國立交通大學

資訊科學與工程研究所

碩 士 論 文

使用排列陣列的隱藏通訊系統

A Steganographic Communication System with Permutation Arrays

研 究 生：李慈潔
指導教授：蔡錫鈞 教授

中 華 民 國 101 年 6 月

使用排列陣列的隱藏通訊系統
A Steganographic Communication System with Permutation Arrays

研 究 生：李慈潔　　Student : Tzu-Chieh Li

指導教授：蔡錫鈞　　Advisor : Shi-Chun Tsai

國立交通大學
資訊科學與工程研究所
碩士論文

A Thesis
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in
Computer Science and Engineering
June 2012
Hsinchu, Taiwan, Republic of China

中華民國101年6月

國立交通大學

研究所碩士班

論文口試委員會審定書

本校 資訊科學與工程 研究所 李慈潔 君

所提論文:

使用排列陣列的隱藏通訊系統

合於碩士資格水準、業經本委員會評審認可。

口試委員：

指導教授：

所　　長：

中華民國 101 年 6 月 26 日

Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
Hsinchu, Taiwan, R.O.C.

As members of the Final Examination Committee, we certify that we have read the thesis prepared by Tzu Chieh Li entitled A Steganographic Communication System with Permutation Arrays

and recommend to be accepted as fulfilling the thesis requirements for the Degree of Master of Science.

Committee Members:

Thesis Advisor:

Director:

Date: 101. 6. 26

摘　　要

本論文應用兩個演算法來實作隱藏通訊系統，mapping演算法和embedding演算法。首先，根據排列陣列，mapping演算法利用k-buffer排列器來產生一個排列陣列，此可抵抗敵人的被動攻擊。而mapping演算法則是加入了隨機的元素來加強mapping演算法的安全性。隱藏通訊系統應用這裡個演算法來產生一個排列陣列，並利用了SCTP (Stream Control Transmission Protocol)封包中的傳輸序列號(Transmission Sequence Number)欄位來重新排列所要傳送的資料封包，藉此來傳送所要隱藏的資訊。在隱藏通訊系統中有兩個模型，多媒體模型和檔案模型。根據附信 (Cover letter)的類型和欲隱藏的秘密訊息來分類。當欲傳送的秘密訊息是一個檔案，則選擇多媒體資料型態來傳送，此為多媒體模型；若是要傳送字串為祕密訊息，則利用檔案來作為付信，此為檔案模型。

Abstract

We design a Steganographic Communication system for secure steganographs comumication. We apply the encoding and decoding algorithms for the permutation arrays to embed (extract) secret into (from) cover letters, respectively. We apply these schemes and use the Transmission Sequence Number field within SCTP packet format to reorder the sequence of packets. Our system can handle two models: one is for multimedia model and the other is for file model. They are classified according to the type of cover letter and secret message. We use multimedia as the cover letter while sending a file as secret message and file as the cover letter while sending a string as the secret message.

誌謝

首先，我最先要感謝的就是指導老師蔡錫鈞教授，在老師的細心督促與教誨之下，這篇論文才能順利完成。同時也要感謝實驗室的學長名全學長和旻錚學長給予解惑，使我能克服論文寫作上所遇到的問題；並且要感謝CBN的主管總是信任我能做得到，還有CBN的同事，賢彬和彥瑋，在我需要幫助的時候從不吝於付出。還要感謝我的好朋友，婷，郁蕙和涴棋，總是適時給予問候與關懷，一起分擔我的壓力，感謝你們。

最後，謹以此篇論文獻給我的母親，您的鼓勵是我奮發向上的最大的動力，今後我也會更加努力。

李慈潔 謹誌

中華民國101年6月

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Network steganography today is a popular topic for network security. The aim of steganogrpahic communication for all of applications is that sender hides secret data (steganograms) in the cover letter and sends it to the receiver which is aware of this hiding procedure. What distinguishes traditional steganographic methods from modern ones is only the form of the cover letter (carrier) for secret data. Traditional applications used human skin, wax tables or letters etc., and nowadays applications are nothing more than digital media like pictures, audio, video which are transmitted using networks.

Stegnography studies the scheme to transmit secret to a recipient such that no third party can detect the existence of the secret. There are several steganographic schemes proposed in [2], [3], [5], [8].

Chakinala et al[1] discussed three possible communication modes: the k-distance permuter, the k-buffer permuter and the k-stack permuter, and proposed efficient embedding and extraction algorithms for hiding information. Next Tapiador et al.[9] analyzed the distribution of distance between normal permutation and those with hidden messages based on the model of k-buffer permuter and showed that these permutations can be distinguished. Furthermore they showed that an adversary can obtain the hidden messages without the information of secret key k by observing the permutation encoding by the embedding algorithm mentioned in [1]. We focus on this issue and use two algorithms to protect against such attacks. In this thesis, we apply the idea of permutation arrays[6] [4] for steganographic communication. We enhance the deterministic embedding algorithm in [1] by using randomness on producing permutations to hide messages. Because of the randomness, it becomes harder for an adversary to identify the hidden messages from the random permutations. To prevent a passive adversary from decoding the permutations, we also strengthen a k-buffer permuter with randomness.

Recent studies in steganography utilizes the network protocols as a steganogram carrier by modifying content of the packets. There are very detailed surveys about Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) protocols on steganographic communication by Zander et al. [12] and Petitcolas et al. [13]. All of the information hiding methods that may be used to exchange steganograms in telecommunication networks is described in terms of network steganography which was originally introduced by Szczypiorski in 2003 [10]. Chakinala et al[1] proposed a formal model for transmitting information by packet-reordering. The scheme introduced in the model is to rearrange the order of packets for hiding secret data in ordered channels, such as TCP, which is linearly ordered, and to make use of the sequence number in TCP packet format, which is used to recover the order of packets. Lucena et al. [11] presented a hiding method in IPv6 protocol header.

Stream Control Transmission Protocol (SCTP) protocol is a new IP transport protocol existing at the same level as UDP and TCP and similar to them. It was published as RFC2960[14] by IETF in 2000. The primary purpose of SCTP protocol is to provide a reliable end-to-end message transportation service over IP layer. Wojciech et al. [12] brought up several concepts of steganographic methods by hiding covert information within some fields of the packet format for SCTP protocol.

In this thesis, we implement a system called STEC with SCTP protocol to create the communication channel and the transmission sequence number field within SCTP packet as the sequence number within TCP packet to hide secret data. In the STEC system, there are two sides, server and client, to communicate in SCTP channel and the secret is sent by server to client under cover letter.

The rest of this thesis is organized as follows. The Chapter 2 introduces three algorithms from permutation arrays for the steganographic communication, i.e., mapping algorithm, embedding algorithm, and extraction algorithm. In Chapter 3, we outline the software architecture of the STEC system with the algorithms described in Chapter 2 and explain the operations between server and client. In this thesis, we formalize two models according to the type of cover letters: multimedia model and file model. The former is used to transfer a file as secret and the latter is used to transfer a string as secret. Chapter 3 explains the software architecture of STEC system. Chapter 4 discusses the experimental technique and performance analysis of the STEC system. Finally, in Chapter 5, we conclude with some applications possible for the implemented system and future work.

Chapter 2

Preliminary

2.1 Background of the Steganographic Communication system

In STEC system, we need an ordered channel to permutate the sequence of sending packets. A common protocol of an ordered channel is the TCP (Transmission Control Protocol). TCP protocol[15] has been around for quite some time and it has provided a protocol to move data from one point to another in computer networks. Despite of the sequence number field used to order the packets, TCP has many limitations.

TCP provides both reliable transfer and strict order-of-transmission delivery of data. However the strict sequence maintenance in TCP not only makes the service of partial ordering of data impossible but also potentially results in unnecessary delay to the overall data delivery. Loss of a single TCP segment may block delivery of all subsequent data in TCP stream until the lost TCP segment is delivered. In order to resolve the above limitations of TCP, the Signaling Transport (SIGTRAN) working group in the IETF developed Stream Control Transmission Protocol (SCTP). SCTP is a new IP transport protocol, existing at an equivalent level with UDP (User Datagram Protocol) and TCP.

One of the differences between SCTP and TCP is the startup stage. SCTP uses a four-way handshake whereas TCP uses a three-way handshake. SCTP uses its four-way handshake with a cookie to protect against a blind SYN attack. Although some more modern TCP im-

plementations also use a cookie mechanism, it is not a standard mechanism but an option. On the other hand, SCTP requires that a cookie mechanism be used.
At the beginning, the development of SCTP was motivated by transportation of the Public Switched Telephone (PSTN) signaling message. But the design is also a good match for other applications with real-time requirements. Some researches have been done on developing applications using SCTP to improve the performance of multimedia streaming delivery[19] [20].

Since the requirements of our encoding schemes are reliable transmission without sequence maintenance, we consider that SCTP provides flexibility to suite each situation with optional ordered delivery. Therefore, we make use of SCTP protocol as the novel covert channel.

2.1.1 Overview of SCTP protocol

SCTP packets are made up with an SCTP common header and specific building blocks called chunks. The SCTP common header provides SCTP with verification and the properties of an association (a connection between two SCTP endpoints is called an association). Chunks provide SCTP with the basic structure used to carry information and are classified into two types: control and data. A control chunk carries information that is used to control and maintain the SCTP association. Data chunks are used to convey user messages through the association. Within a SCTP packet, one SCTP common header can be followed by one or more chunks as illustrated in Figure 2.1.

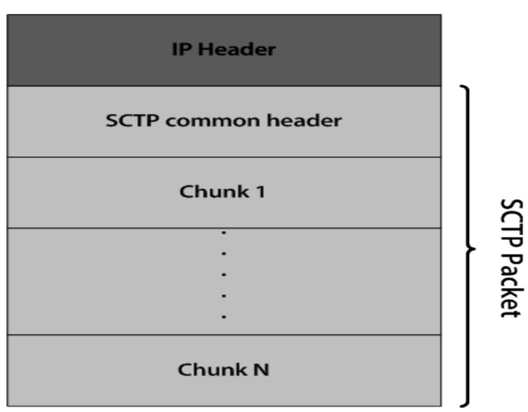

Figure 2.1: SCTP Packet Format

The SCTP common header: The SCTP common header described in Figure 2.2 contains four fields, source port number, destination port number, verification tag and checksum, and these fields perform three basic services.

1. To confirm an SCTP packet with an association: the source and destination port numbers carried in the SCTP packet's common header, and the source and destination IP addresses carried in the IP header, can uniquely identify the SCTP association.

2. To verify the SCTP packet belonged to the specific association: verification tag prevents an SCTP packet belonging to a previous association from being mistaken for an SCTP packet belonging to the current association, where these SCTP associations are created by the same pair of SCTP hosts.

3. To check the data integrity on the transport level to ensure the correctness of the SCTP packet: the checksum covers the SCTP common header and all chunks in the SCTP packet.

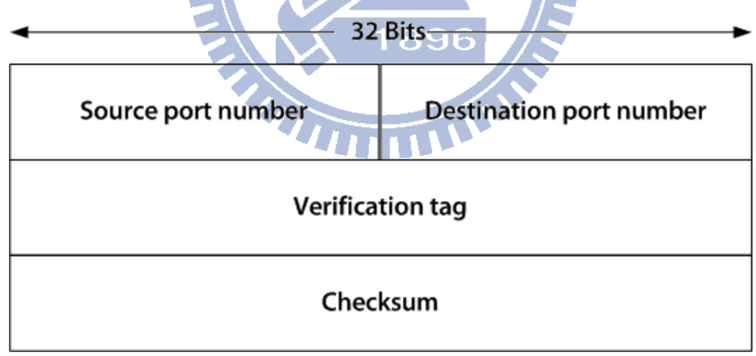

Figure 2.2: The SCTP common header

Elements in a chunk: The layout of a chunk is displayed in Figure 2.3. The chunk type field is an 8-bit value and represents the type of chunk. Hence the format allows a total of 256 distinct chunk types to be defined. Currently there are 16 chunk types defined in base SCTP protocol and the remaining 240 chunk types may be defined in the future.

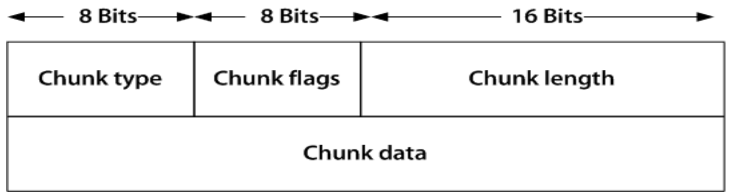

Figure 2.3: The layout of a chunk

Now we only list 8 chunk types and an extension chunk type for partial reliability feature in Table 2.1.

Table 2.1: Chunk type defined by chunk name

Chunk type	**Chunk Name**	**Description**
0x00	DATA	This chunk carries a user data payload.
0x01	INITIATION (INIT)	This chunk is used to start an SCTP association.
0x02	INITIATION ACKNOWLEDGE (INIT-ACK)	This is used as the response to an INITIATION chunk.
0x03	SELECTIVE DATA ACKNOWLEDGE (SACK)	This chunk is used to acknowledge the reception of data by an SCTP receiver.
0x07	SHUTDOWN	Used to initiate a termination.
0x08	SHUTDOWN ACKNOWLEDGE (SHUTDOWN-ACK)	A response to a SHUTDOWN.
0x0a	COOKIE ECHO (COOKIE-ECHO)	This is used to pass the state cookie.
0x0b	COOKIE ACKNOWLEDGE	Used to response a COOKIE ECHO.
0xc0	FORWARD CUMULATIVE TSN (FORWARD-TSN)	Used to inform the data receiver to adjust its cumulative received TSN.

Data chunks carrying fragments of user messages will be reassembled by the receiver, where the receiver uses the transmission sequence number (TSN) included in each data chunk to order and reconstruct the original user message. We pick up the most important parts of fields within DATA chunk for this thesis to explain briefly as follows.

Figure 2.4 illustrates the packet format of a data chunk.

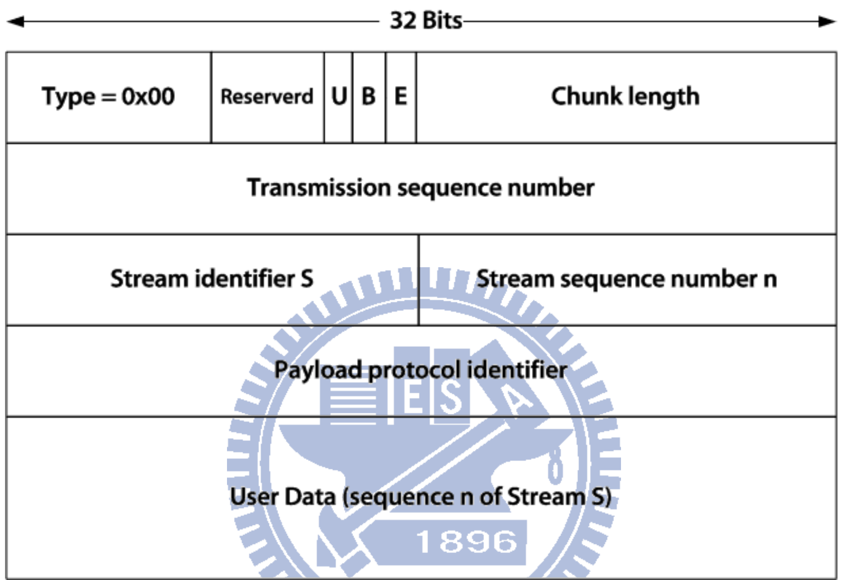

Figure 2.4: The DATA chunk

1. Stream Identifier (SID): Every data chunk must carry a valid stream identifier.

2. Stream Sequence Number (SSN): The SSN is a 16-bit sequence number to assure sequenced delivery of the user messages within a given stream and orders this data chunk within the stream.

3. Transmission Sequence Number (TSN): The TSN is a 32-bit sequence number attached to each chunk containing user data to permit the receiving SCTP hosts to acknowledge its receipt and detect duplicate deliveries. This field is similar to the sequence number used internally by TCP. The TSN is used to order, reassembly, retransmission.

4. The U, B, and E bits: The U, B, and E bits have the following definitions:

(a) U bit – SCTP protocol supports in-sequence delivery (like TCP) and order-of-arrival delivery (like UDP) dependent on the value of U bit.

Within a stream, a host must deliver data chunks received with the U bit set to 0 to the upper layer according to the order of their stream sequence number. If data chunks arrive out of the order of their stream sequence numbers, the host must hold the received data chunks until they are re-ordered.

TCP requires a strict order-of-transmission delivery service for all data passed between two hosts.

However, an SCTP host can indicate that no ordered delivery is required for a particular data chunk transmitted within the stream by setting the U bit of the data chunk to 1.

The SSN field in a data chunk with U bit set to 1 is insignificant. In our system, we reorder the sequence of packets to hide secret and thus we use this feature which is order-of-arrival delivery.

(b) B bit – For fragmented user message, if this bit is set to 1, it indicates the first part in a sequence of data chunks.

(c) E bit – This bit is opposite of B bit. If this bit is set to 1, it indicates the last part in a sequence of data chunks. When an entire user message can be carried in a single data chunk, both the B and E bits will be set to 1.

In this chapter, we will introduce multi-streaming feature and the extension feature of SCTP protocol, that is partial reliability service.

Multi-Streaming feature: This feature allows data to be partitioned into multiple streams and each stream delivery is independent so that any message loss only affects delivery within the stream.

In contrast, TCP assumes a single stream of data and ensures that delivery of that stream is byte sequence preservation. While this is desirable for delivery of data, it causes additional delay when message loss or sequence error occurs within the network. And then TCP must delay delivery of data until retransmission of a lost message or correct sequencing is restored. For our encoding scheme, the property of strict sequence preservation is not truly necessary.

SCTP accomplishes multi-streaming by creating independence between data transmission and data delivery. In particular, each payload of a data chunk in the protocol uses two sets of sequence numbers, a Transmission Sequence Number that governs the transmission of messages and the detection of message loss, and the Stream ID/Stream Sequence Number pair, which is used to determine the sequence of delivery of received data.

This independence of mechanisms allows the receiver to determine immediately when a gap in the transmission sequence occurs (e.g., due to message loss), and also whether or not messages received following the gap are within an affected stream. If a message is received within the affected stream, there will be a corresponding gap in the Stream Sequence Number, while messages from other streams will not show a gap. The receiver can therefore continue to deliver messages to the unaffected streams while buffering messages in the affected stream until retransmission occurs.

Partial Reliability service: This feature is an extension to SCTP protocol and introduced in RFC 3758[16]. The protocol extension consists of two new elements. One is used in INIT/INIT-ACK exchange to indicate whether the sender and the receiver both support this extension and the other is a new chunk type, FORWARD TSN, that indicates that the receiver should update its cumulative tsn. For example shown in Figure 2.5, when the receiver detects that tsn 4 and 5 are missing, it will send out sack containing tsn gap information immediately. As long as the sender keeps receiving sack including tsn gap, it will treat these missing data chunks as being acked and send out FORWARD TSN SCTP packet to inform the receiver to move its cumulative tsn to TSN 6.

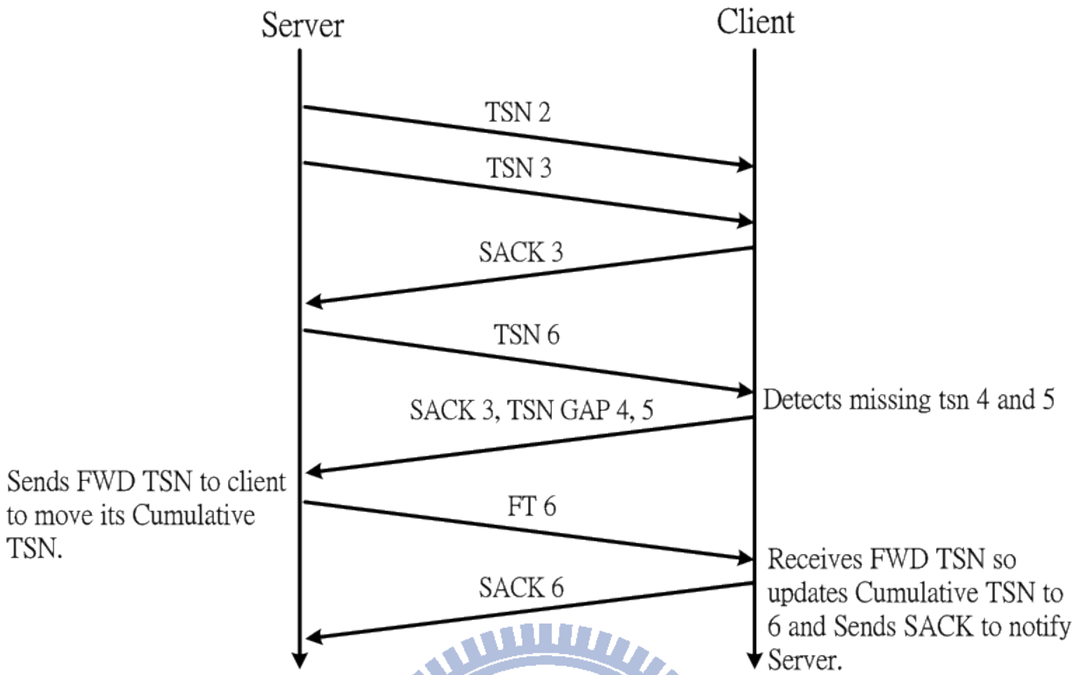

Figure 2.5: Example of Parital Reliability feature

2.1.2 Java based tool

Jpcap[16] is a Java class package that allows Java applications to capture and/or send packets to the network and is based on libpcap and WinPcap. Libpcap and WinPcap are system-independent interfaces for user-level packet capture and they provide a set of functions independent of the hardware and the operating system. Following the completion of the Pcap (packet capture) interface, that is libpcap/WinPcap, jpcap was able to successfully capture packets from a local network interface.

Raw Socket API allows an application to directly access lower-level communication protocols to bypass the TCP/IP Stack and have access to build the entire packet including Ethernet and IP headers. Therefore, Raw Socket API offers programmers the freedom to create the format of packets which are being sent or received through the network interface and this is the reason why we use jpcap library to develop the STEC system.

By creating a raw sockets based on SCTP protocol and binding to network interface card (NIC) driver, any SCTP packets will be sent/received through the bound interface. We also design the graphic user interface (GUI) for the STEC system by JAVA programming language so that users can operate the system conveniently.

As our developing platform can be under Windows (32-bits) and Linux operating environment, our works deal with the Windows/Linux packets available in the network. Figure 2.6 shows the library needed by jpcap on Windows and Linux operating system respectively.

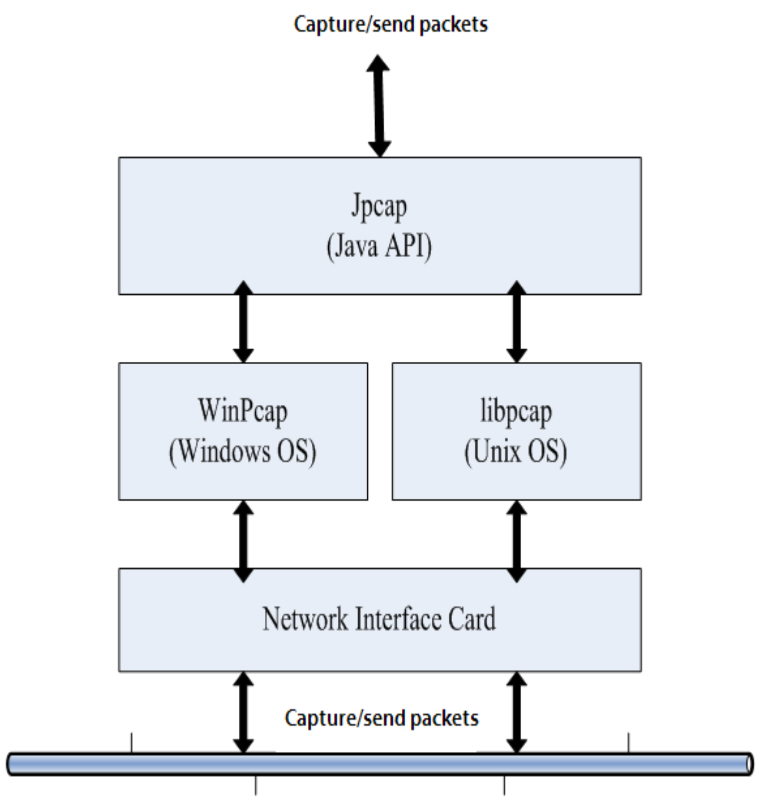

Figure 2.6: The jpcap process model

In STEC system, one of models is to play audio at client and the format of audio file is Waveform audio format. Waveform audio format (WAV) is a Microsoft/IBM audio file and is used to store uncompressed, lossless audio. We worry about packet loss within the network and hence we use the WAV audio files even if uncompressed WAV files are quite large in size. The java class we use is AudioSystem class acts as the entry point to the audio system resources. It contains a number of methods for converting audio data between different formats, and for translating between audio files and streams including WAV audio files.

2.2 Algorithms

In this section, we introduce the steganographic communication and the permutation arrays.

2.2.1 Definitions

First we define the k-buffer permuters.

Definition 1. *A k-buffer permuter uses a random access buffer of size k. There are two operations that a k-buffer permuter can perform.*
1) **put :** *The k-buffer permuter removes one element from the input stream and places it in the buffer.*
This operation can be performed when the buffer is not full.
2) **remove :** *The permuter removes one element from the buffer and places it in the output stream.*
This operation can be performed when the buffer is not empty.

Permutations produced based on a k-buffer are called k-buffer permutations. Note that 1-buffer permutation is the identity permutation.

Denote S_n as the set of all permutations of length n. An (n,d) permutation array is a subset of S_n with the property that the distance between any two permutations in the array is at least d. Lin et al. [6], [4] have presented an encoding and decoding schemes for permutation arrays where the distance metric is l_∞-norm. That is, given π and $\sigma \in S_n$, the distance $d_\infty(\pi,\sigma) = max_i|\pi_i - \sigma_i|$, where $\pi = \{\pi_1, ..., \pi_n\}$ and $\sigma = \{\sigma_1, ..., \sigma_n\}$. In this thesis, we define a specific permutation array for k-buffer permutations.

Definition 2. *An (n,d) permutation array for k-buffer permutations, denoted as $PA_k(n,d)$, is a subset of k-buffer permutations such that the distance (l_∞-norm) between any two permutations in $PA_k(n,d)$ is at least d.*

2.2.2 Mapping, embedding and extraction algorithms

Let d be the minimum distance between any two permutations in $PA_k(n, d)$. Algorithm 1 is to find the proper permutation from the k-buffer permutations for a given binary message.

Algorithm 1 Mapping Algorithm from Binary Message to k-buffer permutations.

Input: a message $\{x_1, ..., x_{n-d}\} \in Z_2^{n-d}$
Output: a k-buffer permutation $\pi = \{\pi_1, ..., \pi_n\}$

Fill the k-buffer with $\{1, ..., k\}$; $S = \{1, 2, ..., n\} \backslash \{1, ..., k\}$;
let max be the largest number in the k-buffer, min be the smallest number in the k-buffer, m be the smallest number in S;
for $i = 1$ to $n - d$ **do**
  if $x_i == 1$ **then**
    $\pi_i = max$;
    remove max from k-buffer;
  else
    $\pi_i = min$;
    remove min from k-buffer;
  end if
  Output π_i;
  Add m to k -buffer; remove m from S;
  Update max, and min, and m;
end for
for $i = n - d + 1$ to n **do**
  $\pi_i = min$;
  Output π_i;
  Add m to k -buffer; remove m from S;
  Update max, and min, and m;
end for

We assume that the elements in the k-buffer are sorted in increasing order. According to Algorithm 1, we have the following theorem.

Theorem 1. *Algorithm 1 generates a $PA_k(n, d)$, where d = k-1 and the length of the binary message is n-k+1.*

Proof:

For any two permutations π and σ generated by Algorithm 1 where $\pi = \{\pi_1, ..., \pi_n\}$ and $\sigma = \{\sigma_1, ..., \sigma_n\}$. Let i be the first position such that $\pi_i \neq \sigma_i$. Since π_i does not equal σ_i, by the configuration of Algorithm 1, the one of π_i and σ_i is *max* and the other is *min*. Because there are k numbers in the k-buffer, $|\pi_i - \sigma_i| > k - 1$. Hence the l_∞-norm distance between π and σ is at least $k - 1$. By [6] and [4], the length of the binary message is $n - d$ and because of $d = k - 1$. Therefore the length of the binary message is $n - k + 1$. ■

Since the value of k is the secret key for the sender and the receiver, passive adversaries can not know the information of the set $PA_k(n, d)$ and the length of messages. The passive adversaries can not obtain the permutations in the permutation array without the secret key k even though the knowledge of encoding algorithm for the permutation array is understood.

According to Algorithm 1, we add randomness on selecting a permutation to design Algorithm 2. Within Algorithm 2, randomly swap l pairs in the k-buffer permutation where $2l$ elements selected are all distinct and the difference between each pair is at most $\frac{k-1}{2}$.

Algorithm 2 Embedding Algorithm.

Input: a message $\{x_1, ..., x_{n-d}\} \in Z_2^{n-d}$
Output: a k-buffer permutation $\pi = \{\pi_1, ..., \pi_n\}$

Randomly select l pairs such that for each pair (a, b), $|a - b| < \lfloor \frac{k-1}{2} \rfloor$ where $a, b \in \{1, ..., n\}$, $l < \lfloor \frac{n}{2} \rfloor$ and all a's and b's are distinct.
Run Algorithm 1 to obtain $\pi = \{\pi_1, ..., \pi_n\}$
for $i = 1$ to n **do**
  for each pair (a,b) **do**
    if $\pi_i == a$ **then**
      $\pi_i = b$;
    end if
    if $\pi_i == b$ **then**
      $\pi_i = a$;
    end if
  end for
  Output π_i;
end for

Our extracting algorithm is actually the decoding algorithm for $PA_k(n, k-1)$ which is described in Algorithm 3.

Algorithm 3 Extraction Algorithm.

Input: a k-buffer permutation $\pi' = \{\pi'_1, ..., \pi'_n\}$
Output: a binary message $\{\widehat{x}_1, ..., \widehat{x}_{n-d}\} \in Z_2^{n-d}$

Fill the k-buffer with $\{1, ..., k\}$; $S = \{1, 2, ..., n\} \backslash \{1, ..., k\}$;
let max be the largest number in the k-buffer, min be the smallest number in the k-buffer, m be the smallest number in S;
for $i = 1$ to $n - d$ **do**
  if $|max - \pi_i| < |\pi_i - min|$ **then**
    $\widehat{x}_i = 1$;
    remove max from k-buffer;
  else
    $\widehat{x}_i = 0$;
    remove min from k-buffer;
  end if
  Output π_i;
  Add m to k -buffer; remove m from S;
  Update max, and min, and m;
end for

Chapter 3

System Construction

Our system can handle two kinds of secret message: one is a string and the other is a file. If we want to send a string as a secret, we make use of a file as the cover letter. If we want to send a file as a secret, we use media streams as the cover letter. When sending a file within another file, it needs a file whose size is sufficiently large to provide enough number of packets to apply our mapping algorithm described in Chapter 2. For instance, if the size of the file that we consider as a secret is just 1024 bytes (8192 bits), then we need at least 8192 packets in the mapping algorithm. If the size of secret data is 512 bytes, in fact, the requirement of a file larger than 4MB is necessary. However aside from the 8192 bits, we also require additional bits to be untapped bits for our system. Therefore, we choose multimedia to be the cover letter when the secret is a file. It is an extra load for a server to select an appropriate file to carry the secret. If the type of cover letter is multimedia, we do not need to care about the size of cover letter and just keep playing the multimedia stream until the completion of the secret transferring.

The system hides the secret by permutating the order of data packets and it does not need additional bandwidth to reach this goal. The receiver also won't be aware of receiving extra information.

This system consists mainly of two components, Server and Client. Figure 3.1 shows the basic operation functions between Server and Client. Next we will explain the functions briefly

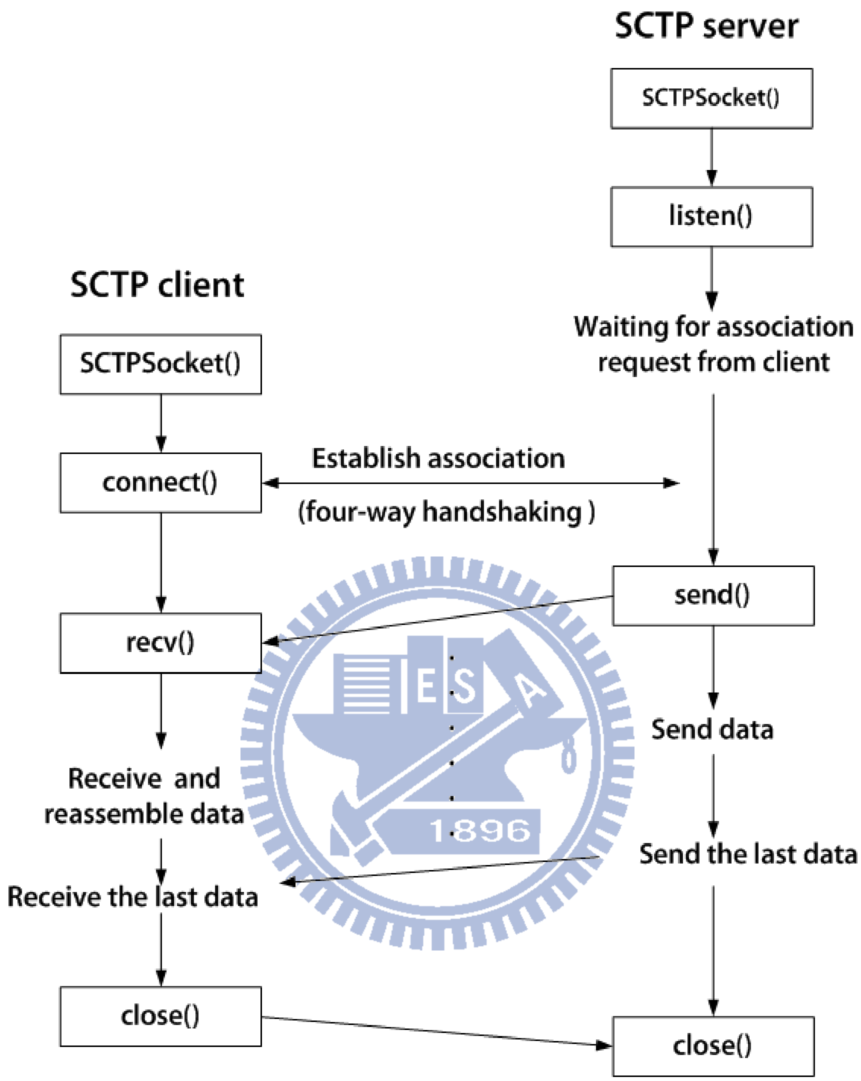

Figure 3.1: Basic operations between Server and Client

3.1 Software architecture of the system

Server Side Server is responsible for sending files or media streams. It transfers secret, a string or a file, to (0,1)-string and applies alogrithm1 and algorithm2 to produce a permutation according the (0,1)-string. Next divide the cover letter into several fragments and label the number for them, and then send these fragments in the order of permutation.

Table 3.1: The class of Server

Category	**Class**	**Description**
Main	ServerSCTPmain	The Server program entrance.
	MainFrame	This is the graphic user interface for selecting network interface and cover letter.
FileSend	SendFileFrame	This is the GUI for send file.
	SendFileThread	It is in charge of sending file data.
	SendFileInfo	It is used to compute the partition information of the sending file.
MeidaSend	SendMeidaFrame	This is the GUI for sending media stream data.
	SendMeidaThread	It is responsible for sending media stream data.
ProcessPackets	Recvfrom	It is used to monitor incoming SCTP packetsr.
SCTPPacket	SCTPPacket	It is used to establish SCTP packet.
Send	SendPacket	It is used to send SCTP packets.
Algorithms	MappingAlgorithm	It is mapping algorithm described in Chapter2.
	EmbeddingAlgorithm	It is embedding algorithm described in Chapter2.
Control	FlowControl	If receives SACK chunk packet from client, control the packet flow.

The flow of programs for Server are shown in Figure 3.2 to Figure 3.5. Gray blocks represent graphic user interface.

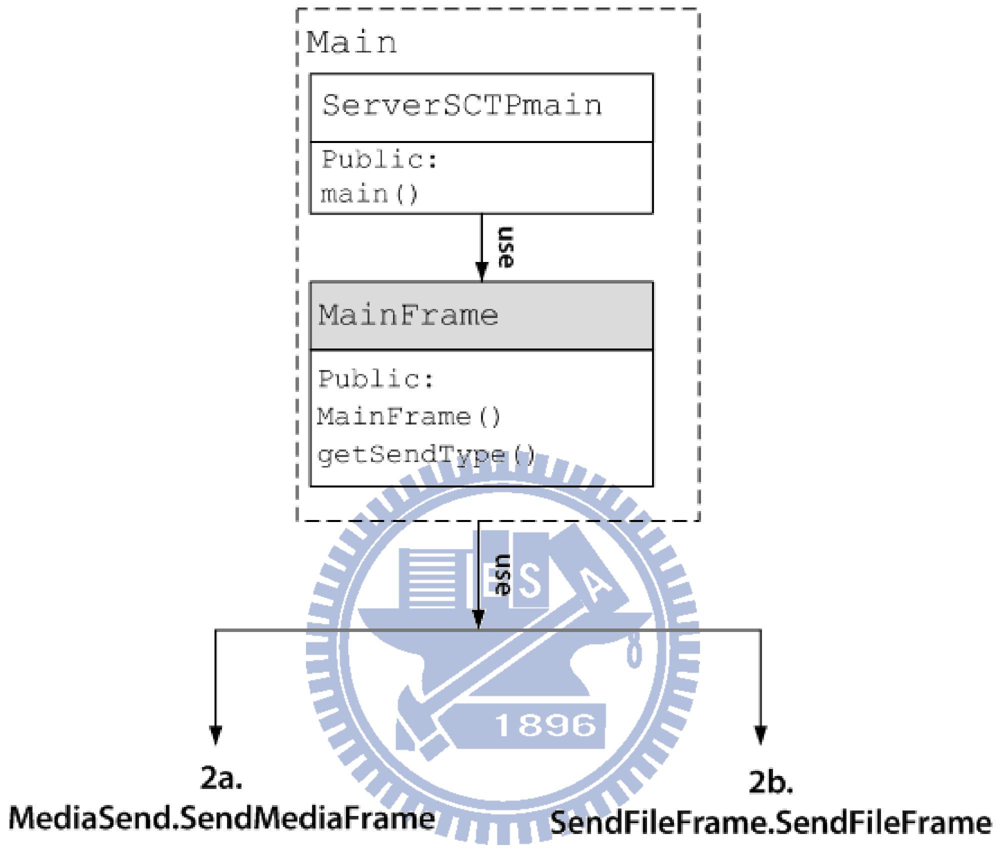

Figure 3.2: Part1: program flow diagram for Server

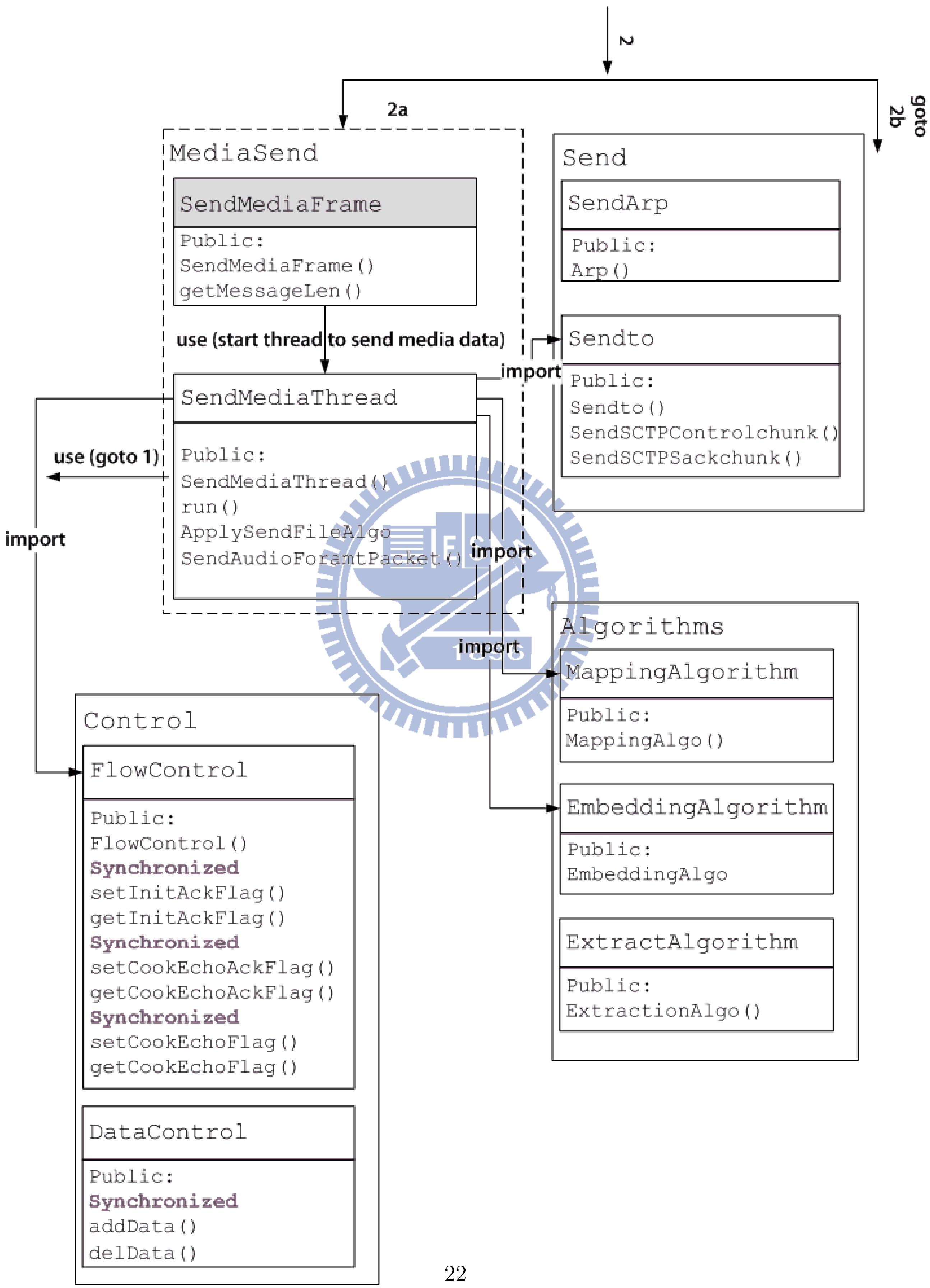

Figure 3.3: Part2: program flow diagram for Server

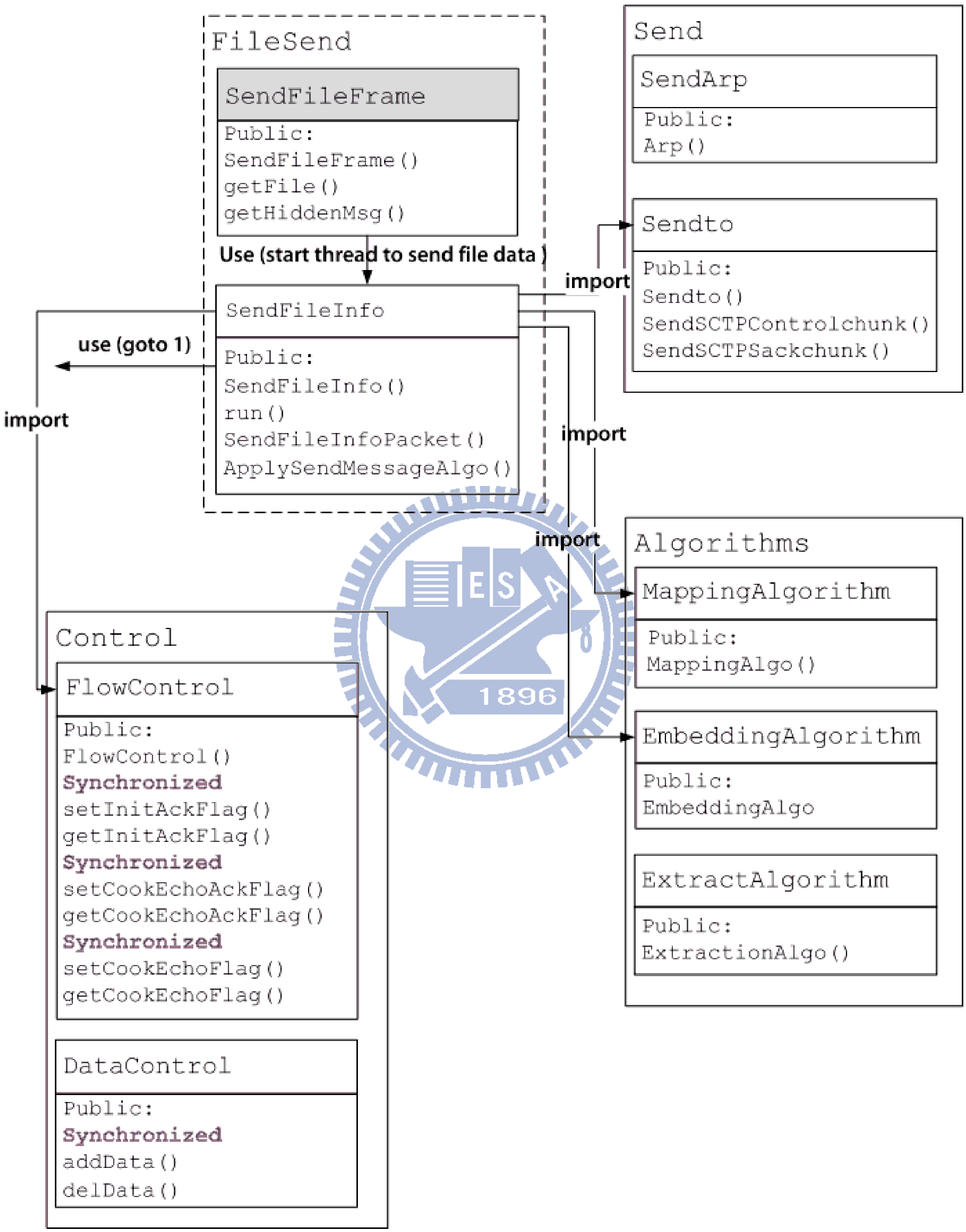

Figure 3.4: Part3: program flow diagram for Server

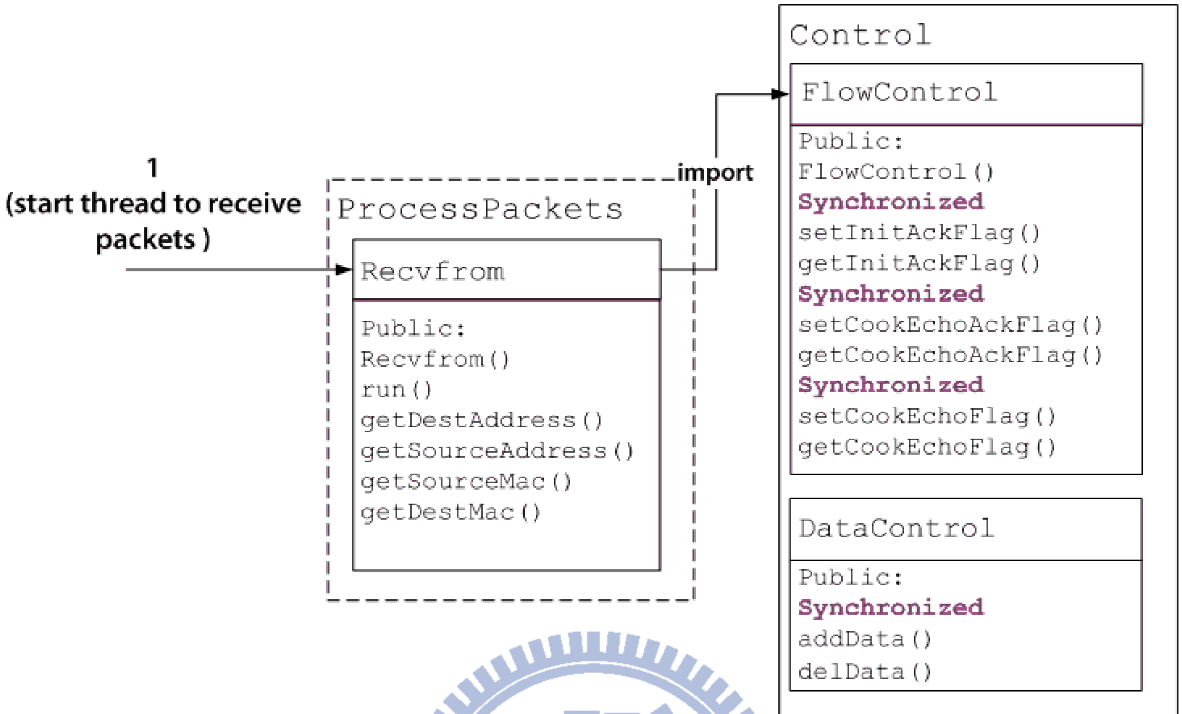

Figure 3.5: Part4: program flow diagram for Server

Next we will explain the basic operations of Server.

- Step 1: Select network interface card and the kind of cover letter you want to send. If you choose "Send Media Stream", you will go to Step2a.

Figure 3.6: Step 1 for Server (MainFrame)

- Step 2a: Fill the d variable which is needed by the algorithms. Select media files which are played out at Client and hidden file you want to send to Client. Finally push ”Start to listen” button. Information field will display where client connect.

Figure 3.7: Step 2a for Server (SendMediaFrame)

- Step 2b: Fill the d variable which is needed by the algorithms. Select a file and hidden message which you want to send to Client. Information field will display where client connect.

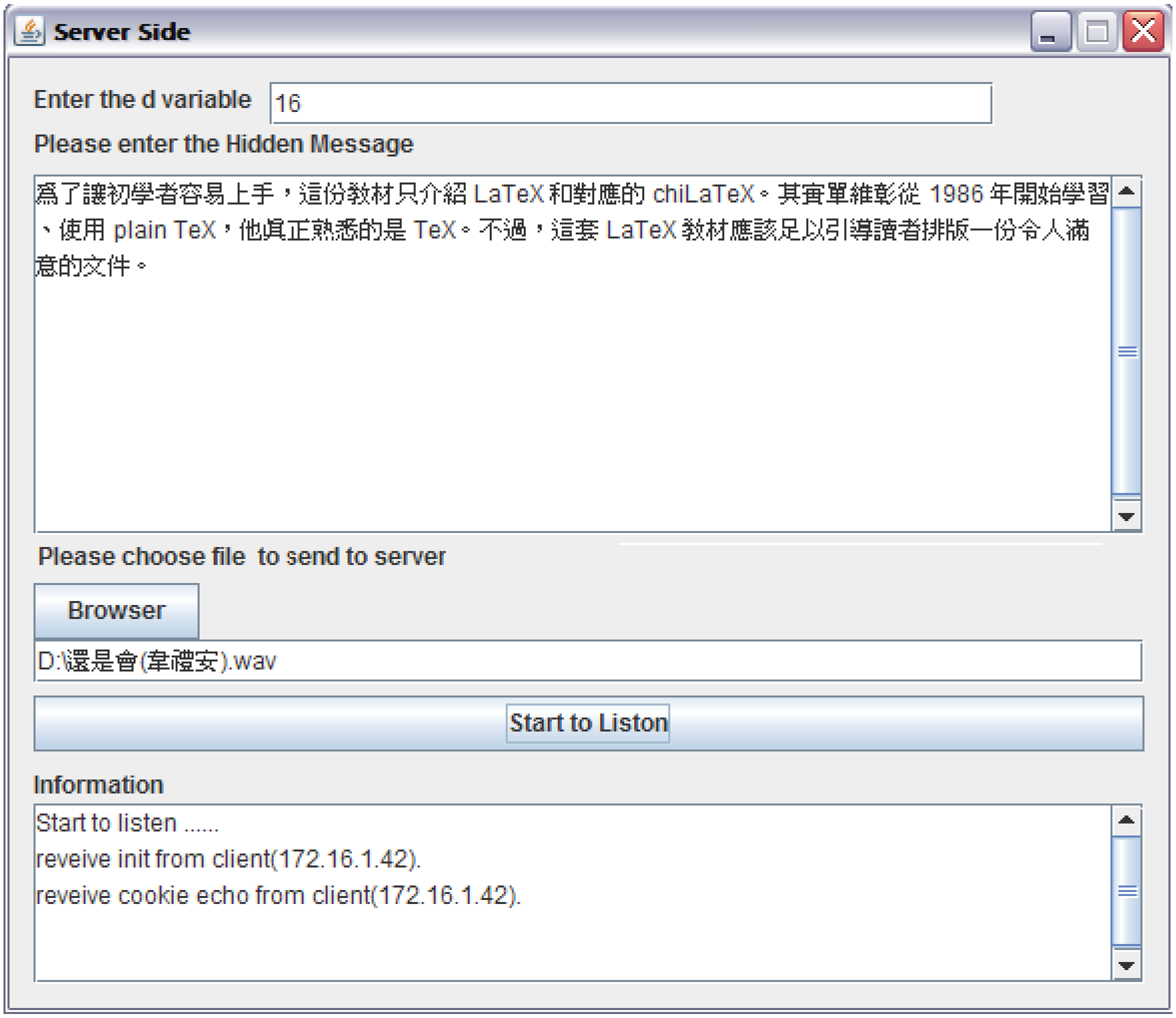

Figure 3.8: Step 2b for Server (SendFileFrame)

Client Side Client is responsible for receiving files or playing media streams. When Client receives packets, it will store the arriving packets in the corresponding position of the buffers according to the sequence number of packets. It will keep space for out-of-order packets in advance. By the time that buffer is full or no data will be received any more, Client will write these data into files or play the media streams.

Server will agree with Client on the length of a permutation. Hence whenever the number of packets received by Client is equal to the length of a permutation, Client applies algorithm3 to decode the permutation to produce (0,1)-string. Finally Client transfers (0,1)-string to a string or a files.

Category	**Class**	**Description**
Main	ClientSCTPmain	The Client program entrance.
	MainFrame	This frame is the graphic user interface for selecting network interface and filling server ip address.
FileReceive	RevFileFrame	This frame is to display the receiving sercet message.
	HandlePartitions	It is used to reassemble the receiving packets.
MediaPlay	RevMediaFrame	This frame is to display the playlist.
	HandleMediaFile	It is used to reassemble the receiving media stream packets.
	PlayAudioThread	This thread is responsible for play media stream data.
ProcessPackets	Recvfrom	It is used to monitor incoming SCTP packets to client.
SCTPPacket	SCTPPacket	It is used to set the fields of the IP and SCTP packet.
Send	Sendto	It is used to send SCTP packets.
Algorithms	ExtractionAlgorithm	Contains extraction algorithm described in Chapter 2.
Control	FlowControl	It is used to control the time to send SACK chunk packets to server.
	DataControl	It is used to control data buffer loading.

Table 3.2: The class of Client

The flow of programs for Client are shown in Figure 3.9 to Figure 3.12. Gray blocks indicate graphic user interface.

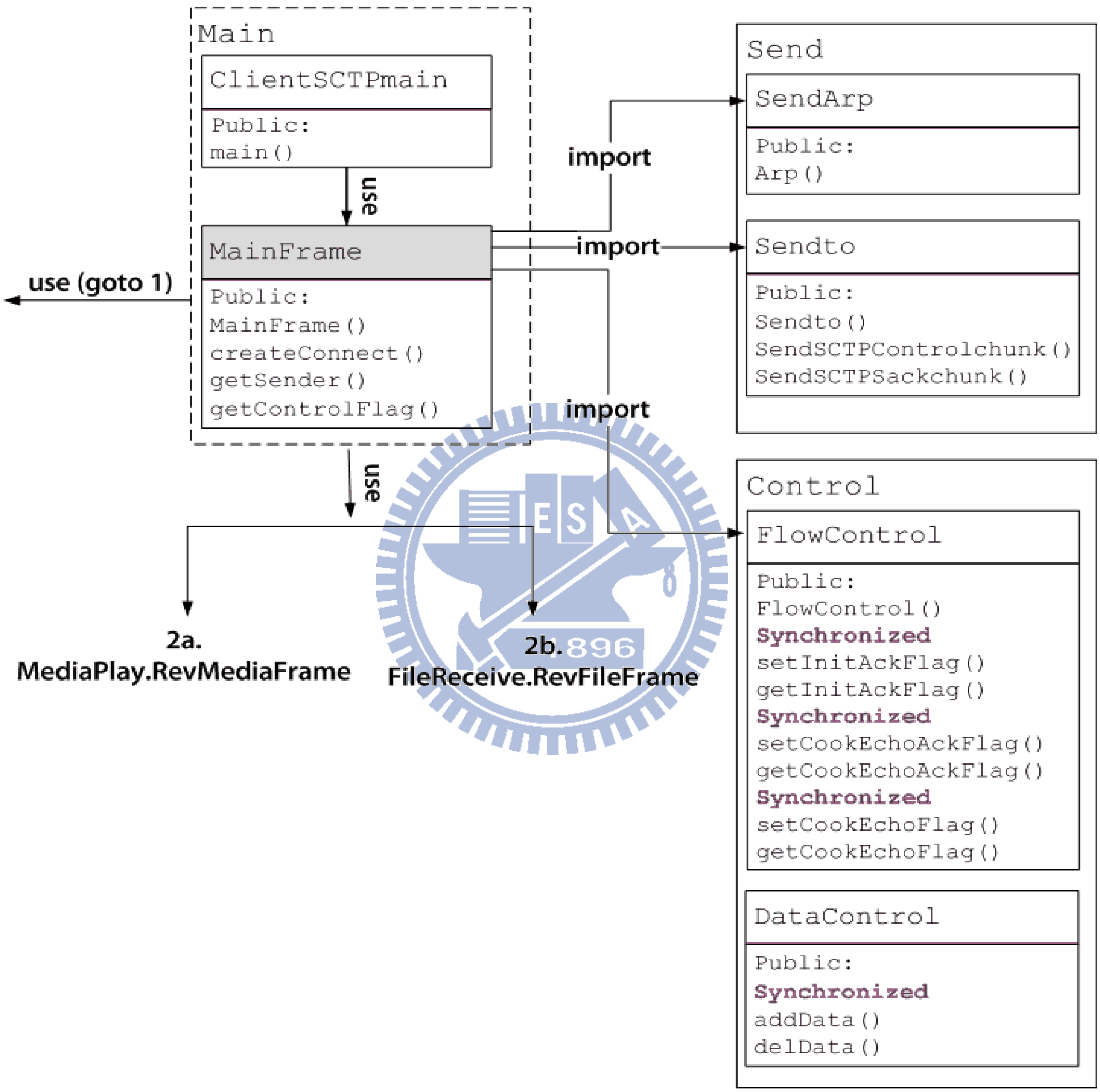

Figure 3.9: Part1: program flow diagram for Client

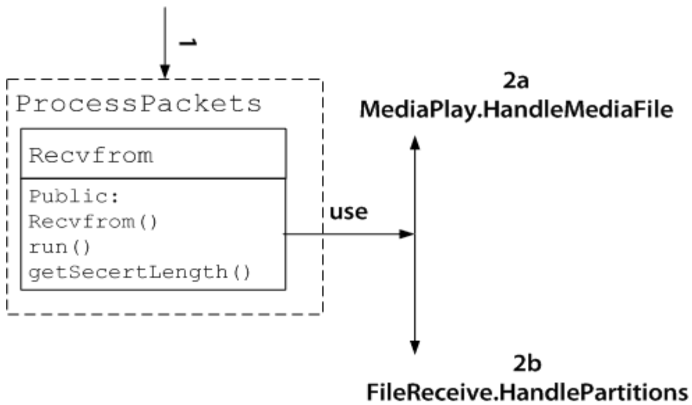

Figure 3.10: Part2: program flow diagram for Client

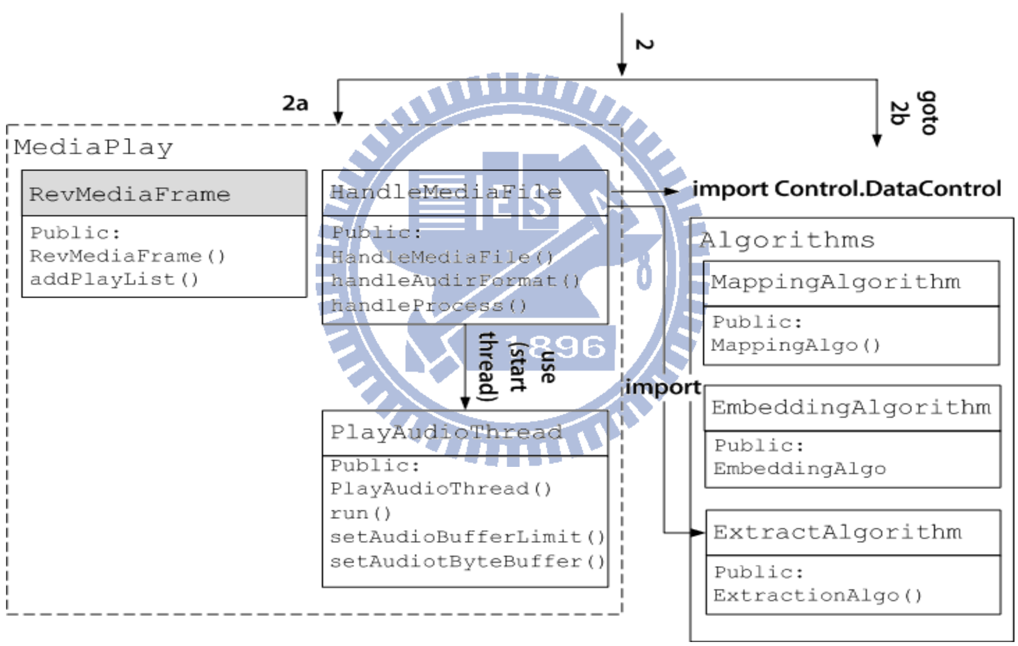

Figure 3.11: Part3: program flow diagram for Client

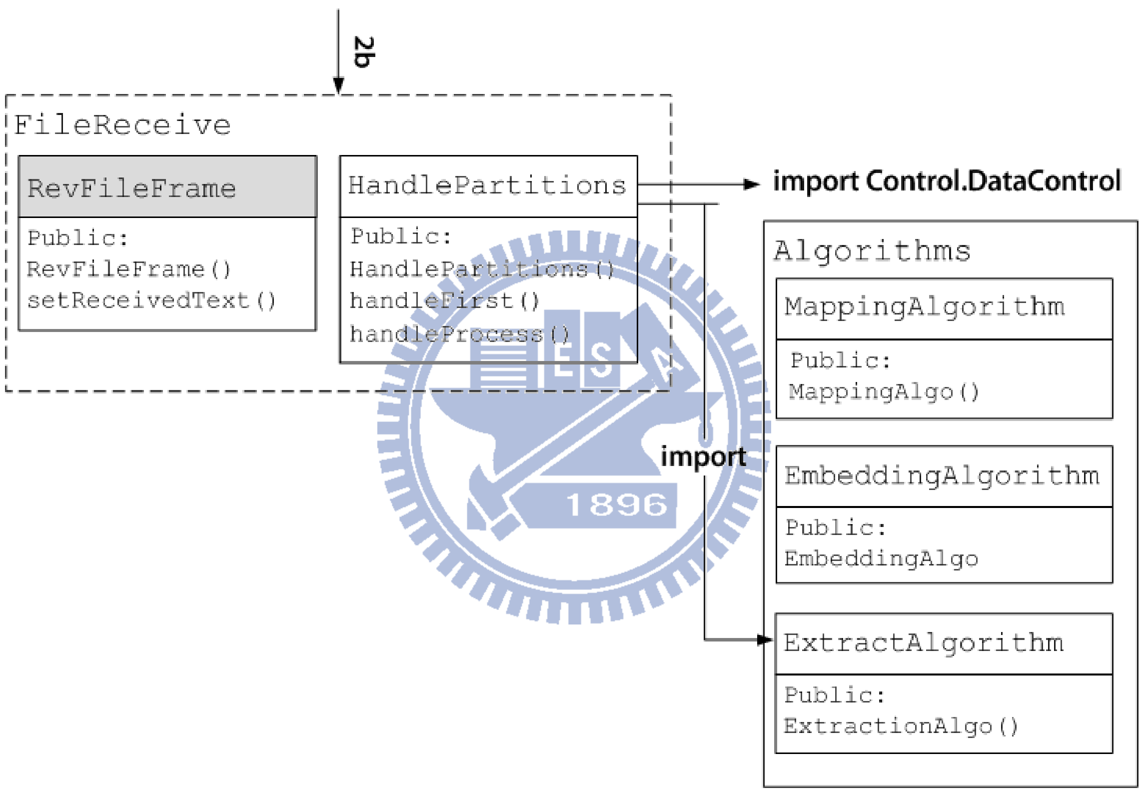

Figure 3.12: Part4: program flow diagram for Client

Next we will explain the procedures of Client.

- Step 1: Choose the network interface responsible for receiving and sending packets and fill the server ip address.

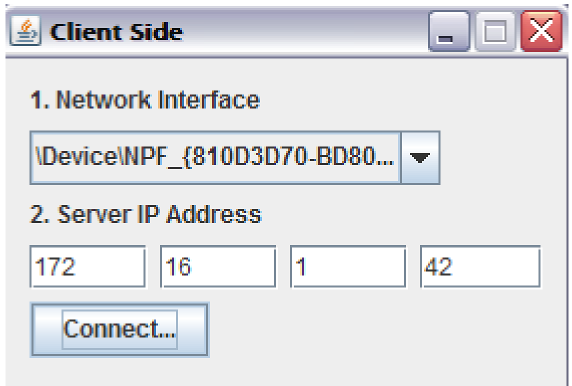

Figure 3.13: Step 1 for Client (MainFrame)

- Step 2a: If Server sends media stream as cover letter, then Client will go into Step2a. And frame will display the the name of media files to play.

Figure 3.14: Step 2a for Client (RevMediaFrame)

- Step 2b: If Server sends a file as the cover letter, then Client will go into Step2b. If the transfer is completed, then this frame will display the secret message and write the secret message into a text file named message.txt.

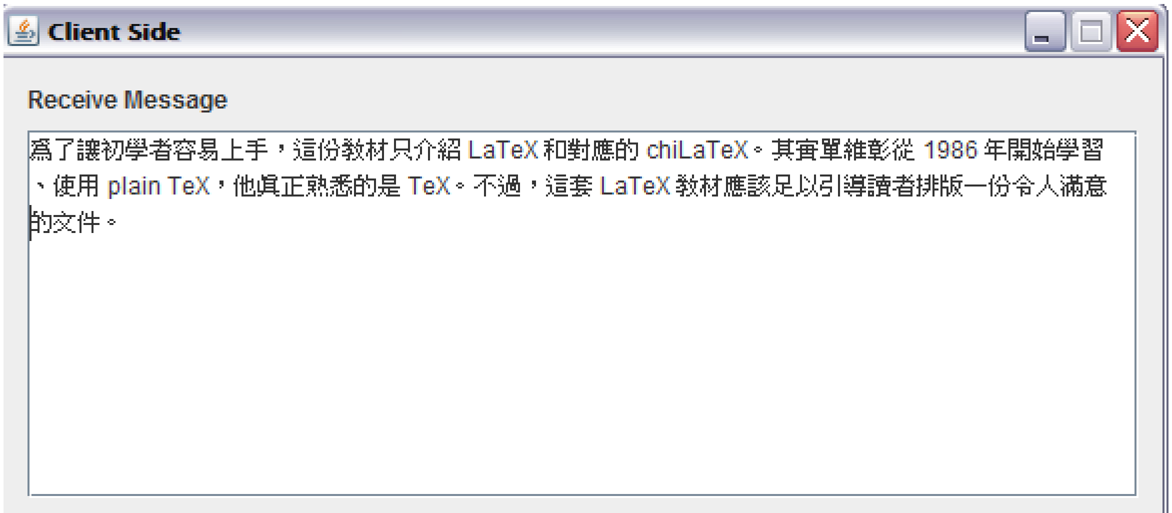

Figure 3.15: Step 2b for Client (RevFileFrame)

3.2 Common functions of the system

1. SCTPPacket:

   Because we use the raw socket to implement the system, we must construct SCTP packet by ourselves. Hence the SCTPPacket function is used to set IP header and all fields about SCTP packet format described in Chapter 2.

2. Send:

   The function is in charge of sending packets to Server (Client).

   After the four-way handshake between Client and Server, server will start to send SCTP data chunk packets.

   (a) Sendto: During four-way handshake process, Client will send SCTP control packet, init packet, to start a new association, cookie echo packet contains some information to make sure the association, finally send shutdown packet and shutdown complete packet to close the association.

Server will send acknowledge for INIT, COOKIE ECHO, SHUTDOWN SCTP packets to respond the Client. After establishing the association successfully, Server starts to send SCTP data chunk packets.

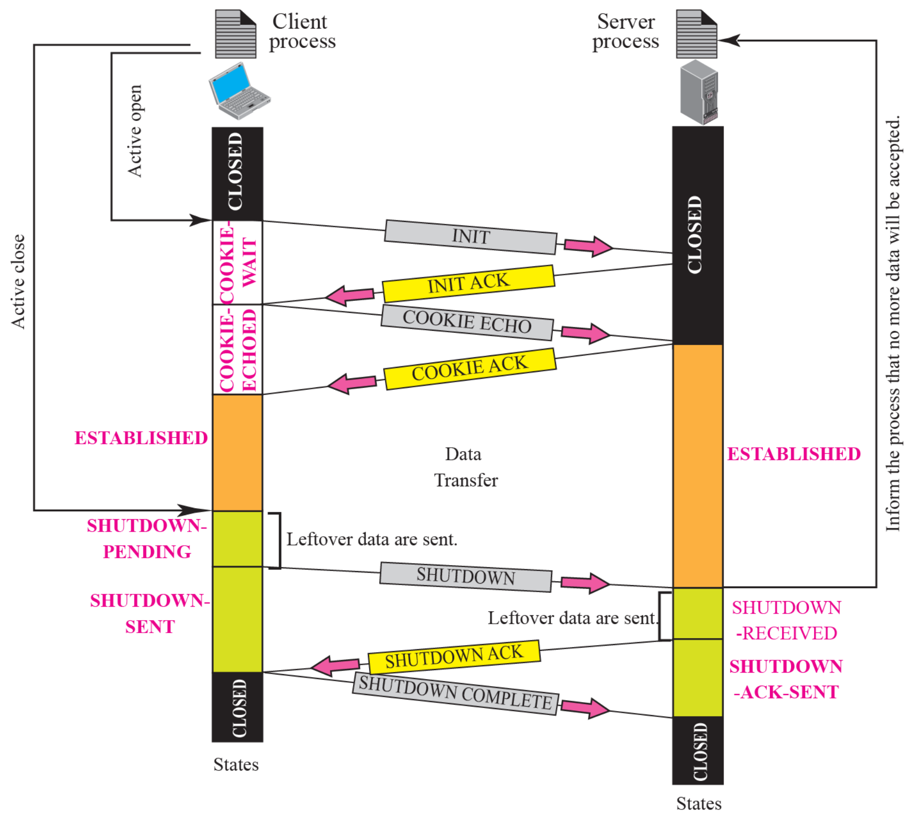

Figure 3.16: The packets between Server and Client

3. Algorithms:

   Server and Client have encoding and decoding functions respectively. Encoding function contains Mapping algorithm and Embedding algorithm and decoding function is Extraction algorithm. These algorithms are all described in Chapter 2 and are also the main idea of this thesis. Server applies encoding function to reorder the data packets and although Client receives out-of-order data packets, it can recover the order by decoding function.

4. Control:

   There are two types of Control, DataControl and FlowControl. DataControl is responsible for controling whether the buffer used to store SCTP data chunks is full or empty or not. If the buffer is full, then notify the Client to send SACK chunk to Server and if empty, notify the Client to wait for data chunks.

   FlowControl is responsible for receiving and sending SCTP Control chunks, i.e., INIT, INIT ack, COOKIE ECHO, COOKIE ECHO ack,etc.

Chapter 4

System Performance Analysis

4.1 Overview of implementation

The aim of the STEC system is to reorder the sequence of fragmentation when sending data. If we want to send user message whose size is greater than the current PMTU, the sender will break the message into a series of pieces such that each of them is small enough to be put into a SCTP data chunk packet and assign in sequence a distinct TSN to each of SCTP data chunk packets.

Before introducing the details of the STEC system, we give some definitions. We call the user data used to permute packets to hide secret stegano-chunks and the remainder data untapped-chunks in a representative group. The secret data is stored in stegano-chunks in a group. In order to transfer all stegano-chunks, we need a lot of such groups.

For example in Figure 4.1, we suppose the total number of secret data is 150 bits (we ignore the d variable for simplifying this case) and the length of stegano-chunks is 8, the length of untapped-chunks is 11. Hence we need 19 (the ceil of 150/8) groups to send out all secret data and the number of packets is at least 700 (there are 150 packets for stegano-chunks and 19*11 packets for untapped-chunks).

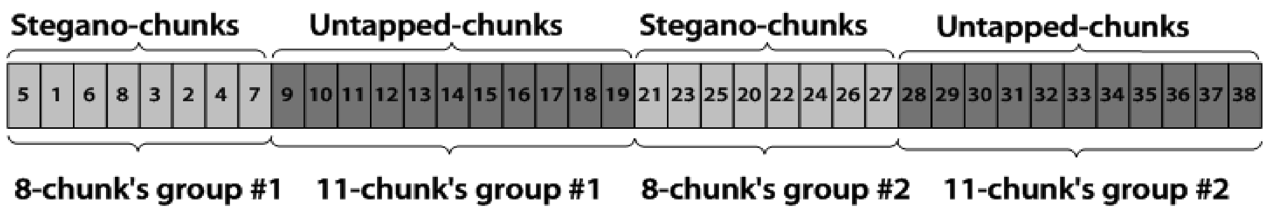

Figure 4.1: stegano-chunks and untapped-chunks

In order to enhance the security of STEC system, we randomly generate a (0,1) string for untapped-chunks. We also apply the encoding algorithms to the (0,1) string to permute these untapped-chunks. Hence adversary can not easily find the length of stegano-chunks.

In the STEC system, we formalize two models which are classified by the cover letters used to convey the stegano-bits: one is by the multimedia and the other is by the file.

Multimedia model In real time multimedia application, it is not necessary to care about the correctness of the sequence of data, but the arrival of packet carrying multimedia format is important. In other words, a flexible transport layer protocol offering a partial reliability service is ideal for multimedia transmission that only need to send partial-order data. Partially ordered delivery may provide less delay, and use fewer memory resources than ordered delivery. Except data packets carrying information about audio format, others do not wait for retransmission for missing. Therefore, we use the partial-reliability feature of SCTP protocol to transmit multimedia data.

Since we divide data into stegano-chunks and untapped-chunks, how to divide them will affect the transmission time. For the above example, we may think that if the number of stegano-chunks is small in a group, the number of transmitted packets carrying secret will be small in a period of time.

According to the specification of SCTP protocol, if the receiver detects the missing of one or more TSN after processing a newly received SCTP packet carrying data chunks, the receiver should immediately send out a SACK chunk with Gap Ack blocks to report the missing TSN(s) to sender. For the property of partial-reliability, the sender will send FORWARD TSN SCTP packets to update the cumulative tsn for the receiver without delay.

Therefore, during a transmission of multimedia, if there are a lot of out-of-order packets, it must generate additional load of SACK from the receiver and FORWARD TSN packets for partial reliability service from the sender. In case there are no extra packet loss in addition to the packet loss caused by stegano-chunks, the size of sent secret determines the number of SACK SCTP packets containing gap blocks and FORWARD TSN SCTP packets.

File model Sourabh et al[18] proposed a new concept of FTP over SCTP protocol and implemented it in three ways: (1) simply replacing TCP calls with SCTP calls, thus using one SCTP association for control and one SCTP association for each data transfer, (2) using a single multistreamed SCTP association for control and all data transfers, and (3) enhancing (2) with the addition of command pipelining. In the research, they gave the comparison of performance between over TCP and over SCTP and apart from the improvement of transferring time, they also showed three advantages achieved by running FTP over multistreamed SCTP. Therefore, in file mode, we apply the idea for transferring the cover letter, that is, a file.

We divide a file into at most three partitions and use a stream for transmitting a partition individually in STEC system.

4.2 Analysis of multimedia model

STEC is implemented and works reliably in the proposed models. The scheme is based on permutation encoding of the sequence of packets transferring in the network.

The multimedia data is real-time data and today's Internet multimedia applications use of UDP protocol. Despite out-of-order packets except the packet containing the format of audio, transmission will not be blocked. We use the first packet to convey the information about audio format and the first packet does not participate in the encoding process to ensure that multimedia can be played out normally by Client.

If there are a lot of out-of-order packets about multimedia data, it will affect the quality of playing. In order to provide better quality, we keep a buffer to maintain the sequence of packets. In our application, we set the size of a buffer for the number of received packets. For instance, if we set 128 packets as the size of a buffer and the size of a packet is 512 bytes, then the size of a buffer is 128*512 bytes. The buffer can be considered the congestion windows (cwnd) in a SCTP packet and the field of SCTP protocol is 16 bits, i.e., the maximum cwnd value is 65535 bytes.

Figure 4.2 is the example for the buffer design.

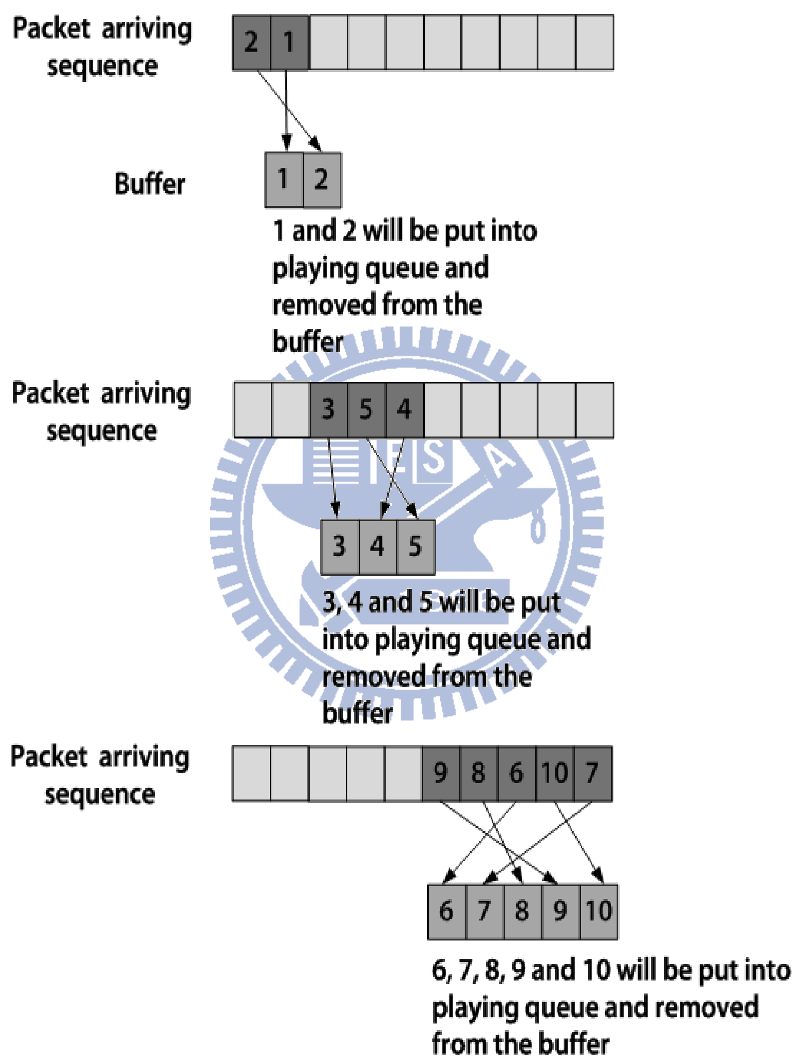

Figure 4.2: Example of data buffer for the multimedia model

Because multimedia is a kind of real-time data, client will put them into playing queue as soon as all sequence data arrive. For example shown in Figure 4.2, TSN 2 will wait for TSN 1 even if TSN 2 arrives first. Hence all out-of-order packets must wait for the forward packets. For this design, the quality of playing is not affected by these out-of-order packets because the distance between adjacent packets is at most k.

4.3 Analysis of file model

Like FTP over TCP, file model provides an application which is to transfer a file through SCTP data chunks. Because there will be a lot of out-of-order packets, our approach is to use a random access file which can be accessed in any order and hence we do not need a buffer to store out-of-order packets temporarily for reordering.

In order to improve the performance of transferring files, we use multiple threads in the file model. For multi-threads system, the running thread keeps running until it performs an operation that requires to wait. In file model, it needs to wait a delay time after sending data chunks for each thread. During waiting, the computer can swap the wating thread out of the processor. Thus the multi-threads are designed to maximize processor usage for file model.

We divide files into n partitions averagely and new a thread for a partition. Each thread is responsible for sending a partition and encoding a substring. We experimented with the the number of threads on a single-core processor and a dual-core processor.

In our experiment, the size of sending file is 47.2 MB and the length of (0,1)-string is 960 bits. The clock speed of single-core/dual-core processor is 1.70/2.53 GHz. The comparison between single-core processor and dual-core processor is shown in Table 4.1.

Table 4.1: Comparison between single-core processor and dual-core processor

Number of threads	Single-core processor	Dual-core processor
1	354.47 sec	351.17 sec
2	177.05 sec	174.21 sec
4	88.6 sec	86.43 sec
8	44.53 sec	43.12 sec
10	35.83 sec	32.27 sec
15	24.34 sec	21.58 sec
20	18.78 sec	16.5 sec
25	15.59 sec	13.46 sec
30	15.37 sec	11.2 sec
36	15.05 sec	10.5 sec
37	16.8 sec	10.11 sec
40	18.61 sec	9.53 sec

By our experiment, we observe that the performance of using multiple threads is better than a single thread. And we find that for a single-core processor, while the number of threads is up to 25, the transmission time decreases very slowly, even decreases no more. This is because the loading of a single-core processor is close to full while the number of threads up to 25, so the performance does not grow up proportionally to the increase of threads. However a dual-core processor improves the performance while the loading of a core is full. When the number of threads is above 36, the transmission time for the single-core processor is not reduced. We observe that the loading of a single-core processor is full as the number of threads is above 36. The threads will ues the processor concurrently and race conditions will occur.

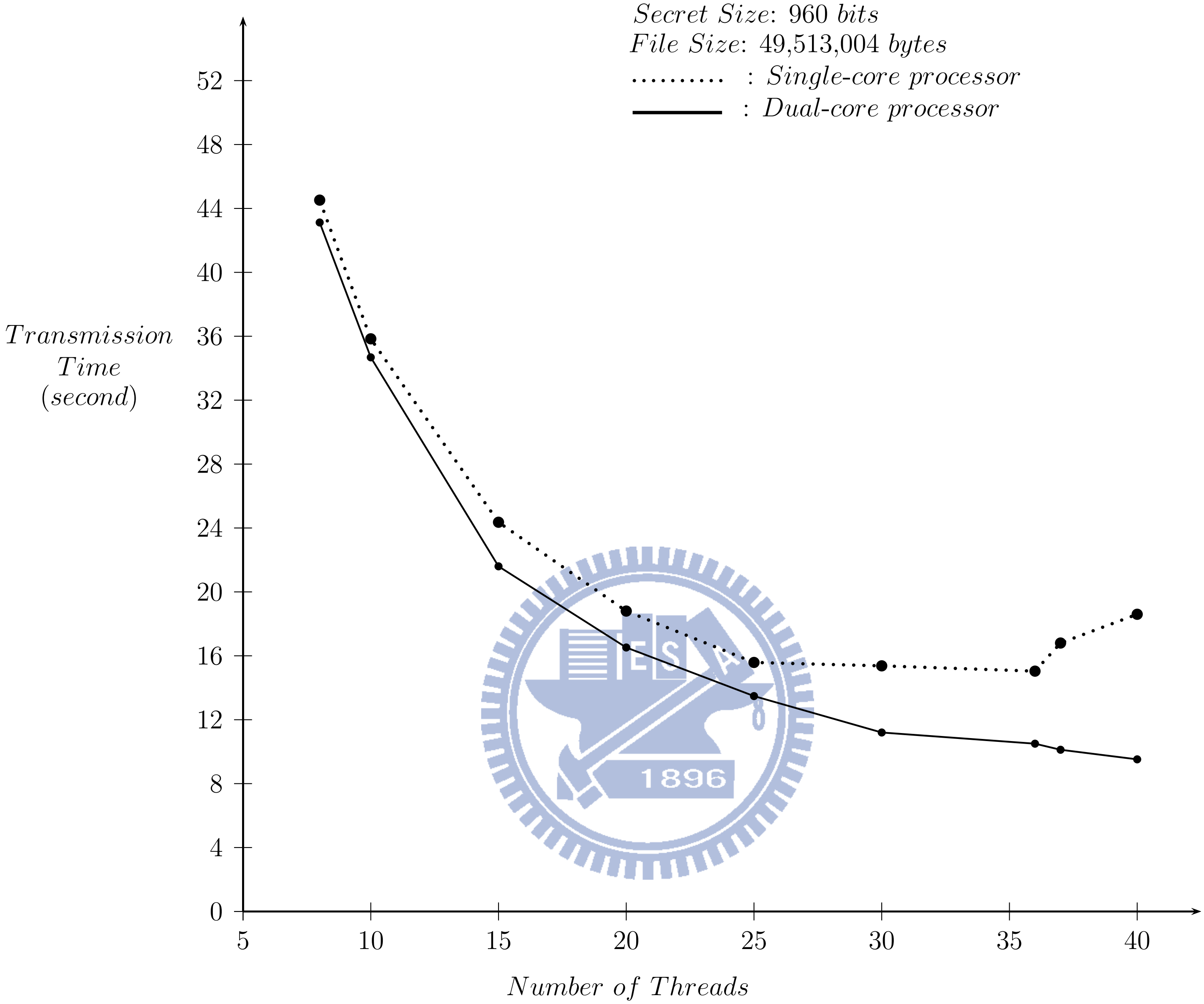

Figure 4.3: Transmission Time vs. Number of Threads for the file model

Chapter 5

Conclusions and Future Work

In this thesis, based on permutation array, we apply two encoding algorithms and one decoding algorithm for secure steganographic communication. We take the advantage of the permutation array to confuse a passive adversary and to resist the attack from an active adversary. We aslo apply these algorithms to implement the STEC system using SCTP protocol. STEC is a network steganography system which permutes the order of data packets intentionally to transmit steganograms. However we consider the network environment is ideal, that is there is no lost and out-of-order packets under experimental environment and the server in STEC only supports a single connection. The developing platform can only be under 32-bits operating system and there are only two kinds of services for STEC. In the future we will address these shortcomings and limitations of STEC to improve it and make STEC more adaptive to real network environment. We will develop STEC to support multiple connections and extend the services.

Because STEC can deliver more information without additional bandwidth and users are not aware of extra information during using STEC, we may consider some interesting applications that can utilize these properties. For example, there are currently many applications able to block advertisement in web. STEC can be used to solve this problem. When users use some service provided by STEC server, it will download advertisement to them simultaneously. Some products need a key to be used legitimately. We can install the product through the server owning this feature to deliver the key. As long as a server wants to send additional undetected information, the character of STEC is suitable for such applications.

Bibliography

[1] R. C. Chakinala, A. Kumarasubramanian, R. Manokaran, G. Noubir, C. Pandu Rangan, and R. Shndaram, ”Steganographic communication in ordered channels,” *Proceedings of the 8th Information Hiding Workshop*, Lecture Notes Comput. Sci., Old Town Alexandria, VA, Jul. 10-12, 2006.

[2] A. Bogomjakov, C. Gotsman, and M. Isenburg, ”Distortion-Free Steganography for Polygonal Meshes,” *Proceedings of the 8th Information Hiding Workshop*, Lecture Notes Comput. Sci., Old Town Alexandria, VA, Jul. 10-12, 2006.

[3] D. Inouc and T. Matsumoto, ”A scheme of Standard MIDI Files steganograohy and its evaluation,” *Security and Watermarking of Multimedia Contents IV*, San Jose, CA, United States, pp. 194-05, Jan 21-24 2002.

[4] T. Kløve, T. -T. Lin, S. -C Tsai and W. -G Tzeng, ”Permutation Arrays Under the Chebyshev Distance,” *IEEE Transactions on Information Theory*, vol. 56, no. 6, pp 2611-2617, 2010.

[5] M. Kwan, The Gifshuffle, http://www.darkside.com.au/gifshuffle/

[6] T. -T. Lin, S. -C Tsai and W. -G Tzeng, ”Efficient Encoding and Decoding with Permutation Arrays,” *IEEE International Symposium on Information Theory (ISIT)*, Toronto, Canada, pp. 211-214, July 6-11, 2008.

[7] G. J. Simmons, ”The prisoners’ problem and the subliminal channel,” *Advances in Cryptology: Proceedings of CRYPTO ’83*, Plenum Press, New York, pp. 51-67, 1984.

[8] D. E. Stevenson, "PNG Palette Permuter," *Proceedings of the 11th annual SIGCSE, Conference on Innovation and Technology in Computer Science Education*, pp. 143-147, 2006.

[9] J. M. E. Tapiador, J.C. Hernandez-Castro, A. Alcaide, A. Ribagorda, "On the distinguishability of distance-bounded permutations in ordered channels," *IEEE Transactions on Information Forensics and Security*, vol. 3, no. 2, pp. 166-172, 2008.

[10] K. Szczypiorski, "Steganography in TCP/IP Networks," State of the Art and a Proposal of a New System – HICCUPS, Institute of Telecommunications' seminar, Warsaw University of Technology, Poland, November 2003.

[11] N. B. Lucena, G. Lewandowski, and S. J. Chapin, "Covert Channels in IPv6," Proc. Privacy Enhancing Technologies (PET), pp. 147-66, May 2005.

[12] W. Fraczek, W. Mazurczyk, K. Szczypiorski, "Stream Control Transmission Protocol Steganography," Second International Workshop on Network Steganography (IWNS 2010) co-located with The 2010 International Conference on Multimedia Information Networking and Security (MINES 2010), Nanjing, China, November 4-6, 2010.

[13] R. Stewart, Q. Xie, "Stream Control Transmission Protocol," IETF RFC 4960, October 2000.

[14] F. Petitcolas, R. Anderson, M. Kuhn, "Information Hiding Survey: IEEE Special Issue on Protection of Multimedia Content," July 1999.

[15] J. Postel, "Transmission Control Protocol," IETF RFC 793, September 1981.

[16] http://bolero.ics.uci.edu/kfujii/Jpcap/doc/index.html

[17] R. Stewart, M. Ramalho, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension," IETF RFC 3758, May 2004.

[18] S. Ladha and P. D. Amer, "Improving Multiple File Transfers Using SCTP Multistreaming," IEEE Performance, Computing, and Communications, 2004 IEEE International Conference, pp. 513-522, 2004.

[19] A.L. Caro, P.D. Amer, P. Conrad, G. Heinz, ”Improving Multimedia Performance over Lossy Networks via SCTP,” ATIRP 2001, College Park, MD, Mar. 2001.

[20] M. Molteni and M Villari, ”Using SCTP with Partial Reliability for MPEG-4 Multimedia Streaming,” BSDC on Europe 2002, October 2002.