RMTP

작성: 엄 기 종

Reliable Multisession Transaction Protocol

Revision 1.3

2002.09.16

Keyword

RMTP, IPC, Protocol

차례

History

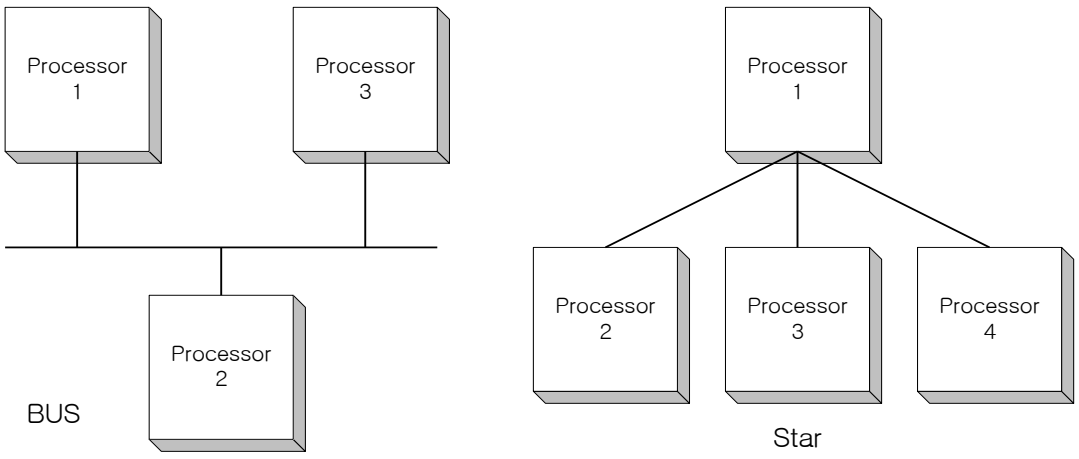
날 짜	Rev	설 명
2001.02.01	1.0	최초 문서
2001.03.02	1.1	기존 프로토콜 비교 간략화, 망 구성도 추가, 프로토콜 모델 추가, 용어 정의 추가 Flow control, error control 추가, CONNECT, CLOSEACK type 추가, 시나리오 추가 장점에 대한 자세한 설명 추가
2002.09.12	1.2	원본 문서 분실로 재작성 CONNECT 메시지 제거, Event/Action list 추가, User's Guide 추가
	1.3	메모리 할당/해제 방법 추가, 시뮬레이션 결과 추가, Known BUG 추가

1. 개 요

현재 개발되는 가입자 통신 장비는 대부분 모국측 장비와 자국측 장비 두 가지 종류로 나눌 수 있다. 두 장비는 서로 밀접하게 관련이 되어 맞물려 돌아가게 되어 있어서, 두 장비 사이에 메시지를 주고 받는 일이 많을 수 밖에 없다. 또한, 한 장비 내에서라도 두 개 이상의 프로세서(CPU)가 존재한다면 각 프로세서간에 메시지를 주고받는 일이 많다. 각 장비 또는 프로세서(이하 프로세서로 용어 통일함)에서 메시지를 주고 받는 일을 IPC (Inter Processor Communication)이라고 말한다. 이런 환경에서 장비 사이, 또는 프로세서 사이에서 안정적으로 메시지를 주고 받기 위해서는 메시지의 전송을 보장할 수 있는 프로토콜이 필요하다. 이 문서에서는 IPC의 특징을 서술하고, IPC의 transmission layer 용도로 기존에 널리 사용되고 있는 프로토콜의 장단점을 비교하고, 기존 프로토콜의 단점을 개선한 새 프로토콜을 제안한다.

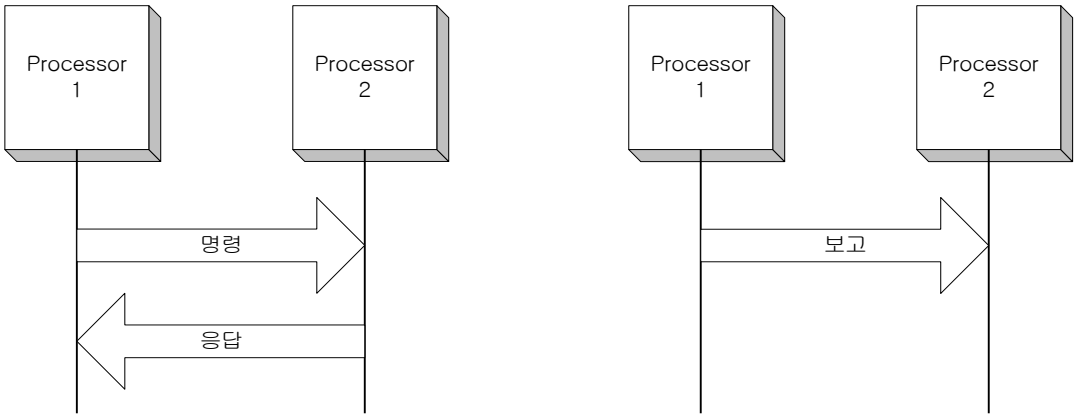
2. IPC의 특징

- IPC 메시지를 주고 받는 프로세서의 토폴로지는 아래 그림과 같다.



- IPC 메시지의 크기는 수 Byte에서 수 MByte 까지 다양하다.
- IPC 메시지는 burst하게 발생하며, 여러 종류의 메시지가 동시에 전송될 수 있어야 한다. (즉, 수 MByte 짜리 메시지를 전송하는 동안 수 Byte 짜리 메시지가 기다리는 상황이 발생하면 안된다.)
- 메시지의 송신측과 수신측에서 성공적인 메시지 전달 및 명령 수행 여부를 알아야 한다.
- 제품의 단가 및 처리 속도를 줄이기 위해 가벼운 프로토콜이 필요하다.
- IPC 메시지는 크게 다음과 같은 두가지 타입으로 나눌 수 있다:

- 보고: 한 프로세서에서 다른 프로세서로 메시지 전달 (일방적)
- 명령/응답: 한 프로세서에서 다른 프로세서로 명령 전달 후 명령 수행 결과를 되돌려 받음 (양방향)



이런 성격을 가진 IPC 메시지를 처리하기 위해서는 다음과 같은 조건이 필요하다:

- 메시지 전달의 보장
- 긴 메시지를 잘라서 보내줌
- Transaction 처리 기능 (명령, 응답이 한 세션에서 이루어짐)
- Multi-session 처리 기능
- 구현 및 포팅이 쉬워야 함
- 가벼움

3. 기존 프로토콜 비교

항목	TCP/IP	OSI	LAP-D
안정성	검증됨	검증됨	검증됨
메시지 전달 보장	O	O	O
Multi-session	O	O	X
Transaction	X	X	X
Large Message	O	O	X
네트워크 모델	대규모 mesh 네트워크	대규모 mesh 네트워크	point-to-multipoing
Protocol Load	큼	큼	작음
가격	쌈	비쌈	비쌈
기타	Global IP address 부족	사용하기 어려움	포팅이 어려움

TCP/IP는 현재 가장 많이 사용되고 있는 프로토콜이지만, transaction 처리가 안되고, 애초에 다양

한 네트워크를 모두 지원하기 위해 설계된 것이라서, 프로토콜 로드가 크다. OSI 프로토콜은 필요한 것은 거의 모두 지원하지만, 사용하기가 어렵고, 가격이 비싸고, 로드가 매우 크다는 단점이 있다. LAP-D는 multisession, transaction이 지원되지 않고, 수 메가 바이트의 큰 메시지도 전달할 수 없다.

4. RMTP (Reliable Multisession Transaction Protocol)

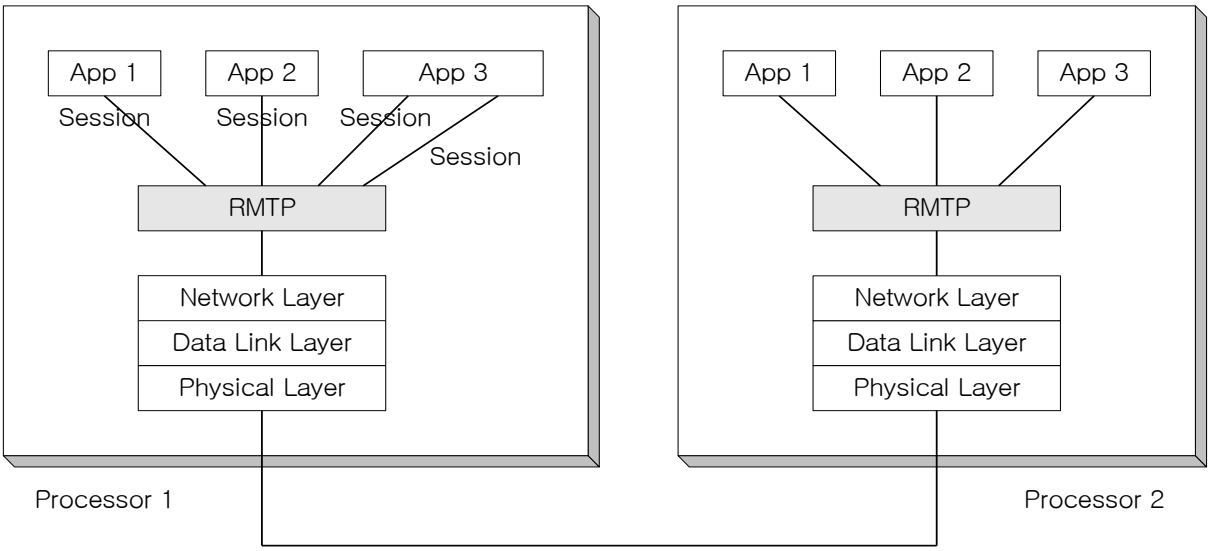
여기서는 앞에서 지적한 사항들을 모두 지원하는 RMTP를 제안한다.

4.1. 용어 정의

- 세션 (session): state diagram의 START에서 END까지 이르는 과정.
- 트랜잭션 (transaction): 명령을 보내고 응답을 완전히 받기까지, 또는 보고를 완전히 보내기까지의 과정. 하나의 세션은 하나의 트랜잭션으로 이루어짐.
- 명령 (COMMAND): 한 프로세서가 다른 프로세서로 전송하는 작업 요청. 명령을 받은 프로세서는 그에 대한 응답을 보냄.
- 응답 (REPLY): 명령에 대한 작업 결과.
- 보고 (REPORT): 한 프로세서가 다른 프로세서로 일방적으로 전송하는 내용.
- 메시지: 한 장비에서 다른 장비로 보내야 하는 명령, 응답 또는 보고 내용.
- 패킷: 큰 메시지를 전송할 때 적당한 크기로 잘라서 보내게 되는데, 그 잘려진 한 조각의 내용과 전송에 필요한 헤더.
- Commander: 명령을 보내고 그에 대한 응답을 받는 프로세서. 또는 보고를 보내는 프로세서.
- Replier: 명령을 받아서 그에 대한 응답을 보내는 프로세서. 또는 보고를 받는 프로세서.
- 송신측: 현재 명령/응답/보고를 보내고 있는 프로세서. State diagram에서 WAIT_ACK 상태에 있음.
- 수신측: 현재 명령/응답/보고를 받고 있는 프로세서. State diagram에서 WAIT_DATA 상태에 있음.
- MTU: 한 패킷에 담길 수 있는 최대 데이터 크기.
- SLIDING_WINDOW: ACK 패킷을 받기 전까지 보낼 수 있는 최대 데이터 크기.
- timeout: 일정 시간 동안 event가 발생하지 않는 경우.

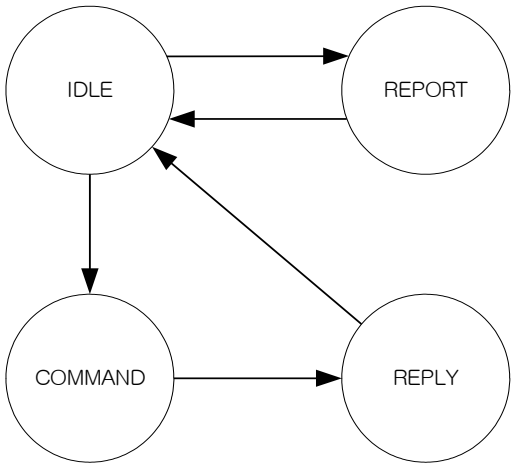
4.2. 프로토콜 개요

RMTP는 다음 그림과 같이 OSI model 상에서 transport layer 기능과 session layer 기능을 포함하는 프로토콜이다.

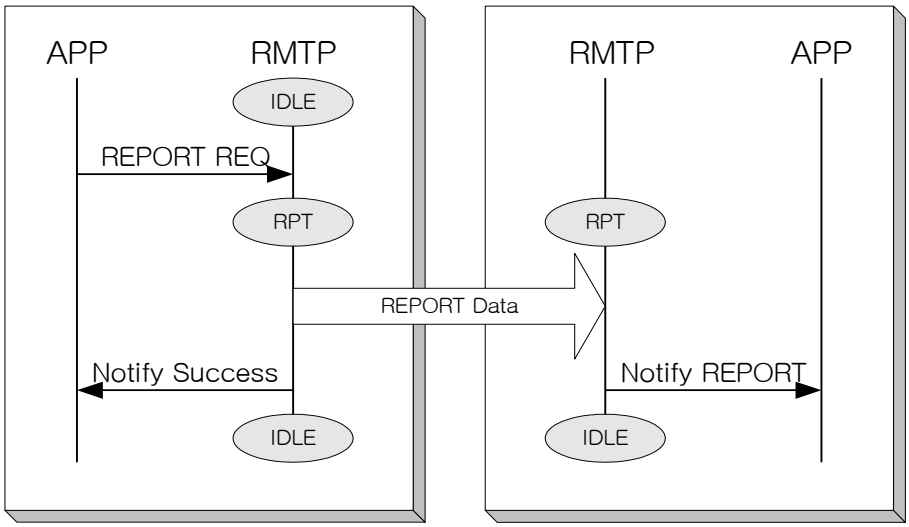


4.3. PHASE

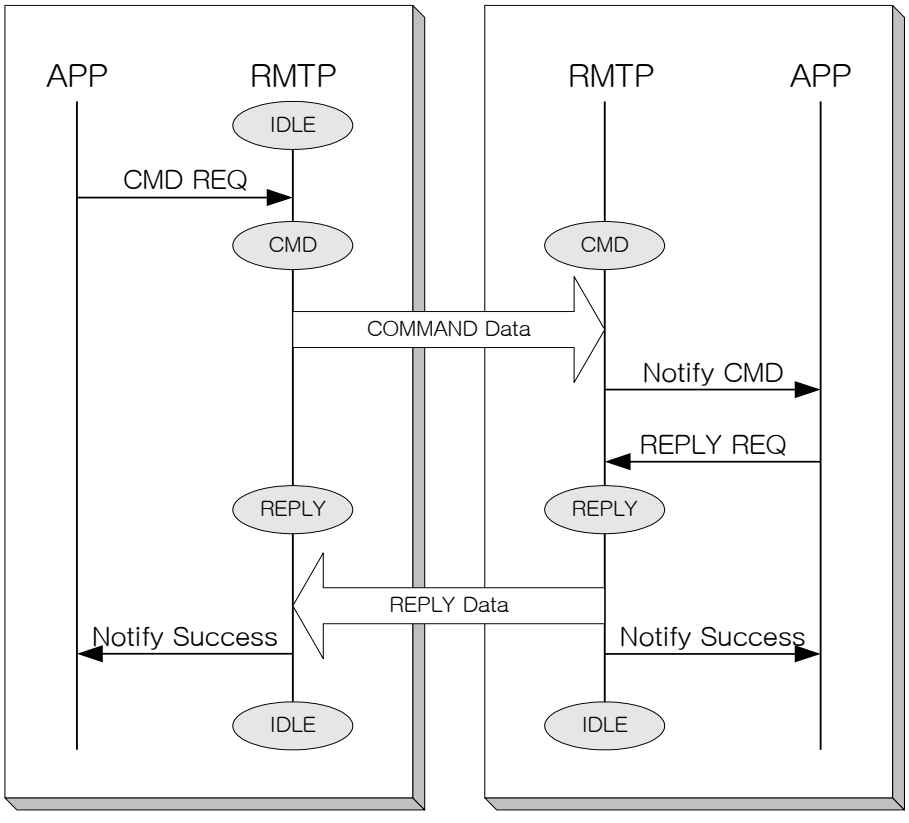
RMTP는 다음 그림과 같은 4개의 phase를 가진다. PHASE_COMMAND는 commander가 replier에게 명령을 전송하는 단계이고, PHASE_REPLY는 replier가 commander에게 응답을 전송하는 단계이다. PHASE_REPORT는 commander가 replier에게 보고를 전송하는 단계이다.



위의 네 개의 phase는 다음 예와 같이 변화한다.

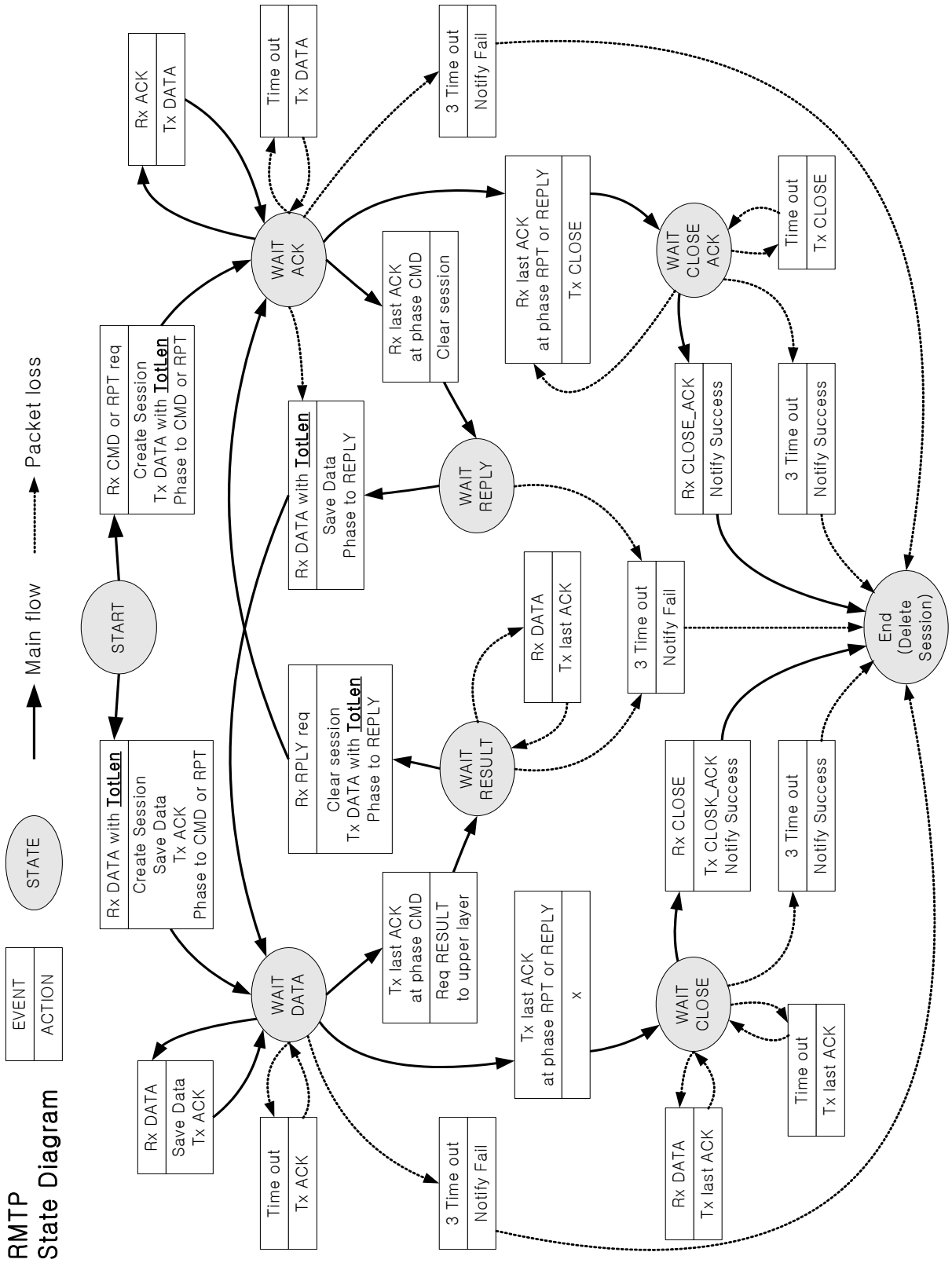


보고 메시지 전송시



명령/응답 메시지 전송시

4.4. State Transition Diagram



4.4.1. Events

RMTP상의 event는 다음과 같이 정의된다.

EVENT 이름	발생 상황
Rx REQUEST	상위 레이어로부터 request를 받음
Rx DATA	DATA 패킷을 받음.
Rx ACK	ACK 패킷을 받음.
Rx last ACK	마지막 데이터에 대한 ACK 패킷을 받음. (받은 ACK 패킷의 Sequence Number가 TotLen과 같음)
Rx CLOSE	CLOSE 패킷을 받음.
Rx CLOSE_ACK	CLOSE_ACK 패킷을 받음.
TIMEOUT	일정 시간동안 아무런 EVENT도 발생하지 않음.
3TIMEOUT	TIMEOUT EVENT가 연속으로 3회 발생함.

4.4.2. Actions

RMTP상의 action은 다음과 같이 정의된다.

ACTION 이름	설명
Create Session	새로운 세션을 생성한다. 이때, 상위 레이어의 request에 의해 생성하는 경우, Session ID를 생성하여 부여하고, DATA 패킷에 의해 생성하는 경우, DATA 패킷의 session ID를 그대로 사용한다.
Tx ACK	ACK 패킷을 보냄
Tx last ACK	마지막 데이터에 대한 ACK 패킷 보냄. (보낸 ACK 패킷의 Sequence Number가 TotLen과 같음)
Tx DATA	DATA 패킷을 보냄.
Save Data	DATA 패킷의 내용을 버퍼에 저장함.
Tx CLOSE	CLOSE 패킷을 보냄.
Tx CLOSE_ACK	CLOSE_ACK 패킷을 보냄.
Req RESULT	상위 레이어에 명령의 수신을 알리고, 응답을 요구함.
Notify	상위 레이어에 명령/응답/보고의 전송 성공 여부를 알림. 필요한 경우, 전송받은 데이터를 상위 레이어에 전달함.
Phase to	현재 세션의 PHASE를 변경함.
Clear Session	현재 세션을 초기화 함.
Delete Session	세션을 종료함.

4.5. Header Format

RMTP가 사용하는 패킷의 header format은 다음 그림과 같다.

TYPE (1 byte)	FLAG (1 byte)	LENGTH (2 byte)
Session ID (4 byte)		
Sequence Number (4 byte)		

4.5.1. TYPE

TYPE field에는 다음과 같은 메시지 타입이 들어간다.

- DATA
- ACK
- CLOSE
- CLOSE_ACK

4.5.2. FLAG

FLAG field는 각 비트별로 다음과 같은 의미를 가진다.

					REPORT	AckPlz	TotLen

TotLen: Sequence Number field의 값이 전송할 데이터의 전체 길이임을 나타낸다.

AckPlz: 수신측에게 ACK 패킷을 보내 달라는 요청.

REPORT: 현재 송신측이 PHASE_REPORT 단계임을 나타낸다.

4.5.3. LENGTH

패킷에 포함된 데이터의 길이를 나타낸다. 데이터가 없는 ACK, CLOSE, CLOSE_ACK 패킷의 경우에는 0.

4.5.4. Session ID

Session ID는 Commander 측에서 부여한다. Session ID의 처음 두 바이트는 자신의 장비/프로세서 고유번호를 사용하고, 나중 두 바이트는 새로운 session이 생성될 때 마다 1씩 증가시켜 부여하여, 전체 네트워크에서 다른 session에 대해 동일한 session ID가 부여되는 일을 방지한다. Replier 측에서는 Commander 측에서 부여한 session ID를 그대로 사용한다.

4.5.5. Sequence Number

DATA 패킷의 경우에는 현재 패킷에 실린 데이터의 첫번째 바이트와 전체 보내야 할 데이터의 offset을 의미하고, ACK 패킷의 경우에는 수신측이 다음 데이터로 원하는 바이트의 sequence number (next expected sequence number)를 의미한다.

4.6. Flow Control

RMTP의 flow control은 AckPlz flag와 sliding window를 사용하여 이루어진다.

RMTP는 큰 메시지를 전송해야 할 경우, 이 메시지를 일정한 크기(MTU)로 잘라서 보내게 된다. 이 때, SLIDING_WINDOW 만큼의 데이터를 MTU 단위로 잘라 전송하게 되며, 마지막 패킷에는 AckPlz FLAG를 세텅하여 전송한다.

데이터의 수신측에서는 받은 패킷의 sequence number를 검사하여 버퍼에 저장한다. 받은 패킷의 AckPlz flag가 세팅되어 있는 경우에는 ACK 패킷을 송신측으로 전송한다.

송신측에서 정해진 시간 내에 ACK 패킷을 받은 경우에는, 수신측에서 데이터를 충분히 처리할 능력이 있는 것으로 판단하여, SLIDING_WINDOW 크기를 MTU 만큼 늘리고, SLIDING_WINDOW 만큼의 데이터를 전송한다.

만약 송신측에서 일정한 시간 이내에 ACK 패킷을 받지 못하면 (timeout이 발생), SLIDING_WINDOW의 크기를 반으로 줄이고, 데이터를 재전송한다.

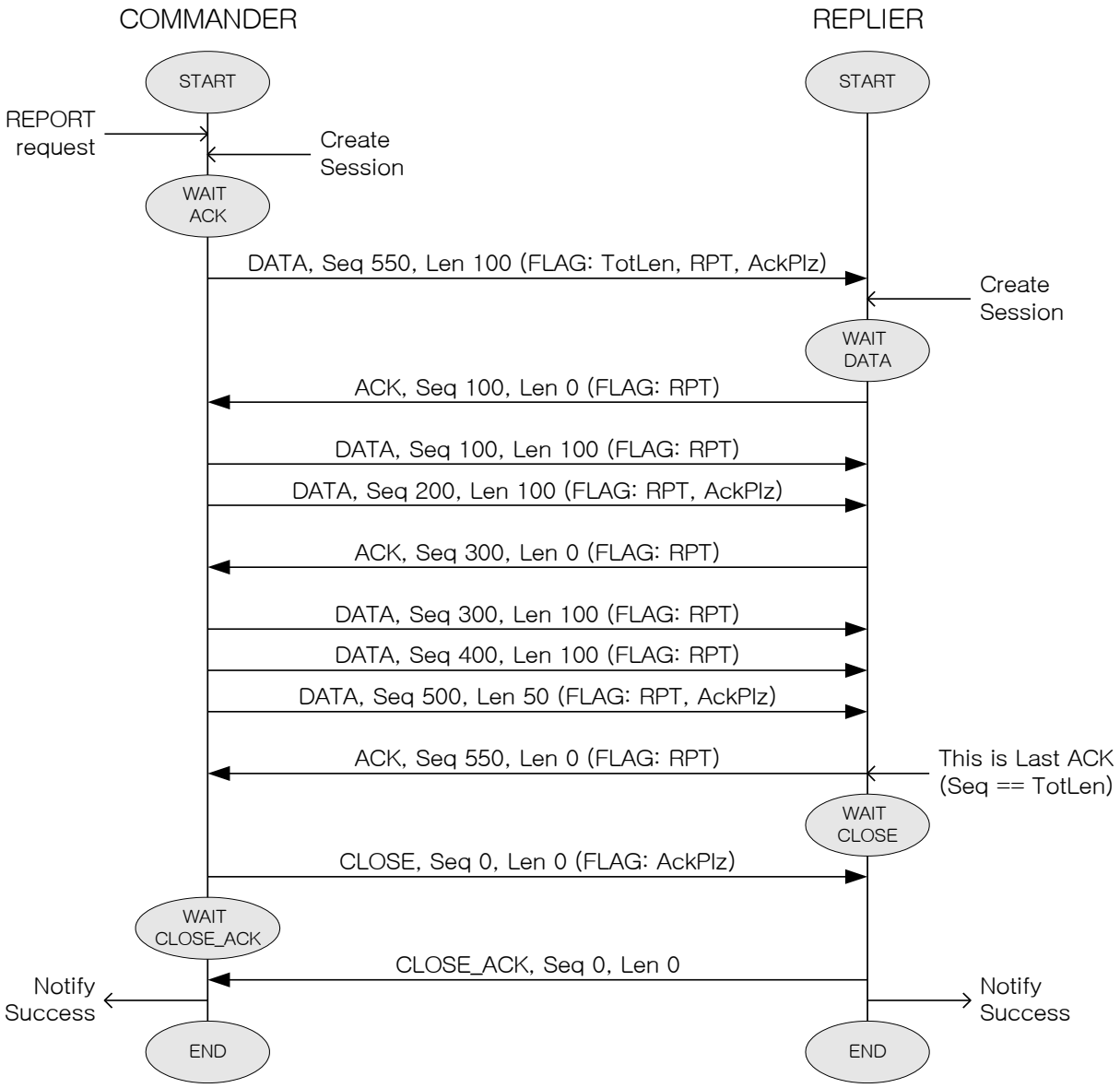
4.7. Error Control

RMTP의 error control은 sequence number를 사용하여 이루어진다.

수신측에서는 수신한 패킷의 sequence number를 검사하여, 중간에 유실된 패킷이 있다고 판단되면 그 이후에 받은 모든 패킷을 폐기한다. 만약 AckPlz flag가 세팅된 패킷을 수신한 경우에는 송신측으로 전송할 ACK 패킷에 자신이 앞으로 받기를 원하는 sequence number를 보내어 올바른 sequence의 데이터를 보내도록 유도한다.

4.8. Typical Scenario

다음 그림은 550 byte 길이의 보고 메시지를 전송할 때의 시나리오이다. (MTU = 100 Byte)



5. RMTP의 장점

5.1. Reliable

패킷의 전달에 대해 수신측에서 ACK 패킷을 보내므로, 송신측에서도 데이터의 확실한 전송 여부를 알 수 있다. 또한, sequence number를 사용하여, 패킷의 순서가 뒤바뀌거나, 재전송, 메시지 유실등의 오류를 복구할 수 있다.

5.2. Multisession

모든 상태에서의 event와 그에 대한 action이 다른 session에 독립적으로 정의되어 있고, packet format에 session ID를 이용하여 패킷의 세션을 구분할 수 있으므로, multisession 지원이 가능하다. Session ID는 Commander 측에서 부여하며, 맨 처음 2 byte는 자신의 프로세서의 고유번호를, 나중 2 byte는 일련번호를 부여함으로서 중복을 방지한다.

5.3. Transaction

COMMAND/REPLY/REPORT 세 종류의 phase를 둠으로써, 명령/응답의 경우 하나의 세션으로 처리가 가능하다. 또한 그 세션에 대해 메시지 전송의 성공 또는 실패 여부가 상위 레이어데 보고되므로 transaction 처리가 가능하다.

5.4. Simple

타 프로토콜은 다양한 종류의 네트워크 및 애플리케이션에 모두 적합하도록 디자인된 것이라, 많은 알고리즘이 들어있다. TCP/IP의 예를 들면, congestion control, delayed ACK, nagle algorithm, urgent mode, fast retransmission, fast recovery, sliding window, congestion window, persist timer, keep alive timer, checksum, half close 등의 많은 알고리즘이 적용되어 있다. 이것들의 대부분은 burst traffic이 주종을 이루는 IPC 환경에서는 필요없는 것들이다. 특히 checksum은 대부분의 하위 프로토콜에서 지원하는 것으로, transport layer에서 다시 검사하는 것은 오버헤드라고 볼 수 있다.

RMTP에서는 IPC 환경에 꼭 필요한 요소 (sliding window, sequence number, ACK)만을 채택하여, simple하고 가벼운 구현이 가능하다.

5.5. Robust

모든 state에서 timeout을 검사하여 timeout이 발생한 경우 재전송을 하고, 3회 연속 timeout이 발생한 경우에는, 자동적으로 세션이 종료되고 상위로 상황이 보고된다. 따라서, 상대방 장비/프로세

서의 갑작스런 다운이나 링크의 절단과 같은 상황에도 대처할 수 있다.

6. User's Guide

6.1. Porting Guide

현재 RMTP는 Linux 상에서 구현 및 시뮬레이션이 완료된 상태이다. RMTP의 소스코드는 다음과 같은 파일로 구성되어 있다.

파일 이름	설명
rmtp_core.c, rmtp_core.h	RMTP의 주요 루틴.
sestable.c, sestable.h	Session table을 관리하는 루틴.
rmtp_if.c, rmtp_if.h	RMTP와 상위/하위 레이어와의 인터페이스 루틴.

RMTP 소스코드를 타 시스템으로 포팅하기 위해서는 rmtp_if.c, rmtp_if.h 파일을 수정해야 한다.

6.1.1. Porting Functions

rmtp_if.c 파일에 구현된 주요 함수 및 포팅 방법은 다음과 같다.

함수 명	기능	포팅 방법
rmtp_initlalize()	RMTP 모듈 초기화.	변경하지 말 것.
rmtp_periodic_routine()	timeout check.	변경하지 말 것.
rmtp_receive_packet()	하위 레이어로부터 패킷 수신.	변경하지 말 것.
rmtp_request_report()	상위 레이어로부터 보고 요구 수신.	변경하지 말 것.
rmtp_request_command()	상위 레이어로부터 명령 요구 수신.	변경하지 말 것.
rmtp_request_reply()	상위 레이어로부터 응답 요구 수신.	변경하지 말 것.
rmtp_set_timeout()	명령 처리후 응답 발생 까지 시간이 많이 걸릴 경우, 이 함수를 호출하여 timeout 시간을 바꿀 수 있음.	변경하지 말 것.
rmtp_get_buffer()	데이터 저장용 버퍼 요구	버퍼 요구 부분 변경 요망.
rmtp_release_buffer()	데이터 저장용 버퍼 해제	버퍼 해제 부분 변경 요망.
rmtp_get_second()	현재 초단위 시간을 얻음.	현재 초단위 시간을 얻는 부분 변경 요망.
rmtp_get_msecond()	현재 mili 초단위 시간을 얻음.	현재 mili 초단위 시간을 얻는 부분 변

		경 요망. 구현이 어려울 경우, 변경하지 말 것.
rmtp_send_packet()	하위 레이어로 패킷 송신.	하위 레이어 호출 부분 변경 요망.
rmtp_finish_report()	보고 단계에서 세션이 종료되었음을 상위 레이어에게 알림.	상위 레이어 호출 부분 변경 요망.
rmtp_finish_command()	명령 단계에서 세션이 종료되었음을 상위 레이어에게 알림.	상위 레이어 호출 부분 변경 요망.
rmtp_finish_reply()	응답 단계에서 세션이 종료되었음을 상위 레이어에게 알림.	상위 레이어 호출 부분 변경 요망.
rmtp_notify_command()	상위 레이어로 명령의 수신을 알림. 상위에서는 명령 처리후 request_reply() 함수를 호출함.	상위 레이어 호출 부분 변경 요망. 상위 레이어에서는 세션ID를 보관하고 있다가, 응답 전송시에 파라미터로 넘겨줘야 함.

6.1.2. Configuration

rmtp_if.h에 정의된 주요 파라미터는 다음과 같다.

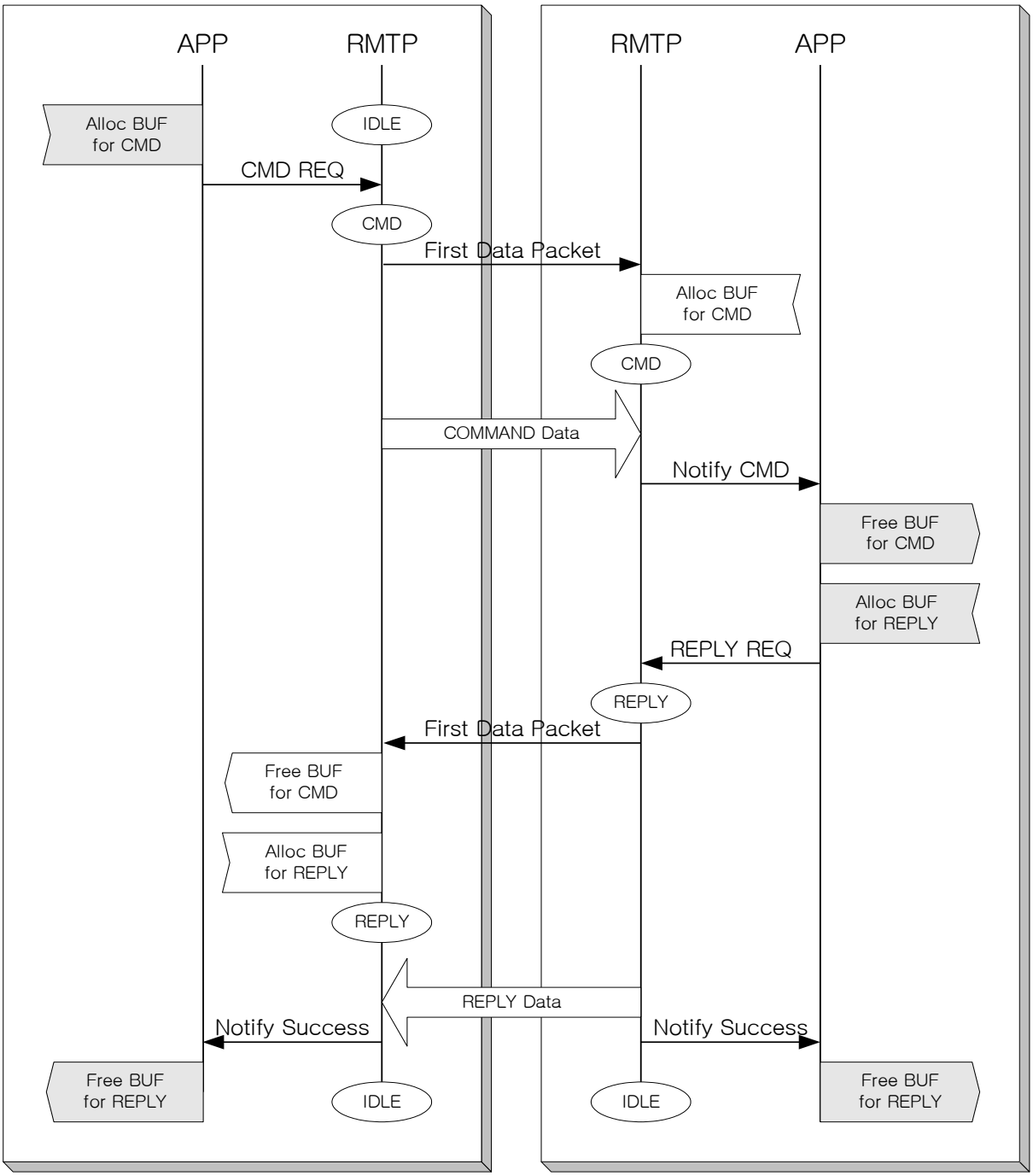
정의 이름	설명
UNIT_WIN_SIZE	MTU
MAX_WIN_SIZE	Sliding window의 최대 크기
RMTP_TIMEOUT_SEC	Timeout 초 값
MAX_SESSION_ENTRY	RMTP 모듈에서 최대 가능한 세션 개수
RMTP_BIG_ENDIAN	Big-endian을 사용하는 시스템인 경우, 정의함. Little-endian을 사용하는 시스템에서는 정의하지 않음.

포팅시에, 위의 파라미터를 환경에 맞는 적당한 값으로 변경한다.

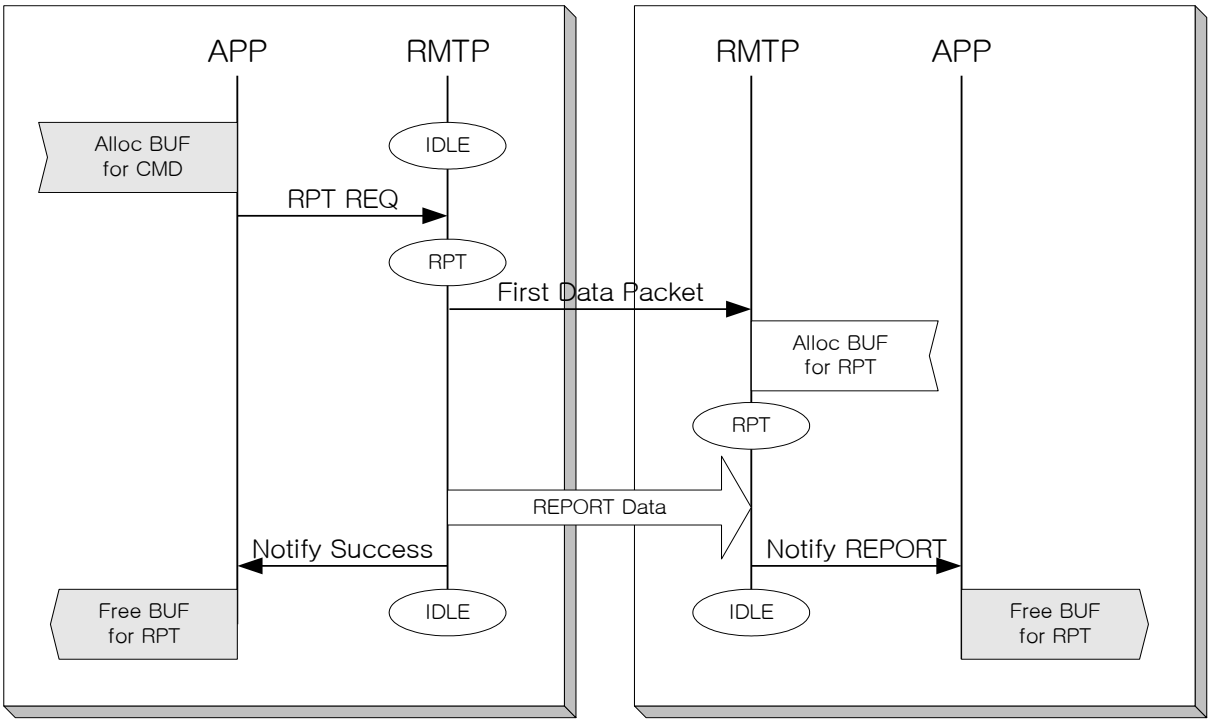
6.1.3. Memory Management

RMTP 모듈은 rmtp_get_buffer(), rmtp_release_buffer() 함수를 사용하여 메모리를 할당/해제 한다.

다음 그림은 RMTP를 운용할 경우 메모리를 할당/해제하는 방법을 보여준다.



명령/응답 메시지 전송시 메모리의 할당/해제



보고 메시지 전송시 메모리의 할당/해제

6.2. 성능

다음 표는 Linux 상에서 수행된 RMTP 모듈의 시뮬레이션 결과이다.

시험 환경:

- MTU = 116 Byte
- MAX_WIN_SIZE = 1160 Byte
- 명령/응답, 보고 메시지를 반복해서 전송함

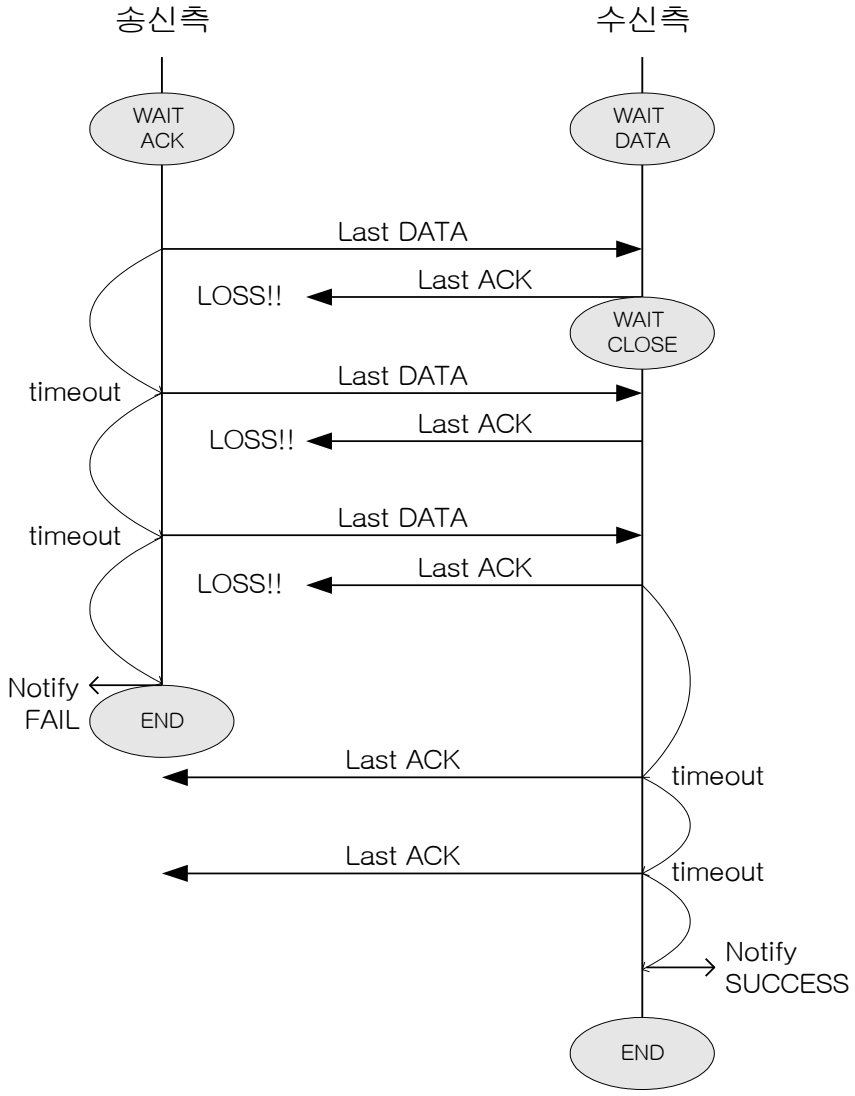
시험 결과 세션의 성공률:

Msg size / Pkt Loss	100	500	1000	10000	100000	2000000
0 %	100 %	100 %	100 %	100 %	100 %	100 %
1 %	100 %	100 %	100 %	99.80 %	100 %	97.74 %
2 %	100 %	100 %	100 %	100 %	99.48 %	84.38 %
5 %	100 %	99.52 %	99.38 %	96.54 %	83.66 %	4.16 %
10 %	99.85 %	97.00 %	96.18 %	81.05 %	19.13 %	0 %

6.3. Known Bug

전송 에러가 발생하지 않는 상황에서는 송신측과 수신측 사이의 transaction이 완전히 일치한다. (메시지의 전송 성공 여부가 양측에 동일하게 인식된다.)

그러나, 전송 에러가 발생하여 패킷의 손실이 생기면, 송신측과 수신측 사이의 transaction 불일치가 발생할 수 있다. 즉, 송신측에서는 전송 실패라고 인식하고, 수신측에서는 전송 성공이라고 인식하는 경우가 생길 수 있다. 이 현상은 다음 그림과 같은 경우에 발생한다.



위의 경우, 송신측에서는 전송 실패, 수신측에서는 전송 성공으로 인식된다.

시뮬레이션 결과, 패킷 로스가 심한 상황에서 (packet loss 10 %) 위와 같은 트랜잭션 불일치 현상이 0.2 %의 확률로 발생한다. 패킷 로스가 5 % 이하의 경우에서는 발생하지 않았다.