DEVELOPMENT OF A VIDEO STREAMING SYSTEM FOR TELEHEALTH APPLICATIONS

Arun Shankar Narayanan, Lam Woon Cherk, Tan Kok Kiong, Koh Choon Huat, Clarence W. de Silva
1, 2, 3, 4 National University of Singapore, Singapore
5 University of British Columbia, Canada
1 a0035708@nus.edu.sg, 2 kktn@nus.edu.sg

ABSTRACT
In this paper, the design of a video streaming system for telehealth applications is presented. Deviating from general-purpose objectives of off-the-shelf teleconferencing solutions, the system is customized to enable doctors and patients to be virtually linked over a broad range of connection speeds yet maintaining a satisfactory video/audio streaming experience to carry out remote diagnosis and assessment. Specifically, a key objective is to maintain a frame rate of around 15 fps at various connection speeds through an appropriate selection and characterization of transport protocols, encoding technologies and other innovations.

Through the use of the Microsoft’s Kinect sensor, under low bandwidth conditions, the system allows an optional streaming of human body structure in binary colour mode. Images are much smaller in size but sufficient for certain applications like tele-rehabilitation. The system also allows selective cropping for the user to discard the background but send only the useful information. Finally, through the use of adaptive streaming approach, the system is able to adapt the image quality to the network speed while maintaining the frame rate. Collectively, with these customized innovations incorporated, the system is able to perform video calls at frame rate of around 15 fps even in 128 kbps network.

KEY WORDS
Tele-health, video-streaming, adaptive streaming, selective streaming.

1. Introduction
While technology has contributed positively towards humanity in many aspects, huge potential remains to be tapped in the area of healthcare. Medical consultation today still takes place predominantly over face to face meetings. While this is still the preferred mode of medical practice, it is not economically or socially viable and sustainable in certain situations, such as an aging population, low medical personnel to patient ratios and a general scarcity of affordable medical resources in rural areas.

With the advent in computer and communication technology, simple diagnosis or follow-ups can be effectively done from home or a community focus point without the need for the physical presence of the patient in front of the doctor. Through the use of Internet and telecommunications, medical resources can be better optimized resulting in higher efficiency in the operations of the medical institutions and savings in cost, time and effort on the part of patients, particularly the groups dependent on caregivers for medical appointments and follow-ups.

The United Kingdom’s Department of Health has launched a two-year program named Whole System Demonstrator in 2008 to explore how technology can help people manage their health independently [1]. Involving more than 6000 patients and more than 200 general practitioners, the program ran telehealth trials on patients with medical problems such as diabetes, chronic heart failure and chronic obstructive pulmonary diseases. The results gathered were promising, showing statistics of 45% reduction in mortality rates, 20% reduction in emergency admissions and 8% reduction in tariff costs, among others.

A New York-based consulting firm, Health Research Institute Unit of PricewaterhouseCoopers, surveyed on 1000 consumers on their acceptance of telemedicine [2] and half of the respondents were willing to replace face-to-face non-emergency visits with telehealth technology.

Telehealth technology has much to offer to developing countries, where access to affordable medical treatment is limited. In fact, 87% of the 195 countries in the world are developing ones, and together, they represent a huge 84% of the total global population [3]. Conversely, the healthcare spending of these countries only constitute 11% of the total healthcare spending of the world, largely due to the lagging economies and very low doctor-to-patient ratios. Despite lagging in general healthcare, many of these developing countries are well equipped with basic telephony and Internet access due to the introduction of telecommunication technologies. For instance in 2011, statistics shows that 62% of the world’s total number of Internet users resides in developing countries [4]. With the availability of such basic infrastructure, it is structurally viable to deploy telehealth technology in developing countries to help bridge the gaps in medical services to the people at large and bring forth the following benefits [3]:

- Cost savings – e-consultations are about seven times cheaper than physical visits
- Easier access – rural and sub-urban populations do not need to travel to remote sites over difficult terrains for a few minutes of consultation
- Efficiency – optimize the distribution of limited medical resources to the mass population at large.
Telehealth systems can generally be placed into two categories; store-and-forward and real-time telehealth. In store-and-forward telehealth, patients capture various data in the form of images, video, audio, and other medical measurements in digital formats and transferred to the doctor using a digital link. Real-time telehealth systems establish real-time telecommunication links that allows the patients to interact with the doctors in either one-way or two-way mode.

This paper focuses on realizing a real-time telehealth system that allows doctors to diagnose or assess patients remotely with close interaction as though they are physically present in the clinic. The communication will be carried via video and audio channels. One key requirement is to sustain communication which allows satisfactory diagnosis and assessment over a wide range of bandwidths. This is crucial as a consistently high bandwidth cannot be taken for granted in both developed and developing countries. Besides, a higher bandwidth will most certainly be associated with a high cost to the consumer. Being able to operate under low bandwidth is very important especially in developing countries due to the limited network speed connections. Thus, a top challenge to be addressed in the paper is in the optimization of the usage of available bandwidth to create a low-latency video streaming system that is capable of running under low bandwidth. It should be reiterated that the focus of this paper is not to create another Skype-like video calling application, but to customize and optimize such a service to better meet the requirements of healthcare.

2. Literature Review

Most video streaming applications are designed for broadband connections. Nevertheless, there are some works focusing on narrowband connections. Gualdi et al. developed a low-latency video streaming system that is able to transmit images in CIF (352 x 288) format at 10fps over GPRS with 1.73 seconds latency on average [5]. The system is specifically targeting at low-bandwidth networks, and the video quality is kept at the same level even when the transmission is done over broadband network. Lim et al. also developed a narrowband video stream solution which achieved a modest frame rate over GPRS connection [6].

Many papers focused on adaptive streaming, whereby the bit-rate of the encoded images are adjusted based on the allowable bandwidth. Song and Golubchik devised an adaptive video streaming solution to deliver better video quality among other similar technologies [7]. However, Song gave emphasis to video quality rather than the efficiency of bandwidth usage which is the focus of this work. There are also mainstream products such as Microsoft IIS Smooth Streaming [8], Adobe Flash Dynamic Streaming [9] and Apple HTTP Adaptive Bitrate Streaming [10]. However, all of these products are intended for use in broadcasts for mass entertainment purposes. Leijdekkers et al. proposed a tele-monitoring solution that allows caregivers to monitor patients in real-time [11]. The system proposed was more of a monitoring system, where the caregivers are able to see the patients and there is no mention of the system’s performance under low bandwidth connection.

To the best of our knowledge, there is no work reported which has specifically addressed the aforementioned issues and challenges of video streaming for healthcare purposes. Each of the work attempts to address one (or a few) requirement(s), leaving the others intact or compromised. This paper focuses directly on performance cost functions which are critical for healthcare applications through appropriate selection of existing technologies, and measured customized innovations to bridge the gaps in off-the-shelf video conferencing solutions.

3. Proposed Solution

The main requirements to be met by the proposed system are listed as follows:

- Low bandwidth – the system is able to work under bandwidth as low as 128 kbps
- Low jitter – the system is able to sustain streaming without unacceptable lag
- Reasonable frame rate – the system is able to maintain a reasonable frame rate of about 15 fps
- Adaptation of video quality – the system is able to adapt key parameters to deliver adequate video quality which is optimal under the network constraints.

To meet these requirements concurrently, the system leverages on three key functionalities not commonly associated with standard video conferencing solutions.

1. Adaptive Streaming

Adaptive streaming adjusts aspects of the video quality based on the detected bandwidth. The adjustment can be in terms of the compression rate, frame rate and/or resolution. In the system, the objective function maintains the frame rate by varying the compression rate and resolution.

2. Depth Sensing

To further reduce the image size for smoother streaming, the application makes use of Microsoft’s Kinect sensor to detect the subject’s body structure, and represents it in red (body structure) and black (other objects). Since the images contain only two colours, the compression can yield encoded images of much smaller sizes compared to full coloured ones at the same resolution.

3 Selective Streaming

In many situations in telehealth, the interesting object of concern occupies only part of the full image. For example, the upper or lower limbs may be the areas to observe and assess in a therapy session while the other parts can be ignored. Thus, the system will allow the users to select only part of the video to be transmitted to the remote end and reduce file size further which is needed at low bandwidths.
4. System Architecture Overview

4.1 System Architecture

The overall system employs a distributed, connected configuration comprising of clients linked to a server over the Internet as shown in Fig. 1.

![Server-client architecture](image)

Fig. 1 – Server-client architecture

The server is on an open connection so that it can be publicly reached with the corresponding TCP (Transmission Control Protocol) port(s) unblocked. It is very common in a modern network that computers are placed behind network switches and routers, which in turn implement NAT (Network Address Translation), firewall and TCP port blocking features. NAT allows multiple computers to work behind a single router and a single IP (Internet Protocol) address. However, NAT may block some uncommon TCP ports. In other words, for a computer behind network routers to act as a server, special configuration needs to be put in place, including port forwarding and firewall unblocking.

The main purpose of the server is to facilitate the connection and communication between the clients. This allows both of the clients to be mobile, and special configuration is not needed before they can connect to the server. Such a configuration is amenable to multi-party communication to be supported via the server. The configuration is also scalable to situations when the clients need to be mobile. With the popularity of smart phones and tablets, it will become common for the doctors and patients to be linked in from multiple places with different network settings. A client-server setup further facilitates data security measures and user authentication.

4.2 Hardware Setup

The requirements for video streaming quality at the doctor’s client and the patient’s client are quite different. Accordingly, the hardware requirements differ too. At the doctor’s site, a computer equipped with a speaker, a microphone and a webcam is sufficient. At the patient’s site, the Microsoft’s Kinect sensor (Fig 2) replaces the standard webcam to provide extra functionalities.

![Kinect sensor](image)

Fig. 2 – Kinect sensor

The Kinect sensor is a special webcam-like device that has interesting features not found in a normal webcam, including depth sensing and skeletal tracking. During depth sensing, Kinect is able to detect the distance of the object from the sensor. By making use of the depth sensing, Kinect is able to separate objects located at different distances away from it, and thus detects human body skeletal structure between 0.8m to 4m from the sensor, based on the information [12]. The Kinect is also equipped with a normal RGB webcam feature to act as a substitute of the normal webcam. As will be further explained in later section, the system makes use of Kinect’s depth sensing technology to decrease image size while maintaining the same level of information sufficient for some telehealth applications.

Kinect is capable of returning colour and depth images at different format, resolution and frame rate. The configuration chosen in this system for colour imaging is (RGB, 640x480) while that of depth imaging is (640x480).

4.3 Software Setup

Different software platforms were considered for system development. Android platform was considered first but the limitation is that, it cannot accept external devices such as camera or audio device in case need to be included in further development stages. Another consideration was ASP.Net web development platform. In this case, image manipulation has limitations and encoding can only be done at server side. Third consideration was Microsoft .Net windows application development platform. This has the advantage of image manipulation as well as ability to accept multiple devices for future improvement as well as ease of use. Though it can only be run on windows platform, due to the wide acceptance and usage, it was chosen for final application development using Microsoft Visual Studio.

The system runs on two programs, one for the clients and one for the server. The software setup is the essence of the overall system. All the programs are written using C# language running on WPF (Windows Presentation Foundation) technology of the .Net Framework. The .Net Framework is a software programming framework for Microsoft Windows that has a set of comprehensive API (Application Programming Interface) and WPF is a presentation technology for User Interface development.

The high-level nature of C# language means that many of the low-level technical issues have been addressed. Also, the WPF technology facilitates the development of user interface better than similar frameworks of other languages such as C and C++.

5. Video and Audio Processing

5.1 Camera and Image Processing

Since the client program works with two types of cameras, it interfaces with them differently according to the camera type. For Kinect, the official Kinect for Windows SDK is used while DirectShowNet [13] open-source library is utilized for communicating with normal webcam. Another open-source library used is WriteableBitmapEx [14], which is capable of doing per-pixel manipulation.

5.1.1 Kinect

Kinect for Windows SDK supports two ways of getting image data from Kinect; event-based and polling-based method [12]. In the event-based method, an event handler is setup during program initialization to act as a callback
mechanism when the corresponding event is fired. There are four events available for different type of images, namely AllFramesReady, ColorFrameReady, DepthFrameReady and SkeletonFrameReady. Since each of the colour, depth and skeleton streams may have a different frame rate, AllFramesReady is fired when data for all streams are ready. In the polling-based method, the program actively queries the next available frame using the OpenNextFrame function of each stream. The polling can generally be done in for loops or using a timer. In the proposed system, polling is used to better control the frame rate whereby a will activate every 50 ms to obtain the next frame in the stream.

Data returned from the colour stream are available in either YUV or RGB format. RGB is chosen due to its simplicity. The data is stored in the form of a RGB byte array, and it takes 4 bytes to represent a pixel in the image. For example, if the image has a resolution of 640x480, the byte array returned will be 640 x480x4 bytes = 1,228,800 bytes in size. The 4 bytes that represent a pixel is made up of alpha, red, green, blue components and each takes up 1 byte.

For the depth stream, data is returned in a different format from the colour data. Instead of a RGB byte array, depth stream returns a byte array in which each pixel contains the distance from the camera to the nearest object within the camera’s field-of-view. Each pixel in the depth stream is represented by 16 bits to store the distance information. A 0 value in a pixel denotes that no data is available at the pixel, either because the object is too close or too far from the camera. To convert the 16-bit depth data into millimetre, the following formula is used [15]. Assuming $P_i$ represents a 16-bit depth data at a particular pixel, the distance, $D_i$ is then:

$$D_i = P_i \gg 3$$  \hspace{1cm} (1)

(where $\gg$ denotes the logical right shift operation)

To further simplify the processing of Kinect data, an open-source library named Coding4Fun Kinect Toolkit [16] is used. The library contains a set of helper methods for easy manipulation of Kinect data.

5.1.2 Webcam

On Windows platform, the most common way to access media devices (including webcams) is through the use of DirectShow [17]. DirectShow is an API for streaming media natively written in C++ based on Microsoft’s COM (Component Object Model) technology. However, there is no official support of DirectShow API for C#. Thus, this project makes use of the DirectShowNet open-source library, which is a wrapper around the original DirectShow API, to communicate with the webcam. Being a wrapper library, DirectShowNet merely provides a C# interface for other codes to call the underlying DirectShow C++ API.

The main building block for DirectShow is the filter [18] which is simply a component that receives input and produces output in a specific form. In the proposed system, the filters involved are Video Input filter for getting data from the video input (webcam) and Sample Grabber filter for obtaining individual samples from the video source.

5.1.3 Image Processing

After obtaining the images in raw format the program processes the images to render them suitable for transporting over the network. There are three main steps involved: cropping, scaling and encoding.

Cropping is done optionally and selectively on the image such that irrelevant area of the image is wiped off to yield a smaller and compact image size containing only the necessary data. The selection is carried out explicitly by the user by drawing and placing a rectangular window over the interesting part of the image. The program controls the crop area such that the selection is always within the image itself.

Assuming the original image has left-most corner position at $(X_i, Y_i)$ and has width and height $(W_i, H_i)$ while the selection has corresponding parameters of $(X_2, Y_2)$ and $(W_2, H_2)$, the program ensures that:

$$X_i \leq X_2 \leq X_i + W_1$$ \hspace{1cm} (2)
$$Y_i \leq Y_2 \leq Y_i + H_1$$ \hspace{1cm} (3)
$$W_2 \leq W_1 + X_1 - X_2$$ \hspace{1cm} (4)
$$H_2 \leq H_1 + Y_1 - Y_2$$ \hspace{1cm} (5)

Scaling is done to scale down the images to smaller sizes. In this project, three scaling factors are used; 1.0, 0.5 and 0.25. Since both Kinect and the webcam produce images at 640 x 480 resolutions, the scaling thus results in images with the following possible resolutions: 640 x 480, 320 x 240 and 160 x 120.

The final step of the manipulation is to encode the (optionally) cropped and scaled image before sending to the remote end. JPEG codec is invoked in the system using the built-in .Net API. The scaling factor and the compression rate are both determined based on the upstream bandwidth of the client. Basic characteristics of JPEG include image resolution priority at low bandwidth availability and consistent file size of images. With JPEG, adjustment of the compression scheme can be easily configured, enabling a balance between quality and video size to be adapted to the bandwidth.

5.2 Audio Recording and Playback

Other than video, the audio component is also a crucial part of the system. Two open-source libraries are used, namely NAudio [19] and NSpeex [20]. NAudio is used to record and playback audio data while NSpeex is used to encode and decode the audio data. Playback and recording are directly accomplished using NAudio. Basically, this system utilizes the WaveIn and WaveOut API of Window’s Waveform Audio Interface for the playback and recording [19]. Similar to DirectShowNet, NAudio is just a C# wrapper library over the C++ Waveform Audio Interface API. Speex is an open-source audio codec specifically designed for speech [21]. While Speex has many features similar to other codecs, it is chosen mainly because of its open-source nature and the availability of NSpeex library for implementation in C#. Speex supports compression in three bands; narrowband (8 kHz), wideband (16 kHz) and ultra-wideband (32 kHz) [21]. Narrowband is used here to conserve bandwidth.
6. Data Transfer

Once the audio/video data is acquired and processed, it will be transferred over to the server, which in turn sends the data to the other connected client. The system uses TCP as the transport layer protocol. TCP can provide reliable, ordered delivery of full-duplex data stream from the clients to the server and vice versa and this is needed for future development purpose as well.

6.1 Server Program and Client Program

The server program is a C# console program. The server program actively listens on ports 8826, 8827 and 8828 for incoming connections. These are used for video, audio and text message streams respectively. Upon receiving data from one end, the server loops through the connection list to see if there is any connected client on the other end. If so, the server continues to transmit data over to the available client.

To communicate with the other client on the remote end, a client first connects to the server through the three ports. After connections are established, the client continues to send data over the corresponding connection. For efficiency, the connections are kept open until the client disconnects. For transmission of video images, the sending client maintains a timer that fires every 67 milliseconds (approximate to 15 fps) to send video data over to the receiving client. Transmission of audio is similar to that of video while that of text data happens on demand as and when users input data.

6.2 Bandwidth Detection for Smooth Transfer of Video

One of the main features this system offers above standard teleconferencing programs is the ability to adjust the video quality according to upstream bandwidth. The bandwidth detection happens during the client program launch.

To measure the upstream bandwidth, a client first sends a packet of size 200 kB to the server and keeps track of the sent time. Upon receiving the complete packet, the server then acknowledges the receipt of the packet by sending a small acknowledge packet. Assuming 200kB (204,800 B) packet is sent from the client at time \( T_1 \) and the acknowledge packet is received from the server at time \( T_2 \), the bandwidth, \( B_w \) in bytes per second, of the client’s upstream can be estimated as

\[
B_w = 0.8 \times \frac{204800}{T_2 - T_1}
\]

Now that the allowable number of bytes in each frame at different network speeds is determined, the final step is to determine the JPEG file size at different compression quality and scaling factor that can fit snugly into the allowable bytes space. However, the JPEG file size cannot be determined directly from the compression rate and resolution, as the file size is determined only after the image has been encoded. To this end, an empirical approach is used to set up a look-up table by running multiple tests on different pairing of compression quality (10 to 65) and scaling factor (0.25, 0.5 and 1.0) of images selected to be representative of those to be transmitted during the actual application. A high value in compression quality means a low compression rate and thus results in large file size of the encoded image. The key idea is to fit video data of the highest, sustainable quality that just fall within the file size constraints of Table 2. For example, assuming the network speed is B kbps has allowing \( B_j \) bytes per frame, and that there are two sets of data in the form of file size, compression quality, scaling factor), which are \((S_i, F_i, Q_i)\) and \((S_j, F_j, Q_j)\) with the condition \( S_i \leq S_j \), \( F_i \leq F_j \), \( F_i \leq F_j \), the combination \((F_i, Q_i)\) is chosen over the other set. The look-up table is thus constructed in Table 3.

Table 1 – Allowable bytes in 1 second for different network speeds

<table>
<thead>
<tr>
<th>Network speed, B (kbps)</th>
<th>Bytes per second, ( X = B \times 1024 \div 8 )</th>
<th>Allowable bytes in 1 second, ( B_w = 0.8X )</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>16384</td>
<td>13107</td>
</tr>
<tr>
<td>256</td>
<td>32768</td>
<td>26214</td>
</tr>
</tbody>
</table>

6.3 Choosing the Image Quality for Streaming

Detecting the bandwidth is only the first step for smooth streaming of the video. The next step is to decide the desired corresponding compression rate and resolution such that the frame rate requirement of 15fps can be maintained. To accomplish this, it is imperative to know how much data can fit into each of the 15 frames at different network speeds. Table 2 shows the maximum size of each frame for different network speeds.

Table 2 – Allowable bytes in 1 frame for different network speeds

<table>
<thead>
<tr>
<th>Network Speed, B (kbps)</th>
<th>Allowable bytes in 1 second, ( B_w )</th>
<th>Allowable bytes in 1 frame, ( B_f = B_w / 15 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>13107</td>
<td>873</td>
</tr>
<tr>
<td>256</td>
<td>26214</td>
<td>1747</td>
</tr>
<tr>
<td>384</td>
<td>39321</td>
<td>2621</td>
</tr>
<tr>
<td>512</td>
<td>52428</td>
<td>3495</td>
</tr>
<tr>
<td>768</td>
<td>78643</td>
<td>5242</td>
</tr>
<tr>
<td>1024</td>
<td>104857</td>
<td>6990</td>
</tr>
<tr>
<td>2048</td>
<td>209715</td>
<td>13981</td>
</tr>
</tbody>
</table>

Now that the allowable number of bytes in each frame at different network speeds is determined, the final step is to determine the JPEG file size at different compression quality and scaling factor that can fit snugly into the allowable bytes space. However, the JPEG file size cannot be determined directly from the compression rate and resolution, as the file size is determined only after the image has been encoded. To this end, an empirical approach is used to set up a look-up table by running multiple tests on different pairing of compression quality (10 to 65) and scaling factor (0.25, 0.5 and 1.0) of images selected to be representative of those to be transmitted during the actual application. A high value in compression quality means a low compression rate and thus results in large file size of the encoded image. The key idea is to fit video data of the highest, sustainable quality that just fall within the file size constraints of Table 2. For example, assuming the network speed is B kbps has allowing \( B_j \) bytes per frame, and that there are two sets of data in the form of file size, compression quality, scaling factor), which are \((S_i, F_i, Q_i)\) and \((S_j, F_j, Q_j)\) with the condition \( S_i \leq S_j \), \( F_i \leq F_j \), \( F_i \leq F_j \), the combination \((F_i, Q_i)\) is chosen over the other set. The look-up table is thus constructed in Table 3.

Table 3 – Compression quality and scaling factor for different network speeds

<table>
<thead>
<tr>
<th>Network Speed, B (kbps)</th>
<th>Allowable bytes in 1 frame, ( B_f )</th>
<th>Compression Quality</th>
<th>Scaling Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>873</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>256</td>
<td>1747</td>
<td>12</td>
<td>0.25</td>
</tr>
<tr>
<td>384</td>
<td>2621</td>
<td>35</td>
<td>0.25</td>
</tr>
<tr>
<td>512</td>
<td>3495</td>
<td>60</td>
<td>0.25</td>
</tr>
<tr>
<td>768</td>
<td>5242</td>
<td>23</td>
<td>0.5</td>
</tr>
<tr>
<td>1024</td>
<td>6990</td>
<td>35</td>
<td>0.5</td>
</tr>
<tr>
<td>2048</td>
<td>13981</td>
<td>20</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Note that in the look-up table, there is no combination of compression quality and scaling factor that can fit into the allowable bytes for network speed of 128 kbps. This is when the measures to free up bandwidth can be invoked. Thus, to still achieve the frame rate of 15 fps on 128 kbps connection, the client program has to go beyond the 80% network capacity, and evoke bandwidth conservation measures of depth sensing and selective cropping.

6.4 Jitter and De-jitter Buffer
Although the audio and video packets are sent at fixed interval at the sending side, the same series of packets may arrive at the receiving side with jitter. Jitter represents the deviation from true periodicity of an assumed periodic signal, and it is an undesirable and inevitable effect associated with network congestion. Fig. 3 shows how a smooth periodic series of packets on the sending sides becomes a non-periodic series at the receiving side.

The phenomenon of jitter in audio and video packets at the receiving side can result in jerky and unsmooth motion in the video stream or noise in the audio stream received. A common workaround this problem is by using a de-jitter buffer for both the video and audio streams.

A de-jitter buffer is a fixed size buffer that queues the incoming packets. When there are enough packets for smooth playback, the packets is then de-queued to produce output by using a timer set at 15 fps. As larger buffer can lead to higher latency, it is designed to be large enough for de-jittering so as not to introduce unnecessary. In this system, the de-jitter buffer for video stream is 3 frames while that of audio stream is 30ms. These parameters are determined by repeated testing of the system.

7. Main Results
Key screenshots of the client and server programs are first presented to give an overall picture of the system operations. Fig. 5 shows the server console program that is responsible for facilitating communication between two clients.

Fig. 5 – Server console program

Fig. 6 shows the main window of the doctor’s client program while Fig. 7 shows the main window of the patient’s client program. The bordered area in Fig. 7 shows that patient’s client program has control over the Kinect’s settings. There is also a text message panel at the bottom right of the client window, allowing the conservation of bandwidth using text in lieu of audio conversation.

Fig. 6 – Doctor’s client program

Fig. 7 – Patient’s client program

Fig. 8 shows the patient side’s main window when Kinect’s depth sensing feature is turned on. Instead of normal colour video, the window is displaying (and sending over the network) a black and red video showing the structure of the nearest object.

Fig. 8 – Video with depth sensing turned on

Fig. 9 and Fig. 10 show the cropping feature that is available on the client programs. Note that only the cropped part is sent over the network, resulting in substantial reduction in the amount of data to be sent.
The most important quality factor of concern in this system is the frame rate of the video and the key objective function is thus to maintain a frame rate of around 15 fps at various connection speeds for satisfactory smooth video rendering. In what follows, the performance of the system will be evaluated in terms of the frame rate of the received video.

### 7.1 Analysis Environment and Methodology

Since each of the received frames is in the form of JPEG image, it is easy to measure the frame rate as each received image represents a frame. To measure the frame rate of the received video, the receiving client program is launched in the debug mode with a slightly modified programming code. At the place where the program is periodically getting the next frame from the de-jitter buffer, the code is injected with lines that check and record whether the de-jitter buffer is empty. If the buffer is empty, a frame is said to be dropped. By examining 1000 frames for each connection speeds, the ratio of dropped packet versus total packet is then calculated.

When evaluating the performance, the programs are run on different computers to reflect a realistic setup. To simulate different connection speeds, NetLimiter which is capable of manipulating the downstream and upstream bandwidth. The measurement is then carried out by limiting the upstream bandwidth of the sending side and the frame drop ratio is then examined at the receiving side.

### 7.2 Streaming Results

Table 4 reflects the streaming quality in terms of frame rate at each network speed. Each entry, corresponding to one network speed, is examined over the duration of 1000 frames for the total number of dropped frames, and the frame drop ratio was tabulated.

The results in Table 4 show that the system is able to maintain a frame rate of above 13 fps. The reason that a full 15 fps is not achieved is due to the excessive jitter in the network during the time of experiments causing frame drop in de-jitter buffer. However, 13 fps is an acceptable frame rate. Note that at 128 kbps, the frame rate falls below 13. As explained previously, there is no JPEG configuration that can ensure 15 fps at 128 kbps and features such as cropping and depth-sensing should be invoked at this time.

### 7.3 Comparison with Skype

Although the design of this system is guided by a set of requirements different from those for commercially available applications, Skype is used in a comparison study to evaluate the relative performance at low bandwidths. Since Skype is a proprietary program and codes cannot be injected into it, there is no rigorous way to compare it with this system.

Thus, the comparison is done using the following method. A video conferencing session using Skype is recorded. At the patient’s end, a short clip running at 15 fps, looping a sequence of numbers from 1 to 15 in each frame, is captured by the camera (refer here for clip: http://bit.ly/H11Iv1). At the doctor’s end, the video is received and recorded. Again, NetLimiter is used to limit the bandwidth at the uploading side to 128 kbps. The clip is recorded for a duration of three minutes and subsequently analysed by randomly choosing 10 instants of the recording. At each time instant, the frame rate is observed for one second to see if all numbers (“1” to “15”) that are displayed. Since each number takes up a frame, the number of numbers appearing over one second is equivalent to the frame rate. The results are tabulated in Table 5. Based on the median, Skype’s frame rate at 128 kbps is about 8 fps. Comparing to 13 fps achieved with this system, the results obtained shows that this system can perform better than Skype in terms of frame rate at low bandwidth.
However, no conclusion is drawn that Skype is overall a poorer general solution for video streaming. In fact, Skype strives to maintain a very high video quality at 128 kbps, thus resulting in network congestion. This system is able to circumvent that by lowering quality and scaling rate in response to the bandwidth via a delicate balancing act in a zero sum game in favour of the telehealth purpose. A comparison video has also been recorded (http://bit.ly/Hi7GRx) to show the relative outcome of Skype and this system. As can be seen from the video, Skype stops once in a while to buffer the video for as long as 8 seconds. This system, however, is able to maintain a steady playback.

8. Conclusion
In this paper, a system that is similar to standard teleconferencing applications, but has customized and optimized features for telehealth usage is designed and implemented. Through the use of features such as adaptive video quality, selective streaming via video cropping and depth sensing, the system is able to facilitate telehealth conferencing between doctors and clients while maintaining an acceptable video call quality of around 15fps.

There are several aspects which can be further improved. Among them, the more significant one is to include the bandwidth detection not only on the upstream bandwidth but on the downstream bandwidth as well. This addition can greatly improve the experience when both the clients have very different bandwidths (e.g. the upstream bandwidth of one side is different from the downstream bandwidth of the other side) so that some balancing can be done at the server.

Another important future improvement is to introduce a dynamic bandwidth detection algorithm. Currently, the detection is only done once at the start of the connection. However, available bandwidth in the real world is not fixed but fluctuates over time. Moreover, the users may be accessing other Internet content at the same time, causing the usable bandwidth to vary accordingly. To solve this problem, a dynamic bandwidth detection algorithm would be useful.

Finally, a more efficient video compression technology can be attempted. JPEG was chosen for the system for its ease of use especially in configuring compression rate. The efficiency of JPEG could be further enhanced relative to codec such as H.264 to improve the video quality and bandwidth usage efficiency.

References