram illustrates, a sound (metronome) was played from an MP3 source and split into two branches. One branch went through the end-to-end pTHINC architecture, namely to a speaker which allowed the metronome to be recorded by Ekiga running on the pTHINC client. The client then sends the sound to the THINC server's sound daemon across the wireless network. Once received by the sound daemon, the sound is played out through the laptop's headphone jack and recombined with the other direct line from the other branch. This combined signal is then recorded by Audacity, which was used to determine the timing difference between the two paths.

The following graph compares the latency results of sending audio over UDP vs TCP:



It can be observed from the graph above that utilizing UDP offers about a 29 ms improvement in latency over TCP on average. For comparison's sake and curiosity, a latency measurement was also taken with Skype running natively (i.e., not involving the THINC architecture) on both the device and callee machine, and that produced an average latency of about 190 ms. This result seems rather surprising given all the computational overhead going on to support the whole THINC architecture only adds roughly 100 ms of latency (ignoring any latency difference between Ekiga and Skype).

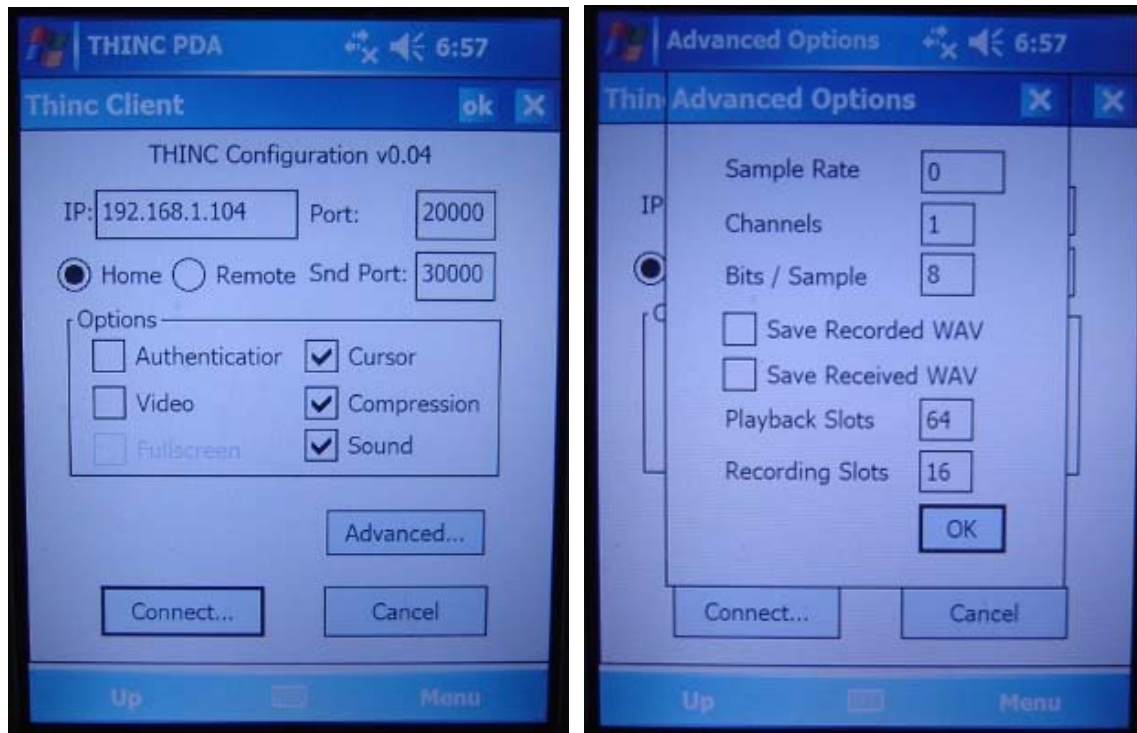
### Transport Layer Impact on Video Quality

Transporting audio over UDP also seemed to benefit video playback compared to TCP. Please refer to the two videos [5][6] illustrate the differences. You will notice that the video with UDP increases the frame rate by a factor of 2-3 compared to that of TCP.

### Secondary Contributions

To support the debugging and general use of the pTHINC client, some enhancements were added to the user interface. Most noticeably, an Advanced settings window was added to allow a user to customize the amount of playback and recording buffer that is utilized. This proved instrumental for debugging the UDP audio path as well as finding the buffer equilibrium between VoIP and music streaming quality.

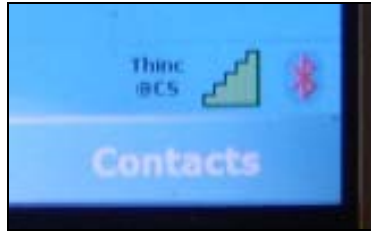
Screenshots of the new window and accompanying modification to the main window are pictured below:



The audio parameters at the top of the Advanced Options window were part of an earlier debugging effort; they allow a user to specify how the audio is recorded from the microphone. In general, however, the user should not manually specify the recording parameters and instead use those received from the sound daemon's `thinc_sndOpen` message. Setting any of the 3 audio properties to zero (as shown in the screenshot) indicates that the client should accept the sound daemon's parameters like usual.

In addition, the Advanced window also has options to make the client write WAV files of the audio recorded from the microphone before being sent to the THINC server, or the audio received from the THINC server before being played out. It should be noted, however, that these options incur a fairly significant performance overhead (especially when both options are enabled) and poor quality heard in the recordings might be a result of resource over-utilization rather than an actual audio problem (a lesson this author learned the hard way).

Lastly, a Windows Mobile installation project was created for pTHINC so the client may be installed via the standard Windows Mobile method of a CAB file. This registers the application with the OS, places a shortcut on the programs menu, and allows the application to be removed with the Add/Remove Programs utility. Furthermore, a system tray icon and functionality was added so that the application may be minimized to the system tray with a "Thinc @ CS" icon:



## Conclusion

To summarize, the primary limiting factor of audio quality over pTHINC is limited computational resources at the device. Improvements were realized through careful tweaking of memory and CPU memory usage. That said, it seems that network bandwidth does not play a part and therefore the usage of audio compression may actually compound the computation issues already present. Finally, due to the observed benefits of reduced VoIP call latency and increased video frame rate, it is recommended that UDP be the preferred method of audio transport.

Special thanks go to Professor Nieh for his patience and encouragement during this extended project.

## Bibliography

- [1] R. Baratto, J. Nieh, L. Kim, THINC: A Remote Display Architecture for Thin-Client Computing. In *Technical Report CUCS-027-04*, July 2004
- [2] J. Kim, R. Baratto, J. Nieh, pTHINC: a Thin-Client Architecture for the Mobile Wireless Web, May 2006
- [3] D. Shetty, T. Singh, pTHINC project report, Sept 2007
- [4] <http://www.speex.org/>, Speex: A Free Codec For Free Speech
- [5] [http://www.columbia.edu/~jm2873/pthinc/pthinc\\_video\\_tcp.mpg](http://www.columbia.edu/~jm2873/pthinc/pthinc_video_tcp.mpg), Movie of video playback over TCP
- [6] [http://www.columbia.edu/~jm2873/pthinc/pthinc\\_video\\_udp.mpg](http://www.columbia.edu/~jm2873/pthinc/pthinc_video_udp.mpg), Movie of video playback over UDP
- [7] S. Baset, H. Schulzrinne, An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol, In *INFOCOM 2006*