MMSP: An Alternative Transport Protocol for Multiple Co-existing Networks

by

Haoran Song

B.Sc., The University of Victoria, 2006

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

The Faculty of Graduate Studies

(Computer Science)

The University Of British Columbia

(Vancouver)

July, 2008

Abstract

Mobile Multi-Streaming Protocol (MMSP) is a new protocol combining and abstracting the essential functionalities of TCP and UDP for multiple coexisting networks. It inherits the good characteristics from TCP while overcomes many drawbacks from it. The most important features of MMSP is multi-streaming within a single connection. Our work proposes, investigates and provides a concept for maintaining a reliable connection by setting up multiple streams via different network interfaces. We designed and implemented a bunch of new packet formats, algorithms and state machines.

Our experiments have successfully proved that MMSP provides a functional solution to satisfy many requirements for up-layer protocols and applications in both wired and wireless networks, producing higher network throughput, security and reliability. Especially for mobile networks, the unique design and characteristics of MMSP is in the ascendant. It is a successful protocol extension of the transport layer on IP stack.

Table of Contents

List of Tables

List of Figures

Acknowledgements

This thesis would have never been completed without the support of many people. I would like to acknowledge the invaluable contribution of my supervisor Dr. Son Vuong who has been a constant source of inspiration for me. He has always been generous with his time and discussions for this thesis and reminded me to take time to enjoy the small things in life. I appreciate he always encouraged me and provided constructive feedback on our work.

I also want to thank Dr. George Tsiknis for reading my thesis and providing useful feedback.

My thanks go out to my family for their constant interest in my research, even though I never took time to give a satisfying explanation of it.

Thanks to Yvonne Chen, Billy Cheung, and Aaron Hilton for improving this thesis with insightful comments and suggestions.

Finally, there are many people in the computer science department who have made my life easier and more enjoyable over the years. I would like to thank all of the members of the NIC lab for their help and for interesting discussions.

Chapter 1

Introduction

MMSP is a reliable, multi-streamed, message-oriented transport protocol layered on top of the unreliable and connectionless IP layer. It operates the same level as TCP and UDP protocol and inherits lots of good characteristics from TCP, such as acknowledgement based handshaking, duplicate message checking, data corruption and packet loss detection. Furthermore, it introduces a new concept of multi-streaming, which can support multiple streams to deliver data within one message-oriented connection. This idea overcomes the instability of the wireless network due to the frequently changed topology. MMSP let mobile nodes have more flexibility to move around within larger areas without losing the already established connection as multiple links can coexist to keep the connections alive [1].

1.1 Motivations

MMSP is not a replacement of current TCP and UDP. Instead, it is an optimized, feature extended and updated protocol derived from TCP and UDP. Figure 1.1 presents the layer where the middleware MMSP is designed on IP stack. TCP is one of the most important protocol suites in use today. It provides reliable, in-order delivery of a stream of bytes, making it suitable

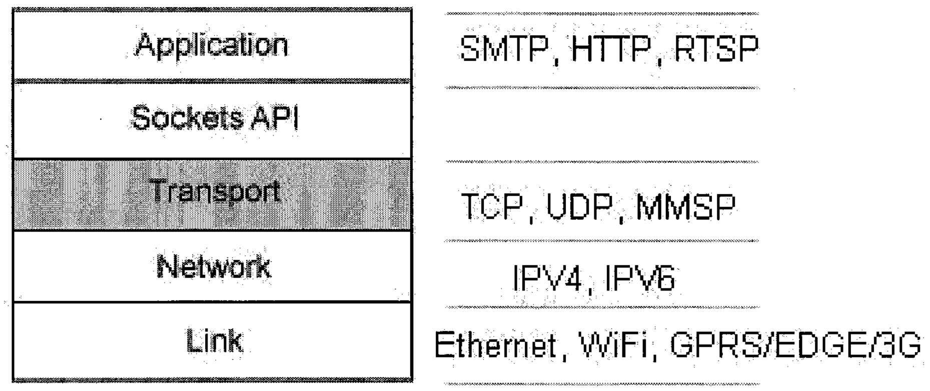

Figure 1.1: MMSP on IP stack

for applications like file transfer and email. It guarantees that unreliable IP packets are all transmitted without data loss. In terms of congestion, it specifies its congestion control algorithm and minimizes network congestion. TCP retransmits discarded packets, rearranges out-of-order packets and passes the consecutive byte stream to upper layer applications [1]. It significantly simplifies the task of writing top layer protocols such as RTSP, SMTP and other network applications.

User Datagram Protocol (UDP) is a simple transport layer protocol which provides the a basic data transmission between hosts. As opposed to TCP, UDP uses datagram as the unit to delivered data. It does not guarantee reliability or ordering in the way TCP does. Data is not transmitted as consecutive streams. UDP also avoids the overhead of checking sequence numbers and retransmissions [14]. All of these simplicities result in UDP becoming faster and more efficient. Therefore, many time-sensitive applications choose to use UDP since dropped packets is not considered as a major factor of the performance

Both TCP and UDP have been extensively used in Internet due to their admitted robustness, usability and stability. However, both of them have their respective limitations which make them inappropriate for satisfying specific group of applications' demands. For example, video streaming with TCP requires RTSP packets and RTP packets interleaved [18]. But it is not always necessary to have RTP packets to arrive in order or without packets loss. Most of the time, TCP video streaming only requires RTSP command packets to be in a manner of connection-oriented, but it is not necessary for the RTP data packets. Having RTP packets all delivered in a connection-oriented manner may result in unnecessary overhead of TCP streaming. In this case, TCP protocol can not totally fit into some mobile devices with slow processors in terms of streaming.

As discussed above, some applications need reliable transfer without sequence in-order maintenance; others may only require partial ordering of the data. Under these circumstances, TCP incurs unnecessary delay and overhead. As opposed to the limitations of TCP and UDP, MMSP provides more flexibilities and powerful mechanisms for solving the above discussed problems. MMSP is also motivated by the idea of automatically switching between different wireless networks. An example is to use WLAN to transmit messages and switch to GPRS/EDGE/UMTS while WiFi becomes out of range.

Recently, as more and more mobile device vendors choose to unite 3G, WiFi and WiMax into one entity, simply using TCP and UDP becomes unrealistic to support these hybrids of different networks or multiple networks with the same type. For example, in the past two years, cellphones like Nokia and LG, smart phones like BlackBerry and laptops all have different network chipsets installed to support different types of network interfaces. GPRS/3G and WiFi have been widely unified and incorporate into a single device. Although TCP and UDP are still widely accepted as the most decent transport layer protocols, their original design can no longer satisfy the latest multi-network requirements, such as multi-streaming by sharing one connection via multi-streaming paths. In a leading case study in 2007, UMA technology has become widely incorporated into mobile devices. As a result, automatical phone-call switching from WiFi to GSM Relay has become reality [2]. More and more requirements from recent technology tells us that just using one protocol TCP or UDP is no longer good enough to provide functional services for devices that require multi-access networks. Therefore, engineers begin to search for a more generic protocol that not only provides services for each network media, but also transparently combines different networks in holy matrimony.

Unlike TCP and UDP, MMSP presents a more updated and powerful transport layer protocol to solve the handover problems and source sharing issues among various networks. This thesis introduces the basic concept of MMSP, detailing its design, implementation, evaluation and compares MMSP against TCP.

1.2 Thesis Contributions

In this thesis, we propose MMSP to be an extension of current transport layer protocols to qualify additional requirements that could not be handled by TCP or UDP. We demonstrate that MMSP provides a higher throughput than TCP in wireless networks and contains a mechanism to solve handover between multiple transmission networks. Besides that, we also prove that MMSP is more secure and reliable.

We highlight MMSP's advantages by evaluating the current TCP drawbacks with respect of some special purpose applications, such as UMA and Video Streaming. We compare the performance and functionality between TCP and MMSP, to learn what kind of applications are suitable to use MMSP.

We have designed and implemented a prototype of MMSP driver on Windows XP and implemented several client-server testclamps to evaluate our experiment results. During the design portion, we investigate the shortcomings of TCP and introduce new features using MMSP to alleviate those said shortcomings. One novel and great advantage using MMSP is maintaining multiple streams within one connection even when one of the link interfaces gets disconnected. We realize that as a middleware, MMSP largely reduces the unnecessary delay time of the head-of-line block issue that exists in TCP. Moreover, by using the third party application Iperf, we could clearly and easily gather different test results from TCP and MMSP experiments, and compare them. After a set of experiments, we proved a fact that the throughput of MMSP benefits from using multiple streams in various wireless networks. The second indirect contribution is with respect to Windows network development. Our work provides evidence that the most used operating system Windows XP is able to add a third party driver on their protocol stack.

1.3 Thesis Organization

The whole thesis contains five parts. The remaining thesis is organized as follows. In chapter 2, we give a background overview of TCP and discuss its unsuitable properties against some specific applications, such as UMA and video streaming. In this chapter, we also introduce the concept of MMSP, its properties, features and advantages. In Chapter 3, we discuss our motivation for using MMSP and provide the design and implementation details. In Chapter 4, we analyze the gathered data from various experiments carried out on our MMSP module and analyze the experiment results. The final conclusion and future work is discussed in chapter 6.

Chapter 2

Background and Related Work

2.1 Evaluation of Unsuitable TCP Service

Since MMSP is derived from TCP, it is helpful to firstly examine the advantages and shortcomings of TCP. The Transmission Control Protocol (TCP) is one of the core protocols of the Internet protocol suite. TCP provides reliable, in-order delivery for a stream of bytes. It was firstly formally specified in 1974 and was designed to be flexible enough to handle the physical differences in most computers, routers and networks in general, but still provide a standard to allow these physically different entities to be able to transmit data amongst themselves. It is on top of the IP layer and makes the unreliable IP packet transmission become reliable. It has been widely used for more than twenty years with little changes [7].

However, the initial TCP design does not cover the special case that a host has multiple points attached to the Internet. A TCP connection is only able to hold a single stream using one end point. This drawback is very costly since it causes unnecessary message delay and wastes the transmission capability when an alternate available path exists. The limitation of the TCP design makes it difficult to apply multi-steaming to many signal-control applications such as video streaming and IP phone calls. To adhere to TCP standard, these applications can only set up one TCP connection through one internet interface and all other interfaces are wasted.

One instance that highlights the shortcoming of TCP is:

Example

Video/Audio streaming can be established by two ways: TCP streaming or UDP streaming. Streaming using TCP requires both RTSP control packets and UDP packets interleaved within one TCP stream [11]. Both RTSP and RTP are encapsulated into the TCP payload, shown in Figure 2.1. Figure 2.2 presents the current multimedia streaming network. However, having both RTSP control packets and RTP data packets in one TCP stream may result in unnecessary and serious head-of-line block issue [8]. For example, if several RTP packets are out of order, lost and wait for being retransmitted, the delivery of RTSP control packets may be delayed. The delayed RTSP control packets will lead to inaccurate media control in the application layer [10].

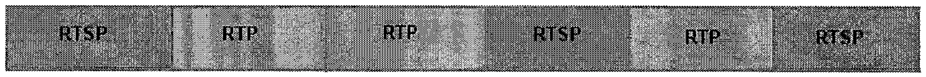

Figure 2.1: Interleaved RTSP and RTP packets with TCP streaming

Also, since TCP only supports one streaming connection, it enlarges the dependency on only one access point and restricts the lifespan of the whole

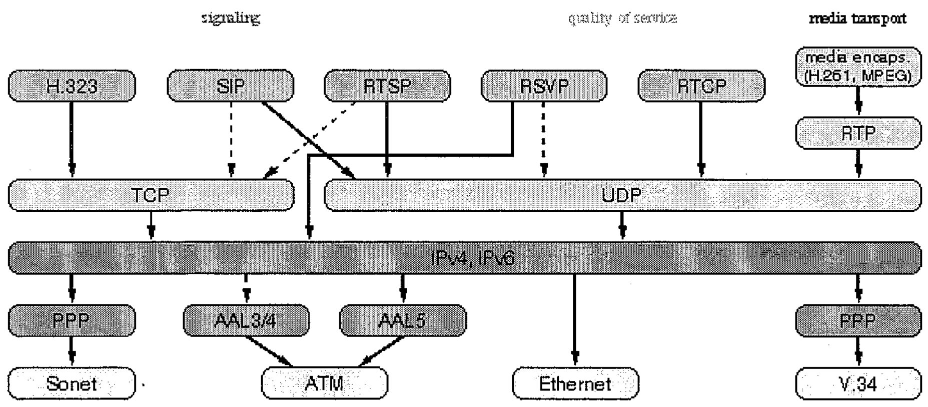

Figure 2.2: RTSP Streaming Protocol Stack

streaming process to the reachability of this access point. Mobility in short range networks like WLAN exposes an obvious weakness at this point. The whole streaming process has to be terminated when the only used access point becomes unavailable. To solve this problem, multi-streaming becomes a good solution for devices that have more than one access points to maintain a connection through other active links. In other words, when a mobile device gets one link down, it should still be able to keep the connection by using other available paths. For instance, video streaming on a mobile phone with accesses to both WiFi and EDGE should be still able to continuously watch the video via EDGE network after WiFi becomes out of range.

Since TCP is not enough to satisfy applications' requirements such as those we discussed above, we propose our new type of protocol which can keep the basic functionality of TCP and overcome its shortcomings. We name it MMSP.

Services or Features	MMSP	TCP	UDP
Connection-oriented	yes	yes	no
Full Duplex	yes	yes	yes
Reliable Data Transfer	yes	yes	no
Partial-reliable Data Transfer	optional	no	no
Ordered Data Delivery	yes	yes	no
Unordered Data Delivery	optional	no	yes
Flow Control	yes	yes	no
Congestion Control	yes	yes	no
Application PDU Fragmentation	yes	yes	no
Application PDU Bundling	yes	yes	no
Multi-streaming	yes	no	no
Protection against SYN Flooding Attacks	yes	no	not exist
Allows Half-closed Connections	no	yes	not exist
Multiple Path Selection	yes	not exist	not exist

Table 2.1: Services or Features between MMSP, TCP and UDP

2.2 Overview of MMSP

Mobile Multi-Streaming Protocol (MMSP) is at the same layer as TCP and UDP and it overcomes some limitations of them. It unites the advantages of TCP and UDP while providing a multi-streaming capability. MMSP is designed to allow one oriented connection to be split into multiple streams. Even when one link is offline, data flow is still able to spread out through the other links. MMSP can transmit data via multiple paths to avoid the so-called head-of-line blocking. Another strong point of MMSP is that it supports both ordered and unordered data delivery. This makes it suitable for a wide class of applications. In addition to many new features, MMSP is more efficient and robust. Table 2.1 presents a comparison of MMSP, TCP and UDP. As Table 2.1 indicates, MMSP inherits TCP-like mechanisms such as reliable transmission and ordered delivery. It still uses the flow and congestion control algorithms similar to TCP, such as slow start, fast recovery and fast retransmit. Moreover, it uses 32 bit checksum as opposed to the 16 bit checksum. Unlike TCP, it allows several streams to coexist within one MMSP connection. It defines a stream as a sequence of messages (like data chunks in UDP) rather than bytes. It uses 4-way handshaking to set up a connection and 3-way handshake to shutdown a connection. MMSP does not have a half-open or half-close state as TCP. The details of all these new features are discussed in Chapter 4.

2.3 Multi-access and Multi-streaming

With the increasing demand for wireless personal area networks (WPAN), Wireless Local Area Networks (WLAN) are being developed to provide high bandwidth wireless access for mobile devices in a limited geographical area [5]. WLAN service is cheaper than most GPRS services. However, the performance of WLAN suffers from radio frequency interference and severely limits the coverage area of WiFi signals. Fortunately, the convergence of IP and telephony networks enables user terminals to have multiple accesses to Internet. The combination of several network accesses provides user terminals better connectivity and more robustness.

The multi-access technology normally consists of link layer, network layer and transport layer multiple access. We use MMSP as a solution of the transport layer for multi-access scenario.

A multi-streaming host that has more than one IP address can establish one MMSP connection containing multiple streams by allocating each separate stream for each assigned IP. The host can effectively control and aggregate these multiple streams for delivering data. Since TCP only allows one connection to be set up on one IP address, it does not maximize the network's usage when a host has multiple network paths. Figure 2.3 is one example of hosts that have multiple network interfaces. The client has two network interfaces: EDGE/3G and WiFi; the server uses another two network interfaces: Ethernet and WiFi. If using TCP, the client and server are only able to use network interface to set up a connection (either the IP1 or IP2 for the client and ether the IP3 or IP4 for the server), but with MMSP the client and the server are able to have multiple streams coexisted in one connection. In the example of Figure 2.3, stream1 uses IP1-IP3 and stream2 uses IP2-IP4. Every stream in MMSP is bundled by a stream number and

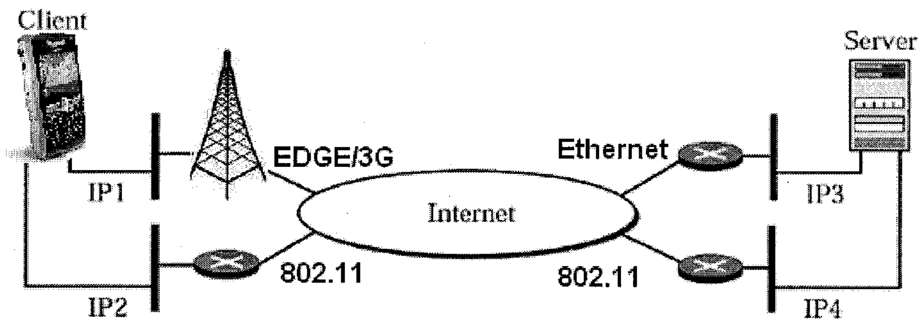

Figure 2.3: Peers with Multiple Interfaces

encoded into the MMSP packets, all of which are transmitted to the network via one connection. The concept of multi-steaming is very important in that it avoids the head-of-line blocking problem that happens very often in TCP. In other words, blocking one stream can not result in blocking other streams. Figure 2.4 provides a visual relation between a MMSP connection and its streams.

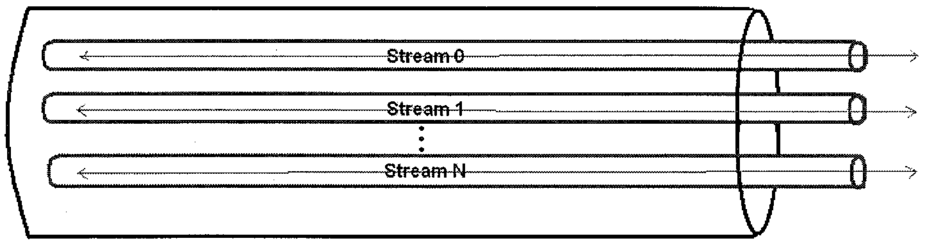

Figure 2.4: Multiple Streams in One MMSP Connection

2.4 Functionality and Terminology of MMSP

MMSP uses its special Automatic Changeover Correction and Retransmission Correction algorithm to control and monitor the whole connection path selection. When one link becomes unavailable, it redirects the data from that broken link to other available paths. The top layer applications are not affected and are not even aware of these underground changes. MMSP is totally transparent to the top layers.

Automatic Changeover Correction maintains the connectivity of network applications. Let us take an example that a laptop has both WiFi and Ethernet interfaces to Internet. When the laptop has a very fixed position, people prefer to use the high-speed Ethernet network. In MMSP, this stable Ethernet interface is selected to build a primary path and the wireless one is used for a secondary path. The sender will continuously monitor whether any of IP addresses of the destination host are reachable. If the network link of the primary address fails, for example the laptop is moved away from the original place and gets disconnected form Ethernet, MMSP will maintain the connection by switching the stream from Ethernet to WiFi. Later, after the Ethernet network is recovered, MMSP will resume the communication using Ethernet.

Retransmission Correction is a policy to utilize multiple streams to maximize transmission performance. When retransmission happens, the sender presumes a network error or congestion occurs. The sender will retransmit data through alternate network paths. This largely avoids the head-of-line blocking issue in TCP and results in a better chance of success.

SSEQ Number and TSEQ Number

SSEQ Number stands for Stream Sequence Number. The MMSP user can specify the number of streams supported by the MMSP connection at the connection startup time. In order to ensure all messages are delivered in order, MMSP assigns SSEQ number to each message passed to it by the MMSP user. This can refer to its usage in TCP. However, as opposed to TCP, when one stream gets blocked by waiting for the next in-sequence user message, data delivery via other streams can still resume.

TSEQ Number stands for Transmission Sequence Number. MMSP assigns a TSEQ number to each user data fragment or unfragmented message. The TSEQ is independent from any SSEQ Number assigned at stream level. The receiver acknowledges all TSEQs arrived, even if there are gaps in the middle. This way, reliable delivery is kept functionally separate from sequenced stream delivery [15].

User Data Fragmentation is provided on demand. MMSP fragments user messages to guarantee the MMSP packet's size plus the lower layers' encapsulation fit in the path MTU. Upon being received, fragments are reassembled into complete messages before being passed to the MMSP consumer.

Chapter 3

Design and Implementation

3.1 MMSP Packet Format

MMSP introduces a new mechanism of building packets. Different from TCP packet (TCP header plus payload), a MMSP packet is composed of a common header and EXT (extension) blocks. Each EXT block contains either control information or user data. Multiple EXT blocks can be bounded into one packet, except the SYN, SYN-ACK, RST and SHUTDOWN blocks. For these special signal-control blocks, they must not be bundled with any other blocks in a packet. The total size of resultant IP datagram including the MMSP packet and IP header shall not be greater than the current path MTU [13]. All integer fields in an MMSP packet shall be transmitted in network byte order.

When building a packet with EXTC (Extension Control) blocks and DATA blocks, an endpoint must locate the EXTC blocks firstly in the MMSP packet and followed by DATA blocks. See Figure 3.1 for the MMSP packet format.

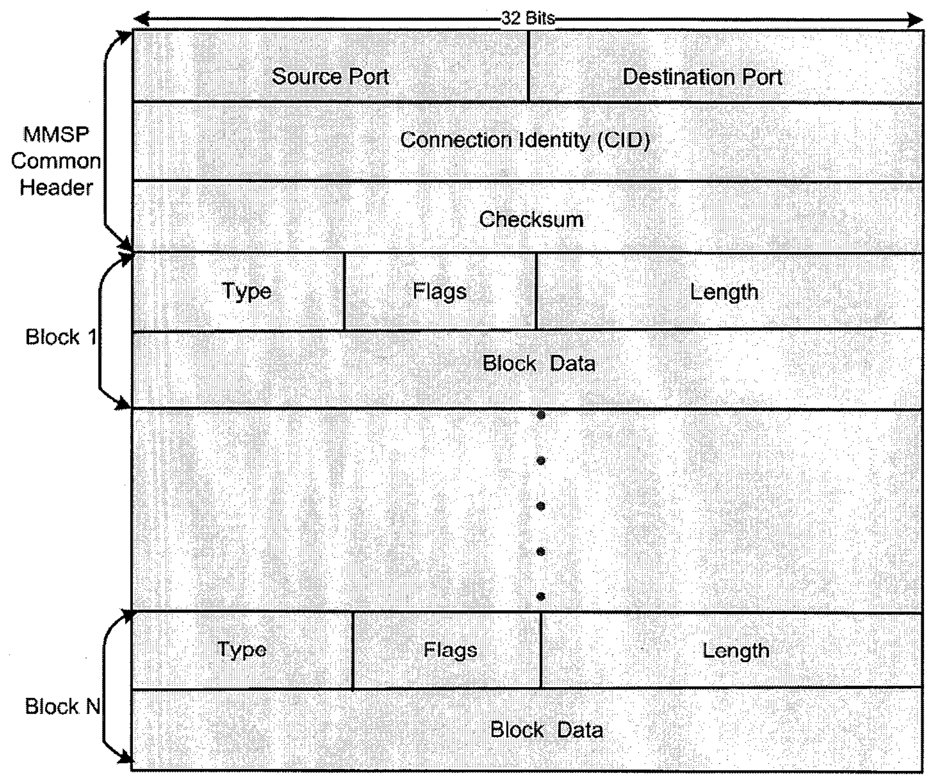

Figure 3.1: MMSP Packet Format

3.1.1 MMSP Common Header Description

Source Port: 16 bits unsigned integer It indicates the sender's port number. Receiver can combine the source IP with this port number, destination IP and port to identify which connection this packet belongs [15].

Destination Port Number: 16 bits unsigned integer It indicates the port number where this packet is delivered. The receiver will use this port number to demultiplex the MMSP packet to the correct receiving endpoint

[15].

Checksum: 32 bits unsigned integer The checksum uses Adler-32 algorithm to do the calculation [6].

Connection Identity: 32 bits unsigned integer The packet's receiver uses CID to validate the sender of this MMSP packet. During transmission, the value of this Identification is set to the initial packet received by the peer endpoint during the connection initialization [12].

3.1.2 MMSP Block Field Description

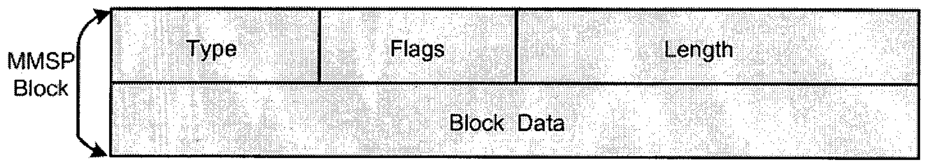

Figure 3.2: MMSP Generic Block Format

Different from TCP, MMSP introduces the concept of blocks. The Figure 3.2 illustrates the field format for the blocks to be transmitted in the MMSP packet.

Block Type: 8 bits unsigned integer This field identifies type of information that the block contains in the Data filed. It ranges from 0 to 255. The types are classified in Table 3.1:

Block Types are encoded such that the highest-order 1 bit indicating the action that must be taken if the processing endpoint does not recognize the

Value	Type
0	Payload Data
1	SACK
2	SYN
3	SYN ACK
4	PING (INFO)
5	PONG
6	FIN
7	FIN-ACK
8	FIN-Done
9	RST
10	ERROR
11	PROBE
12	PROBE ACK

Table 3.1: Block Types

block type.

0: Stop processing this MMSP packet and discard it. Also stop processing any further blocks within this packet.

1: Skip this Block and continue processing.

Block Flags: 8 bits These 8 bits are reserved for usage of different block types as given by the Block Type. They are set to zero on delivering unless otherwise specified.

Block Length: 16 bits unsigned integer The field represents the whole size of the block in bytes including the Block Type, Block Flags, Block Length, and the Block Data fields. But the total length of the Block can be greater than the Block Length. The total length of a block (including Type, Length, and Data fields) is designed to be a multiple of 4 bytes in that the whole MMSP packet is always in 32 bits alignment for processing. If the length of the whole block does not satisfy the multiple of 4 bytes, the sender must append the block with all zero bytes until the multiple of 4 bytes occurs. The sender shall never pad with more than 3 bytes. The receiver must ignore the padding bytes.

Block Data: a variable field This field contains the actual payload to be transmitted in the Block. Different Block Types may use different formats of this field.

3.2 MMSP Block Definition

MMSP blocks are mainly defined by two groups of types: DATA Block and Control Block.

3.2.1 Payload Data (DATA) Type 0

The following format is used for the DATA Block:

Type: 8 bits 0 indicates this Block is DATA type Block.

Reserved: 5 bits These bits are reserved for further use. They should be set to all 0's and ignored by the receiver.

U bit: 1 bit The U bit means unordered bit. If it is set to 1, it indicates that this is an unordered DATA Block, and no matter there is a SSEQ or not, the SSEQ will be ignored.

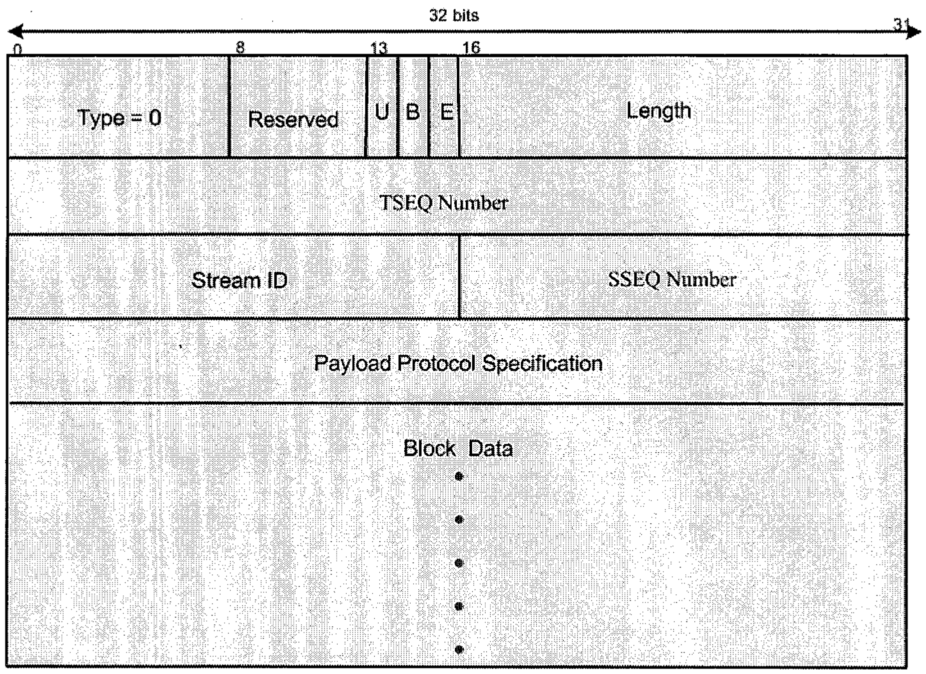

Figure 3.3: Data Block Format

B bit: 1 bit The B bit means the beginning bit of the fragment. If this bit is set to 1, it indicates the first fragment of a user message.

E bit: 1 bit The E bit means the ending bit of the fragment. If this bit is set to 1, it indicates the last fragment of a user message.

Table 3.2 is the combination of B and E bits, used to present different types of fragments.

We define both B and E bits to 0 to indicate the middle piece of a fragment and 1 to indicate an unfragmented message.

B	E	Combination Description
0	0	Middle piece of a fragmented message
0	1	Last piece of a fragmented message
1	0	First piece of a fragmented message
1	1	Unfragmented message

Table 3.2: MMSP Block Fragment Description

Length: 16 bits unsigned integer This field contains the value of the length of the DATA block in bytes from the beginning of the block to the end of the data block field, but excluding any padding 0's. A DATA block without user data field has Length set to 16 bytes and the last field is Payload Protocol Specification.

TSEQ: 32 bits unsigned integer This field indicates the TSEQ number for this DATA block. TSEQ can only range from 0 to 2^32 -1 and wraps back to 0 once becomes overflowed. Note: when a user message is fragmented into multiple blocks, the receiver needs to use TSEQ to reassemble the message. This means the TSEQ of each fragment of a disassembled message must be strictly sequential.

Stream ID: 16 bits unsigned integer This value identifies to what stream the user data belongs.

SSEQ: 16 bits unsigned integer This field represents the stream sequence number of the following data within the stream identified by the Stream ID. It ranges from 0 to 65535.

When a user message is fragmented for delivering, each of the fragments must use the same Streaming Sequence Number (SSEQ).

Payload Protocol Specification: 32 bits unsigned integer This field is used by application layer instead of MMSP. The value is passed to MMSP by its upper layer and sent to its peer. The peer uses this specification to identify the type of information being carried in this DATA block. If it is set to 0, it indicates no application identifier is specified by the upper layer for this payload data.

Data: variable length This is the real carried payload data. MMSP pads this field with all 0 bytes to make the whole Block be multiple of 4 bytes. But this padding shall never be added more than 3 bytes.

3.2.2 SACK Type 1

SACK is Selective Acknowledgement block used to respond to the received DATA blocks. It informs the peer receipt of DATA blocks and missed blocks as represented by their TSEQs. MMSP also uses end-to-end window-based flow and congestion control mechanism from TCP [3]. The receiver of the DATA blocks can control the transmission rate at which the sender is sending by providing a 32-bit based window size

Flags: 8 bits All these 8 bits are set to 0 by the sender and ignored by the receiver.

Ack TSEQ: 32 bits unsigned integer This field stores the last DATA block's TSEQ received in sequence before the first missed one occurs.

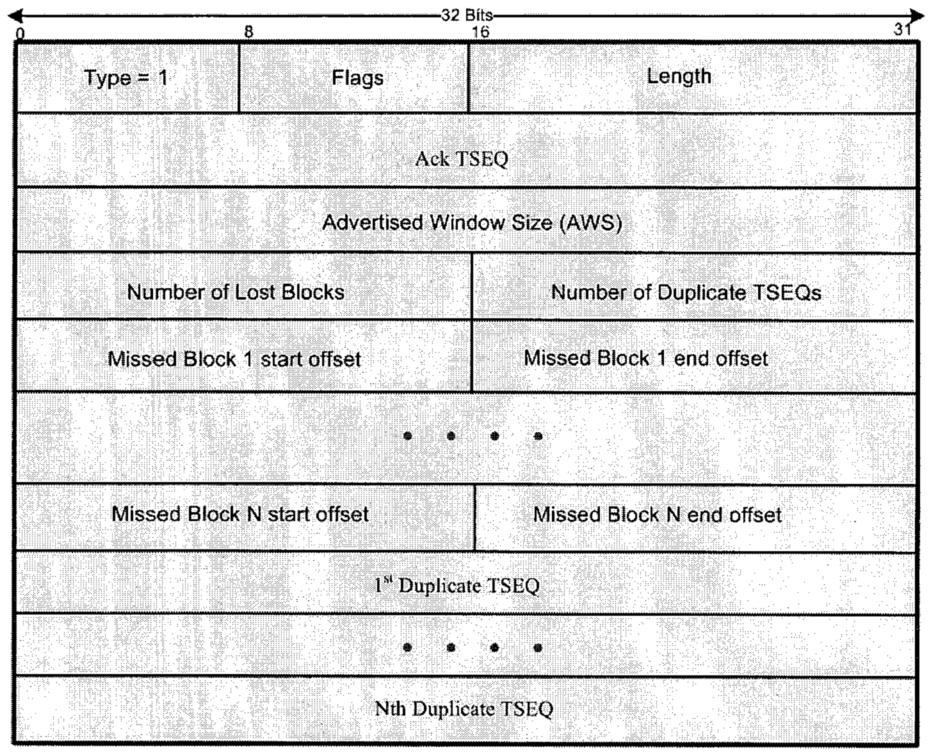
32 Bits
0
8
16
31
Type = 1
Flags
Length
Ack TSEQ
Advertised Window Size (AWS)
Number of Lost Blocks
Number of Duplicate TSEQs
Missed Block 1 start offset
Missed Block 1 end offset
Missed Block N start offset
Missed Block N end offset
1st Duplicate TSEQ
Nth Duplicate TSEQ

Figure 3.4: SACK Block Format

Advertised Window Size (AWS): 32 bits unsigned integer The field represents the buffer space in bytes reserved by the sender within a connection. The receiver can change the value of AWS when it sends SACK blocks to acknowledge the peer its own buffer space.

Number of Lost Blocks: 16 bits unsigned integer It indicates the number of blocks that are lost during transmission, i.e. the gap between the first arrived DATA block up to last arrived block.

Number of Duplicate TSEQs: 16 bit unsigned integer This field indicates the number of duplicate TSEQS the endpoint has received. All these duplicate TSEQs are listed in the duplicate block list.

Missed Blocks fields All these consecutive fields are used to contain lost blocks. The total number of these fields equals to the value stored in the field of Number of Lost Blocks. Each missed block has a 16-bit start offset and end offset of this DATA block. The actual TSEQ of these lost blocks is calculated by:

First missed block's TSEQ = Start offset + Ack TSEQ - 1;

Last missed block's TSEQ = End offset + Ack TSEQ - 1;

Note: If blocks are successfully received, their TSEQs must fall in the following condition:

$$AckTSEQ + startoffset \leq block'sTSEQ \leq AckTSEQ + endoffset$$

Duplicate TSEQ Fields: 32 bits unsigned integer This field indicates the number of duplicate TSEQs that have been received since last SACK was sent. Every time a block with the duplicate TSEQ arrived at the receiver, it is added to SACK's Duplicate list and the Number of Duplicate TSEQs field is incremented by 1. If there is no duplicate, this field is set to 0. For example, if a receiver received the TSEQ 5 two times, it should list TSEQ 5 once in the outbound SACK. After sending the SACK if it still received one more TSEQ 5, it would list TSEQ 5 as a duplicate once in the next outgoing SACK.

3.2.3 SYN Type 2

This Block is used to initiate a MMSP connection between two peers. The SYN Block is a control block that has format shown in Figure 3.5:

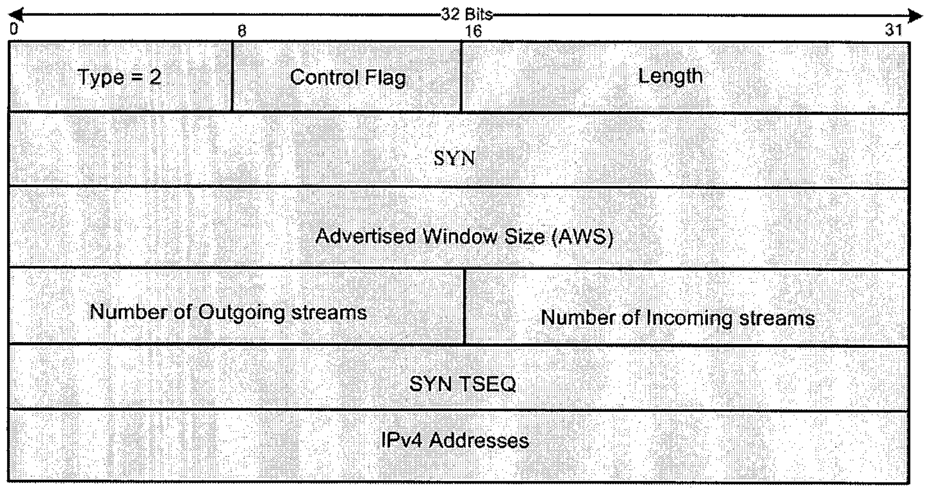
32 Bits
0
8
16
31
Type = 2
Control Flag
Length
SYN
Advertised Window Size (AWS)
Number of Outgoing streams
Number of Incoming streams
SYN TSEQ
IPv4 Addresses

Figure 3.5: SYN Block Format

Type 2: 8 bits SYN Block uses value 2.

Control Flag: 8 bits The Control Flag for SYN Block is reserved. All these 8 bits are set to 0 by the sender and ignored by the receiver.

SYN: 32 bits unsigned integer The value of SYN ranges from 0 to 2^32 -1. It is randomly generated by the sender, used to protect the transmission from the Sequence Number attacks as in TCP. The SYN tag is also placed into the Connection Identity field of all the MMSP SYN packets.

Advertised Window Size (AWS): 32 bits unsigned integer The field represents the buffer space in bytes reserved by the sender within a connection. The receiver can change the value of AWS when it sends SYN-ACK blocks to acknowledge the peer its own buffer space.

Number of Outgoing Streams: 16 bits unsigned integer It indicates the maximum outbound streams that the sender of this SYN block proposes to create in this connection. The value 0 must not be used in this field.

Number of Incoming Streams: 16 bit unsigned integer It indicates the maximum inbound streams that the sender of this SYN block allows the other end to create in this connection. The value 0 must not be used in this field.

SYN TSEQ: 32 bits unsigned integer This field defines the initial TSEQ that the sender will use in the SYN packet. It ranges from 0 to 2^32 -1.

IPv4 Addresses: 32 bits (Optional) This area contains a list of IPv4 addresses that the sender of this block supports. The number of IP addresses in the list must be equivalent to the value stored in the field of Number of

Outgoing Streams.

3.2.4 SYN-ACK Type 3

The SYN-ACK block is used to acknowledge the SYN block of an MMSP connection as the initiation state.

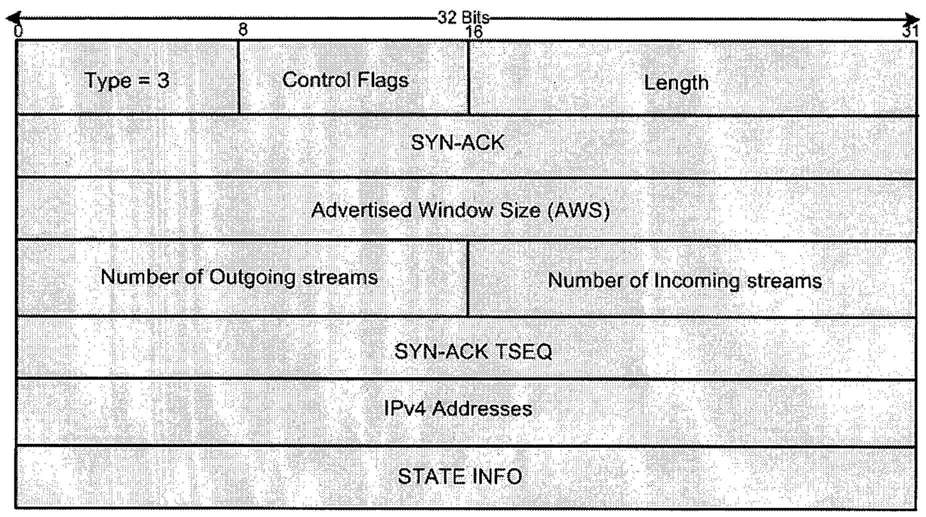

Figure 3.6: SYN-ACK Block Format

The description for Type, Control Flogs, and Length of SYN-ACK are similar to SYN block.

Advertised Window Size (AWS): 32 bits unsigned integer Similar to the AWS field in the SYN block, but it indicates the dedicated buffer space of the sender of the SYN-ACK packet.

Number of Outgoing Streams: 16 bits unsigned integer It indicates the maximum outbound streams that the sender of this SYN-ACK block proposes to create in this connection. The value 0 must not be used in this field.

Number of Incoming Streams: 16 bit unsigned integer It indicates the maximum inbound streams that the sender of this SYN-ACK block allows the other end to create in this connection. The value 0 must not be used in this field.

SYN-ACK TSEQ: 32 bits unsigned integer This field defines the initial TSEQ that the sender will use in the SYN-ACK packet. It ranges from 0 to 2^32 -1.

IPv4 Addresses: 32 bits (Optional) This area contains a list of IPv4 addresses that the sender of this block supports. The number of IP addresses in the list must be equivalent to the value stored in the field of Number of Outgoing Streams.

STATE INFO Format SYN-ACK contains mandatory information of the SYN receiver using variable-length block format. We name this variable-length block VEXT (Variable Extension).

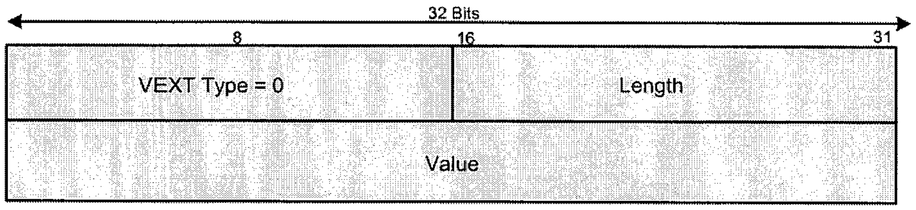

Figure 3.7: STATE INFO Format

VEXT Type: 0 INFO uses type 0.

VEXT Length The length is a variable size, depending on size of the INFO structure.

VEXT Value This filed contains the necessary state information for the sender of this SYN-ACK to build up a connection.

After one endpoint receives a SYN packet, it shall send a SYN-ACK as a response. The SYN-ACK packet must carry State INFO including a time stamp of this INFO being created, a lifespan of State INFO, and all other necessary information for it to establish the connection.

3.2.5 PING (INFO) Type 4

PING is a Block used during the initialization of a MMSP connection. It is sent firstly from the sender to its peer to complete the initialization process, upon receiving the SYN-ACK from its peer. This Block must precede any Data Block sent within the connection, but it can be bundled with one or more DATA blocks in the same packet.

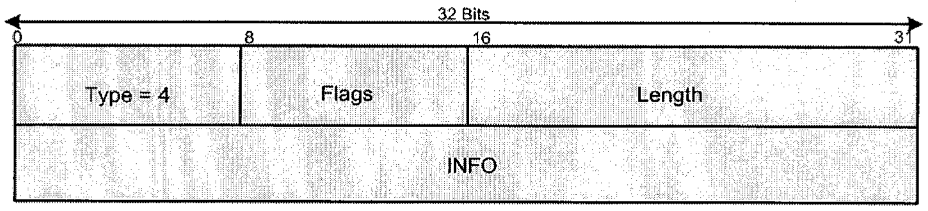

Figure 3.8: PING Block with STATE INFO Format

Flags: 8 bits integer For PING block, this field is set to 0 by the sender and ignored by the receiver.

Length: 16 bits unsigned integer It is the size of the block in bytes, including the block header (4 bytes) and the size of the INFO.

INFO: variable size INFO has a variable size. It contains the same INFO received in the SYN-ACK block.

3.2.6 PONG Type 5

PONG is a block used during the initialization of a MMSP connection. It acknowledges the sender's INFO PING block. This block must precede any Data Block sent within the connection, but it can be bundled with one or more DATA blocks in the same MMSP packet.

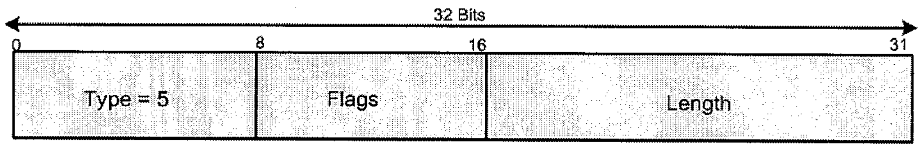

Figure 3.9: PONG Format

Flags: 8 bits integer For PONG block, this field is set to 0 by the sender and ignored by the receiver.

Length: 16 bits unsigned integer PONG block does not contain the INFO in its header. So the Length field is always set to 4.

3.2.7 FIN Type 6

FIN type block is used to initiate a polite close of a MMSP connection. It informs the other end to stop sending data upon receiving this FIN. It uses the following FIN format:

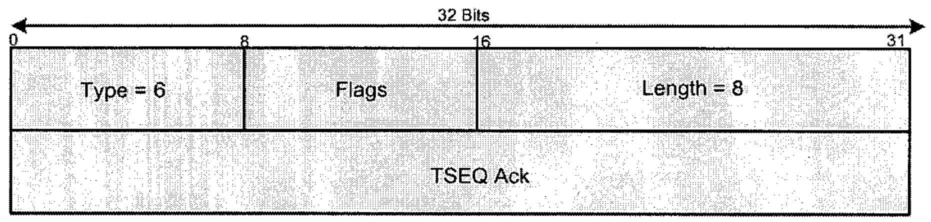

Figure 3.10: FIN Format

Flags: 8 bits integer In FIN block, this field is set to 0 by the sender and ignored by the receiver.

Length: 16 bits unsigned integer FIN block indicates the length of the whole FIN block. It is set to 8.

TSEQ ACK: 32 bits unsigned integer This field contains the TSEQ number of the last block received in sequence. It is used to acknowledge to sender's last block and inform its peer it is ready to shutdown the connection.

3.2.8 FIN-ACK Type 7

This Block is used to acknowledge the receipt of the FIN block at the completion of the Close process.

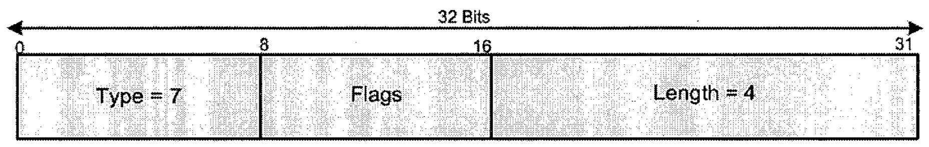

Figure 3.11: FIN-ACK Format

Flags: 8 bits integer This field is set to 0 by the sender and ignored by the receiver.

Length: 16 bits unsigned integer FIN-ACK only contains 4 bytes.

3.2.9 FIN-DONE Type 8

This block uses the exactly same format as FIN-ACK, but Type is set to 8. The receiver of a FIN-DONE shall accept a packet only if the CID field of this packet matches its own CID or its peer's CID. Otherwise, the receiver will discard the packet for security purpose [10, 12]. The receiver should not take any further action when it receives a FIN-DONE block in the FIN-ACK-SENT state.

3.2.10 RST Type 9

The RST block is sent to the other end of a connection to abort the connection. The RST block contains a header and may contain the reason parameters as an option. Data blocks and other Control blocks such as SYN and FIN are not permitted to be bundled with RST. If an endpoint receivers a RST for a connection that does not exist or a wrong format, it must discard it without resetting the connection. The receiver can decide if it shall respond to the sender a RST when receives a RST.

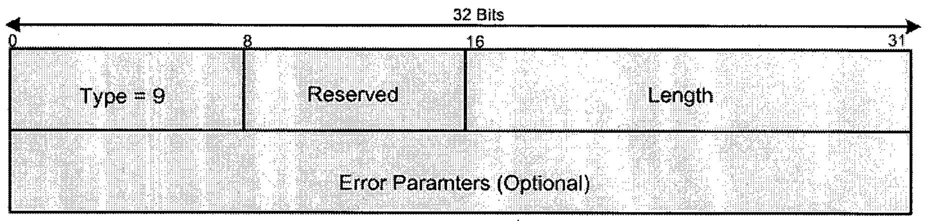

Figure 3.12: RST Format

Reserved: 8 bits This field is reserved for future usage. It can be used to indicate what affection this Error block has on the sender.

Length: 16 bits unsigned integer It is set to the size of the block in bytes, including the block header and all the error parameters.

3.2.11 ERROR Type 10

Error parameter(s) is/are appended to the RST block as an indication of the cause of certain error condition. Each RST may contain zero, one or more than one ERROR parameters. An ERROR is not always considered fatal of itself, but it may be used to report a fatal condition. ERROR uses the exactly same block format as RST, but Type is 10 and the Reserved field is replaced by Flags. Also, ERROR block contains at least one Error Cause.

Flags: 8 bits It is set to 0 by the sender and ignored on receipt.

Length: 16 bits unsigned integer It is set to the size of the block in bytes. It includes the block header and all the Error Cause fields present.

Error Causes: Variable length Error Cause field uses variable-length block VEXT (Variable Extension) as described in Figure 3.6 in section 3.2.2. The first two bytes of Error Cause block is Error Code field. The second two bytes is Cause Length field. Other following bytes contain Specific Cause Information defined in Table 3.3

Error Code: 16 bits unsigned integer It defines the type of error conditions being specified.

Cause Length: 16 bits unsigned integer It counts the size of Error parameters in bytes. It includes the Error code, Cause Length, and Specific Cause Information.

Specific Cause Information: This field contains the specific information of this Error block. See Table 3.3 for the definition of each error condition.

3.2.12 PROBE Type 11

This block is sent by one endpoint to probe the reachability of a particular destination address that is defined in the connection setup.

Flags: 8 bits It is set to 0 by the sender and ignored by the receiver.

Length: 16 bits unsigned integer It is set to the size of the block in bytes. It includes the block header and the optional Probe Parameter field.

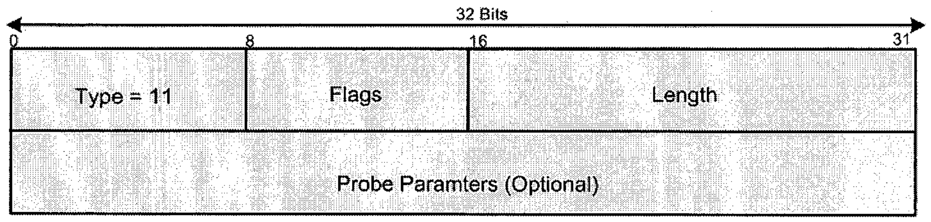

Figure 3.13: PROBE Format

Probe Parameters: variable length This field defines a variable-length parameter using the VEXT format. This field contains the time when this Probe is sent and the destination IP address that this Probe is sent to.

3.2.13 PROBE ACK Type 12

Once an endpoint receives the Probe block, it acknowledges the sender with this block. The PROBE ACK block must be sent to the source IP address of the PROBE block's sender. This block uses the same block format as PROBE. The parameter field should contain the time when this PROBE-ACK is sent and its own IP address.

3.3 MMSP State Transition

This section describes the transport process within the three main states of a MMSP connection: connection setup, data transmission and connection close.

3.3.1 Connection Setup

Both MMSP and TCP are designed to exchange messages to establish an end-to-end connection. However, the way MMSP used to set up a connection and delivery data is different from TCP. Traditional TCP performs a three-way handshake to establish a connection, whereas MMSP uses a four-way handshake by involving a state INFO to help protect from DoS attacks. The DoS attack uses the SYN flood to tie up resources on the server machine, so that it is unable to respond to legitimate clients' requests. It is accomplished by having the hacker's client discard the returning SYN-ACK message from the server and not send the final ACK. This results in the server retaining the partial state that was allocated by the initial hacker's SYN [17].

MMSP protects against this attack by importing the concept of INFO as described in section 3.2.3. The INFO block is bundled with the SYN-ACK from the server to the client. The server will not allocate a TCB (transmission control block) until it receives the INFO sent back from the client. Since the server only derives a TCB for the connection from an INFO-PING, it becomes more defensible to denial of service attacks [17].

To achieve an improved security, MMSP introduces a 4-way handshake. See Figure 3.14 for the 4-way handshaking and Figure 3.15 for its state transition diagram. The endpoint that initiates the connection setup is called the client and the other endpoint is called the server. The whole mechanism is described below:

1. Firstly, both the server and client are in the CLOSED state. The server is prepared to accept an incoming MMSP connection request.

This preparation is normally set up by calling socket, bind, and listen. On server side, we call it passive open.

2. The client initiates an active open by sending a SYN request to the server to set up a connection. The client builds up a transmission control block (TCB) that contains all necessary parameters for the maintenance and manipulation of the connection [20]. The SYN message provides the server with all the necessary information of the client, such as a list of IP addresses of the client, the initial sequence number, CID to identify all packets in this connection, number of outgoing streams the client is requesting, and the number of incoming streams the client can support.

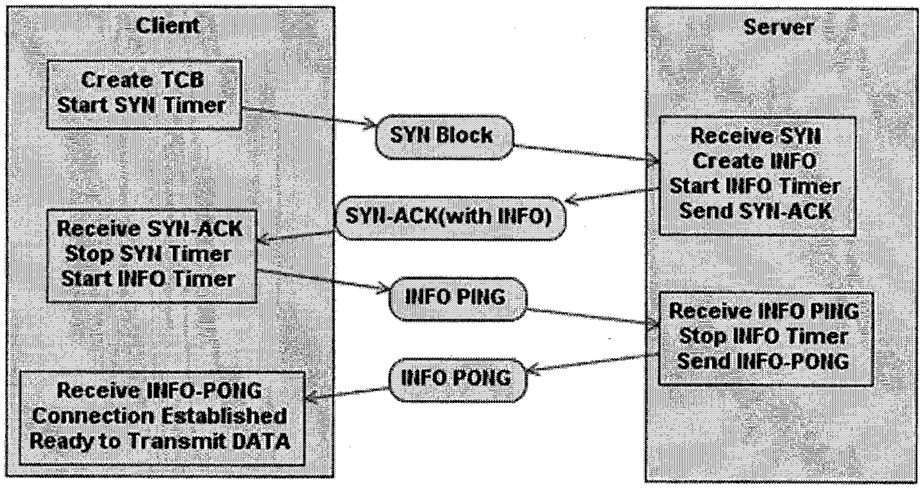

Figure 3.14: Four Way Handshaking

3. The server processes the SYN request. If it wishes to accept the connection, it generates a temporary TCB. This TCB contains a minimal

subset of information to recreate the TCB [4]. The TCB's creation time is set to the current time. The TCB is further packaged up and sent back to the client in a SYN-ACK packet. Besides the basic information such as a timestamp on when the INFO is created and the lifespan of the State INFO as indicated in section 3.2.3, the creator of the INFO should include MAC (Message Authentication Code) for message validation [9]. Once the SYN-ACK is delivered, the server deletes all information associated with the temporary TCB and goes back to the CLOSED state.

4. After receiving the SYN-ACK block, the client stops SYN Timer. Then it pings the INFO back to the server within the INFO-PING block. The client then enters the INFO-PINGED state and starts the INFO timer. DATA blocks may be also bundled in this packet.

5. When the server receives the INFO from the client, it checks the validity of the INFO. Then server side recreates the TCB from the information contained in the INFO. Only on this time, that server actually assigns its resources to the connection and enters ESTABLISHED state. Then the server PONGs the INFO back to the client.

6. On the receipt of INFO-PONG, the client enters the ESTABLISHED state. The whole MMSP connection is completed.

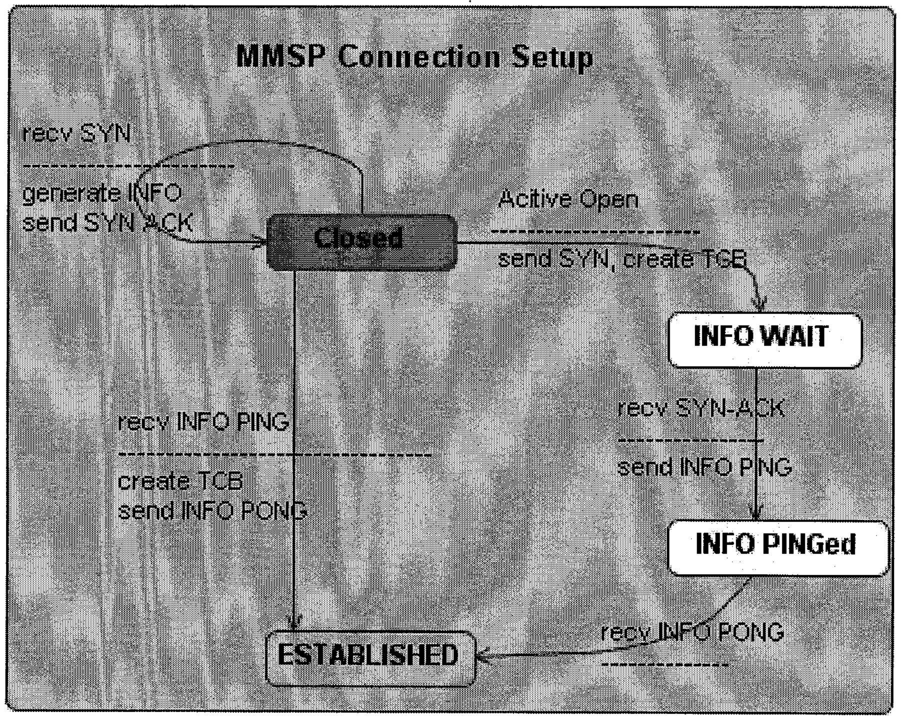

Figure 3.15: Connection Setup State Diagram

3.3.2 Data Transmission

Two types of blocks are transmitted throughout the data transmission process: DATA block and PROBE block. DATA blocks carry the actual data between the client and server; PROBE blocks are exchanged between the nodes to test the connectivity of the endpoints to preserve the validity of the data transmission, at regular-time intervals triggered by the Probe Timer.

Three steps happen in Data Transmission Process:

1. DATA blocks are exchanged between peers.

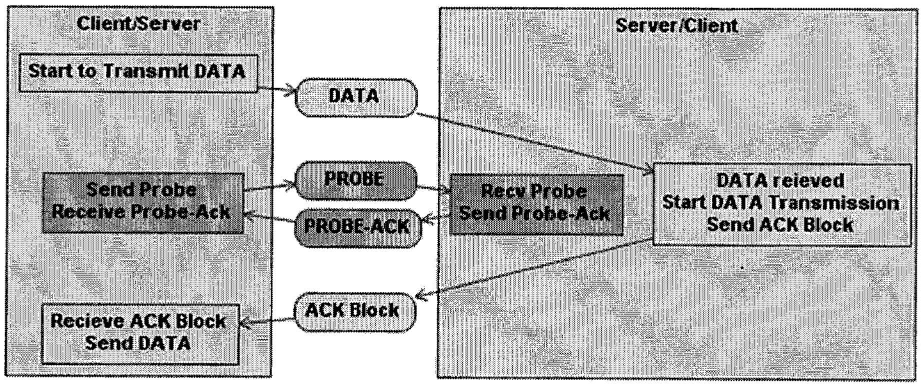

Figure 3.16: Data Transmission

2. After each DATA block is received, the receiving endpoint returns an ACK block to the sender.

3. The data exchange continues until the endpoints initiate the CLOSE request.

MMSP still defines an end-to-end window based flow and congestion control mechanisms similar to TCP. The receiver uses a 32-bit window size in SACK to inform the other endpoint an expected sending rate. The sender itself maintains a congestion window to control number of outstanding bytes that can be sent before they are acknowledged. The SACK is used to acknowledge each DATA block that has been received. The receiver adds a cumulative TSEQ value in each SACK to indicate all previously received blocks. Sometimes, DATA blocks are lost during the transmission, and then SACK specifies a sequence of missed blocks and responds back to the sender for retransmission. But if SACK is also lost, all those unacknowledged data blocks will still be retransmitted after the transmission timer expired on the sender's side.

3.3.3 Connection Termination

Figure 3.17 and Figure 3.18 show the MMSP Shutdown diagram and its state machine. MMSP does not support the "half-closed" state presented in TCP, where one endpoint stays open while the other endpoint closes [15]. MMSP is not designed to maintain the "half-closed" state since very few applications require it. MMSP only allows full-closed: when Close is initiated by one endpoint, the other endpoint must finish sending outstanding packets and continue the Close forwards. The MMSP connection is fully completed after both initial Close request and its acknowledgement have been responded by their respective receivers. The close request can be initiated by any of the two nodes. Figure 3.18 illustrates an example with the Close request sent from the server. The process of Close can be described in three steps:

1. The server sends a FIN request to the client and starts the Close timer.
2. The client acknowledges the receipt of the FIN block by generating of FIN-ACK block, and sent the FIN-ACK back to the server.
3. Server receives the FIN-ACK and responds by stopping its Close timer and deleting its TCB. Then server creates a FIN-DONE block and sends it back to the client. Once the client receives the FIN-DONE, the client side completes the full close of this MMSP connection.

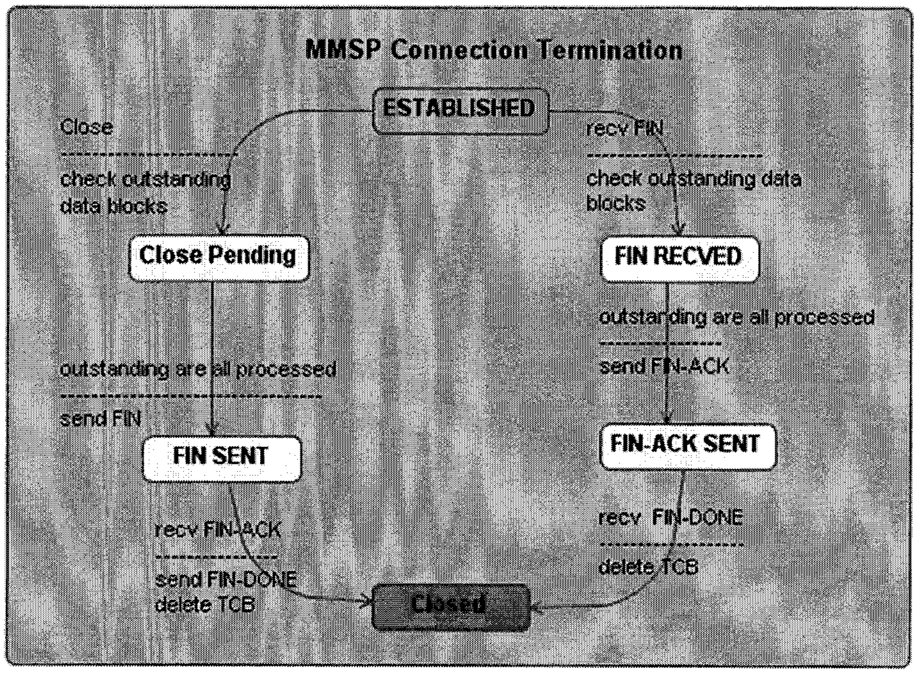

Figure 3.17: Connection Termination State Diagram

3.3.4 Path Selection

An important concept of MMSP middleware is that multiple paths coexist in one MMSP connection. A node can be reached through different IP addresses. Paths of endpoints are initially informed to each peer within the SYN block. Therefore, the client only needs to know one IP address of the server because the all other available IP addresses supported by the server are delivered within the SYN-ACK block. Currently, MMSP is only designed to handle IPv4 addresses.

The endpoint monitors all transmission paths to its peer in a MMSP connection by periodically sending PROBE packets over all paths, even in-

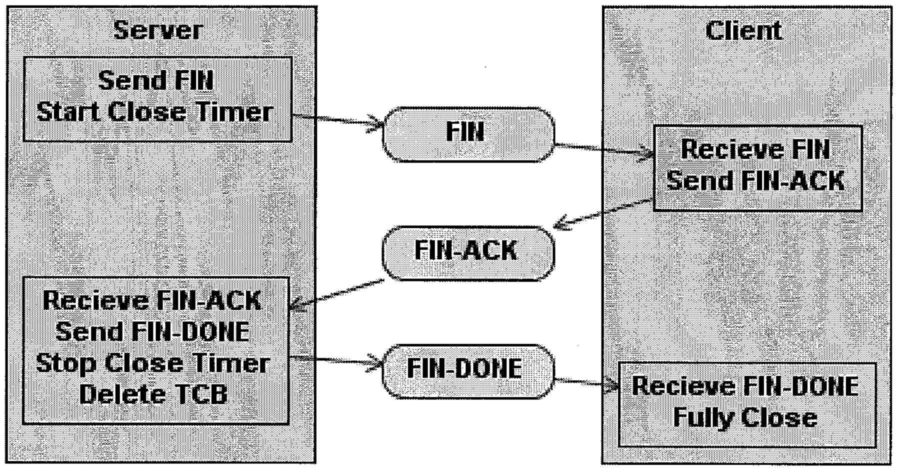

Figure 3.18: Connection Shutdown

cluding those paths that are not used for transmission of data blocks. Each PROBE block must be acknowledged by a Probe-ACK block. If the number of unacknowledged PROBEs reaches up to three (this value may be configurable), MMSP shall presume this path is unavailable and put an unreachable mark on this path in its data structure. When a PROBE block is acknowledged again, this path is marked to be reachable. If a connection only has one path and this main path becomes unreachable, the connection has to be terminated. If all the paths becomes unreachable, MMSP uses the last unreachable path to set the out-of-service timer (normally we set this timer to 5 seconds for experiment). Once the timer on the latest unreachable path is expired, the connection has to be shutdown.

Because MMSP is designed to maintain a list of network interfaces, one of the IP addresses is selected as the primary path. All data blocks are transmitted over this path by default. However, for retransmission, another active path is selected. In our design, all addresses are listed in an IP address priority queue, in which path interfaces are ordered by their bandwidth from high to low. Currently, we designed to select an available path that has second fastest bandwidth for data retransmission. If the primary path is down permanently, MMSP may either automatically select a path that has the second fastest bandwidth as a new primary path or return a status update to its user and let the user choose another path as the primary path. In terms of measurement of the round trip time of each path, Probe and Probe-ACK blocks are used in calculation.

3.3.5 Flow Control and Congestion Control

TCP uses sequence number to detect packet loss and duplication. MMSP also keeps this functionality by numbering all data blocks with the TSEQ number. Retransmissions are timer controlled. Whenever a retransmission timer expires, all unacknowledged data blocks shall be retransmitted. And this timer is restarted and double its time-up period. This is similar to TCP. The receiver will acknowledge each MMSP packet by sending a SACK. Sometimes, if some segments of data blocks arrive with some data blocks missed in the middle, the responding SACK should report all the lost blocks. Whenever three consecutive SACKs report the same data block hole, the receiver of these SACKS shall immediately retransmit the reported data blocks. We call this Fast Retransmit [3]. Each MMSP endpoint maintains a data buffer with a specific window size. The flow control of MMSP is similar to the TCP. The receiver can control the transmission rate by advertising its window size in the SACK block. The sender also maintains a variable to specify how many outstanding packets shall be sent before they are acknowledged. We call it congestion window size. The sender's congestion widow size must be less than or equal to receiver's window size. The congestion control of MMSP is derived from TCP congestion control such as slow start and congestion avoidance, as described in RFC 2581. Instead of one flow and congestion control for one TCP connection, MMSP has a discrete set of flow and congestions control on each available path.

3.4 Implementation

As a middleware, MMSP can be implemented on many Operating Systems. It can be added as a further extension of the different OS kernels. In our research, we implemented our MMSP on Windows Operating system and used Winsock API to access network functions. MMSP is designed within 14 modules, as shown in Figure 3.19. The module MMSP_control is the interface provided for the upper layer protocol. This module also controls the connection setup and termination. It receives the primitives from the upper layers via message Dispatcher module or the peer via recvBundler. In response to these input-signals, MMSP_control sends control_primitives to the upper layers via message dispatcher or the peer via sendBunlder. This module also contains the state of a connection. The whole MMSP domain diagram is presented by Fiure 3.19.

All other modules' functionalities are listed below:

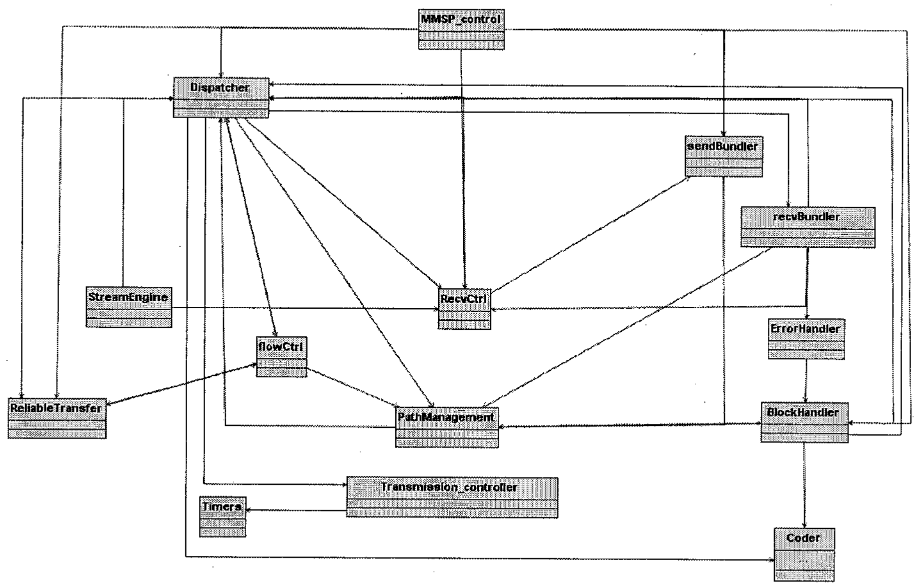

Figure 3.19: MMSP Domain Diagram

Dispatcher: This module holds a private list of streams. Since a MMSP instance usually has more than one stream, the purpose of this module is the distribution of signals from the upper layer and from the peer via the socket to the addressed endpoints. Signals from the WinSocket interface are forwarded to the recvBundler module.

SendBundler: This module bundles blocks to be sent into UDP datagrams. Blocks are accepted with the putBlock function until sendBlock is called, which causes the transmission of all blocks accumulated so far.

RecvBundler: This module deals with received blocks on receipt of MMSP data from MMSP message Dispatcher. All received MMSP data are disassembled into blocks. Depending on the block type, the blocks are distributed to MMSP_control, RecvCtrl, PathManagement or ReliableTransfer module respectively.

RecvCtrl: This module creates SACK data structures that are used to acknowledge received and lost data to the connection peer.

FlowCtrl: This module implements most parts of the flow control mechanisms.

StreamEngine: This module holds a list of streams to process sending and receiving data blocks.

ReliableTransfer: This module implements the retransmission mechanism. It stores all data that have not been acknowledged by the peer.

PathManagerment: This module does fault management for paths.

ErrorHandler: This module decodes error blocks and handles them respectively.

BlockHandler: This module provides functions for assembling and disassembling control blocks. When a block is created, the caller gets a block-ID, with which it can address the block in the other following calls. In order to handle more than one block at the same time, pointers to blocks are stored in an array of length MAX“BLOCKS. This is required for modules that receive a block and want to respond by sending another block. A block should be deleted after a signal is handled. All the blocks are in host byte order.

Coder: This module is borrowed from the Md5 message digest algorithm. This document takes as input a message of arbitrary length and produces as output a 128-bit fingerprint of the input. This algorithm is designed for digital signature applications, where a large file must be compressed in a secure manner before being encrypted with a secret key under a public-key cryptosystem such as RSA [16]. MMSP uses this module to assign a signature to blocks, such as PROBE and INFO blocks.

Timers: This module implements a linked list of timer events.

Error Code	Reason
0001	Wrong Stream ID: It indicates the received DATA block is delivered to a invalid or nonexistent stream.
0002	Unresolved Address: It indicates the type of address is not supported by the sender. For example, end points that only has IPV4 stack is not able to resolve the IPV6 address.
0003	Unrecognized Block Type: This error means the receiver can not understand the unrecognized block in the upper bits of its Block Type.
0004	Invalid Field Value: When a field value is out of the defined domain, this error parameter is appended to the RST block and returned to the originator.
0005	Missing Mandatory Field: It indicates some mandatory fields are missing in a received SYN or SYN-ACK block.
0006	Unrecognized Block Field: This error is returned to the originator if the receiver can not recognize the one or more fields of the received block.
0007	No User Data: If an end point receives a DATA block with no user DATA, it shall return this error cause to the sender.
0008	No INFO: This error cause shall be returned to the originator if the received SYN-ACK or INFO-PING does not contain INFO.
0009	Stale INFO or Invalid INFO: It indicates the received INFO has expired. This error parameter has an EXPIRE field containing the time difference between the current time and the time INFO expired, in millisecond.
000A	Unrecognized Error: All other undefined errors are included in this set.

Table 3.3: Error Code Table

Chapter 4

Experimental Evaluation

In this chapter we describe the experiments carried out to compare our MMSP middleware with the Windows TCP module. Our experimental setup consists of two different hosts. Both of these two hosts have three interfaces to a local private network assigned by a D-link router; one interface is wired link and the other two interfaces are wireless links using 802.11g. This chapter is formed by two sections. Section one discusses the reliability of MMSP; Section two introduces the performance tests on MMSP and TCP.

To verify the MMSP reliability and functionality, we tested if a connection can be setup via one path, two paths and even three paths. Then we manually block one or two of three paths to verify if the MMSP connection is still maintained. In the last step, we block all available paths and to see if MMSP can be successfully terminated.

To compare the throughput between MMSP and TCP, we brought in a popular Network tool, called Iperf. Iperf is a commonly used network testing tool that can create TCP and UDP data streams and measure the throughput of a network that is carrying them. It is a modern tool for network performance measurement.

Section 2 contains two experiments. We modified Iperf to be adapted to our MMSP layer and get throughput results under the best-effort condition. Then we did another performance test with a concern of bad network connectivity. We configured a simulated noisy environment by adding a data corruption function on Iperf. We manually configured Iperf to send 100MB data from the client to the server and corrupted parts of sent-out data for the error-handling experiment. Then, the test data for both TCP and MMSP were gathered under this environment.

All experiments were executed several times to verify the consistency of our results.

4.1 Experiment One - Functionality and Reliability

Experiment 1 has two hosts (host A and host B) involved in a local private network. Each host has two wireless interfaces and one wired interface connected to a Wilress 802.11g router. Three different IP addresses are assigned to each host. A MMSP application with the client and server mode was implemented to separately run on host A with the client mode and on host B with the server mode. The MMSP application was programmed to use the wired path as the primary path and the other two wireless paths as the secondary paths for data retransmission.

The purpose of this experiment is to verify the corrent switching between multiple streams in one MMSP connection by enabling and disabling different paths. The Table 4.1 presents the testing results against six different cases. Firstly the client built up three paths with the server within one

Test Cases	MMSP Connectivity
All Three Paths Coexist	Successfully Connected
Primary Path Disconnected	Still Connected
Primary Path Reconnected	Still Connected
One of Secondary Paths Disconnected	Still Connected
Both Secondary Paths Disconnected	Still Connected
All Paths Disconnected	The connection successfully Shutdown

Table 4.1: Functionality Experiment

MMSP connection and began to transmit data through the primary path. Then we manually disconnected the Primary Path. On response, MMSP chose an idle path to transmit the data. Since MMSP keeps probing the availability of each link, after the primary link was recovered, MMSP was informed and switched back to the primary link. We then disconnected the other two paths to verify the connection is still maintained. Finally, we disconnected all three links. After timers of both server and client were expired, the MMSP instance on both hosts was deleted and their connection was closed.

4.2 Experiment Two - Throughput

This section introduces two throughput experiments that we did on TCP and our MMSP on the Windows XP platform. We brought in Iperf, a third party tool to accomplish these two experiments. Iperf is an open source application used to measure network performance. It allows the user to set various parameters for special testing purposes, such as network throughput, loss ratio, jitter and so forth. Iperf has a client and server mode, either

Number of Streams	Throughput in Mb/s	
	TCP	MMSP
1 Wired Link	174 Mb/s	171 Mb/s
1 Wireless Link	18 Mb/s	18 Mb/s
1 Wired Link + 1 Wireless Link	N/A	172 Mb/s
1 Wired Link + 2 Wireless Links	N/A	172 Mb/s
2 wireless Links	N/A	21 Mb/s

Table 4.2: Throughput Experiment 1

unidirectional or bidirectional [19].

The first experiment is used to evaluate the best-effort throughput between TCP and MMSP with different configurations of path; the secondary experiment is a throughput test under various error conditions.

In the first experiment, we setup an Iperf client on the host A and an Iperf server on the host B. See Table 4.2 for each Path configuration. Each row in this table is a test case. Case 1 and case 2 have only one stream as their primary path. To test the throughput, the client sent data as fast as possible to the server via TCP and MMSP respectively. We found that under the best-effort testing environment, the throughput of MMSP is a little slower than TCP. This may be caused by the fact that MMSP is not implemented in the OS kernel. Also, since MMSP is a little complicated than TCP, the added-in complexity may also result in some extra overhead that TCP does not have. In case 3 and case 4, extra paths are added into one MMSP connection. We found the throughput is improved a little bit since secondary links can be used to reduce the head-of-line block problem. However, in the above four tests, adding additional paths into MMSP does not directly present the advantages of multi-streaming in that data loss and retransmission are less than 1%. On the other hand, test case 5 presents a clear improvement of streaming through multiple paths when we used a relatively noisy wireless network as the primary path. The throughput reaches 21Mb/s, improved by 36% against the case 2 of TCP, which is only 18Mb/s.

The second experiment was to test the performance between TCP and MMSP in a noisy network configuration. We did two subsets of tests. We first let both MMSP and TCP use a wired link as their primary paths. Different from TCP, MMSP has extra 802.11g wireless links as its secondary path for retransmission. In the other subset of test, we chose a 802.11g wireless link as the primary link, but at this time we used wired links as the secondary paths for MMSP. In both of these two cases, we let the client send 100 MB data to the server and corrupted 1MB of every 10MB data that were sent.

Figure 4.1 presents the throughput between TCP and MMSP with respect of the increased number of error packets being delivered. This benchmark shows that the throughput of TCP and MMSP are decreased while the errors increased during the transmission. However, TCP's performance goes down much faster than MMSP. Since TCP uses one stream only, the large number of error occurrences results in a serious head-of-line block issue. Opposite to TCP, MMSP can use other streams to avoid head-of-line issue. For example, when 80% of the corrupted data happen, the MMSP's throughput becomes two times greater than TCP.

Figure 4.2 presents a clearer performance difference between MMSP and TCP. Different from previous one in Figure 4.1, MMSP uses the wireless

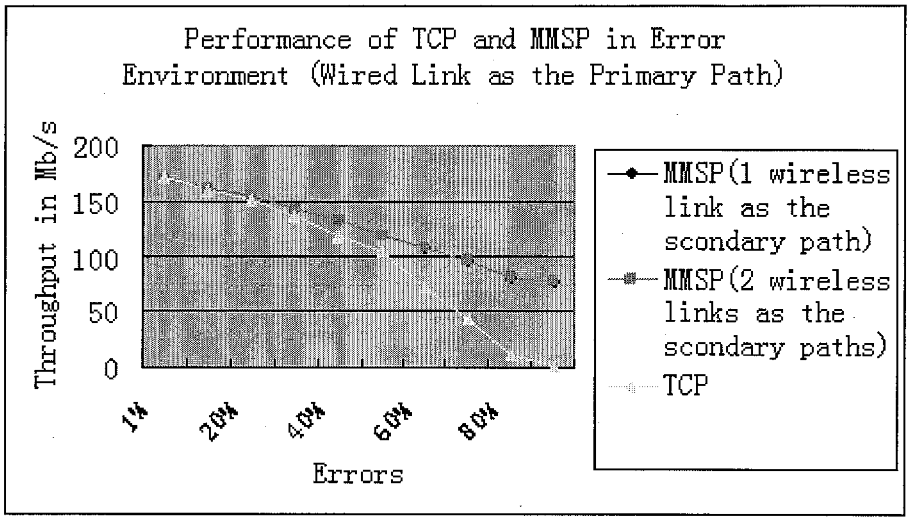

Figure 4.1: Throughput Experiment 2 Subset 1

link as the primary path and the wired links as secondary paths. TCP's throughput still dramatically goes down when transmission errors increase. However, because MMSP used the wired links as the retransmission paths, large portion of retransmission data could be delivered through the secondary path. Therefore, throughput is not affected too much by the data retransmission.

From the experiments, we also noticed that the throughput of MMSP is not changed by adding extra secondary paths. This is what we expected since our path selection algorithm was designed to only select a third path for retransmission until the already selected secondary path is unavailable. In other words, the third path keeps idle when the one of the secondary paths is still active for data retransmission.

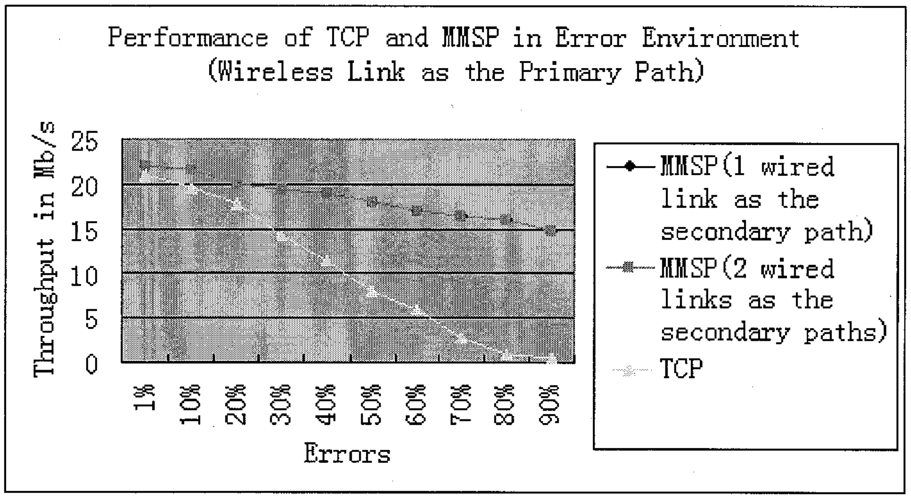

Figure 4.2: Throughput Experiment 2 Subset 2

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, we propose MMSP, a novel middleware as an extension of transport layer on IP stack to support multiple streaming via difference network interfaces within one connection. Using this architecture, mobile devices are able to utilize multiple access points to maintain the Internet connectivity. The important concept we introduced here is utilizing multiple streams in one connection. This avoids the characteristic drawbacks of TCP, such as head-of-line block issue and single stream limitations. In other words, MMSP is designed to organize and manipulate multiple TCP streams into one MMSP stream, but with more powerful functionality with respect of reliability, security and usability. To deploy this middleware onto current IP transport stack, we used the Windows platform and its winsocket API. A new MMSP state machine was designed to handle each transition condition. We reused and inherited some algorithms from TCP, such as the flow control and congestion control; we also redesigned the MMSP packet format and discussed the multi-path selection algorithm.

We showed that the proposed architecture has four advantages:

1. It provides better usability. It allows up-layer applications and users to select more than one links for data delivery instead of just using one link.

2. It is slick and powerful. It can keep a streaming connection alive with the outer network as long as one link is active. At this point, it proposes a very good blueprint for WiMax network coming in the short future. For example, users can keep a MMSP connection alive when walking from a WiMax covered area into a new WiMax area without worrying about the reassociation.

3. It is more reliable and secure. It uses new four-way handshake to avoid the DoS attacks.

4. It is more efficient. Since multiple paths are used into one connection, other paths can be used to deliver the retransmission messages when the primary path becomes indecent. The results of our evaluation demonstrate this protocol is a reasonable extension for the current IP stack, especially for the mobile devices which often change their access points.

5.2 Future Work

Our work has the potential to spawn a new body of research where other five features of MMSP can be investigated. First, our MMSP only supports

IPv4 address format. The need of IPv6 can be studied and can be added to MMSP to support for IPv6 networks.

Second, our MMSP was implemented on the windows platform sitting on the application layer. It uses UDP and Winsocket API as a simulation of directly using IP layer in the kernel. Since MMSP is not directly accessing the IP interface, the interaction between UDP and IP may result in some overhead. Evaluation will be more precise if MMSP could be integrated into the kernel of an Operating System, like Linux.

Third, our path selection algorithm is designed to choose the next available path for data retransmission from the path queue. It is the simplest way, but not the smartest way. This algorithm could be redesigned. For example, whenever retransmission happens, MMSP could be designed to choose the second fastest links as the data retransmission path by monitoring each path's capability.

Fourth, since MMSP is more complicated than TCP, the optimization of MMSP could be redesigned and simplified for resource limited devices, especially for some mobile phones with slow processors and low capacity of memory. The MMSP can be partially ported to the specific IP stack of different manufactures' devices and evaluated against their TCP's performance.

Fifth, in respect of the performance, lots of work be can investigated by using multiple streams to simultaneously send data instead of using just one primary link to deliver data.

Sixth, our current MMSP does not contain a minimum cost algorithm with the concern of path selection. Lacking of this algorithm may lead to unnecessary users' cost when a costly path is selected as the primary path while the original primary is dropped. In most cases, 3G or GPRS data are much more expensive than WiFi. A proper algorithm could be designed to notify users, ask them to select a cheap data service as the primary path or provide a list of options to users for primary path recommendations.

Bibliography

[1] Website:. `http://www.webopedia.com/TERM/T/TCP.html`.

[2] *3GPP and UMA System Engineeringy.* Mpirical Limited, second edition, 2006.

[3] M. Allman, V. Paxson, and W. Stevens. RFC2581: TCP Congestion Control. *Internet RFCs*, 1999.

[4] S. Bellovin, J. Ioannidis, A.D. Keromytis, and R.R. Stewart. On the use of stream control transmission protocol (SCTP) with IPSec. *IETF RFC3554*, 2003.

[5] K.C. Chen. MULTIPLE ACCESS FOR WIRELESS PACKET NETWORKS. *Multiaccess, Mobility and Teletraffic: Advances in Wireless Networks*, 1998.

[6] P. Deutsch. Aladdin Enterprises JL. *Gailly,ZLIB Compressed Data Format Specification version*, 3:1–10, 1996.

[7] T.F. Herbert. *The Linux TCP/IP Stack: Networking for Embedded Systems.* Charles River Media, 2004.

[8] H.Y. Hsieh and R. Sivakumar. A Transport Layer Approach for Achieving Aggregate Bandwidths on Multi-Homed Mobile Hosts. *Wireless Networks*, 11(1):99–114, 2005.

[9] H. Krawczyk and M. Bellare. R. Canetti," HMAC: Keyed-Hashing for Message Authentication. Technical report, RFC 2104, February 1997.

[10] R. Krishnappa. Image Compression Techniques and Video Streaming for Wireless Multimedia Communication. *Illinois Institute of Technology.*

[11] F. Lohan, I. Defée, and M. Vlad. The Architecture of an Integrated RTSP, RTP and SDP Library. *IEEE International Conference on Telecommunications*, pages 338–342, 2001.

[12] D. Maughan, M. Schertler, M. Schneider, and J. Turner. Internet Security Association and Key Management Protocol (ISAKMP). *Request for Comments (Proposed Standard)*, 2408, 1998.

[13] JC Mogul and SE Deering. RFC1191: Path MTU discovery. *Internet RFCs*, 1990.

[14] J. Postel. RFC 768: User Datagram Protocol, 1980. *URL http://www. ietf. org/rfc/rfc768. txt.*

[15] J. Postel et al. Transmission Control Protocol. 1981.

[16] R. Rivest. RFC1321: The MD5 Message-Digest Algorithm. *Internet RFCs*, 1992.

[17] C. Schuba, I. Krsul, M. Kuhn, E. Spafford, A. Sundaram, and D. Zamboni. Analysis of a Denial of Service Attack on TCP. *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, 223, 1997.

[18] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP), 1998.

[19] A. Tirumala and J. Ferguson. Iperf. *Avaliable from: http://dast. nlanr. net/Projects/Iperf.*

[20] J. Touch. TCP Control Block Interdependence. Technical report, RFC 2140, April 1997.