2026/10/09, 2026 보안 기초 세미나

# TCP/IP 프로토콜

## - 16장 SCTP, 17장 응용계층 개요 -

이 성 현(seonghyeon@pel.sejong.ac.kr)

세종대학교 프로토콜공학연구실

# 목 차

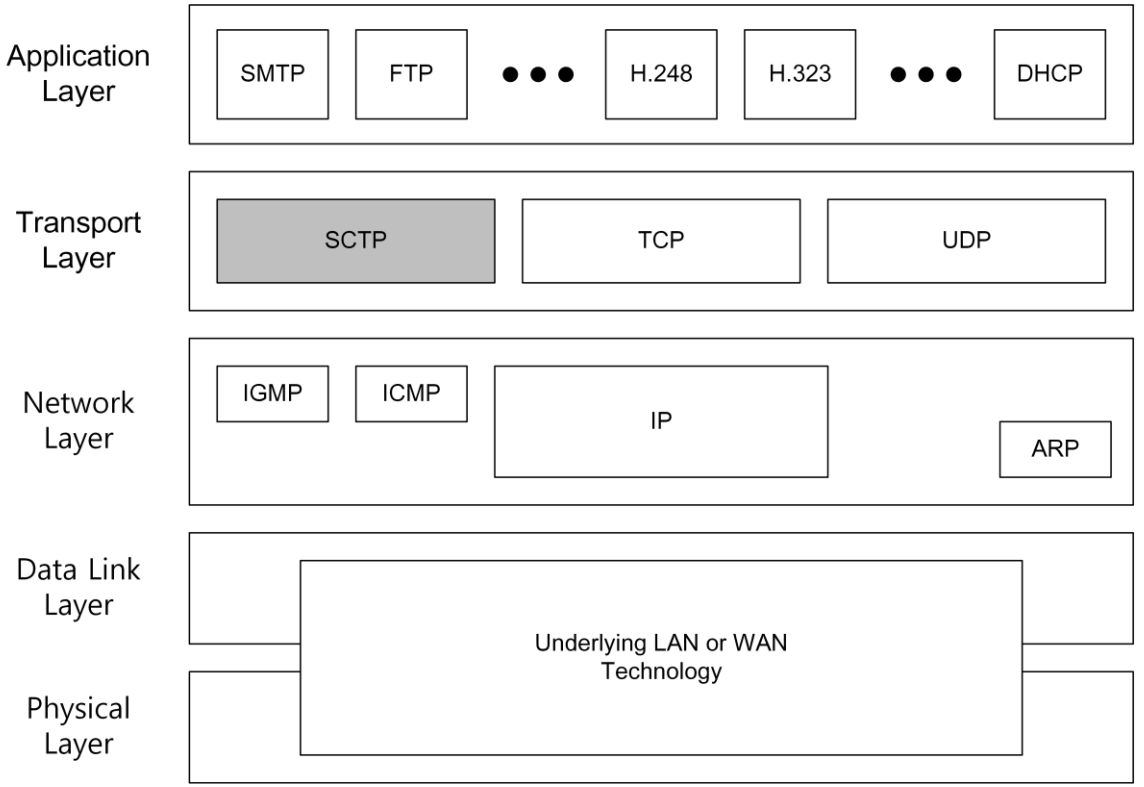
- SCTP
- 응용계층 개요

# 목 차

- SCTP
- 응용계층 개요

# SCTP

- 개요 (1/2)
  - SCTP (Stream Control Transport Protocol)는 응용계층과 네트워크 계층 사이에서 응용 프로그램과 네트워크 동작 간의 매개체 역할을 함



# SCTP

- 개요 (2/2)
  - UDP (User Daram Protocol)는 메시지 지향적인 프로토콜이지만, 비신뢰성 프로토콜임
    - 실시간 데이터 전송에 유용하나, 전송한 데이터가 제대로 수신됐는지는 확인하기 어려움
  - TCP (Transmisson Control Protocol)는 신뢰성 프로토콜이지만, 바이트 지향적인 프로토콜임
    - 오류 제어를 통해 전체 스트림을 순서에 맞춰 오류 없이 전송하는 것을 보장하나, 메시지 경계가 존재하지 않음
  - SCTP는 메시지 지향적인 프로토콜이면서 신뢰성 프로토콜임
    - 메시지 경계가 존재하면서 오류 제어 메커니즘을 통해 신뢰성 있는 전송을 보장함

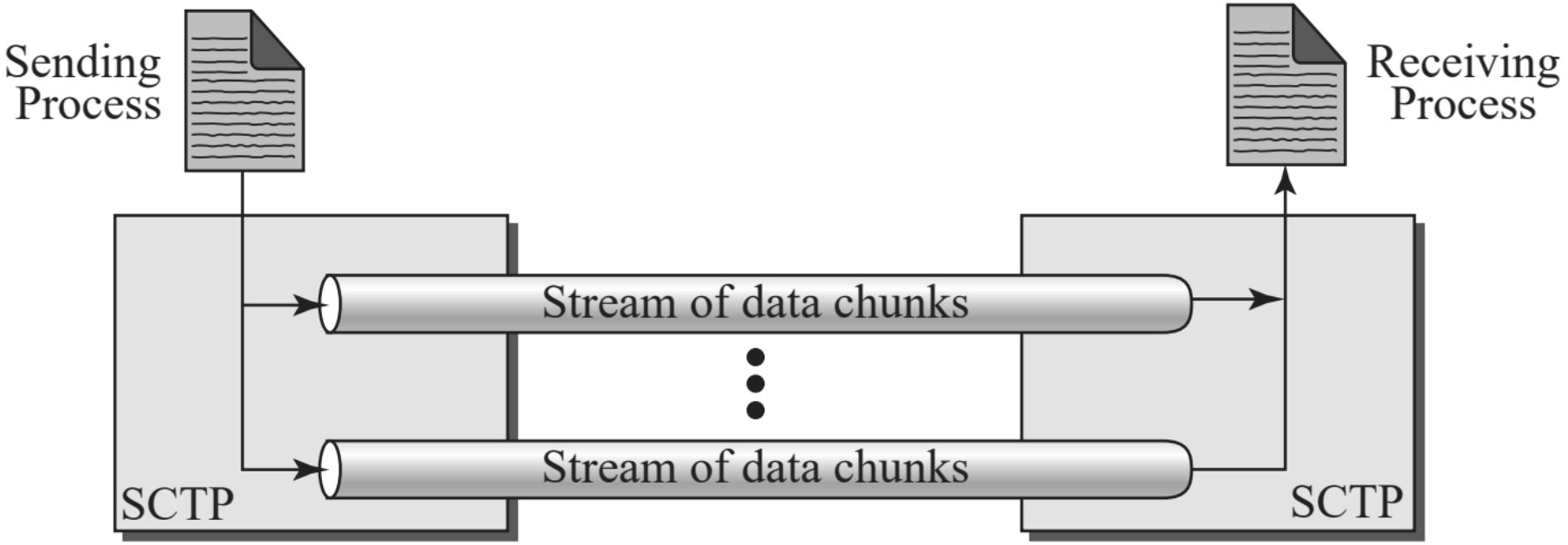
# SCTP

- 서비스 (1/4)
  - 프로세스 대 프로세스 통신
    - SCTP는 TCP 및 UDP와 마찬가지로 포트 번호를 사용하여 프로세스 대 프로세스 통신을 지원함

| Protocol | Port Number | Description |
|---|---|---|
| IUA | 9990 | IP 기반 ISDN 신호 전달 |
| M2UA | 2904 | SS7 전화 신호 전달 |
| M3UA | 2905 | SS7 전화 신호 전달 |
| H.248 | 2945 | 미디어 게이트웨이 제어 |
| H.323 | 1718, 1719, 1720, 11720 | IP 전화 통신 |
| SIP | 5060 | IP 전화 통신 |

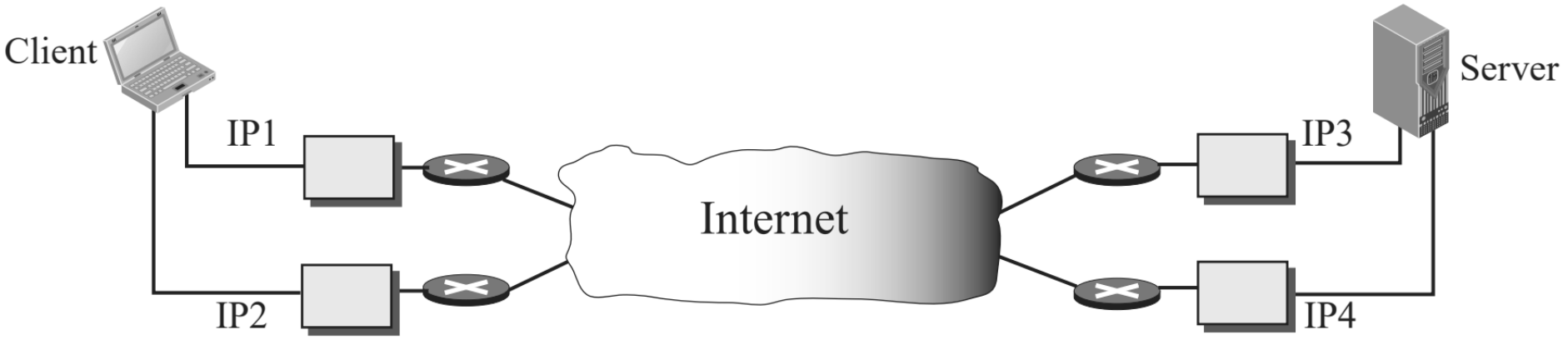
# SCTP

- 서비스 (2/4)
  - 다중스트림
    - SCTP는 어소시에이션(Association)이라고 불리는 연결마다 다중스트림 서비스를 제공하여 스트림 중 하나가 블록되어도 다른 스트림이 계속 데이터 전송이 가능함



# SCTP

- 서비스 (3/4)
  - 멀티호밍(Multi-homing)
    - 송신 및 수신 호스트는 한 어소시에이션에 해당하는 각 단말의 다중 IP 주소들을 정의할 수 있음
    - 장애감내(Fault-tolerant) 관점에서, 하나의 경로가 실패하면 다른 인터페이스가 인터럽트 없이 데이터를 전달 가능함
    - 다른 경로 간 부하 공유(Load Sharing)은 허용하지 않으며, 통신에는 한 쌍의 IP 주소만을 사용함



# SCTP

- 서비스 (4/4)
  - 전이중 통신
    - SCTP의 송수신 호스트는 각각 전송 버퍼와 수신 버퍼를 가지고 데이터를 양방향으로 보낼 수 있는 전이중 서비스 (Full-duplex Service)를 제공함
  - 연결 지향 서비스
    - 두 SCTP가 상호 간에 어소시에이션 수립, 양방향 데이터 교환, 어소시에이션 종료 순서로 진행되는 연결 지향 서비스를 지원함
  - 신뢰할 수 있는 서비스
    - 안전하고 정상적인 데이터 도착을 보장하기 위해 확인응답 메커니즘(Acknowledgment Mechanism)을 사용함

# SCTP

- 특징 (1/2)
  - 청크 및 스트림 단위 관리
    - 전송 순서 번호(TSN, Transmisson Sequence Number)
      - SCTP의 데이터 단위는 데이터 청크(Chunk)이며, 32 비트 길이의 전송 순서 번호를 통해 번호화하여 제어함
    - 스트림 식별자(SI, Stream Identifier)
      - 한 어소시에이션에 여러 개의 스트림이 존재할 수 있으므로, 16 비트 식별자로 각 스트림을 구분함
    - 스트림 순서 번호(SSN, Stream Sequence Number)
      - 같은 스트림에 속하는 다른 데이터 청크들을 구분하기 위해 16 비트 식별자를 사용함
    - 확인응답 번호
      - 확인응답은 청크 지향적으로 TSN을 이용하여 이루어지며, 제어 청크는 TSN을 소모하지 않음

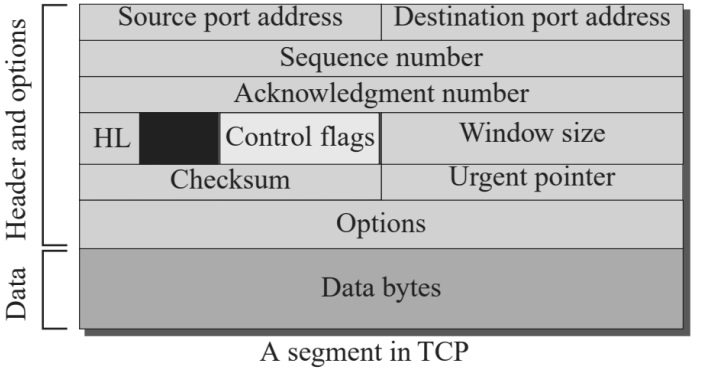
# SCTP

- 특징 (2/2)
  - 흐름 제어
    - 수신자가 처리할 수 있는 양보다 많은 데이터를 받는 것을 방지하기 위해 흐름 제어 기능을 제공함
  - 오류 제어
    - 신뢰성을 제공하기 위해 TSN와 확인응답 번호를 이용하여 오류제어를 제공함
  - 혼잡 제어
    - 혼잡 상태를 고려하여 송신자가 한 번에 네트워크로 전송할 데이터 청크의 양을 조절하기 위해 혼잡 제어를 제공함

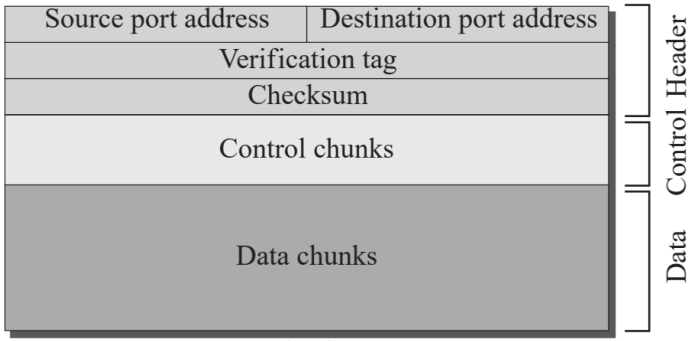
# SCTP

- 패킷 (1/29)
  - SCTP는 메시지 단위로 패킷을 사용함
  - 데이터는 데이터 청크, 제어 정보는 제어 청크를 통해 전송됨

| 비교 항목 | TCP | SCTP |
|---|---|---|
| 패킷 구조 | 헤더 + 데이터 | 공통 헤더 + 여러 청크 |
| 제어 정보 | 헤더에 포함 | 제어 청크에 포함 |
| 데이터 | 하나의 바이트 스트림 | 여러 DATA 청크, 스트림 |
| 옵션/확장 | Options 필드 | 새로운 Chunk Type |
| 기본 헤더 길이 | 최소 20바이트 | 12바이트 |
| Checksum | 16비트 | 32비트 |
| 순서 관리 | Sequence Number | TSN + SI + SSN |
| 연결 식별 | Verification Tag 없음 | Verification Tag 사용 |



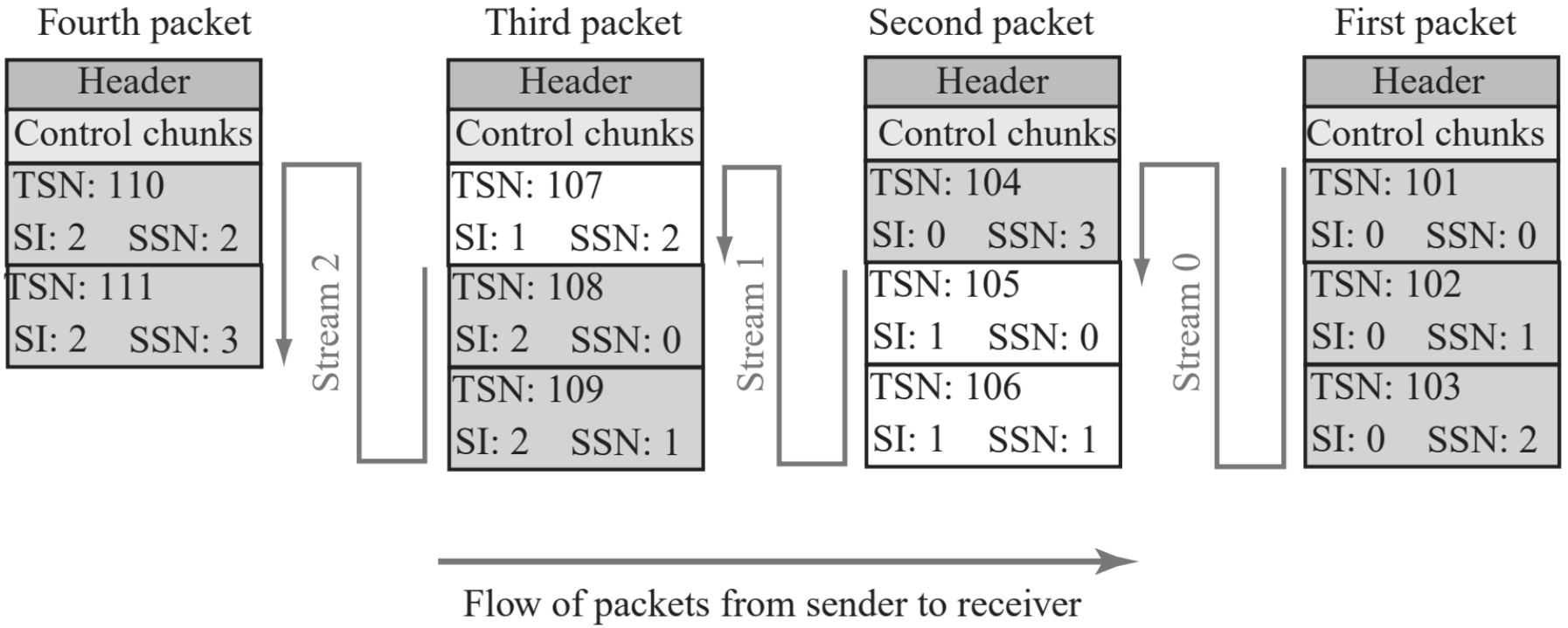
A segment in TCP



A packet in SCTP

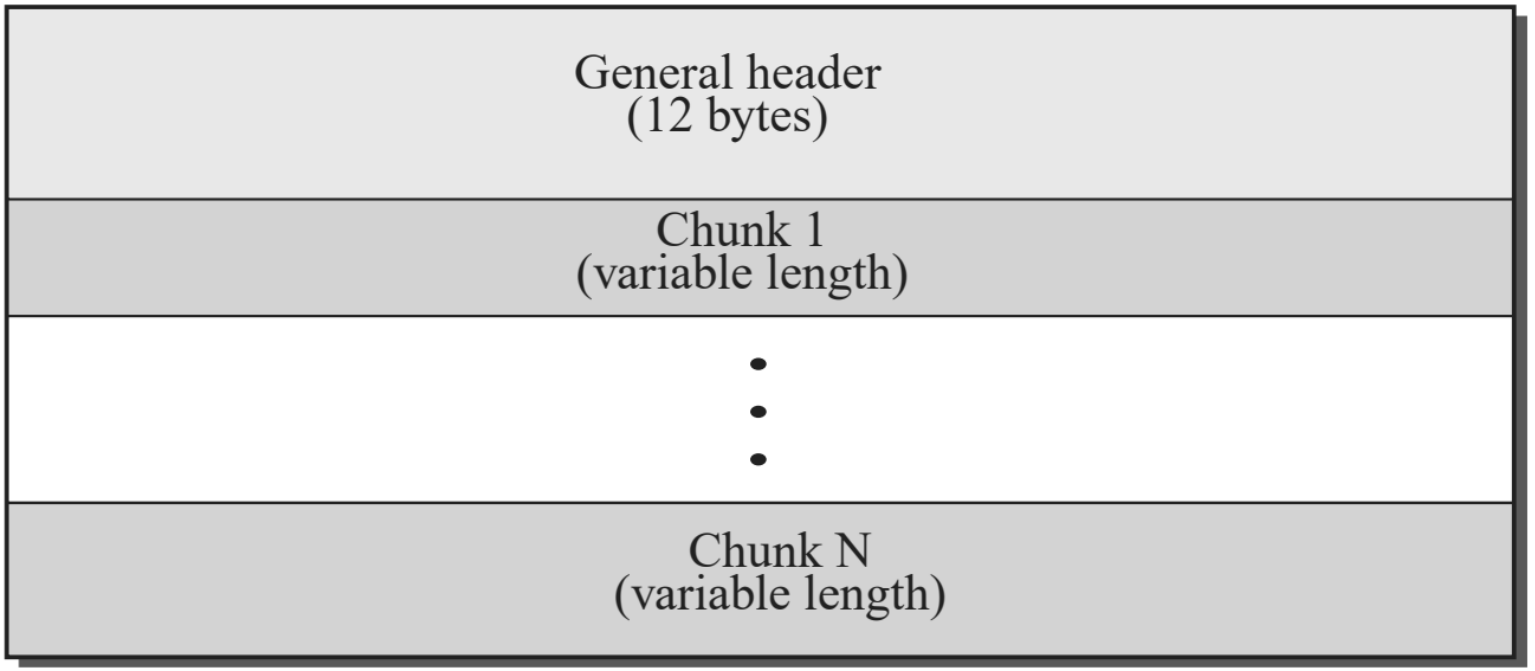
# SCTP

- 패킷 (2/29)
  - 데이터 청크는 어소시에이션 내 청크 순서를 정의하는 TSN, 스트림을 정의하는 SI, 스트림 내 청크 순서를 정의하는 SSN에 의해 구분됨



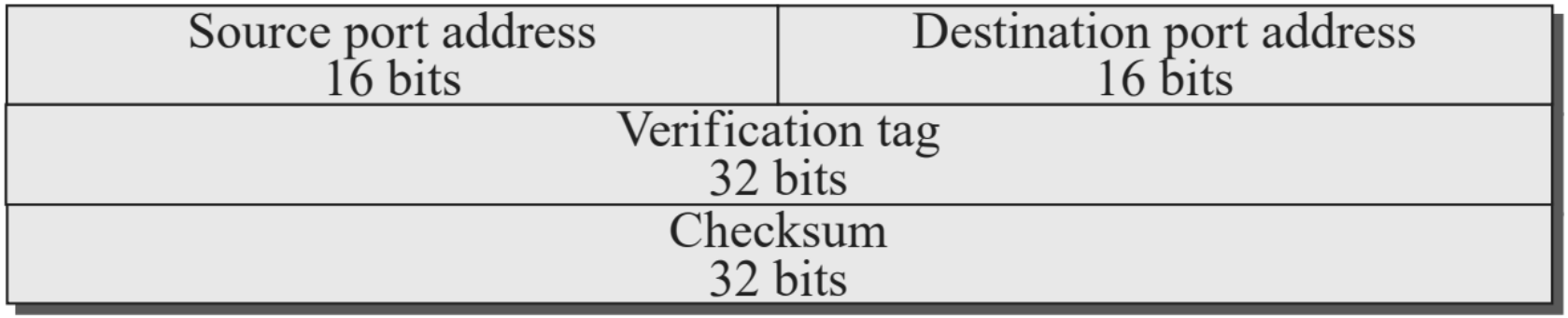
# SCTP

- 패킷 (3/29)
  - SCTP 패킷은 일반 헤더와 청크라 불리는 블록으로 구성되며, 청크는 어소시에이션을 제어하고 유지하는 제어 청크와 사용자 데이터를 제어하는 데이터 청크로 구분됨



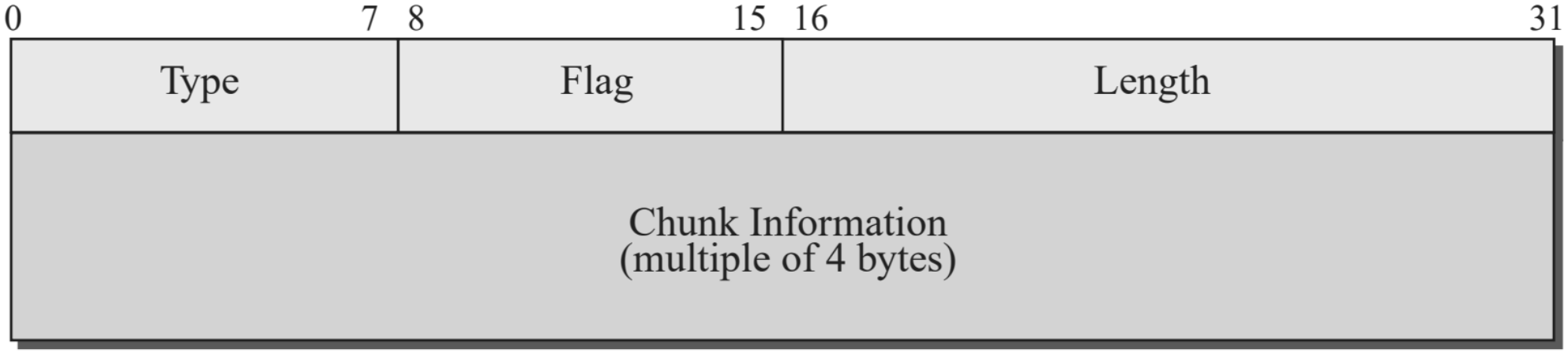
# SCTP

- 패킷 (4/29)
  - 일반 헤더
    - 발신지 포트 주소(Source Port Address)
      - 16 비트 필드로 패킷을 송신하는 프로세스의 포트 번호를 정의함
    - 도착지 포트 주소(Destination Port Address)
      - 16 비트 필드로 패킷을 수신하는 프로세스의 포트 번호를 정의함
    - 확인 태그(Verification Tag)
      - 32 비트 필드로 각 어소시에이션에 대한 식별자 역할을 수행함
    - 검사합(Checksum)
      - 32 비트 필드로 CRC-32 검사합을 포함함



# SCTP

- 패킷 (5/29)
  - 청크 (1/25)
    - 공통 형식
      - 개요
        - 모든 청크는 3개의 공통 필드와 청크 정보(Chunk Information) 필드를 포함함
        - 32 비트 단위로 구성되며, 청크 정보 필드가 32 비트 단위가 아닌 경우에는 패딩 바이트를 추가하여 32 비트 단위로 구성함



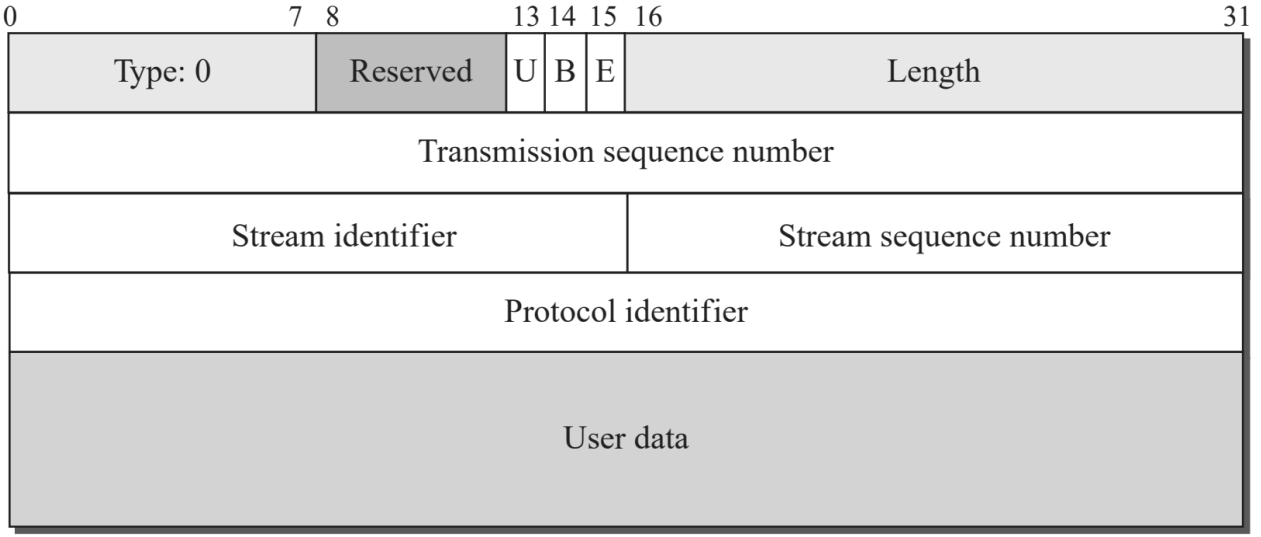
# SCTP

- 패킷 (6/29)
  - 청크 (2/25)
    - 공통 형식
      - 유형(Type)
        - 8 비트 필드로 청크의 유형을 정의함
      - 플래그(Flag)
        - 8 비트 필드로 특정 청크 유형이 필요로 하는 플래그 값으로 구성됨
      - 길이(Length)
        - 16 비트 필드로 유형, 플래그, 길이 필드를 포함한 청크의 전체 크기를 바이트 단위로 표현함
        - 패딩 바이트는 길이 필드 값을 계산할 때 포함되지 않음

| *Type* | *Chunk* | *Description* |
|---|---|---|
| 0 | DATA | User data |
| 1 | INIT | Sets up an association |
| 2 | INIT ACK | Acknowledges INIT chunk |
| 3 | SACK | Selective acknowledgment |
| 4 | HEARTBEAT | Probes the peer for liveliness |
| 5 | HEARTBEAT ACK | Acknowledges HEARTBEAT chunk |
| 6 | ABORT | Abort an association |
| 7 | SHUTDOWN | Terminates an association |
| 8 | SHUTDOWN ACK | Acknowledges SHUTDOWN chunk |
| 9 | ERROR | Reports errors without shutting down |
| 10 | COOKIE ECHO | Third packet in association establishment |
| 11 | COOKIE ACK | Acknowledges COOKIE ECHO chunk |
| 14 | SHUTDOWN COMPLETE | Third packet in association termination |
| 192 | FORWARD TSN | For adjusting cumulating TSN |

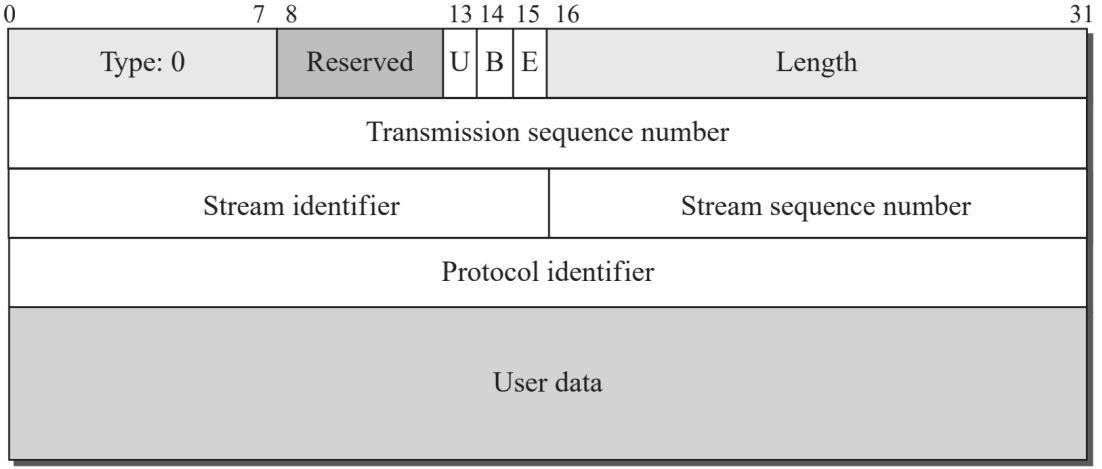
# SCTP

- 패킷 (7/29)
  - 청크 (3/25)
    - DATA
      - 개요
        - 사용자 데이터를 전송하는 청크이며, 플래그 필드에서 비순서화 여부를 나타내는 U 비트, 단편화된 메시지 중 첫 단편임을 나타내는 B 비트, 마지막 단편임을 나타내는 E 비트를 사용함
        - 사용 시 최소 한 바이트 이상의 데이터를 포함해야 하므로, 길이는 최소 17 바이트 이상이 됨



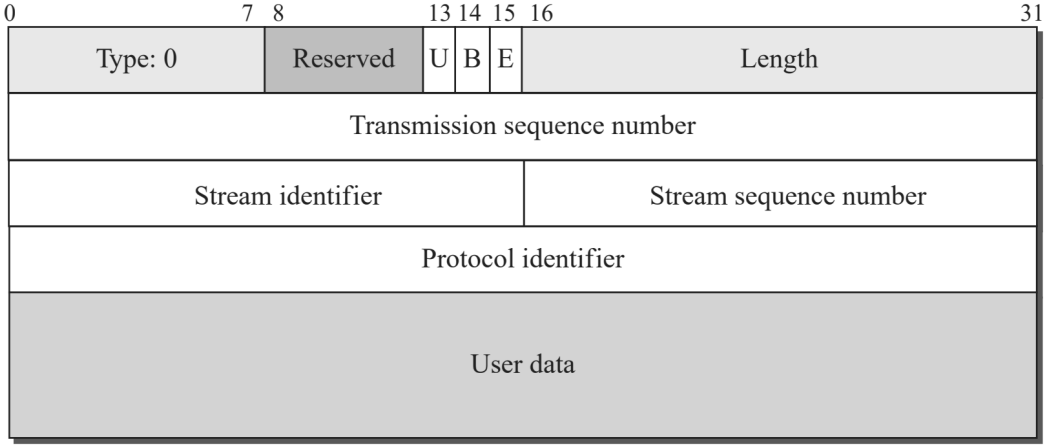
# SCTP

- 패킷 (8/29)
  - 청크 (4/25)
    - DATA
      - 전송 순서 번호(TSN)
        - 32 비트 필드로 전송 순서 번호를 정의하며, 송수신자 간 통일하지는 않음
      - 스트림 식별자(SI)
        - 16 비트 필드로 한 어소시에이션 내 각 스트림을 정의함
      - 스트림 순서 번호(SSN)
        - 16 비트 필드로 한 스트림 내에서 각 청크를 정의함



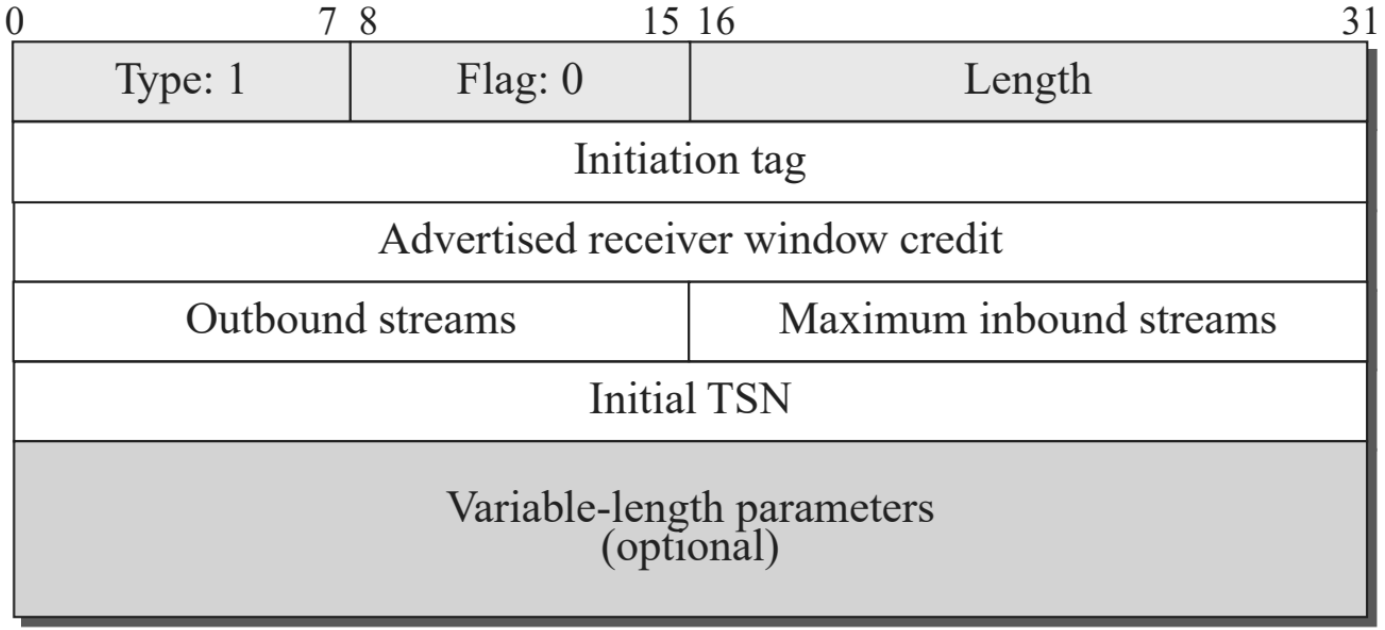
# SCTP

- 패킷 (9/29)
  - 청크 (5/25)
    - DATA
      - 프로토콜 식별자(Protocol Identifier)
        - 32 비트 필드로 응용 프로그램이 데이터의 유형을 정의하기 위해 사용됨
      - 사용자 데이터(User Data)
        - 최소 한 바이트 이상이어야 하며, 32 비트 단위가 아닐 경우 패딩 바이트를 추가하여 32 비트 단위를 맞춰야 함
        - 하나의 메시지는 여러 개의 청크로 분할될 수 있으나, 하나의 DATA 청크는 두 개 이상의 메시지에 속할 수 없음



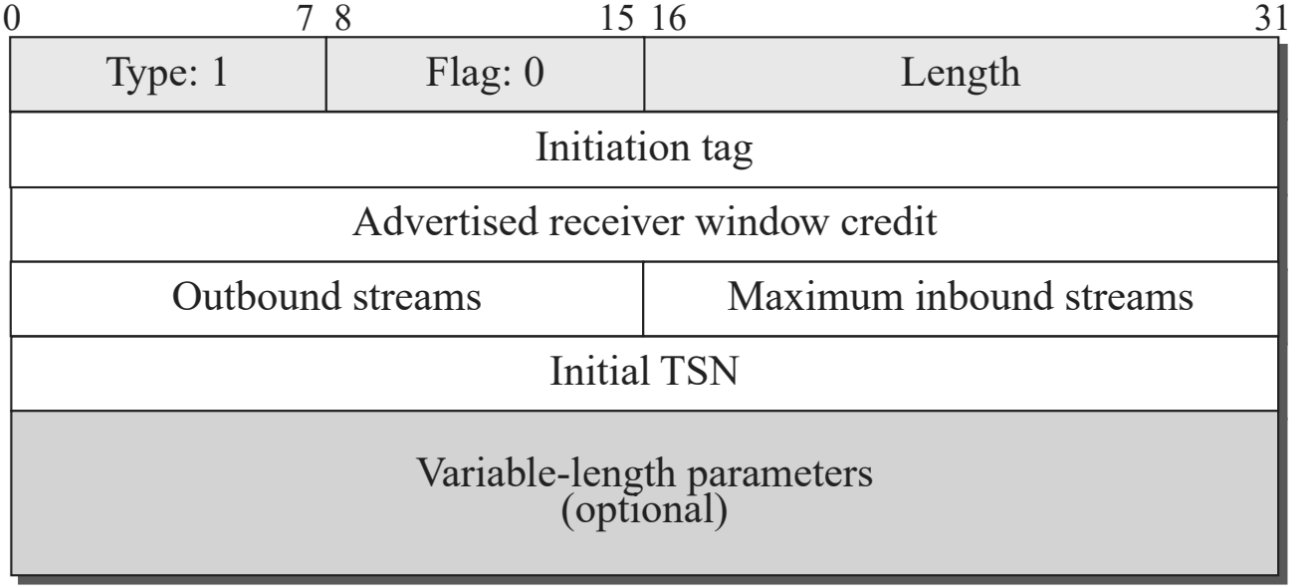
# SCTP

- 패킷 (10/29)
  - 청크 (6/25)
    - INIT
      - 개요
        - 어소시에이션을 수립하기 위해 가장 먼저 송신해야 함
        - 해당 청크를 전송하는 패킷은 다른 청크를 함께 전송할 수 없으며, 확인 태그가 0으로 설정됨



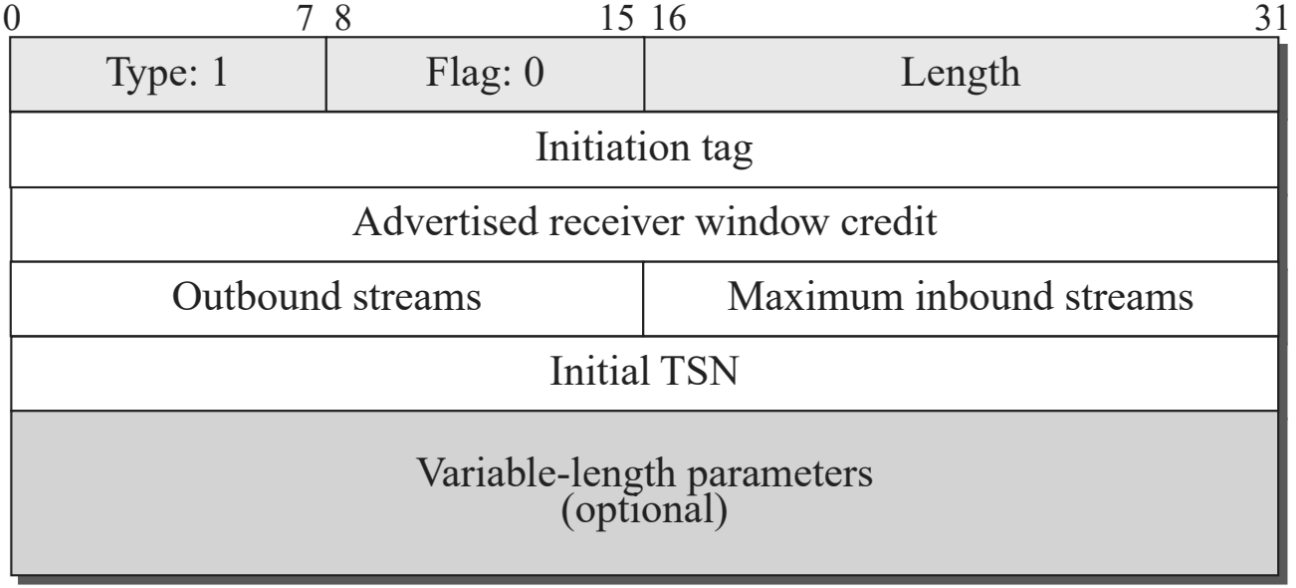
# SCTP

- 패킷 (11/29)
  - 청크 (7/25)
    - INIT
      - 초기화 태그(Initiation Tag)
        - 32 비트 필드로 상대가 전송하는 패킷의 확인 태그를 정의함
      - 광고된 수신자 윈도우 크레딧(Advertised Receiver Window Credit)
        - 32 비트 필드로 흐름 제어에 사용되는 rwnd 값이 바이트 단위로 정의됨



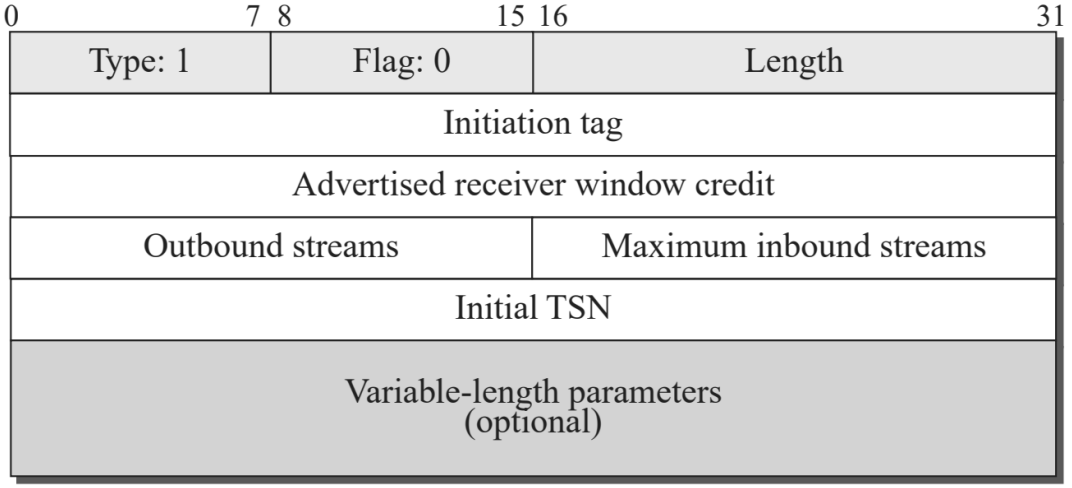
# SCTP

- 패킷 (12/29)
  - 청크 (8/25)
    - INIT
      - 송신 스트림(Outbound Stream)
        - 16 비트 필드로 송신에 사용하고자 하는 스트림의 최대 개수를 정의함
      - 최대 수신 스트림(Maximum Inbound Stream)
        - 16 비트 필드로 허용하고자 하는 수신 스트림의 최대 개수를 정의함



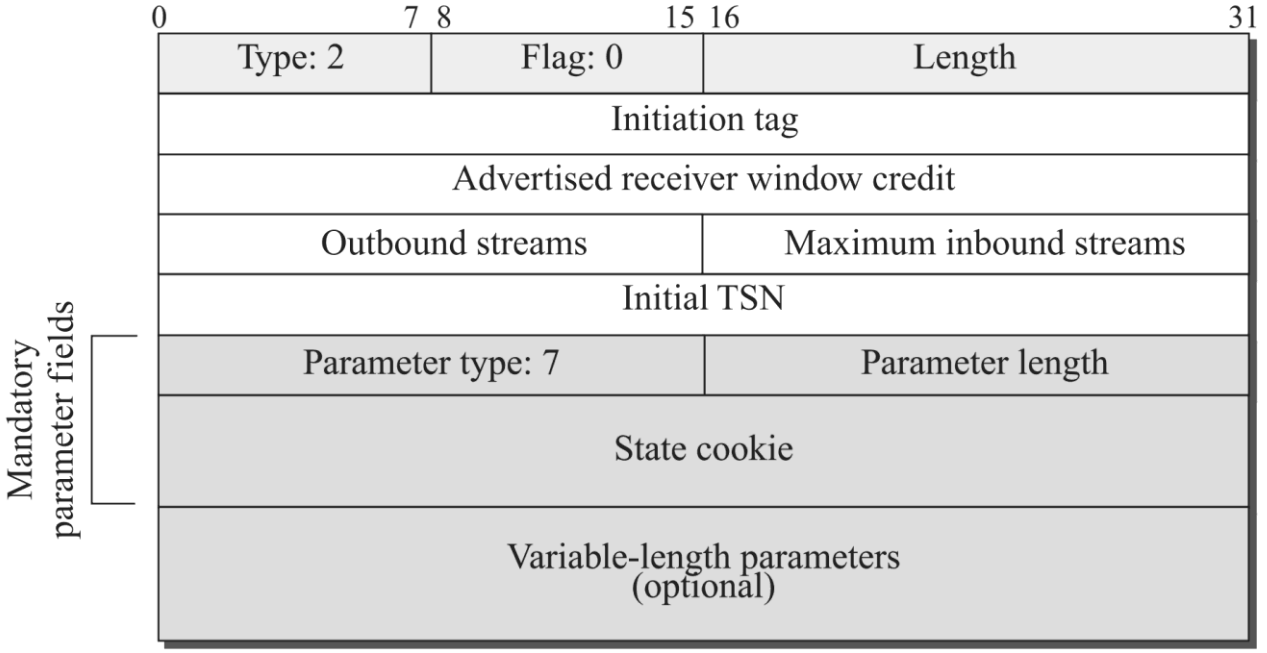
# SCTP

- 패킷 (13/29)
  - 청크 (9/25)
    - INIT
      - 초기 TSN(Initial TSN)
        - 32 비트 필드로 송신에 사용할 전송 순서 번호를 초기화함
      - 가변 길이 파라미터(Variable-length Parameter)
        - 송신 IP 주소, 지원 가능한 IP 주소 개수, 쿠키 상태 유지, 주소 유형, 명확한 혼잡 통지(ECN, Explicit Congestion Notification) 지원 여부 등을 정의하기 위해 사용될 수 있음



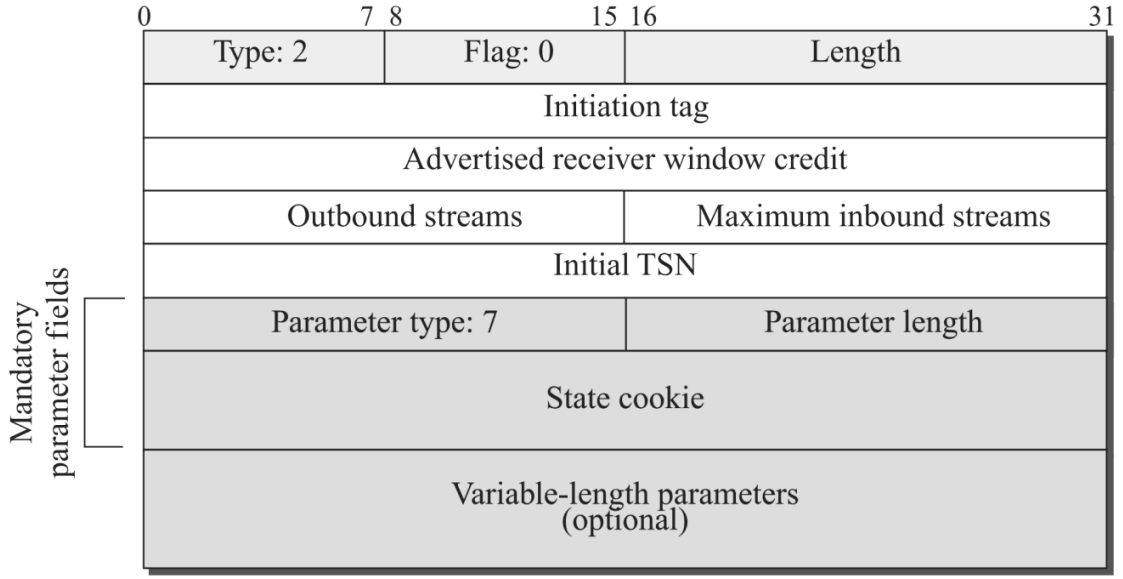
# SCTP

- 패킷 (14/29)
  - 청크 (10/25)
    - INIT ACK
      - 개요
        - INIT 청크와 동일한 필드들로 구성돼있으나, 추가적으로 파라미터 유형 필드를 7로 설정하고 상태 쿠키를 정의해야 함



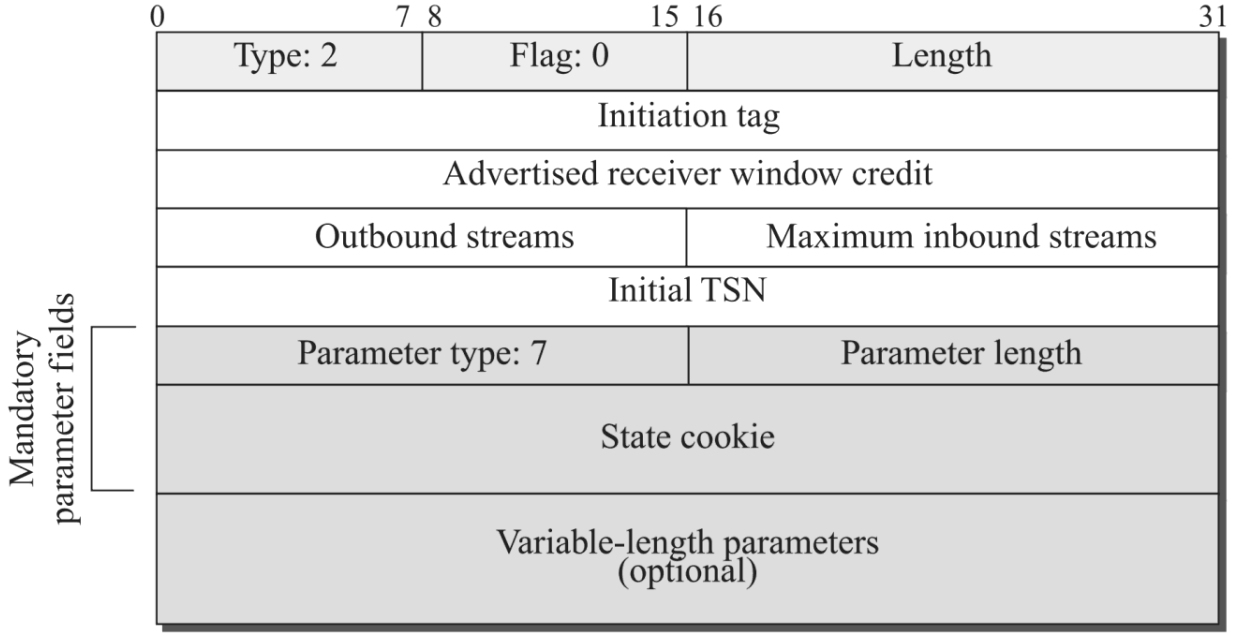
# SCTP

- 패킷 (15/29)
  - 청크 (11/25)
    - INIT ACK
      - 파라미터 유형(Parameter Type)
        - 파라미터의 유형을 정의하며, 7은 상태 쿠키임을 나타냄
      - 파라미터 길이(Parameter Length)
        - 파라미터 유형, 파라미터 길이 필드를 포함한 파라미터 전체 길이를 바이트 단위로 정의함

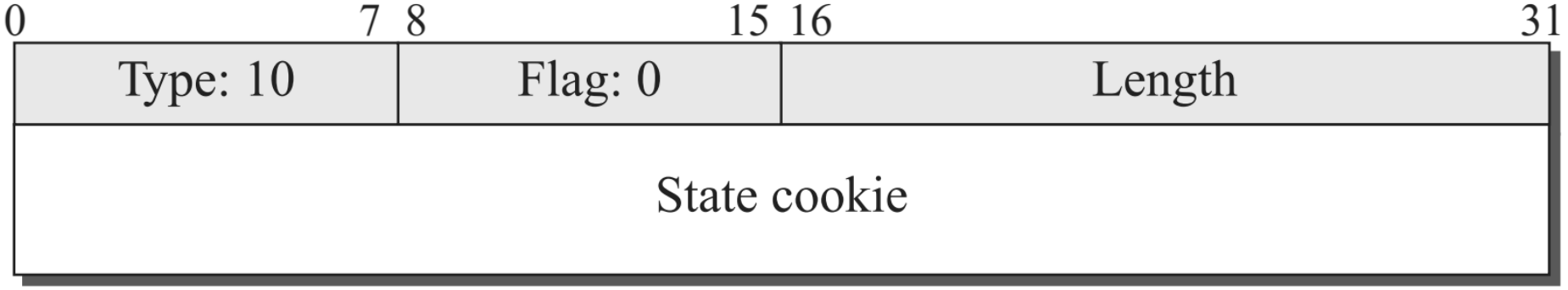


# SCTP

- 패킷 (16/29)
  - 청크 (12/25)
    - INIT ACK
      - 상태 쿠키(State Cookie)
        - 어소시에이션 정보, 쿠키 생성 시각, 쿠키 유효 시간, 어소시에이션 정보에 대한 MAC (Message Authentication Code)를 담아 상태 쿠키를 생성하고 저장함

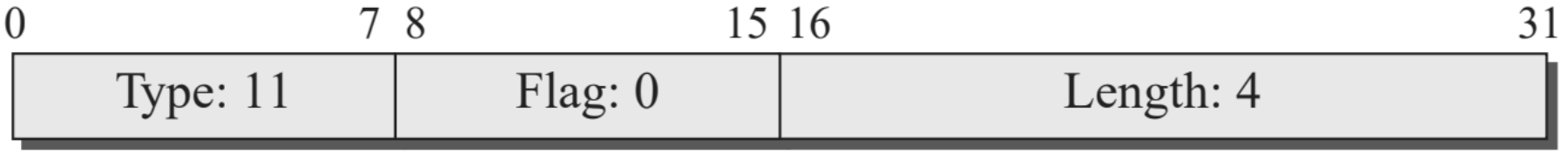


- 패킷 (17/29)
  - 청크 (13/25)
    - COOKIE ECHO
      - 개요
        - 상태 쿠키를 수신받은 호스트가 송신해야하는 청크로, INIT, INIT ACK 청크와 다르게 DATA 청크도 함께 송신할 수 있음
      - 상태 쿠키(State Cookie)
        - INIT ACK를 통해 수신한 상태 쿠키를 변경 없이 전송함



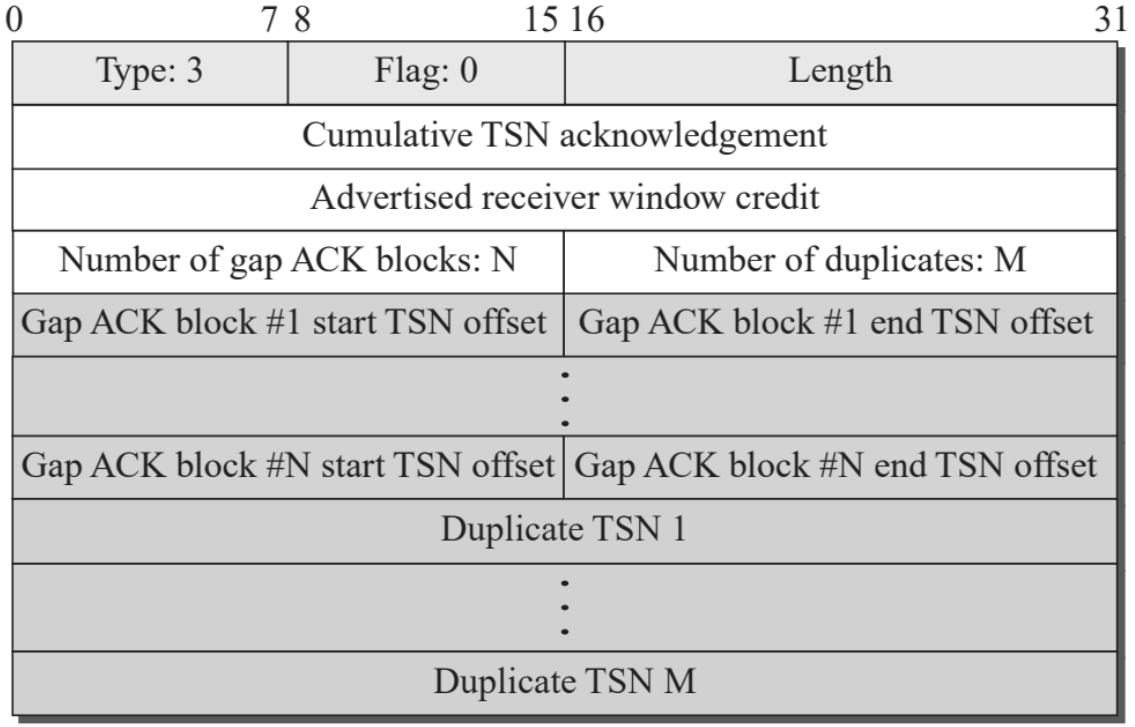
# SCTP

- 패킷 (18/29)
  - 청크 (14/25)
    - COOKIE ACK
      - 개요
        - COOKIE ECHO 청크를 수신받은 호스트가 송신해야하는 청크로, COOKIE ECHO 청크와 마찬가지로 DATA 청크도 함께 송신할 수 있음



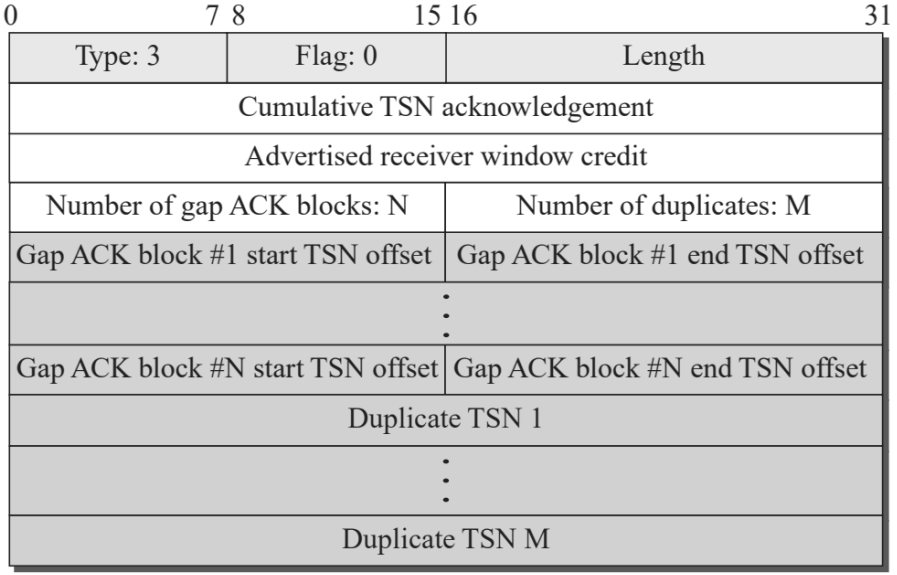
# SCTP

- 패킷 (19/29)
  - 청크 (15/25)
    - SACK
      - 개요
        - 데이터 패킷의 수신을 확인응답함



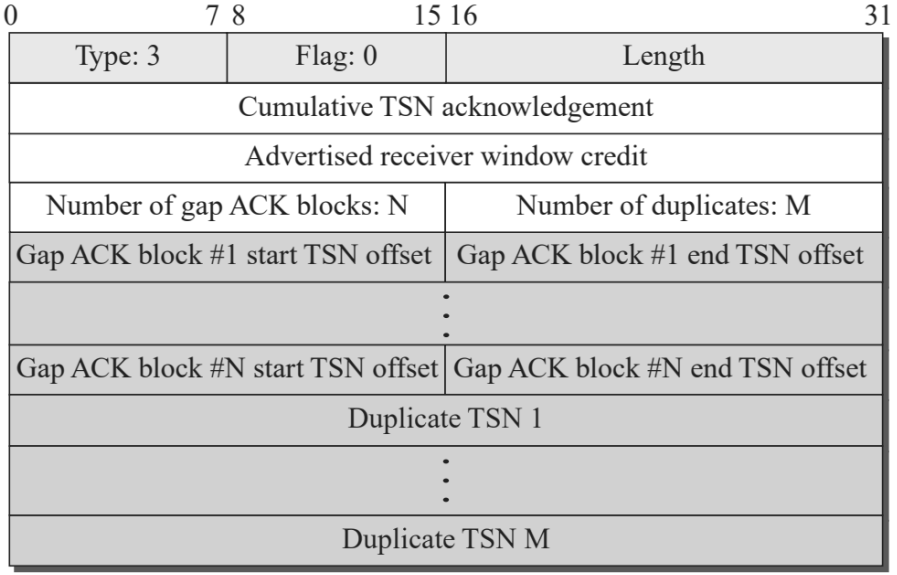
# SCTP

- 패킷 (20/29)
  - 청크 (16/25)
    - SACK
      - 누적 TSN 확인응답(Cumulative TSN Acknowledgement)
        - 32 비트 필드로 마지막으로 수신받은 데이터 청크의 TSN을 정의함
      - 광고된 수신자 윈도우 크레딧(Advertised Receiver Window Credit)
        - 32 비트 필드로 흐름 제어에 사용되는 rwnd 값이 바이트 단위로 정의됨



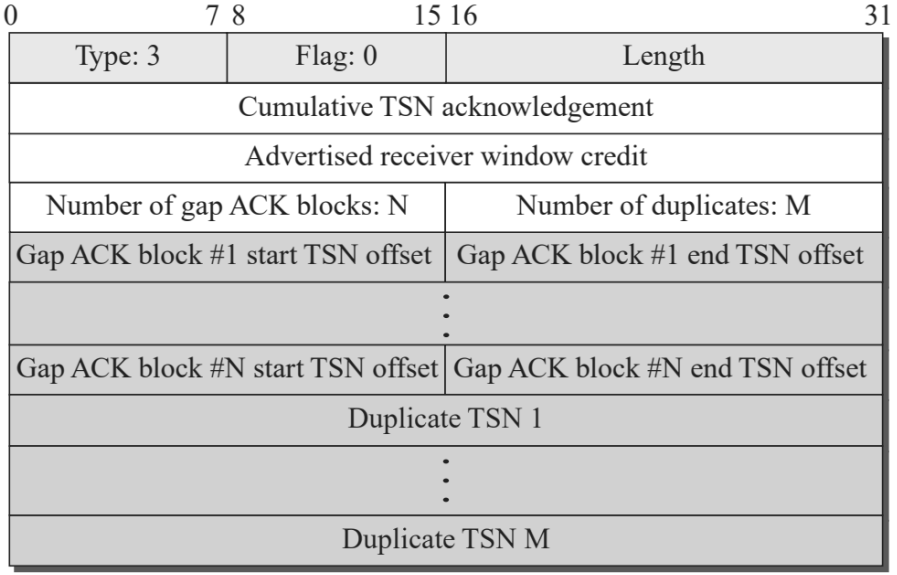
# SCTP

- 패킷 (21/29)
  - 청크 (17/25)
    - SACK
      - 간격 ACK 블록 수(Number of Gap ACK Blocks)
        - 16 비트 필드로 이후 나타낼 간격 ACK 블록의 개수를 정의함
      - 중복 수(Number of Duplicates)
        - 16 비트 필드로 간격 ACK 블록 이후 나타낼 중복 청크 수를 정의함



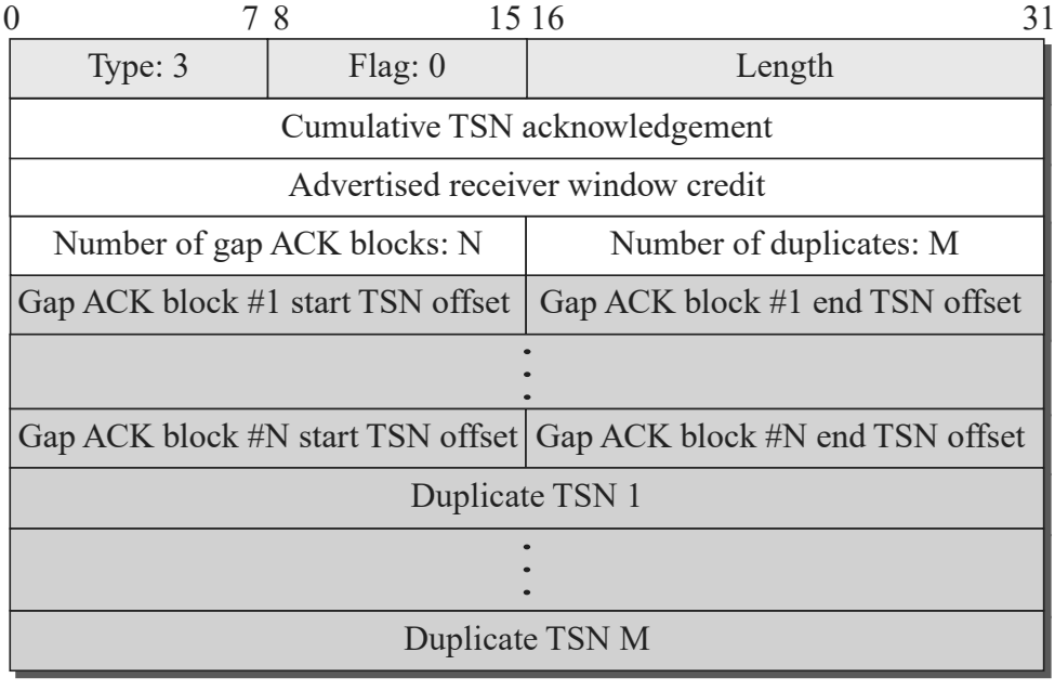
# SCTP

- 패킷 (22/29)
  - 청크 (18/25)
    - SACK
      - 간격 ACK 블록 시작 오프셋(Gap ACK Block Start TSN Offset)
        - 16 비트 필드로 각 간격 ACK 블록에 대해 TSN에 상대적인 시작 TSN을 정의함
      - 간격 ACK 블록 끝 오프셋(Gap ACK Block End TSN Offset)
        - 16 비트 필드로 각 간격 ACK 블록에 대해 TSN에 상대적인 끝 TSN을 정의함



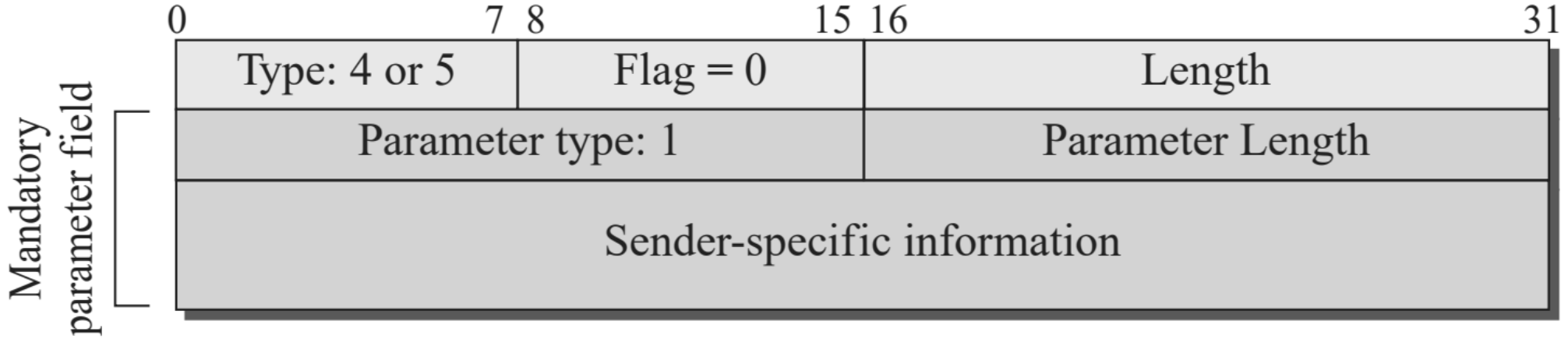
# SCTP

- 패킷 (23/29)
  - 청크 (19/25)
    - SACK
      - 중복된 TSN(Duplicate TSN)
        - 32 비트 필드로 중복된 청크의 TSN를 정의함



# SCTP

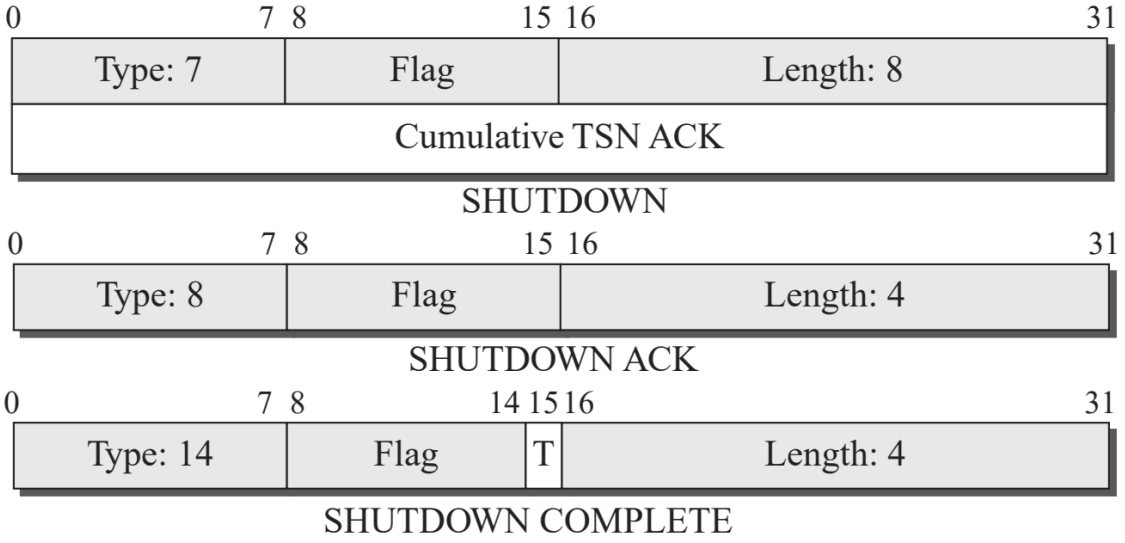
- 패킷 (24/29)
  - 청크 (20/25)
    - HEARTBEAT/HEARTBEAT ACK
      - 개요
        - HEARTBEAT 청크를 전송하고 HEARTBEAT ACK 청크로 답장함으로써 어소시에이션 상태를 점검할 수 있음
      - 송신자 지정 정보(Sender-specific Information)
        - HEARTBEAT 청크 송신자가 임의의 송신자 지정 정보를 설정하여 송신하면 HEARTBEAT ACK 청크 송신자가 해당 송신자 지정 정보를 변경 없이 전송함



# SCTP

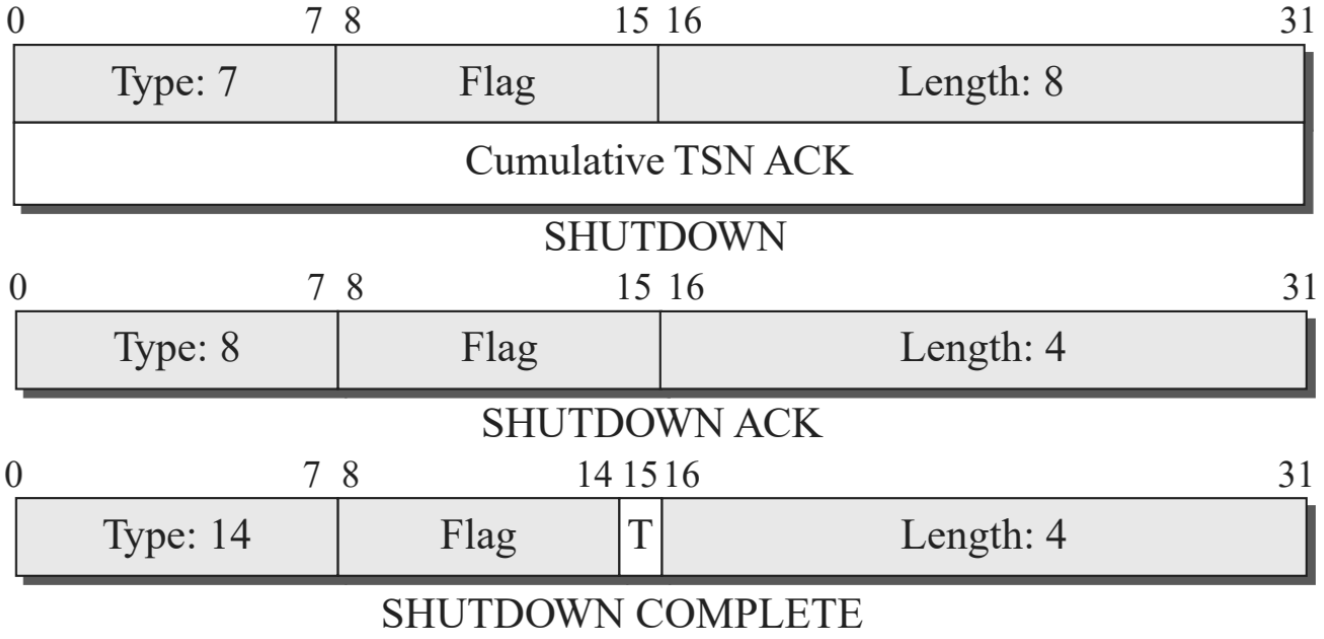
- 패킷 (25/29)
  - 청크 (21/25)
    - SHUTDOWN/SHUTDOWN ACK/SHUTDOWN COMPLETE
      - 개요
        - 어소시에이션 종료에 사용하는 청크로, 어소시에이션 종료를 알리기 위해 SHUTDOWN 청크를 송신하고, SHUTDOWN ACK 청크를 통해 답장하여 종료 요청을 처리했음을 알림
        - TCB 테이블 소지 여부를 알리는 T 플래그와 함께 SHUTDOWN COMPLETE 청크를 송신하여 어소시에이션이 완전히 종료됨을 알릴 수 있음

**TCB (Transmission Control Block):**
어소시에이션 상태와 송수신에 필요한 제어 정보를 저장 및 관리하는 자료구조

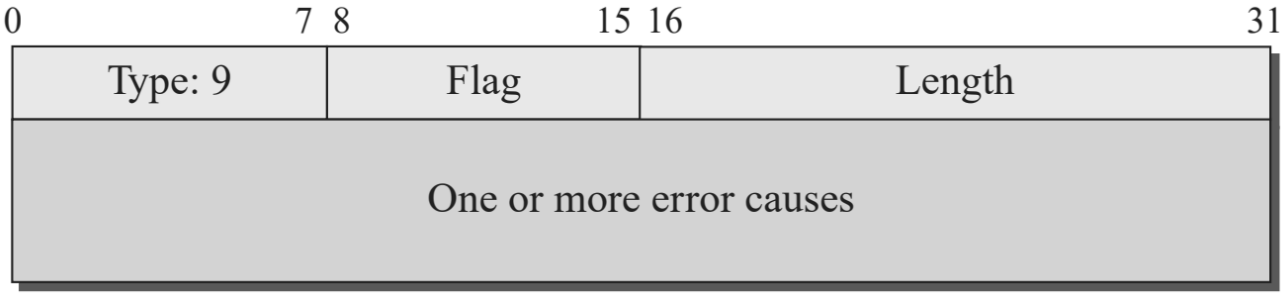


# SCTP

- 패킷 (26/29)
  - 청크 (22/25)
    - SHUTDOWN/SHUTDOWN ACK/SHUTDOWN COMPLETE
      - 누적 TSN 확인응답(Cumulative TSN Acknowledgement)
        - 32 비트 필드로 마지막으로 수신받은 데이터 청크의 TSN을 정의함



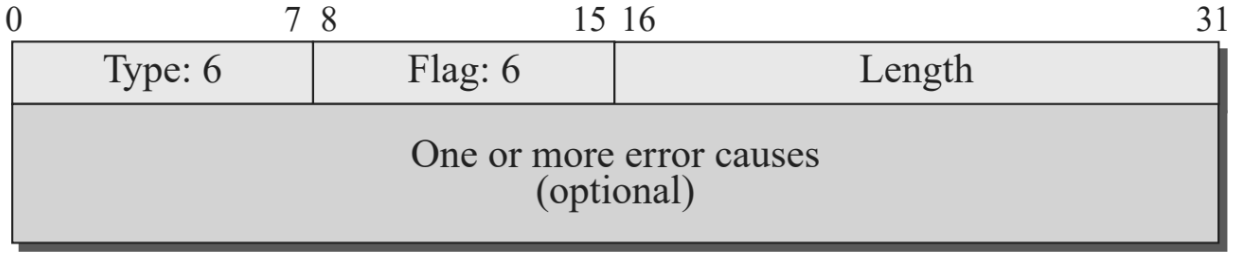
- 패킷 (27/29)
  - 청크 (23/25)
    - ERROR
      - 개요
        - 수신받은 패킷에서 오류가 발생할 경우, ERROR 청크를 통해 오류를 알릴 수 있으며 어소시에이션을 중지하지 않음




| Code | Description |
|---|---|
| 1 | Invalid stream identifier |
| 2 | Missing mandatory parameter |
| 3 | State cookie error |
| 4 | Out of resource |
| 5 | Unresolvable address |
| 6 | Unrecognized chunk type |
| 7 | Invalid mandatory parameters |
| 8 | Unrecognized parameter |
| 9 | No user data |
| 10 | Cookie received while shutting down |

# SCTP

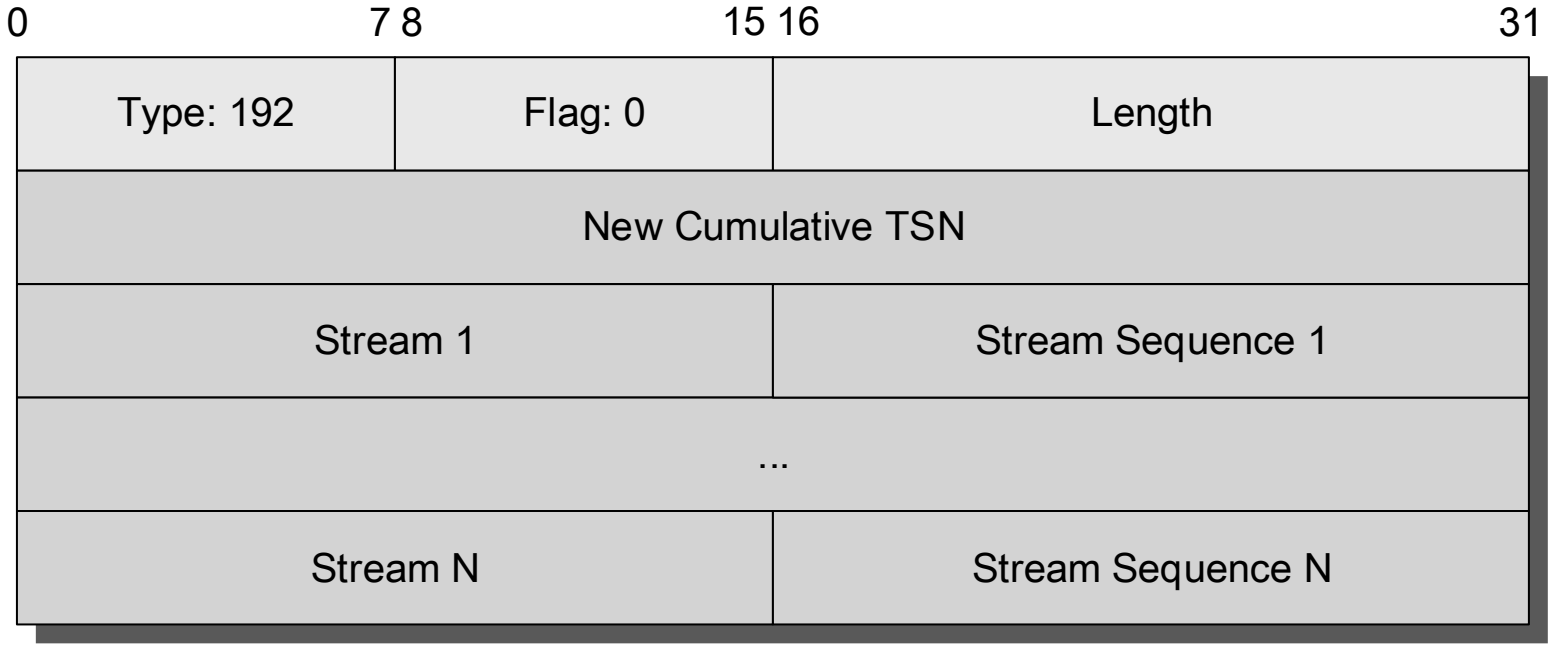
- 패킷 (28/29)
  - 청크 (24/25)
    - ABORT
      - 개요
        - 치명적 오류로 인해 어소시에이션을 강제 종료 시켜야 할 때 전송하며, 오류 유형은 ERROR 청크와 동일하게 사용함




| Code | Description |
|---|---|
| 1 | Invalid stream identifier |
| 2 | Missing mandatory parameter |
| 3 | State cookie error |
| 4 | Out of resource |
| 5 | Unresolvable address |
| 6 | Unrecognized chunk type |
| 7 | Invalid mandatory parameters |
| 8 | Unrecognized parameter |
| 9 | No user data |
| 10 | Cookie received while shutting down |

# SCTP

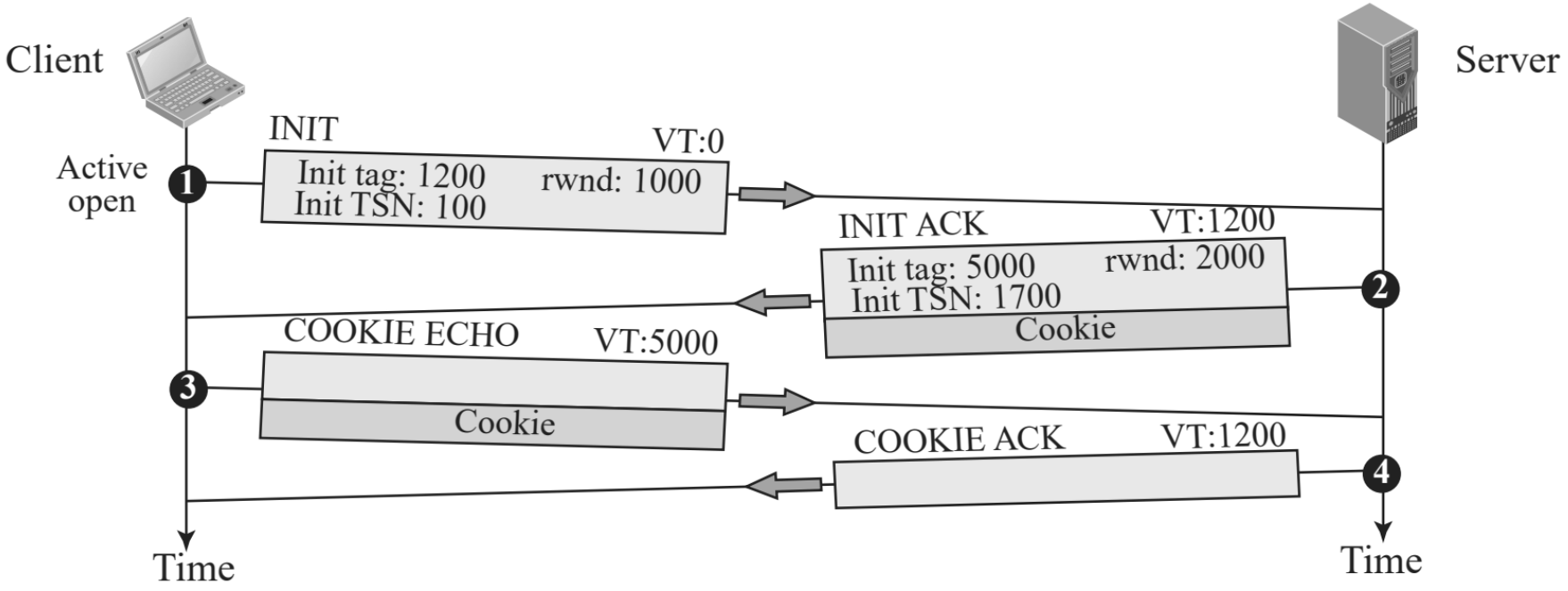
- 패킷 (29/29)
  - 청크 (25/25)
    - FORWARD TSN
      - 개요
        - 누락된 TSN의 재전송을 포기하는 경우, 수신자의 누적 TSN을 변경하기 위해 FORWARD TSN 청크를 전송할 수 있음



# SCTP

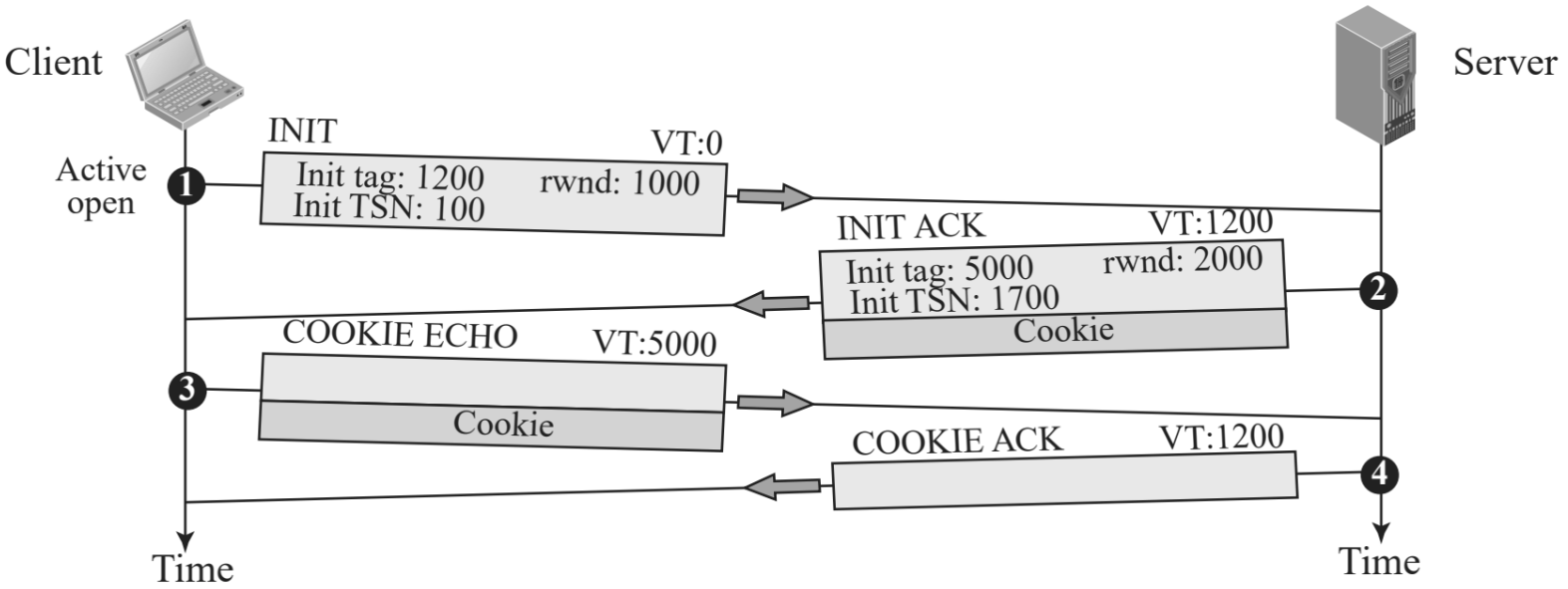
- 어소시에이션(Association) (1/13)
  - 개요
    - SCTP는 연결 지향적 프로토콜(Connection-oriented Protocol)이며, SCTP에서의 연결을 어소시에이션이라고 함
    - 일반적으로 어소시에이션 수립, 데이터 전송, 어소시에이션 종료의 과정을 통해 진행되며, 어소시에이션에 문제가 발생하면 중지를 진행할 수 있음

- 어소시에이션(Association) (2/13)
  - 수립(Establishment) (1/4)
    - 개요
      - 어소시에이션 수립을 위해 4단계 핸드셰이크(Four-way Handshake)를 진행함

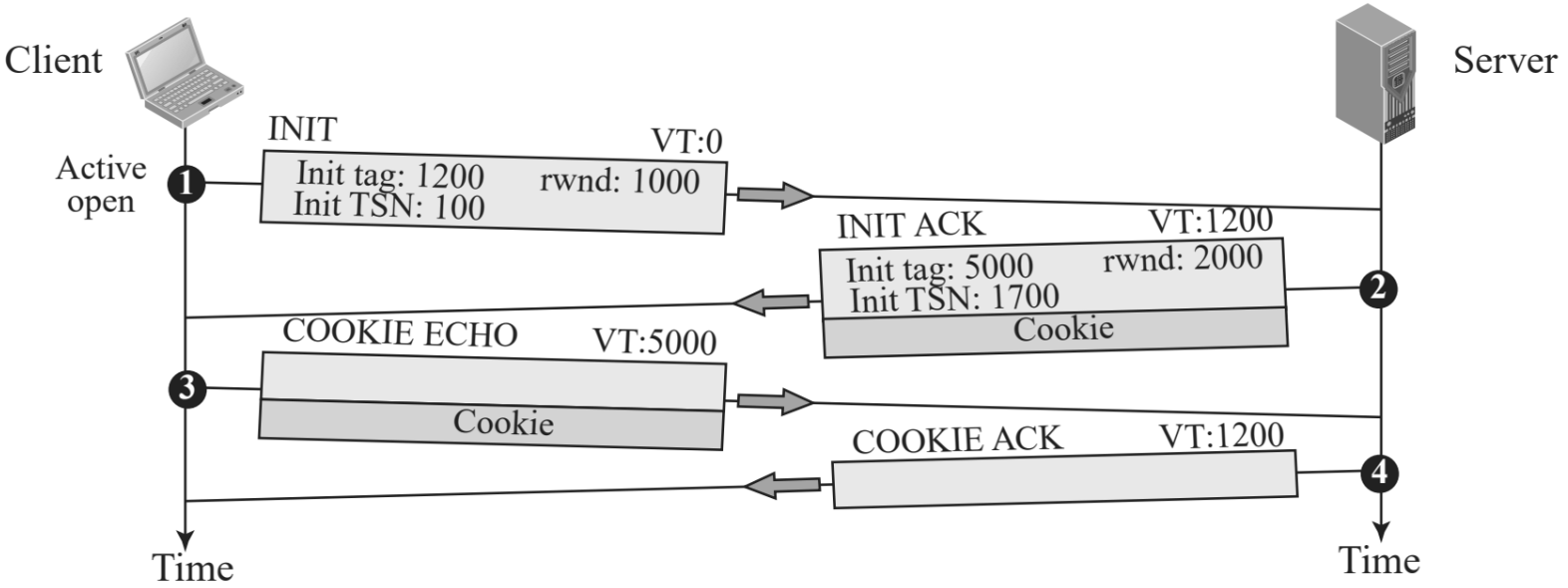


# SCTP

- 어소시에이션(Association) (3/13)
  - 수립(Establishment) (2/4)
    - 과정 (1/2)
      1. INIT 청크에 어소시에이션 설정 정보를 포함하여 전송함으로써 어소시에이션 수립을 시작함
      2. INIT ACK 청크에 어소시에이션 설정 정보 및 쿠키를 포함하여 전송함으로써 어소시에이션 수립을 수락함



- 어소시에이션(Association) (4/13)
  - 수립(Establishment) (3/4)
    - 과정 (2/2)
      3. 상태 쿠키에 대한 응답인 COOKIE ECHO 청크를 전송하며 해당 패킷에는 데이터 청크가 포함될 수 있음
      4. COOKIE ECHO를 정상적으로 받았음을 확인하는 COOKIE ACK 청크를 전송하며 해당 패킷에는 데이터 청크가 포함될 수 있음

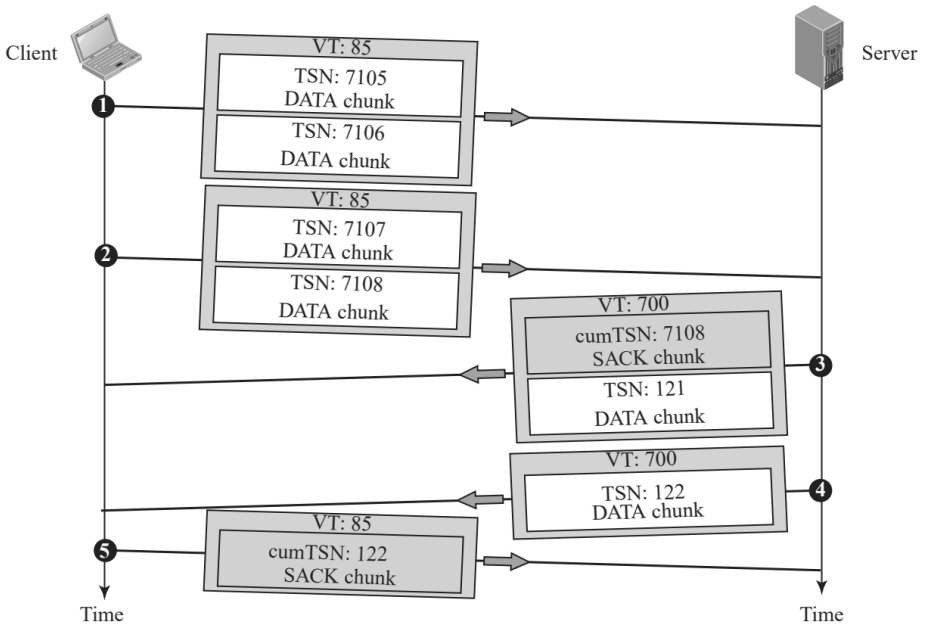


# SCTP

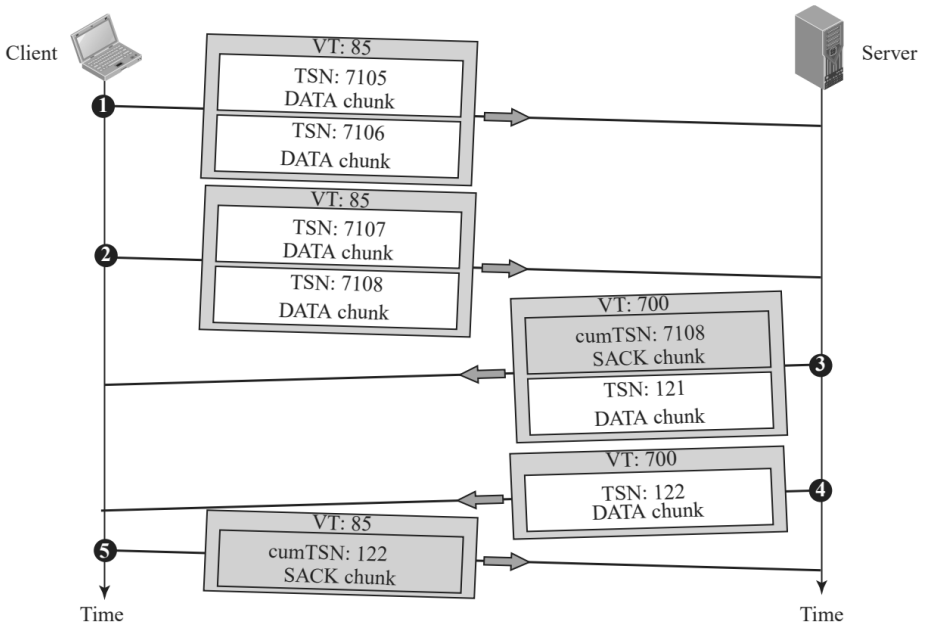
- 어소시에이션(Association) (5/13)
  - 수립(Establishment) (4/4)
    - TCP와의 비교
      - 수립을 위한 패킷 수
        - TCP는 3단계 핸드셰이크를 진행하지만 SCTP는 4단계 핸드셰이크를 진행함
          - SCTP는 단계가 하나 늘지만 세 번째 단계부터 데이터를 전송할 수 있음
      - 확인 태그
        - TCP는 발신지 주소와 포트 번호로 호스트를 구분하지만 SCTP는 확인 태그를 이용하여 호스트를 구분함
          - SCTP는 확인 태그를 통해 잘못된 IP 주소, 포트 번호에서 온 수립 요청이나 이전 어소시에이션에서 오는 패킷을 검증할 수 있음
      - 쿠키
        - TCP는 쿠키가 존재하지 않으나, SCTP는 쿠키를 통해 어소시에이션을 검증함
          - SCTP는 COOKIE ECHO 청크와 COOKIE ACK 청크를 수신 받아야 어소시에이션 수립을 확정하므로 수립 과정에서의 서비스 거부 공격에 향상된 보안을 제공함

- 어소시에이션(Association) (6/13)
  - 데이터 전송 (1/6)
    - 개요
      - SCTP는 메시지 경계를 유지하며 데이터 전송을 진행하며 Piggybacking을 지원함
      - 하나의 메시지로부터 나온 DATA 청크는 모두 같은 TSN을 가지며, DATA 청크만이 TSN을 사용하고 SACK 청크에 의해 확인응답됨

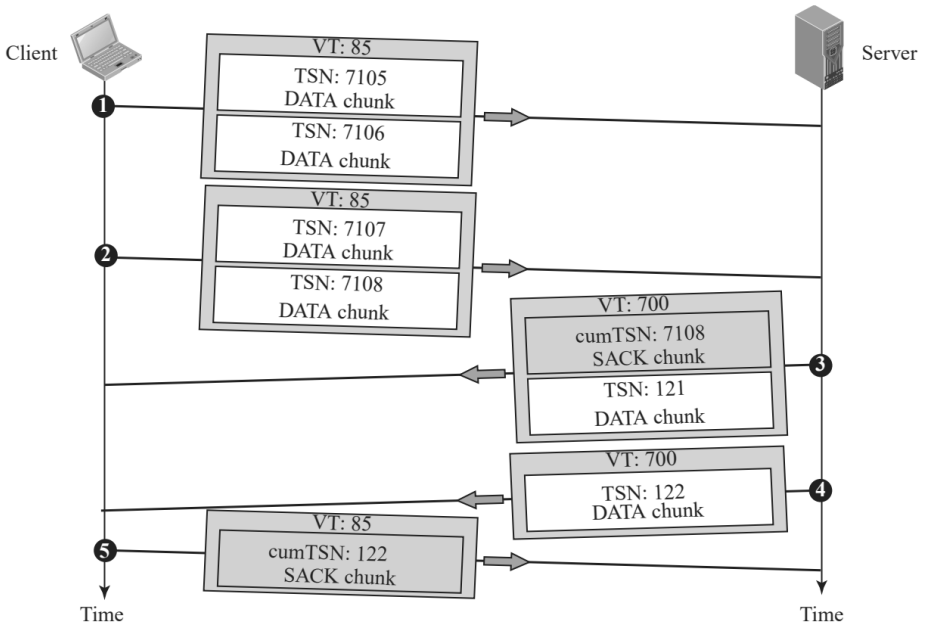
**Piggybacking:**
반대 방향으로 전송할 데이터에 ACK 등의 제어 정보를 함께 포함하여 전송하는 방식



- 어소시에이션(Association) (7/13)
  - 데이터 전송 (2/6)
    - 과정 (1/3)
      1. DATA 청크마다 TSN을 소모하여 패킷을 전송함
      2. SACK 청크를 받기 전 패킷을 전송하는 경우, 마지막에 보낸 DATA 청크 TSN의 다음 번호를 TSN으로 하여 전송함

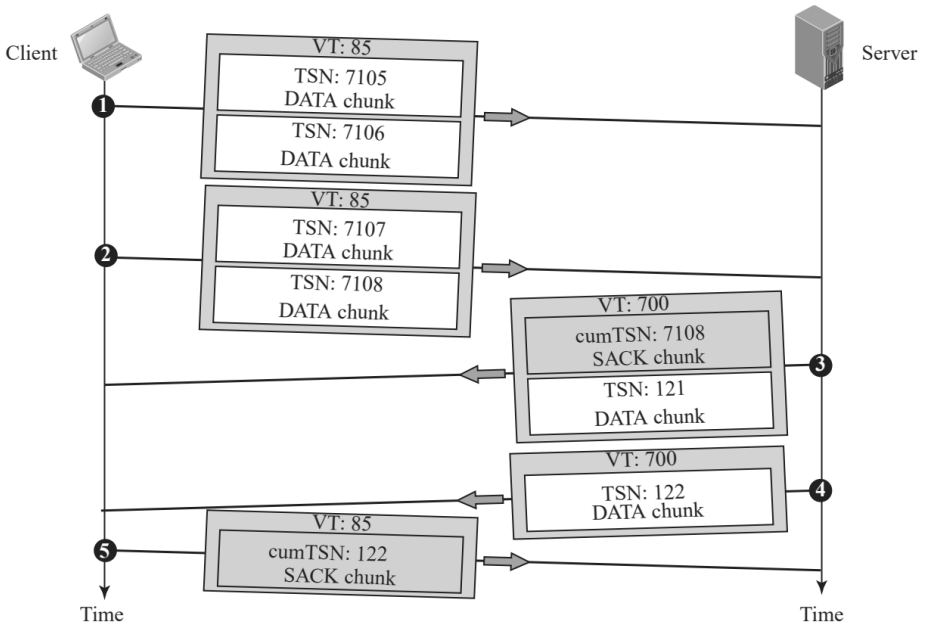


- 어소시에이션(Association) (8/13)
  - 데이터 전송 (3/6)
    - 과정 (2/3)
      3. 마지막으로 수신받은 DATA 청크의 TSN에 대한 SACK 청크와 전송할 DATA 청크를 한 패킷에 담아 전송함
      4. SACK 청크 전송 이후 수신받은 DATA 청크가 없는 경우, 별도의 SACK 청크 없이 DATA 청크만 담아 패킷을 전송함



# SCTP

- 어소시에이션(Association) (9/13)
  - 데이터 전송 (4/6)
    - 과정 (3/3)
      5. 전송할 DATA 청크가 없는 경우, 수신받은 DATA 청크에 대한 SACK 청크만 담아 패킷을 전송함



# SCTP

- 어소시에이션(Association) (10/13)
  - 데이터 전송 (5/6)
    - 멀티호밍
      - 통신을 위해 다중 IP 주소 정의를 허용하나, 해당 주소 중 하나만 주 주소(Primary Address)가 되고 나머지는 대안 주소(Alternative Address)가 됨
      - 송신자가 목적지에 대한 주 주소를 선택할 수 있으며, SACK 청크는 DATA 청크를 수신한 주소로 전송함
    - 다중스트림 전달
      - 하나의 어소시에이션에 대해 하나 이상의 스트림을 설정할 수 있고, 메시지는 각 스트림 중 하나에 속할 수 있음
      - DATA 청크의 U 플래그를 설정하면 해당 스트림 내 DATA 청크를 수신받을 때 SSN 없이 비순서적으로 송수신함

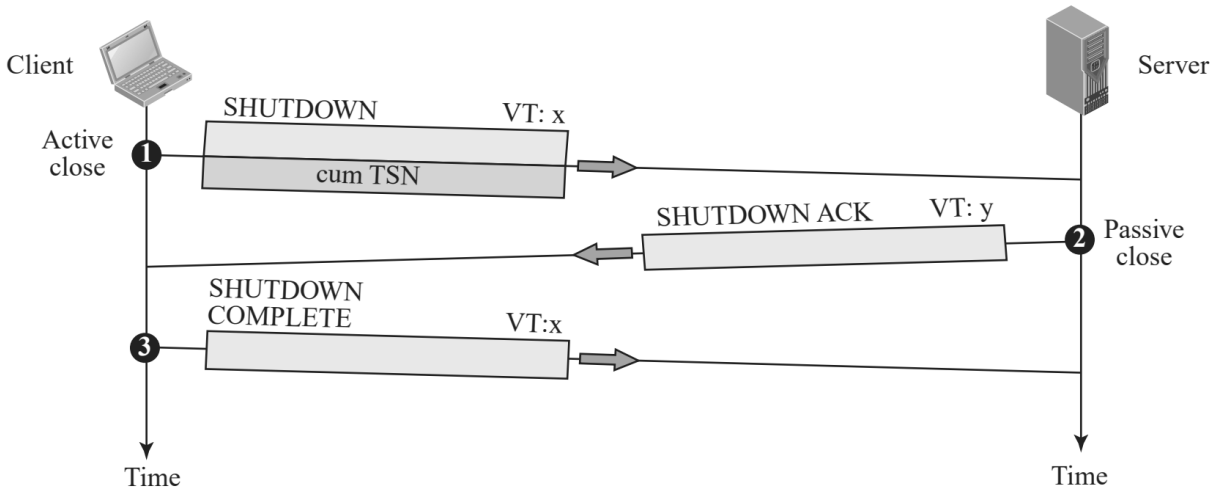
# SCTP

- 어소시에이션(Association) (11/13)
  - 데이터 전송 (6/6)
    - 단편화
      - SCTP 메시지 크기로 인해 MTU가 초과되는 경우, 단편화를 통해 메시지를 여러 개의 단편으로 만들어 전송함
      - 모든 단편은 서로 다른 TSN을 가지는 DATA 청크 헤더를 가지며, SI, SSN, 프로토콜 식별자, U 플래그 여부 등의 설정을 통일함
      - B, E 플래그에 대해, 첫 번째 단편인 경우 10, 중간 단편인 경우 00, 마지막 단편인 경우 01, 단편화되지 않은 경우 11로 설정하여 전송함

MTU (Maximum Transmission Unit):
하나의 네트워크 링크에서 단편화 없이 전송할 수 있는 최대 IP 패킷 크기

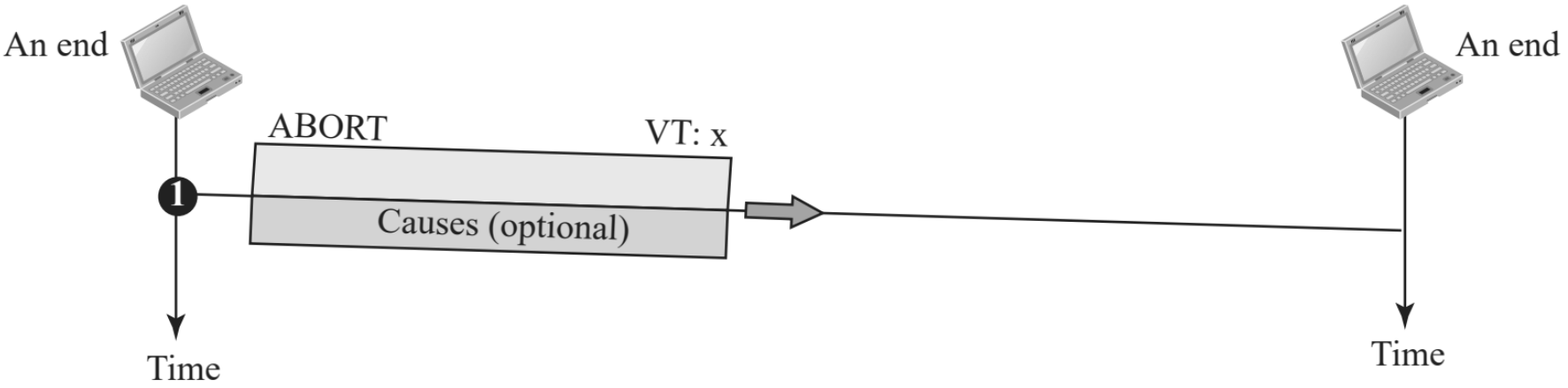
# SCTP

- 어소시에이션(Association) (12/13)
  - 종료
    - 클라이언트나 서버 중 어느 쪽이든 어소시에이션을 종료할 수 있으나 Half-closed를 허용하지 않으며, 어소시에이션 종료 시 상대는 새로운 데이터 전송을 멈춰야 함
    - 종료 요청 수신 큐에 데이터가 남아있는 경우, 해당 데이터가 전부 전송된 후 어소시에이션이 종료됨



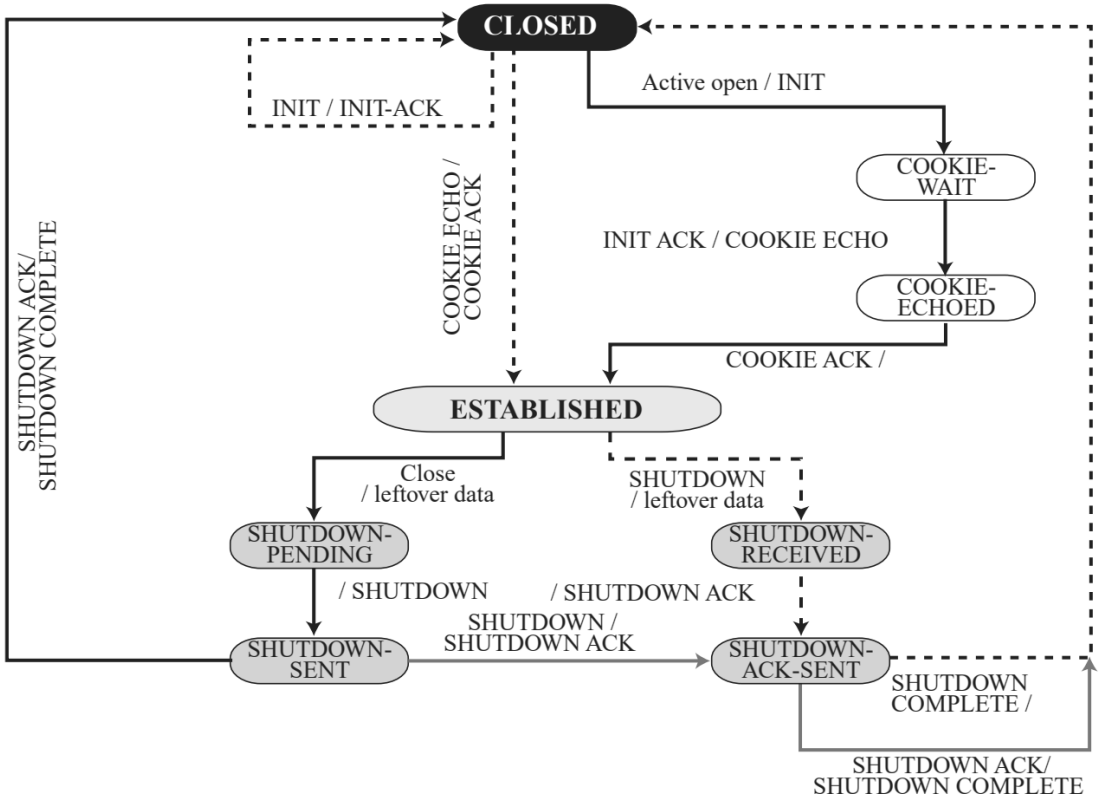
# SCTP

- 어소시에이션(Association) (13/13)
  - 중지
    - 프로세스, INIT 청크 등에 문제가 있거나, 운영체제가 종료되어야 하는 경우, 어소시에이션을 중지할 수 있음
    - 종단 어느 쪽에서든 ABORT 청크를 통해 어소시에이션을 중지할 수 있으며, 추가적인 청크는 불필요함



# SCTP

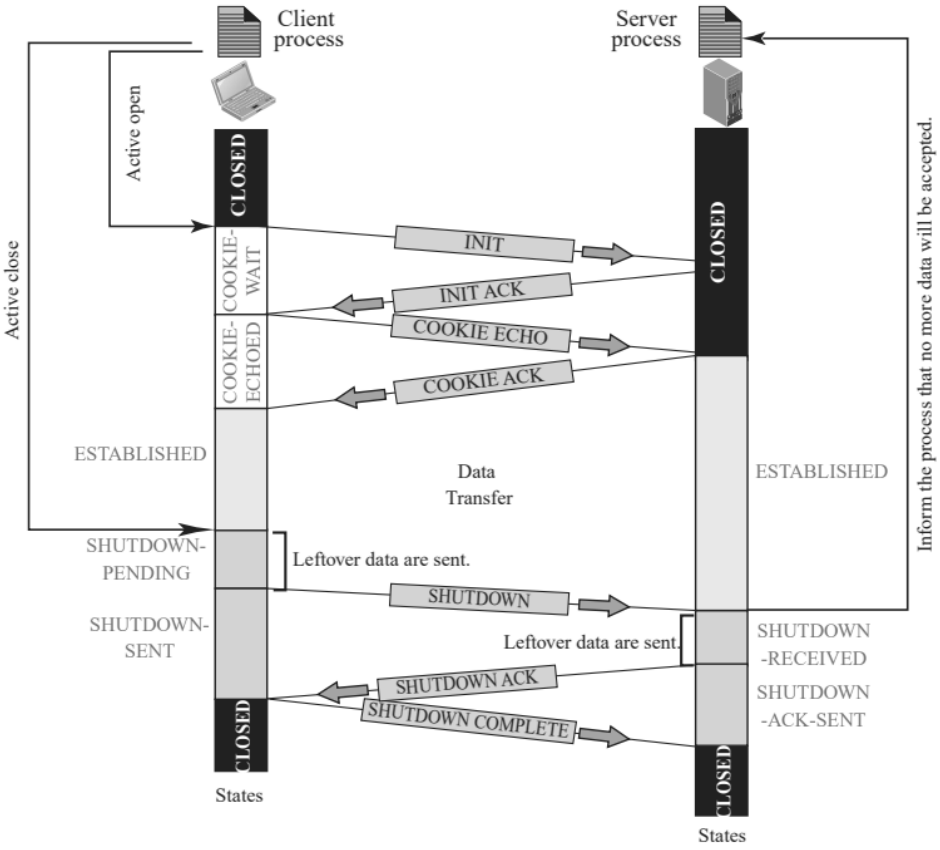
- 상태 천이 다이어그램
  - 유한 상태 기기(FSM, Finite State Machine)을 이용하여 어소시에이션 과정의 상태를 나타내는 다이어그램을 그릴 수 있음




| 상태 | 설명 |
|---|---|
| CLOSED | 연결 없음 |
| COOKIE-WAIT | 상태 쿠키 대기중 |
| COOKIE-ECHOED | COOKIE ACK 대기중 |
| ESTABLISHED | 연결이 수립되었으며 데이터 전송중 |
| SHUTDOWN-PENDING | SHUTDOWN 요청 이후 데이터 전송중 |
| SHUTDOWN-SENT | SHUTDOWN ACK 대기중 |
| SHUTDOWN-RECEIVED | SHUTDOWN 수신 이후 데이터 전송중 |
| SHUTDOWN-ACK-SENT | 종료 완료 대기중 |

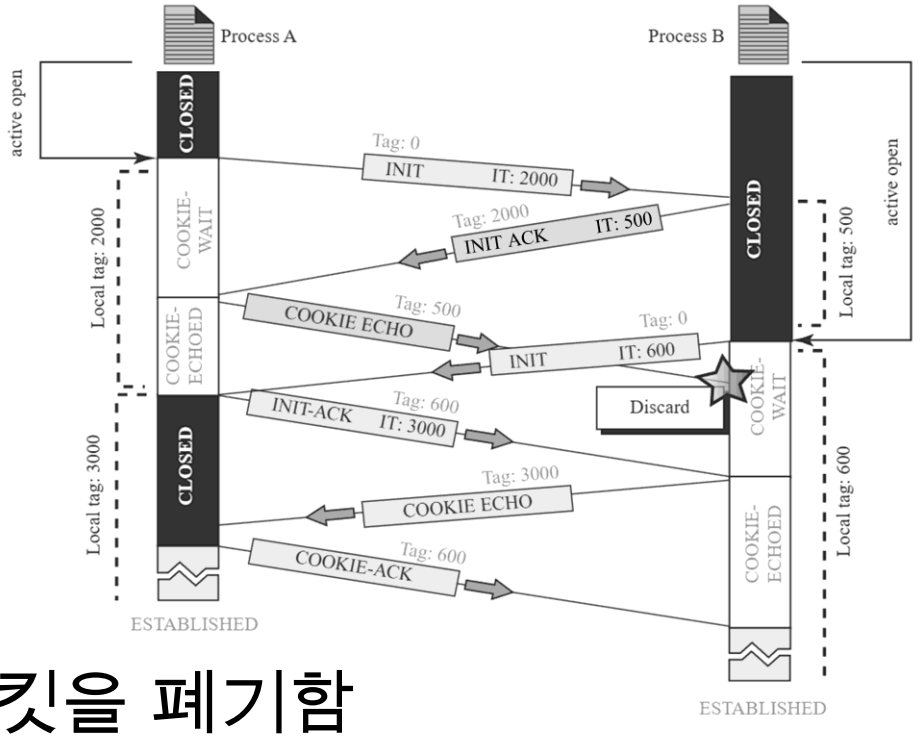
# SCTP

- 통신 과정 (1/3)
  - 공통
    1. 4단계 핸드셰이크를 통해 어소시에이션을 수립함
       - 서버의 경우, 수립 전까지 CLOSED 상태를 유지하나, 클라이언트는 COOKIE-WAIT, COOKIE-ECHOED 상태를 거침
    2. 패킷을 송수신하며 데이터를 전송함
    3. SHUTDOWN 관련 청크를 통해 어소시에이션을 종료함
       - 클라이언트와 서버는 각각 SHUTDOWN-PENDING 상태와 SHUTDOWN-RECEIVED 상태를 통해 남은 데이터를 전송한 후 어소시에이션 종료를 진행함



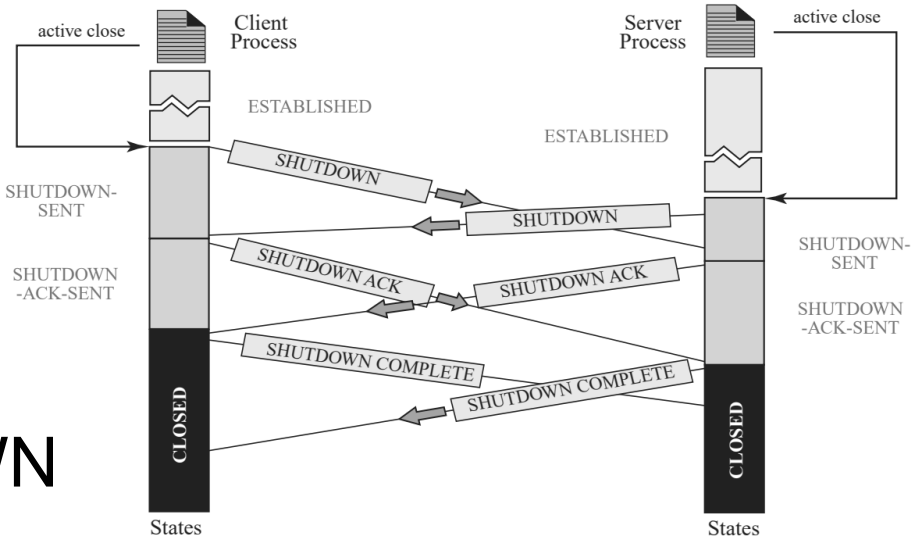
# SCTP

- 통신 과정 (2/3)
  - 동시 개방
    1. 프로세스 A가 수립 요청을 위해 INIT 청크를 전송함
    2. 프로세스 B가 COOKIE ECHO 수신 전 수립 요청을 위해 INIT 청크를 전송함
    3. 로컬 태그와 다른 패킷을 수신한 프로세스는 해당 패킷을 폐기함
    4. COOKIE ACK 청크를 받지 못한 첫 번째 어소시에이션은 중지되고, 새롭게 설정된 두 번째 어소시에이션이 수립됨



# SCTP

- 통신 과정 (3/3)
  - 동시 종료
    1. 한 쪽 프로세스가 종료를 위해 SHUTDOWN 청크를 전송함
    2. 상대 프로세스가 SHUT-DOWN 수신 전 SHUTDOWN 청크를 전송함



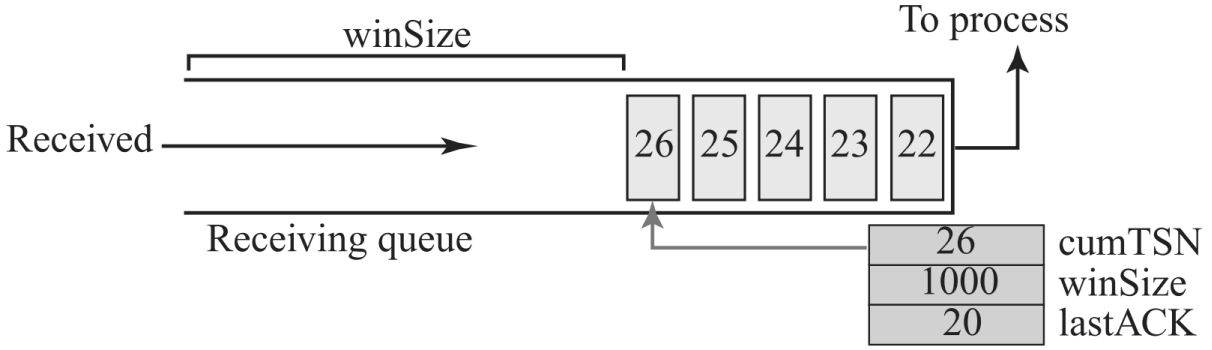

    3. 두 프로세스 모두 SHUTDOWN 수신 이후 SHUTDOWN ACK 청크 전송 후 SHUTDOWN-ACK-SENT 상태가 되고 SHUTDOWN ACK를 기다림
    4. SHUTDOWN ACK 수신 시, SHUTDOWN COMPLETE 청크를 보내고 어소시에이션을 종료함

# SCTP

- 흐름 제어 (1/4)
  - 개요
    - SCTP는 흐름 제어를 위해 여전히 바이트 지향적 윈도우를 사용하지만, rwnd는 바이트 단위, TSN, SACK은 청크 단위를 사용함
    - 수신자는 흐름 제어를 위해 하나의 버퍼, 수신한 마지막 TSN인 cumTSN, 가능한 버퍼 크기 winSize, 누적 확인응답 lastACK 변수를 이용함
    - 송신자는 흐름 제어를 위해 하나의 버퍼, 전송할 다음 TSN인 curTSN, 바이트 단위 수신 윈도우 크기 rwnd, 확인응답 되지 않은 전송중인 바이트 수 inTransit을 이용함

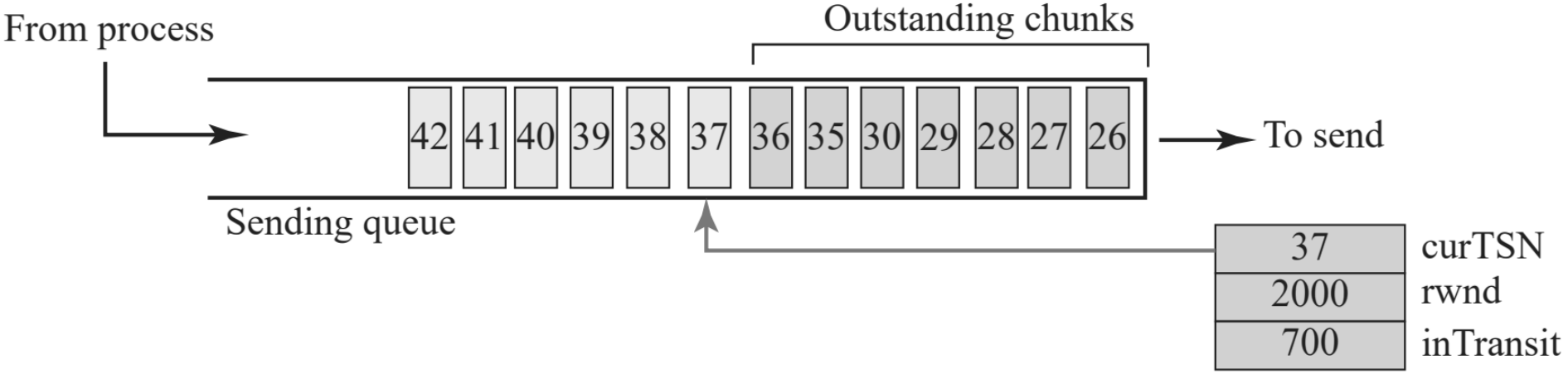
# SCTP

- 흐름 제어 (2/4)
  - 수신측
    - DATA 청크를 수신하면 버퍼 끝에 저장하며, winSize에서 청크 크기를 빼고 cumTSN에 DATA 청크의 마지막 TSN을 저장함
    - 프로세스가 청크를 읽으면 해당 청크를 버퍼에서 제거하고, 제거된 청크 크기를 winSize에 추가함
    - SACK 청크를 보낼 때 lastACK 값을 검사해서 cumTSN보다 작으면 누적 확인응답을 cumTSN로 하고 광고된 수신자 윈도우 크레딧을 winSize로 하여 전송 후 lastACK를 최신화함



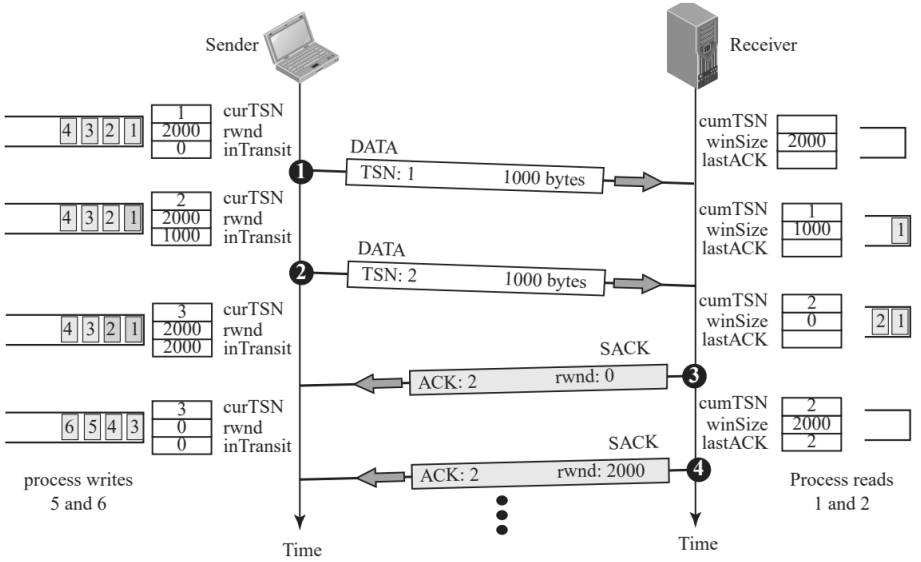
# SCTP

- 흐름 제어 (3/4)
  - 송신측
    - curTSN이 가리키는 청크의 크기가 (rwnd – inTransit) 값과 같거나 작은 경우, 해당 청크를 전송하고 curTSN을 하나 증가시킨 후 해당 청크 크기만큼 inTransit을 증가시킴
    - SACK 수신 시, SACK의 누적 확인응답과 같거나 작은 TSN을 가진 청크는 버퍼에서 제거하고 inTransit을 제거한 청크 크기의 총합만큼 감소시킨 후 rwnd를 최신화함



# SCTP

- 흐름 제어 (4/4)
  - 예시

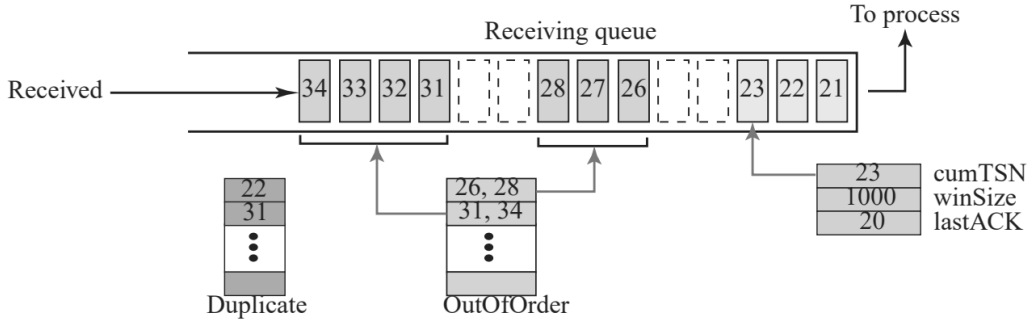



1. 송신자가 1000 바이트 청크를 전송하여 수신자 버퍼에 쌓이고 winSize가 감소함
2. 송신자가 1000 바이트 청크를 추가로 전송하여 수신자 버퍼에 쌓이고 winSize가 0이 됨
3. 수신자가 수신받은 두 패킷에 대한 SACK 청크를 전송하며 rwnd가 0이 되었음을 알림으로써 송신자는 더 이상 청크를 전송할 수 없게 됨
4. 수신자 프로세스가 청크를 읽어들이고 다시 SACK 청크를 전송하여 rwnd가 2000으로 증가했음을 알림

# SCTP

- 오류 제어 (1/6)
  - 개요
    - SCTP는 송신자에게 수신자 버퍼 상태를 알릴 수 있는 SACK 청크와 타이머 등을 이용하여 오류 제어를 구현함
    - 수신자는 오류 제어를 위해 간격 ACK 블록과 중복 TSN을 SACK 청크에 포함시켜 전송함
    - 송신자는 오류 제어를 위해 전송 큐, 재전송 큐, 전송할 다음 TSN인 curTSN, 바이트 단위 수신 윈도우 크기 rwnd, 확인응답 되지 않은 전송중인 바이트 수 inTransit을 이용함

- 오류 제어 (2/6)
  - 수신측
    - 순서에 어긋난 패킷을 수신한 경우, 순서에 맞춰 프로세스로 보내기 위해 계속 보관함
    - 중복된 패킷은 TSN만 기록하고 폐기한 후 간격 ACK 블록 정보와 함께 SACK 청크에 담아 전송함

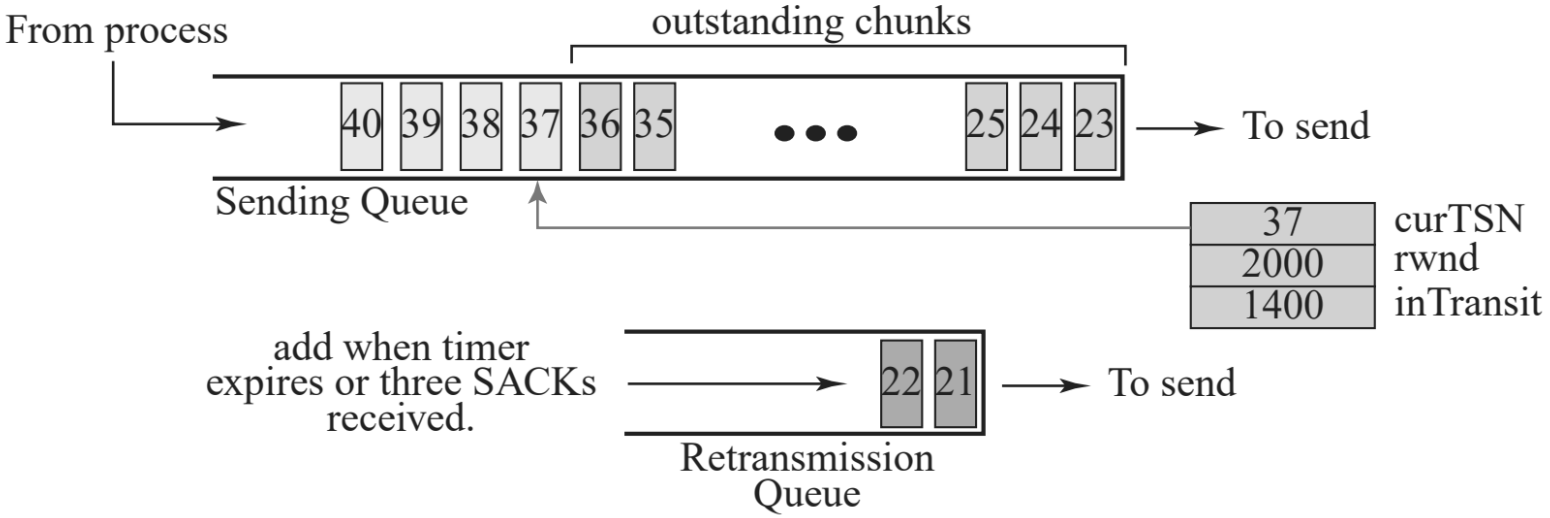



SACK chunk

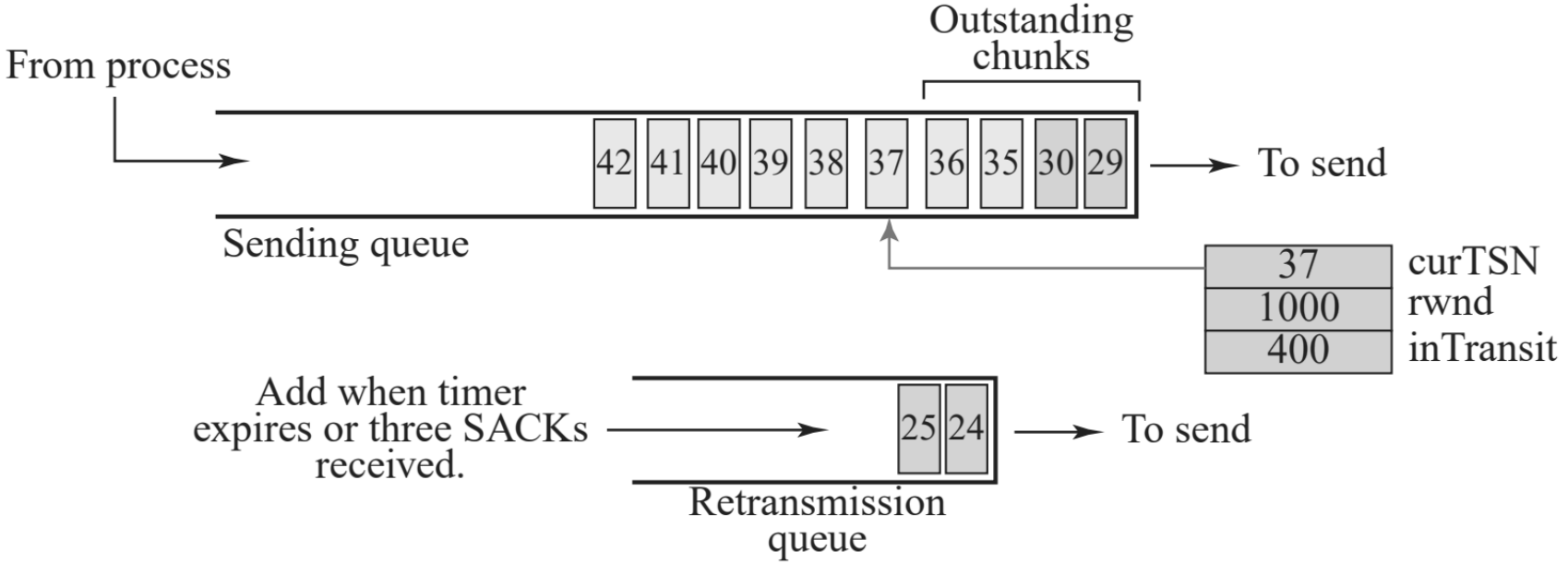
| | | |
|---|---|---|
| Type: 3 | Flag: 0 | Length: 32 |
| Cumulative TSN: 23 | | |
| Advertised receiver window credit: 1000 | | |
| Number of gap ACK blocks: 2 | Number of duplicates: 2 | |
| Gap ACK block #1 start: 3 | Gap ACK block #1 end: 5 | |
| Gap ACK block #2 start: 8 | Gap ACK block #2 end: 11 | |
| Duplicate TSN: 22 | | |
| Duplicate TSN: 31 | | |

Numbers are relative to cumTSN

- 오류 제어 (3/6)
  - 송신측 (1/2)
    - 전송 큐와 함께 재전송 큐를 가지고 있으며, 재전송 타이머가 만료되거나 4개의 중복된 SACK를 수신하면 패킷 내 청크를 재전송 큐로 전달함
    - 재전송 큐 내의 청크들은 손실된 것으로 간주하고 이후 청크 전송에서 우선적으로 전송됨



- 오류 제어 (4/6)
  - 송신측 (2/2)
    - SACK을 수신한 경우, SACK의 cumTSN과 같거나 작은 TSN을 가지는 청크와 중복 청크는 모든 큐에서 제거함
    - 간격 ACK 블록에 포함된 TSN를 가진 청크의 경우, 모든 큐에서 제거하나, 구현에 따라 cumTSN과 같거나 작아지기 전까지 유지될 수 있음



# SCTP

- 오류 제어 (5/6)
  - DATA 청크 송신 규칙
    - 전송 큐보다 재전송 큐 내 청크를 우선 전송함
    - 송신 시 DATA 청크 크기가 (rwnd – inTransit)보다 크거나 패킷 크기가 MTU를 초과하지 않아야 함
    - 재전송 청크와 새 전송 청크를 한 패킷에 담을 수 있음
    - 현재 전송중이며 inTransit 계산에 포함되는 청크는 SACK 청크를 수신하거나 손실되기 전까지 재전송할 수 없음
    - 재전송 타이머가 만료되거나 네 번의 연속적인 SACK에서 손실이 보고되면 청크가 손실된 것으로 판단함

# SCTP

- 오류 제어 (6/6)
  - SACK 청크 생성 규칙
    - 상대 호스트에 보낼 DATA 청크가 있는 경우, SACK 청크를 함께 전송함
    - 수신받은 DATA 청크에 대해서 특정 시간(주로 200ms) 안에는 SACK 청크를 전송해야 함
    - 수신하는 DATA 청크 2개당 최소 하나 이상의 SACK 청크를 전송해야 함
    - 순서에 어긋난 청크를 수신한 경우, 즉시 SACK 청크를 전송해야 함
    - 중복된 청크를 수신한 경우, 즉시 SACK 청크를 전송해야 함

# SCTP

- 혼잡 제어 (1/2)
  - 개요
    - 혼잡 제어를 위해 TCP와 동일하게 느린 시작(Slow Start), 혼잡 회피(Congestion Avoidance), 혼잡 검출(Congestion Detection), 빠른 복구(Fast Recovery)를 사용함
  - 멀티호밍
    - SCTP는 멀티호밍으로 인해 어소시에이션 내 한 호스트에게 여러 IP 주소를 정의할 수 있으므로, 각 목적지 IP 주소별로 별도의 cwnd를 설정하여 사용함

# SCTP

- 혼잡 제어 (2/2)
  - 명확한 혼잡 통지
    - 수신자가 네트워크에서 데이터 전송 중 경험한 혼잡을 송신자에게 명확하게 알리기 위해 INIT/INIT ACK 청크를 통해 ECN의 사용을 선택할 수 있음
    - CWR (Congestion Window Reduce) 청크 수신 전까지 ECNE (Explicit Congestion Notification Echo) 청크를 전송하는 것으로 정의돼있으나, 아직 표준화되지 않음

# 목 차

- SCTP
- 응용계층 개요

# 응용계층 개요

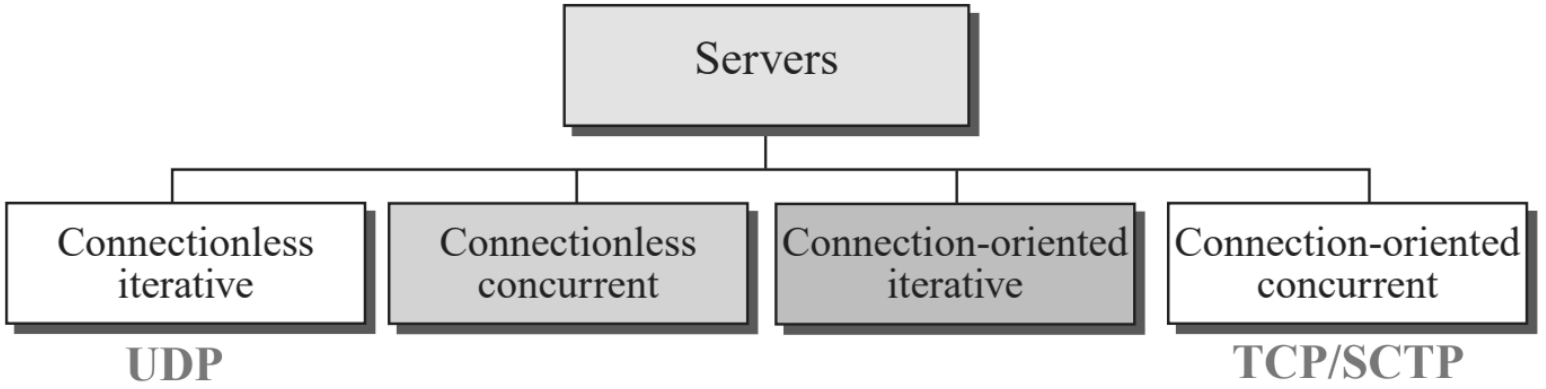
- 클라이언트-서버 (1/27)
  - 개요
    - 로컬측의 사용자는 원격에 있는 컴퓨터로부터 서비스 받기를 원하며, 이를 구현하기 위한 방법으로 클라이언트-서버 구조를 고려할 수 있음
    - 서비스를 요청하는 작업을 수행하는 클라이언트 프로그램과 서비스를 제공하는 작업을 수행하는 서버 프로그램으로 분리할 수 있음
    - 필요할 때만 동작해야 하는 클라이언트와 항상 대기해야 하는 서버는 다-대-일 관계를 가짐

# 응용계층 개요

- 클라이언트-서버 (2/27)
  - 클라이언트
    - 로컬 컴퓨터에서 동작하는 프로그램이며, 필요한 서비스를 서버에게 요청함
    - 서버의 IP 주소와 Well-known 포트를 이용하여 통신 채널을 개설하고, 해당 채널을 통해 서비스 요청 및 응답 수신을 진행함
    - 순차적으로 혹은 동시에 수행될 수 있으나, 순차적으로 수행될 경우, 다른 클라이언트를 수행하기 위해서는 기존의 클라이언트가 종료되어야 함

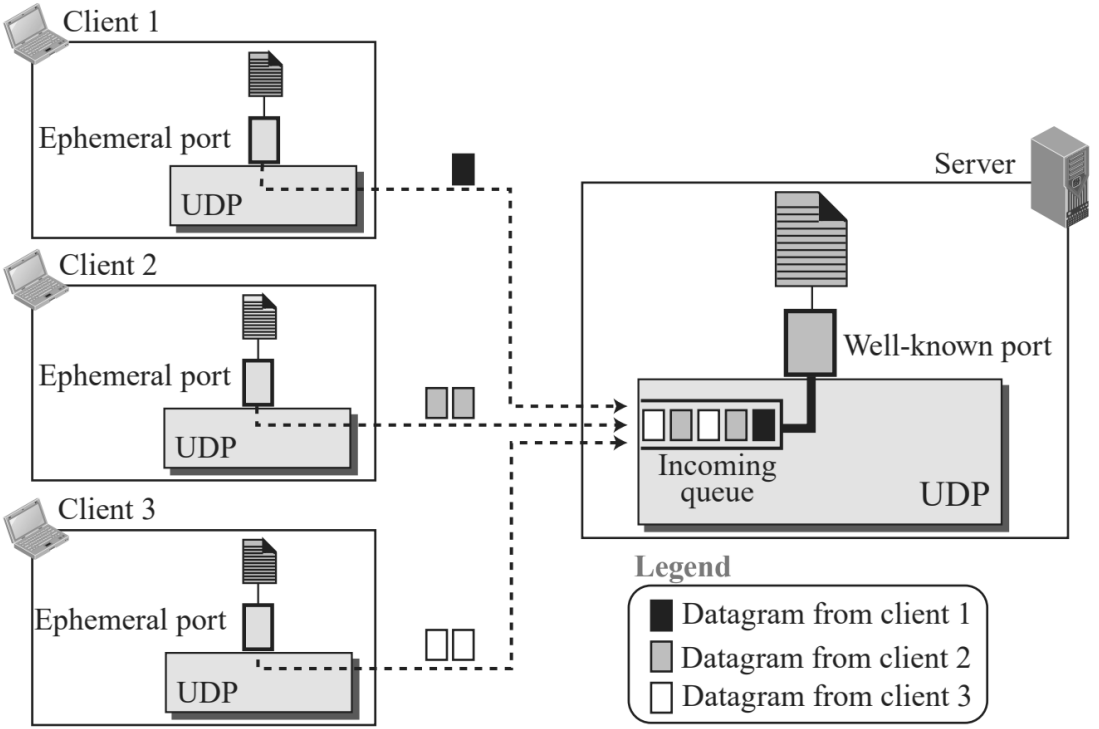
# 응용계층 개요

- 클라이언트-서버 (3/27)
  - 서버 (1/3)
    - 원격으로 동작하는 프로그램이며, 클라이언트 요청에 따라 순차적으로 혹은 동시에 응답하여 서비스를 제공함
    - 프로그램 시작 시, 문제가 생기기 전까지 클라이언의 요청을 대기하며, 요청받기 전까지는 서비스를 제공하지 않음
    - 전송계층 프로토콜과 서비스 방식에 따라 비연결형 순차 서버, 비연결형 동시 서버, 연결 중심 순차 서버, 연결 중심 동시 서버로 구분할 수 있음



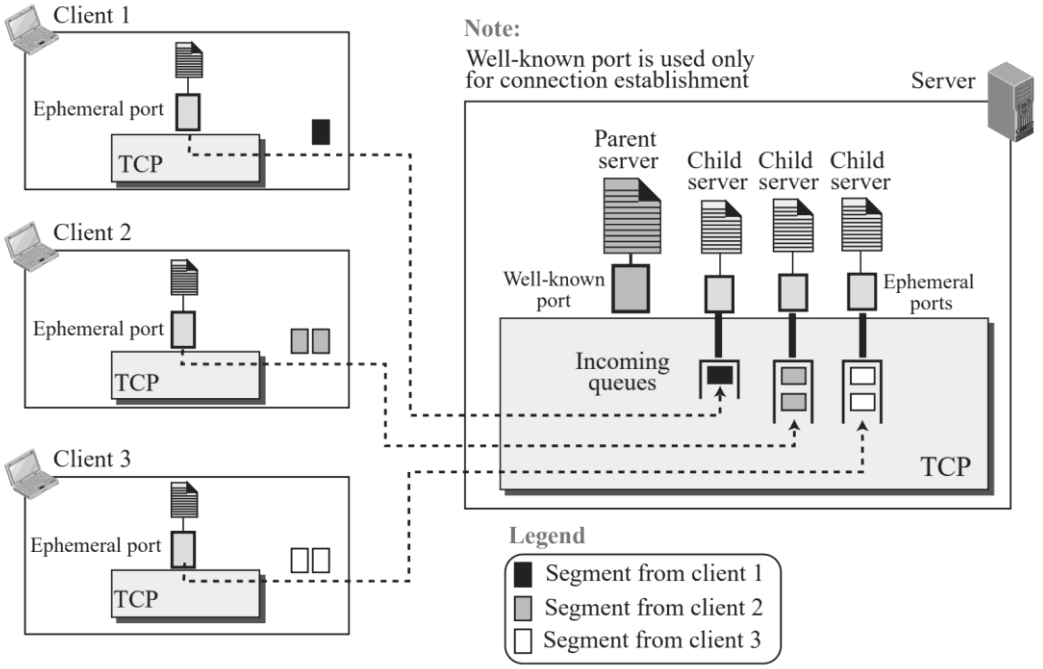
# 응용계층 개요

- 클라이언트-서버 (4/27)
  - 서버 (2/3)
    - UDP를 사용하는 서버는 주로 순차적으로 요청을 수행하며, 하나의 Well-known 포트만을 사용하여 도착하는 모든 데이터그램을 순차적으로 처리함



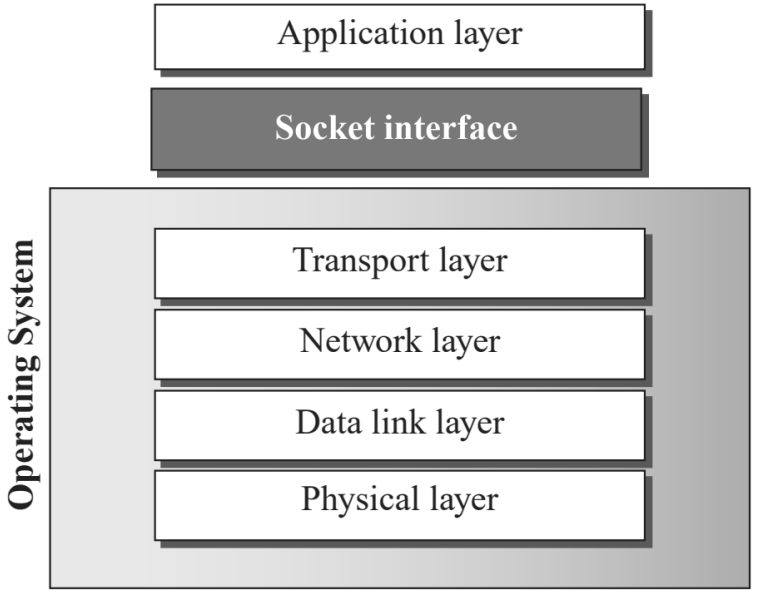
# 응용계층 개요

- 클라이언트-서버 (5/27)
  - 서버 (3/3)
    - TCP/SCTP를 사용하는 서버는 주로 동시에 요청을 수행하며, 하나의 Well-known 포트로 연결을 수립한 후, 생성한 자식 프로세스에 임시 포트를 할당하여 요청을 동시에 처리함



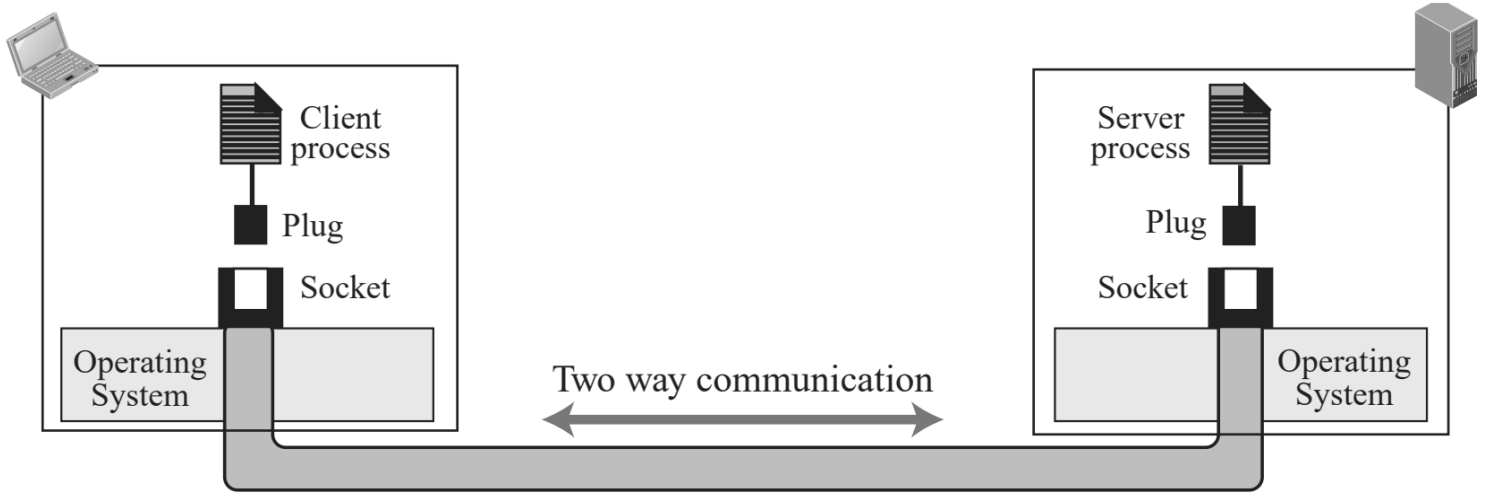
# 응용계층 개요

- 클라이언트-서버 (6/27)
  - 소켓 인터페이스 (1/14)
    - 개요
      - 두 개체 사이의 상호작용을 위해 설계된 명령들의 집합을 인터페이스(Interface)라고 함
      - 소켓 인터페이스(Socket Interface)는 응용계층과 전송계층 사이의 상호작용을 위한 인터페이스로써 활용됨



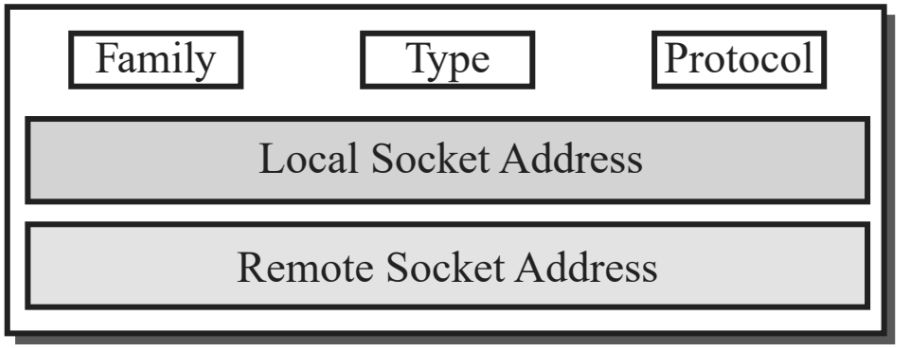
# 응용계층 개요

- 클라이언트-서버 (7/27)
  - 소켓 인터페이스 (2/14)
    - 소켓
      - 응용 프로그램은 통신 채널을 사용하기 위해 운영체제에 소켓 생성 요청을 해야하며, 데이터를 송수신하기 위해 소켓에 플러그를 꽂을 수 있음
      - 데이터 통신이 시작되기 위해서는 통신하는 종단에 각각 소켓이 필요함



# 응용계층 개요

- 클라이언트-서버 (8/27)
  - 소켓 인터페이스 (3/14)
    - 소켓 – 데이터 구조 (1/3)
      - 개요
        - 소켓은 데이터 구조체 형태로 구현될 수 있음
        - 소켓 관련 구조체는 헤더 파일에 이미 정의되어 있으며, 프로그래머가 구조를 직접 정의하지 않고 필요한 헤더를 포함한 뒤 이미 정의된 구조체와 그 필드를 사용해야 함

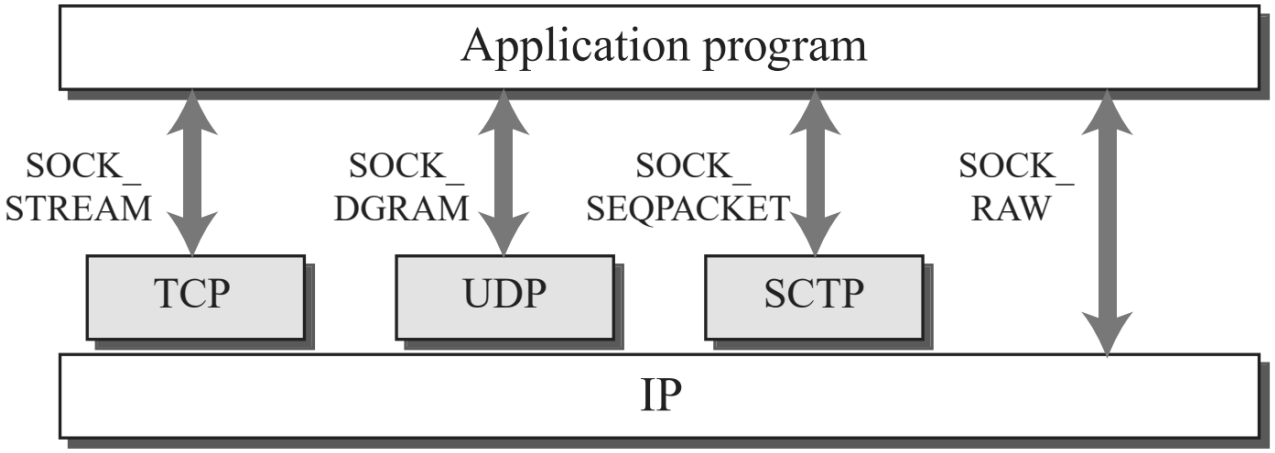


Fields

```
struct  socket
{
      int family;
      int type;
      int protocol;
      socketaddr local;
      socketaddr remote;
};
```
Generic definition

# 응용계층 개요

- 클라이언트-서버 (9/27)
  - 소켓 인터페이스 (4/14)
    - 소켓 – 데이터 구조 (2/3)
      - Family
        - IPv4, IPv6, UNIX 도메인 프로토콜 등의 프로토콜 그룹을 정의하며, IPv4는 AF_INET, IPv6는 AF_INET6, UNIX 도메인 프로토콜은 AF_UNIX와 같이 표기함
      - Type
        - SOCK_STREAM(TCP용), SOCK DGRAM(UDP용), SOCK_SEQPACKET (SCTP용), SOCK_RAW(IP 서비스를 직접 이용하는 프로토콜용)의 4가지 소켓 유형을 정의함

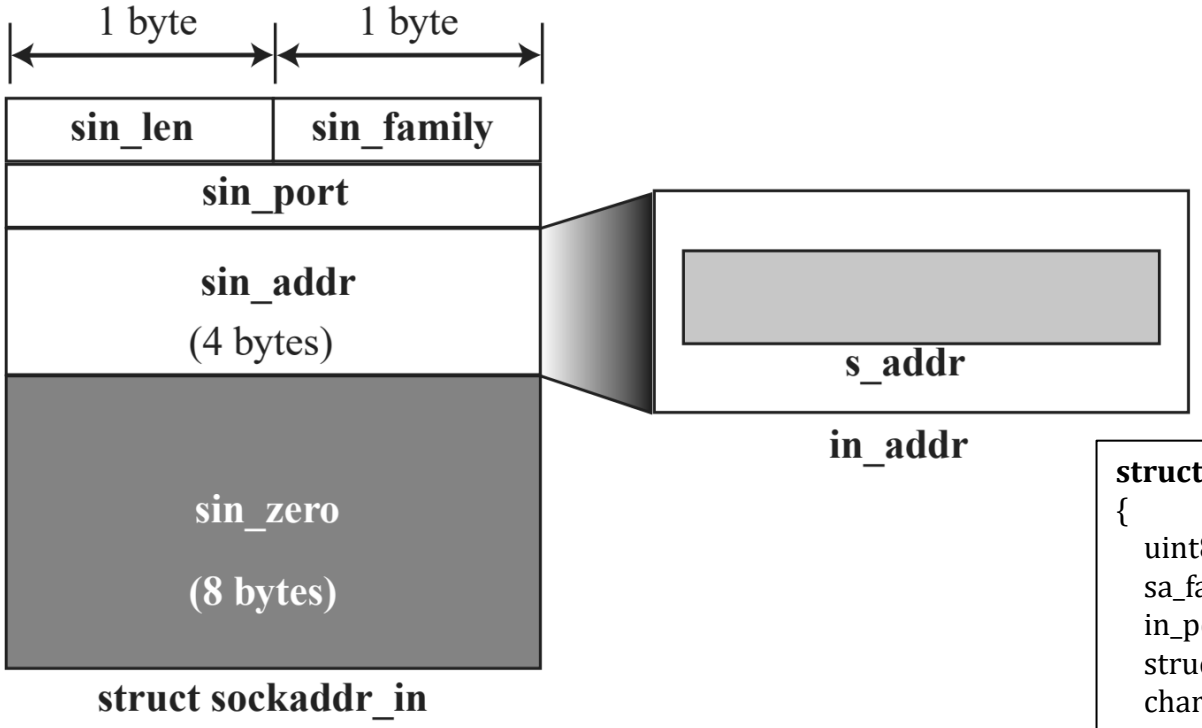


# 응용계층 개요

- 클라이언트-서버 (10/27)
  - 소켓 인터페이스 (5/14)
    - 소켓 – 데이터 구조 (3/3)
      - Protocol
        - 인터페이스를 사용하는 프로토콜을 정의하며, 0으로 설정하면 Family와 Type에 맞는 기본 프로토콜로 정의됨
      - Local Socket Address
        - IP 주소와 포트 번호를 조합한 로컬 소켓 주소를 정의함
      - Remote Socket Address
        - IP 주소와 포트 번호를 조합한 원격지 소켓 주소를 정의함

# 응용계층 개요

- 클라이언트-서버 (11/27)
  - 소켓 인터페이스 (6/14)
    - 소켓 – 주소 구조
      - 소켓 주소는 복합 데이터 구조체로 구현할 수 있음




```
struct in_addr
{
   in_addr_t  s_addr;    // A 32-bit IPv4 address
}
```

```
struct sockaddr_in
{
   uint8_t          sin_len;      // length of structure (16 bytes)
   sa_family_t      sin_family;   // set to AF_INET
   in_port_t        sin_port;     // A 16-bit port number
   struct in_addr   sin_addr;     // A 32-bit IPv4 address
   char             sin_zero[8];  // unused
}
```

# 응용계층 개요

- 클라이언트-서버 (12/27)
  - 소켓 인터페이스 (7/14)
    - 함수 (1/7)
      - socket
        - 프로세스가 소켓을 생성하기 위해서 사용하며, 성공적으로 호출된 경우에는 음수가 아닌 정수인 소켓 기술자(Socket Descriptor) *sockfd*를 반환하고, 호출에 실패하면 -1을 반환함

**int socket (int *family*, int *type*, int *protocol*);**

      - bind
        - 소켓을 로컬 컴퓨터의 로컬 포트에 결합하기 위해 사용하며, 호출 실패 시 -1을 반환함

**int bind (int *sockfd*, const struct sockaddress* *localAddress*, socklen_t *addrLen*);**

      - connect
        - 원격 소켓 주소를 소켓 구조체에 추가하기 위해 사용하며, 호출 실패 시 -1을 반환함

**int connect (int *sockfd*, const struct sockaddress* *remoteAddress*, socklen_t *addrLen*);**

# 응용계층 개요

- 클라이언트-서버 (13/27)
  - 소켓 인터페이스 (8/14)
    - 함수 (2/7)
      - listen
        - TCP 서버에 의해서만 호출되며, 운영체제에게 클라이언트 요청을 수신할 준비가 됐음을 알림
        - *backlog*는 연결 요청의 최대 개수이며, 호출에 실패 시 -1을 반환함

```
int listen (int sockfd, int backlog);
```

      - accept
        - TCP 서버가 클라이언트의 연결 요청을 수락하기 위해 사용하며, listen 함수로 대기 중인 소켓의 연결 요청 큐를 확인하고 연결 요청이 없으면 요청이 도착할 때까지 프로세스가 대기함
        - 호출에 성공하면 클라이언트와 통신하기 위한 새로운 소켓이 생성되고 해당 소켓의 기술자를 반환하며, 호출에 실패하면 -1을 반환함

```
int accept (int sockfd, const struct sockaddress* clientAddr, socklen_t* addrLen);
```

# 응용계층 개요

- 클라이언트-서버 (14/27)
  - 소켓 인터페이스 (9/14)
    - 함수 (3/7)
      - fork
        - 현재 프로세스를 복제하여 새로운 자식 프로세스를 생성하기 위해 사용하며, fork 함수를 호출한 기존 프로세스를 부모 프로세스(Parent Process), 새로 생성된 프로세스를 자식 프로세스(Child Process)라고 함
        - 부모 프로세스에서 반환 값은 양의 정수(부모 프로세스의 프로세스 ID)이며, 자식 프로세스에서 반환 값은 0이고 오류가 있으면 -1을 반환함

```
pid_t fork (fork);
```

      - send/recv
        - 연결된 소켓을 통해 데이터를 송수신하기 위해 사용하며, TCP, SCTP와 같이 연결이 설정된 상태에서 사용함
        - *sendbuf, recvbuf*는 송수신된 데이터가 저장된 버퍼의 포인터이며, *nbytes*는 데이터의 크기임
        - 호출 성공 시 송신하거나 수신된 실제 바이트 수를 반환하며, 호출 실패 시 -1을 반환함

```
int send (int sockfd, const void* sendbuf, int nbytes, int flags);

int recv (int sockfd, void* recvbuf, int nbytes, int flags);
```

# 응용계층 개요

- 클라이언트-서버 (15/27)
  - 소켓 인터페이스 (10/14)
    - 함수 (4/7)
      - sendto/recvfrom
        - UDP와 같은 비연결형 통신에서 데이터를 송수신하기 위해 사용하며, 상대방과 연결을 설정하지 않으므로 데이터 송수신 시 상대방의 소켓 주소를 함께 처리해야 함
        - *buffer*는 전송되거나 수신된 데이터가 저장된 버퍼의 포인터이며, *buflen*은 버퍼의 크기임
        - 호출 성공 시 송신하거나 수신된 실제 바이트 수를 반환하며, 호출 실패 시 -1을 반환함

```
int sendto (int sockfd, const void* buffer, int buflen, int flags, struct sockaddr* destinationAddress, socklen_t addrLen);

int recvfrom (int sockfd, void* buffer, int buflen, int flags, struct sockaddr* sourceAddress, socklen_t* addrLen);
```

      - close
        - 프로세스가 소켓을 종료하기 위해 사용하며, *sockfd*는 해당 함수를 호출한 후에는 유효하지 않게 됨
        - 호출 성공 시 정수 0을 반환하고, 실패 시 -1을 반환함

```
int close (int sockfd);
```

# 응용계층 개요

- 클라이언트-서버 (16/27)
  - 소켓 인터페이스 (11/14)
    - 함수 (5/7)
      - 바이트 순서화
        - 컴퓨터마다 다를 수 있는 바이트 저장 순서를 네트워크에서 동일하게 처리하기 위해 사용하며, 네트워크 바이트 순서는 빅 엔디언(Big-endian)을 사용함
        - *htons, htonl*은 호스트 바이트 순서를 네트워크 바이트 순서로, *ntohs, ntohl*은 네트워크 바이트 순서를 호스트 바이트 순서로 변환함
        - *s*가 붙은 함수는 16비트 정수, *l*이 붙은 함수는 32비트 정수를 대상으로 함

```
uint16_t htons (uint16_t shortValue);

uint32_t htonl (uint32_t longValue);

uint16_t ntohs (uint16_t shortValue);

uint32_t ntohl (uint32_t longValue);
```

# 응용계층 개요

- 클라이언트-서버 (17/27)
  - 소켓 인터페이스 (12/14)
    - 함수 (6/7)
      - 메모리 관리
        - 소켓 프로그래밍에서 구조체나 버퍼의 메모리를 초기화, 복사, 비교하기 위해 사용함
        - *memset*은 지정된 메모리 영역을 특정 값으로 채우고, *memcpy*는 메모리 영역의 내용을 다른 영역으로 복사하며, *memcmp*는 두 메모리 영역의 내용을 비교함
        - *memcmp*는 두 영역이 같으면 0을 반환하며, 다르면 두 영역의 비교 결과에 따라 0보다 크거나 작은 값을 반환함

```
void* memset (void* destination, int chr, size_t len);

void* memcpy (void* destination, const void* source, size_t nbytes);

int memcmp (const void* ptr1, const void* ptr2, size_t nbytes);
```

# 응용계층 개요

- 클라이언트-서버 (18/27)
  - 소켓 인터페이스 (13/14)
    - 함수 (7/7)
      - 주소 변환
        - 사람이 읽을 수 있는 문자열 형태의 IP 주소와 소켓에서 사용하는 이진 형태의 IP 주소를 서로 변환하기 위해 사용함
        - *inet_pton*은 문자열 형태의 IP 주소를 이진 형태로 변환하며, 성공 시 1, 주소 형식이 잘못된 경우 0, 오류 발생 시 -1을 반환함
        - *inet_ntop*은 이진 형태의 IP 주소를 문자열 형태로 변환하며, 성공 시 변환된 문자열의 주소를 반환하고 오류 발생 시 NULL을 반환함

```
int inet_pton (int family, const char* stringAddr, void* numericAddr);

char* inet_ntop (int family, const void* numericAddr, char* stringAddr, int len);
```

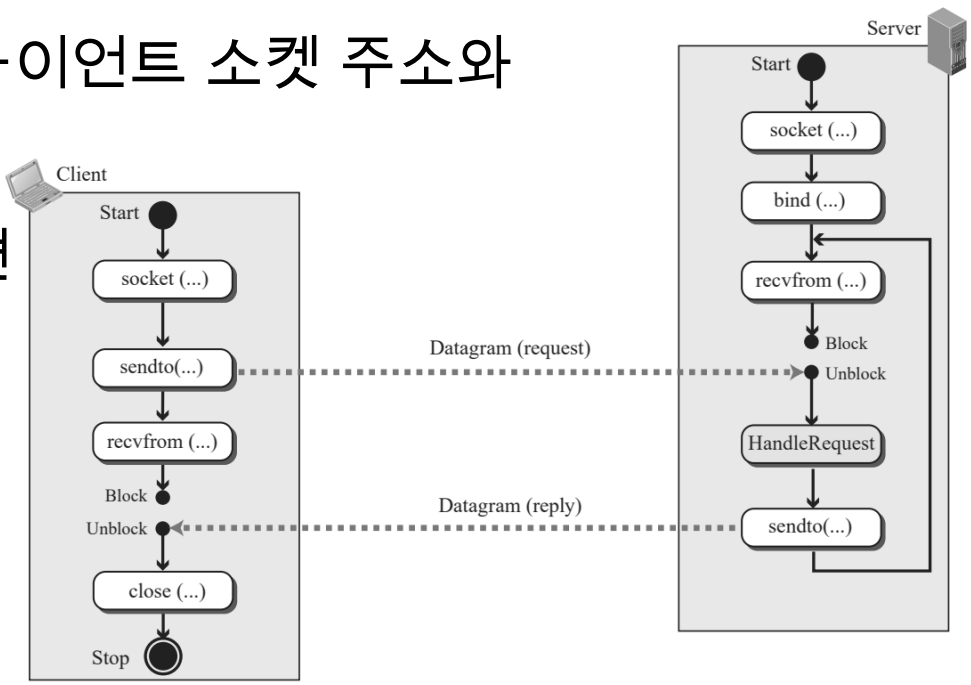
# 응용계층 개요

- 클라이언트-서버 (19/27)
  - 소켓 인터페이스 (14/14)
    - 헤더 파일
      - 함수를 사용하기 위해 다수의 헤더 파일이 필요하며, “headerFiles.h” 등으로 이름 붙인 별개의 파일을 정의하면 모든 헤더 파일을 나열하지 않아도 됨

```
// "headerFiles.h"
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <errno.h>
#include <signal.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/wait.h>
```

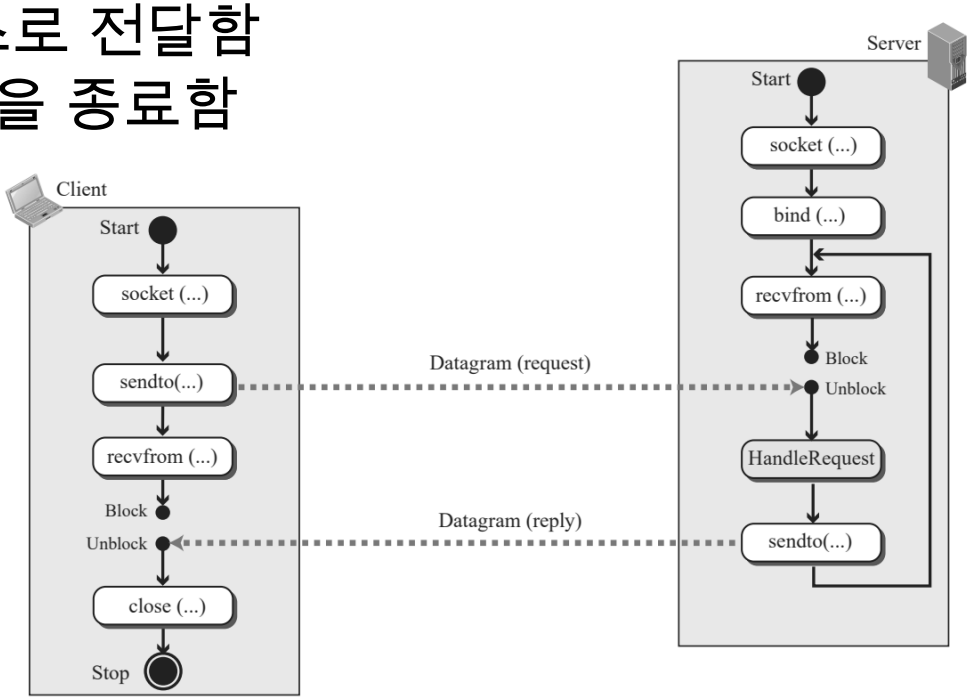
# 응용계층 개요

- 클라이언트-서버 (20/27)
  - UDP 통신 (1/2)
    - 서버 프로세스
      1. socket 함수로 소켓을 생성하고 bind 함수로 IP 주소와 포트 번호를 소켓에 결합함
      2. recvfrom 함수를 호출하고 데이터그램이 도착할 때까지 대기함
      3. 데이터그램을 수신하면 클라이언트 소켓 주소와 주소 길이를 프로세스에 반환함
      4. 프로세스가 요청을 처리하면 sendto 함수를 통해 결과를 클라이언트에게 전송함
      5. 루프를 통해 요청 대기 및 결과 전송을 반복함



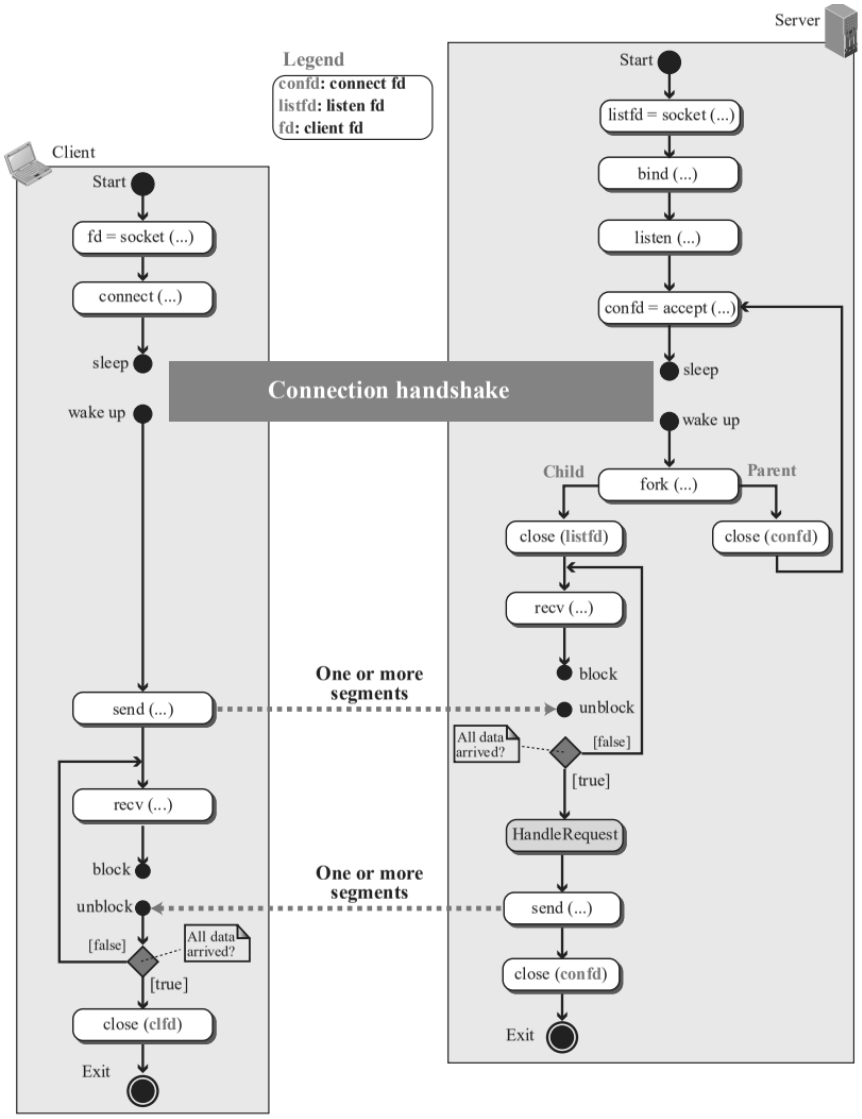
# 응용계층 개요

- 클라이언트-서버 (21/27)
  - UDP 통신 (2/2)
    - 클라이언트 프로세스
      1. socket 함수를 호출하여 소켓을 생성함
      2. sendto 함수를 통해 서버 프로세스로 요청을 전송함
      3. recvfrom 함수로 요청에 대한 응답을 수신할 때까지 대기함
      4. 데이터를 수신하면 프로세스로 전달함
      5. close 함수를 호출하여 소켓을 종료함



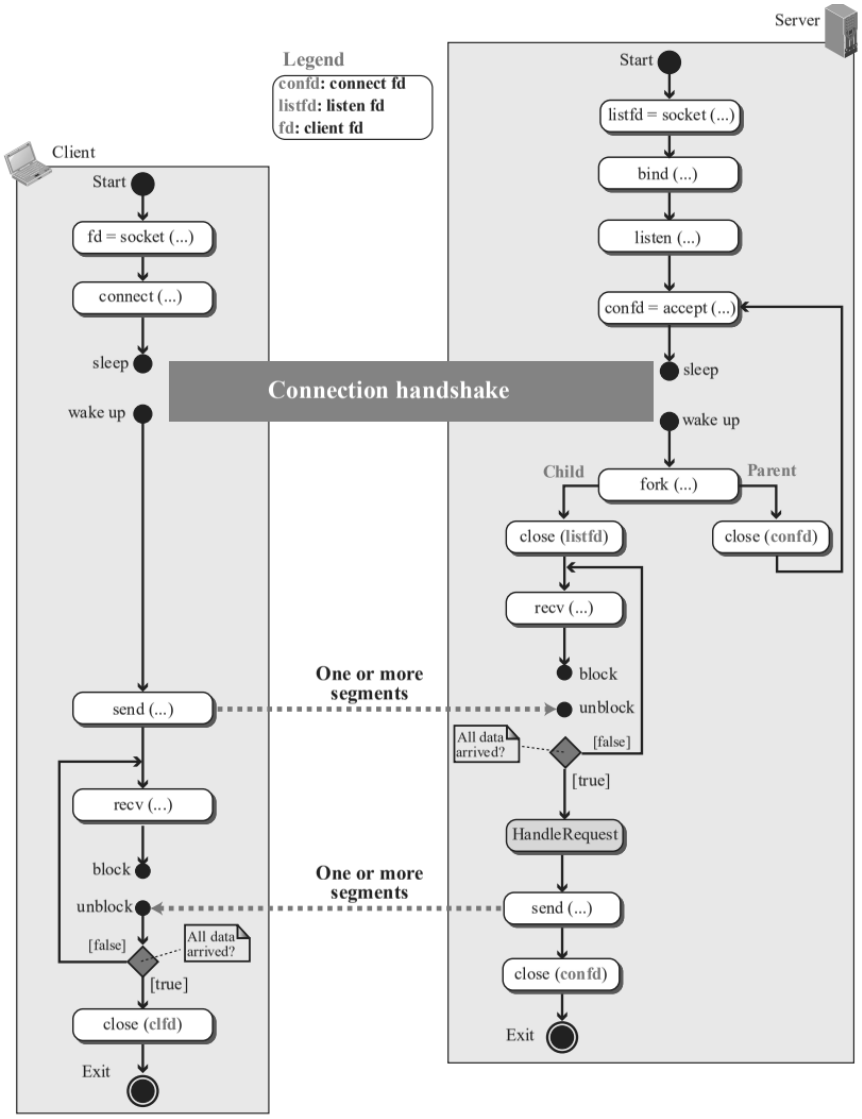
# 응용계층 개요

- 클라이언트-서버 (22/27)
  - TCP 통신 (1/6)
    - 서버 프로세스 (1/3)
      1. socket 함수로 소켓을 생성하고 bind 함수로 IP 주소와 포트 번호를 소켓에 결합함
      2. listen 함수로 클라이언트의 연결 요청을 받을 준비를 함
      3. accept 함수를 호출하고 클라이언트의 연결 요청이 도착할 때까지 대기함
      4. 연결이 설정되면 클라이언트와 통신하기 위한 새로운 연결 소켓을 생성함



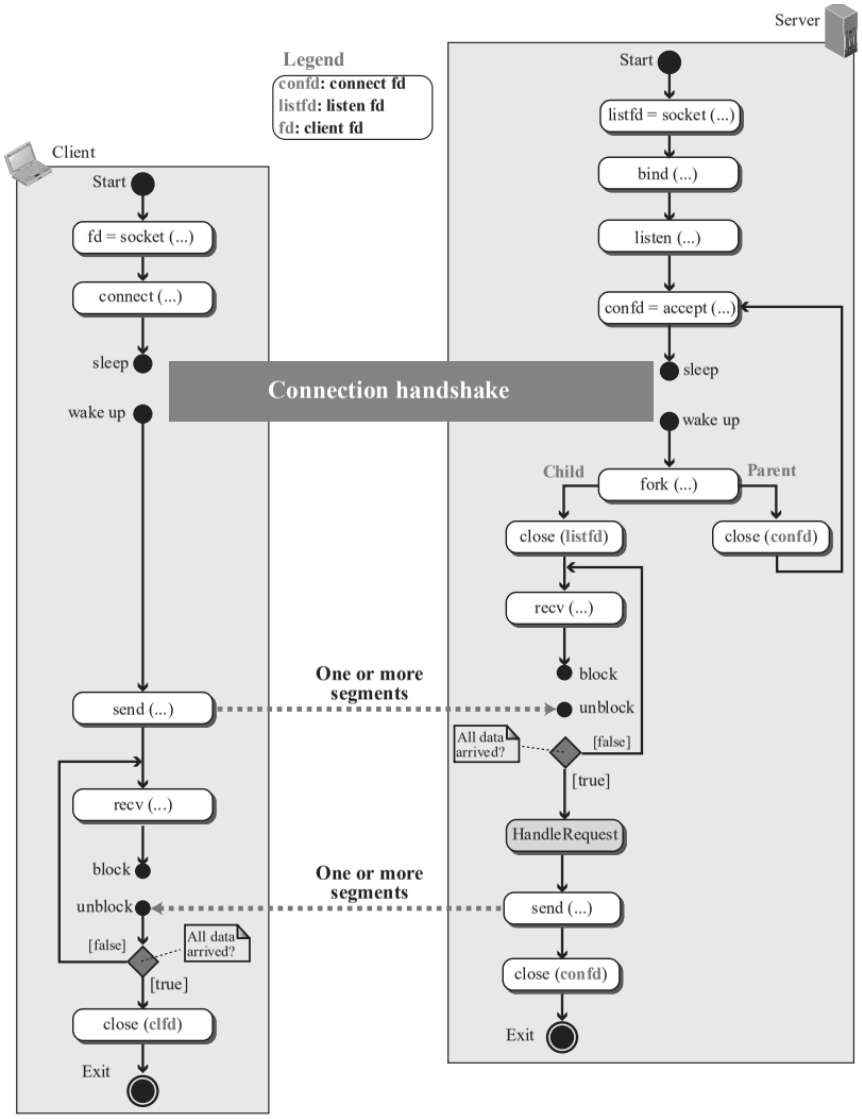
# 응용계층 개요

- 클라이언트-서버 (23/27)
  - TCP 통신 (2/6)
    - 서버 프로세스 (2/3)
      5. fork 함수로 자식 프로세스를 생성하여 클라이언트와의 통신을 담당하게 함
      6. 부모 프로세스는 소켓을 닫고 다시 accept 함수를 호출하여 다른 클라이언트의 연결 요청을 기다림
      7. 자식 프로세스는 수신 대기용 소켓을 닫고 recv 함수를 호출하여 클라이언트의 요청이 도착할 때까지 대기함



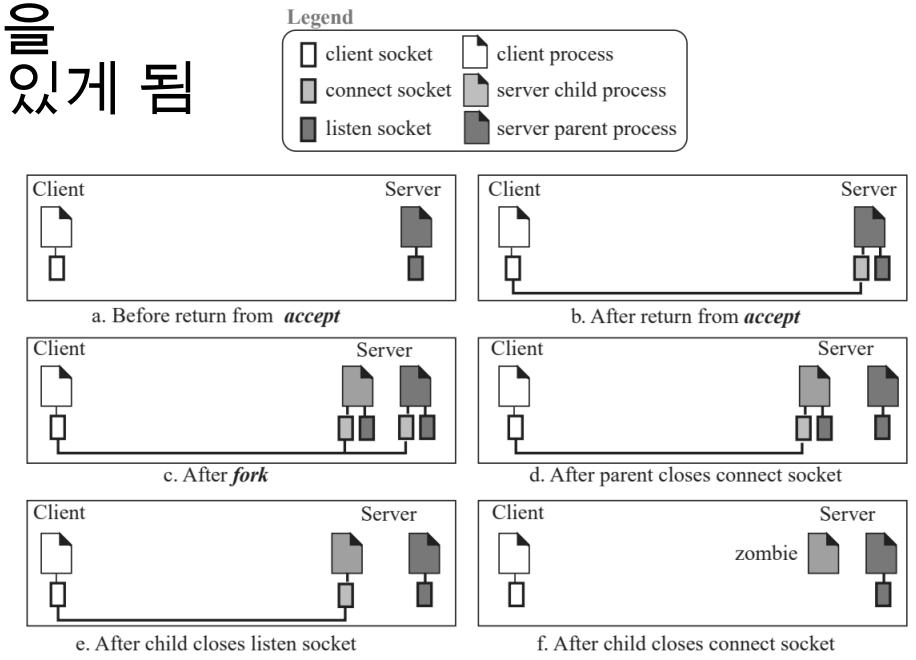
# 응용계층 개요

- 클라이언트-서버 (24/27)
  - TCP 통신 (3/6)
    - 서버 프로세스 (3/3)
      8. 요청을 수신하면 프로세스가 요청을 처리하고 send 함수를 통해 결과를 클라이언트에게 전송함
      9. recv와 send를 통해 요청 수신과 결과 전송을 반복함
      10. 클라이언트가 연결을 종료하면 연결 소켓을 닫고 자식 프로세스를 종료함



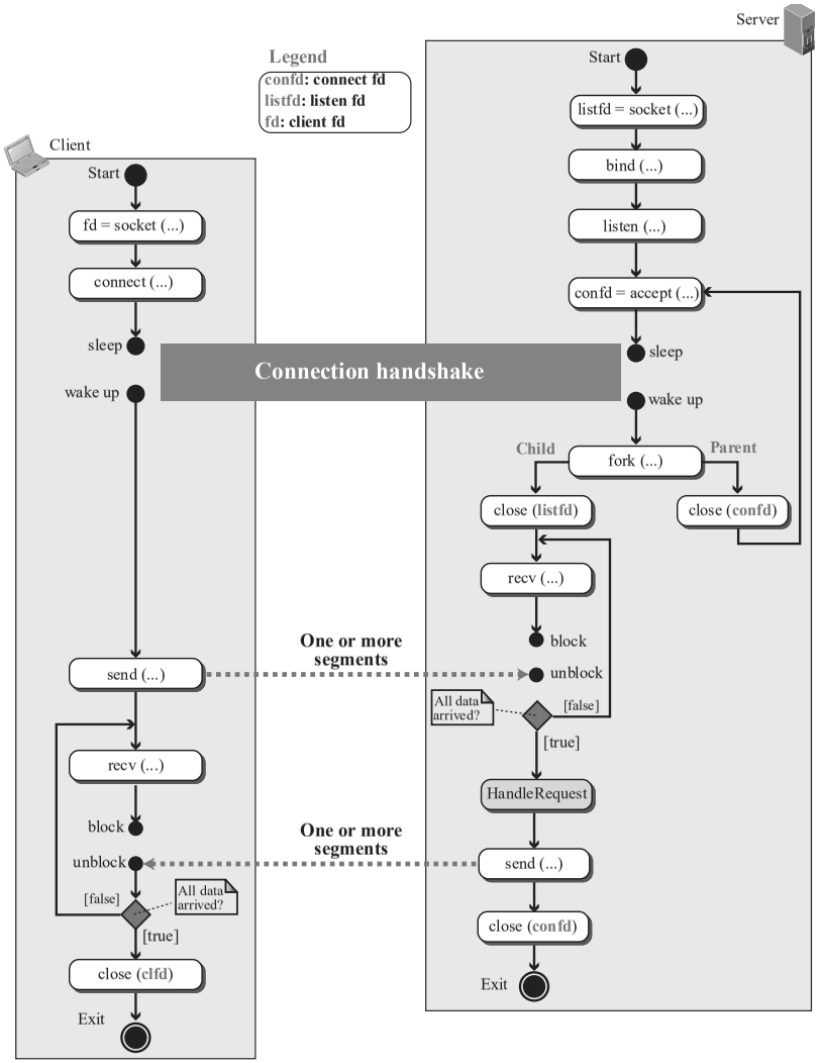
# 응용계층 개요

- 클라이언트-서버 (25/27)
  - TCP 통신 (4/6)
    - 서버 프로세스 – 소켓 관점
      1. accept 함수가 반환되기 전에는 연결 수립 없이 listen 소켓을 가짐
      2. accept 함수 이후에 연결이 수립되고 connect 소켓이 추가됨
      3. fork 함수를 통해 자식 프로세스를 추가함
      4. 부모 프로세스가 connect 소켓을 종료하고 다른 요청을 받을 수 있게 됨
      5. 자식 프로세스가 listen 소켓을 종료하고 다른 요청을 받지 않게 됨
      6. 자식 프로세스는 요청받은 서비스를 제공하고나면 connect 소켓을 종료하고 클라이언트와 분리함



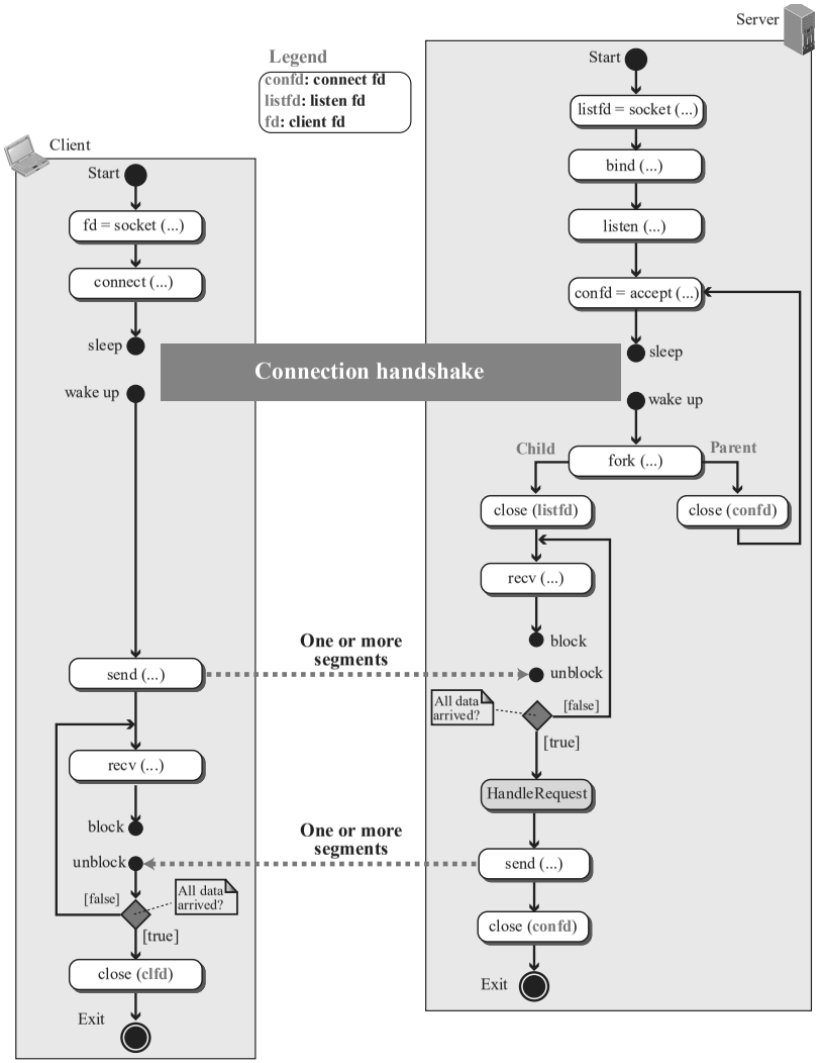
# 응용계층 개요

- 클라이언트-서버 (26/27)
  - TCP 통신 (5/6)
    - 클라이언트 프로세스 (1/2)
      1. socket 함수로 소켓을 생성함
      2. connect 함수에 서버의 IP 주소와 포트 번호를 전달하여 서버에 연결을 요청하고, TCP 연결이 설정될 때까지 대기함
      3. 연결이 설정되면 send 함수를 통해 서버에 요청 데이터를 전송함
      4. recv 함수를 호출하여 서버의 응답이 도착할 때까지 대기하고, 도착한 데이터를 수신함



# 응용계층 개요

- 클라이언트-서버 (27/27)
  - TCP 통신 (6/6)
    - 클라이언트 프로세스 (2/2)
      5. 필요한 경우 send와 recv를 통해 요청 전송과 응답 수신을 반복함
      6. 통신이 끝나면 close 함수로 소켓을 닫고 프로세스를 종료함



# 응용계층 개요

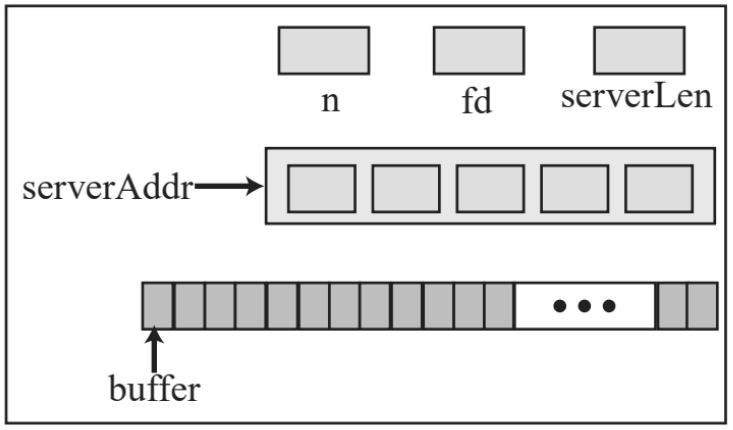
- Peer-to-peer
  - 정의
    - 두 개 이상의 대등한 피어가 서로 서비스를 요청하고 제공하는 통신 구조
  - 특징
    - 각 피어는 역할이 고정되지 않고 필요에 따라 클라이언트와 서버의 역할을 모두 수행 가능함
    - 오디오나 비디오 파일 등의 대용량 파일을 공유하거나 피어 간 직접적인 통신이 필요한 경우 유용함
    - 중앙 서버를 거치지 않고 피어 사이에서 서비스를 교환할 수 있음

# Thanks!

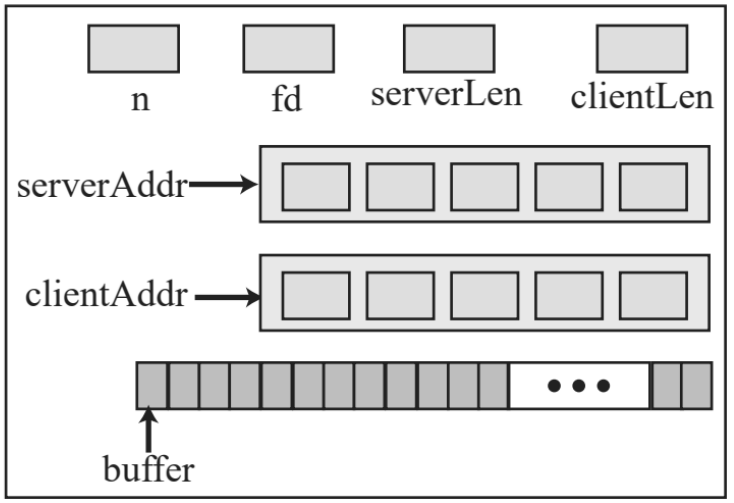
이 성 현(seonghyeon@pel.sejong.ac.kr)

# 부록 #2

- 클라이언트-서버 (1/12)
  - UDP 통신 (1/5)
    - 예시 (1/5)
      - 클라이언트가 짧은 텍스트를 요청으로 전송하고 서버가 동일한 텍스트를 응답하는 ECHO 서비스



Variables used by the client process



Variables used by the server process

# 부록 #2

- 클라이언트-서버 (2/12)
  - UDP 통신 (2/5)
    - 예시 (2/5)
      - ECHO 서비스의 서버 프로그램

```
1.  // UDP echo server program
2.  #include "headerFiles.h"
3.
4.  int main (void)
5.  {
6.      // Declaration and definition
7.      int sd;                             // Socket descriptor
8.      int nr;                             // Number of bytes received
9.      char buffer [256];                  // Data buffer
10.     struct sockaddr_in serverAddr;      // Server address
11.     struct sockaddr_in clientAddr;      // Client address
12.     int clAddrLen;                      // Length of client Address
13.
14.     // Create socket
15.     sd = socket (PF_INET, SOCK_DGRAM, 0);
16.
17.     // Bind socket to local address and port
18.     memset (&serverAddr, 0, sizeof (serverAddr));
```

# 부록 #2

- 클라이언트-서버 (3/12)
  - UDP 통신 (3/5)
    - 예시 (3/5)
      - ECHO 서비스의 서버 프로그램

```
19.     serverAddr.sin_family = AF_INET;
20.     serverAddr.sin_addr.s_addr = htonl (INADDR_ANY);  // Default address
21.     serverAddr.sin_port = htons (7);                  // We assume port 7
22.
23.     bind (sd, (struct sockaddr*) &serverAddr,
24.           sizeof (serverAddr));
25.
26.     // Receiving and echoing datagrams
27.     for ( ; ; )                 // Run forever
28.     {
29.         nr = recvfrom (sd, buffer, 256, 0,
30.                        (struct sockaddr*)&clientAddr,
31.                        &clAddrLen);
32.
33.         sendto (sd, buffer, nr, 0,
34.                 (struct sockaddr*)&clientAddr,
35.                 sizeof(clientAddr));
36.     }
37. } // End of echo server program
```

# 부록 #2

- 클라이언트-서버 (4/12)
  - UDP 통신 (4/5)
    - 예시 (4/5)
      - ECHO 서비스의 클라이언트 프로그램

```
1.  // UDP echo client program
2.  #include "headerFiles.h"
3.
4.  int main (void)
5.  {
6.      // Declaration and definition
7.      int sd;                                 // Socket descriptor
8.      int ns;                                 // Number of bytes send
9.      int nr;                                 // Number of bytes received
10.     char buffer [256];                      // Data buffer
11.     struct sockaddr_in serverAddr;          // Socket address
12.
13.     // Create socket
14.     sd = socket (PF_INET, SOCK_DGRAM, 0);
15.     // Create server socket address
16.     memset (&servAddr, 0, sizeof(serverAddr));
17.     servAddr.sin_family = AF_INET;
18.     inet_pton (AF_INET, "server address",
19.                &serverAddr.sin_addr);
```

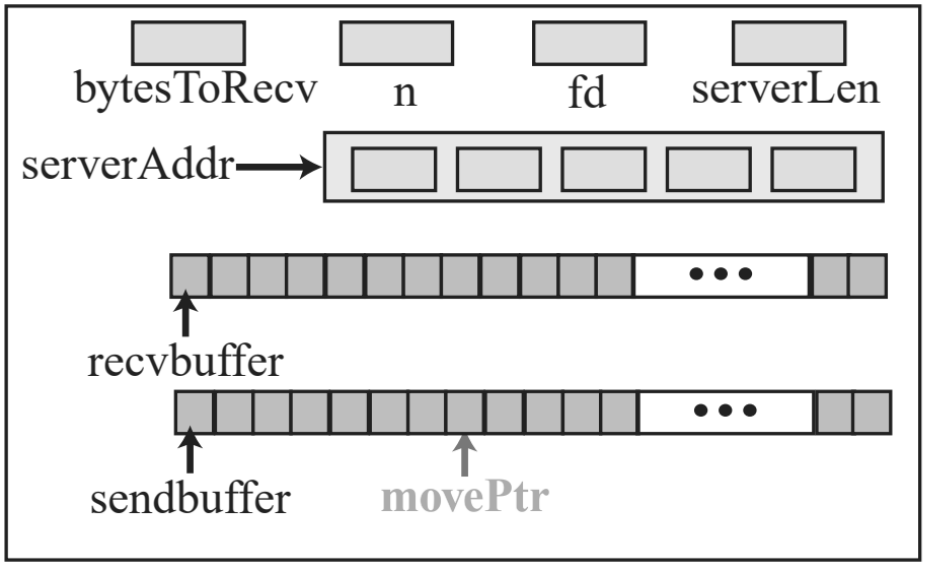
# 부록 #2

- 클라이언트-서버 (5/12)
  - UDP 통신 (5/5)
    - 예시 (5/5)
      - ECHO 서비스의 클라이언트 프로그램

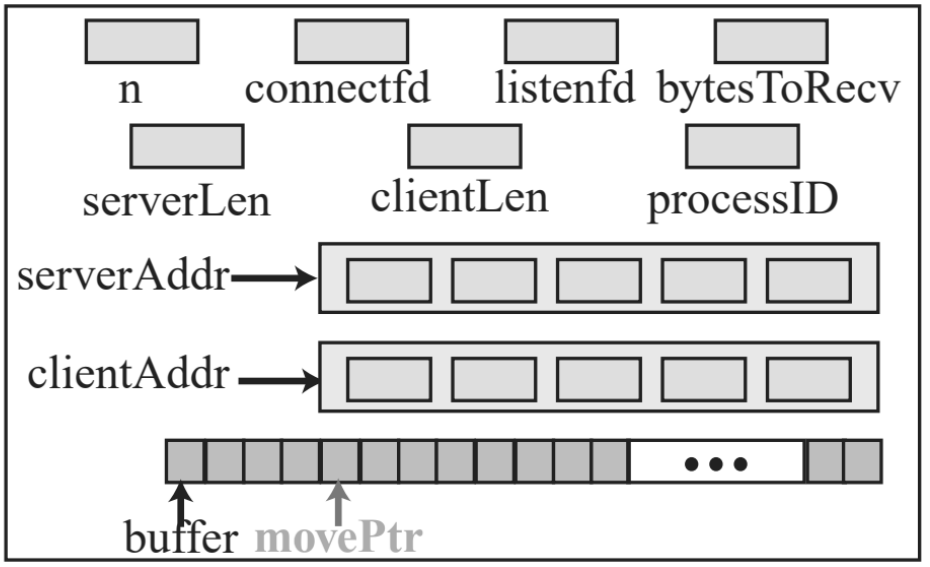
```
20.     serverAddr.sin_port = htons (7);
21.
22.     // Send and receive datagrams
23.     fgets (buffer, 256, stdin);
24.
25.     ns = sendto (sd, buffer, strlen (buffer), 0,
26.                  (struct sockaddr)&serverAddr,
27.                  sizeof(serveraddr));
28.
29.     recvfrom (sd, buffer, strlen (buffer),
30.               0, NULL, NULL);
31.     buffer [nr] = 0;
32.
33.     printf ("Received from server:");
34.     fputs (buffer, stdout);
35.     // Close and exit
36.     close (sd);
37.     exit (0);
38. } // End of echo client program
```

# 부록 #2

- 클라이언트-서버 (6/12)
  - TCP 통신 (1/7)
    - 예시 (1/7)
      - 클라이언트가 짧은 텍스트를 요청으로 전송하고 서버가 동일한 텍스트를 응답하는 ECHO 서비스




Variables used by the client process




Variables used by the server process

# 부록 #2

- 클라이언트-서버 (7/12)
  - TCP 통신 (2/7)
    - 예시 (2/7)
      - ECHO 서비스의 서버 프로그램

```
1.  // Echo server program
2.  #include "headerFiles.h"
3.
4.  int main (void)
5.  {
6.      // Declaration and definition
7.      int listensd;                       // Listen socket descriptor
8.      int connectsd;                      // Connecting socket descriptor
9.      int n;                              // Number of bytes in each reception
10.     int bytesToRecv;                    // Total bytes to receive
11.     int processID;                      // ID of the child process
12.     char buffer [256];                  // Data buffer
13.     char* movePtr;                      // Pointer to the buffer
14.     struct sockaddr_in serverAddr;      // Server address
15.     struct sockaddr_in clientAddr;      // Client address
16.     int clAddrLen;                      // Length of client address
17.
18.     // Create listen socket
19.     listensd = socket (PF_INET, SOCK_STREAM, 0);
```

# 부록 #2

- 클라이언트-서버 (8/12)
  - TCP 통신 (3/7)
    - 예시 (3/7)
      - ECHO 서비스의 서버 프로그램

```
20.     // Bind listen socket to the local address and port
21.     memset (&serverAddr, 0, sizeof (serverAddr));
22.     serverAddr.sin_family = AF_INET;
23.     serverAddr.sin_addr.s_addr = htonl (INADDR_ANY);
24.     serverAddr.sin_port = htons (7);     // We assume port 7
25.
26.     bind (listensd, &serverAddr, sizeof (serverAddr));
27.
28.     // Listen to connection requests
29.     listen (listensd, 5);
30.
31.     // Handle the connection
32.     for ( ; ; )                          // Run forever
33.     {
34.         connectsd = accept (listensd, &clientAddr, &clAddrLen);
35.
36.         processID = fork ();
```

# 부록 #2

- 클라이언트-서버 (9/12)
  - TCP 통신 (4/7)
    - 예시 (4/7)
      - ECHO 서비스의 서버 프로그램

```
37.         if (processID == 0)              // Child process
38.         {
39.             close (listensd);
40.             bytesToRecv = 256;
41.             movePtr = buffer;
42.             while ((n = recv (connectfd, movePtr,
43.                               bytesToRecv, 0)) > 0)
44.             {
45.                 movePtr = movePtr + n;
46.                 bytesToRecv = movePtr - n;
47.             } // End of while
48.
49.             send (connectsd, buffer, 256, 0);
50.             exit (0);
51.         } // End of if
52.         close (connectsd):
53.     } // End of for loop
54. } // End of echo server program
```

# 부록 #2

- 클라이언트-서버 (10/12)
  - TCP 통신 (5/7)
    - 예시 (5/7)
      - ECHO 서비스의 클라이언트 프로그램

```
1.  // TCP echo client program
2.  #include "headerFiles.h"
3.
4.  int main (void)
5.  {
6.      // Declaration and definition
7.      int sd;                                 // Socket descriptor
8.      int n;                                  // Number of bytes received
9.      int bytesToRecv;                        // Number of bytes to receive
10.     char sendBuffer [256];                  // Send buffer
11.     char recvBuffer [256];                  // Receive buffer
12.     char* movePtr;                          // A pointer the received buffer
13.     struct sockaddr_in serverAddr;          // Server address
14.
15.     // Create socket
16.     sd = socket (PF_INET, SOCK_STREAM, 0);
17.
18.     // Create server socket address
19.     memset (&serverAddr, 0, sizeof(serverAddr));
```

# 부록 #2

- 클라이언트-서버 (11/12)
  - TCP 통신 (6/7)
    - 예시 (6/7)
      - ECHO 서비스의 클라이언트 프로그램

```
20.     serverAddr.sin_family = AF_INET;
21.     inet_pton (AF_INET, "server address",
22.                &serverAddr.sin_addr);
23.     serverAddr.sin_port = htons (7);     // We assume port 7
24.
25.     // Connect
26.     connect (sd, (struct sockaddr*)&serverAddr,
27.              sizeof(serverAddr));
28.     // Send and receive data
29.     fgets (sendBuffer, 256, stdin);
30.
31.     send (fd, sendBuffer, strlen (sendbuffer), 0);
32.
33.     bytesToRecv = strlen (sendbuffer);
34.     movePtr = recvBuffer;
35.
36.     while ((n = recv (sd, movePtr, bytesToRecv, 0)) > 0)
37.     {
38.         movePtr = movePtr + n;
```

# 부록 #2

- 클라이언트-서버 (12/12)
  - TCP 통신 (7/7)
    - 예시 (7/7)
      - ECHO 서비스의 클라이언트 프로그램

```
39. bytesToRecv = bytesToRecv - n;
40.     } // End of while loop
41.
42.     recvBuffer[bytesToRecv] = 0;
43.
44.     printf ("Received from server:");
45.     fputs (recvBuffer, stdout);
46.
47.     // Close and exit
48.     close (sd);
49.     exit (0);
50. } // End of echo client program
```