



THE UNIVERSITY OF

MELBOURNE

COMP90015
DISTRIBUTED SYSTEMS

Project 2:
SimpleStreamer Webcam Conference

Author:

Meghan MANN

663657

Mingyu MA

629946

Professor:

Dr. Aaron HARWOOD

Semester 2, 2014

1 Bandwidth Consumption

Bandwidth refers to the amount of data that can be transmitted over a given time, usually reported in bytes per second. The default resolution of the images created by SimpleStreamer is **320 x 240 pixels**. With 3 bytes per pixel for RGB colors, a raw image is 230400 bytes. Ten images are sent per second by the stream, so with no compression algorithm applied to the data **2.3MB/s of bandwidth is required**.

1.1 Ways to Reduce Bandwidth

1.1.1 Reduce Image Resolution

The resolution of the images to be sent can be reduced before sending in order to reduce the amount of bytes that must be sent per image.

1.1.2 Reduce Color Depth

Color Depth is the amount of bits used to specify to color of a pixel. In this project, we are using **24 bit RGB**, which uses 8 bits to specify R, B, and G (requiring 3 bytes per pixel in total). This mode is most common for images, such as JPG. However, by converting images to a different color mode, such as 16 bit Grayscale, we can reduce the amount of bandwidth required to transmit them.

1.1.3 Reduce Frame Rate

The **frame rate**, or rate that images are written a socket, can be specified in order to reduce the amount of bandwidth consumed. The frame rate can be reduced by introducing a **rate limit**, as is done in our project. The rate limit specifies the amount of time that a thread should wait in between sending images. The default rate limit of our project is **100ms**, meaning a maximum **ten images per second** can be sent.

1.1.4 Compression Algorithms

Compression Algorithms can be applied to data before it is sent in order to reduce the bandwidth. The data is then **decompressed** once it reaches its destination and is processed accordingly. In the project, we are required to use the **GZIP** algorithm to compress images before they are sent, however, more efficient alternative exist. One aspect to consider when selecting a compression algorithm is whether it produces **lossy** or **lossless** compressed images.

- **Lossy vs. Lossless Compression**

Lossy image compression algorithms remove data from an image in order to

provide an approximation of the image in which it appears the same as the original. This results in an image being represented by fewer pixels. The original, uncompressed image **cannot** be reconstructed by the compressed image. Examples include **JPEG** compression.[4] Lossless image compression results in the original, uncompressed image being able to be reconstructed from compressed image when decompressed. Examples include **PNG** and **GIF** files. This type of compression is typically used for images that are archived. [3] For the purpose of SimpleStreamer, lossy compression is a suitable choice for images do not have to be saved or of particularly high quality.

As stated earlier, the current compression model we are using for SimpleStreamer is **GZIP**, a **lossless** compression method that will work on any stream of bytes. However, it is not an ideal compression algorithm for images and is better suited for text. When applied to the raw images in our data, it will only provide a **compression ratio of 1.7:1**.

A better alternative would be **JPEG**, which is a very popular **lossy** compression method for images. The amount of bits per pixel in for the compressed image can be specified, which results in different qualities of images. The highest quality JPEG images use 8.25 bits per pixel, resulting in a **2.6:1 compression ratio** for 24 bit RGB images. Lower quality images can have a **compression ratio of 46:1**. [2]





Image	Quality	Size (bytes)	Compression ratio
	Highest quality (Q = 100)	83,261	2.6:1
	High quality (Q = 50)	15,138	15:1
	Medium quality (Q = 25)	9,553	23:1
	Low quality (Q = 10)	4,787	46:1

Figure 1: JPEG Compression [2]

By comparing the compression ratios of GZIP and JPEG, it is easy to see that our compression rate could have been increased by using a different compression technique, which would effectively **decrease the bandwidth** required.

2 Security

2.1 Security Threats

The implementation of our program exposes the users to security threats, for they are not communicating on a **secure channel**. The receiver's socket is open to receive data from anyone who sends a message to it, and does not require that the data received is authenticated to be from the user they expect before it is processed. Furthermore, the data sent by the sender is not encrypted, so it can be accessed by other users it is not intended for.

2.1.1 Eavesdropping

Eavesdropping involves obtaining copies of messages sent by a user without their permission. Because the messages sent are not **encrypted**, if they are intercepted by someone they are not intended for the interceptor will be able to interpret them without requiring additional information to for their decryption.

2.1.2 Masquerading

Masquerading involves sending or receiving messages using the identity of a user without their permission or being aware. Listening on a socket for incoming image data opens up a receiver to potential threat because the the messages received are not authenticated. This means it is not guaranteed that they are from the source which the user believes them to be. A receiver may receive data from unwanted sources **masquerading** as the source they are expecting a message from.

2.1.3 Message tampering

Message tampering involves intercepting messages, altering their contents, and then passing them on to the intended receiver. This is a possibility with our program as the data received is not authenticated or encrypted.

2.2 How the Protocol Could Be Changed To Address Security Concerns

In order to provide users of our program with a secure channel, the **Secure socket layer protocol (SSL)** can be used in order to ensure that message received are from the intended source and have not been tampered with. SSL can be implemented before any messages are exchanged. Messages are encrypted so they cannot be intercepted or altered, and the server is authenticated.

2.2.1 SSL

SSL is a protocol for establishing a secure connection between two hosts. A Java library exists that would allow us to implement SSL on our project.

Steps In Creating An SSL Connection:[1]

1. If the server does not have a certificate with a public key, they must create one. This can be done using **keytool** that is part of **J2SE SDK**.
2. The sender and receiver exchange messages to establish compression and cryptographic algorithms to use.
3. Once agreed upon, the receiver sends the sender a certificate containing their **public key**, which must be signed by a trusted authority.
4. The sender replies with a randomly generated **premaster key** encrypted with the sender's public key.
5. A **session key** is now created from the exchanged keys, and is used to encrypt all further correspondence.

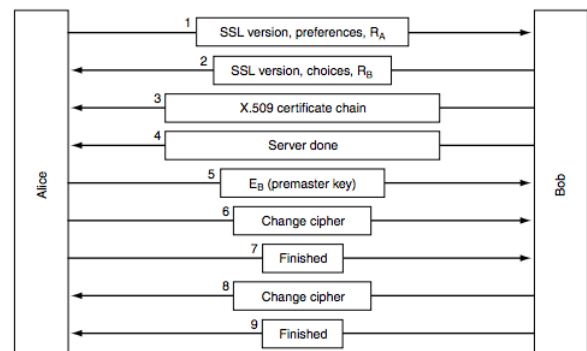


Figure 2: SSL Protocol Diagram [1]

Although the receiver has been authenticated by the sender (they have sent a certificate doing so), the sender has not been authenticated by the receiver. At this point, the receiver may request that the sender to send their public key or **log in** to a previously established user account.

3 Scalability and Rate Limiting

3.1 Rate Limiting

Rate limiting enforces a limit on the amount of images that can be written to a socket per second.

This is achieved by forcing each thread to pause for a certain period of time or **sleep interval** before writing another image. The default rate limit of our project is **100 ms**, meaning that a maximum of **10 images per second** can be written to a given socket.

3.1.1 Why is Rate Limiting Done?

The rate at which images can be written to a socket is determined by the bandwidth available to the sender, receiver, and network. Using **TCP protocol**, images will automatically no longer be able to be written to a socket once the sender's socket's buffer is filled, meaning that they are waiting for data to reach the receiver. However, using this approach alone and writing the maximum amount of images possible to sockets will saturate the bandwidth. **Rate limiting** is done in order to reduce the amount of images sent per second, therefore **regulating the amount of bandwidth used**.

3.2 Scalability

Scalability refers to the amount of user that the a user can simultaneously stream with.

3.2.1 Does Rate Limiting Achieve Scalability?

Rate limiting does help achieve scalability by ensuring that each socket is serviced equally. However, adding multiple connections will still put a heavy load on the network with the current bandwidth required. Better scalability could be achieved by implementing better compression algorithms.

4 Improvements

The implementation of SimpleStreamer could be improved in various ways.

- **Security**

Currently, there are no security protocols to ensure that the sender and receiver are communicating on a secure channel. Implementing SSL protocol as described previously (perhaps along with a log-in authentication for clients) would be highly beneficial to the users of our program.

- **Compression**

SimpleStreamer currently uses **GZIP** to compress the images being transmitted. However, this is not the ideal compression

technique. Using an alternate model specifically catered to image compression such as **JPEG** could reduce the bandwidth to **only 5%** of what GZIP requires.

References

- [1] David J. Wetherall Andrew S. Tanenbaum. *Computer Networks*. Pearson, fifth edition, 2011.
- [2] Wikipedia. Jpeg — wikipedia, the free encyclopedia, 2014. [Online; accessed 23-October-2014].
- [3] Wikipedia. Lossless compression — wikipedia, the free encyclopedia, 2014. [Online; accessed 23-October-2014].
- [4] Wikipedia. Lossy compression — wikipedia, the free encyclopedia, 2014. [Online; accessed 23-October-2014].