SCTPLIB 분석

<CPL-TR-06-02>

2006년 6월

통신프로토콜연구실

경북대학교

하종식(mugal1@cs.knu.ac.kr)

김동필(dpkim@cs.knu.ac.kr)

박재성(knucsid@hotmail.com)

윤성식(tothepolaris@cs.knu.ac.kr)

요 약 문

sctplib는 raw socket을 이용해 user level에서 구현한 sctp 프로토콜 구현코드이다. 본 문서에서는 rfc2960문서에 제안된 stream control transmission protocol을 어떻게 user level에서 sctplib로 구현되었는지를 분석하였다. 각각의 기능들을 class로 나누고, 데이터의 흐름을 중심으로 분석한 자료이기에 user level에서 sctp프로토콜의 분석 및 연결 지향형의 새로운 프로토콜의 구현시 도움이 되리라 생각된다.

Table of Contents

1. Introduction

sctplib는 raw socket을 이용해 user level에서 구현한 sctp stack이다. sctplib rfc2960에 정의 되어 있는 sctp의 특징들을 수렴하고 있고 각각의 sctp의 특징은 문서를 통해 참조 하길 바란다. 본 문서는 sctplib와 응용과의 관계 sctplib를 이용한 sctpapi와 응용과의 관계에서 함수 호출 순서에 따라 적용 되어지는 매커니즘을 분석 하였다. 본 문서에 사용된 코드는 sctplib 코드를 c++ 형식으로 약간의 수정을 하였으나 기본적인 전체 코드는 sctplib와 일치한다.

1.1 Code introduction

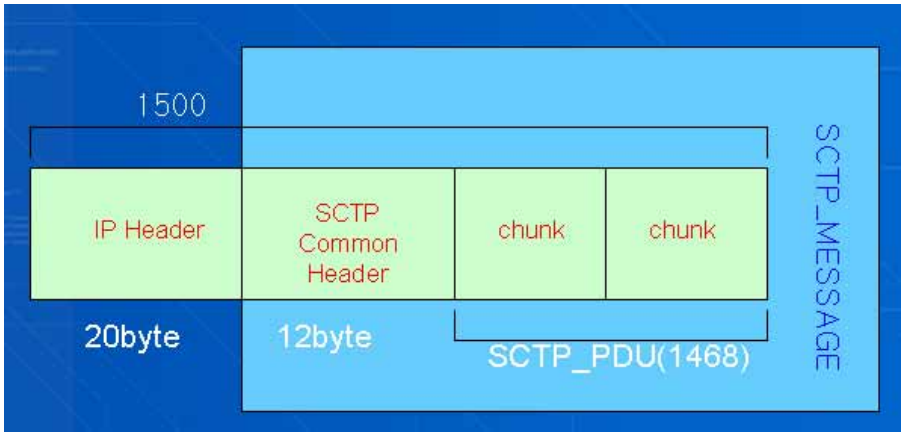
17개의 헤더 파일과 15개의 cpp파일이 있다. 각각의 파일의 특징을 뒷장에서 설명을 하도록 하겠다.

class	description
SSM_Adaptation	Ulptosctp layer
SSM_Auxiliary	데이터 무결성 검증
SSM_Bundling	Bundling 과 Bundling된 데이터 re-bundling
SSM_ChunkHandler	Make chunk and chunk 무결성 확인
SSM_Distribution	Packet process interface
SSM_Errorhandler	Error 처리 관련 루틴
SSML_Flowcontrol	흐름제어 관련, 전송시 cwnd, outbound packet, rcwnd 관리
SSM_Md5	Md5관련 처리
SSM_Pathmanagement	Multi-homing 관련 , heartbeat발생, cwnd 적용
SSM_Recvctrl	데이터 수신 관련 버퍼, tsn값만 가지고 처리, sack발생
SSM_Reltransfer	Sack수신시 처리 , 재전송, cwnd 적용,
SSM_SCTP_Control	Association state 관리

먼저 sctplib를 분석해 보도록 하자. Sctplib는 raw socket을 이용해서 sctp를 구현 하였다.

```
int sctpsock = socket(af, SOCK_RAW, IPPROTO_SCTP);
```

위의 코드는 sctp packet을 받아 들이기 위한 law소켓의 생성코드이다. Sctpsock로부터 받아 들이는 데이터는 ip패킷을 포함한 sctp데이터이다. 위와 같은 간단한 함수만으로도 sctp프로토콜을 관리할 수 있는 soket의 생성이 가능하다. 소켓 디스크립터 sctpsock을 통해 데이터를 받을 수 있는데 거기서 받아 오는 데이터는 ip 헤더를 포함한 sctp프로토콜 데이터이다.



데이터의 수락시 위의 그림에서 보는것과 같이 rawsocket을 통해 들어오는 데이터는 ip 헤더를 포함한 데이터이며 lib코드에서 ip헤더를 떼고, sctp_message단위로 데이터처리를 수행한다.

본 문서에서는 기본적인 sctp의 스펙을 알아보고 스펙에 따른 sctplib의 user level protocol 구현 방법을 논하려 한다.

1.2 Architecture View of SCTP

SCTP는 SCTP 사용자와 IP와 같은 비 연결 지향적인 패킷 network 서비스와의 사이에 있는 하나의 계층으로 보여진다. 이 문서의 이후 부분에서 SCTP가 IP의 위에서 동작한다는 것을 추측 할 수 있다. SCTP에 의해서 제공되는 기본적인 서비스는 SCTP의 User 들간 user message의 신뢰성 있는 전송이다. SCTP는 두개의 SCTP endpoint 시스템 사이에 있는 하나의 연결 환경에서 서비스를 수행한다.

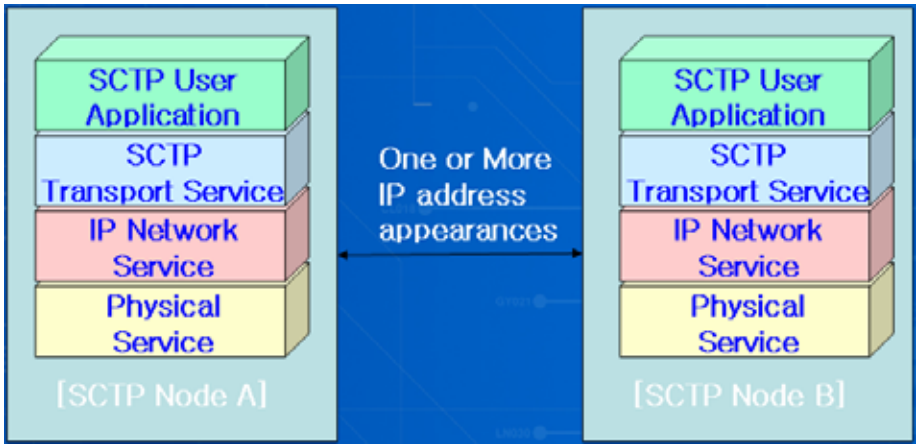


Figure 1: An SCTP Association

SCTP는 연결 지향적인 성향이 있다. 그러나 SCTP 연결은 TCP 연결보다 더 넓은 개념이다. SCTP는 endpoint가 SCTP 패킷을 보낼 수 있고 받을 수 있는 전송 주소(하나의 SCTP port와 조합 될 수 있는 여러 개의 IP Address)의 목록을 다른 endpoint(연결이 이루어지는 동안)에게 제공하기 위하여 각자의 endpoint들에게 수단을 제공한다. 그 연결의 범위는 각자의 endpoint의 리스트로부터 생성될 수 있는 가능한 source/destination 조합의 모두를 전송한다.

1.3. Functional View of SCTP

SCTP 전송 서비스는 여러 개의 기능으로 분해 할 수 있다. 이런 기능은 Figure-2와 같고 이 단락의 나머지 부분에서 설명한다.

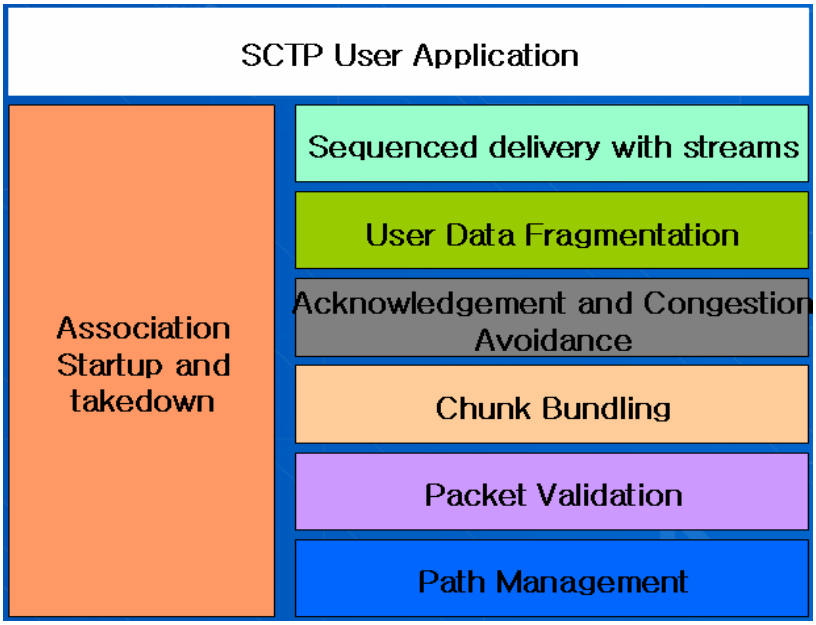


Figure 2 : Function View of the SCTP Transport Service

1.3.1 Association Startup and Takedown (SSM_Adaptation, SSM_SCTP_Control)

RFC 2522에서 Karn과 Simpson에 의해 기술된 메커니즘과 유사한 Cookie mechanism은 보안 공격에 대한 보호를 제공하기 위해 연결이 진행 되는 동안 사용되어진다. 이 Cookie mechanism은 four-way handshake를 사용하고 four-way handshake는 마지막 두 단계에서는 빠른 연결이 이루어지는 동안에는 데이터 전송이 가능하다.

SCTP는 한 endpoint가 데이터를 지속적으로 보내고 있고 반면에 다른 endpoint는 close 상태에 있는 half-open 상태를 지원 하지 않는다. 어떤 endpoint 가 SHUTDOWN을 수행하고 있을 때 각 peer에 있는 연결은 새로운 데이터를 user로부터 받는 것을 멈추게 되고 graceful close가 진행되는 동안에 queue안에 있는 데이터를 user에게 가져 온다.

1.3.2 Sequence Delivery within Stream(SSM_Pathmanagement)

SCTP안에서 사용되는 " stream" 이란 용어는 같은 스트림 안에서의 순서화 된 다른 메시지들이 upper-layer protocol에 전달 되어지도록 하는 사용자 메시지들의 순서와 관련 되어있다. 이것은 TCP에서 사용되어지는 byte 의 순서와 관련되어 있는 stream과 대조적이다. (이 문서에서는 1byte는 8bit로 가정)

SCTP user은 연결(Association) 내에서 지워 가능한 스트림의 수를 연결이 진행되는 시간에 규정 할 수 있다. 이 스트림의 수는 다른 endpoint 와 협상을 하게 된다. 내부적으로

SCTP는 SCTP 사용자에 의해 SCTP에 전달되는 각 메시지들에 stream sequence number을 부여한다. 수신측에서 SCTP는 메시지들이 주어진 스트림 내에서 순서적으로 SCTP 사용자에게 전달하는 것을 보장해야 한다. 하지만 하나의 스트림이 다음 순서의 user 메시지를 기다리면서 blocked 되어 있는 동안, 다른 스트림으로부터 메시지 전달은 처리될 수 있다.

1.3.3 User Data Fragmentation

SCTP가 단편화를 요구 할 때, path MTU(Maximum Transmission Unit)에 확인된 하부 계층에 SCTP 패킷의 전달을 보장하기 위하여 SCTP는 user message를 단편화 한다. 수신측에서 단편화는 SCTP user에 전달되기 전에 완전한 메시지로 재 조합된다.

1.3.4 Acknowledgement and Congestion Avoidance

SCTP는 단편화 되거나 단편화 되지 않은 메시지의 각 user 데이터에 TSN(Transmission Sequence Number)을 할당한다. 그 TSN은 스트림 레벨에서 할당된 SSN에 독립적이다. 데이터를 수신한 endpoint는 비록 순서에 갭이 발생 했다고 할지라고 수신한 모든 TSN을

알린다. 이러한 방법은 신뢰성 전송이 순서화된 스트림 전송으로부터 기능적으로 분리되어지게 한다. Acknowledgement와 Congestion avoidance 기능은 적당한 시간에 Acknowledgement 가 도착 하지 않을 때 패킷을 재전송 해야 하는 책임이 있다. 패킷 재전송은 TCP에서 사용되어 지는 유사한 혼잡제어에 의해서 조절되어진다.

1.3.5 Chunk Bundling (SSM_Bundling)

Section 3에서 설명하는 것처럼 SCTP 패킷은 한 개의 common header가 한 개 이상의 chunk들로 구성되어 하부 계층에 전달 되어 진다. 각 Chunk들은 user data나 SCTP 제어 정보를 가지고 있을 것이다. SCTP 사용자는 하나의 SCTP 패키 안에 하나 이상의 사용자 메시지를 번들링 요청 할 수 있는 옵션을 가지고 있다. SCTP의 Chunk bundling 기능은 완전한 SCTP 패킷을 조합 해야만 하고 수신 endpoint에서 재 조합 해야 한다.

SCTP는 혼잡이 있는 시간동안에는 비록 SCTP 사용자가 bundling 하지 않도록 요청했다 할지라도 bundling 을 수행한다. User 의 bundling 비활성은 전송전 짧은 시간동안에 delay가 될 수 있게 하는 SCTP 구현에 영향을 준다. User 계층에서 번들링 하는 것을 비활성 시켰다 하더라도 small delay는 혼잡시간이나 재전송 동안에는 번들링이 되는 것을 막는다.

1.3.6 Packet Validation (SSM_Auxiliary)

필수 항목인 Verification Tag 와 32bit Checksum (Adler-32 checksum 의 설명을 위해서는 Appendix B를 참조)는 SCTP Common header에 포함되어진다. Verification Tag 값은 연결이 이루어지는 동안에 연결의 각 endpoint에서 선택되어 진다. Verification Tag 값이 없이 패킷을 수신하는 경우는 패킷을 버린다. 이는 blind masquerade attacks에 대한 보호와 이전 연결로부터 쓸모 없는 SCTP 패킷을 수시하는 것을 막기 위한 차원이다. Adler-32 Checksum은 Network 상에서 data corruption 대한 추가적인 보호를 제공하기 위해 각 SCTP 패킷 의 송신자에 의해 설정되어져야만 한다. 유효하지 않은 Adler-32 checksum과 같이 수신한 패킷은 그 패킷을 폐기한다.

1.3.7 Path management(SSM_Flowcontrol)

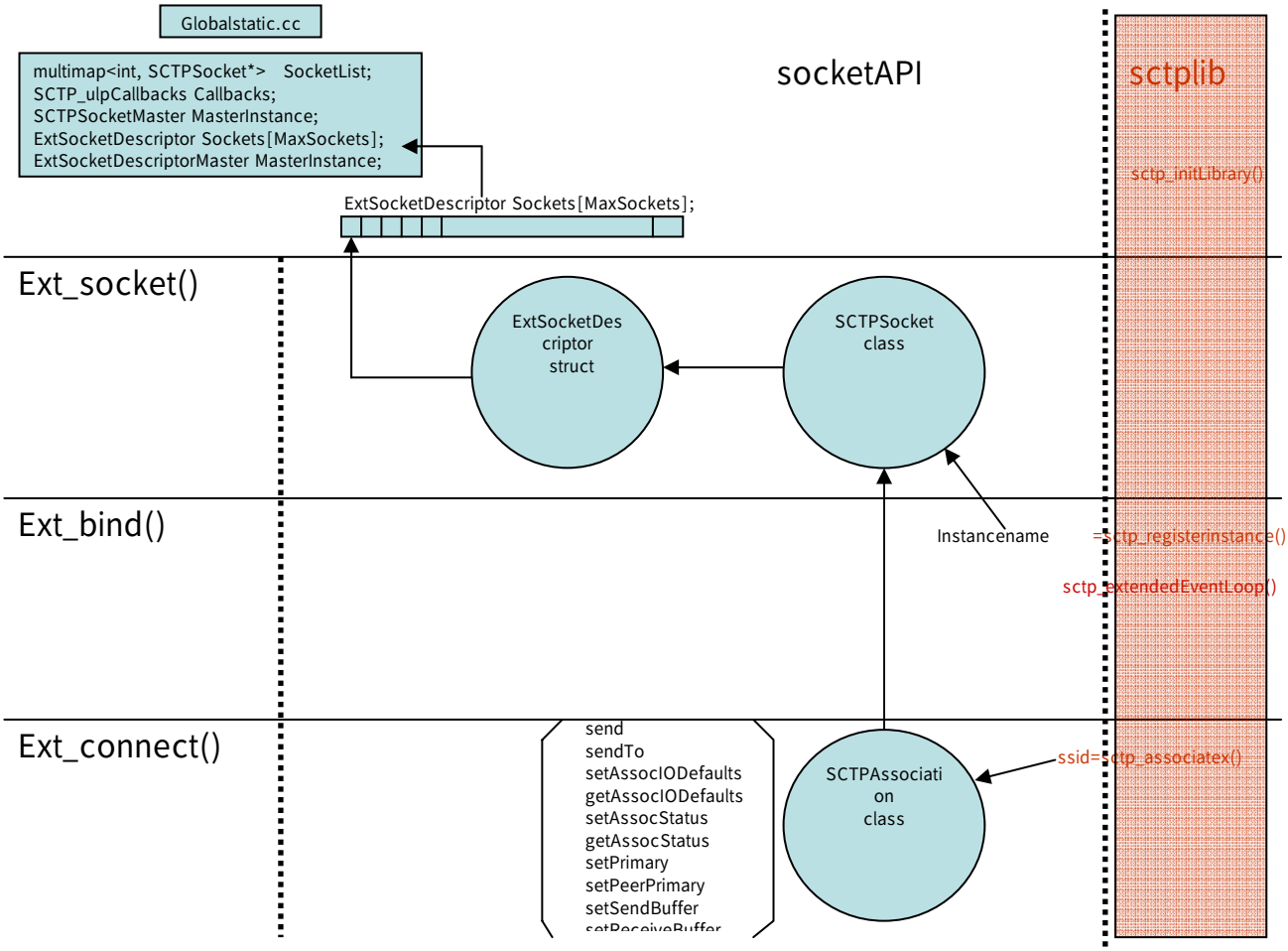
목적지로써 사용되는 전송주소를 조작 할 수 있다. SCTP 경로 관리 기능은 SCTP 사용자의 명령을 기반으로 한 나가는 SCTP 패킷을 위한 목적지 전송 주소로 선택하고 현재 파악된

목적지의 도착 여부의 상태를 확인하여 설정한다. 경로 관리기능은 다른 패킷 traffic이 정보를 제공하기에 적절하지 않게 제공 되고 멀리 떨어져 있는 endpoint의 변경된 전송주소가 도착이 가능 할 때 SCTP User에게 알릴 때 HEARTBEAT를 이용하여 도착 가능성을 모니터 한다.

경로 관리 기능은 로컬 전송주소의 적절한 설정을 보고기능과 멀리 있는 endpoint로부터 SCTP User에게 반환되는 전송주소를 보고해야 할 책임이 있다.
연결이 시작될 때 첫 번째 경로는 각 SCTP endpoint를 위하여 정의 되어지고 SCTP 패킷의 일반적인 전송을 위하여 사용되어 진다. 수신하고 있는 endpoint에 있어서 경로 관리는 상층에서 처리 하기 위해 패킷이 전달 되기 전에 들어오는 SCTP 패킷이 속해 있는 유효한 SCTP 연결의 존재에 대하여 유효성을 검사할 책임이 있다.

Note : 경로관리와 패킷 유효성 검사는 같은 시간에 실행 되어진다. 비록 위에서는 분리되어서 설명되었다 할지라도 실제적으로 유효성 검사와 경로 관리는 분리된 항목으로써 수행 되지 않는다.

1.4 SCTPLIB 구조도

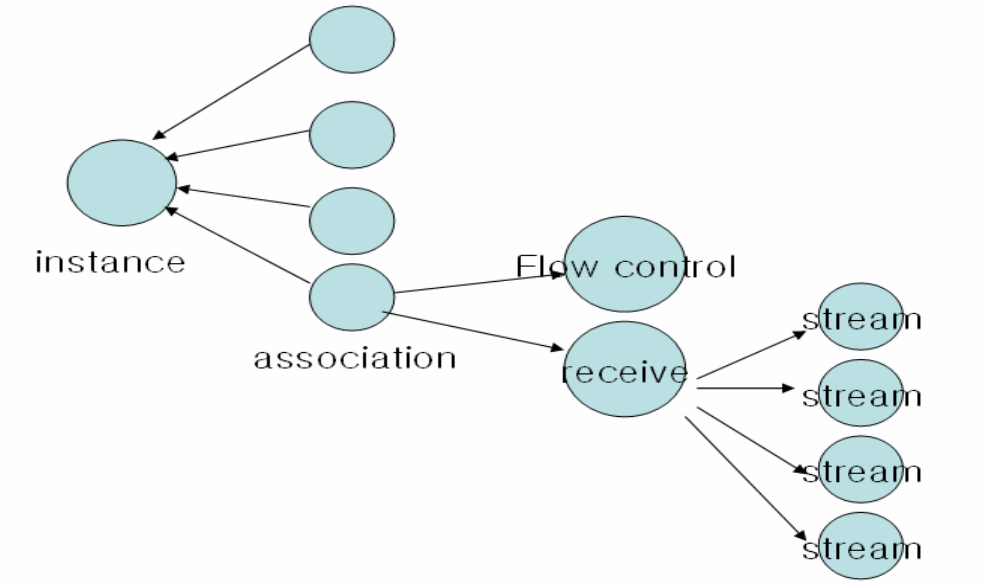


위의 그림은 sctplib에서의 응용에서의 기본적인 수행 루틴을 설명하고 있다 붉은색으로 표현된 부분이 실제적으로 호출되는 sctplib의 코드이다 왼쪽편에 있는 코드는 socketapi에서 수행되는 코드이며 위의 그림과 같이 api와 lib코드가 매핑 되어 진다. api코드는 추후 살펴 보도로 하고 우리가 관심을 두어 살펴볼 것은 sctplib쪽 코드이다.

호출 되는 기본적인 함수들은 다음과 같다.

Sctp_initlibary()
Sctp_registerinstance()
Sctp_eventloop()
Sctp_associate()

위의 네 가지 함수를 호출 함으로써 기본적인 sctp동작을 수행 할 수 있다.
먼저 sctplib에서는 raw socket을 이용하여 sctp프로토콜을 구현하였다고 하였다. Sctp_initlibary()함수에서 raw_socket 을 생성하게 되는데 전역적으로 생성이 되고 먼저 initlibary가 로딩이 되어있다면 위의 코드를 수행하지 않는다. 결론적으로 sctplib를 수행하기 위해서 한번만 호출 되는 함수이다. 위의 코드를 통해 생성된 raw socket 은 sctp 데이터를 받아들이는 interface 역할을 하게 된다. 이 raw socket을 통해 받아들인 데이터는 ip를 포함한 sctp데이터를 받아들인다. Sctp_initlibary()를 수행하면 기본적으로 sctp형태의 데이터를 받아들일 준비가 되어 있다고 볼 수 있다.
그 후 응용에서 Sctp_registerinstance()를 호출함으로써 실제적인 sctp socket 을 생성하게 된다. 여기서는 socket이라는 말보다는 instance라는 말로 응용프로그램 에서 사용을 할 수 있게 한다.



뒤에서 언급을 하겠지만, Sctp_registerinstance()를 호출하면 하나의 instance 구조체가 생성을 하게 되고 Sctp_associate()함수를 통해서 하나의 association 구조체를 생성을 하게 된다. 이 association구조체는 생성한 instance(socket)의 포인터를 가지고 있고 추후 association이 맺어 지게 된다면 4-way hand-shake 과정 중에 TCB를 생성하게 된다. TCB는 위에서 표현된 receive관련 구조체 flowcontrol과 관련된 구조체 들과 최종 association과 연결 맺어지는 in-stream buffer out-stream buffer 구조체의 생성을 일컬는다.

2. Interface with Upper Layer

Upper Layer Protocol(ULP)은 SCTP에 primitive를 전달하여 서비스를 요청하고 다양한 이벤트를 위해 SCTP로부터 notification을 수신한다.

2.1 ULP-to-SCTP

다음 섹션은 기능적으로 ULP/SCTP를 특성화 한다. 사용되어지는 표시는 상위 level 언어 안에 있는 모듈이나 함수의 호출과 유사하다. ULP primitive 아래에서 설명되는 함수들은 SCTP가 내부 프로세스 통신을 지원하기 위하여 수행 되어져야 하는 기본적인 함수를 이다. 개인적인 구현은 그들 소유의 형태에 맞게 구현되어야 하며 한번의 호출에서 기본적인 기능의 세부 항목이나 조합을 제공해야 할 것이다.

A) Initialize
Format : int
sctp_registerInstance(unsigned short port,
unsigned short noOfInStreams,
unsigned short noOfOutStreams,
unsigned int noOfLocalAddresses,
unsigned char localAddressList[][SCTP_MAX_IP_LEN],
SCTP_ulpCallbacks ULPcallbackFunctions)

이 primitive는 SCTP 가 내부적인 데이터 구조를 초기화 하고 SCTP의 운용환경을 설정하기 위한 필요한 자원을 할당하는데 이용되어진다.
일단 SCTP가 초기화가 되면 ULP는 이 primitive를 재호출 하는 것 없이 다른 endpoint함께 직접적으로 통신 할 수 있다. 첫번째 파라메터는 자신의 포트를 가르킨다 만약, 서버측이 아라니라 클라이언트 측이라면 RANDOM PORT를 사용하게 된다. 두번째,세번째 파라메터는 자신이 사용하고자하는 STREAM 숫자이다.이것은 ASSOCIATION설정시 END측과 협상을 통하여 설정 되게 된다.

B) Associate
Format : unsigned int sctp_associatex(unsigned int SCTP_InstanceName,
unsigned short noOfOutStreams,
unsigned char
destinationAddresses[SCTP_MAX_NUM_ADDRESSES][SCTP_MAX_IP_LEN],
unsigned int noOfDestinationAddresses,
unsigned int maxSimultaneousInits,
unsigned short destinationPort,
void* ulp_data)

이 primitive는 상위 계층이 특정에 peer endpoint에게 연결을 초기화 하도록 한다. Peer endpoint는 endpoint를 정의한 전송주소중의 하나에 의해서 규정할 것이다. 만약 local SCTP 인스턴스가 초기화 되지 않는 다면 ASSOCIATE는 에러로 간주된다. 하나의 연결 ID는 SCTP 연결에 대한 local 조정은 연결의 성공적인 성립에서 반환 될 것이다.
만약 SCTP가 endpoint와 함께 SCTP 연결을 시작할 없다면 에러를 반환 할 것이다.
다른 연결 파라미터들이 local endpoint의 나가는 스트림의 숫자뿐만 아니라 peer의 완전한 목적지 전송주소를 포함하여 반환 될 것이다.
반환되는 목적지 주소로부터 전송주소 중의 하나는 이 peer에게 SCTP 패킷을 보내기 위하여 기본 경로로서 local endpoint에 의해서 선택되어진다. 반환되는 목적지 전송주소

목록은 기본 주경로 변경을 위하여 ULP에 의해서 사용되어 질 수 있거나 강제적으로 전송 패킷을 특정 전송주소에 보내게 할 수 있다.

C) Shutdown
Format : int sctp_shutdown(unsigned int associationID)

우아하게 연결을 종료한다. 어떤 호스트의 큐에 저장된 사용자 데이터는 상대에게 전달되어야 할 것이다. 그 연결은 상대가 모든 전송된 SCTP 패킷의 수신을 알린 후에 종료될 것이다. 성공적인 코드는 연결의 성공적인 종료를 반환 할 것이다. 연결 종료 시도의 결과가 실패 했다면 에러코드가 반환 될 것이다.

E) Send
Format int sctp_send_private(unsigned int associationID,
unsigned short streamID,
unsigned char *buffer,
unsigned int length,
unsigned int protocolId,
short path_id, /* -1 for primary path, else address index to be taken */
void * context, /* SCTP_NO_CONTEXT */
unsigned int lifetime, /* 0xFFFFFFFF-> infinite, 0->no retransmit, else msecs */
int unorderedDelivery, /* use constants SCTP_ORDERED_DELIVERY, SCTP_UNORDERED_DELIVERY */
int dontBundle); /* use constants SCTP_BUNDLING_ENABLED, SCTP_BUNDLING_DISABLED */

SCTP을 통해서 사용자 데이타를 전송하기 위한 주요 함수이다.

Mandatory attributes :
Associated id SCTP 연결에 대한 local 조작
Buffer address(M) : 전송될 사용자 메시지가 저장된 곳
Byte count(M) : Bytes 단위의 사용자 데이타의 크기

Optional attributes
Context(Option) : 사용자 메시지의 전송이 실패했을 경우 ULP에게 실패 notification을 전송할때 포함되는 선택적 32-bit integer
stream id(Option) : 데이타 전송시 사용된 stream identification
life time(Option) : 사용자 메시지의 유효한 시기를 결정한다. life time이 만료되면 사용자 데이타는 SCTP에 전송되지 않을 것이다. 이 파라미터는 사용하지 못하는 사용자 메시지 전송을 위한 노력을 피하기 위하여 사용되어 질 수 있다. 만약 사용자 테이터를 메시지 생명시간 안에 전송하기(i.e. sent to the destination via SCTPs send primitive) 위하여 초기화 할 수 없다면 SCTP ULP에 통보한다. 그러나 만약 생명시간 만기 전에 chunk를 전송하기 위하여 시도 되었다면 사용자 데이터는 전송될 수 있을 것이다.

G) Receive
int sctp_receivefrom(unsigned int associationID,
unsigned short streamID,
unsigned char *buffer,
unsigned int *length,
unsigned short *streamSN,
unsigned int * tsn,
unsigned int *addressIndex,

unsigned int flags)

이 primitive는 만약 사용 할 수 있는 사용자 메시지가 있다면, SCTP에서 ULP에 의해 규정된 버퍼 안의 SCTP in queue 있는 첫번째 user message를 읽는다. length 에 있는 읽을 수 있는 메시지의 크기는 반환 될 것이다.
순서화 된 메시지를 위하여 메시지의 stream sequence number는 반환될 것이다.

association id (M) : SCTP 연결에 대한 local 조작
buffer address(M) : 수신된 메시지를 저장하고 있는 메모리 위치
buffer size(M) : 수신된 데이타의 최대 크기
stream id(Option) : data를 수신하기 위한 스트림을 가리킨다.
stream sequence number() : 송신측에 의해 할당된 SSN

P) Destroy SCTP instance
Format : int sctp_unregisterInstance(unsigned short instance_name)

local SCTP instance name : initial primitive안에 응용프로그램에게 전달 되는 값이고 그것은 소멸될 SCTP Instance를 가리킨다.

2.2 SCTP-to-ULP

Operating system이나 응용 환경은 SCTP에게 비동기적인 ULP 처리 신호를 위한 수단을 제공한다. SCTP가 ULP 처리 신호를 보낼 때 어떤 정보들이 ULP에게 전달된다.구현상 SCTP와 ULP간 interface는 separate socket나 error channel을 통해 구현할 수 있다.

IMPLEMENTATAION NOTE : 몇몇 경우에 이것은 분리된 소켓이나 에러 체널을 통해 수행된다.

A) DATA ARRIVE notification
void dataArriveNotif(unsigned int assocID, unsigned short streamID, unsigned int len,
unsigned short streamSN, unsigned int TSN, unsigned int protoID,
unsigned int unordered, void* ulpDataPtr)
SCTP 는 사용자 메시지가 성공적으로 수신되고 복구를 위한 준비가 되었을 때 이를 ULP에 통보한다.

다음은 부수적으로 notification 와 같이 전달 되는 것들이다.
association id (M) : SCTP 연결에 대한 local 조작
stream id : 이 것은 data 가 송신되는 스트림을 가리키는 값이다
B) SEND FAILURE notification
void sendFailureNotif(unsigned int assocID,
unsigned char *unsent_data, unsigned int dataLength, unsigned int *context, void* dummy)
만약 메시지가 전달 될 수 없다면 SCTP ULP에 이 notification을 호출 할 것이다.
다음은 부수적으로 notification 와 같이 전달 되는 것들이다.
association id (M) : SCTP 연결에 대한 local 조작
data retrieval id(Option) : 보내지지 않고 인지되지 않은 데이타를 검색하는데 사용된 id
cause code(Option) : 예로 메시지 사이즈가 너무 큰 경우, 메시지 life-time의 만료
context(Option) : 이 함수와 관련된 부가적인 정보

C) NETWORK STATUS CHANGE notification
void networkStatusChangeNotif(unsigned int assocID, short destAddrIndex, unsigned short newState, void* ulpDataPtr)

목적지 전송 주소가 비 활동이나 활동으로 표기 되었을 때 SCTP는 ULP에 이 notification을 호출 할 것이다.
다음은 부수적으로 notification 와 같이 전달 되는 것들이다.

association id (M) : SCTP 연결에 대한 local 조작
destination transport address : 변화에 의해 영향 받는 상대의 목적지 전송주소를 가리킨다.
new-status : 이것은 새로운 상태를 가리킨다.

D). COMMUNICATION UP notification
이 notification은 SCTP가 메시지를 보내거나 받기가 준비 되었을 때나 endpoint에 대한 잃어버린 연결을 복구하기 위해 사용되어진다.

void* communicationUpNotif(unsigned int assocID, int status,
unsigned int noOfDestinations,
unsigned short noOfInStreams, unsigned short noOfOutStreams,
int associationSupportsPRSCTP, void* dummy)

다음은 부수적으로 notification 와 같이 전달 되는 것들이다.

association id (M) : SCTP 연결에 대한 local 조작
status : 어떤 종류의 이벤트가 발생했는지를 나타냄
destination transport address list : 상대 전송주소의 완전한 설정
outbound stream count : stream의 최대 수는 ULP에서 이 연결을 사용하게 한다.
inbound stream count : 이 연결에서 상태 endpoint가 연결을 요청해왔던 스트림의수

E) COMMUNICATION LOST notification
SCTP가 endpoint에 대한 통신을 완전히 잃었을 경우나 상대가 비정상 종료를 진행 중에 있을 때 그것은 ULP에 이 notification을 호출 한다.
다음은 부수적으로 notification 와 같이 전달 되는 것들이다.
void communicationLostNotif(unsigned int assocID, unsigned short status, void* ulpDataPtr)

association id : SCTP 연결에 대한 local 조작
status : 이것은 발생한 이벤트의 타입이 무엇인지를 나타낸다. 즉 shutdown이나 abort 요청에 대한 응답에서 발생한 이벤트가 정상인지 fail인지를 나타냄
data retrieval id : 보내지지 않고 알려지지 않은 data 를 복구하기 위하여 이용되어지는 식별자
last-acked : peer endpoint에 의해 마지막으로 acked된 TSN
last-sent : peer endpoint에 마지막으로 보내진 TSN

3. SCTP Packet Format

각각의 패킷 구조체의 정의는 message.h안에 포함되어져 있다. SCTP packet은 SCTP의 protocol data units(PDU)로써, common header와 chunk들로 구성된다. 하나의 Chunk에는 제어정보 또는 유저 데이터를 포함한다. 다중 Chunk는 INIT, INIT ACK, SHUTDOWN COMPLETE chunk를 제외하고 하나의 SCTP packet에 MTU 크기까지 bundling 할 수 있다. 그러나, 하나의 패킷에 여러 개의 chunk를 추가해야만 하는 당위적인 것은 아니다. 또한, 하나의 패킷에 들어가지 않는 사용자 데이터 메시지는 6.9절(chunk bundle)에서 정의된 진행 절차를 이용해서 다중 chunk로 분해 가능하다.

SCTP 패킷에 사용되는 모든 integer 필드는 Network byte order(BIG Endian)으로 전송되어야만 한다.

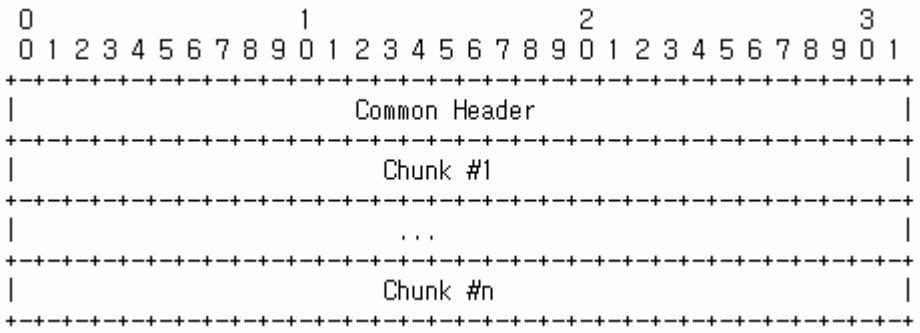
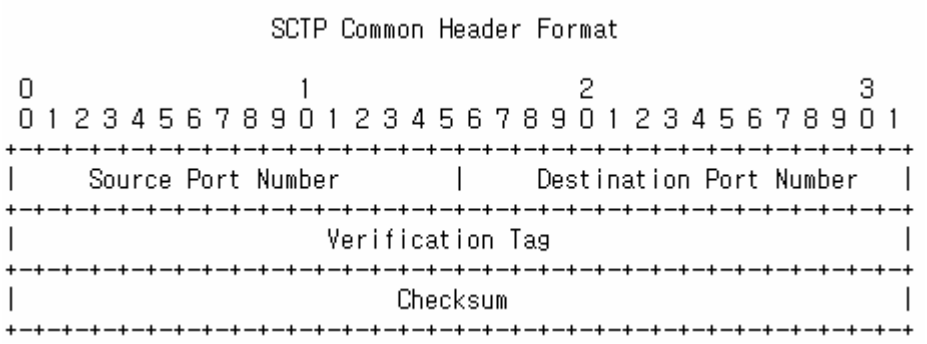


Figure 3 : An SCTP Packet format

3.1 SCTP Common Header Field Description

Note : common header는 12바이트로 구성되어 있으며, association의 확인을 위해 SCTP는 TCP와 UDP처럼 동일한 포트 개념을 사용한다. 전송에러를 검출하기위해, 각각의 SCTP 패킷은 TCP와 UDP의 16비트 checksum보다 강력한 32비트 checksum(Adler-32 algorithm)을 사용한다. 따라서, 실효성이 없는 checksum을 갖는 SCTP packet은 그냥 버려 버린다. Common header에는 32비트 값의 verification tag를 또한 포함한다. 이 검증 태그는 association 시작 시 endpoint들 사이에서 변환되어지는 association 특성이다. 따라서, 하나의 association에서 두개의 tag값이 사용된다.



----------message.h
```
typedef struct SCTP_COMMON_HEADER
{
    gushort src_port;
    gushort dest_port;
    guint32 verification_tag;
    guint32 checksum;
}
SCTP_common_header;
```
----------message.h

Source port Number : 16 bits (unsigned integer)
Source port Number는 SCTP송신자의 포트 번호이다. 포트번호는 수신자에 의해서 송신지 IP주소, SCTP 목적지 포트, 패킷이 속해있는 연결을 식별하기 위해 사용 가능한 목적지 IP 주소와 조합하여 사용되어 질 수 있다.

Destination Port Number : 16 bits (unsigned integer)
패킷이 도착되어지는 SCTP 포트 번호이다. 수신하는 host는 Destination Port Number를 이용하여 SCTP 패킷이 정확하게 수신 endpint나 애플리케이션으로 다중 송신 하도록 사용한다.

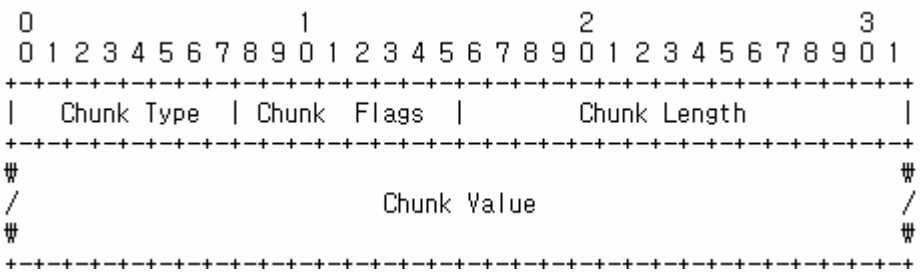
Verification Tag : 32bits (unsigned integer) Packet 수신자는 SCTP packet의 송신자를 검증하기위해 Verification Tag를 사용한다. 전송상에서 Verification Tag의 값은 association 초기화를 하는 동안에 peer endpoint로부터 수신된 Initiate Tag 값으로 설정 되어야만 한다. 다음은 예외 사항이다. INIT chunk를 포함하는 패킷은 zero Verification Tag를 갖아야만 한다.
T-bit가 설정된 SHUTDOWN-COMPLETE chunk를 포함하는 패킷은 SHUTDOWN-ACK chunk를 가진 패킷에서 Verification Tag 값을 복사해야만 한다. ABORT chunk를 포함하는 패킷은 ABORT를 보내도록 원인이 된 패킷에서 Verification Tag를 복사할 수 있다. SCTP packet 내에서 전송되는 INIT Chunk는 다른 Chunk 가 추가되지 않는 INIT chunk가 유일한 Chunk이어야만 한다.

Checksum : 32bits (unsigned integer)
이 필드는 SCTP 패킷의 checksum을 포함한다.

3.2 Chunk Field Descriptions

아래의 그림은 SCTP 패킷에 전송되어질 수 있는 chunk를 위한 필드들의 틀을 설명한다. 각각의 chunk는 chunk 타입, chunk-specific Flag, chunk length, value 부분으로

구성된다.



----------message.h

```
typedef struct SCTP_CHUNK_HEADER
{
    guint8 chunk_id;          /* e.g. CHUNK_DATA etc. */
    guint8 chunk_flags;       /* usually 0     */
    guint16 chunk_length;     /* sizeof(SCTP_chunk_header)+ number of bytes in the chunk */
}
SCTP_chunk_header;
```

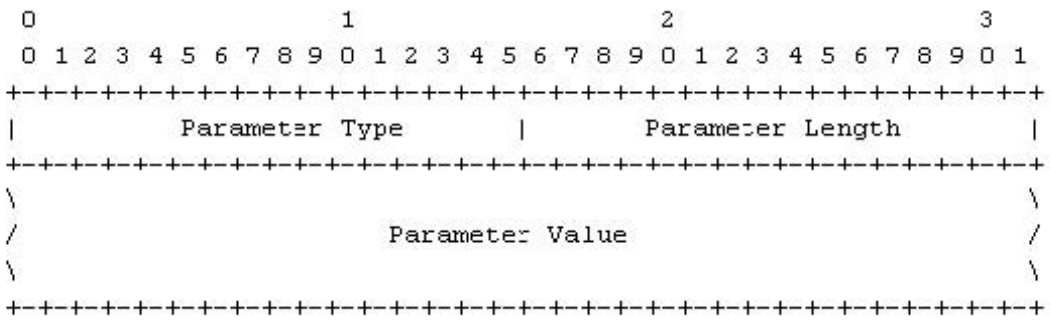
----------message.h

Chunk type : 8 bits (unsigned integer) Chunk type 필드는 Chunk Value field안에 포함되어 있는 정보의 Type을 식별한다. Chunk type 필드는 0 ~ 254의 값을 취하고 255는 향후의 사용을 위해 예약되었다. 다음 아래의 그림은 정의 되어진 Chunk Type 들이다.

ID Value	Chunk Type
0	- Payload Data (DATA)
1	- Initiation (INIT)
2	- Initiation Acknowledgement (INIT ACK)
3	- Selective Acknowledgement (SACK)
4	- Heartbeat Request (HEARTBEAT)
5	- Heartbeat Acknowledgement (HEARTBEAT ACK)
6	- Abort (ABORT)
7	- Shutdown (SHUTDOWN)
8	- Shutdown Acknowledgement (SHUTDOWN ACK)
9	- Operation Error (ERROR)
10	- State Cookie (COOKIE ECHO)
11	- Cookie Acknowledgement (COOKIE ACK)
12	- Reserved for Explicit Congestion Notification Echo (ECNE)
13	- Reserved for Congestion Window Reduced (CWR)
14	- Shutdown Complete (SHUTDOWN COMPLETE)
15 to 62	- reserved by IETF
63	- IETF-defined Chunk Extensions
64 to 126	- reserved by IETF
127	- IETF-defined Chunk Extensions
128 to 190	- reserved by IETF

3.2.1 Optional/Variable-length Parameter Format

SCTP control chunk의 chunk values는 zero 또는 그 이상의 parameters가 따라오는 요구된 필드의 chunk-type-specific 헤더로 구성된다. 따라서, 하나의 chunk 내에 포함되어 있는 optional/Variable-length parameter는 아래에서 보여지는 것처럼 Type-Length-Value 형태로 정의된다.



Chunk Parameter Type : 16 bits (unsigned integer)

Parameter의 type을 결정하는 값으로써 0 ~ 65534값을 갖는다. 65535는 IETF의 확장을 위해 예약되어있다. 특정 SCTP Chunk 설명 내에 정의된 것들 이외의 값은 IETF에 의해서 사용되어지기 위해 예약 되어진다.

Chunk Parameter Length : 16 bits (unsigned integer)

Chunk Parameter Length filed 는 Parameter Type, Parameter Length, Parameter Values를 포함하는 bytes 단위의 parameter의 크기이다. Parameter Values field 가 0의 길이를 가지는 파라미터는 4의 Length field를 가져야 한다. 파라미터 길이는 어떠한 Padding bytes도 포함하지 않는다.

Chunk Parameter Value : variable-length Chunk Parameter Value filed는 파라미터 안에서 실제 전달되는 정보를 포함 한다.

파라미터의 전체 길이(including Type, Parameter Length 와 Value field)는 4의 배수이어야만 한다. 만약 파라미터의 길이가 4의 배수가 아니면 송신자는 파라미터 끝에 0의 값을 가지는 것을 붙인다. 패딩의 길이는 파라미터 길이 필드 안에 포함하지 않는다. 송신자는 3bytes의 이상을 패드하지 않는다. 수신자는 패딩 바이트를 수신하면 무시해야만 한다.

Chunk Type은 Chunk를 처리하는 endpoint가 chunk type을 인식 못했다면 최상위 2비트에서 규정한 행동을 취하도록 기호화 된다.

00 SCTP packet 처리를 중지하고 버리고, 향후 처리하지 않음

01 SCTP packet 처리를 중지하고 버리고, 향후 처리하지 않음. ERROR 또는 INIT ACK에 Unrecognized Parameter Type 기록

10 현재의 chunk는 skip하고 계속해서 진행

11 현재의 chunk는 skip하고 계속해서 진행하지만 ERROR chunk에 Unrecognized Parameter Type 을 사용하도록 기록

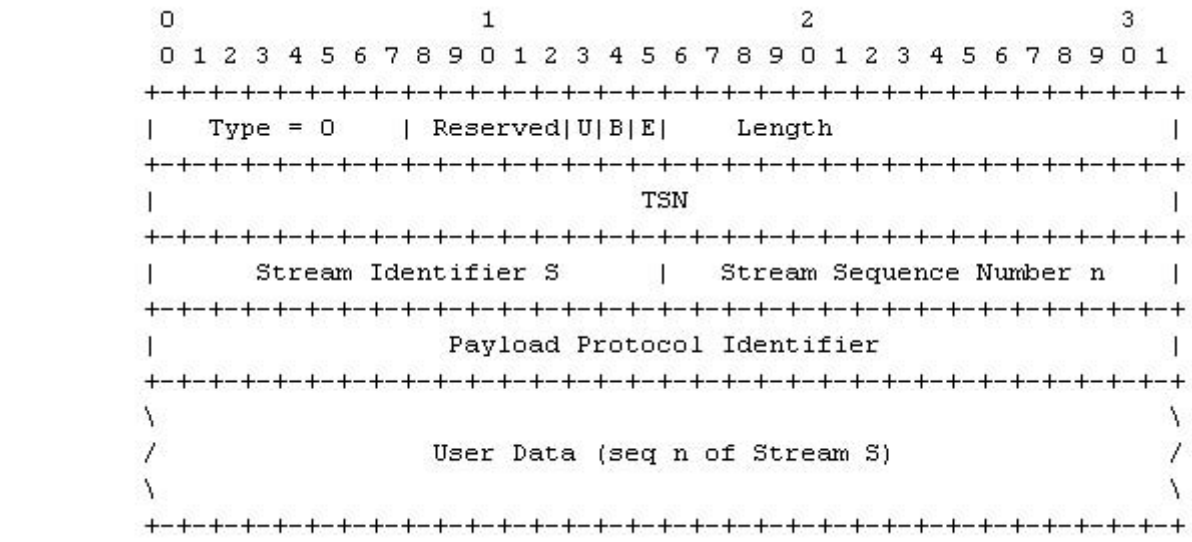
실제 SCTP 파라미터들은 특정 SCTP Chunk Section 안에 정의 되어져 있다.

3.3 SCTP Chunk Definitions

이 섹션에서는 서로 다른 SCTP Chunk 타입들의 포맷을 정의 한다.

3.3.1 Payload Data (DATA) (0)

Data chunk를 위해 사용되는 필드이다.



----------message.h

```
typedef struct SCTP_DATA_CHUNK
{
    guint8 chunk_id;
    guint8 chunk_flags;
    guint16 chunk_length;
    guint32 tsn;
    guint16 stream_id;
    guint16 stream_sn;
    guint32 protocolId;
    guchar data[MAX_DATACHUNK_PDU_LENGTH];
}
SCTP_data_chunk;
```

----------message.h

Reserved : 5 bits 모두 0으로 설정하고 수신측에서는 무시되어진다.

U bit : 1 bit (U)nordered bit로써 1 로 설정하면 unordered DATA chunk를 의미하고, 이 DATA chunk에는 Stream Sequence Number를 할당하지 않는다. 그러므로, 수신측에서는 Stream Sequence Number를 무시해야만 한다.

이후에 필요에 따라 재 조합하려고 한다면, unordered DATA chunk는 순서의 재 정렬 없이 수신자에 의해 상위측으로 보내져야만 한다. 만일, unordered user message가 분해되었다면, 메시지 각각의 조각은 U bit가 1로 설정되어야만 한다.

B bit : 1 bit (B)eginning fragment bit로써, 1로 설정되었다면 user message의 첫 조각을 가리킨다.

E bit : 1bit (E)nding fragment bit로써, 1로 설정되었다면 user message의 마지막 조각을 가리킨다.

Unfragmented user message는 B 와 E bit모두 1 로 설정되어야 한다. B와 E bit 모두 0 으로 설정 된 것은 multi-fragment된 user message의 중간 fragment를 의미한다.

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

Table 1: Fragment Description Flags

User message가 여러 chunk에 fragment 되었을 경우에는 메시지를 재조합 하기위해 수신측에서 TSNs를 사용한다. 이것은 하나의 fragment된 user message의 각각의 조각들에 대한 TSNs는 정확하게 순차적이다라는 것을 의미한다.

Length : 16 bits (unsigned integer)
Length field는 패딩되는 데이터를 배제하고 타입부분의 시작부터 user data 의 끝에 이르는 부분까지 해당하는 DATA chunk 의 byte 길이를 가리킨다. User data가 없는 하나의 DATA chunk의 길이는 16 이 된다.

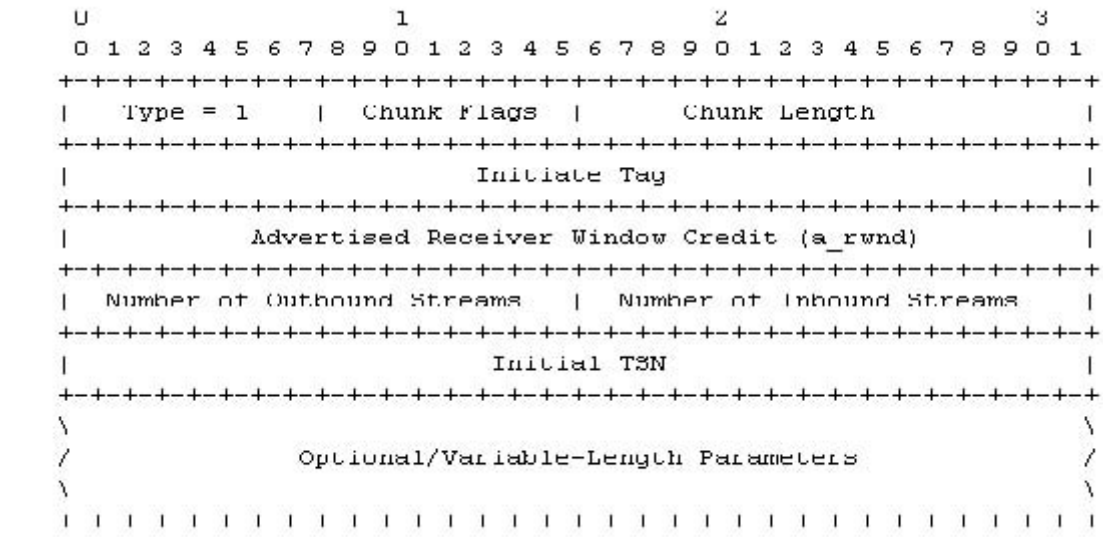
TSN : 32 bits (unsigned integer) TSN 는 DATA chunk에 대한 TSN(Transmission Sequence Number) 값으로써, 0 ~ 2**32-1 범위를 갖는다.

Stream Identifier S : 16 bits (unsigned integer) 연속되는 user data들이 속해 있는 스트림 stream 식별자

Stream Sequence Number n : 16 bits (unsigned integer) Stream S 내의 연속된 user data의 stream sequence number로써 0 ~ 65535의 범위를 갖는다. 전송을 위해 user message가 fragment된다면, 동일한 stream sequence number가 메시지의 fragment 각각에 같은 값이 전송되어야만 한다.

Payload Protocol Identifier : 32 bits (unsigned integer) Payload Protocol Identifier 는 Application 또는 upper layer가 규정된 프로토콜 식별자를 표현한 값으로써, 이 값은 upper layer에 의해 SCTP에 전달되고 상대 peer에 보내진다. 이 값은 SCTP에 의해 사용되는 것은 아니지만, DATA chunk내에 운반된 정보의 타입을 구별하기 위해 상대 application 뿐만 아니라 확실한 network 요소들에 의해 사용된다. 이 부분은 심지어 fragment된 DATA chunks에도 포함되어 보내져야 한다. (to make sure it is available for agents in the middle of the network) 0은 payload data를 위해 upper layer에 의해 기술된 application identifier가 없다는 것을 가리킨다.

3.3.2 Initiation (INIT) (1)



----------message.h

```
typedef struct SCTP_INIT_FIXED
{
    guint32 init_tag;
    guint32 rwnd;
    guint16 outbound_streams;
    guint16 inbound_streams;
    guint32 initial_tsn;
}
SCTP_init_fixed;

/* init chunk structure, also used for initAck */
typedef struct SCTP_INIT
{
    SCTP_chunk_header chunk_header;
    SCTP_init_fixed    init_fixed;
    guchar             variableParams[MAX_INIT_OPTIONS_LENGTH];
}
SCTP_init;
```

----------message.h

INIT Chunk는 두 개의 endpoints 사이의 SCTP 연결을 초기화 하기 위해 사용된다. INIT Chunk의 형태는 위와 같다. INIT chunk는 연속된 파라미터를 포함한다. 만약 그렇지 않으면 각각의 파라미터는 INIT chunk에 오직 한번만 포함되어야 한다는 것을 인지 해야 한다.

Fixed Parameters	Status
Initiate Tag	Mandatory
Advertised Receiver Window Credit	Mandatory
Number of Outbound Streams	Mandatory
Number of Inbound Streams	Mandatory
Initial TSN	Mandatory

Variable Parameters	Status	Type Value
IPv4 Address (Note 1)	Optional	5
IPv6 Address (Note 1)	Optional	6
Cookie Preservative	Optional	9
Reserved for ECN Capable (Note 2)	Optional	32768 (0x8000)
Host Name Address (Note 3)	Optional	11
Supported Address Types (Note 4)	Optional	12

Note 1. INIT chunk는 Ipv4와(또는) Ipv6을 갖는 multiple 주소가 포함된다.
Note 2. ECN(Explicit Congestion Notification) 가능 부분은 ECN의 사용을 위해 예약됨.
Note 3. INIT chunk는 하나의 호스트 이름 주소 파라미터를 포함해서는 안 된다. 더욱이, INIT 송신자는 INIT안에 호스트 이름을 갖는 또 다른 주소를 포함해서는 안된다. INIT 수신자는 수신된 INIT chunk에 호스트 이름 주소 파라미터가 있다면, 다른 주소는 무시해야만 한다.

Initiate Tag : 32 bits (unsigned integer) INIT의 수신자가 Initiate Tag parameter를 기록하고, 이 값은 association내에서 전송되는 INIT 수신자의 모든 SCTP packet의 Verification Tag 부분에 대치되어야만 한다.

Advertised Receiver Window Credit (a_rwnd) : 32 bits (unsigned integer) Bytes 수의 버퍼 공간으로써, INIT의 송신자는 association에 이 window를 확보한다. 버퍼공간은 association의 수명 동안 줄여서는 안될 것이다. 그러나 endpoint는 , SACK chunk 안에 보내어지는 a_rwnd의 값을 변경할 수 있다.

Number of Outbound Stream (OS) : 16 bits (unsigned integer) INIT chunk의 송신자가 association안에서 생성하기를 원하는 outbound stream 수를 정의 한다. 0의 값을 사용되어서는 안 된다. Note : 0 으로 설정된 OS를 갖는 INIT의 수신자는 association을 중지해야만 한다.

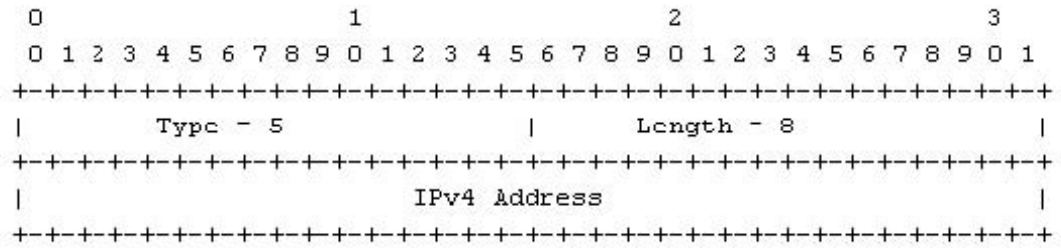
Number of Inbound Stream (MIS) : 16 bits (unsigned integer) INIT chunk의 송신자는 association 내에서 상대 endpoint가 생성할 수 있는 최대 stream의 최대 수를 정의 한다. 0 의 값은 사용되어서는 안 된다. Note : steam의 실제적인 수는 협상하지 않지만, 대신에 두 endpoints는 최소치(요청된 것, 제공된 것)를 사용할 것이다. (5.1.1 참조) Note : 0 의 MIS 값을 갖는 수신측은 association을 abort 할 것이다.

Initial TSN (I-TSN) : 32 bits (unsigned integer) Initial TSN은 송신자가 사용할 것이며, 감의 범위는 0 ~ 4294967295이다. 이 필드는 Initiate Tag 부분의 값으로 설정될 것이다.

3.3.2.1 Optional/Variable Length Parameters in INIT

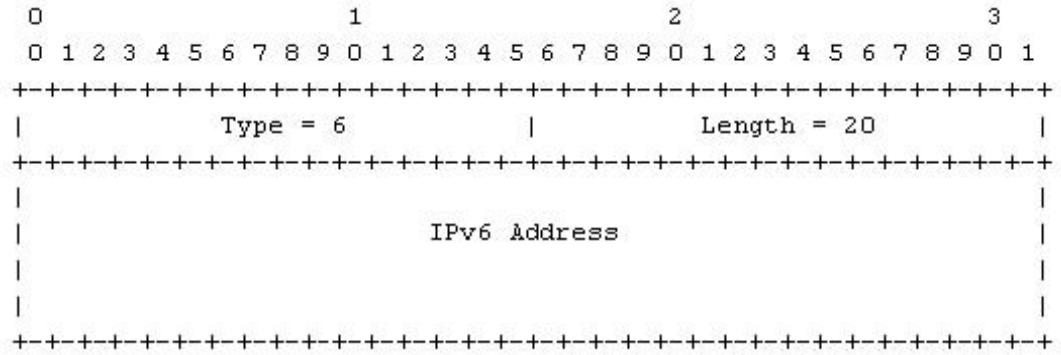
Parameter들은 3.2.1에서 정의한 것과 같이 Type-Length-Value의 형태로 정의된다. 모든 Type-Length-Value 필드는 앞선 section에서 정의한 고정 필드 후에 와야 한다.

IPv4 Address Parameter (5)



IPv4 Address : 32 bits (unsigned integer) 송신측 endpoint의 IPv4 주소를 포함하며, 이진 부호화된다.

IPv6 Address Parameter (6)



SCTP common header안의 Source Port Number를 조합되어 졌다면, 그 값은 association 을 초기화 하기 위해 INIT를 지원하는 송신측의 전송 주소를 가리키는 IPv4 또는 IPv6 파라미터로 보내진다. Association의 생명주기동안 IP 주소는 INIT의 송신자로부터 IP datagram의 소스 주소 필드 안에 나타난다. 그리고, INIT의 수신측으로부터 IP datagram을 보내는 목적지 주소로 사용될 수 있다.

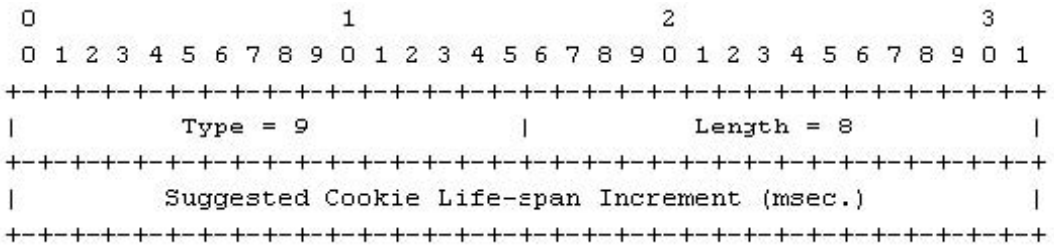
INIT 송신자가 multi-homed 일 때 INIT chunk내에 하나 이상의 IP 주소 파라미터가 포함될 수 있다. 더욱이 multi-homed endpoint는 네트웍의 다른 타입으로 접근할 수 있기 때문에 하나 이상의 주소타입이 하나의 INIT chunk내에 나타날 수 있다. 예를 들어 IPv4와 IPv6 주소는 동일한 INIT chunk에 전송이 가능하다.

INIT이 적어도 하나의 IP Address parameter를 갖는다면, INIT chunk를 포함하는 IP datagram의 소스 주소와 INIT 내에 제공되어진 추가된 모든 주소는 INIT를 수신하는 endpoint에 의해서 목적지로 사용 가능하다. 만일 INIT이 어떠한 IP Address parameter도 포함하고 있지 않다면, INIT를 수신하는 endpoint는 association에 대한 단일 목적지 주소로써 수신된 IP datagram와 연관된 소스 주소를 사용해야만 한다.

INIT와 INIT-ACK 안의 어떤 전송 주소를 이용하지 않는 것은 NAT Box를 통하여 작업하기에 유사한 연결을 만들기 위한 선택이다.

Cookie Preservative (9)

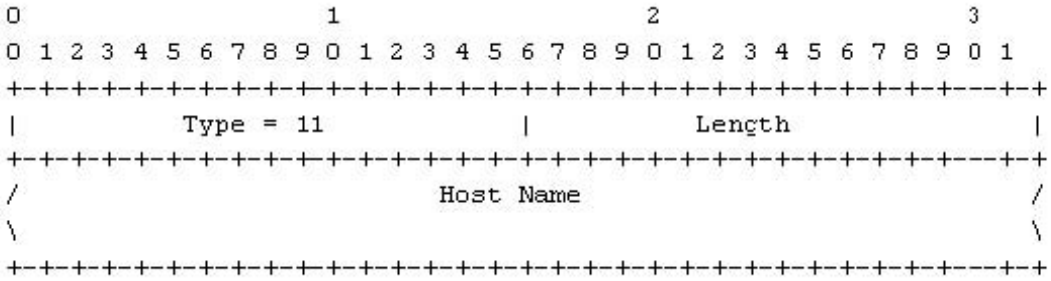
INIT 송신측은 보다 긴 State Cookie의 생명기한을 위해서 INIT의 수신측에 제안하기위한 parameter를 사용할 것이다.



Suggested Cookie Life-span Increment : 32 bits (unsigned integer) 송신자는 수신자가 cookie 생명주기의 기본값으로 추가하기를 원하는 밀리초 증가분
을 수 신자에게 보낸다.
이 선택적 파라미터는 오래된 cookie operation error 때문에 이전에 설정한 association이 실패 된 peer와 association을 설정하기위해 재시도했을 때 송신측에 의해 INIT chunk에 추가되어질 것이다. 수신측은 보완의 이유로 제안된 cookie 생명주기의 증가를 무시할지를 선택할 것이다.

Host Name Address (11)

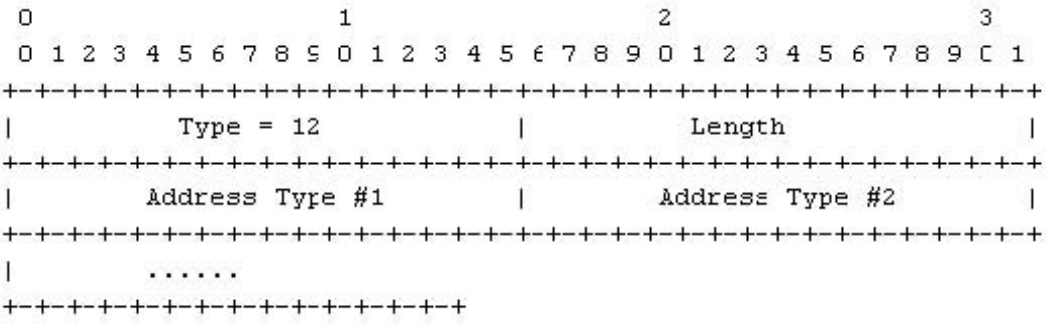
INIT 송신자는 peer에 IP address 위치에 Host 이름으로 보내기 위해 사용한다. Peer는 이름을 결정할 의무가 있다.



Host Name : variable length 이 필드는 RFC1123의 2.1 섹션의 host name syntax에 의해서 호스트 이름을 포함한다. 호스트 이름을 결정하기위한 방법은 SCTP의 범위 밖이다. Note : 적어도 하나의 널 터미널은 Host Name 스트링에 포함되어지고 길이에 포함되어져야만 한다.

Supported Address Types (12)

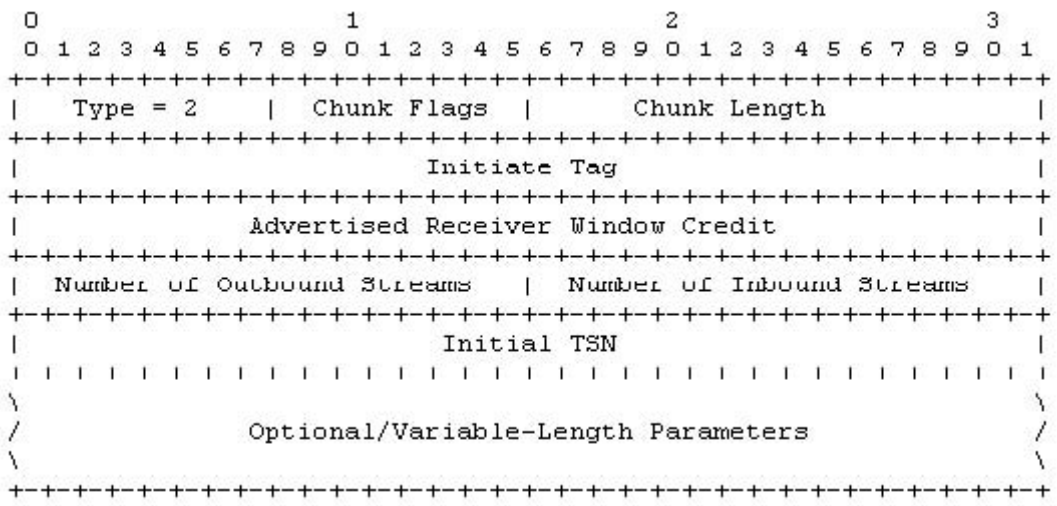
INIT 송신측은 지원 가능한 모든 주소 타입의 리스트로 사용한다.



Address Type : 16 bits (unsigned integer) 주소 TLV에 상응하는 타입의 값으로 채워진다. 예를 들어 IPv4 = 5, IPv6 = 6, Hostname = 11)

3.3.3 Initiation Acknowledgement(INIT ACK)

Init Chunk의 ACK를 보내기 위해 사용되어지며 형태는 INIT Chunk와 유사하나 State Cookie와 Unrecognized Parameter가 Variable Parameter에 포함된다. INIT ACK 청크의 형태는 아래에서 보여진다.



Initiate tag : 32bits(unsigned integer)

INIT ACK 수신자는 Initiate tag의 파라미터를 기록하고 그 값은 INIT ACK수신자가 이 연결 안에서 전송하는 모든 SCTP 패킷의 Verification Tag에 기록한다. 수신된 INIT ACK Chunk 안에 Initiate tag 값에 0이 발견된다면 수신자는 그 값을 에러로 처리하고 ABORT를 전송하여 연결을 종료한다.

Advertised Receiver Window Credit(a_rwnd): 32bit(Unsigned integer) :

Bytes 수의 버퍼 공간으로써, INIT의 송신자는 association에 이 window를 확보한다. 버퍼공간은 association의 수명 동안 줄여서는 안될 것이다.

Number of Outbound Stream(OS) : 16bit(Unsigned integer)

INIT ACK송신자가 연결상에서 생성하고자 하는 outbound streams 의 숫자를 정의하며 0은 사용할 수 없다.

Number of Inbound Stream(MIS) : 16bit(Unsigned integer)

INIT ACK송신자가 연결상에서 가지고자 하는 최대 inbound streams 의 숫자를 정의하며 0 은 사용할 수 없다.

Initial TSN(I-TSN) : 32bits(unsigned integer)

값의 범위는 0 ~ 4294967295의 범위 내에서 사용되며 INIT ACK송신자가 사용하게 된다.

Fixed Parameters	Status
Initiate Tag	Mandatory
Advertised Receiver Window Credit	Mandatory
Number of Outbound Streams	Mandatory
Number of Inbound Streams	Mandatory
Initial TSN	Mandatory

Variable Parameters	Status	Type Value
State Cookie	Mandatory	7
IPv4 Address (Note 1)	Optional	5
IPv6 Address (Note 1)	Optional	6
Unrecognized Parameters	Optional	8
Reserved for ECN Capable (Note 2)	Optional	32768 (0x8000)
Host Name Address (Note 3)	Optional	11

IP주소 :

INIT ACK Chunk 안의 IP주소 파라미터에 여러 개의 IP주소를 기록 할 수 있으며 그 값은 IPv4/IPv6 모두 포함된다.

ECN capable field :

INIT ACK Chunk 안의 ECN capable field는 미래의 Explicit Congestion Notification 위하여 예약된다.

Host Name Address :

INIT ACK Chunk 안의 Host Name Address는 한 개 이상의 값이 올 수 없고 송신자는 INIT ACK안에 Host Name Address를 가지는 어떤 다른 주소 값을 넣을 수 없다.
만약 값이 있는 경우는 다른 endpoint에서 무시를 함

Implement :

코드 구현 시 INIT ACK Chunk를 받기 위하여 준비되어지는 버퍼는 최소 1500 byte이의 값을 가지고 있어야 하며 그 이유는 State cookie와 가변 주소 리스트의 크기가 변하기 때문이다.

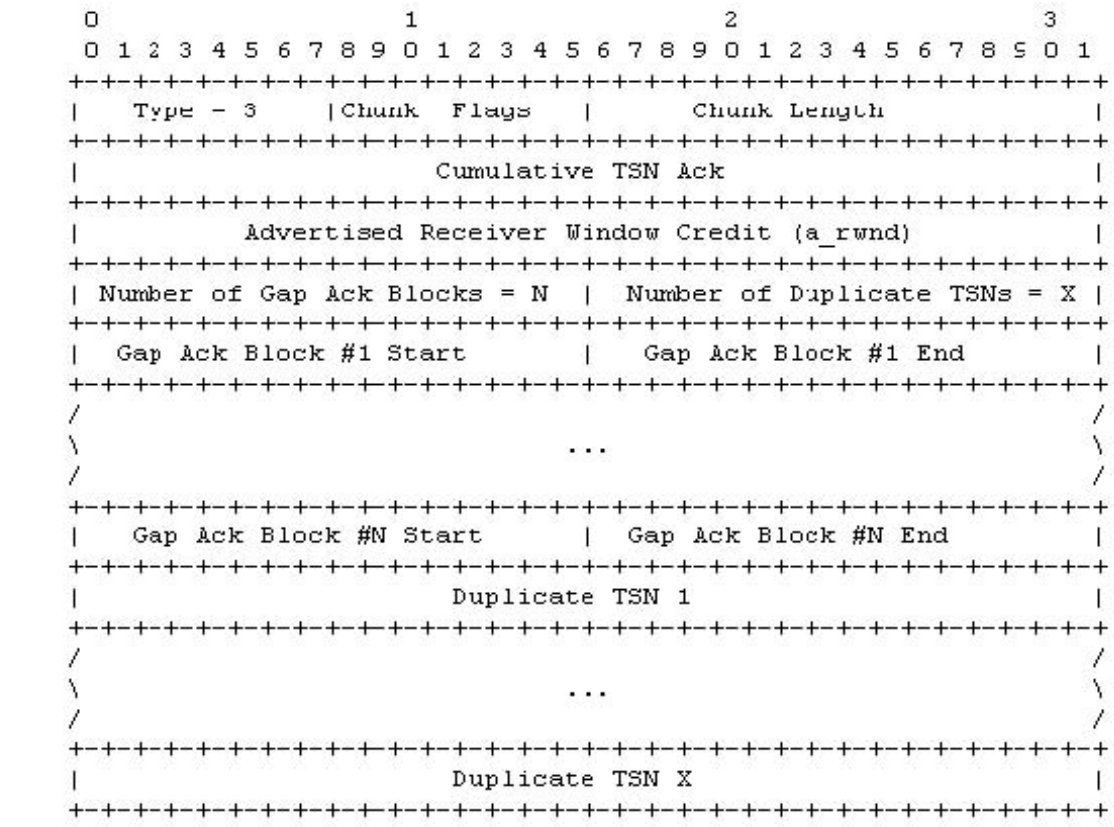
SCTP Common header안에서 전송된 Source port와 조합 된 INIT ACK 안의 각 IP주소는 초기화된 연결의 Lifetime을 위하여 INIT ACK Chunk의 송신자에 의해서 지원되는 유효한 전송주소를 INIT ACK의 수신자를 지적한다.

만약 INIT ACK가 적어도 하나의 주소를 가지고 있고 INIT ACK를 포함하는 IP datagram의 주소와 INIT ACK내에 제공되는 어떤 주소는 INIT ACK Receiver 에 의해서 목적지 주소로 사용된다.
만약 INIT ACK가 어떤 목적지 주소도 가지고 있지 않으면 INIT ACK수신자는 목적지 주소로 수신된 IP Datagram과 연관된 Source Address를 이용한다.

3.3.4 Selective Acknowledgement(SACK) :

이 청크는 endpoint에게 수신된 DATA Chunk를 알리고 수신된 DATA Chunk의 TSN에으로 나타나지는 DATA Chunks의 하부 순서내의 갭을 Endpoint에게 통보하기 위해서 보내어진다. 또한 SACK는 Cumulative TSN ACK와 Advertise Receiver Window Credit(a_rwnd)파라미터가 있어야 한다.
정의에 의해서 Cumulative TSN ACK은 수신된 TSN의 순서 내에서 발생한 갭이

생기기전의 마지막 TSN의 값이다 : 즉 이 Cumulative TSN ACK 뒤에 따라오는 다음 TSN 값은 SACK을 보낸 endpoint 에게 수신되지 않았다. 그러므로 모든 TSN 의 수신된 acknowledged의 파라미터는 TSN 의 값보다 같거나 작다. ACK는 0 또는 한 개 이상의 GAP ACK Block을 포함하며 이는 전체 수신된 TSN순서상에서 단절 다음에 수신된 TSN의 다음 순서를 기록하게 된다.




```
 ---------------------------------------------------------------
 typedef struct SCTP_SACK_CHUNK
{
    SCTP_chunk_header chunk_header;
    guint32 cumulative_tsn_ack;
    guint32 a_rwnd;
    guint16 num_of_fragments;
    guint16 num_of_duplicates;
    guchar fragments_and_dups[MAX_VARIABLE_SACK_SIZE];
}
SCTP_sack_chunk;
---------------------------------------------------------------
```

Chunk Flag : 8bits
    모두 0으로 설정되었고 수신은 무시한다.

Cumulative TSN ACK : 32bit(Unsigned Integer)
    수신된 DATA Chunk 내에서 갭 전에 나타난 마지막 TSN번호이다.

Advertised Receiver Window Credit (a_rwnd) : 32 bits(unsigned integer)
    자신이 수신할 수 있는 버퍼의 크기를 알린다.

Number of Gap Ack Blocks : 16bits(unsigned integer)
    SACK안에 포함된 Gap Ack Blocks의 숫자이다.

Number of Duplicate TSN : 16bit 중복된 TSN의 숫자이다.

Gap Ack Blocks :
이 필드는 Gap Ack Block를 가리키며 이들은 Number of Gap Ack Blocks에서 정의된 숫자에 이르기까지 계속 반복되어 나타난다. TSN을 가지는 모든 DATA Chunk의 모든 Gap Ack Block 값은 Cumulative TSN Ack + GAP Ack Block Start 보다 크거나 같고 Cumulative TSN Ack + GAP Ack Block End 의 값과 작거나 같다.

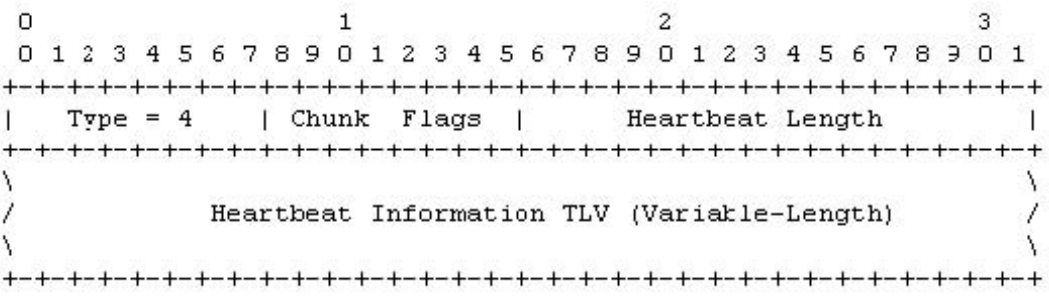
Gap Ack Blocks Start : 16bits(unsigned integer)
Gap Ack Block의 Start offset값을 나타낸다. 이 값에 Cumulative TSN값을 더하면 Gap Ack Block의 시작 TSN 값이 나온다.

Gap Ack Blocks End : 16bits(unsigned integer)
Gap Ack Block의 End offset값을 나타낸다. 이 값에 Cumulative TSN값을 더하면Gap Ack Block의 끝TSN 값이 나온다. 예를 들어 Selective ACK를 송신한다고 결정할 때 수신자가 최근에 한번에 도착한 연속된 청크를 가지고 있다고 가정 할 수 있다.

Duplicate TSN : 32 bits(unsigned integer)
최종 송신한 SACK 이후에부터 중복돼서 수신된 TSN의 숫자를 가리킨다. 매번 수신자는 SACK를 보내기 전 중복 TSN를 계산하고 중복 list에 더한다. SACK 송신 후에는 그 값은 다시 0으로 초기화 된다.

3.3.5 Heartbeat Request(HEARTBEAT)(4) :

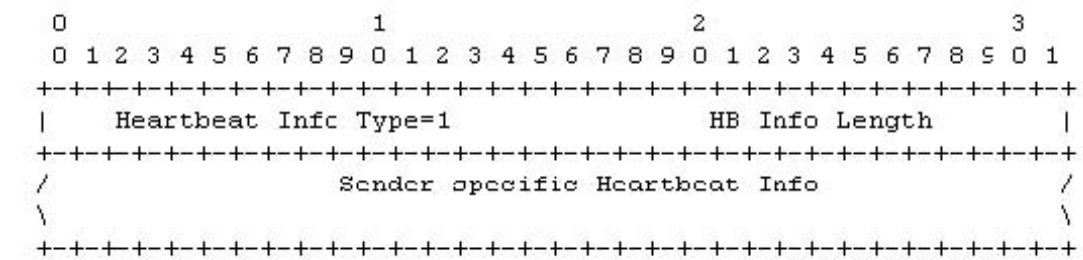
하나의 End point가 현재 연결에서 정의된 특정 목적지 주소의 Reach ability를 판별하기 위해 연결상에 있는 다른 end point에게 보내게 된다.



Chunk Flags : 8 bits 전송 시0 으로 설정되고 수신은 무시

Heartbeat Information(variable length)
HEARTBEAT가 보내 질 때 송신자의 현재 시간과 HEARTBEAT가 보내어지는 목적지 주소가 들어 있다.

Variable Parameters	Status	Type Value
Heartbeat Info	Mandatory	1



3.3.6 Heartbeat Acknowledgement(HEARTBEAT ACK)

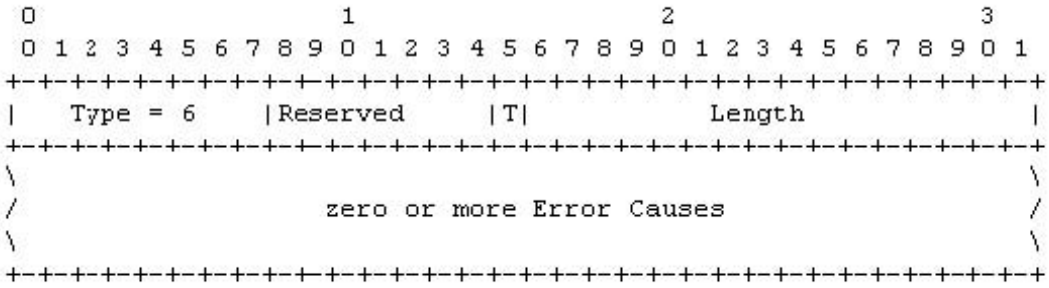
HEARTBEAT Chunk 에 대한 응답으로서 peer endpoint 에게 보낸다.

Chunk Flags : 8 bits 전송상에 0으로 설정되고 수신은 무시

Heartbeat Ack length : 16bits(unsigned integer) Byte 크기의 청크의 크기를 설정
Heartbeat Information : variable length
Heartbeat Request의 파라미터 정보를 Heartbeat Information 정보에 담는다.

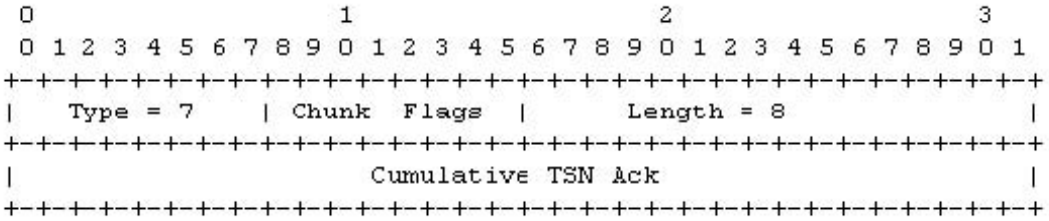
3.3.7 Abort Association(ABORT)(6):

ABORT는 연결을 close하기 위해 endpoint에게 보내어진다. 이는 비정상 종료 시 사용된다. ABORT는 DATA Chunk와 같이 bundle 되지 않고, Control chunk(except for INIT, IINIT ACK, and SHUTDOWN COMPLETE)는 Abort와 같이 사용될 수 있다. 단 Abort Chunk 앞에 위치 해야 한다. ABORT Receiver는 응답하지 않아도 된다.



3.3.8 Shutdown Association (SHUTDOWN)

End point peer와 graceful close를 시작하기 위해 보내어지는 Chunk



Chunk Flags : 8 bits 전송 시 0 에 설정 수신은 무시

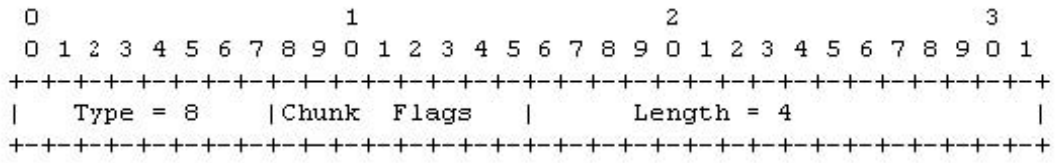
Length 16bits : 8의값을 가진다.

Cumulative TSN Ack :

SHUTDOWN 메시지는 Gap Ack Blocks를 포함하지 않고 TSN을 알리기 위해서 이용되지 않는다. 이전에 보내어진 TSN을 보낸다.

3.3.9 Shutdown Acknowledgment (SHUTDOWN ACK)

SHUTDOWN CHUNK의 응답으로 보내어지며 어떠한 파라미터도 갖지 않는다.



4. SCTP Association State

SCTP 연결이 처리되는 동안 SCTP endpoints 연결 진행은 다양한 Event와 Action 에 대한 응답에 따른 상태의 변화를 나타낸다. 이러한 이벤트는

User primitive calls : ASSOCIATE, SHUTDOWN, ABORT등에
Reception : INIT, COOKIE ECHO, SHUTDOWN, 등의 Control CHUNK
Event : Timeout event 등으로 구성 된다.

Assocation의 상태에 대한 구현은 association 구조체의 멤버변수인 sctp_controldata구조체에서 가지고 있다. 각각의 asociaton상태에 따라 controldata의 멤버변수인 association_state값을 설정을 하고 association_state상태에 따라서 sctp protocol은 동작한다. 다음은 state에 따른 동작 과정이다.

1) 수신된 COOKIE ECHO 안에 State Cookie가 유효하지 않으면 수신된 패킷을 지워 버린다. 또한 수신된 State Cookie가 time이 만료가 되었다면 수신자는 ERROR CHUNK를 전송한다. 그리고 수신자는 CLOSE상태에 남아 있는다.

2) T1-Init timer가 만기가 되었으면 endpoint는 INIT를 재전송하고 상태의 변화 없이T1-t imer를 재가동한다. 그리고 이러한 작업은 Max.Init.Retransmits 횟수가 될 때까지 반복 실행 해야만 하고 그 이후에는 초기화 과정을 멈추고 SCTP 사용자에게 에러를 보고한다.

3) T1-cookie timer가 만기가 되었으면 endpoint는 INIT를 재전송 해야만 하고 상태의 변화 없이 T1-cookie timer를 재가동한다. 그리고 이러한 작업은 Max.Init.Retransmits 횟수가 될 때까지 반복 실행 해야만 하고 그 이후에는 초기화 과정을 멈추고 SCTP 사용자에게 에러를 보고한다.

4) SHUTDOWN-SENT endpoint는 지연 없이 수신된 DATA CHUNK를 알려야 한다.

5) SHUTDOWN-RECEIVED 상태에서 endpoint는 SCTP User로부터 새로운 요청을 받을 수 없다.

6) SHUTDOWN-RECEIVED 상태에서 endpoint는 Queue에 들어 있는 모든 데이터를 전송하고 실패 한 경우에는 재전송해야만 한다.

7) SHUTDOWN ACK-SENT 상태에서 SCTP User에서 새로운 요청을 받지 않는다.

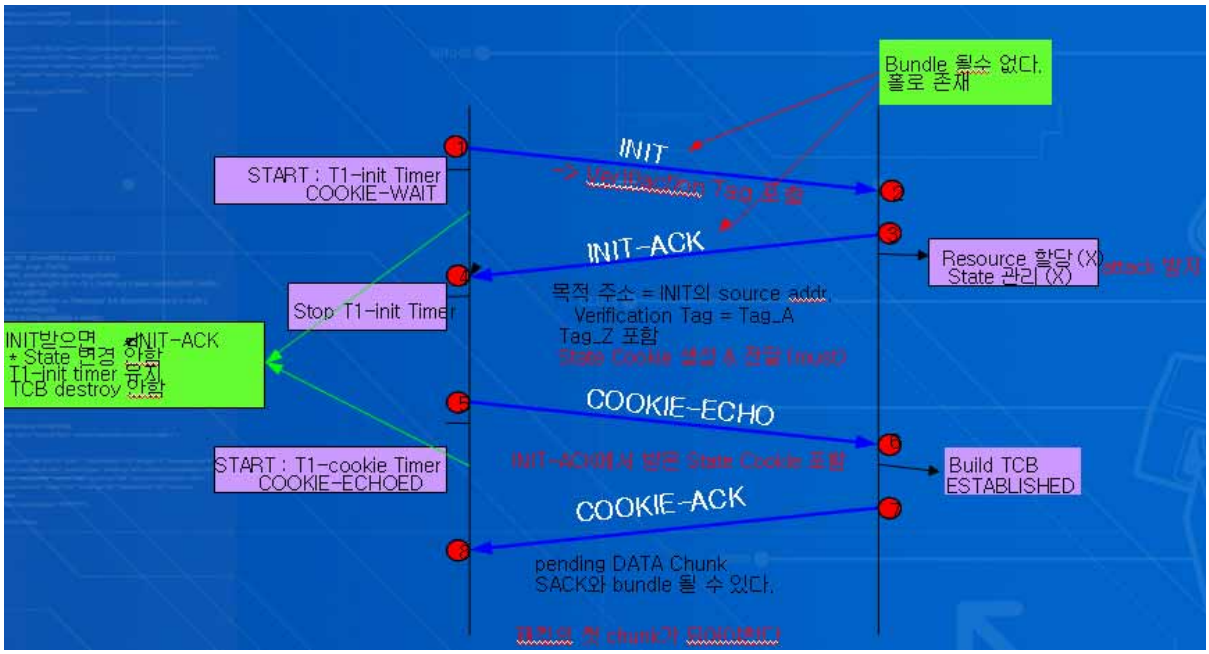
Close상태는 연결이 생성되지 않은 상태가 아니라 존재하지 않는 생태를 나타낸다.

5. Association Initialization

첫번째 데이터를 전송하기 전에 하나의 SCTP endpoint A로부터 다른 endpoint Z에게 발생할 것이다. 두개의 endpoint는 둘 사이의 SCTP연결을 설정하기 위하여 초기화 과정을 완성해야 한다. SCTP User는 SCTP연결을 설정하기 위하여 ASSOCIATE primitive를 이용한다.

5.1 Normal Establishment of an Association

초기화 단계는 연속된 단계로 이루어져 있으며 SCTP endpoint A에서 다른 SCTP endpoint Z 에게 연결을 시도하고 있고 Z는 이를 수락 한다고 가정한다.



A) A는 처음에 Z에게 INIT CHUNK를 보낸다. A는 Initiate Tag 안에 A의 Verification Tag(Tag_A)를 제공해야 하며 이때 Tag-A는 1 ~ 4294967295의 범위 안에서 임의의 숫자가 되어야 한다. A는 INIT를 전송 후에 T1-init timer를 가동하고 COOKIE_WAIT상태로 들어간다.

B) Z는 INIT ACK를 즉시 응답한다.
INIT ACK의 목적지 IP주소를 INIT ACK가 응답하려 하는 INIT의 Source IP로 설정한다. 응답에 있어서 다른 파라미터들을 채우는 것 이외에 Z는 Tag-A(A의 Initiate Tag)를 Verification Tag field에 설정하고 자신의 Verification Tag(Tag-Z)를 Initiate Tag에 설정하여 제공한다.
Note : State Cookie 파라미터와 함께 INIT Ack를 전송한 후에 Z 는 어떠한 자원도 할당해서는 안되고 새로운 연결을 위해서 어떠한 상태도 유지하지 안는다. 그렇지 않으면 Z는 Resource attacks에 침입을 받을 수 있게 되어진다.

C) A는 Z로부터 INIT ACK를 수신하자 마자 A는 T1-init timer를 멈추고 COOKIE

WAIT상태를 떠난다. 그리고 COOKIE ECHO CHUNK안에 INIT ACK CHUNK에서 받아진 State Cookie를 보낸 후 T1-cookie timer를 가동한다. 그리고 COOKIE ECHOED 상태로 들어간다.

Note : COOKIE ECHO chunk는 미결정된 나간 DATA Chunk와 번들링 될 수 있다. 그러나 그것은 패킷 안의 첫번째 Chunk가 되어야 하며 COOKIE ACK가 반환 될 때까지 송신자는 peer에게 어떤 다른 패킷을 보내어서는 안 된다.

COOKIE ECHO CHUNK를 받자 마자 Z는 TCB를 구축하고 ESTABLISHED 상태로 이동 후에 COOKIE ECHO ACK를 전송 한다. COOKIE ACK CHUNK는 다른 DATA CHUNK와 같이 전송 될 수 있으나 반드시 패킷의 처음에 와야 한다.(이때 연결을 위한 자원일 할당된다.)

IMPLEMENTATION NOTE :
COOKIE ECHO CHUNK가 유효한 것이 도착 하였다면 Communication Up Notification 은 SCTP사용자에게 알리는 것은 선택 사항이다.
A가 COOKIE ECHO ACK CHUNK를 받자 마자 EXTABKISHED상태로 변경 되어야 한다. 그리고 T1-cookie timer를 멈춘다. Communication Up Notification을 이용하여 연결의 성공을 ULP에게 알려야 한다.

INIT이나 INIT ACK chunk는 다른 chunk들과 함께 bundling 될 수 없다. 그것 들은 자신들을 실어 나르는 SCTP 패킷 안에서 나타나는 유일한 Chunk들이어야 한다. 하나의 endpoint는 INIT를 수신한 곳의 IP주소에 INIT ACK를 보내주어야 한다.

ABORT Chunk를 포함하는 나가는 SCTP 패킷의 common header 안의 Verification Tag Field는 peer의 Initiate Tag 값에 설정되어야만 한다.

IMPLEMENTATION NOTE : IP 주소와 SCTP 포트는 일반적으로 하나의 SCTP 인스턴스 안에서 TCB를 찾기 위한 Key로서 이용되어진다.

5.2 Normal Establishment of an Association code

sctplib에서 Association을 맺기 위한 과정을 이야기 한다.

SCTP input은 sctplib 코드 중 가장 많은 분량을 차지 한다. input과정은 최초 라이브러리가 로딩 되고 sctpsock이 데이터를 받아 들일 준비를 수행을 하게 되는데 adl_receive_message를 통해서 raw socket인 sctpsock을 통해 데이터를 받아 들이고 들어 온 데이터(ip를 포함하는 sctp프로토콜 데이터)의 전체길이가 ip 헤더에 명시된 hlen길이보다 작다면 그냥 버리고 정상적인 패킷이라면 mdi_receiveMessage를 호출하여 받아온 sctp 패킷에 bundling 되어진 chunk들이 정상적인지 과정을 수행 하게 된다. 받아온 패킷에 source address를 가지고 state 상태에 따라 패킷을 처리 할지 안 할지 결정하게 되고 정상적인 패킷 이라면 rbu_rcvDatagram()을 호출하게 된다.
rbu_rcvDatagram()에서는 bundling된 count만큼 반복하여 각각의 chunk들에 대한 처리를 하게 되는데 code는 다음과 같다.

```
gint SSM_Bundling::rbu_rcvDatagram(guint address_index, guchar * datagram, guint len)
{

    guchar *current_position;
    gushort processed_len = 0, chunk_len;
    gushort pad_bytes;
    SCTP_simple_chunk *chunk;
    gboolean data_chunk_received = FALSE;

    int association_state = STATE_OK;
    gboolean send_it = FALSE;
```

```
    int result;

    bu_lock_sender();

    current_position = datagram; /* points to the first chunk in this pdu */
while (processed_len < len) {

        chunk = (SCTP_simple_chunk *) current_position;
        chunk_len = CHUNKP_LENGTH((SCTP_chunk_header *) chunk);



         switch (chunk->chunk_header.chunk_id) {
        case CHUNK_DATA:
                  SSM_Recvctrl::rxc_data_chunk_rx((SCTP_data_chunk*) chunk,
         address_index);
            data_chunk_received = TRUE;
            break;
        case CHUNK_INIT:
                  association_state = SSM_SCTP_Control::sctlr_init((SCTP_init *) chunk);
            break;
        case CHUNK_INIT_ACK:
         case CHUNK_SACK:
         case CHUNK_HBACK:
            event_log(INTERNAL_EVENT_0, "********************  Bundling received
        if (data_chunk_received == TRUE){
            send_it = SSM_Recvctrl::rxc_create_sack(&address_index, FALSE);
                        SSM_Streamengine::se_doNotifications();
            if (send_it==TRUE) bu_sendAllChunks(&address_index);
        }
        /* if the association has already been removed, we cannot unlock it anymore */
        bu_unlock_sender(&address_index);
    }

    return 0;

}
```

switch문을 통해서 bundling 된 chunk의 타입을 확인하고 각각의 chunk타입에 맞게 처리를 하게 된다.

```
/**
  SCTP-control structure. Stores also the current state of the state-machine.
 */
typedef struct SCTP_CONTROLDATA
{
    /*@{ */
    /** the state of this state machine */
    guint32 association_state;
    /** stores timer-ID of init/cookie-timer, used to stop this timer */
    TimerID initTimer;
    /** */
    unsigned int initTimerDuration;
    /**   stores the association id (==tag) of this association */
    unsigned int associationID;
    /** Counter for init and cookie retransmissions */
    short initRetransCounter;
    /** pointer to the init chunk data structure (for retransmissions) */
    SCTP_init *initChunk;
    /** pointer to the cookie chunk data structure (for retransmissions) */
    SCTP_cookie_echo *cookieChunk;
```

```
    /** my tie tag for cross initialization and other sick cases */
    guint32 local_tie_tag;
    /** peer's tie tag for cross initialization and other sick cases */
    guint32 peer_tie_tag;
    /** we store these here, too. Maybe better be stored with StreamEngine ? */
    unsigned short NumberOfOutStreams;
    /** we store these here, too. Maybe better be stored with StreamEngine ? */
    unsigned short NumberOfInStreams;
    /** value for maximum retransmissions per association */
    int assocMaxRetransmissions;
    /** value for maximum initial retransmissions per association */
    int assocMaxInitRetransmissions;
    /** value for the current cookie lifetime */
    int cookieLifeTime;
    /** the sctp instance */
    void * instance;
    /*@} */
} SCTP_controlData;
```

각각의 chunk에 대해서는 chunk를 수락하였을 때 assocation의 상태 정보에 따라서 수행을 하게 된다. SCTP_CONTROLDATA는 association의 설정 시 포함되어야 하는 association 상태 정보와 timer, init chunk point, cookie_echo 정보 등을 가짐으로써 현재 association상태를 나타내게 된다.

다음은 init메시지를 수신에서 받았을때의 데이터 처리이다.

Case CHUNK_INIT: 일 때 다음과 같은 코드를 호출한다.

```
int SSM_SCTP_Control::sctlr_init(SCTP_init * init)
{
if ((localData = (SCTP_controlData *) SSM_Distribution::mdi_readSCTP_control()) == NULL) {
        event_log(VERBOSE, " DO_5_1_B_INIT: Normal init case ");
        /* DO_5_1_B_INIT : Normal case, no association exists yet */
        /* save a-sides init-tag from init-chunk to be used as a verification tag of the sctp-
           message carrying the initAck (required since no association is created). */
        SSM_Distribution::mdi_writeLastInitiateTag(SSM_ChunkHandler::ch_initiateTag(initCID));

        /* Limit the number of sendstreams a-side requests to the max. number of input streams
           this z-side is willing to accept.
         */
        inbound_streams = min(SSM_ChunkHandler::ch_noOutStreams(initCID),
SSM_Distribution::mdi_readLocalInStreams());
        outbound_streams = min(SSM_ChunkHandler::ch_noInStreams(initCID),
SSM_Distribution::mdi_readLocalOutStreams());
        /* fire back an InitAck with a Cookie */
        initAckCID = SSM_ChunkHandler::ch_makeInitAck(SSM_Distribution::mdi_generateTag(),
                                SSM_Distribution::mdi_getDefaultMyRwnd(),
                                outbound_streams,
                                inbound_streams, SSM_Distribution::mdi_generateStartTSN());

        /* retreive a-side source addresses from message */
        supportedTypes = SSM_Distribution::mdi_getSupportedAddressTypes();

        nrAddresses = SSM_ChunkHandler::ch_IPaddresses(initCID, supportedTypes, rAddresses,
&peerSupportedTypes, &last_source);


        if ((supportedTypes & peerSupportedTypes) == 0)
            error_log(ERROR_FATAL, "BAKEOFF: Program error, no common address types in sctlr_init()");

        /* enter variable length params initAck */
        SSM_Distribution::mdi_readLocalAddresses(lAddresses, &nlAddresses, &last_source, 1,
peerSupportedTypes,TRUE);
        /* enter local addresses into initAck */
        if (nlAddresses > 1)
```

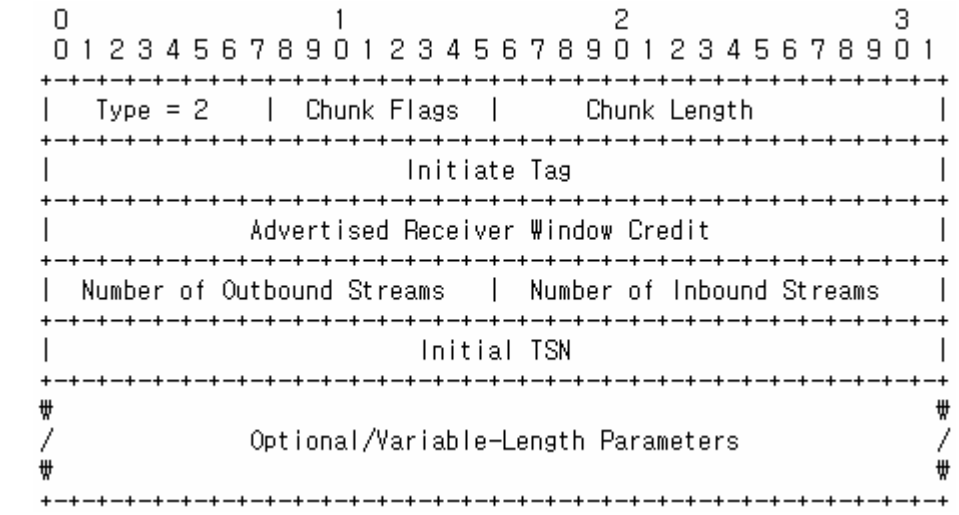
```
            SSM_ChunkHandler::ch_enterIPaddresses(initAckCID, lAddresses, nlAddresses);

        /* append cookie to InitAck Chunk */
        SSM_ChunkHandler::ch_enterCookieVLP(initCID, initAckCID,
                        SSM_ChunkHandler:: ch_initFixed(initCID),
SSM_ChunkHandler::ch_initFixed(initAckCID),
                         SSM_ChunkHandler::ch_cookieLifeTime(initCID), 0, /* tie tags are both zero */
                         0, lAddresses, nlAddresses, rAddresses, nrAddresses);


        process_further = SSM_ChunkHandler::ch_enterUnrecognizedParameters(initCID, initAckCID,
supportedTypes);

        if (process_further == -1) {
         /*    ch_deleteChunk(initAckCID);
            ch_forgetChunk(initCID); */
            return_state = STATE_STOP_PARSING; /* to stop parsing without actually removing it */
            /* return return_state; */
        } else {
            if (process_further == 1) {
                return_state = STATE_STOP_PARSING; /* to stop parsing without actually removing it */
            }
            /* send initAck */
            SSM_Bundling::bu_put_Ctrl_Chunk(SSM_ChunkHandler::ch_chunkString(initAckCID),NULL);
        }
        SSM_Bundling::bu_sendAllChunks(NULL);
        SSM_Bundling::bu_unlock_sender(NULL);
        SSM_ChunkHandler::ch_deleteChunk(initAckCID);
        event_log(INTERNAL_EVENT_1, "event: initAck sent");

return return_state;
}
    }
```



Init_chunk를 받았을 상황은 현재 associaton이 설정되지 않은 상황이고 현재 associaton에 대한 아무런 정보도 가지고 있지 않은 상황이라는 가정하에서 구현 되어야 한다. sctlr_init(SCTP_init * init)함수 안에서 먼저 in, ,out stream number을 체크하고 이상이 없으면, 현재 받은 init를 이용하여 simple init_ack chunk를 생성한다. mdi_generateStartTSN();를 통해 초기 tsn값을 설정하고, 받은 init_chunk의 in stream과 out stream을 가지고 현재 자신에게 설정된 in out을 비교하여 작은 것을 in, out으로 init_ack field에 설정한다. 현재 지원하는 local ip address를 설정하고 cookey정보를 ch_enterCookieVLP()를 통하여 init_ack 에 삽입한다. 이 과정이 순조롭게 끝나면 bu_put_Ctrl_Chunk()을 이용하여 번들링 준비된 buffer에 삽입하고, SSM_Bundling::bu_sendAllChunks(NULL);을 호출하여 전송한다.

다음은 init을 받은 후 INIT_ACK를 수행 할 때의 데이터 처리이다.

```
---------------------------------------------------------------------------------------------
case CHUNK_INIT_ACK:

SSM_SCTP_Control::sctlr_initAck((SCTP_init *) chunk)
{
state = localData->association_state;

    switch (state) {
    case COOKIE_WAIT:

        event_log(EXTERNAL_EVENT, "event: initAck in state COOKIE_WAIT");

        /* Set length of chunk to HBO !! */
        initCID = SSM_ChunkHandler::ch_makeChunk((SCTP_simple_chunk *)
localData->initChunk);

        /* FIXME: check also the noPeerOutStreams <= noLocalInStreams */
        if (SSM_ChunkHandler::ch_noOutStreams(initAckCID) == 0 ||
SSM_ChunkHandler::ch_noInStreams(initAckCID) == 0 ||
SSM_ChunkHandler::ch_initiateTag(initAckCID) == 0) {
            if (localData->initTimer != 0) {
                sctp_stopTimer(localData->initTimer);
                localData->initTimer = 0;
            }
           /* make and send abort message */
            abortCID = SSM_ChunkHandler::ch_makeSimpleChunk(CHUNK_ABORT,
FLAG_NONE);
            SSM_ChunkHandler::ch_enterErrorCauseData(abortCID,
ECC_INVALID_MANDATORY_PARAM, 0, NULL);

SSM_Bundling::bu_put_Ctrl_Chunk(SSM_ChunkHandler::ch_chunkString(abortCID),NULL);
            SSM_ChunkHandler::ch_deleteChunk(abortCID);

            SSM_Bundling::bu_unlock_sender(NULL);
            SSM_Bundling::bu_sendAllChunks(NULL);
            /* delete all data of this association */
            SSM_Distribution::mdi_deleteCurrentAssociation();
            SSM_Distribution::mdi_communicationLostNotif(0);
            SSM_Distribution::mdi_clearAssociationData();
            return_state = STATE_STOP_PARSING_REMOVED;
            return return_state;
        }

        result = SSM_Distribution::mdi_readLastFromAddress(&destAddress);
        if (result != 0) {
            if (localData->initTimer != 0) {
                sctp_stopTimer(localData->initTimer);
                localData->initTimer = 0;
            }
            SSM_Bundling::bu_unlock_sender(NULL);
            SSM_Distribution::mdi_deleteCurrentAssociation();
            SSM_Distribution::mdi_communicationLostNotif(0);
            SSM_Distribution::mdi_clearAssociationData();
            return_state = STATE_STOP_PARSING_REMOVED;
            return return_state;
        }

        supportedTypes = SSM_Distribution::mdi_getSupportedAddressTypes();
        /* retrieve addresses from initAck */
        ndAddresses =SSM_ChunkHandler:: ch_IPaddresses(initAckCID, supportedTypes,
dAddresses, &peerSupportedTypes, &destAddress);
```

```
        SSM_Distribution:: mdi_writeDestinationAddresses(dAddresses, ndAddresses);

        /* initialize rest of association with data received from peer */

        inbound_streams = min(SSM_ChunkHandler::ch_noOutStreams(initAckCID),
localData->NumberOfInStreams);
        outbound_streams = min(SSM_ChunkHandler::ch_noInStreams(initAckCID),
localData->NumberOfOutStreams);

        peerSupportsPRSCTP = SSM_ChunkHandler::ch_getPRSCTPfromInitAck(initAckCID);



SSM_Distribution::mdi_initAssociation(SSM_ChunkHandler::ch_receiverWindow(initAckCID),
/* remotes side initial rwnd */
                        inbound_streams, /* # of remote output/local input streams */
                        outbound_streams, /* # of remote input/local output streams */
                        SSM_ChunkHandler::ch_initialTSN(initAckCID), /* remote initial TSN */
                        SSM_ChunkHandler::ch_initiateTag(initAckCID), /* remote init tag */
                        SSM_ChunkHandler::ch_initialTSN(initCID), /* local initial TSN for
sending */
                        peerSupportsPRSCTP,
                        FALSE);

       event_logii(VERBOSE, "sctlr_InitAck(): called mdi_initAssociation(in-streams=%u,
out-streams=%u)",
                    inbound_streams,outbound_streams);


        /* reset length field again to NBO... */
        SSM_ChunkHandler::ch_chunkString(initCID),
        /* free initChunk memory */
        SSM_ChunkHandler::ch_forgetChunk(initCID);

        cookieCID =
SSM_ChunkHandler::ch_makeCookie(SSM_ChunkHandler::ch_cookieParam(initAckCID));

        if (cookieCID < 0) {
            event_log(EXTERNAL_EVENT, "received a initAck without cookie");

            /* stop shutdown timer */
            if (localData->initTimer != 0) {
                sctp_stopTimer(localData->initTimer);
                localData->initTimer = 0;
            }
            missing_params.numberOfParams = htonl(1);
            missing_params.params[0] = htons(VLPARAM_COOKIE);

            scu_abort(ECC_MISSING_MANDATORY_PARAM, 6, (unsigned
char*)&missing_params);
            SSM_Bundling::bu_unlock_sender(NULL);
            /* delete this association */

            return_state = STATE_STOP_PARSING_REMOVED;
            localData->association_state = CLOSED;
            localData = NULL;
            return return_state;
        }


        process_further = SSM_ChunkHandler::ch_enterUnrecognizedErrors(initAckCID,
```

```
                                                supportedTypes,
                                                &errorCID,
                                                &preferredPrimary,
                                                &preferredSet,
                                                &peerSupportsIPV4,
                                                &peerSupportsIPV6,
                                                &peerSupportsPRSCTP,
                                                &peerSupportsADDIP);


        if (process_further == -1) {
            SSM_ChunkHandler::ch_forgetChunk(initAckCID);
            SSM_ChunkHandler::ch_deleteChunk(cookieCID);
            if (errorCID != 0) SSM_ChunkHandler::ch_deleteChunk(errorCID);
            SSM_Bundling::bu_unlock_sender(NULL);
            if (localData->initTimer != 0) {
                sctp_stopTimer(localData->initTimer);
                localData->initTimer = 0;
            }
            SSM_Distribution::mdi_deleteCurrentAssociation();
            SSM_Distribution:: mdi_communicationLostNotif(SCTP_COMM_LOST_FAILURE);
            SSM_Distribution::mdi_clearAssociationData();
            localData->association_state = CLOSED;
            localData = NULL;
            return_state = STATE_STOP_PARSING_REMOVED;
            return return_state;
        } else if (process_further == 1) {
            return_state = STATE_STOP_PARSING;
        }

        localData->cookieChunk = (SCTP_cookie_echo *)
SSM_ChunkHandler::ch_chunkString(cookieCID);
        /* populate tie tags -> section 5.2.1/5.2.2 */
        localData->local_tie_tag = SSM_Distribution::mdi_readLocalTag();
        localData->peer_tie_tag = SSM_ChunkHandler::ch_initiateTag(initAckCID);


        localData->NumberOfOutStreams = outbound_streams;
        localData->NumberOfInStreams =  inbound_streams;


        SSM_ChunkHandler::ch_forgetChunk(cookieCID);
        SSM_ChunkHandler::ch_forgetChunk(initAckCID);

        /* send cookie back to the address where we got it from      */
        for (index = 0; index < ndAddresses; index++)
            if (SSM_Adaptation::adl_equal_address(&(dAddresses[index]),&destAddress))
break;

        /* send cookie */
        SSM_Bundling::bu_put_Ctrl_Chunk((SCTP_simple_chunk *) localData->cookieChunk,
&index);
        if (errorCID != 0) {
            SSM_Bundling::bu_put_Ctrl_Chunk((SCTP_simple_chunk
*)SSM_ChunkHandler::ch_chunkString(errorCID), &index);
            SSM_ChunkHandler::ch_deleteChunk(errorCID);
        }

        SSM_Bundling::bu_sendAllChunks(&index);
        SSM_Bundling::bu_unlock_sender(&index);
        event_logi(INTERNAL_EVENT_1, "event: sent cookie echo to PATH %u", index);
```

```
        if (preferredSet == TRUE) {
            preferredPath = SSM_Distribution::mdi_getIndexForAddress(&preferredPrimary);
            if (preferredPath != -1)
                SSM_Pathmanagement::pm_setPrimaryPath(preferredPath);
        }

        state = COOKIE_ECHOED;

        if (localData->initTimer != 0) sctp_stopTimer(localData->initTimer);
        /* start cookie timer */
        localData->initTimer = SSM_Adaptation::adl_startTimer(localData->initTimerDuration,
                                            &sci_timer_expired, TIMER_TYPE_INIT,
                                            (void *) &localData->associationID, NULL);
        break;

    case COOKIE_ECHOED:
        /* Duplicated initAck, ignore */
        event_log(EXTERNAL_EVENT, "event: duplicatied sctlr_initAck in state
COOKIE_ECHOED");
        break;
    case CLOSED:
    case ESTABLISHED:
    case SHUTDOWNPENDING:
    case SHUTDOWNRECEIVED:
    case SHUTDOWNSENT:
        /* In this states the initAck is unexpected event. */
        event_logi(EXTERNAL_EVENT, "discarding event: sctlr_initAck in state %02d", state);
        break;
    default:
        /* error logging: unknown event */
        event_logi(EXTERNAL_EVENT, "sctlr_initAck: unknown state %02d", state);
        break;
    }
}
```

INIT_ACK chunk를 받았을 때 sctlr_initAck((SCTP_init *)함수가 호출이 된다. Init_ack를 받았을 시에 상황은 init을 이미 요청한 상황이기에 현재 TCB(association 정보)가 생성된 상황이다.(client side). 그렇기에 현재 control 정보에 association_state에 따라서 분기하게 된다. Association_state 가 COOKIE_WAIT 상태일 때만 이 작업을 수행하게 되고, 나머지는 무시하게 된다. 만약 association_state 가 COOKIE_WAIT이라면, 받아온 init_ack chunk를 가지고 현재 스트림 개수 체크하고 mdi_initAssociation()을 이용하여 assocation에 flowcontrol, stream engine등을 수행 할 수 있는 기반을 만든다. Init_ack정보를 가지고 cookey_echo chunk를 생성하여 전송하고 현재 상태를 cookey_echoed 상태로 바꾸고 현재 init_timer를 중지하고 바로 adl_startTimer()를 통하여 /* start cookie timer */를 수행한다. Sctp init과정을 수행하기 위해서는 응용측에서 sctp_associatex() call을 호출하면서 시작하게 된다. Sctp_associatex를 호출하면 lib 내부에서 SSM_SCTP_Control::scu_associate()함수를 호출하게 된다.

scu_associate()함수를 살펴 보도록 하자.

```
void SSM_SCTP_Control::scu_associate(unsigned short noOfOutStreams,
                    unsigned short noOfInStreams,
                    union sockunion* destinationList,
                    unsigned int numDestAddresses,
                    gboolean withPRSCTP)
{
    guint32 state;
    guint16 nlAddresses;
```

```
    union sockunion lAddresses[MAX_NUM_ADDRESSES];
    ChunkID initCID;
    unsigned int supportedTypes = 0, count;

    /* ULP has called sctp_associate at distribution.
       Distribution has allready allocated the association data and partially initialized */

    if ((localData = (SCTP_controlData *) SSM_Distribution::mdi_readSCTP_control()) ==
NULL) {
        error_log(ERROR_MAJOR, "read SCTP-control failed");
        return;
    }

    state = localData->association_state;

    switch (state) {
    case CLOSED:
        event_log(EXTERNAL_EVENT, "event: scu_assocatiate in state CLOSED");
        /* create init chunk and write data to it -- take AssocID as tag !!! */
        initCID = SSM_ChunkHandler::ch_makeInit(SSM_Distribution::mdi_readLocalTag(),
                            SSM_Distribution::mdi_getDefaultMyRwnd(),
                            noOfOutStreams, noOfInStreams,
SSM_Distribution::mdi_generateStartTSN());

        /* store the number of streams */
        localData->NumberOfOutStreams = noOfOutStreams;
        localData->NumberOfInStreams = noOfInStreams;

        supportedTypes = SSM_Distribution::mdi_getSupportedAddressTypes();

        /* enter enter local addresses to message. I send an Init here, so
         * I will include all of my addresses !
         */
        SSM_Distribution::mdi_readLocalAddresses(lAddresses,
                            &nlAddresses,
                            destinationList,
                            numDestAddresses,
                            supportedTypes,
                            FALSE);

        event_logi(VERBOSE, "1: supportedTypes : %u", supportedTypes);

        if (withPRSCTP) {
           SSM_ChunkHandler:: ch_addParameterToInitChunk(initCID, VLPARAM_PRSCTP, 0,
NULL);
        }

#ifdef BAKEOFF
         ch_addParameterToInitChunk(initCID, 0x8123, 17, (unsigned char*)localData);
         ch_addParameterToInitChunk(initCID, 0x8343, 23, (unsigned char*)localData);
         ch_addParameterToInitChunk(initCID, 0x8324, 1, (unsigned char*)localData);
         ch_addParameterToInitChunk(initCID, 0xC123, 31, (unsigned char*)localData);
#endif


        SSM_ChunkHandler::ch_enterSupportedAddressTypes(initCID, TRUE, FALSE, FALSE);


        event_logi(VERBOSE, "2: supportedTypes : %u", supportedTypes);

        if (nlAddresses > 1)
            SSM_ChunkHandler::ch_enterIPaddresses(initCID, lAddresses, nlAddresses);
```

```
        localData->initChunk = (SCTP_init *) SSM_ChunkHandler::ch_chunkString(initCID);
        SSM_ChunkHandler::ch_forgetChunk(initCID);

        /* send init chunk */
        for (count = 0; count < numDestAddresses; count++) {
            SSM_Bundling::bu_put_Ctrl_Chunk((SCTP_simple_chunk *) localData->initChunk,
&count);
            SSM_Bundling::bu_sendAllChunks(&count);
        }

        localData->cookieChunk = NULL;
        localData->local_tie_tag = 0;
        localData->peer_tie_tag = 0;

        /* start init timer */
        localData->initTimerDuration =
SSM_Pathmanagement::pm_readRTO(SSM_Pathmanagement::pm_readPrimaryPath());

        if (localData->initTimer != 0) sctp_stopTimer(localData->initTimer);

        localData->initTimer = SSM_Adaptation::adl_startTimer(localData->initTimerDuration,
                                                &sci_timer_expired,
                                                TIMER_TYPE_INIT,
                                                (void *) &localData->associationID, NULL);

        state = COOKIE_WAIT;
        break;
    default:
        error_logi(EXTERNAL_EVENT_X, "Erroneous Event : scu_associate called in state %u",
state);
        break;
    }

    localData->association_state = state;
    localData = NULL;
}
```

만약 현재 assocation구조체에 있는 control state가 CLOSED 상태일 때만 association을 맺기 위한 init_chunk 패킷을 생성할 수 있다. SSM_ChunkHandler::ch_makeInit()함수를 통해 서 init chunk를 만들고 SSM_Bundling::bu_put_Ctrl_Chunk()함수는 init_chunk를 보내어질 bundling instance에 추가 하게 된다. 하지만 실제적으로 init을 보낼때는 데이터는 번들링 되어 지지 않는다.

5.2.1 Generation State Cookie

INIT CHUNK의 응답으로 INIT ACK를 보낼 때 INIT ACK의 송신자는 State Cookie를 생성하고 INIT ACK 의 State Cookie 파라미터 안에 넣어 보내야만 한다. 이 State Cookie 안에는 송신자의 MAC(Message Authentication Code)와 State Cookie의 생명시간, 그리고 연결을 위해 필요한 모든 정보가 들어 있다.

```
    ------------------------------------------------------------
       /* fire back an InitAck with a Cookie */
        initAckCID                                                              =
SSM_ChunkHandler::ch_makeInitAck(SSM_Distribution::mdi_generateTag(),
                                 SSM_Distribution::mdi_getDefaultMyRwnd(),
                                 outbound_streams,
                                 inbound_streams,
SSM_Distribution::mdi_generateStartTSN());
```

```
        /* retreive a-side source addresses from message */
        supportedTypes = SSM_Distribution::mdi_getSupportedAddressTypes();

        nrAddresses = SSM_ChunkHandler::ch_IPaddresses(initCID, supportedTypes,
rAddresses, &peerSupportedTypes, &last_source);


        if ((supportedTypes & peerSupportedTypes) == 0)
            error_log(ERROR_FATAL, "BAKEOFF: Program error, no common address types in
sctlr_init()");

        /* enter variable length params initAck */
        SSM_Distribution::mdi_readLocalAddresses(lAddresses, &nlAddresses, &last_source, 1,
peerSupportedTypes,TRUE);
        /* enter local addresses into initAck */
        if (nlAddresses > 1)
            SSM_ChunkHandler::ch_enterIPaddresses(initAckCID, lAddresses, nlAddresses);

        /* append cookie to InitAck Chunk */
        SSM_ChunkHandler::ch_enterCookieVLP(initCID, initAckCID,
                        SSM_ChunkHandler::                    ch_initFixed(initCID),
SSM_ChunkHandler::ch_initFixed(initAckCID),
                SSM_ChunkHandler::ch_cookieLifeTime(initCID), 0, /* tie tags are both zero */
                        0, lAddresses, nlAddresses, rAddresses, nrAddresses);
--------------------------------------------------------------------------------
```

위의 코드에서 ch_makeinitAck는 파라메터로 tag값 나의 recv window size 내가 사용할 instream 과 outstream 사이즈를 가지고 init을 보낸 clinet쪽으로 전송할 chunk를 생성한다.
그후 enterCookieVLP함수를 통하여 쿠키 정보를 삽입하여 전송을 한다.

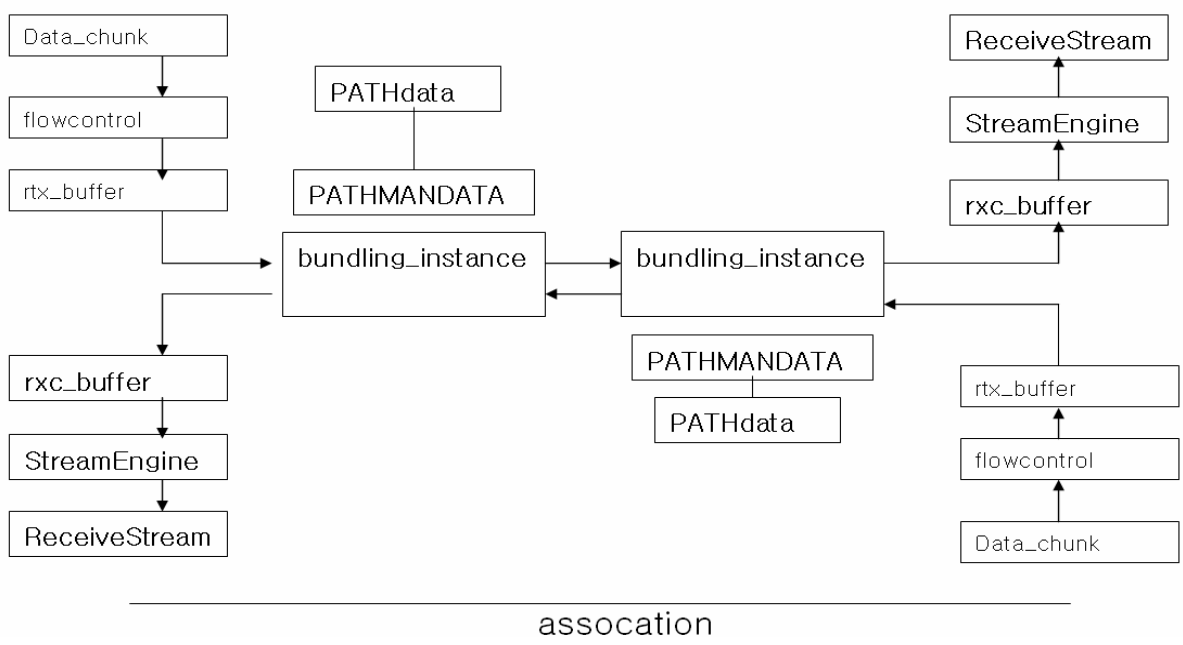
다음의 단계는 State Cookie를 생성하기 위해서 취해야 하는 단계이다.

1) 수신된 INIT와 나가는 outgoing INIT ACK CHUNK로부터 연결 TCB 이용 정보를 생성한다
2) TCB안에 그 날의 현재 시간과 프로토콜 파라미터 Valid.Cookie.Life에 대한 생명주기를 설정한다.
3) TCB로부터 TCB 재 생성을 위해 필요한 정보의 하부 항목을 수집하고 인지하며 이 정보의 항목과 비밀 키를 이용하여 MAC를 생성한다
4) 정보의 세부항목과 MAC의 결과와 조합하여 State Cookie를 생성한다.

State Cookie 파라미터를 가진 INIT ACK의 송신 후에 송신자는 resource Attack막기 위해 TCB와 그 연결과 관련된 어떤 다른 local 자원을 지워야 한다.
MAC를 생성하기 위해 사용되는 Hash method는 INIT Chunk의 수신자를 위한 정확히는 사소한 문제이다. MAC의 이용은 Denial Service Attack 공격을 막기 위한 필수 항목이다. 비밀키는 임의의 몇 가지 정보를 임의적으로 제공해야만 한다. 즉 비밀 키는 적당히 자주 변경 되어야만 하고 State Cookie 안의 생명 주기는 MAC의 유효성을 검증 하기 위하여 이용되는 Key를 결정하는 데 사용된다.
어떤 구현은 상호 운용성을 보장하기 위하여 가능한 적게 cookie를 만들어야 한다.

6. User Data Transfer

lib코드에서 데이터 전송 측에 관련된 구조체는 data_chunk, flowcontrol, rtx_buffer, bundling_instance이다. 응용에서 데이터 전송을 수행서 data_chunk를 구성하고 현재 데이터를 재전송을 위한 임시 버퍼인 rtx_buffer에 삽입 하여 둔다. 그후 bundling_instance에 max mtu안에서 bundling 을 하고 전송을 수행하게 된다. 그 후 정상적으로 패킷이 도착 하였다는 sack packet이 오면 rtx_buffer에 보관했던 data_chunk를 지우고 그렇지 않고 time_out이나 sackpacket에 gap이나 dup packet이 있으면 해당 전송 tsn을 rtx_buffer에서 복사해 다시 전송을 수행한다.



데이터 전송은 반드시 ESTABLISHED, SHUTDOWN-PENDING, 그리고 SHUTDOWN-RECEIVED 상태 상에서 일어나야 한다. 다만 COOKIE-WAIT 상태에서 outbound COOKIE ECHO chunk와 함께 bundle되도록 허용된 DATA chunk들을 받을 때는 예외이다. DATA chunk들은 반드시 ESTABLISHED, SHUTDOWN_PENDING, SHUTDOWN_SENT 내의 규칙들에 따라 수신되어야 한다. CLOSED 상태에서 수신된 DATA chunk는 out of the blue이고 8.4(Handle "out of the blue" Packets)에 의해 처리되어야 한다. 다른 상태에서 수신된 DATA chunk는 discard되어야 한다. SACK는 반드시 ESTABLISHED, SHUTDOWN-PENDING, 그리고 SHUTDOWN-RECEIVED 내에서 처리 되어야 한다.

SCTP 수신자는 반드시 하나의 SCTP packet에 최소한 1500byte는 전송할 수 있어야 한다. 이것은 SCTP endpoint가 INIT나 INIT ACK 시 전송된 Initial a_rwnd에서 1500byte보다 작은 것을 가리켜서는 안 된다는 것을 의미한다. 효율적인 전송을 위해 SCTP는 작은 사용자 메시지들을 bundle하고 커다란 사용자 메시지들을 fragmentation 하기 위한 메카니즘들을 정의한다. 다음 다이어그램은 SCTP를 통한 사용자 메시지들의 흐름을 보여준다.

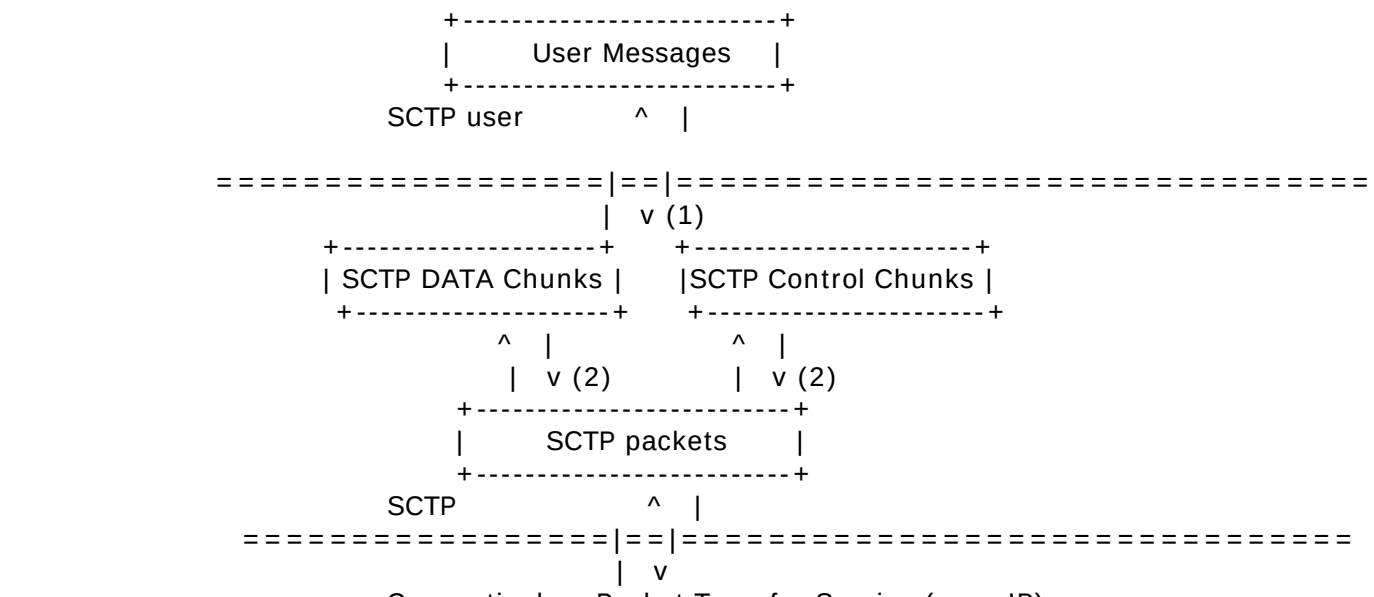


Figure 6: Illustration of User Data Transfer

이번 섹션에서 DATA chunk를 전송하는 endpoint를 "data sender"라고 하고 DATA chunk를 수신하는 endpoint를 "data receiver"라고 한다. data receiver는 SACK chunk를 전송할 것이다.

1) user message를 DATA chunk로 바꿀 때, 어떤 endpoint는 현재 연결에서의 path MTU보다 큰 user message를 다수의 DATA chunk들로 fragment할 것이다. data receiver는 보통 user(upper layer)로 전달하기 전에 DATA chunk들의 fragment된 메시지들을 재조합 할 것이다.
2) 현재의 path MTU의 범위를 초과하지 않는 한 다수의 DATA와 control chunk들을 전송하기 위하여 하나의 SCTP 패킷에서 bundle되어 질 수 있다.recevier는 패킷을 원래의 chunk들로 unbundle할 것이다. Control chunk들은 반드시 패킷 내에서 다른 DATA chunk들 보다 앞에 위치 해야 한다.

Fragmentation과 Bundling 메카니즘은 data sender에 의해 구현되는 OPTIONAL 기능이다. 하지만 이 메카니즘들은 data receiver에서는 반드시 구현되어야 한다. 다시 말해 한 endpoint는 반드시 bundle되거나 fragment된 데이터를 받아 처리해야 한다.

----------streamengine.cpp

```
/* calculate nr. of necessary chunks -> use fc_getMTU() later !!! */
      numberOfSegments = byteCount / SCTP_MAXIMUM_DATA_LENGTH;
      residual = byteCount % SCTP_MAXIMUM_DATA_LENGTH;
      if (residual != 0) {
            numberOfSegments++;
      } else {
            residual = SCTP_MAXIMUM_DATA_LENGTH;
      }

      if (maxQueueLen > 0) {
        if ((numberOfSegments + SSM_Flowcontrol::fc_readNumberOfQueuedChunks()) >
maxQueueLen) return SCTP_QUEUE_EXCEEDED;
      }

      for (i = 1; i <= numberOfSegments; i++)
      {
            cdata = (chunk_data_struct *)malloc(sizeof(chunk_data));
            if (cdata == NULL) {
```

```
            /* FIXME: this is unclean, as we have already assigned some TSNs etc, and
             * maybe queued parts of this message in the queue, this should be cleaned
             * up... */
            return SCTP_OUT_OF_RESOURCES;
        }

        dchunk = (SCTP_data_chunk*)cdata->data;

        if ((i != 1) && (i != numberOfSegments))
        {
            dchunk->chunk_flags = 0;
            bCount = SCTP_MAXIMUM_DATA_LENGTH;
            event_log (VERBOSE, "NEXT FRAGMENTED CHUNK -> MIDDLE");
        }
        else if (i == 1)
        {
            dchunk->chunk_flags = SCTP_DATA_BEGIN_SEGMENT;
            event_log (VERBOSE, "NEXT FRAGMENTED CHUNK -> BEGIN");
            bCount = SCTP_MAXIMUM_DATA_LENGTH;
        }
        else if (i == numberOfSegments)
        {
            dchunk->chunk_flags = SCTP_DATA_END_SEGMENT;
            event_log (EXTERNAL_EVENT, "NEXT FRAGMENTED CHUNK -> END");
            bCount = residual;
        }

    dchunk->chunk_id = CHUNK_DATA;
    dchunk->chunk_length = htons ((unsigned short)(bCount +
FIXED_DATA_CHUNK_SIZE));
    dchunk->tsn = htonl (0);
    dchunk->stream_id = htons (streamId);
    dchunk->protocolId = protocolId;

    if (unorderedDelivery)
    {
        dchunk->stream_sn = 0;
        dchunk->chunk_flags += SCTP_DATA_UNORDERED;
    }
    else
    {   /* unordered flag not put */
        dchunk->stream_sn = htons ((unsigned
short)(se->SendStreams[streamId].nextSSN));
        /* only after the last segment we increase the SSN */
        if (i == numberOfSegments) {
            se->SendStreams[streamId].nextSSN++;
            se->SendStreams[streamId].nextSSN =
se->SendStreams[streamId].nextSSN % 0x10000;
        }
    }

    memcpy (dchunk->data, bufPosition, bCount);
    bufPosition += bCount * sizeof(unsigned char);

    event_logiii (EXTERNAL_EVENT, "======> SE sends fragment %d of chunk
(SSN=%u, SID=%u) to FlowControl <======",
                  i, ntohs (dchunk->stream_sn),ntohs (dchunk->stream_id));

        if (!se->unreliable) lifetime = 0xFFFFFFFF;

        result = SSM_Flowcontrol::fc_send_data_chunk (cdata, destAddressIndex, lifetime,
dontBundle, context);
```

```
            if (result != SCTP_SUCCESS) {
                error_logi (ERROR_MINOR, "se_ulpsend() failed with result %d", result);
                /* FIXME : Howto Propagate an Error here - Result gets overwritten on next
Call */
                retVal = result;
            }
        }
-----------------------------------------------------------------streamengine.cpp
```

Ulp_send function에서 데이터를 받아들인 후 데이터가 mtu보다 크면 segmentation을 수행하게 된다. Segmentation시 첫번째 데이터는 SCTP_DATA_BEGIN_SEGMENT, 중간데이터는 0을 마지막 데이터는 SCTP_DATA_END_SEGMENT를 chunk_flags에 삽입하여 end쪽에서 recv수행시 재조합된다.

6.1 Transmission of DATA Chunks (DATA chunk의 전송)

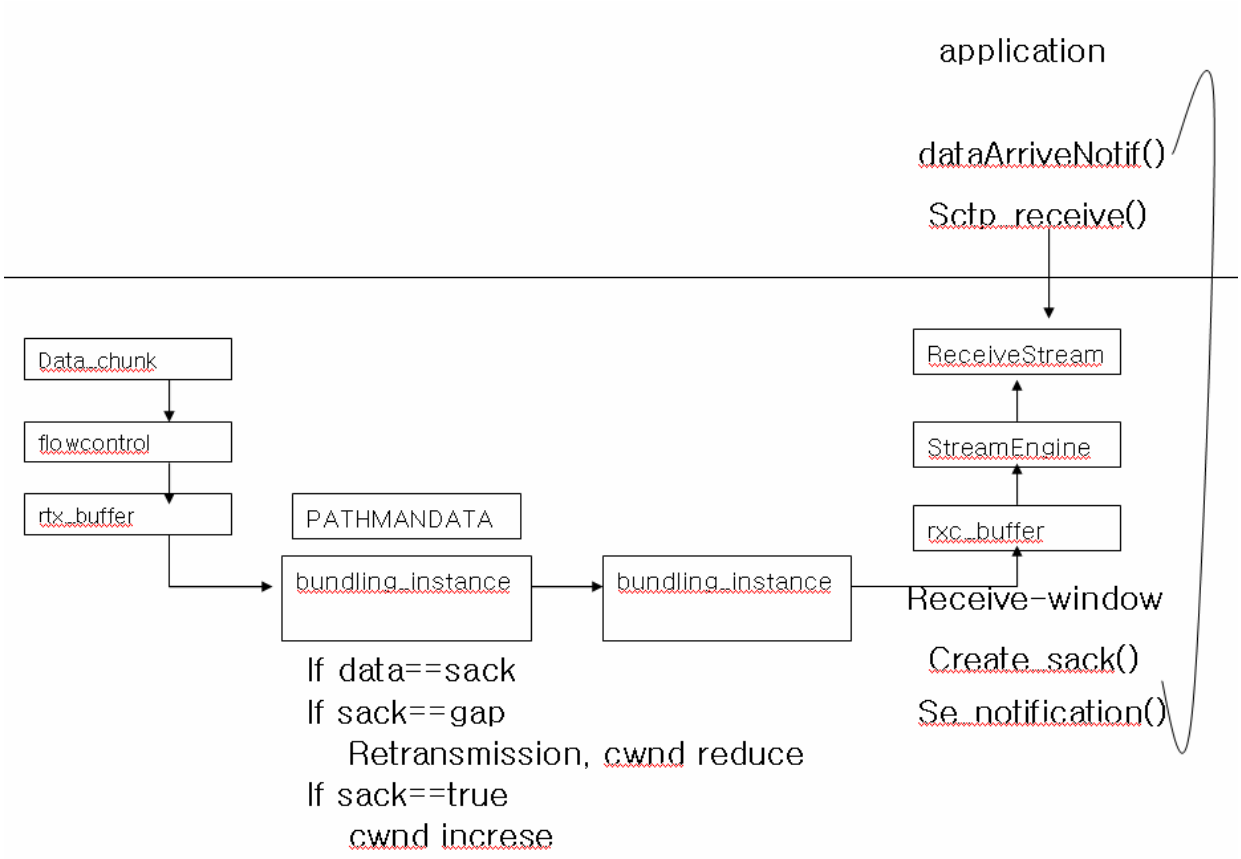
이 문서는 destination transport address마다 단일 재전송 타이머가 있는 것처럼 규정한다. 하지만 구현은 각 DATA chunk마다 재전송 타이머를 갖도록 해야 한다.
다음 일반적인 규칙들은 outbound DATA chunk의 전송 및 재전송을 하는 data sender에 반드시 적용되어야 한다.

A) 어느 주어진 시간에, data sender는 peer의 rwnd가 그 peer의 버퍼 공간이 없음을 가리키고 있다면 어느 destination transport address로든 새로운 데이터를 전송해서는 안 된다. 그러나 만약 cwnd가 전송이 가능이 허락 된다면 rwnd에 상관 없이 data sender는 하나의 Data Chunk를 전송 할 수 있다. 이러한 규칙은 data receiver로부터 data sender에 전송되는 SACK이 손실 되었기 때문에 잃어버린 rwnd의 변화를 data sender가 판별하기 위하여 사용되어진다.
B) 어느 주어진 시간에, sender는 cwnd나 transport address에 처리되지 않은 많은 byte의 data들을 가지고 있다면 주어진 transport address로 새로운 데이터를 전송해서는 안 된다.
C) Sender가 전송을 해야 할 시간이 됐을 때, 새로운 DATA chunk를 보내기 전에 sender는 먼저 (현재 cwnd에 의해 제한 되어) 재전송을 위해 표기된 처리되지 않은 Dada chunk를 전송해야 한다.
D) Sender는 규칙 A와 B의 규칙이 허락 하는 한 새로운 DATA chunk들을 많이 보낼 수 있다.

```
-----------------------------------------------------------------flowcontrol.cpp
 if (peer_rwnd == 0 && fc->one_packet_inflight == TRUE) {     /* section 6.1.A */
        event_log(VERBOSE, "NOT SENDING (peer rwnd == 0 and already one packet in flight ");
        event_log(VERBOSE,   "###################   ->   Returned   in   fc_check_for_txmit
###################");
        return 1;
    }
-----------------------------------------------------------------flowcontrol.cpp
```

전송되도록 허용 된 다수의 DATA chunk는 단일 패킷 내에 bundle될 수 있다. 더욱이, 재전송 될 DATA chunk들은 패킷 크기가 현재Path MTU를 초과하지 않는 한 새로운 DATA chunk들과 함께 bundle될 수 있다.

6.2 Acknowledgement on Reception of DATA Chunks (DATA chunk의 수신에 대한 Acknowledgement)



Data packet을 수신 시 sack packet을 발생을 해야 한다. sack정보에는 현재 정상적으로 수신된 packet의 cumulative ack 값이 설정이 되고 혹은 중복되거나 , gap이 잇는 패킷이 발생을 한다면 그것에 대한 정보를 sack정보안에 삽입 하여 전송을 수행한다. 위의 그림은 association이 맺어 지고 tcb설정이 이 수행된 후에 데이터 전송 시 전송측에서 max mtu안에서 bundling된 데이터를 수신측으로 전송을 수행한다. 그 후 받은 데이터를 rxc_buffer에 삽입을 하고 현재 전송된 데이터가 tsn값이 올바른 데이터라면 gap과 dup값을 표시하지 않고 streamengin에서 de-segmentation을 수행한후 현재 전송된 stream쪽으로 데이터를 넘긴다. 그 후 se_notification호출을 통해 instance설정 시 등록해 놓은 Data_notification 처리를 수행한다. Data_notification에서 sctp_receive 함수를 call을 하고 stream에서 데이터를 읽어 온다. 그 후 읽어 온 데이터 사이즈만큼 현재 receive윈도우를 증가시킨다.

6.2.1 Processing a Received SACK (수신된 SACK의 처리)

Endpoint가 받은 각각의 SACK는 a_rwnd값을 포함한다. 이 값은 SACK를 전송하는 당시 data receiver의 총 수신버퍼 공간에 남아있는 비어있는 버퍼의 용량을 나타낸다. data sender는 a_rwnd를 이용하여 peer의 수신버퍼 공간를 나타내는 것을 개발 할 수 있다.

SACK를 처리할 때 data sender가 account에 take해야 하는 문제들 중 하나는 순서가 맞지않는 SACK가 수신될 수 있다는 것이다. data receiver에 의해 송신된 SACK는 보다 먼저 SACK를 pass할 수 있고 data sender에 먼저 수신될 수 있다. (?). 만약 순서가 맞지않는 SACK가 수신되면 data sender는 peer의 수신버퍼 공간의 옳지않은 view를 개발할 수 있다.

out-of-order SACK를 감지하는데 사용될 수 있는 명확한 식별자가 없으면, data sender는

SACK가 새로운 것인지를 판별하기 위해 발견적 방법(heuristics)을 사용해야만 한다.

rbu_rcvDatagram()에서 case CHUNK_SACK에서 sack chunk 수신 시 데이터를 처리한다.

----------ssm_bundling.cpp

```
case CHUNK_SACK:
        event_log(INTERNAL_EVENT_0, "******************  Bundling received SACK chunk");
        SSM_Reltransfer::rtx_process_sack(address_index, chunk, len);
        break;
```

----------ssm_bundling.cpp

SSM_Reltransfer::rtx_process_sack(address_index, chunk, len);함수는 sack packet을 받았을 때의 구제적인 동작을 명시하고 있다. 수신된 SACK에서 Cumulative TSN Ack와 Gap Ack Block에서 a_rwnd 값을 이용하여 rwnd를 계산하기 위하여 endpoint는 다음 규칙들을 사용한다.

A) association이 이루어지면, endpoint는 INIT이나 INIT ACK에 명시된 peer의 Advertised Receiver Window Credit (a_rwnd)로 rwnd를 초기화한다.

B) DATA chunk가 peer로 전송되면, endpoint는 peer의 rwnd로부터 chunk의 data size를 뺀다.

C) DATA chunk가 재전송을 위해 mark되면, rwnd에 그 chunk의 data 크기만큼 더한다.

D) SACK가 도착하면, endpoint는 다음과 같이 수행한다.

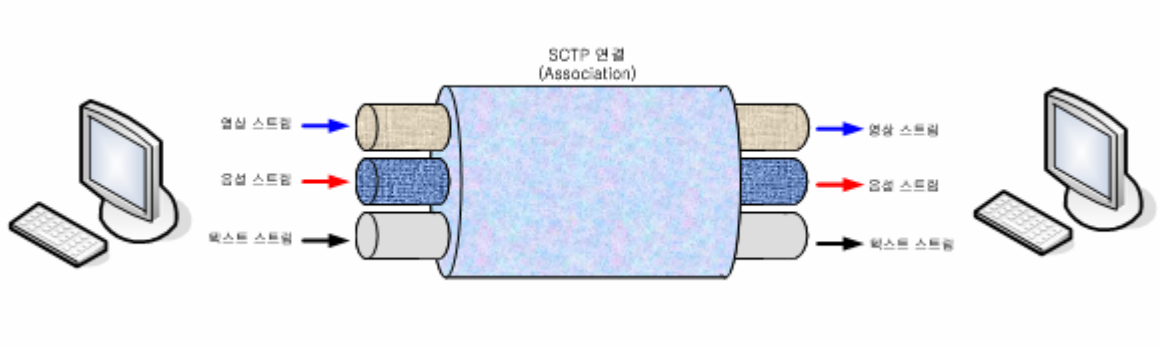
i) 만약 Cumulative TSN Ack가 Cumulative TSN Ack Point보다 작으면, SACK를 drop한다. Cumulative TSN Ack 가 단순히 증가하고 있으면, Cumulative TSN Ack가 Cumulative TSN Ack Point보다 작은 SACK는 out-of-order SACK를 가리킨다.

ii) Cumulative TSN Ack와 Gap Ack Block를 처리한 후 rwnd를 새롭게 수신된 a_rwnd 에서 아직 처리되지 않은 바이트 수만큼 뺀 값과 같도록 설정한다.

iii) 만약 SACK에 Gap Ack Block를 통해 전에 acknowledge되었던 TSN이 빠져있다면, 재전송 가능한 DATA chunk와 일치되도록 mark한다.

7. Sample Application

Sctplib를 사용한 응용프로그램



7.1 프로그램 동작 과정

sctp 서버에 접속하면 클라이언트는 화상데이터와 음성데이터를 각각 하나의 association에서 multi-streaming을 통해 전송을 받는다. 현재 usb-cam camera로부터 받아오는 영상은 0번 stream으로 전송을 받고 마이크로부터 받아 들이는 데이터는 3번 스트림을 사용하여 전송을 받게 되어있다.

7.2 프로그램 구성

Cplsctp.lib
Sctpserver.exe
Sctpclient.exe

7.3 실행방법

현재 윈도우 xp에서는 방화벽이 기본적으로 설정 되어 있기에 방화벽을 꺼주고 서버와 클라이언트를 실행한다. 서버는 usb카메라와 마이크가 필요하기에 각각의 장치를 연결 후 서버를 실행 후 시작 버튼을 누르면 서버는 동작 한다. 클라이언트에서는 소스에서 목적지 address를 설정해 주어야 하는데 destination_address=" 서버주소" 에 설정을 하여 다시 컴파일 후 접속 과정을 수행하면 정상 작동할 것이다.