SCTPLIB 코드분석: Revised

2006년 8월

경북대학교 통신프로토콜연구실

이동화 (yiffie9819@gmail.com)

요 약

SCTPLIB는 RAW socket을 이용해 user level에서 구현한 SCTP 프로토콜 구현코드이다. 본 문서에서는 rfc2960문서에 제안된 stream control transmission protocol을 어떻게 user level에서 SCTPLIB로 구현되었는 지와 추가로 기존 SCTPLIB에 빠져 있었던 Dynamic Address Reconfiguration 관련 부분을 분석하였다. 데이터의 흐름을 중심으로 분석한 자료이기에 user level에서 SCTP 프로토콜의 분석 및 연결 지향형의 새로운 프로토콜의 구현 시 도움이 되리라 생각된다.

목 차

1. 서론

본 문서는 SCTPLIB 코드 분석 및 응용과의 관계, SCTPAPI와 응용과의 관계에서 함수 호출 순서에 따라 적용되는 매커니즘 분석 및 기존 SCTPLIB에 Dynamic Address Reconfiguration 모듈의 추가에 대한 ASCONF Chunk 분석, Procedure 분석에 관한 내용이다. SCTPLIB는 RAW socket을 이용하여 user level에서 구현한 SCTP STACK이다. SCTPLIB RFC2960에 정의 되어 있는 SCTP의 특징들을 수렴하고 있고 각각의 SCTP의 특징은 문서를 통해 참조할 수 있다.

2. Introduction of SCTPLIB

2.1 Architecture view of SCTP

SCTP는 SCTP 사용자와 IP와 같은 비 연결 지향적인 패킷 network 서비스와의 사이에 있는 하나의 계층으로 보여지고 SCTP가 IP의 위에서 동작한다는 것을 추측 할 수 있다. SCTP에 의해서 제공되는 기본적인 서비스는 SCTP의 User들간 user message의 신뢰성 있는 전송이다. SCTP는 두 개의 SCTP endpoint 시스템 사이에 있는 하나의 연결 환경에서 서비스를 수행한다.

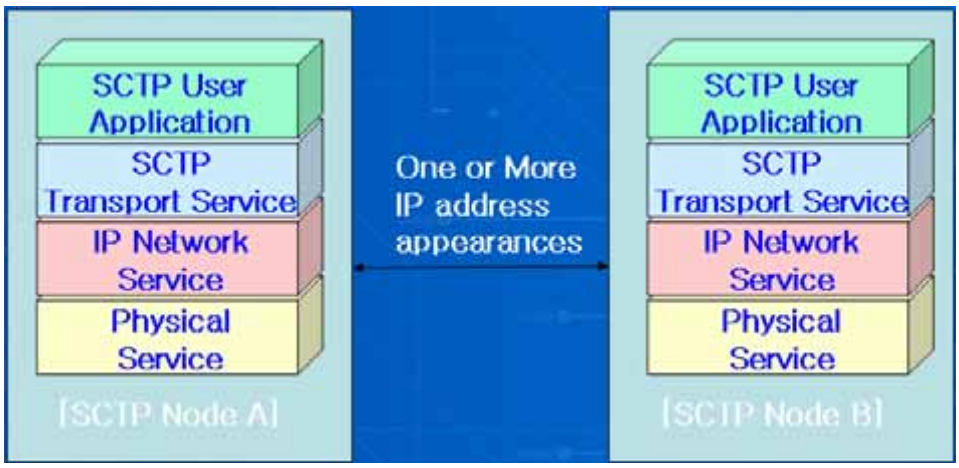


Figure 1: An SCTP Association

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.

3. SCTPLIB의 구성

3.1 Code of SCTPLIB

17개의 헤더 파일과 16개의 c파일이 있다. 각각의 파일의 특징을 뒷장에서 설명을 하도록 하겠다.

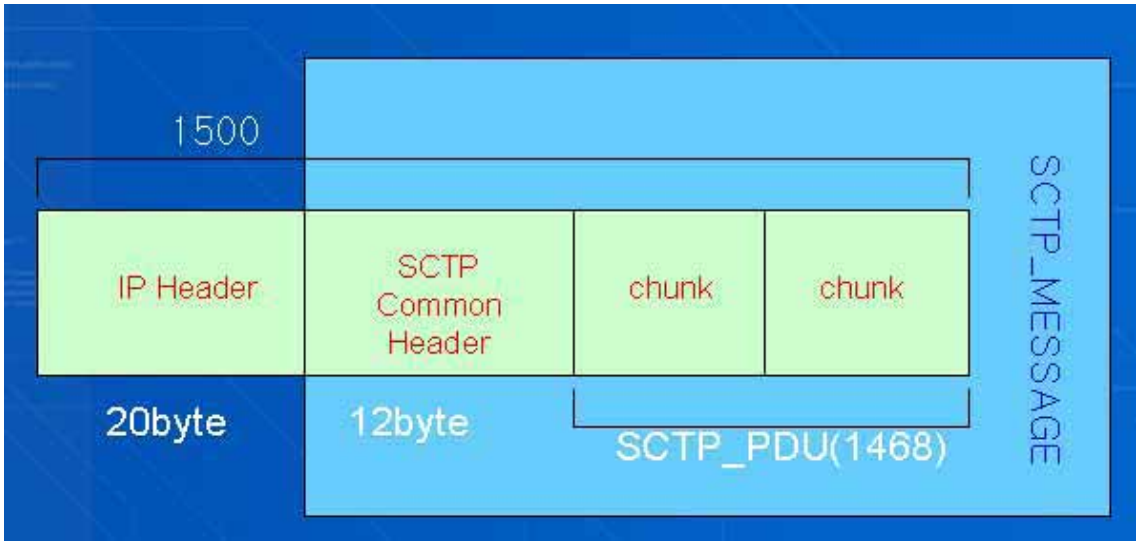
FILE	DESCRIPTION
Adaptation.c Adaptation.h	Ulp To SCTP Layer
Auxiliary.c Auxiliary.h	데이터 무결성 검증
Rbundling.c Bundling.h	Bundling과 Bundling된 데이터 re-bundling
ChunkHandler.c ChunkHandler.h	Make Chunk and Chunk 무결성 확인
Distribution.c Distribution.h	Packet Process interface
Errorhandler.c Errorhandler.h	Error 처리 관련 루틴
Flowcontrol.c Flowcontrol.h	흐름제어 관련, 전송시 cwnd, outbound packer, rcwnd 관리
Md5.c Md5.h	Md5 관련 처리
Pathmanagement.c Pathmanagement.h	Multi-homing 관련, Heartbeat 발생, cwnd적용
Recvctrl.c Recvctrl.h	데이터 수신 관련 버퍼, TSN값만 가지고 처리, SACK발생
Reltransfer.c Reltransfer.h	SACK 수신 시 처리, 재전송, cwnd적용
SCTP_control.c SCTP_control.h	Association state 관리
Streamengine.c Streamengine.h	Datastream을 처리하는 모듈을 구현
Timer_list.c Timer_list.h	Timer event의 linked list를 구현
Global.c Global.h	SCTP 프로젝트에서 알고 있어야만 하는 모듈에 대해서 정의
Sctp.h	SCTP 라이브러리에서의 API를 정의

Message.h	SCTP에서 사용되는 모든 메시지 구조체 정의

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 Raw소켓의 생성코드이다. sctpsock로부터 받아 들이는 데이터는 ip패킷을 포함한 sctp데이터이다. 위와 같은 간단한 함수만으로도 sctp 프로토콜을 관리할 수 있는 socket의 생성이 가능하다. 소켓 디스크립터 sctpsock을 통해 데이터를 받을 수 있는데 거기서 받아 오는 데이터는 IP 헤더를 포함한 SCTP프로토콜 데이터이다.



데이터의 수락 시 위의 그림에서 보는 것과 같이 rawsocket을 통해 들어오는 데이터는 IP 헤더를 포함한 데이터이며 LIB코드에서 IP헤더를 떼고, sctp_message단위로 데이터처리를 수행한다.

3.2 Functional view of SCTP

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

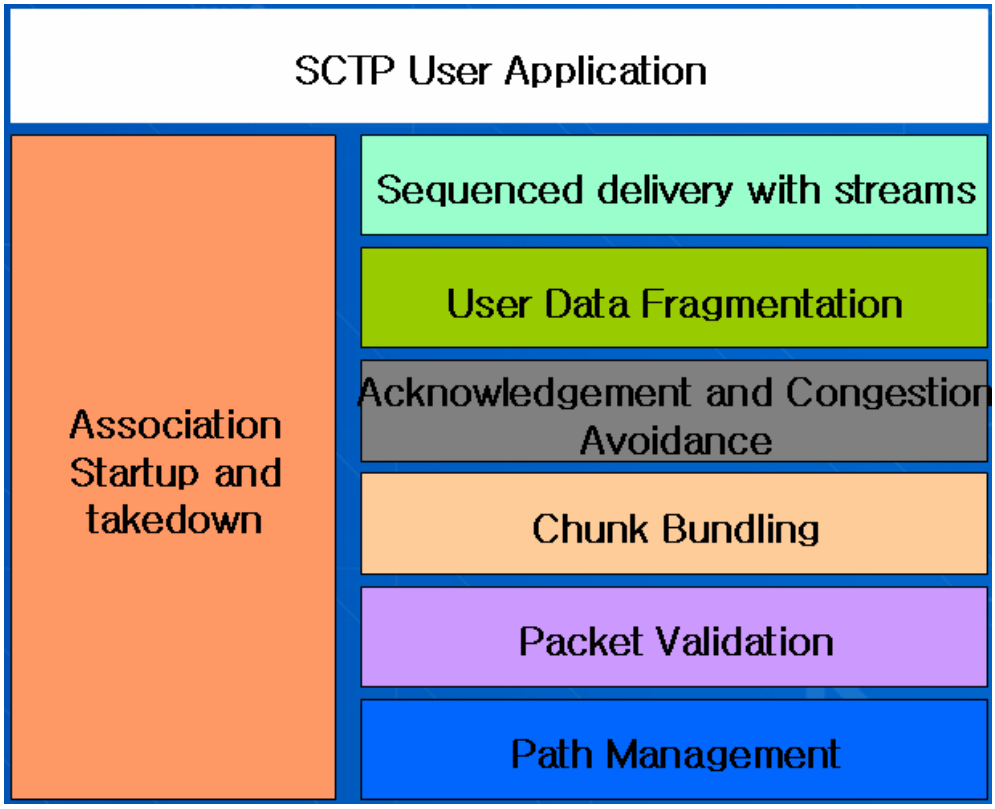


Figure 2 : Function View of the SCTP Transport Service

3.2.1 Association Startup and Takedown

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 endpoint가 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.

3.2.2 Sequence Delivery within Stream

SCTP안에서 사용되는 " stream" 이란 용어는 같은 stream 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 가능한 stream의 수를 연결이 진행되는 시간에 규정할 수 있다. 이 stream의 수는 다른 endpoint 와 협상을 하게 된다. 내부적으로 SCTP는 SCTP 사용자에 의해 SCTP에 전달되는 각 메시지들에 stream sequence number을 부여한다.

수신 측에서 SCTP는 메시지들이 주어진 stream 내에서 순서적으로 SCTP 사용자에게 전달하는 것을 보장해야 한다. 하지만 하나의 stream이 다음 순서의 user 메시지를 기다리면서 blocked 되어 있는 동안, 다른 stream으로부터 메시지 전달은 처리될 수 있다.

3.2.3 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. 수신 측에서 단편화는 SCTP user에 전달되기 전에 완전한 메시지로 재 조합된다.

3.2.4 Acknowledgement and Congestion Avoidance

SCTP는 단편화 되거나 단편화 되지 않은 메시지의 각 user 데이터에 TSN(Transmission Sequence Number)을 할당한다. 그 TSN은 stream 레벨에서 할당된 SSN에 독립적이다. 데이터를 수신한 endpoint는 비록 순서에 갭이 발생 했다고 할 지라도 수신한 모든 TSN을 알린다. 이러한 방법은 신뢰성 전송이 순서화된 stream 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 사용되어지는 유사한 혼잡제어에 의해서 조절되어진다.

3.2.5 Chunk Bundling

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 bundling 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 시간 동안에는 비록 SCTP 사용자가 bundling 하지 않도록 요청했다 할지라도 bundling 을 수행한다. User 의 bundling 비활성은 전송 전 짧은 시간 동안에 delay가 될 수 있게 하는 SCTP 구현에 영향을 준다. User 계층에서 bundling 하는 것을 비활성 시켰다 하더라도 small delay는 혼잡시간이나 재전송 동안에는 bundling이 되는 것을 막는다.

3.2.6 Packet Validation

필수 항목인 Verification Tag와 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 패킷의 송신자에 의해 설정 되어져야만 한다. 유효하지 않은 Adler-32 checksum과 같이 수신한 패킷은 그 패킷을 폐기한다.

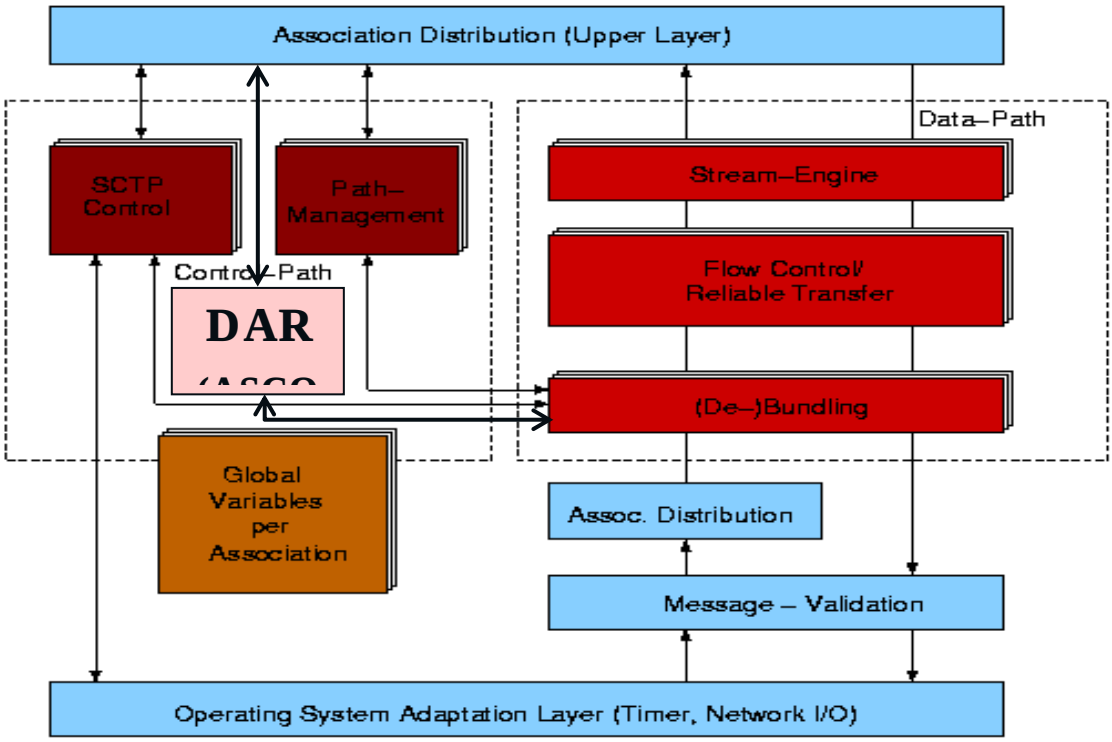
3.2.7 Path management

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 수행되지 않는다.

3.3 SCTPLIB 구조도



위의 그림은 SCTPLIB에서의 전체 흐름을 나타내는 구조도이다.

SCTPLIB에서 호출 되는 기본적인 함수들은 다음과 같다.

Sctp_initlibary()

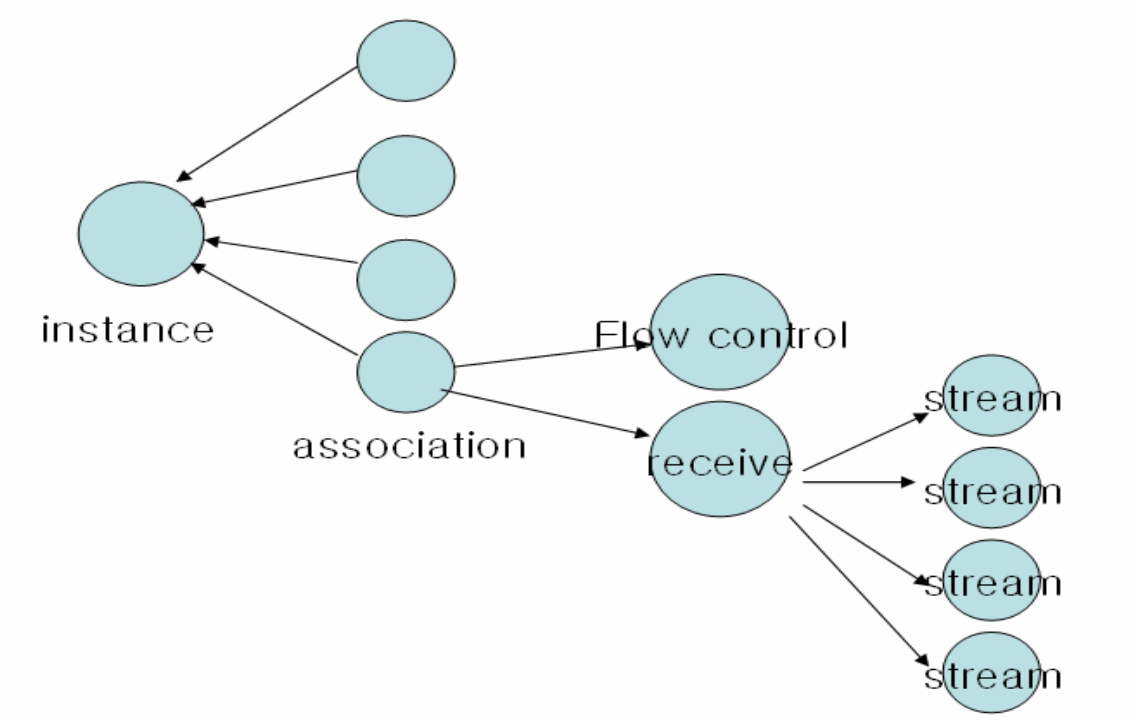
Sctp_registerinstance()

Sctp_eventloop()

Sctp_associate()

위의 네 가지 함수를 호출 함으로써 기본적인 SCTP동작을 수행 할 수 있다. 먼저 SCTPLIB 에서는 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 SCTP 형태의 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 생성

을 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 flow control과 관련된 구조체 들과 최종 association과 연결 맺어지는 in-stream buffer out-stream buffer 구조체의 생성을 일컫는다.

4. Interface with Upper Layer

4.1 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 함수들이다. 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 이용 되어진다. 일단 SCTP가 초기화가 되면 ULP는 이 primitive를 재호출 하는 것 없이 다른 endpoint함께 직접적으로 통신 할 수 있다. 첫 번째 파라메터는 자신의 포트를 가르킨다 만약, 서버 측이 아니라 클라이언트 측이라면 RANDOM PORT를 사용하게 된다. 두 번째, 세 번째 파라메터는 자신이 사용하고자 하는 STREAM 숫자이다.이것은 ASSOCIATION 설정 시 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][SCP
_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 stream의 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 주 경로 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 전달 되어야 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 데이터를 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 데이터의 크기

Optional attributes

Context(Option) : 사용자 메시지의 전송이 실패했을 경우 ULP에게 실패 notification을 전송할 때 포함되는 선택적 32-bit integer

stream id(Option) : 데이터 전송 시 사용된 stream identification

life time(Option) : 사용자 메시지의 유효한 시기를 결정한다. life time이 만료되면 사용자 데이터는 SCTP에 전송되지 않을 것이다. 이 파라미터는 사용하지 못하는 사용자 메시지 전송을 위한 노력을 피하기 위하여 사용되어 질 수 있다. 만약 사용자 데이터를 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 첫 번째 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 데이터의 최대 크기

stream id(Option) : data를 수신하기 위한 stream을 가리킨다.

stream sequence number() : 송신 측에 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.

4.2 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. 구현 상 SCTP와 ULP간 interface는 separate socket나 error channel을 통해 구현할 수 있다.

IMPLEMENTATAION NOTE : 몇몇 경우에 이것은 분리된 소켓이나 에러 채널을 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 stream을 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 데이터를 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 stream의 수

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 : 보내지 않고 알려지지 않은 data를 복구하기 위하여 이용되어지는 식별자

last-acked : peer endpoint에 의해 마지막으로 acked된 TSN

last-sent : peer endpoint에 마지막으로 보내진 TSN

5. ASSOCIATION INITIALIZATION

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

5.1 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.

Association의 상태에 대한 구현은 association 구조체의 멤버변수인 sctp_controldata구조체에서 가지고 있다. 각각의 asociation상태에 따라 control data의 멤버변수인 association_state 값을 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-timer를 재가동한다. 그리고 이러한 작업은 Max.Init.Retransmits 횟수가 될 때까지 반복 실행해야만 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.

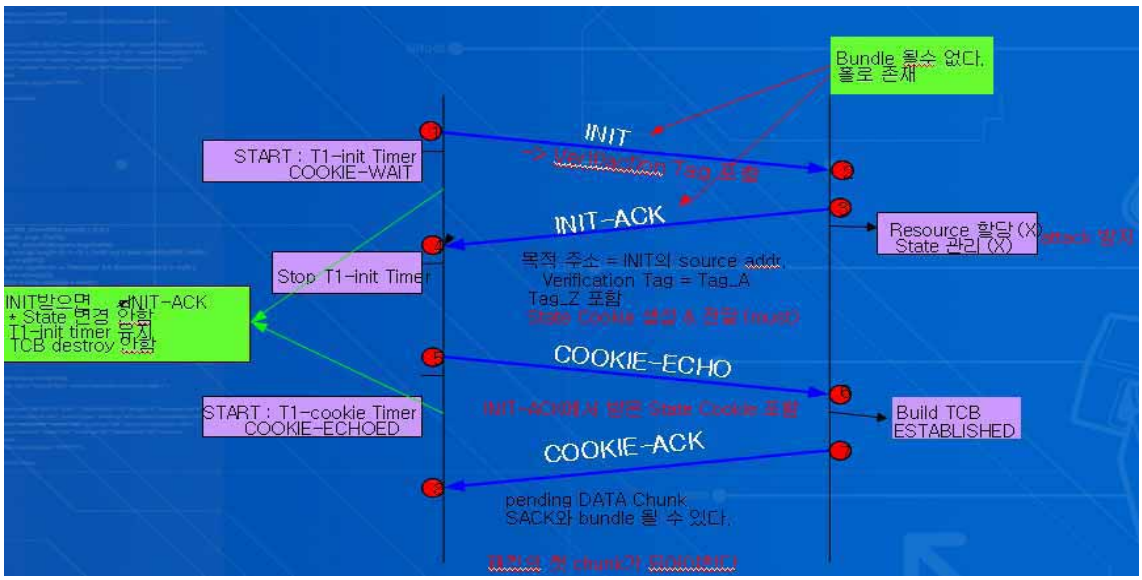
5) SHUTDOWN-RECEIVED 상태에서 endpoint는 SCTP User로부터 새로운 요청을 받을 수 없다.

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.

5.2 Normal Establishment of an association

초기화 단계는 연속된 단계로 이루어져 있으며 SCTP endpoint A에서 다른 SCTP endpoint Z에게 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 설정 한다. 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 첫 번째 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 SCTP 사용자에게 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.

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

각각의 chunk에 대해서는 chunk를 수락하였을 때 association의 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 받았을 때의 데이터 처리이다.

```
if ((localData = (SCTP_controlData *) 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). */
        mdi_writeLastInitiateTag(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(ch_noOutStreams(initCID), mdi_readLocalInStreams());
```

```
outbound_streams = min(ch_noInStreams(initCID), mdi_readLocalOutStreams());
/* fire back an InitAck with a Cookie */
initAckCID = ch_makeInitAck(mdi_generateTag(),
                                    mdi_getDefaultMyRwnd(),
                                    outbound_streams,
                                    inbound_streams, mdi_generateStartTSN());

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

nrAddresses = 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 */
mdi_readLocalAddresses(lAddresses, &nlAddresses, &last_source, 1,
                              peerSupportedTypes,TRUE);
/* enter local addresses into initAck */
if (nlAddresses > 1)
     ch_enterIPaddresses(initAckCID, lAddresses, nlAddresses);

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


process_further = 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 */
            bu_put_Ctrl_Chunk(ch_chunkString(initAckCID),NULL);
        }
        bu_sendAllChunks(NULL);
        bu_unlock_sender(NULL);
        ch_deleteChunk(initAckCID);
        event_log(INTERNAL_EVENT_1, "event: initAck sent");
return return_state;
}
}
```

Init_chunk를 받았을 상황은 현재 association이 설정되지 않은 상황이고 현재 association에 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 cookie정보를 ch_enterCookieVLP()를 통하여 init_ack 에 삽입한다. 이 과정이 순조롭게 끝나면 bu_put_Ctrl_Chunk()을 이용하여 번들링 준비된 buffer에 삽입하고, bu_sendAllChunks(NULL); 을 호출하여 전송한다.

다음은 init을 받은 후 INIT_ACK를 수행 할 때의 데이터 처리이다.

```
gboolean sctlr_initAck(SCTP_init * initAck)
{
                    .
                    .
                    .
    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 = ch_makeChunk((SCTP_simple_chunk *) localData->initChunk);

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

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

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

```
}

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

mdi_writeDestinationAddresses(dAddresses, ndAddresses);

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

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

peerSupportsPRSCTP = ch_getPRSCTPfromInitAck(initAckCID);


mdi_initAssociation(ch_receiverWindow(initAckCID), /* remotes side initial rwnd */
                        inbound_streams, /* # of remote output/local input streams
                                            */
                        outbound_streams, /* # of remote input/local output
                                             streams */
                        ch_initialTSN(initAckCID), /* remote initial TSN */
                        ch_initiateTag(initAckCID), /* remote init tag */
                        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... */
ch_chunkString(initCID),
/* free initChunk memory */
ch_forgetChunk(initCID);

cookieCID = ch_makeCookie(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);
    bu_unlock_sender(NULL);
    /* delete this association */

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


process_further = ch_enterUnrecognizedErrors(initAckCID,
                                             supportedTypes,
                                             &errorCID,
                                             &preferredPrimary,
                                             &preferredSet,
                                             &peerSupportsIPV4,
                                             &peerSupportsIPV6,
                                             &peerSupportsPRSCTP,
                                             &peerSupportsADDIP);


if (process_further == -1) {
    ch_forgetChunk(initAckCID);
    ch_deleteChunk(cookieCID);
    if (errorCID != 0) ch_deleteChunk(errorCID);
    bu_unlock_sender(NULL);
```

```
        if (localData->initTimer != 0) {
            sctp_stopTimer(localData->initTimer);
            localData->initTimer = 0;
        }
        mdi_deleteCurrentAssociation();
        mdi_communicationLostNotif(SCTP_COMM_LOST_FAILURE);
        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 *) ch_chunkString(cookieCID);
    /* populate tie tags -> section 5.2.1/5.2.2 */
    localData->local_tie_tag = mdi_readLocalTag();
    localData->peer_tie_tag = ch_initiateTag(initAckCID);


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


    ch_forgetChunk(cookieCID);
    ch_forgetChunk(initAckCID);

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

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

    bu_sendAllChunks(&index);
```

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

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

        state = COOKIE_ECHOED;

        if (localData->initTimer != 0) sctp_stopTimer(localData->initTimer);
        /* start cookie timer */
        localData->initTimer = 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 stream 개수 체크하고 mdi_initAssociation()을 이용하여 association에 flowcontrol, stream engine등을 수행 할 수 있는 기반을 만든다.

Init_ack정보를 가지고 cookie_echo chunk를 생성하여 전송하고 현재 상태를 cookie_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 scu_associate()함수를 호출하게 된다.

scu_associate()함수를 살펴 보도록 하자.

```
void 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 *) 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 = ch_makeInit(mdi_readLocalTag(),
                              mdi_getDefaultMyRwnd(),
                              noOfOutStreams, noOfInStreams,
mdi_generateStartTSN());

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

        supportedTypes = mdi_getSupportedAddressTypes();

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

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

        if (withPRSCTP) {
            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

#ifdef HAVE_IPV6
        if (supportedTypes == SUPPORT_ADDRESS_TYPE_IPV6) {
```

```
            ch_enterSupportedAddressTypes(initCID, FALSE, TRUE, FALSE);
        } else if (supportedTypes == SUPPORT_ADDRESS_TYPE_IPV4) {
            ch_enterSupportedAddressTypes(initCID, TRUE, FALSE, FALSE);
        } else if (supportedTypes == (SUPPORT_ADDRESS_TYPE_IPV6 |
                    SUPPORT_ADDRESS_TYPE_IPV4)) {
                ch_enterSupportedAddressTypes(initCID, TRUE, TRUE, FALSE);
        } else
            error_log(ERROR_MAJOR, "CHECKME: Did not set correct SUPPORTED ADDR
                        TYPES parram");
#else
        ch_enterSupportedAddressTypes(initCID, TRUE, FALSE, FALSE);
#endif

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

        if (nlAddresses > 1)
            ch_enterIPaddresses(initCID, lAddresses, nlAddresses);


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

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

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

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

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

        localData->initTimer = 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 association구조체에 있는 control state가 CLOSED 상태일 때만 association을 맺기 위한 init_chunk 패킷을 생성할 수 있다. ch_makeInit()함수를 통해 서 init chunk를 만들고 bu_put_Ctrl_Chunk()함수는 init_chunk를 보내어질 bundling instance에 추가 하게 된다. 하지만 실제적으로 init을 보낼 때는 데이터는 번들링 되어 지지 않는다.

5.3.1 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.

다음의 단계는 State Cookie를 생성하기 위해서 취해야 하는 단계이다.

1) 수신된 INIT와 나가는 outgoing INIT ACK CHUNK로부터 연결 TCB 이용 정보를 생성 한다

2) TCB안에 그 날의 현재 시간과 프로토콜 파라미터 Valid.Cookie.Life에 대한 생명 주기를 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.

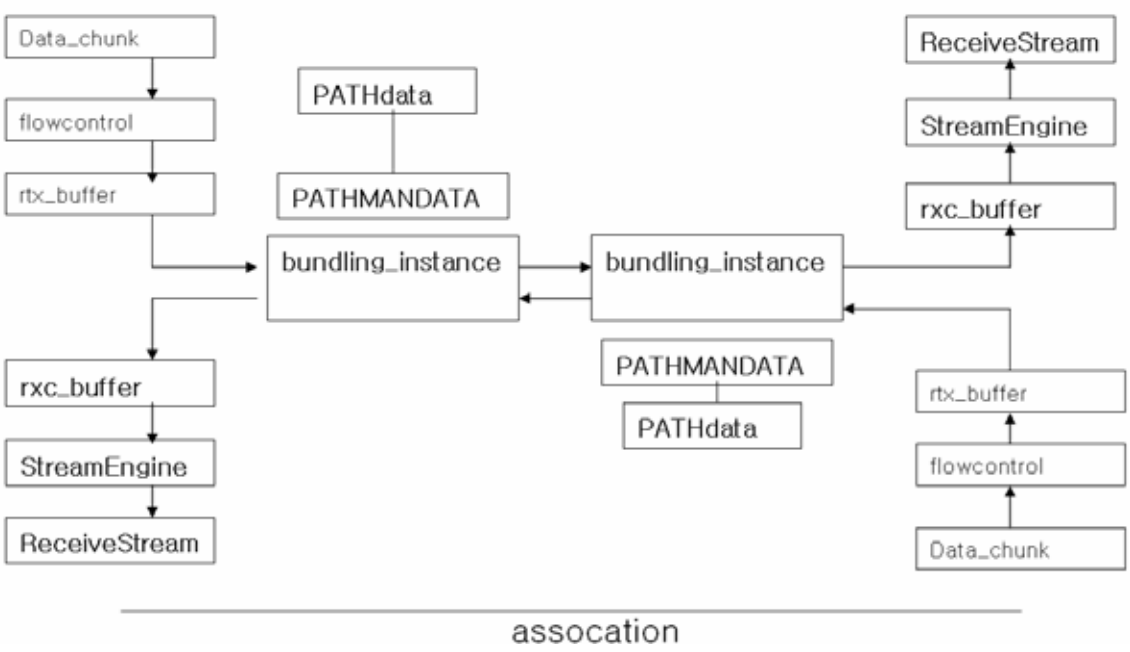
State Cookie 파라미터를 가진 INIT ACK의 송신 후에 송신자는 resource Attack막기 위해 TCB와 그 연결과 관련된 어떤 다른 local 자원을 지워야 한다.

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 되

어야만 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

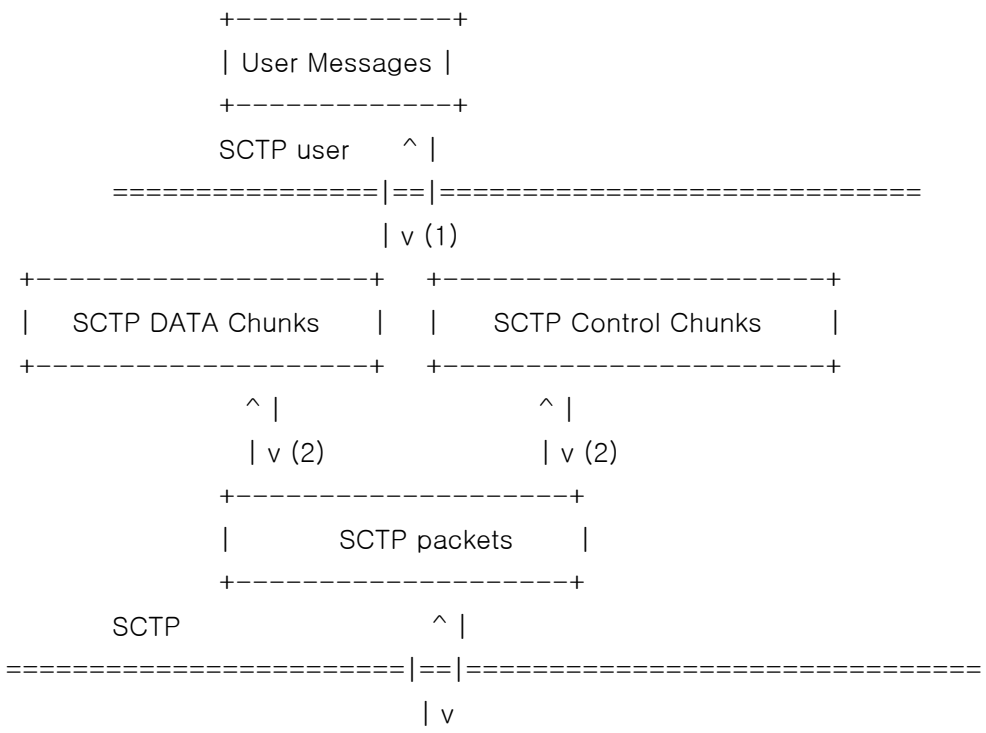
lib코드에서 데이터 전송 측에 관련된 구조체는 data_chunk, flowcontrol, rtx_buffer, bundling_instance이다. 응용에서 데이터 전송을 수행서 data_chunk를 구성하고 현재 데이터를 재전송을 위한 임시 버퍼인 rtx_buffer에 삽입 하여 둔다. 그후 bundling_instance에 max mtu안에서 bundling 을 하고 전송을 수행하게 된다. 그 후 정상적으로 패킷이 도착 하였다는 sack packet이 오면 rtx_buffer에 보관했던 data_chunk를 지우고 그렇지 않고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

시지들을 bundle하고 커다란 사용자 메시지들을 fragmentation 하기 위한 메카니즘들을 정의한다. 다음 다이어그램은 SCTP를 통한 사용자 메시지들의 흐름을 보여준다.



Connectionless Packet Transfer Service (e.g., IP)

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.receiver는 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.c

```
/* 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 + fc_readNumberOfQueuedChunks()) > maxQueueLen)
                return SCTP_QUEUE_EXCEEDED;
        }

        for (i = 1; i <= numberOfSegments; i++)
        {
                cdata = 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 = 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;
            }
        }
    }
  return retVal;
}
```

Ulp_send function에서 데이터를 받아들인 후 데이터가 mtu보다 크면 segmentation을 수행하게 된다. Segmentation시 첫 번째 데이터는 SCTP_DATA_BEGIN_SEGMENT, 중간데이터는 0을 마지막 데이터는 SCTP_DATA_END_SEGMENT를 chunk_flags에 삽입하여 end쪽에서 recv 수행 시 재 조합된다.

6.1 Transmission of DATA Chunks

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.

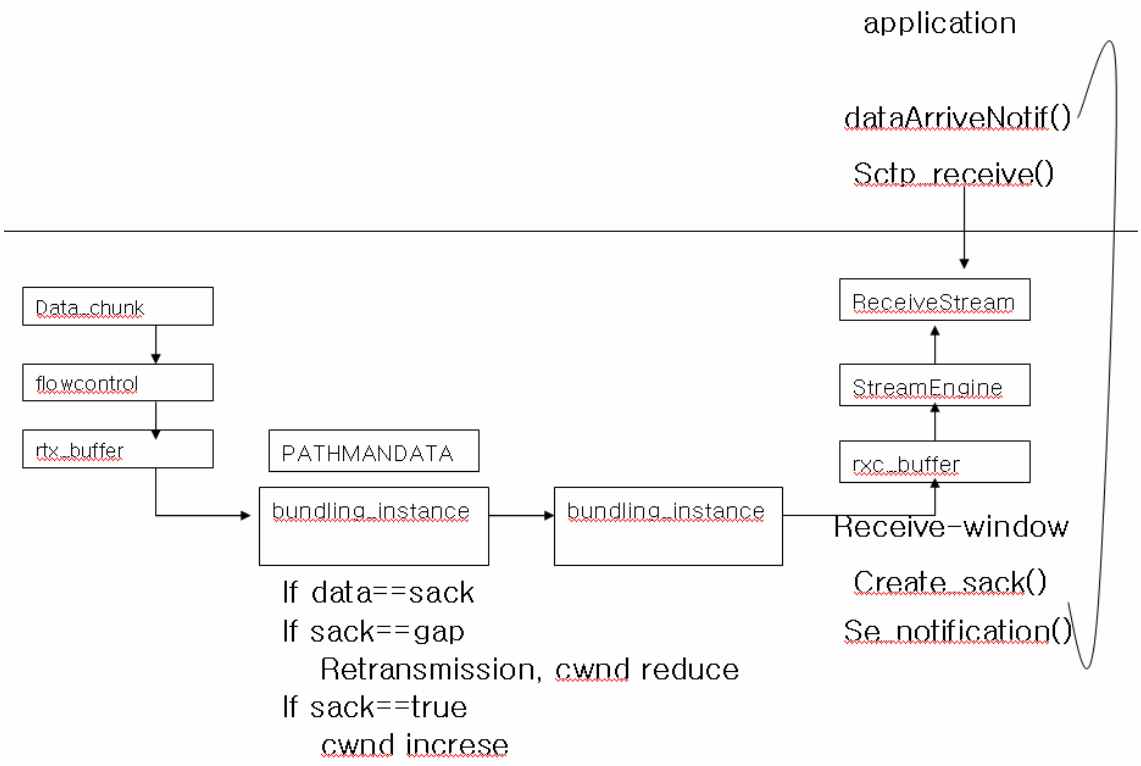
flowcontrol.c

```
-----------------------------------------------------------------
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;
}
```

6.2 Acknowledgement on Reception of DATA Chunk

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.



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 sack정보 안에 삽입 하여 전송을 수행한다. 위의 그림은 association이 맺어 지고 TCB설정이 수행된 후에 데이터 전송 시 전송 측에서 max mtu안에서 bundling된 데이터를 수신 측으로 전송을 수행한다. 그 후 받은 데이터를 rxc_buffer에 삽입을 하고 현재 전송된 데이터가 TSN값이 올바른 데이터라면 gap과 dup값을 표시하지 않고 streamengine에서 de-segmentation을 수행한 후 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

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 맞지 않는 SACK가 수신될 수 있다는 것이다. data receiver에 의해 송신된 SACK는 보다 먼저 SACK를 pass할 수 있고 data sender에 먼저 수신될 수 있다. (?). 만약 순서가 맞지 않는 SACK가 수신되면 data sender는 peer의 수신버퍼 공간의 옳지 않은 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.

Rbundling.c

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

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. Dynamic Address Reconfiguration

SCTP의 확장인 ADDIP 규격i을 토대로 기존 SCTPLIB가 지원하지 않는 동적 주소 재구성 (DAR; Dynamic Address Reconfiguration) 모듈을 추가하여 수송 계층에서의 IP 이동성을 제공하는 라이브러리이다.

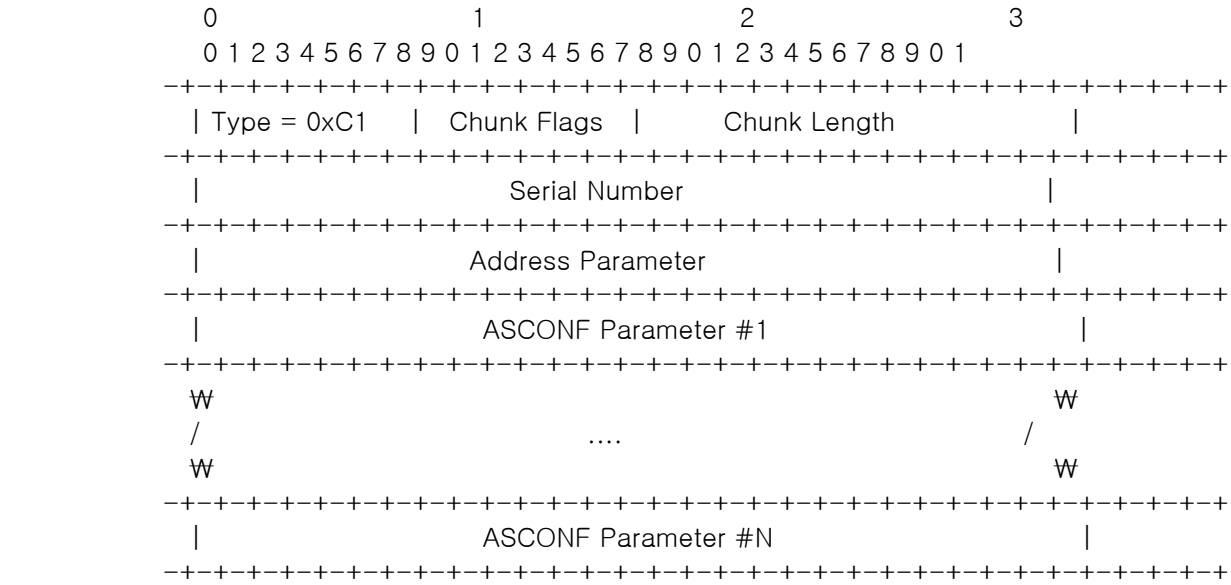
7.1 Additional Chunks and Parameter

reliable한 control information 전송에 사용될 두 가지 새로운 chunk type에 대해 정의

Chunk Type	Chunk Name	
0xC1	Address Configuration Change chunk	(ASCONF)
0x80	Address Configuration Acknowledgment	(ASCONF-ACK)

7.1.1 ASCONF

ASCONF는 remote endpoint와의 통신에 사용된다. ASCONF chunk에서 운반된 정보는 TLV(Type length value)의 형태로 사용된다. 이 chunk는 SCTP_AUTH에 정의된 mechanism을 이용하여 인증되어 전송된다. 만약 이 chunk가 인증되지 않고 수신되면 SCTP_AUTH에서 정의된 데로 버려진다.



Serial Number : 32 bits (unsigned integer)

serial number는 ASCONF chunk에 대한 serial number를 표현한다. serial number의 유효값 범

위는 0-2**32-1이다.

Address Parameter : 8 or 20 bytes (depending on the address type)

이 field는 IPv6, IPv4 형식의 주소 parameter를 포함한다. 이 주소는 ASCONF chunk의 송신자의 주소이다. 주소는 peer endpoint에 의해 association의 부분이 고려되어 져야 하고 이 field는 ASCONF 수신자가 association을 발견하는데 도움이 되게 사용되어진다. 주소 :00이 수신자에게 제공된다면 패킷의 다른 제공된 정보에 의해 association이 검색되어지고 이 parameter는 모든 ASCONF message에서 표현 되어져야 한다.

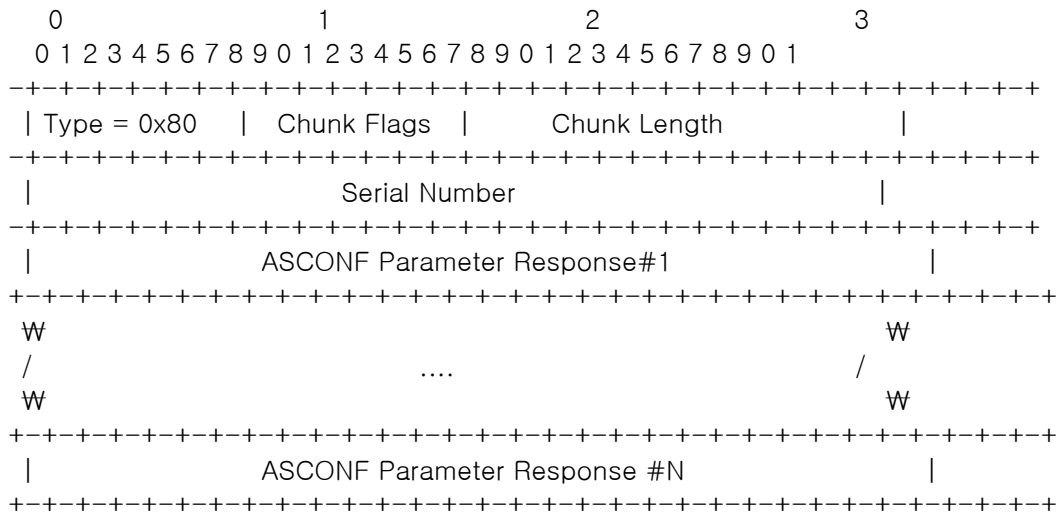
Note: host name address는 보내어 져서는 안되고, 어떤 ASCONF message에서 수신되어진다면 무시되어야 한다.

ASCONF Parameter: TLV format

ASCONF chunk format은 ASCONF chunk를 이해하지 못하면 수신 측이 송신 측에게 보고하는 것을 요구한다. RFC2960에 서술된 것처럼 chunk type의 상위 비트를 세팅함으로써 이것을 이룰 수 있다. ASCONF chunk의 상위 두 비트는 1로 세팅된다. RFC2960에 서술된 것처럼 이 chunk를 이해하지 못한 수신 측은 chunk를 skip하고 processing을 계속하는 이 방법에서 이 상위 비트들을 세팅할 때, ‘Unrecognized chunk type' error의 발생을 사용하는 Operation error chunk에 보고한다. 이것은 association을 중단하지 않지만 송신자에게 더 이상의 다른 ASCONF chunk를 보낼 수 없음을 지시한다.

7.1.2 ASCONF-ACK

이 chunk는 ack 승인에 대한 ASCONF chunk의 수신 측에 의해 사용된다. 수신 측에 의해 처리된 어떤 ASCONF parameter에 대한 결과가 0이나 그 이상의 값으로 전해진다. 이 chunk는 SCTP_AUTH에 정의된 mechanism을 사용함으로써 인증된 방법으로 전송 되야 한다. 인증되지 않은 Chunk가 수신되면 SCTP_AUTH에 정의된 데로 조용히 삭제된다.



ASCONF parameter response는 ASCONF-ACK에서 ASCONF processing의 상태 보고에 사용된다. 디폴트로, responding endpoint가 어떤 error cause도 포함하지 않으면, 성공이 지시된다. 이와 같이 ASCONF-ACK의 송신자는 Chunk Type, Chunk Flag, Chunk Length, Serial Number를 반환함으로써 ASCONF의 모든 TLV의 완전한 성공을 가리킨다.

7.1.3 New Parameter Types

Address Configuration Parameters	Parameter Type
Set Primary Address	0xC004
Adaptation Layer Indication	0xC006

Parameters that can be used in INIT/INIT-ACK chunk

Address Configuration Parameters	Parameter Type
Add IP Address	0xC001
Delete IP Address	0xC002
Set Primary Address	0xC004

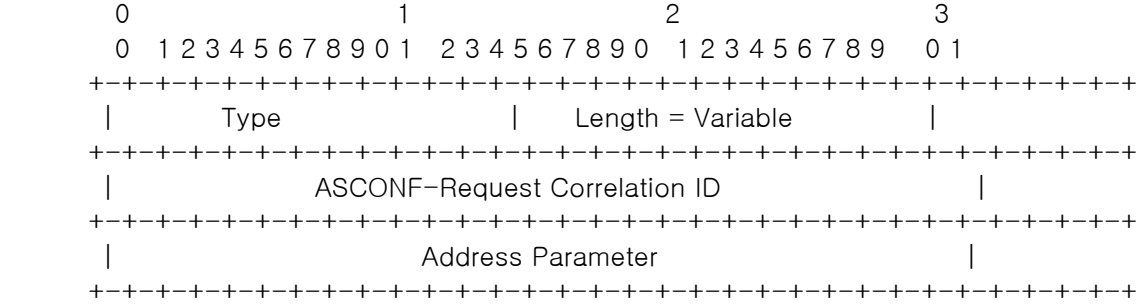
Parameters used in ASCONF Parameter

Address Configuration Parameters	Parameter Type
Error Cause Indication	0xC003
Success Indication	0xC005

Parameters used in ASCONF Parameter Response

허용되지 않은 곳에서 나타나는 어떤 parameter는 invalid parameter의 수신 측에 의해 ABORT로 반응될 수 있다. 수신 측이 abort 하는 것을 선택하지 않으면, 이 parameter는 무시되어 져야 한다

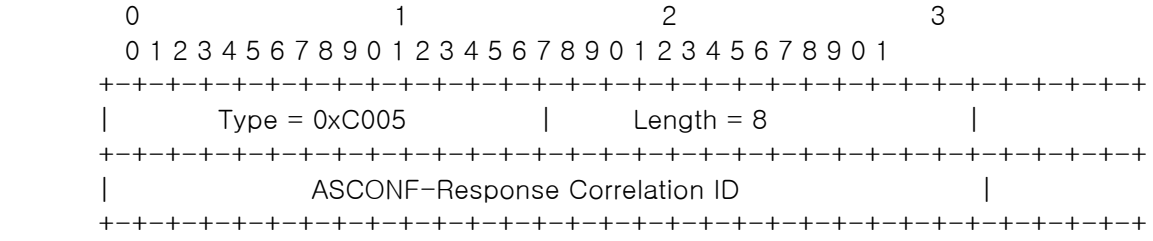
7.1.4 Add IP Address , Delete IP Address, Set Primary Address



각각의 요청된 parameter의 확인을 위해 송신자에 의해 할당된 불명확한 정수이다. ASCONF

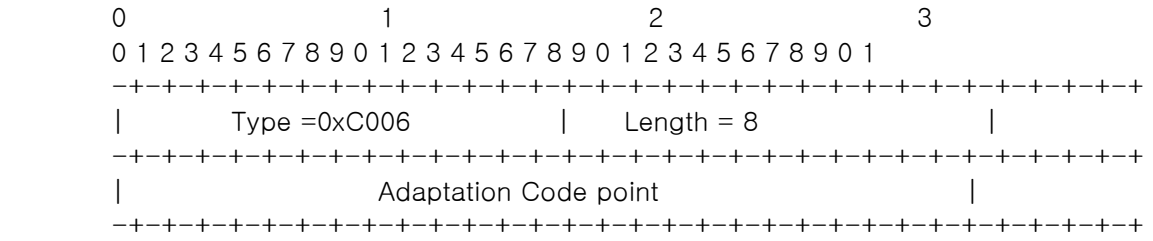
chunk의 수신자는 ASCONF-ACK response parameter의 ASCONF response correlation ID field에 이 32bit 값을 복사한다. ASCONF의 송신자는 response가 요구하는 요청을 발견하기 위한 ASCONF-ACK의 같은 값을 사용할 수 있다. 수신 측이 이 32bit 값을 변경할 수 없음을 주의해야 한다. IP address parameter 추가는 오직 ASCONF chunk type에서 나타난다.

7.1.5 Success Indication



default로써 responding endpoint가 어떤 요청된 TLV에 대한 error를 보고하지 않으면 success는 암묵적으로 표시된다. 이와 같이 ASCONF-ACK의 송신 측은 오직 Chunk type, Chunk Flag, Chunk Length, Serial Number의 반환으로써 ASCONF에서 모든 TLV의 완전한 성공을 표시할 수 있다. responding endpoint는 ASCONF Parameter Response 보고의 성공을 반환함으로써, 요청에 대한 성공적인 보고를 명시적으로 선택할 수도 있다. Success Indication parameter는 오직 ASCONF-ACK에서만 나타날 수 있다.

7.1.6 Adaptation Layer Indication



upper later protocol의 통신에 대해 명시 되어진 parameter. flow control과 INIT, INIT CHUNK에 운반 되어져야 할 요구되는 indication인 다른 adaptation later가 사용되어질 계획.

이 parameter를 사용하기를 원하는 것으로 정의된 각각의 adaptation layer는 RFC에 정의된 사용과 의미에 대한 adaptation code point를 명시해야 한다. 이 parameter는 수신 SCTP implementation에 의해 examine 되어 지면 안되고, upper layer protocol에 불분명하게 pass 되어져야 한다.

Note: 이 parameter는 주소의 addition, deletion 모두에서 사용되지 않지만, upper layer protocol의 편리를 위해 사용된다

adaptation layer indication parameter는 INIT, INIT ACK Chunk에 나타날 수 있고, SCTP 스택의 upper layer protocol configuration에 기초한 수신 upper layer protocol로 보내진다. 이 parameter는 어떤 다른 chunk에 보내지면 안되고, 다른 chunk에서 이것을 수신한다면 chunk는 이것을 무시해야 한다.

7.1.7 New Error Causes

Value	Cause Code
0x0100	Request to Delete Last Remaining IP Address.
0x0101	Operation Refused Due to Resource Shortage.
0x0102	Request to Delete Source IP Address.
0x0103	Association Aborted due to illegal ASCONF-ACK
0x0104	Request refused - no authorization.

다섯 개의 새로운 Error Causes는 주로 ASCONF-ACK Chunk에서의 사용으로 SCTP Operational Errors에 추가된다.

7.2 Procedures

7.2.1 ASCONF Chunk Procedures

endpoint가 ASCONF signal change를 다른 endpoint에게 보냈을 때 다음의 절차를 따른다

A1) ASCONF를 생성 후 Chunk는 remote endpoint에 보내진 TLV의 모든 필요한 정보와 각 request의 unique한 correlation identities를 포함 해야 한다.

A2) 1보다 큰 serial number는 Chunk에 할당되어 져야 하고 association의 시작에 initial TSN과 같은 값으로 초기화 되어져야 한다. ASCONF Chunk가 생성될 때마다 새롭게 생성된 chunk에 serial number가 할당된 후 1씩 증가되어야 한다.

A3) 하나이상의 ASCONF chunk를 가지지 않은 SCTP 패킷이 remote peer에게 전송되면, 송신 측은 T-4 RTO timer를 시작해야 하고 선택된 destination address의 RTO value(보통 primary path)를 사용한다.

A5)마지막 보내진 serial number를 ACK한 ASCONF_ACK가 도착하면, 송신 측은 T-4 RTO timer를 멈춰야 하고 RFC2960의 8.1과 8.2에 정의된 적당한 association과 destination error counter를 clear한다.

A6)endpoint는 보내진 다양한 요청을 반환하는 특정 상태 정보를 발견할 수 있는 ASCONF-ACK사이에서 모든 TLV를 process해야 한다.

A7)TLV parameter에 대한 error response가 수신되면 반응 없는 전에 실패한 TLV와 함께 모든 TLV는 보고되지 않았다면 성공한 것으로 간주된다. failed response 후의 모든 TLV는 명시

된 성공 지시가 parameter에 대한 표시에 없으면 실패로 간주된다.

A8)TLV parameter를 명시하고 실패가 표시된 response가 없으면 모든 요청은 성공으로 간주된다.

A9)peer가 ASCONF Chunk type을 인식하지 않았던 ERROR chunk reporting과 함께 ASCONF에 반응하면 ASCONF의 송신자는 더 이상의 ASCONF Chunk를 보내서는 안되고 그것의 T-4 timer를 멈춰야 한다.

7.2.2 Congestion Control of ASCONF Chunks

ASCONF Chunk transfer procedure 정의에서, 이 전송은 network안에서 congestion을 발생시키지 않는 것이 필수적이다. 이것을 달성하기 위해, ASCONF Chunks의 전송에 제한이 있다.

C1) ASCONF Chunk를 holding하는 하나의 SCTP packet은 어떤 시간에 transit과 unacknowledge 상태에 있을 것이다. 송신 측이 ASCONF Chunk를 전송 후에 다른 ASCONF Chunk 전송이 필요하다고 결정하면, 다음의 ASCONF를 보내기 전에 이전 ASCONF에 대한 ASCONF-ACK Chunk가 돌아올 때까지 기다려야 한다.

C2) ASCONF Chunk가 다른 ASCONF Chunks를 포함하는 다른 chunk type와 bundled 되어 질 수 있다. 만약 다른 ASCONF Chunks이 bundled되면, chunks는 serial number를 반영하는 sequential order로 나타나야 한다.

C3) ASCONF-ACK Chunk는 다른 ASCONF-ACK Chunks를 포함하는 다른 chunk type으로 bundle되어 질 수 있다.

C4) ASCONF와 ASCONF-ACK Chunks는 ESTABLISHED, SHUTDOWN-PENDING, SHUTDOWN-RECEIVED, SHUTDOWN-SENT를 제외한 어떤 다른 SCTP 상태로 보내어져서는 안된다.

C5) ASCONF Chunk와 ASCONF-ACK Chunk는 path MTU보다 커서는 안된다. path MTU가 알려지지 않았다면 path MTU는 최소로 세팅 되어져야 한다. minimum path MTU는 전송에 사용되는 IP version에 의존하고, 576octet보다 작다.

7.2.3 Upon reception of an ASCONF Chunk

endpoint가 remote peer로부터 ASCONF Chunk를 받을 때의 특별한 절차는 ASCONF Chunk가 결합된 association을 확인하는 것이 필요할 것이다. 아래의 절차에 따라 association을 적절히 찾기 위해 다음을 따른다.

D1) source address와 association을 확인하는 것을 시도하는 송신자의 port number를 사용한다.

D2) association이 발견되지 않으면 SCTP common header에 있는 port number가 결합된

Address Parameter TLV에서 발견된 주소를 사용한다.

D2-ext) 하나이상의 ASCONF Chunk가 함께 포장되어 있다면, 각각의 다음 ASCONF Chunk의 ASCONF Address Parameter에 있는 address를 사용한다.

D3) D1,D2,D2ext 모두 association이 없으면, RFC2960에 정의된 Out of The Blue Packet으로 chunk를 다룬다.

D4) RFC2960에 있는 유효한 SCTP verification tag의 일반적인 규칙을 따른다.

association의 확인과 증명 후에, ASCONF Chunk에 대한 적당한 process가 수행되어 져야 한다.

E1) ASCONF Chunk의 serial number에서 발견된 값이 Peer-Serial-Number+1과 같고, ASCONF Chunk의 serial number가 SCTP 패킷의 처음이면, endpoint는 'Peer-Serial-Number'까지 catch 된 ASCONF-ACK의 옛날 것을 clear할 수 있고 E4규칙에 따라 진행된다.

E1-ext) ASCONF Chunk의 serial number에 있는 값은 'Peer-Serial-Number'+1과 같고, 그 ASCONF chunk는 E4에 따라 진행되는 SCTP packet의 첫 번째 serial number가 아니다.

E2) serial number에 있는 값이 'Peer-Serial-Number+1'보다 작으면, 다음 ASCONF로 간단히 skip하고 ASCONF의 serial number를 매치하는 전송되고 저장된 이전의 cashed ASCONF-ACK response인 외부 response packet을 포함한다.

Note: ASCONF-ACK Chunk 존재를 캐치하지 않는 것도 가능하다. 이것은 오래된 ASCONF가 out of order로 도착했을 때 발생할 것이다. 이 같은 경우 수신 측은 ASCONF Chunk를 skip 해야 하고, 그 chunk에 대한 ASCONF-ACK를 포함하지 않는다.

E3) ASCONF의 serial number가 'Peer-Serial-Number+1'보다 작다 할지라도, ASCONF는 위처럼 차례로 process한다.

E4) serial number가 다음 기대되는 것과 매치되면 ASCONF를 process 하고 , ASCONF Chunk를 processing한 후에, ASCONF-ACK Chunk를 response packet과 그것을 복사한 cache에 덧붙인다.(나중에 재전송 필요)

E5) 그렇지 않으면, 그 ASCONF Chunk는 stale 패킷 또는 attacker 때문에 버려진다.이 같은 패킷의 수신 측은 보안적인 목적에 대한 이벤트를 log 할 수도 있다.

E6) 모든 ASCONF Chunk가 이 SCTP packet에 대해 process 되어질 때, 모든 ASCONF-ACK Chunk와 함께 축적된 single response packet를 반환한다. ASCONF-ACK Chunk를 포함하는 SCTP packet의 destination 주소는 ASCONF Chunk를 held한 SCTP packet의 source address가 되어져야 한다.

E7) SCTP packet에서 ASCONF Chunk를 processing하는 동안, response packet이 return path의 PMTU를 초과한다면 수신 측은 response packet으로 추가적인 ASCONF-ACK를 add 하는 것을 멈춰야 한다. 그러나 모든 ASCONF Chunk를 process하는 것과 그것의 cached copy에서

ASCONF-ACK Chunk response를 saving하는 것은 계속한다. ASCONF Chunk의 송신자는 response되지 않은 ASCONF Chunk를 후에(PMTU에 맞지 않는 그 response의 cached copies가 peer에게 전송될 수 있을 때) 재전송 할 것이다.

Note: 이 규칙은 실행이 ACK path에서 SCTP packet의 loss의 경우 old ASCONF-ACK를 caching한다는 가정하에서 제시된 것이다. 같은 ASCONF-ACK sequence에서 결과가 위의 outline처럼 반환된 적당하다고 간주된 다른 형태에서 이 state를 유지하기 위한 실행이 가능할 수 있다.

7.2.4 General rules for address manipulation

IP address를 더하거나 삭제할 ASCONF Chunk에 대한 TLV parameter를 building할 때 다음의 규칙은 적용되어야 한다.

F0) endpoint가 다음 serial number와 같거나 더 큰 ASCONF를 수신한다면, outstanding endpoint는 association을 ABORT해야 한다.

F1) IP address를 association에 추가할 때, IP address는 ASCONF-ACK가 도착할 때까지 association에 완전히 추가되어 지는 것이 고려되지 않는다. 이것은 추가가 송신자에게 ACK되는 것을 포함하는 ASCONF와 같은 시간까지 ASCONF Chunk를 전하는 것을 제외한 어떤 SCTP 패킷에 대한 source로 새로운 IP address를 사용해서는 안된다는 의미이다. 추가 IP address 요청의 수신 측은 destination으로서 그 주소를 즉시 사용할 수 있다. 수신 측은 그 주소를 사용하기 전에 추가된 주소에 대한 path verification procedure를 사용해야 된다. 수신 측은 새로운 path가 verifiy되기 전에 corresponding ASCONF-ACK Chunk 또는 HEARTBEAT Chunk를 제외하고 새로운 주소에 패킷을 보내서는 안된다. ASCONF-ACK가 새로운 주소로 보내진다면, path verification에 대한 HEARTBEAT chunk가 함께 bundle될 것이다.

F2) IP address 추가의 ASCONF-ACK가 도착한 후에, endpoint는 어떤 SCTP chunk의 타입에 대한 source address로서 추가된 IP address를 사용하는 것을 시작할 것이다.

F3a) endpoint가 이해하지 못한 IP address Add 또는 IP address Deletion parameter를 가리키는 Error Cause TLV를 수신하면, endpoint는 operation failed를 고려해야 하고 peer에게 어떤 다음 Add 또는 Delete request 전송 시도를 해서는 안된다.

F3b) endpoint가 이해하지 못한 IP address Set Primary IP Address parameter를 지시하는 Error Cause TLV를 수신하면, endpoint는 operation failed를 고려해야 하고, peer에게 어떤 다음 Set Primary IP Address request를 보내는 것을 시도해서는 안된다.

F4) association으로부터 IP address를 삭제할 때, IP address는 ASCONF-ACK가 도착할 때까지 SCTP packet의 reception에 대한 valid destination address를 고려해야 하고, 어떤 다음 packet에 대한 source address로써 사용해서는 안된다. 이것은 삭제되어질 IP address에 도착한 ASCONF-ACK 전에 도착하는 어떤 datagram이 현재 association의 부분이라는 것이 고려되어 져야 한다. 한가지 특별한 고려는 삭제될 IP address에 도착할 ABORT Chunk는 무시되

어져야 한다.

F5) endpoint는 association으로부터 endpoint의 마지막 남은 IP address를 삭제해서는 안된다. 다른 말로, endpoint가 multi-homed가 아니면 ASCONF Chunk에서 delete parameter를 하기 전에 IP address 추가 없이 삭제한 IP address를 사용해서는 안된다. 또는 endpoint가 delete IP address에 multiple request를 보낸다면, peer가 요청자에 대해 정리한 모든 IP address를 삭제해서는 안된다.

F6) endpoint는 ASCONF Chunk에 의해 삭제된 주소와 동일하게 된 ASCONF Chunk를 holding 하고 있는 SCTP 패킷에 대한 IP header source address를 세팅 해서는 안된다.

F7) peer endpoint의 마지막 남은 IP address를 삭제하려는 request가 수신되면, 수신 측은 새로운 error code 'Request to Delete Last Remaining IP Address'를 설정하는 Error Cause TLV를 보내야 한다. 요청된 삭제는 ASCONF-ACK 전송 이외에는 그 전에 수행되어서는 안된다.

F8) ASCONF chunk가 포함된 IP packet의 source address인 IP address를 삭제하는 요청이 수신되면, 수신 측은 이 요청을 거부한다. 그 요청을 거부하기 위해 수신 측은 'Request to Delete Source IP Address'를 설정하는 Error Cause TLV를 전송해야 한다.

F9) endpoint가 ADD IP address 요청을 수신하고, 이 새로운 주소를 association에 더하는 local resource를 가지지 않는다면 새로운 error code' Operation Refused Due to Resource Shortage'를 설정하는 Error Cause TLV를 반환해야 한다.

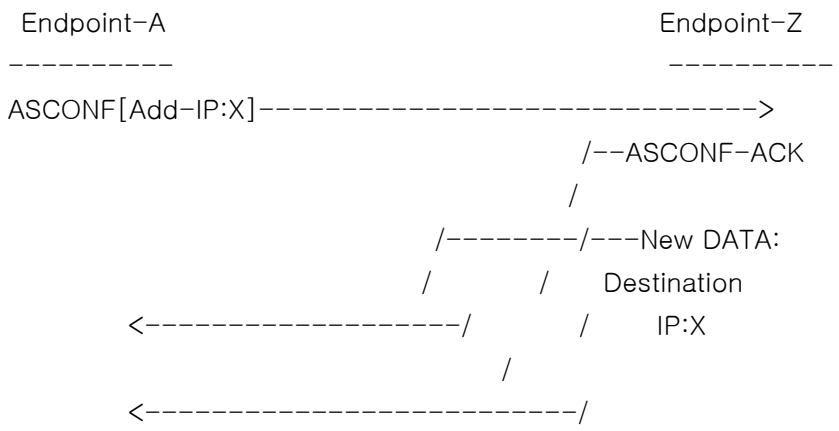
F10) endpoint가 association에 IP address를 더하기 위한 요청의 응답으로 'Out of Resource'를 수신하면, association을 ABORT하거나, association의 주소 부분을 고려하지 않아야 한다. 다른 말로, endpoint가 association을 ABORT하지 않으면, 실패된 시도를 추가하는 것이 고려되어져야 하고, 그것의 peer가 Out Of The Blue packet으로써 목적되어진 주소에서 STCP packet을 다루어야 하기 때문에 이 주소는 사용되어져서는 안된다.

F11) IP address를 추가하는 ASCONF를 수신하는 endpoint가 response로써 'Out Of Resource'를 전송한다면, 다음 add가 실패해야 하거나, ASCONF에서 bundle된 request가 삭제되어야 한다. 수신 측은 ADD를 거부해서는 안되고, 같은 ASCONF Chunk에서 다음 IP Address의 삭제를 받아들여야 한다. 다른 말로, 수신 측이 ADD 또는 DELETE 요청을 failing하기 시작한다면, 그 단일 ASCONF에서 포함된 모든 다음 ADD와 DELETE request는 실패한다.

F12) endpoint가 현재 primary address인 IP address를 삭제하는 요청을 받을 때, endpoint가 새로운 primary address을 선택하는 방법으로서 implementation decision을 결정한다.

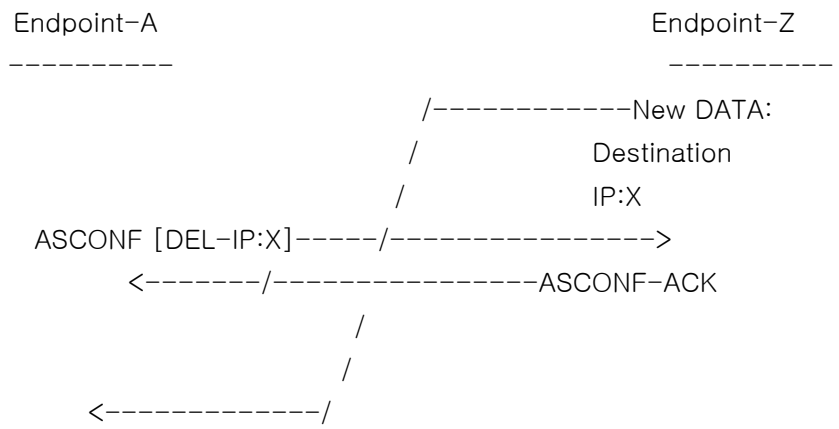
F13) endpoint가 IP address를 삭제하는 valid request를 수신 할 때, endpoint는 association의 부분으로서 주소를 더 이상 고려하지 않는다.

ASCONF를 전송하고 ASCONF-ACK를 수신하는 사이 out of order인 DATA Chunk를 수신하는 것이 가능하다. 이 문제를 예로 설명하였다.



위의 예에서 우리는 Endpoint-A에 추가된 새로운 IP address(X)를 볼 수 있다. 그렇지만, network에서 packet 재 순서화 때문에 새로운 DATA chunk가 보내지고 association에 대한 주소추가를 확인하는 ASCONF-ACK전에 Endpoint-A에 도착한다.

A similar problem exists with the deletion of an IP address as follows:

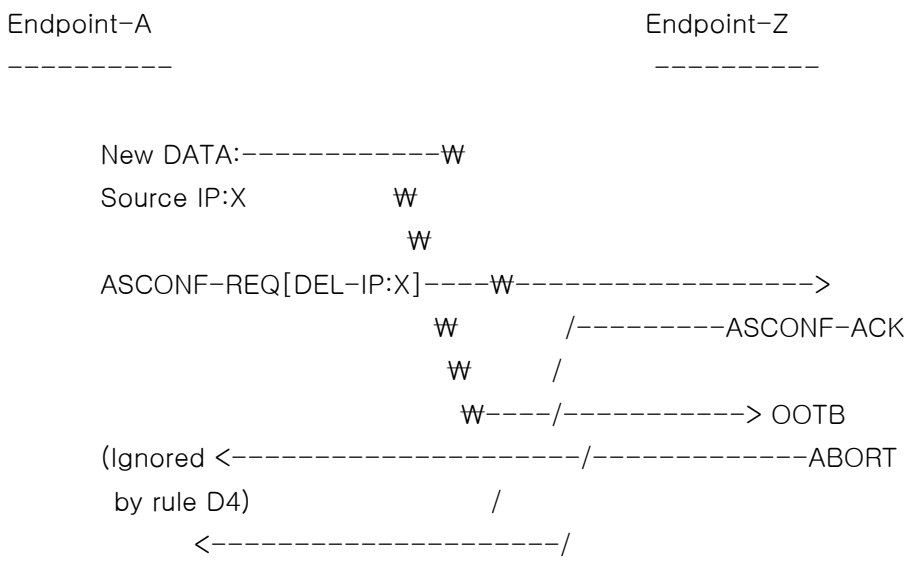


이 예에서 우리는 삭제가 완료된 후에 도착하는 IP:X로의 DATA chunk를 확인할 수 있다. endpoint에 ADD를 하기 위해 새로운 IP address 추가를 고려해야 한다.

ASCONF-ACK를 기다리는 기간 동안 data를 수신하는 association은 유효하다. endpoint는 ASCONF-ACK가 도착할 때까지 이 새로운 주소로부터 source data를 Must NOT, 하지만 설명되어진 것처럼 out of order data를 수신 할 수 있고, OOTB datagram처럼 이 data를 다루어서는 안된다. 조용히 data가 drop할 수 있거나, association의 부분으로 고려되어질 수 있다. 그러나, ABORT를 응답해서는 안된다.

DELETE 경우, endpoint는 OOTB datagram처럼 늦게 도착하는 DATA packet에 반응할 수 있고, 여전히 유효한 시간의 small period에 대한 IP address 삭제를 hold 할 수도 있다. OOTB처럼 DATA packet이 다루어지면, peer는 조용히 ABORT를 버린다. endpoint가 유효한 시간 동안 IP address를 hold하는 것을 결정한다면, destination이 삭제되는 2RTO 이상 유효하게 hold 해서는 안된다.

7.2.5 A special case for OOTB ABORT Chunks



이 경우 IP address의 삭제 동안, Abort message의 destination address가 삭제될 destination 이면 ABORT는 무시된다.

7.2.6 A special case for changing an address.

association에서 송신 측은 오직 하나의 renumbered된 IP address를 가진다. 이것이 일어났을 때, 송신 측은 peer에게 적당한 ADD/DELETE pair를 전송하는 것을 할 수 없을 것이고, IP header에서 source로써 old address를 사용한다. 이 이유로 송신 측은 Address Parameter field를 association의 부분인 address로 채운다. 이것은 IP header에 있는 source address 사용 없이 association을 수신 측에 locate하는 것을 허용한다.

chunk의 수신 측은 항상 association을 찾는 IP header에 있는 source address를 사용한다.

IP header로부터 source address를 사용하는 검색이 실패하면 수신 측은 Address Bytes field에 있는 address를 사용하는 것을 시도해야 한다. 수신 측은 ASCONF에 의해 추가되는 새로운 address인 패킷 source address에 대해 응답해야 한다.

7.2.7 Setting of the primary address

이 옵션의 송신자는 주소의 삭제 또는 추가에 대한 결합을 전송하는 것이 가능하다. 송신자는 오직 이미 association의 부분으로 고려되어진 주소를 요청하는 set primary request를 전송해야 한다. 다른 말로, 송신자가 새로운 IP address를 추가하는 set primary를 conbine한다면 set primary는 버려질 것이다.

set primary에 대한 요청은 INTI or INIT ACK chunk에 나타날 수 있다. 이것은 INIT or INIT-

ACK의 송신자 주소로써 peer endpoint에게 advice할 수 있다.

primary path로써 주소를 설정하는 요청은 수신 측이 수행해야 하는 옵션이다. 이것은 SCTP 패킷을 전송하는데 사용하는 최상의 destination address의 수신자에게 advice 될 수 있다. 수신 측이 존재하지 않는 primary로써 주소를 설정하는 요청이 도착한다면, 수신 측은 request 권한이 없고 primary address를 바뀌지 않은 상태로 남겨둔다.

7.2.8 Bundling of multiple ASCONFs

보통의 경우 packet에서 single ASCONF는 전송되어지고, single reply ASCONF-ACK는 수신된다. 그렇지만, ASCONF와 ASCONF-ACK를 포함하는 SCTP packet loss가 발생하는 경우 재전송에서 추가적인 ASCONF를 bundle 하는 것이 송신 측에게 허용된다.

multiple ASCONF bundling 규칙은 아래와 같다

1. 이전에 전송된 ASCONF Chunk는 변하지 않은 상태로 남아있어야 한다.

2. ASCONF Chunk를 포함하는 각각의 SCTP packet은 패킷에서 처음의 가장 작은 ASCONF Serial Number와 함께 bundle되어져야 하고, lowest에서 highest로 ASCONF Serial Number가 순차적으로 preceding한다.

3. packet 사이의 모든 ASCONF는 각각의 다른 것과 인접해야 한다. 다른 chunk type은 ASCONF를 분리해야 한다.

4. address를 검색하는 각각의 새로운 ASCONF는 이전 ASCONF가 process되고 accept된 것처럼 호환성이 있어야 한다.

8. 결론

지금까지 본 고에서는 SCTPLIB에 대한 분석 및 주요 확장 기능인 Dynamic Address Reconfiguration 모듈 부분에 대한 분석을 하였다. 현재 개발 중인 시스템은 WinXp 및 WinCE 플랫폼을 기반으로 하여 기존 SCTPLIB를 수정하고 Dynamic Address Reconfiguration 기능을 추가하여 이동 단말의 동적인 주소 추가 및 삭제가 가능하게 하도록 하는 것이다. 향후 연구로는 L2/L3 연동 모듈을 개발하여 이 기종 망에서 끊김 없는 핸드오버를 가능하게 하는 시스템으로 확장할 계획이다.

참고 문헌

[1] Stewart R., et al., "Stream Control Transmission Protocol", IETF RFC 2960, October 2000

[2] Pastor J. and Belinchon M., “SCTP Management Information Base”, IETF Internet Draft, draft-ietf-sigtran-sctp-mib-08.txt, November 2002

[3] SCTP implementations by Linux, http://rivus.sourceforge.net/

[4] SCTP tutorial, http://www.iec.org/online/tutorials/sctp/

[5] Stewart, R., et al., " SCTP Dynamic Address Reconfiguration", IETF Internet Draft, draft-ietf-tsvwg-addip-sctp-19.txt, Jun 2007