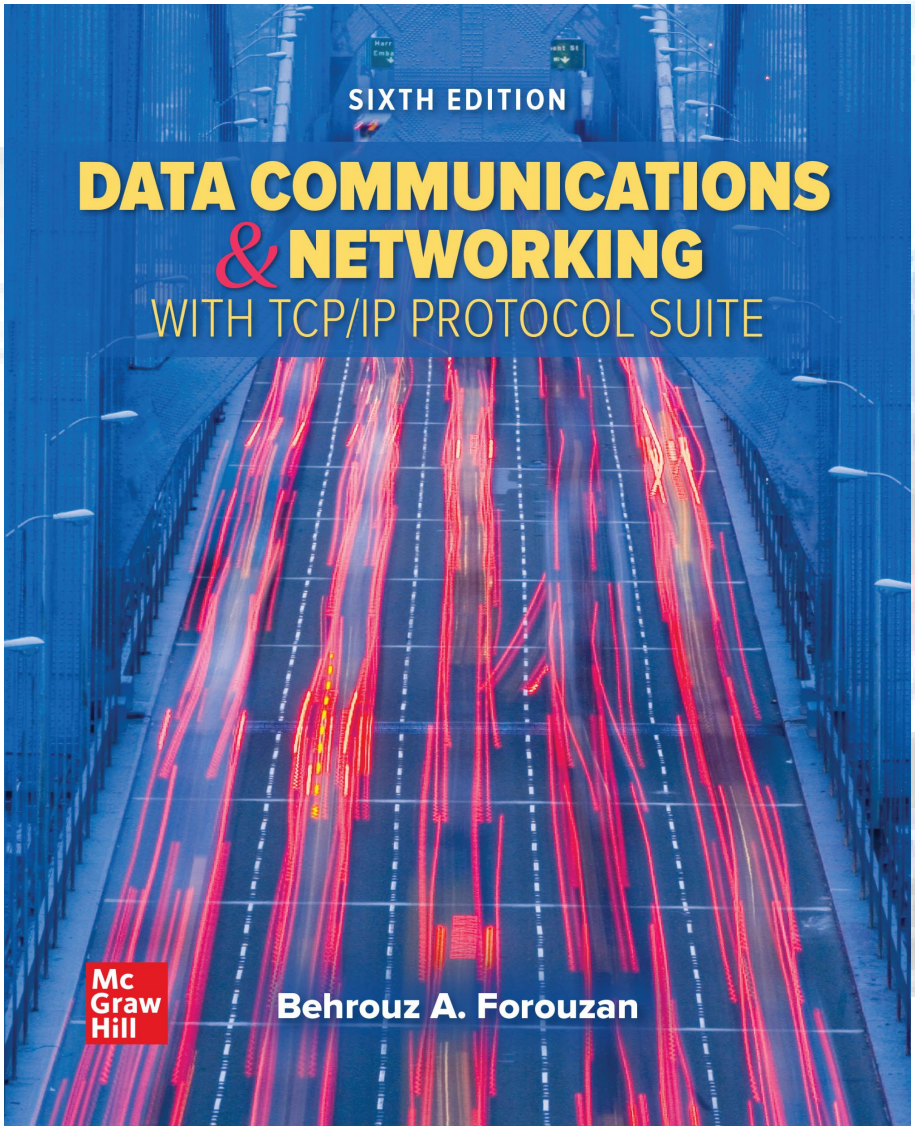
9장

Transport Layer



목 차

9.1 전송계층 서비스

9.2 전송계층 프로토콜

9.3 UDP(User Datagram Protocol)

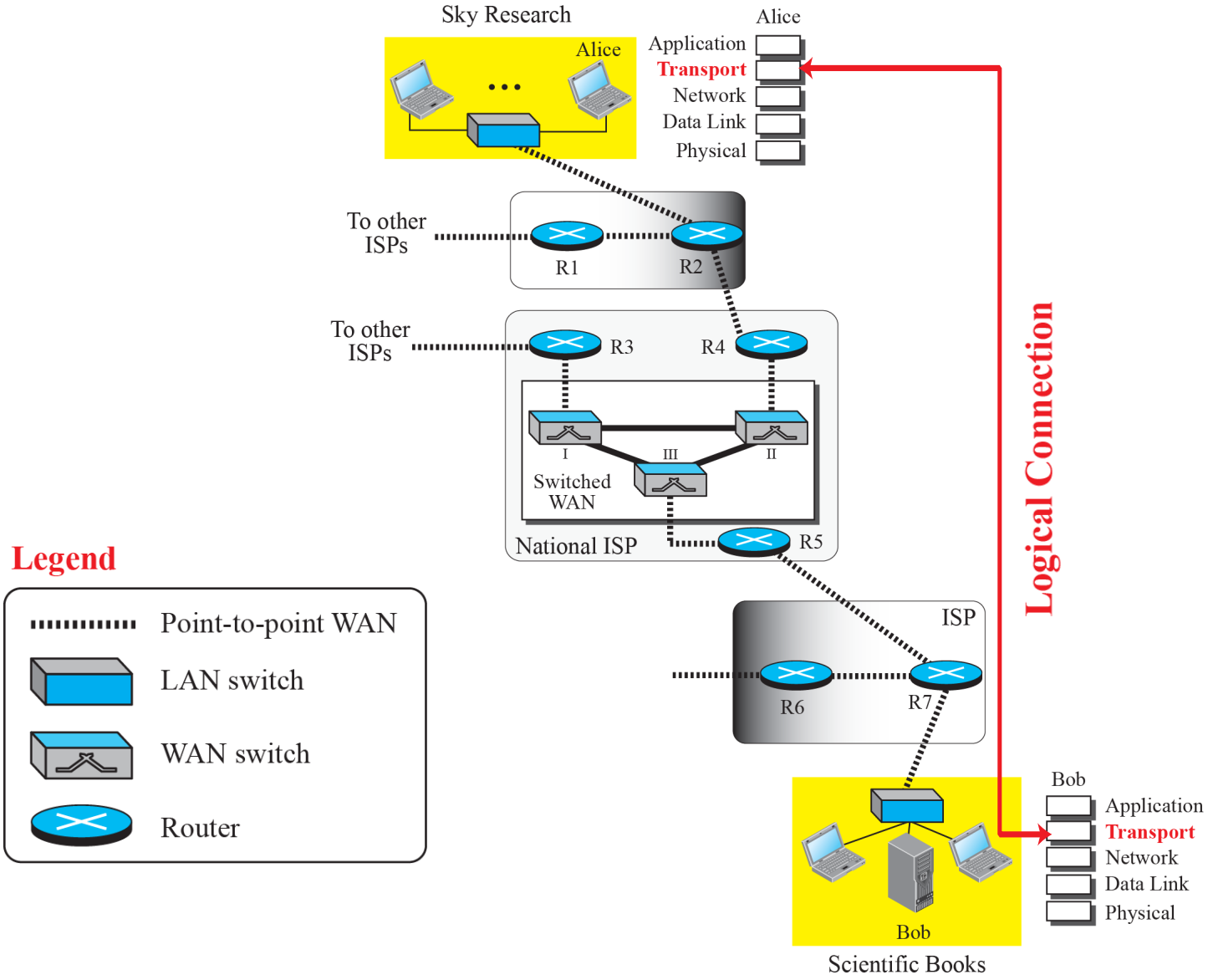
9.4 TCP(Transmission Control Protocol)

9.5 SCTP(Stream Control Transmission Protocol)

9.1 도입

- 전송층은 응용층과 네트워크층 사이에 위치한다.
- 이것은 두 응용층 사이, 하나는 지역 호스트와 다른 하나는 원격 호스트, 프로세스-대-프로세스 통신을 제공한다.
- 통신은 논리적 연결을 이용하여 제공된다. 지구의 다른 지역에 위치한 2개의 응용층이 가상적으로 직접 연결을 통하여 이들 사이에 메시지 송신과 수신이 가능하다는 것을 의미한다.
- 그림 9.1은 이 논리적 연결 뒤의 아이디어를 보여준다.

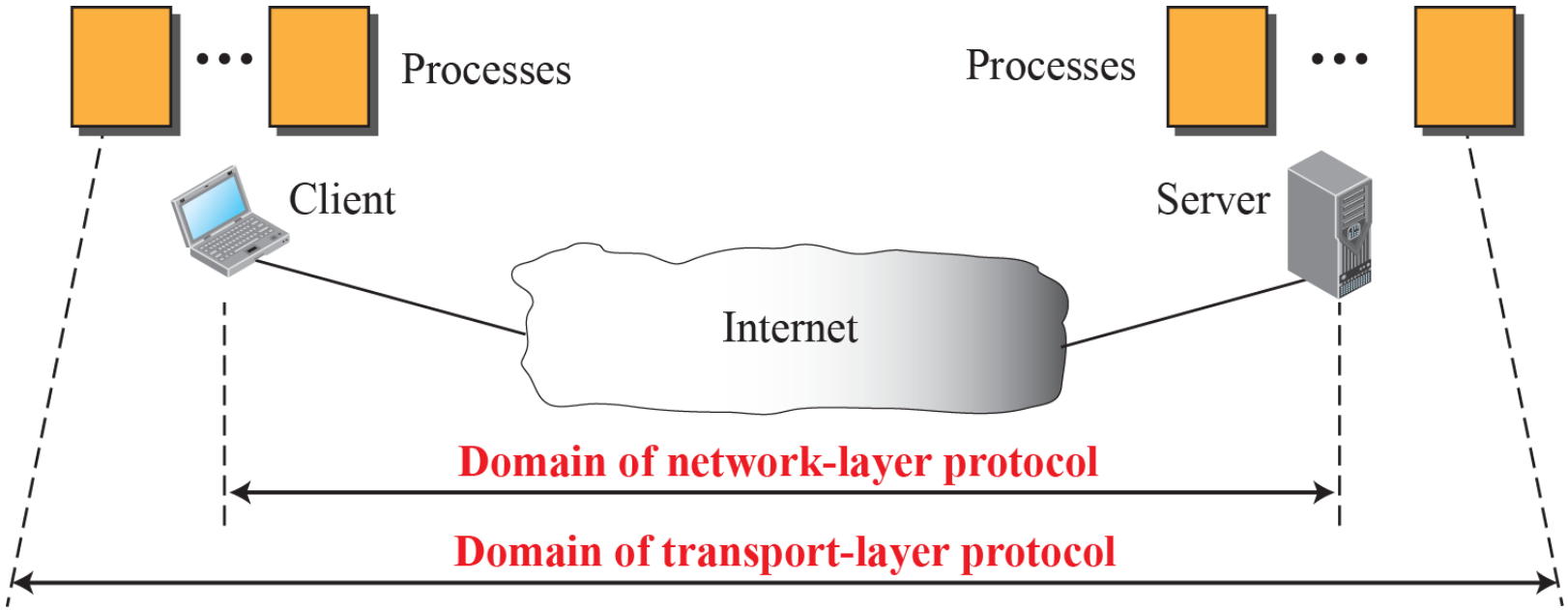
그림 9.1: 전송층에서 논리적 연결



9.1 전송층 서비스

- 전송층은 네트워크층과 응용층 사이에 위치한다.
- 전송층은 응용층에 서비스를 제공하는 책임이 있다. 이것은 네트워크층으로부터 서비스를 받는다.
- 이 절에서 우리는 전송층에 의해 제공될 수 있는 서비스를 다루고, 다음 절에서 다수의 전송층 프로토콜을 다룬다.

그림 9.2: 네트워크층 대 전송층



9.1.2 포트번호(Port Number)

- 프로세스 간 통신을 달성하는 몇 가지 방법이 있지만 가장 일반적인 방법은 클라이언트-서버 패러다임을 사용하는 것이다. 클라이언트라고 하는 로컬 호스트의 프로세스는 일반적으로 서버라고 하는 원격 호스트의 프로세스에서 서비스를 필요로 한다

- 그러나 오늘날 운영 체제는 다중 사용자 및 다중 프로그래밍 환경을 모두 지원한다. 여러 로컬 컴퓨터가 동시에 하나 이상의 클라이언트 프로그램을 실행할 수 있는 것처럼 원격 컴퓨터는 동시에 여러 프로그램을 실행할 수 있다.

그림 9.3: 포트 번호

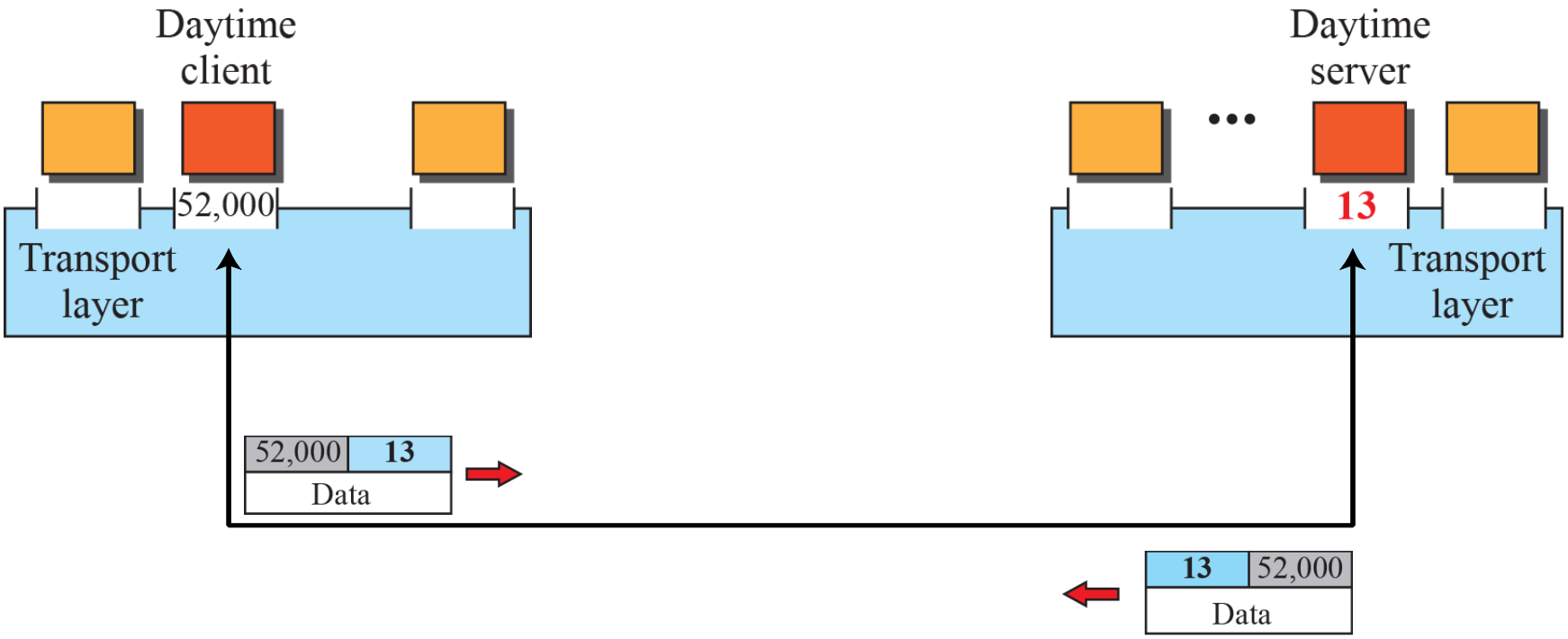

Daytime
client
52,000
Transport
layer
Daytime
server
13
Transport
layer
52,000
13
Data
13
52,000
Data

그림 9.4: IP 주소 대 포트 번호

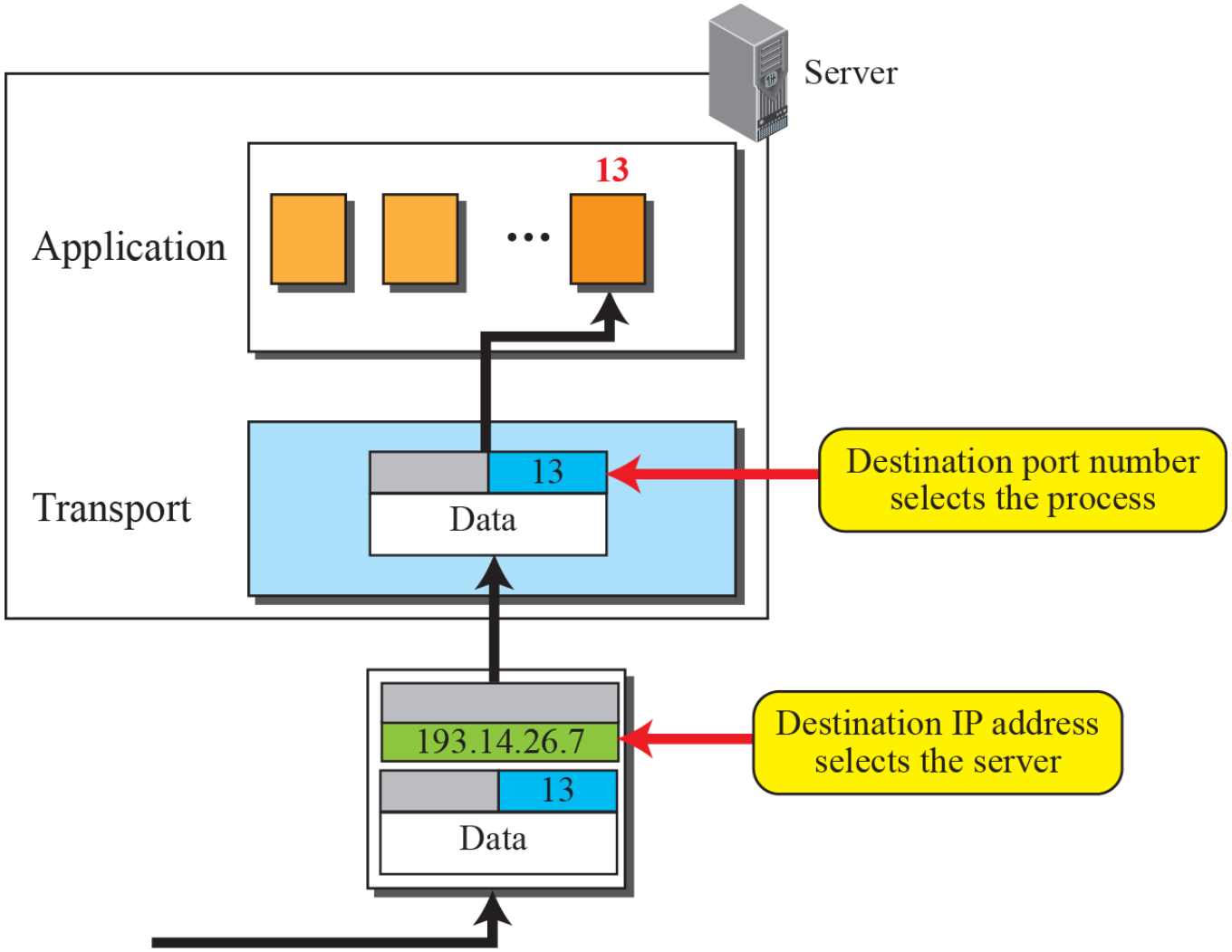


그림 9.5: ICANN 범위

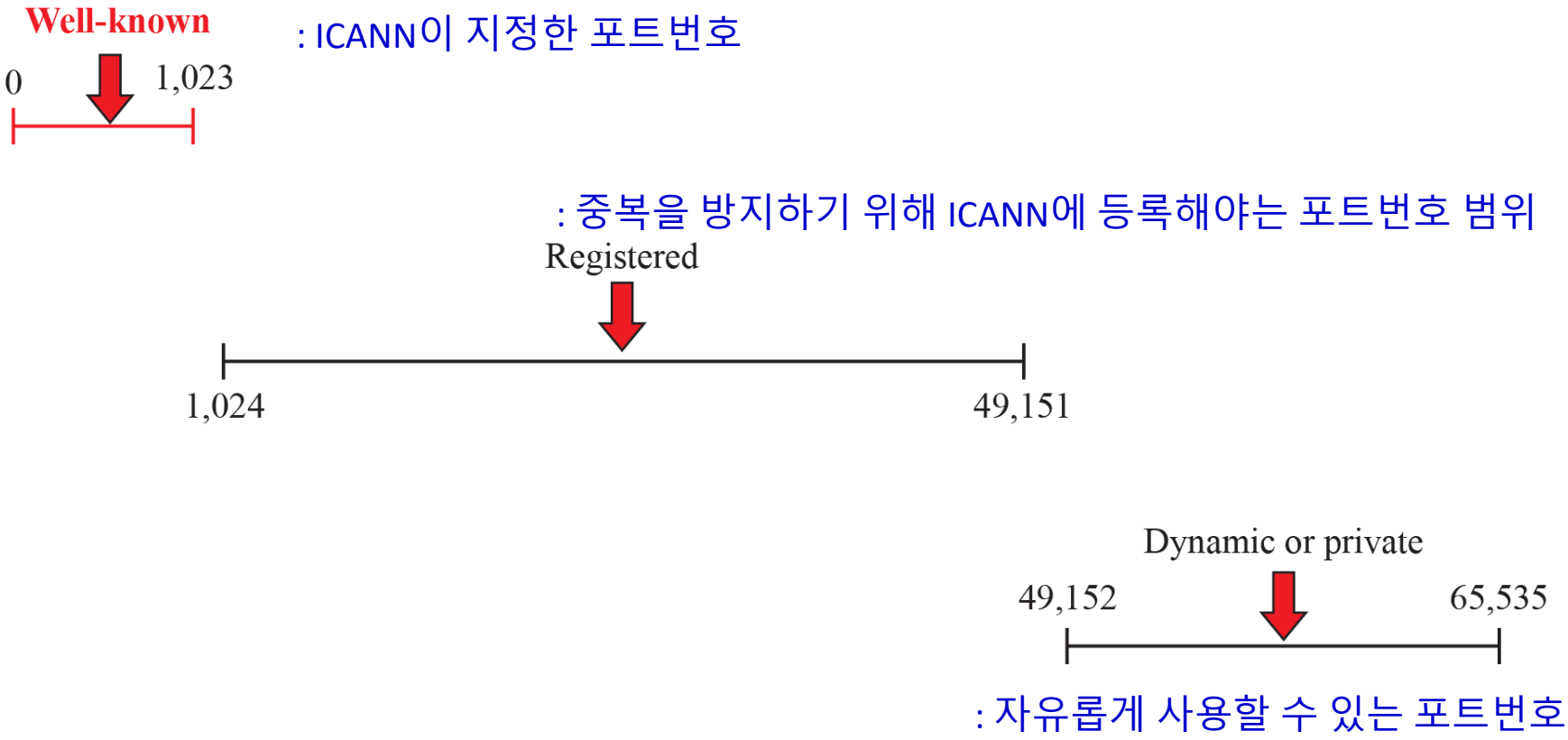
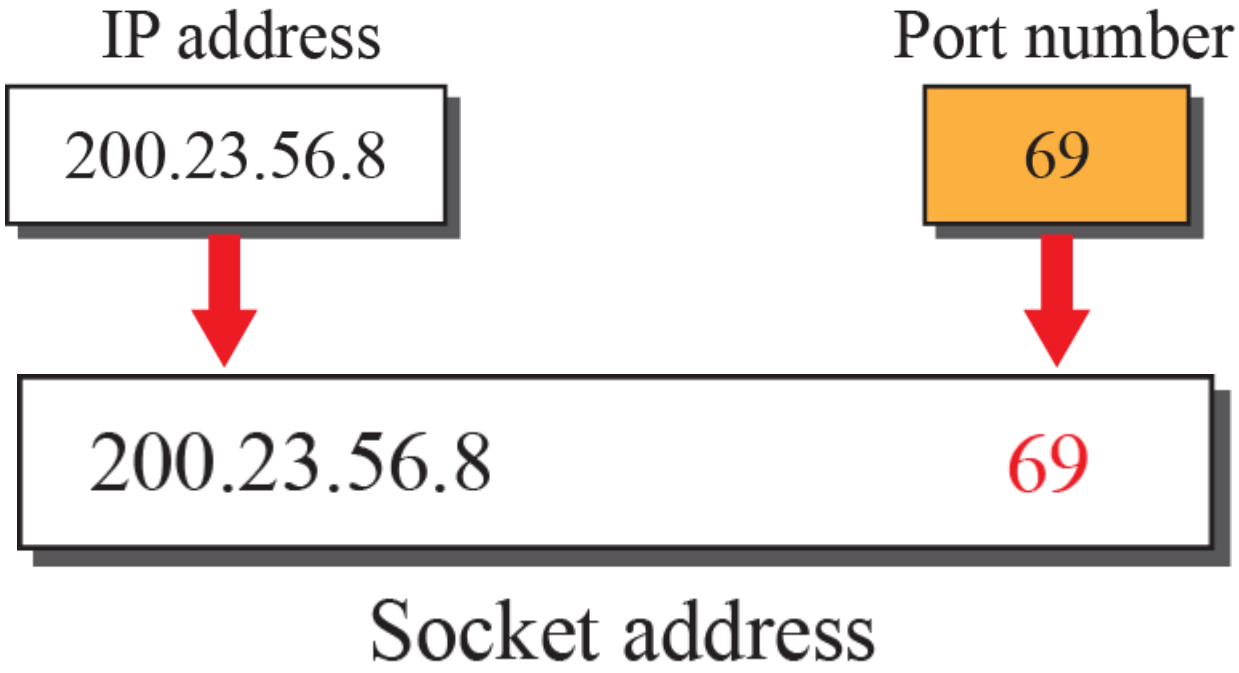


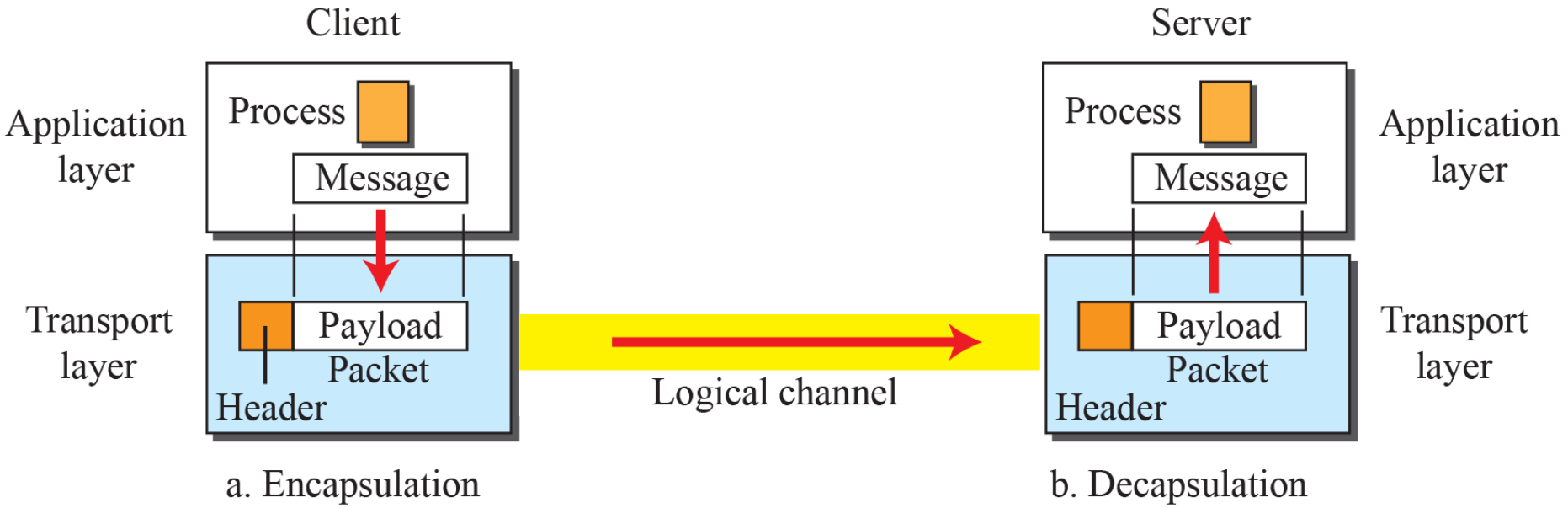
그림 9.6: 소켓 주소


IP address
Port number
200.23.56.8
69
200.23.56.8
69
Socket address

9.1.3 캡슐화와 캡슐제거(Encapsulation & Decapsulation)

- 한 프로세스에서 다른 프로세스로 메시지를 보내기 위해 전송 계층 프로토콜은 메시지를 캡슐화하고 캡슐화를 제거한다. 캡슐화는 송신자 사이트에서 발생한다. 다.
- 프로세스가 보낼 메시지가 있으면 전송 계층 프로토콜에 따라 달라지는 소켓 주소 및 기타 정보와 함께 메시지를 전송 계층으로 전달한다.
- 전송 계층은 데이터를 수신하고 전송 계층 헤더를 추가한다.

그림 9.7: 캡슐화와 캡슐 제거



9.1.4 다중화 및 역다중화(Multiplexing & Demultiplexing)

- 엔티티가 둘 이상의 소스에서 항목(items)을 받을 때, 이를 다중화(다대일: many-to-one)라고 한다.
- 엔티티가 둘 이상의 목적지에 항목을 전달할 때마다 이것을 역다중화(일대다: one-to-many)라고 한다.
- 소스의 전송 계층은 다중화를 수행한다. 목적지의 전송 계층은 역다중화를 수행한다.

그림 9.8: 다중화 및 역다중화

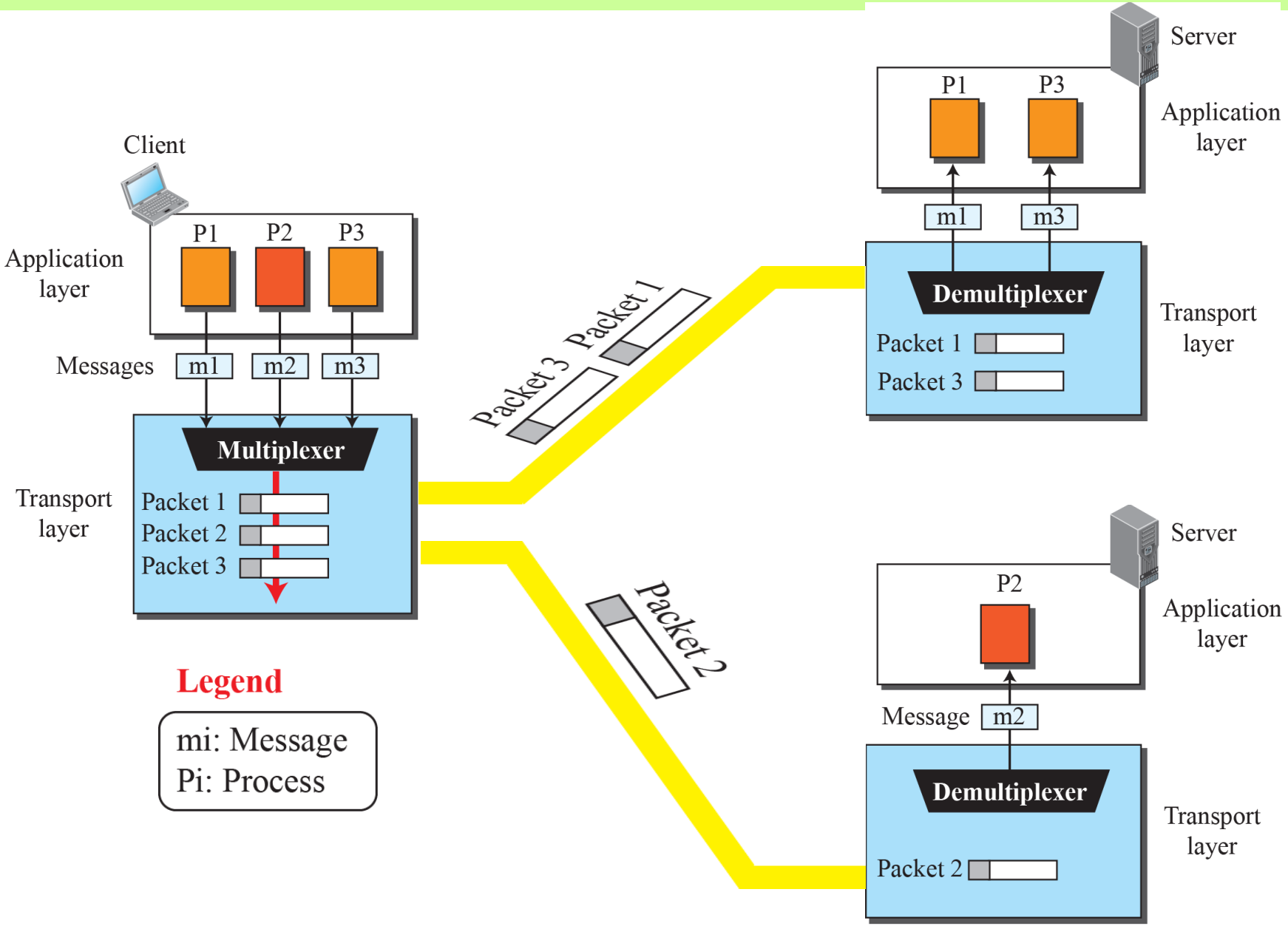
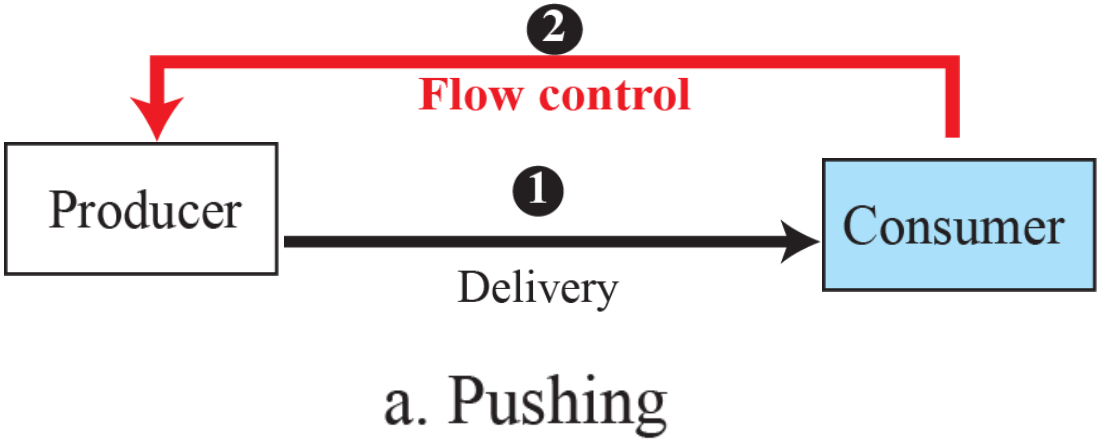
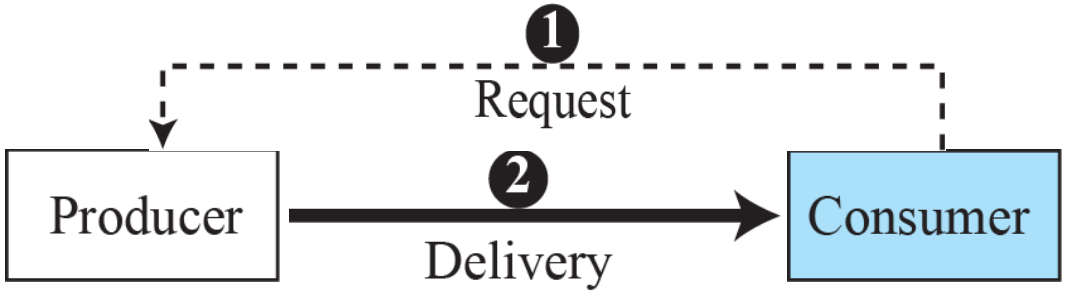


그림 9.9: 밀기(pushing) 혹은 끌기(pulling)



a. Pushing



b. Pulling

: Flow control이 필요 없음

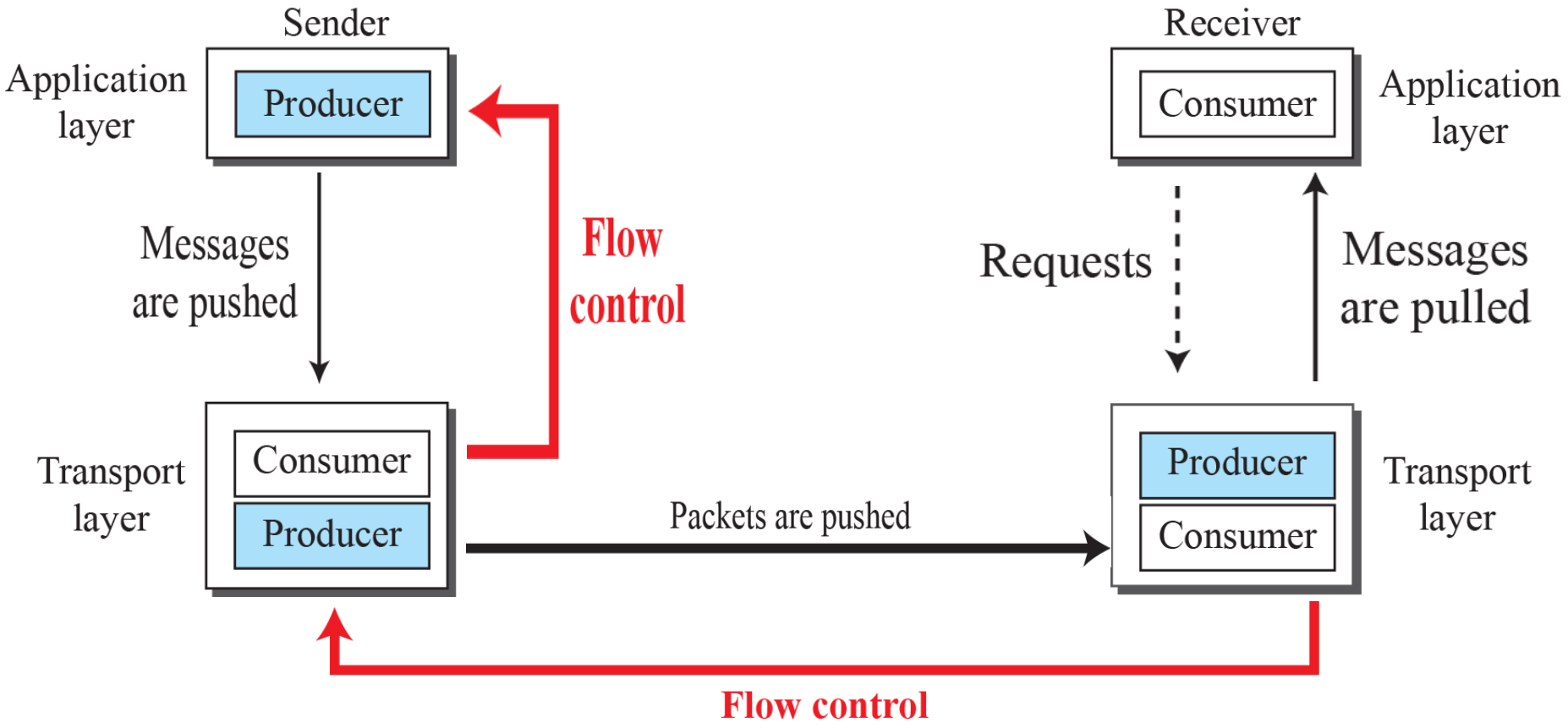
9.1.5 흐름제어(Flow Control)

- 기업이 품목을 생산하고 다른 기업이 이를 소비할 때마다 생산율과 소비율 사이에 균형이 있어야 한다.
- 품목이 소비할 수 있는 것보다 빨리 생산되면 소비자가 압도되어 (Overwhemed) 일부 품목을 폐기해야 할 수도 있다.
- 품목이 소비될 수 있는 것보다 느리게 생산되면 소비자는 기다려야 하고 시스템의 효율성이 떨어진다.
- 흐름 제어는 첫 번째 문제와 관련이 있다. 소비자 사이트에서 데이터 항목이 손실되는 것을 방지해야 한다.

흐름제어 처리 (Handling Flow Control)

- 전송 계층에서의 통신에서 우리는 4개의 엔터티(Entity)를 처리한다.
  - ➲ 발신자 프로세스, 발신자 전송계층, 수신자 전송 계층, 수신자 프로세스.
- 애플리케이션 계층의 전송 프로세스는 생산자(producer)일 뿐이다. 메시지 청크(Chunk)를 생성하여 전송 계층으로 푸시한다.
- 송신 전송 계층은 소비자(consumer)이자 생산자라는 이중 역할을 한다.
- 생산자가 푸시한 메시지를 사용한다.
- 메시지를 패킷으로 캡슐화하여 수신 전송 계층으로 푸시한다.
- 수신 전송 계층도 이중 역할을 한다.
- 메시지를 캡슐을 제거 하고 애플리케이션 계층에 전달하는 것은 발신자 및 생산자로부터 수신된 패킷에 대한 소비자이다.

그림 9.10: 전송층에서 흐름 제어



예제 9.1

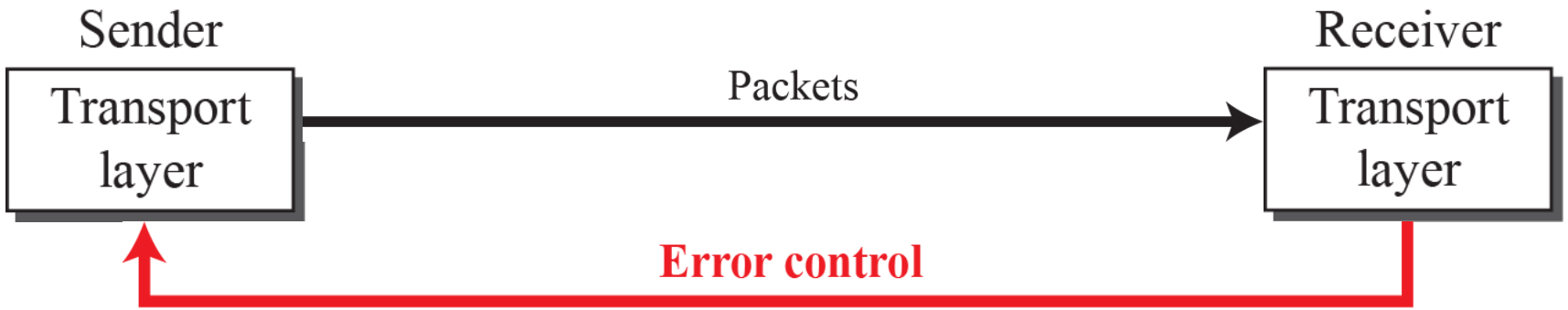
- 앞에서 언급한 내용들은 버퍼가 다 차거나 또는 버퍼에 빈 공간이 생기는 두 경우에 소비자와 생산자 간에 통신이 필요하다는 것을 의미한다.
- 만일 2개체가 단지 한 슬롯의 버퍼를 이용하는 경우의 통신은 비교적 쉽다. 각각의 전송층이 패킷을 저장하기 위하여 하나의 단일 메모리 영역을 사용한다고 가정해 보자.
- 송신 전송층의 단일 슬롯이 비게 되면, 송신 전송층은 다음 메시지를 전송하라는 신호를 응용층에서 전송한다. 반면 수신 전송층에서 이러한 단일 슬롯이 비게 되면, 수신 전송층은 다음 패킷의 전송을 위하여 송신 전송층에게 확인응답(acknowledgement)을 보낸다.
- 뒷부분에서 살펴보겠지만, 송신측과 수신측에서 단일-슬롯 버퍼를 이용하는 이러한 종류의 흐름 제어는 비효율적이다.

9.1.6 에러제어(Error Control)

■ 전송층에서의 오류제어 역할

1) 손상된 패킷(corrupted packet)을 검출하고 폐기
2) 손실되거나 폐기된 패킷을 추적하고 재 전송
3) 중복전송 패킷 인지하고 폐기
4) Missing 패킷이 도착할 때까지 out-of-order 패킷들을 버퍼링

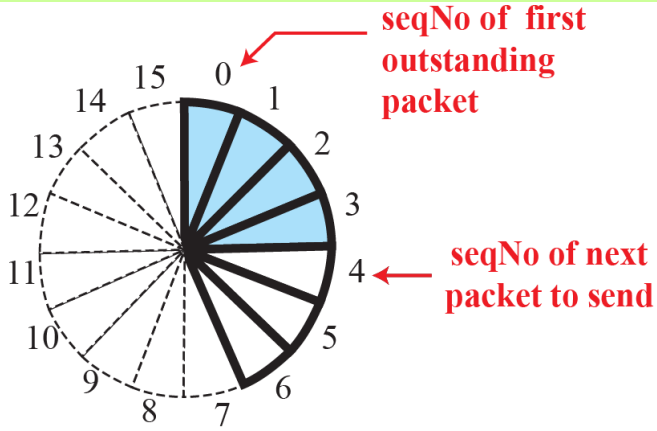
그림 9.11: 전송층에서의 오류 제어(error control)



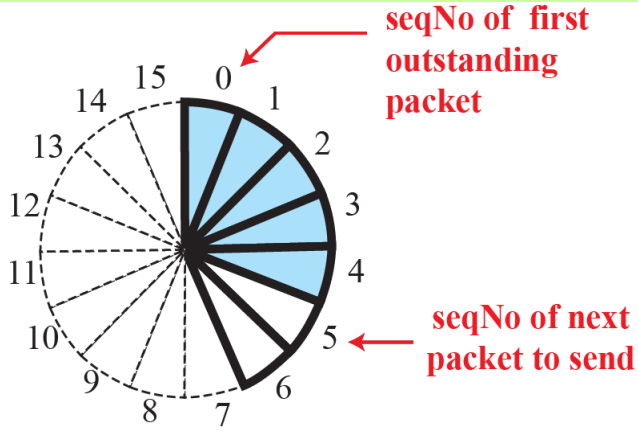
9.1.7 흐름제어와 오류제어의 결합(Combination of Flow and Error Control)

- 흐름 제어는 두 개의 버퍼를 사용해야 한다.
  - ➲ 하나는 발신자 사이트에 있고 다른 하나는 수신자 사이트에 있다.
- 또한 오류 제어를 위해 양쪽에서 시퀀스 및 승인 번호를 사용해야 한다.
- 이 두 가지 요구 사항은 두 개의 번호가 매겨진 버퍼를 사용하는 경우 결합될 수 있다.
  - ➲ 하나는 발신자, 다른 하나는 수신자이다.

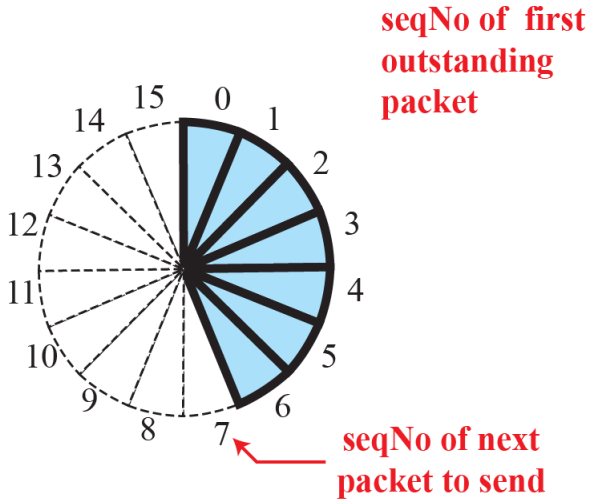
그림 9.12: 원형 형태(circular format)의 슬라이딩 윈도(sliding window)



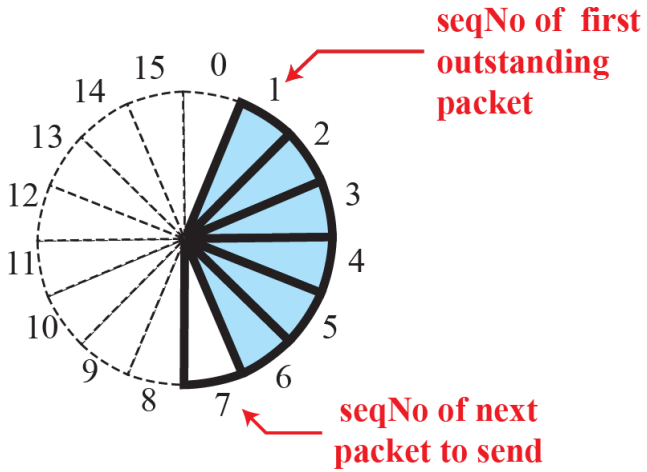
a. Four packets have been sent.



b. Five packets have been sent.



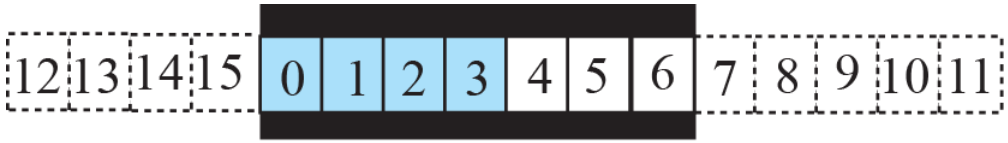
c. Seven packets have been sent; window is full.



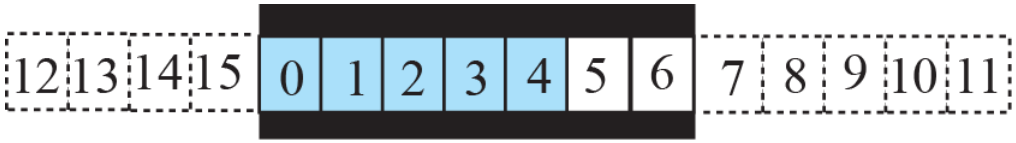
d. Packet 0 has been acknowledged; window slides.

* 시퀀스 번호(sequence number)는 modulo 2^m 사용: 0 – (2^m -1) 개

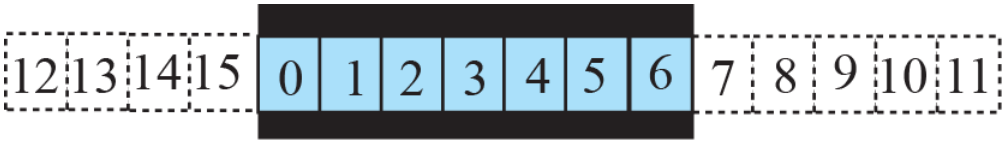
그림 9.13: 선형 형태(linear format)의 슬라이딩 윈도



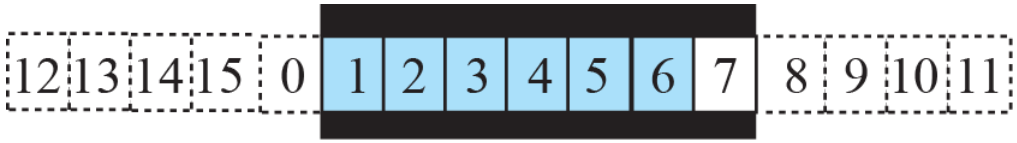
a. Four packets have been sent.



b. Five packets have been sent.



c. Seven packets have been sent;
window is full.



d. Packet 0 has been acknowledged;
window slides.

9.1.9 비연결형과 연결-지향 프로토콜

- 네트워크층 프로토콜과 마찬가지로 전송층 프로토콜도 비연결형(connectionless)과 연결-지향(connection- oriented)의 두 가지 형태의 서비스를 제공한다.
- 그렇지만 전송층에서의 이러한 서비스는 네트워크층의 것과는 본질적으로 다르다.
- 네트워크층에서의 비연결형 서비스는 동일한 메시지에 속하는 서로 다른 데이터그램이 서로 다른 경로를 거칠 수 있다는 것을 의미한다.
- 전송층에서는 패킷의 물리 경로에 대해서는 관여하지 않는다(두 전송층 사이에 논리 연결이 있다고 가정).
- 전송층에서의 비연결형 서비스는 패킷 간의 독립성을 의미하며, 연결 지향은 종속성을 의미한다.
- 두 가지 서비스에 대해서 좀 더 자세히 살펴보도록 하자.

그림 9.14: 비연결형 서비스

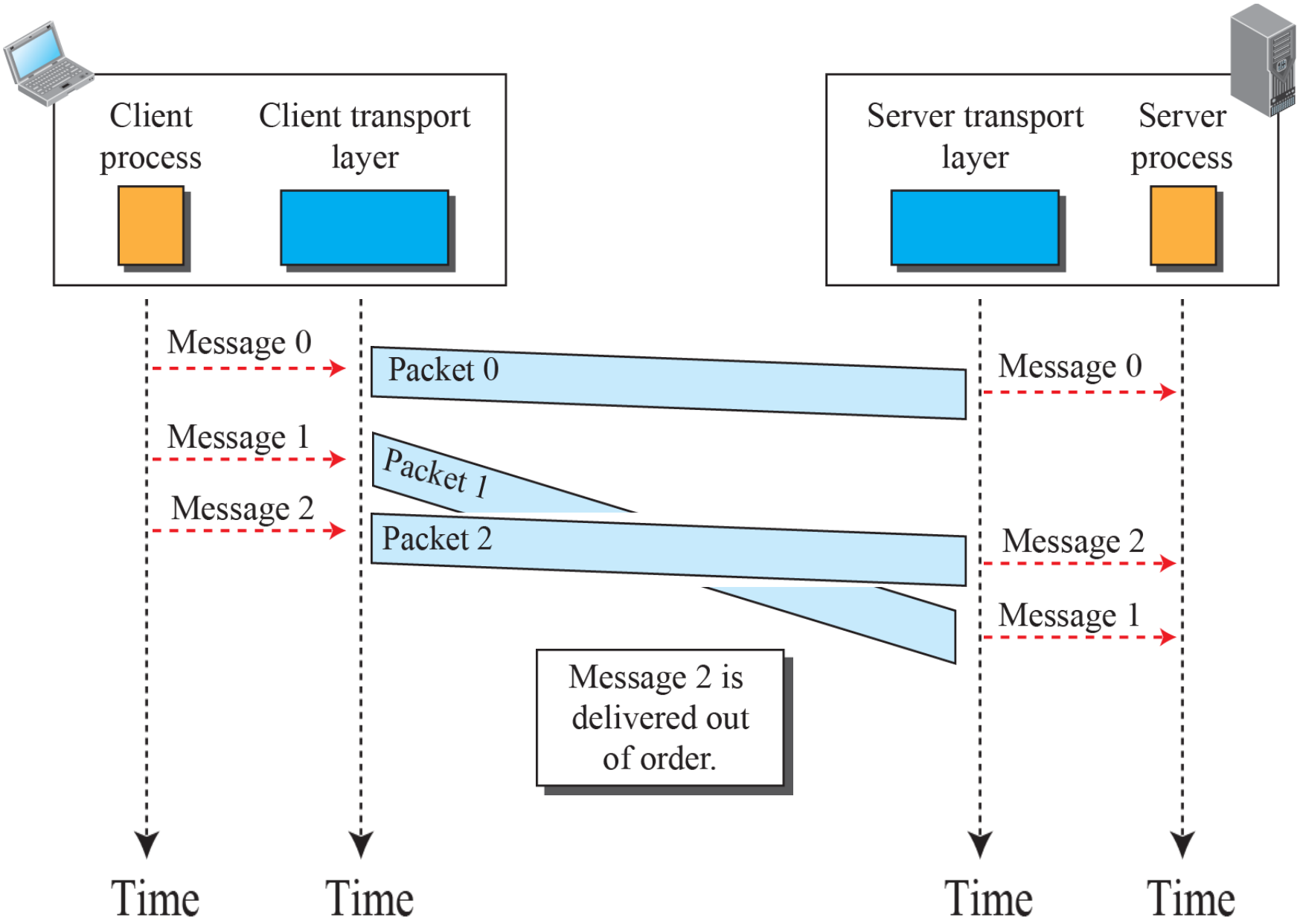
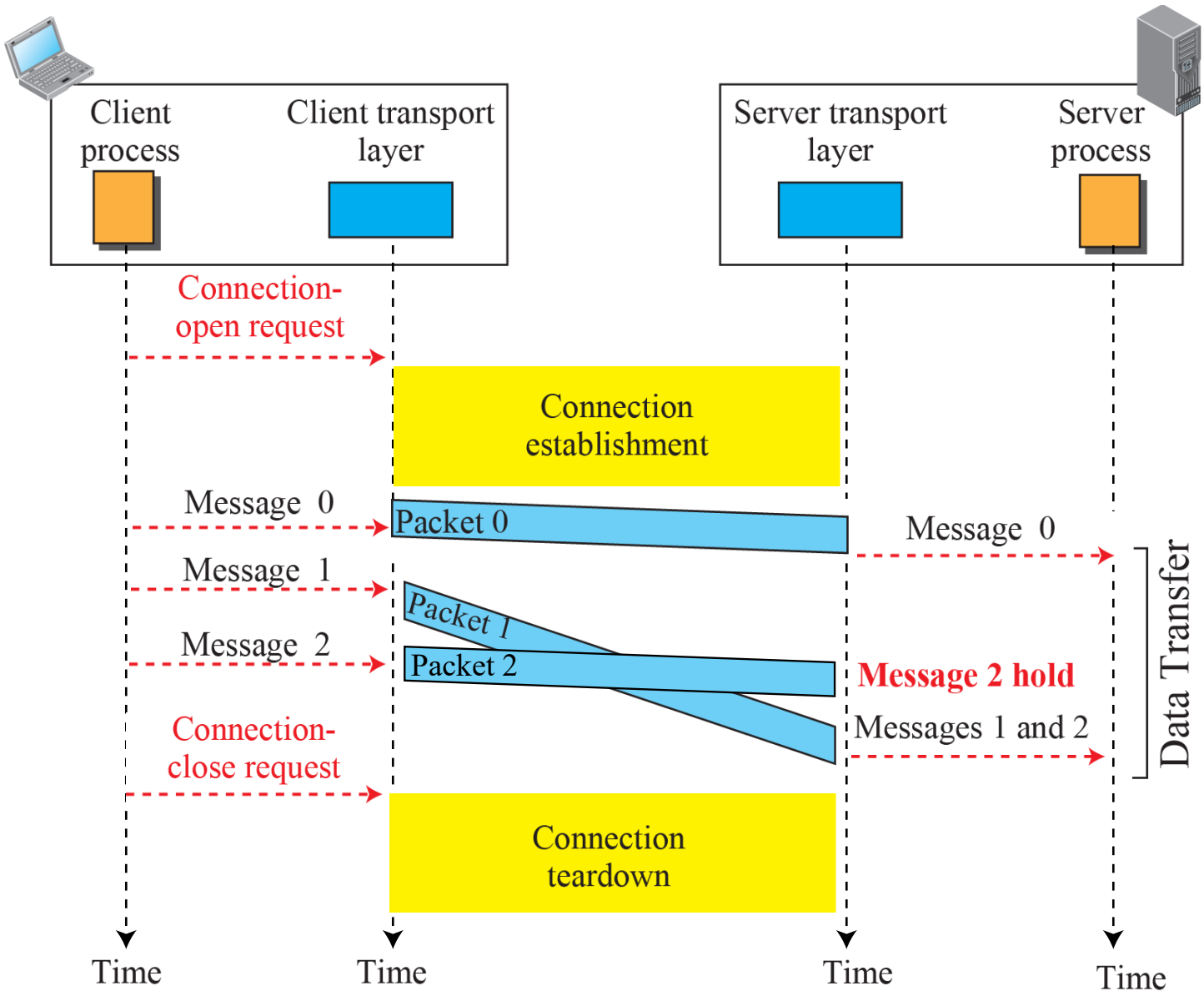


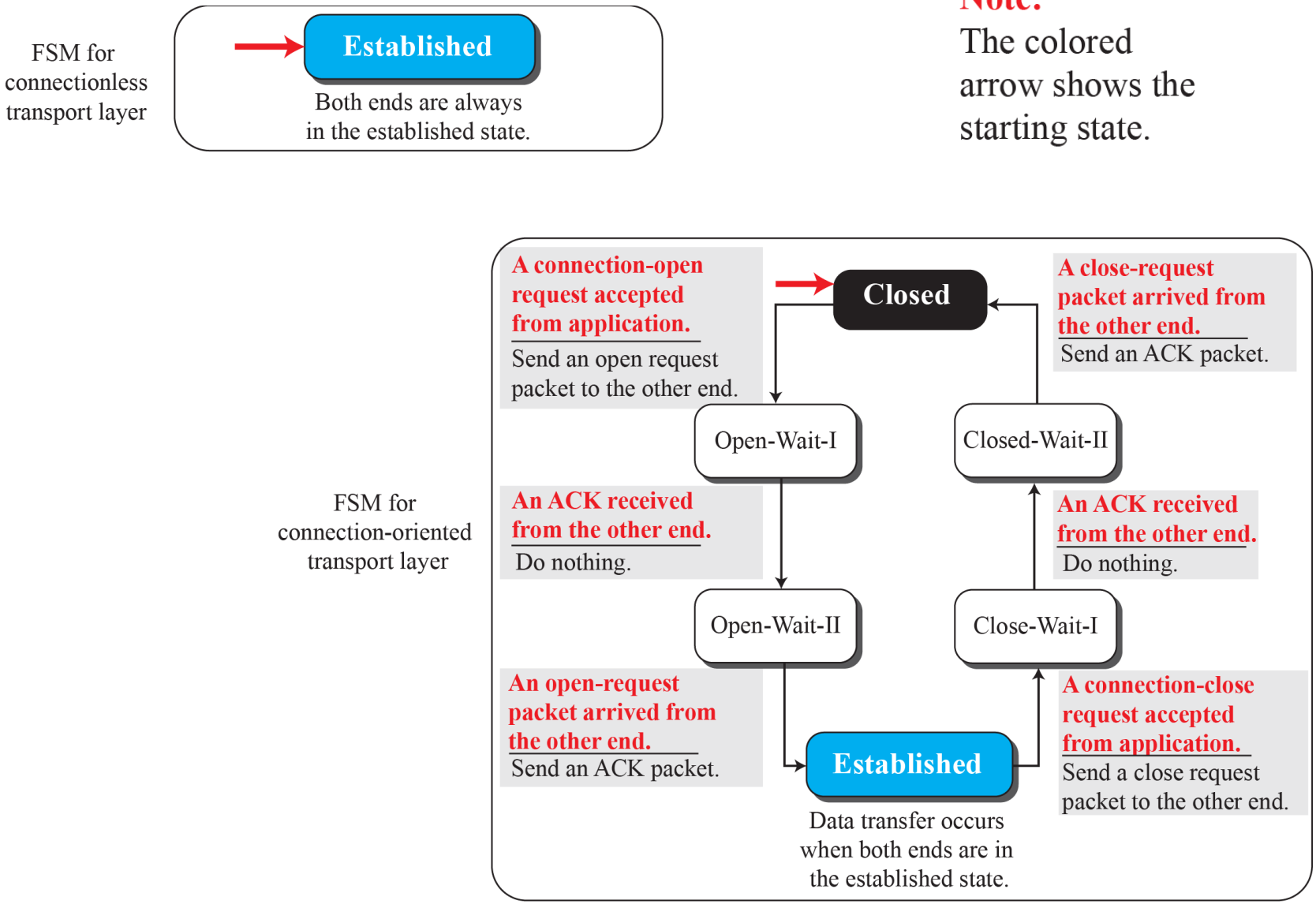
그림 9.15: 연결-지향 서비스



FSM(Finite State Machine)

- 비연결을 제공할 때와 연결 지향 프로토콜을 제공할 때 전송 계층 프로토콜의 동작은 유한 상태 기계(FSM)로 더 잘 표시될 수 있다.
- 그림 9.16은 FSM을 사용하는 전송 계층의 표현을 보여준다. 이 도구를 사용하여 각 전송 계층(발신자 또는 수신자)은 유한 수의 상태를 가진 기계로 학습된다. 기계는 이벤트가 발생할 때까지 항상 상태 중 하나에 있다.

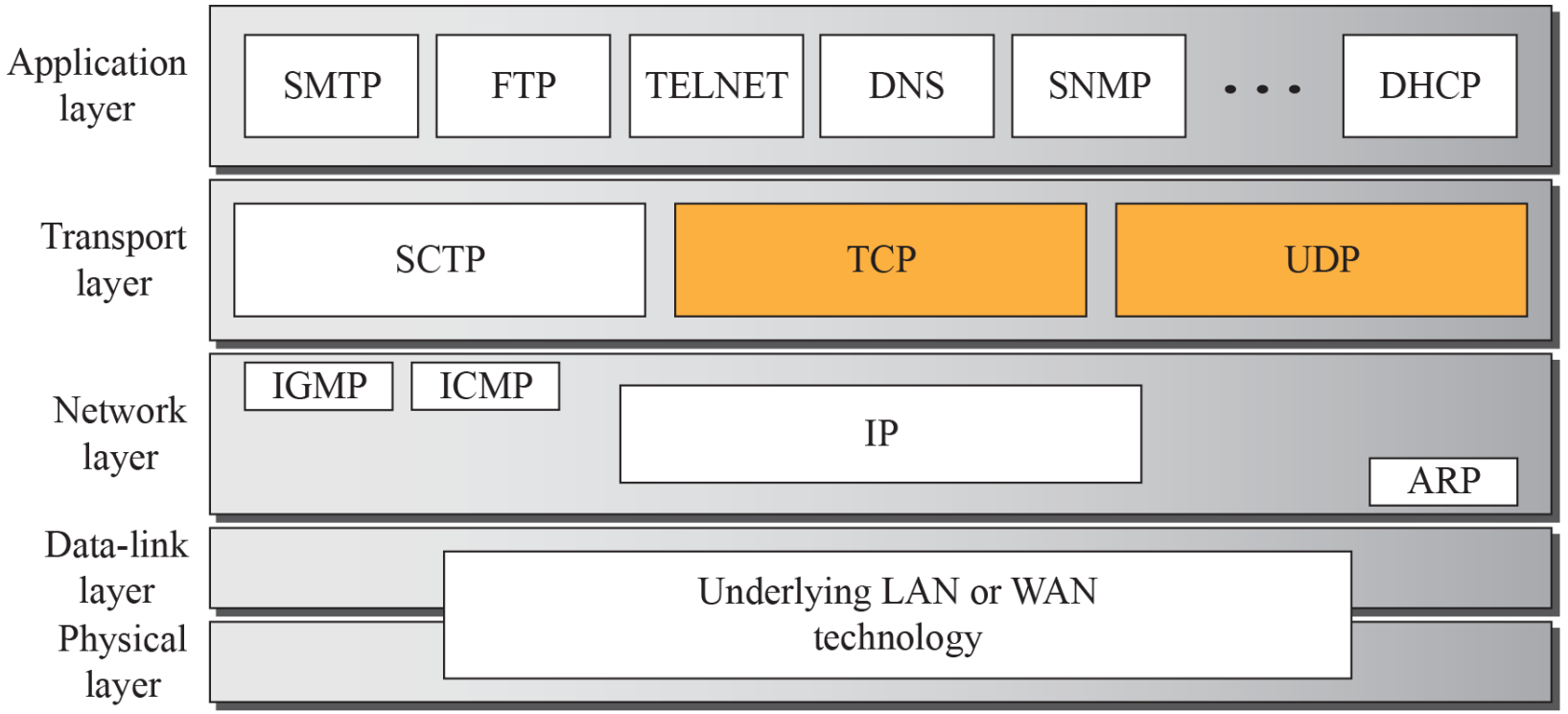
그림 9.16: FSM(finite state machine)으로 표현된 비연결형과 연결-지향 서비스



9.2 전송-계층 프로토콜들

- 이 절에서는 앞 절에서 설명한 일련의 서비스들을 결합함으로써 전송층 프로토콜이 어떻게 만들어지는지를 살펴보고자 한다.
- 이러한 프로토콜들의 동작을 좀더 잘 이해할 수 있도록 하기 위하여 먼저 가장 간단한 프로토콜부터 시작하여 점진적으로 더 복잡한 기능을 추가하도록 한다.
- TCP/IP 프로토콜은 이러한 프로토콜들을 수정하거나 결합한 전송층 프로토콜을 이용한다.

그림 9.1: TCP/IP 프로토콜 그룹에서 전송층 프로토콜의 위치



9.2.1 서비스

- 각 프로토콜은 다른 유형의 서비스를 제공하고 적절하게 사용된다.
- UDP
  - UDP는 오류 제어가 응용층 프로세스에 의해 제공되는 응용에서 단순성과 효율성으로 사용되는 신뢰성 없는 비연결 전송층 프로토콜이다.
- TCP
  - TCP는 신뢰성이 중요한 어떤 응용에 의해 사용될 수 있는 신뢰성 있는 연결-지향 프로토콜이다.
- SCTP
  - SCTP는 UDP와 TCP의 특성을 결합한 새로운 전송층 프로토콜이다.

9.2.2 포트 번호

- 이전 장에서 다루었던 것처럼 전송층 프로토콜은 보통 다수의 책임을 가진다.
- 1개가 프로세스-대-프로세스 통신을 생성하는 것이다.
- 이런 프로토콜은 이것을 수행하기 위해 포트 번호를 사용한다.
- 포트 번호는 전송층에서 종단간 주소를 제공하고 전송 계층에서 마치 네트워크층에서 IP 주소하는 역할과 동일한 다중화와 역다중화를 위해 사용된다.
- 표 9.1은 이 장에서 다룰 모든 세 가지 프로토콜을 위한 일부 공통의 포트 번호를 나타낸 것이다.

표 9.1: UDP와 TCP에 의해 사용되는 일부 잘 알려진 포트들

Port	*Protocol*	*UDP*	*TCP*	*SCTP*	*Description*
7	Echo	√	√	√	Echoes back a received datagram
9	Discard	√	√	√	Discards any datagram that is received
11	Users	√	√	√	Active users
13	Daytime	√	√	√	Returns the date and the time
17	Quote	√	√	√	Returns a quote of the day
19	Chargen	√	√	√	Returns a string of characters
20	FTP-data		√	√	File Transfer Protocol
21	FTP-21		√	√	File Transfer Protocol
23	TELNET		√	√	Terminal Network
25	SMTP		√	√	Simple Mail Transfer Protocol
53	DNS	√	√	√	Domain Name System
67	DHCP	√	√	√	Dynamic Host Configuration Protocol
69	TFTP	√	√	√	Trivial File Transfer Protocol
80	HTTP		√	√	Hypertext Transfer Protocol
111	RPC	√	√	√	Remote Procedure Call
123	NTP	√	√	√	Network Time Protocol
161	SNMP-server	√			Simple Network Management Protocol
162	SNMP-client	√			Simple Network Management Protocol

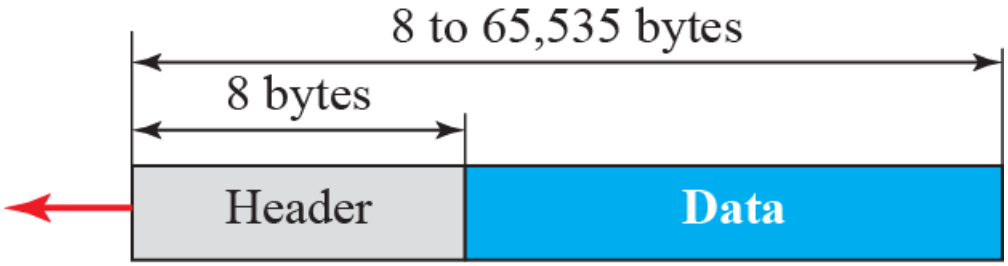
9.3 UDP

- 사용자 데이터그램 프로토콜(UDP, user datagram protocol)은 비연결이고, 신뢰성이 없는 전송 프로토콜이다.
- UDP는 호스트 간 통신 대신에 프로세스 간 통신을 제공하는 것을 제외하고는 IP 서비스에 어떠한 것도 추가하지 않는다.
- UDP가 그렇게 기능이 없다면 왜 프로세스는 이것을 사용하기를 원하는가? 단점들을 가지고 일부 장점을 가져올 수 있기 때문이다.

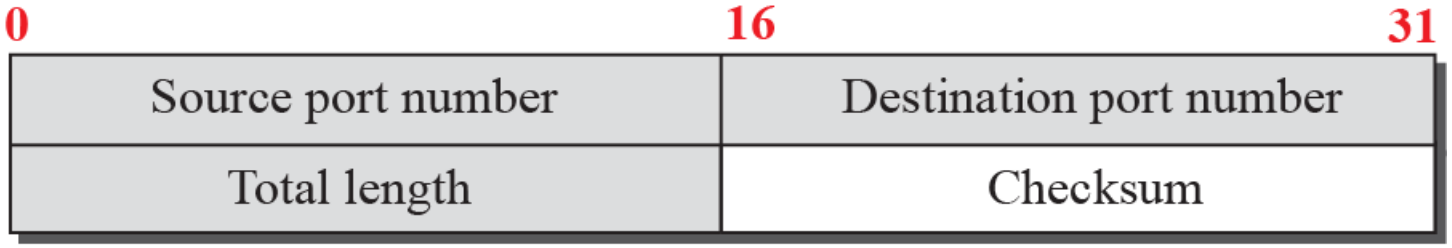
사용자 데이터그램(user datagram)

- 사용자 데이터그램(user datagram)이라고 부르는 UDP 패킷은 각각 2바이트(16비트)인 4개의 필드로 만들어진 고정된 크기의 8바이트 헤더를 가지고 있다.
- 그림 9.2는 사용자 데이터그램의 형식을 보여주고 있다.
- 처음 두 필드는 근원지와 목적지 포트 번호를 정의한다.
- 세 번째 필드는 헤더에 데이터를 더한 사용자 데이터그램의 전체 길이를 정의된다. 16비트는 전체 길이 0부터 65,536바이트를 정의할 수 있다. 그러나 하나의 UDP 사용자 데이터그램은 65,535바이트의 총 길이를 가지는 IP 데이터그램에 저장되기 때문에 사용자 데이터그림의 총 길이는 더 작아야 한다.
- 마지막 필드는 선택적 검사합(뒤에 설명)을 운반한다.

그림 9.18: 사용자 데이터그램 패킷 형식



a. UDP user datagram



b. Header format

예제 9.2

다음은 16진 형식의 UDP 헤더의 내용이다.

CB84000D001C001C

a. 근원지 포트 번호는 얼마인가?
b. 목적지 포트 번호는 얼마인가?
c. 사용자 데이터그램의 전체 길이는 얼마인가?
d. 데이터의 길이는 얼마인가?
e. 클라이언트에서 서버로 향하는 패킷인가 혹은 반대인가?
f. 클라이언트 프로세스는 무엇인가?

예제 9.2 (계속)

Solution

a. 근원지 포트 번호는 첫 번째 4자리의 16진수 $(CB84)_{16}$ 으로, 근원지 포트 번호는 52100을 의미 한다.

b. 목적지 포트 번호는 두 번째 4자리의 16진수 $(000D)_{16}$ 으로, 목적지 포트 번호는 13을 의미한다.

c. 세 번째 4자리의 16진수 $(001C)_{16}$로 전체 UDP 패킷의 길이는 28바이트로 정의된다.

d. 데이터 길이는 전체 패킷 길이에서 헤더의 길이를 뺀 것으로, 28 − 8=20바이트이다.

e. 목적지 포트 번호는 13(잘 알려진 포트)이기 때문에, 패킷은 클라이언트에서 서버로 향한다는 것을 의미한다.

f. 클라이언트 프로세스는 일시(Daytime)이다(표 9.1 참조).

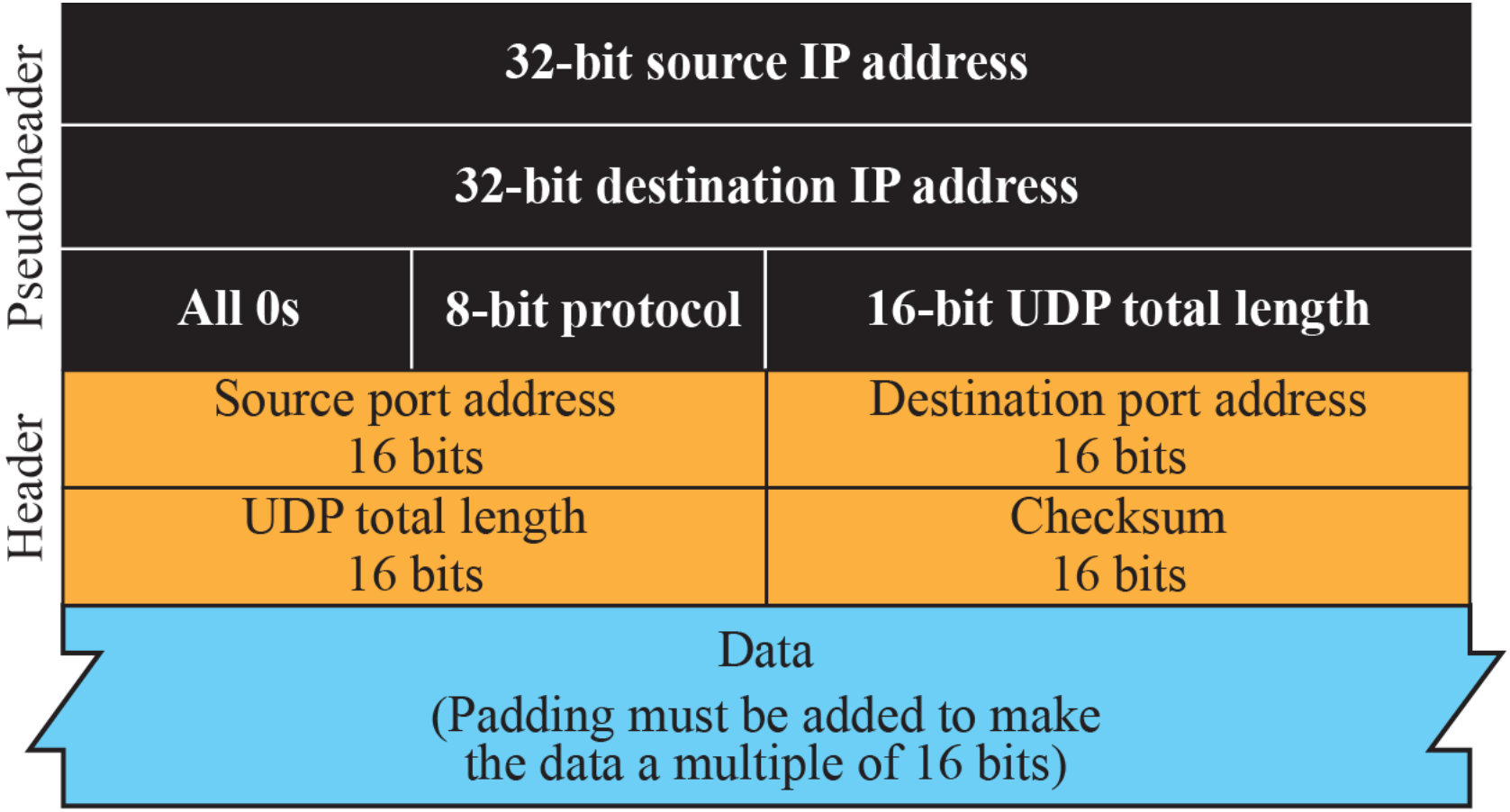
9.3.1 UDP 서비스

- 앞에서 전송층 프로토콜에 의해 제공되는 일반적인 서비스를 다루었다.
- 이 절에서는 UDP에 의해 제공되는 일반적인 서비스에 대해 다룬다.
  - **프로세스-대-프로세스 통신:** UDP는 IP 주소와 포트 번호의 조합인 소켓 주소를 사용하여 프로세스 간 통신을 제공한다.
  - **비연결 서비스:** UDP에 의해 전송된 각 사용자 데이터그램이 독립적인 데이터그램임을 의미한다. 서로 다른 사용자 데이터그램이 동일한 소스 프로세스에서 나와 동일한 대상 프로그램으로 이동하더라도 관계가 없다. 사용자 데이터그램에는 시퀀스번호가 없다.
  - **흐름 제어:** UDP흐름 제어 기능이 없으므로 Window 메커니즘이 없다. 수신자의 큐(queue)는 수신 메시지로 넘칠 수 있다. 흐름 제어 기능이 없다는 것은 UDP를 사용하는 프로세스가 필요한 경우 흐름 서비스를 별도로 제공해야 함을 의미한다.
  - **오류 제어:** 검사합(checksum)을 제외하고 UDP에는 오류 제어 메커니즘이 없다. 즉, 보낸 사람은 메시지가 손실되었거나 중복되었는지 알 수 없다. 수신자가 검사합을 통해 오류를 감지하면 사용자 데이터그램은 자동으로 폐기된다. 오류 제어 기능이 없다는 것은 UDP를 사용하는 프로세스가 필요한 경우 이 서비스를 별도로 제공해야 함을 의미한다.

9.3.1 UDP 서비스

- **검사합** : 의사 헤더를 포함하여 계산
- **혼잡제어** : UDP는 비연결형 프로토콜이므로 혼잡 제어를 제공하지 않는다. UDP는 전송된 패킷이 작고 산발적이며 네트워크에 혼잡을 만들 수 없다고 가정한다. UDP가 오디오 및 비디오의 대화식 실시간 전송에 사용되는 오늘날 이 가정은 사실일 수도 있고 그렇지 않을 수도 있다.
- **캡슐화와 캡슐제거** (encapsulation & decapsulation)
  한 프로세스에서 다른 프로세스로 메시지를 보내기 위해 UDP 프로토콜은 메시지를 캡슐화하고 캡슐을 제거한다.
- **큐잉**(queuing): 클라이언트 사이트에서 프로세스가 시작되면 운영 체제(OS)에 포트 번호를 요청한다. 일부 구현(implementation)은 각 프로세스와 연결된 수신(incoming)및 발신(outgoing) 대기열(queue)들을 모두 생성한다. 어떤 구현은 각 프로세스와 연결된 수신 대기열만 생성한다.
- **다중화와 역다중화**: TCP/IP 프로토콜 제품군을 실행하는 호스트에는 UDP가 하나만 있지만 UDP 서비스를 사용하려는 프로세스는 여러 개일 수 있다. 이러한 상황을 처리하기 위해 UDP는 다중화 및 역다중화한다.

그림 9.19: 검사합 계산을 위한 의사헤더



Example 9.3

다음 각각의 상황에서 검사합을 위해 어떤 값이 전송되는가?

a. 송신기가 검사합을 포함하지 않기로 결정했다.
b. 송신기가 검사합을 포함하기로 결정했으나, 합의 값이 모두 1이다.
c. 송신기가 검사합을 포함하기로 결정했으나, 합의 값이 모두 0이다.

예제 9.3 (계속)

Solution

a.전송된 검사합 필드의 값이 모두 0이며 검사합이 계산되지 않는다는 것을 보여준다.

b.송신기가 합에 보수(complement)를 취하고, 그 결과는 모두 0이다. 송신기는 전송 전에 다시 결과에 보수를 취한다. 검사합을 위해 전송된 값은 모두 1이다. 두 번째 보수 동작은 a번의 경우처럼 혼란을 피하기 위해 필요하다.

c.이런 상황은 합 계산에 포함된 모든 항의 값이 모두 0이라는 것을 의미하기 때문에 결코 발생하지 않는다. 0은 불가능하며 의사 헤더에서 일부 필드는 0이 아닌 값을 가진다.

9.3.2 UDP 응용

- 비록 UDP는 신뢰성 있는 전송층 프로토콜을 위해 이전에 언급했던 기준을 거의 하나도 만족하지 못하지만, UDP는 일부 응용에서 선호된다.
- 이유는 일부 서비스는 받아들일 수 없거나 혹은 선호되지 않는 부수적 효과를 가질 수 있기 때문이다.
- 때때로 응용 설계자는 최적에 맞추기 위해 타협할 필요가 있다.
- 예를 들면, 우리는 일상생활에서 모두 운송회사의 소화물이 1일 배송이 3일 배송보다 매우 비싸다는 것을 알고 있다. 비록 화물 배달에서 고속과 저가가 모두 원하는 형태이긴 하지만, 이들 각각은 충돌한다. 이때 최적의 선택이 필요하다.

UDP 사양

- 비연결 서비스
- 오류 제어의 결여
- 혼잡 제어의 결여

예제 9.4

- DNS(Domain Name System)와 같은 클라이언트-서버 응용은 클라이언트가 짧은 요구를 서버에게 전송하고 서버로부터 짧은 응답을 받아야 할 필요성이 있기 때문에 UDP 서비스를 사용한다.
- 요구와 응답은 각각 1개의 사용자 데이터그램에 적합하다. 한 방향으로 단지 1개의 메시지가 교환되기 때문에, 비연결 특성이 이슈가 아니다.
- 클라이언트 혹은 서버는 메시지가 순서에 벗어나 전달되는 것을 염려하지 않는다.

예제 9.5

- 전자우편에 사용하는 SMTP(Simple Mail Transfer Protocol)와 같은 클라이언트-서버 응용은 사용자가 멀티미디어(사진, 음성, 혹은 영상)를 포함하는 긴 전자우편 메시지를 전송하기 때문에 UDP 서비스를 사용할 수 없다.
- 만약 응용이 UDP를 사용하면 메시지는 한개의 사용자 데이터그램에 포함 할 수 없어, 메시지는 응용에 의해 다른 여러 개의 사용자 데이터그램으로 분리되어야 한다. 이런 경우, 비연결 서비스가 문제를 발생시킬 수 있다.
- 사용자 데이터그램은 순서에 벗어나 도착할 수 있고, 수신 응용에 순서에 벗어나 전달될 수 있다. 이것은 비연결 서비스가 긴 메시지를 전송하는 응용 프로그램을 위해서는 단점을 가진다는 것을 의미한다.
- SMTP에서 메시지를 전송하면, 빠른 응답을 기대할 수 없다(때때로 어떠한 응답도 요구되지 않는다). 이것은 연결지향 서비스에 고유한 추가지연 특성이 SMTP에서는 별 문제가 않된다는 것을 의미한다.

예제 9.6

- 인터넷으로부터 매우 긴 문자 파일을 다운로드한다고 가정하자. 당연히 신뢰성 있는 서비스를 제공하는 전송층을 사용할 필요가 있다.
- 우리는 파일을 열 때 파일 일부가 분실되거나 혹은 오류가 있는 것을 원하지 않는다.
- 부분적으로 전달되는 세그먼드간에 발생하 지연은 큰 염려가 되지 않는다. 파일을 열어 보기 전에 전체 파일이 완성될 때까지 기다린다. 이 경우 UDP는 적합한 전송층이 아니다.

예제 9.7

- 스카이프(Skype)와 같은 실시간 상호동작 응용을 사용한다고 가정하자. 음성과 영상은 프레임으로 나누어져서 차례로 전송된다. 만약 전송층에 오류가 있거나 분실된 프레임을 재전송한다고 가정하면, 전체 전송의 동기는 잃어버릴 수 있다.
- 시청자는 갑자기 빈 스크린을 보게 되고, 두 번째 전송이 도착할 때까지 기다려야 한다. 이것은 참을 수가 없게 된다.
- 그러나 만약 각 스크린의 작은 부분이 단일 사용자 데이터그램(UDP)을 이용하여 전송되면, 수신 UDP는 쉽게 오류가 있거나 분실된 패킷을 무시할 수 있고 응용 프로그램에 나머지를 전달한다. 스크린의 해당 부분은 아주 짧은 시간 동안 비어 있으나, 대부분의 참여자는 인지하지 못할 것이다.

일반적인 응용

- UDP는 흐름 및 오류 제어를 하지 않는 간단한 요청-응답 통신을 요구하는 프로세스에 적당하다.
- UDP는 별도의 흐름 및 오류 제어 기법을 가진 프로세스에 적당하다.
- UDP는 멀티캐스팅을 위한 적당한 전송 프로토콜이다.
- UDP는 SNMP와 같은 관리 프로세스들을 위하여 사용된다.
- UDP는 라우팅 정보 프로토콜(RIP)과 같은 경로 갱신 프로토콜을 위하여 사용된다.
- UDP는 보통 수신 메시지의 영역 사이에 변동성이 많은 지연(uneven delay)을 인내할 수 없는 상호작용적(interactive) 실시간 응용에 사용된다.

9.4 TCP

- 전송 제어 프로토콜(TCP, transmission control protocol)은 연결-지향, 신뢰성 있는 프로토콜이다.

- TCP는 연결지향 서비스를 제공하기 위하여 분명하게 연결 설정, 데이터 전송, 연결 해제 단계를 정의한다.

9.4.1 TCP 서비스

- TCP에 대하여 자세하게 언급하기 전에, 응용층의 프로세스들에게 TCP가 제공하는 서비스들에 대하여 설명해보자.
  - ➲프로세스-대-프로세스 통신: 포트번호를 사용한 프로세스간 통신
  - ➲스트림 전송 서비스 : 송신 및 수신 버퍼, 세그먼트
  - ➲전이중 통신 : 동시에 양방향 통신
  - ➲다중화와 역다중화: 송신자 Multiplexing, 수신자는 Demultiplexing
  - ➲연결지향 서비스 : 전송전 connection설정
  - ➲신뢰성 있는 서비스 : ack 메커니즘 사용

그림 9.20: 스트림 전송

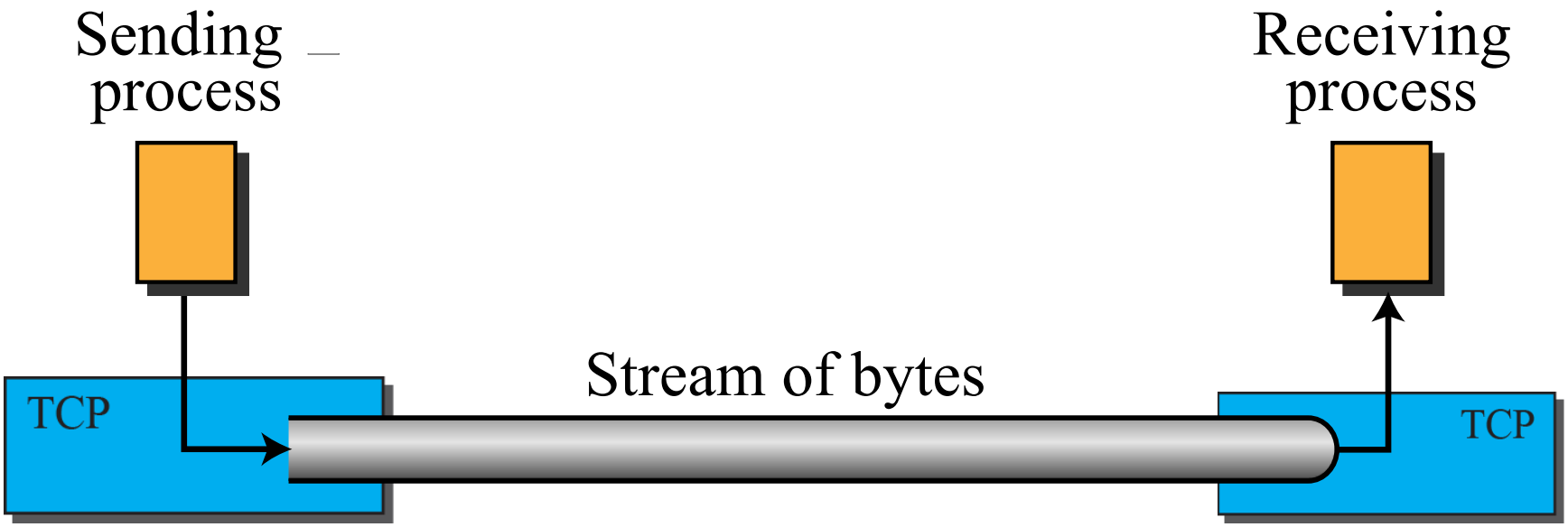


그림 9.21: 송신 및 수신 버퍼

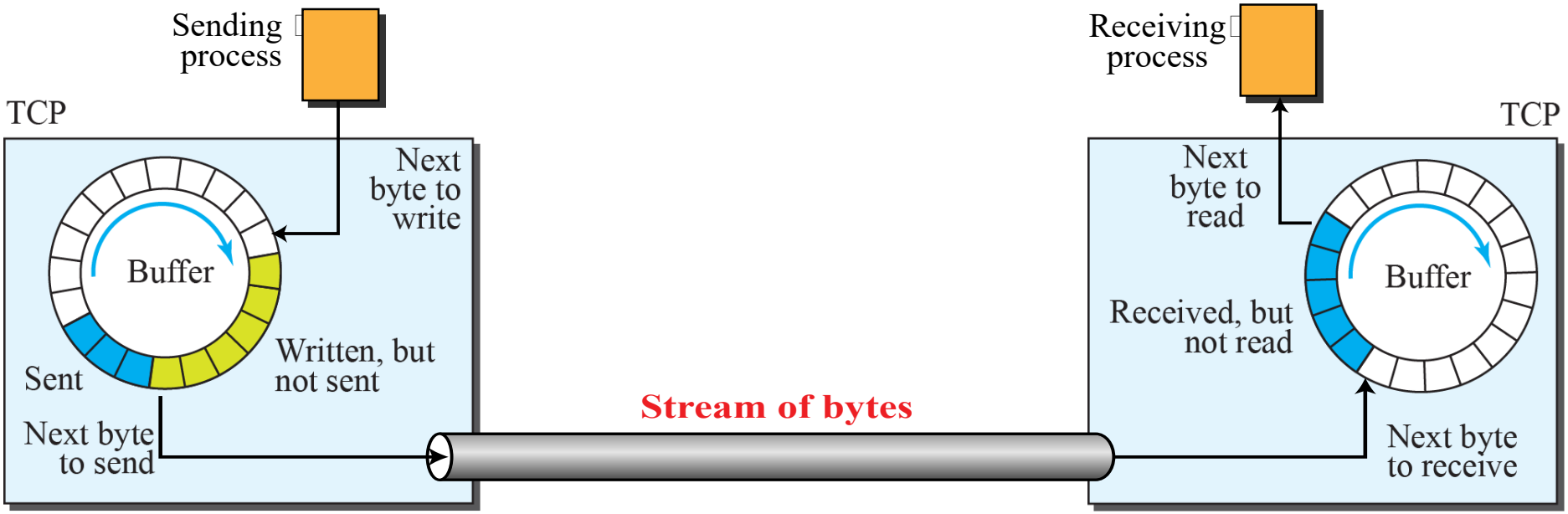
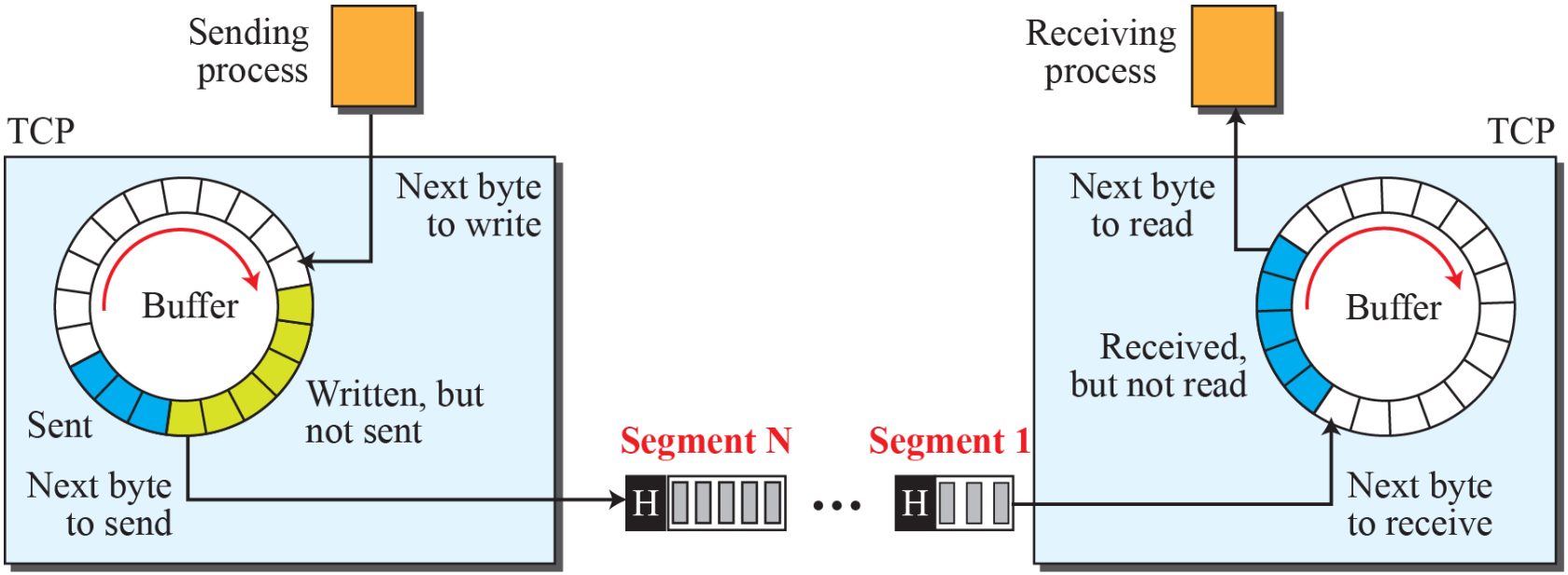


그림 9.22: TCP 세그먼트



9.4.2 TCP 특징

■ 이전 절에서 언급했던 서비스를 제공하기 위하여, TCP는 여러 특징을 가지고 있으며, 여기에서는 짧게 요약하며 나중에 자세히 다룰 것이다.

■ 번호 부여 시스템

- ➲ 바이트 번호
- ➲ 순서 번호
- ➲ 확인 응답 번호

예제 9.8

TCP 연결이 5000바이트의 파일을 전송하고 있다고 가정하라. 첫 번째 바이트는 10001로 번호 가 부여되어 있다. 데이터가 1000바이트씩을 운반하는 5개의 세그먼트로 전송된다면 각 세그먼트에 대한 순서번호는 무엇인가?

Solution

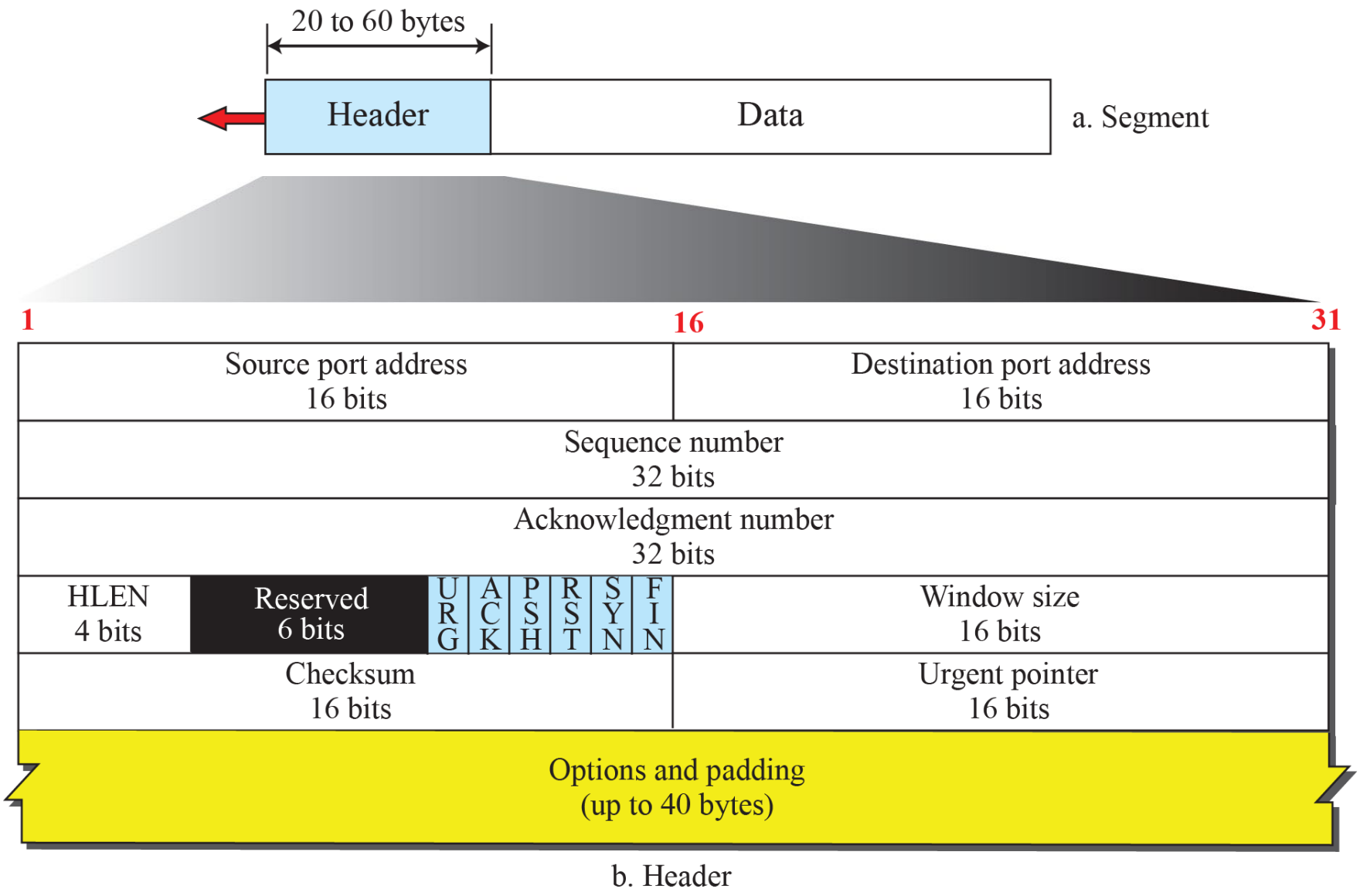
각 세그먼트에 대한 순서번호는 다음과 같다.

Segment 1	→	Sequence Number:	10,001	**Range:**	10,001	to	11,000
Segment 2	→	Sequence Number:	11,001	**Range:**	11,001	to	12,000
Segment 3	→	Sequence Number:	12,001	**Range:**	12,001	to	13,000
Segment 4	→	Sequence Number:	13,001	**Range:**	13,001	to	14,000
Segment 5	→	Sequence Number:	14,001	**Range:**	14,001	to	15,000

9.4.3 세그먼트

- TCP를 좀 더 자세히 설명하기 전에 TCP 패킷 자체를 설명해보자. TCP에서는 패킷을 세그먼트(segment)라고 한다.

그림 9.23: TCP 세그먼트 형식



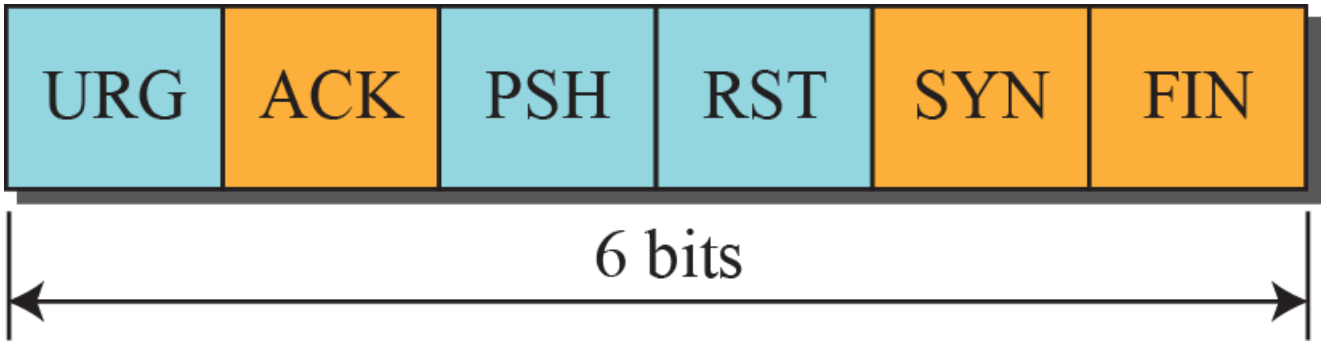
TCP 헤더(1)

- 근원지 포트 주소 : 16 비트
  - 세그먼트를 송신하는 호스트에 있는 응용 프로그램의 포트 번호를 정의하는 필드
- 목적지 포트 주소 : 16 비트
  - 세그먼트를 수신하는 호스트에 있는 응용 프로그램의 포트 번호를 정의하는 필드
- 순서 번호 : 32 비트
  - 세그먼트에 포함된 첫 번째 데이터 바이트에 할당된 번호를 정의한다. TCP는 스트림 전송 프로토콜로 연결성을 확신하기 위해서 전송되는 각 바이트에는 번호가 부여된다. 순서번호는 이 데이터 순서의 어느 바이트가 세그먼트에서 첫 번째 바이트를 의미하는지를 목적지에 알려준다. 연결을 설정하는 동안 양 측은 각각 초기 순서 번호(ISN, initial sequence number)를 생성하기 위하여 랜덤 번호 생성기를 사용하며, 각 방향에서 보통 다른 번호가 생성된다.
- 확인응답 번호(acknowledgement number) : 32 비트
  - 세그먼트의 송신자가 다른 쪽으로부터 받기를 기대하는 바이트 번호를 정의한다. 세그먼트 수신자가 상대방으로부터 바이트 번호 x를 성공적으로 수신하였다면 x+1이 확인응답 번호가 된다.
- 헤더 길이 : 4 비트
  - TCP 헤더를 4바이트 단위의 개수로 나타낸 것. 헤더의 길이는 20에서 60바이트가 될 수 있으며, 따라서 이 필드의 값은 5(5 × 4 = 20)에서 15(15 × 4 = 60)사이의 값이 됨.

TCP 헤더(2)

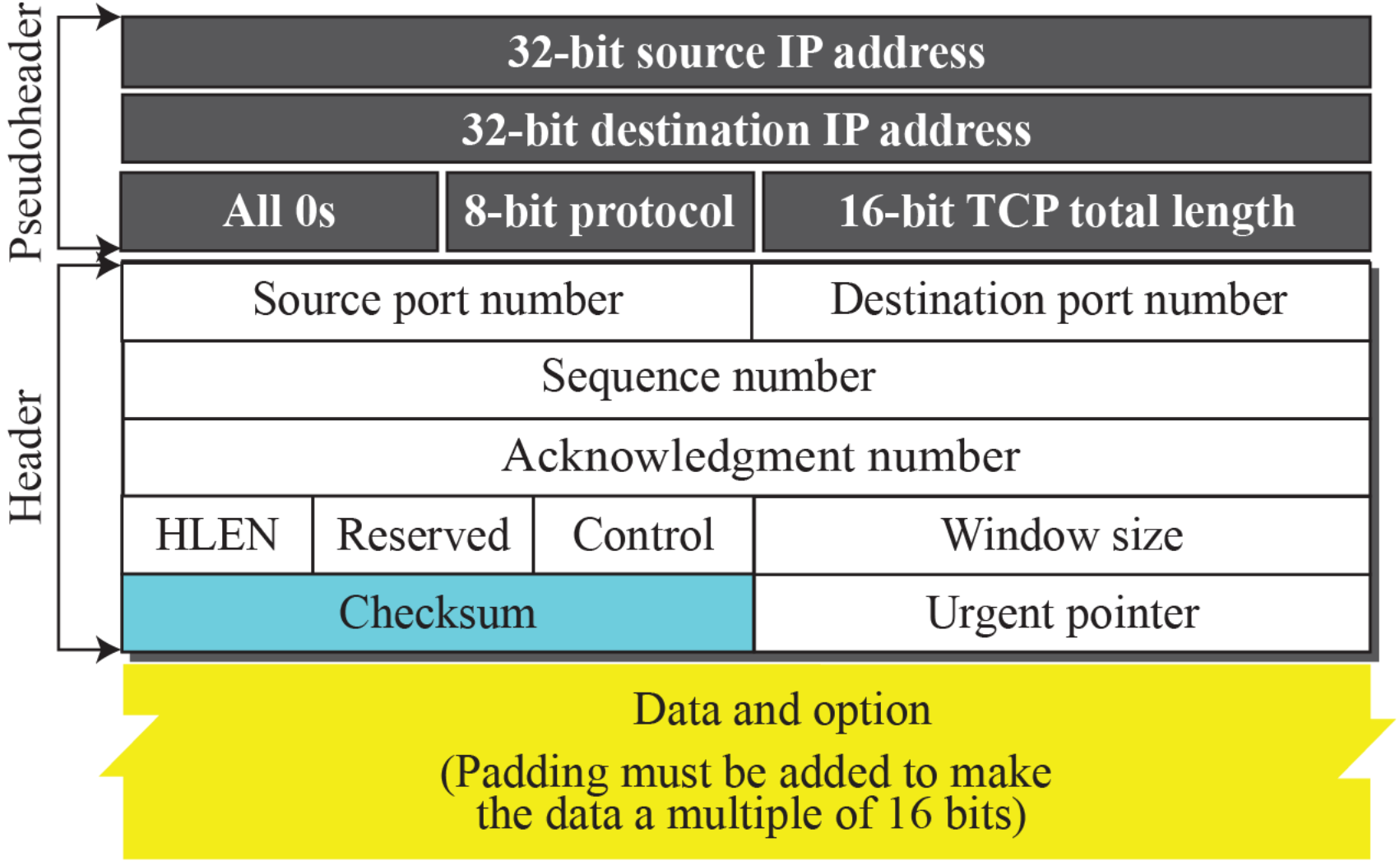
- 제어(control) : 6 비트
  - 여섯 개의 다른 제어 비트 또는 플래그 비트를 정의한다.
- 윈도 크기(window size) : 16 비트
  - 상대방이 반드시 유지해야 하는 바이트 단위의 윈도 크기이다. 윈도 최대 크기가 65,536 바이트임을 의미한다. 이 값은 보통 수신 윈도(rwnd)로 언급이 되고 수신자에 의해서 결정된다.
- 검사합 : 16비트
  - TCP를 위한 검사합의 계산은 이전 절에서 설명했던 UDP와 동일한 절차를 따른다.
  - 그러나 UDP 데이터그램에 있는 검사합의 포함은 선택사항이지만 TCP를 위한 검사합은 필수사항이다. 동일한 목적을 수행하는 동일한 의사헤더가 세그먼트에 부가된다.
- 긴급 지시자(urgent pointer) : 16비트
  - 긴급 플래그(urgent flag) 값이 설정이 되었을 때만 유효하며 세그먼트가 긴급 데이터를 포함하고 있을 때 사용된다.
  - 세그먼트의 데이터 부분에서 마지막 긴급 바이트의 번호(the number of the last urgent byte)를 구하기 위하여 순서번호와 함께 urgent pointer가 필요하다.
- 선택항목
  - TCP 헤더는 40바이트까지 옵션 정보를 가질 수 있다.

그림 9.8: 제어 필드



URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection

그림 9.9: TCP 데이터그램에 추가된 의사헤더



* 검사합(checksum) 계산을 위해 필요한 TCP 데이터그램에 추가된 의사헤더

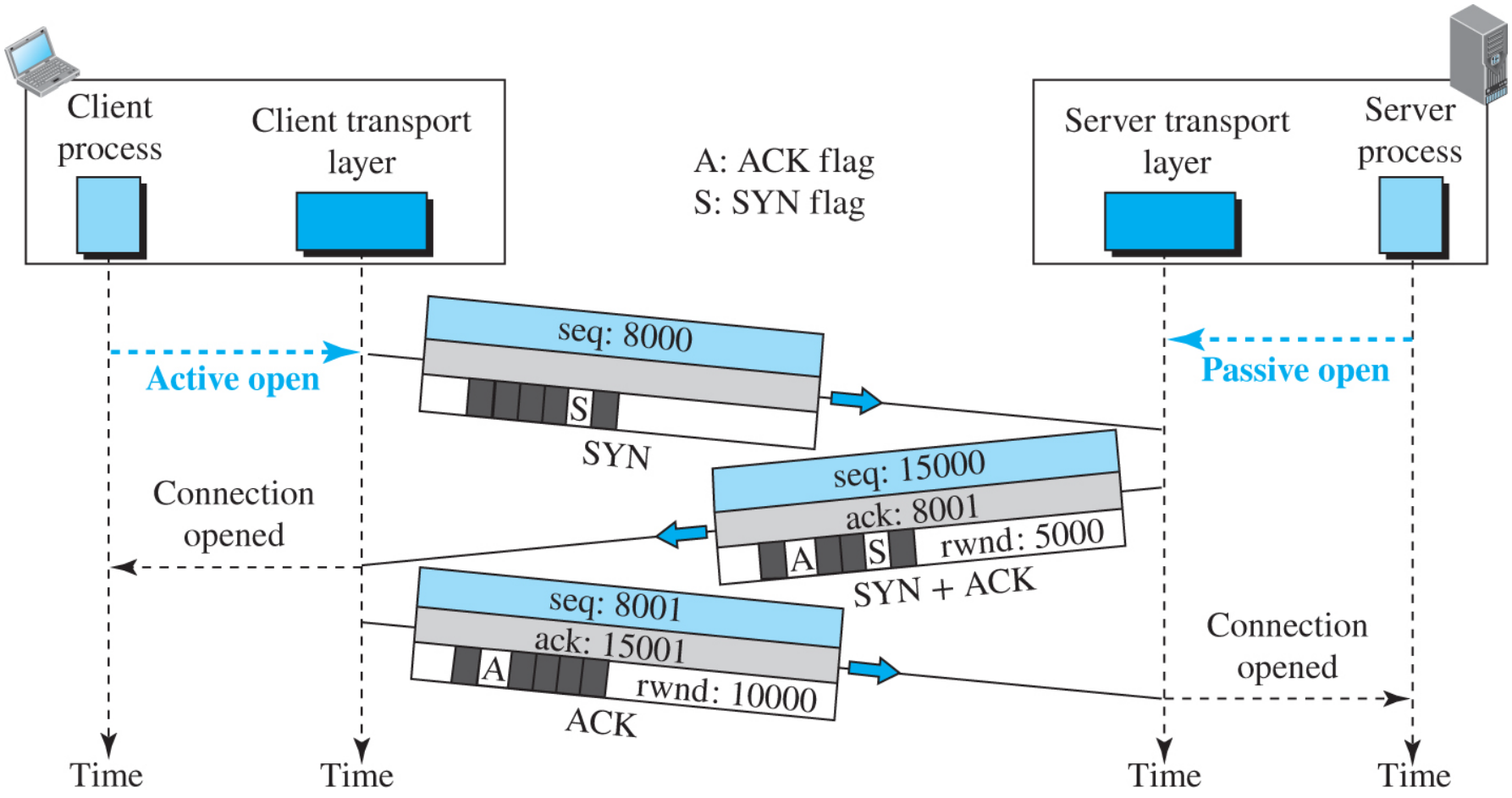
9.4.4 TCP 연결

- TCP는 연결-지향 전송 프로토콜로서 근원지와 목적지 사이에 가상 경로를 설정한다.
- 한 메시지에 속하는 모든 세그먼트들은 이 가상 경로를 통해서 전송된다.
- 전체 메시지를 위하여 단일 가상 경로를 사용하는 것은 훼손(damaged)되거나 손실된(lost) 프레임의 재전송 뿐만 아니라 확인응답 프로세스를 가능하게 한다.
- 비연결 지향성 프로토콜인 IP 서비스들을 사용하는 TCP가 어떻게 연결지향성인지 의문일 것이다.
- TCP 연결은 물리적이지 않고 바로 가상이라는 점이다. TCP는 상위 층에서 동작한다.
- TCP는 세그먼트들을 수신자에게 전달하기 위하여 IP 서비스를 사용하지만 연결 자체를 제어한다.

9.4.4 TCP 연결: 캡슐화(Encapsulation)

- TCP 세그먼트는 응용 프로그램 계층에서 받은 데이터를 캡슐화한다.
- TCP 세그먼트는 IP 데이터그램으로 캡슐화되며, 다음으로 데이터 링크 계층의 프레임에 캡슐화된다.

그림 9.26: 세-방향(Three-way) 핸드셰이크(handshaking)연결 설정



9.4.4 TCP 연결: 데이터 전송

- 연결이 설정되면 양방향 데이터 전송이 수행될 수 있다.
- 클라이언트와 서버는 양방향으로 데이터와 승인정보를 보낼 수 있다.
- 승인(acknowledgment)과 동일한 방향으로 이동하는 데이터가 동일한 세그먼트에 전달된다는 것을 아는 것으로 충분하다.
- 승인은 데이터와 함께 피기백(piggyback)된다.

그림 9.27: 데이터 전송

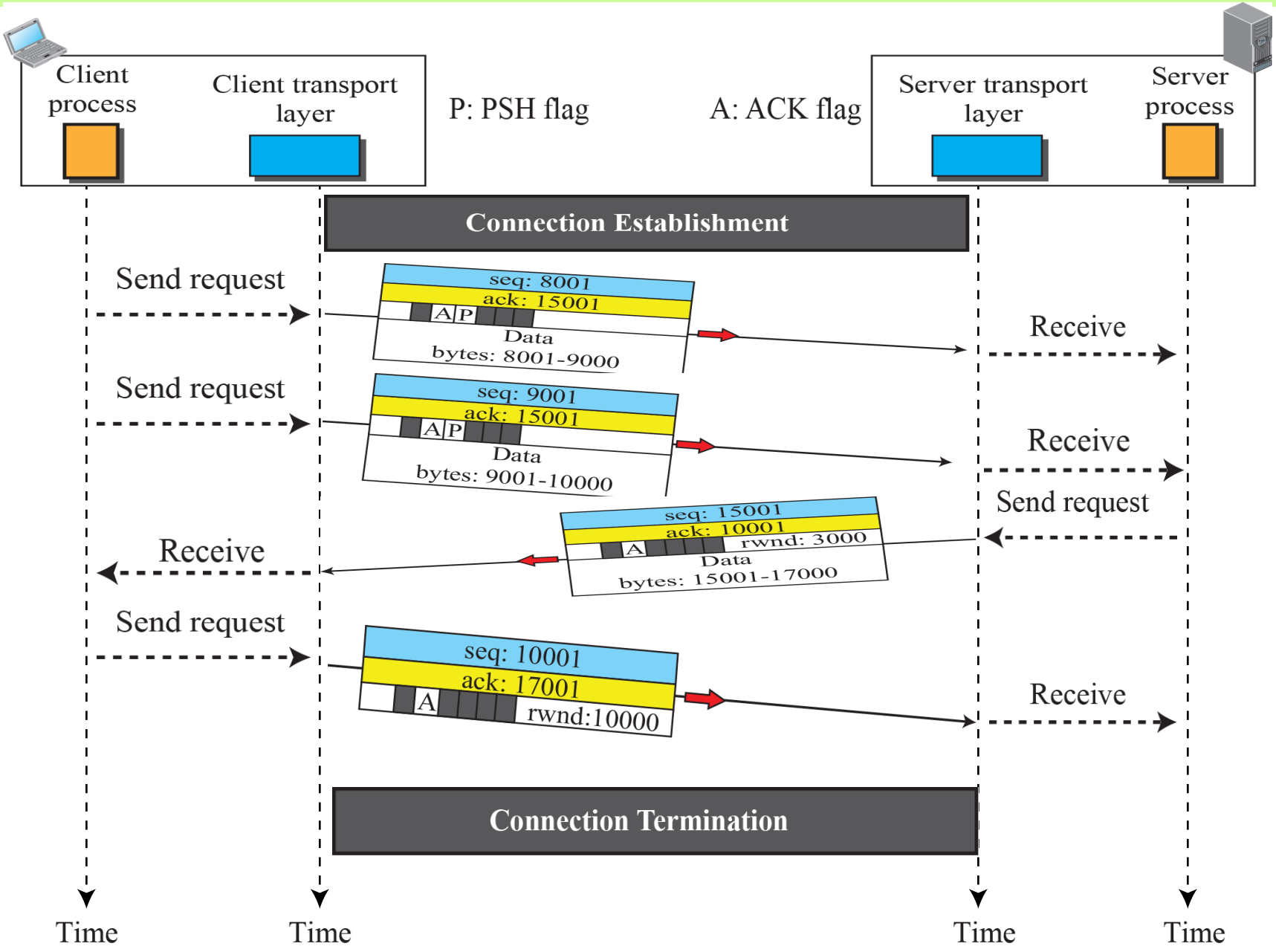

Client process
Client transport layer
P: PSH flag
A: ACK flag
Server transport layer
Server process
Connection Establishment
Send request
seq: 8001
ack: 15001
A P
Data
bytes: 8001-9000
Receive
Send request
seq: 9001
ack: 15001
A P
Data
bytes: 9001-10000
Receive
Send request
seq: 15001
ack: 10001
A
rwnd: 3000
Data
bytes: 15001-17000
Receive
Send request
seq: 10001
ack: 17001
A
rwnd:10000
Receive
Connection Termination
Time
Time
Time
Time

그림 9.28: 세-방향 핸드쉐이킹을 사용한 연결 종료(connection termination)

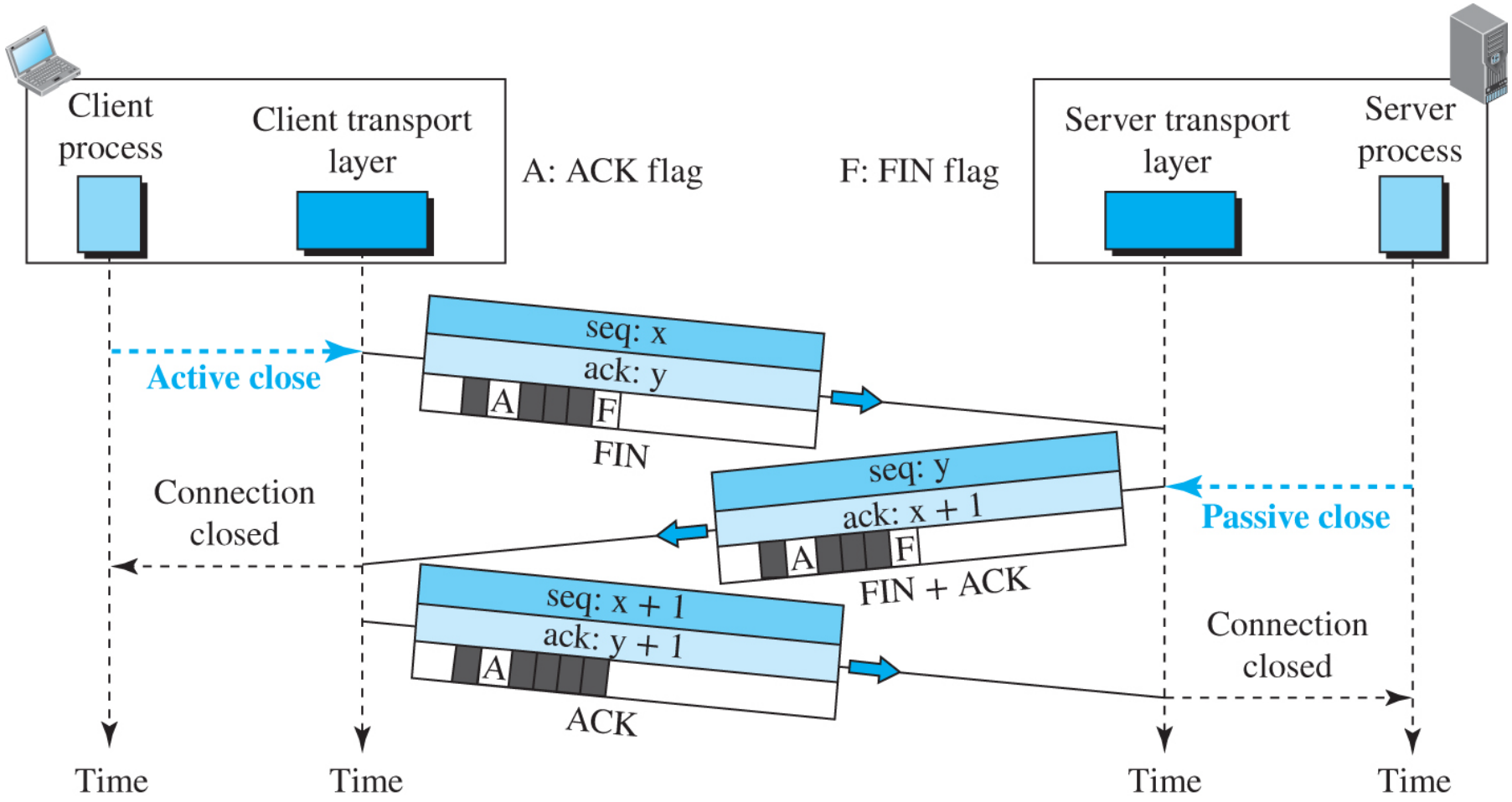
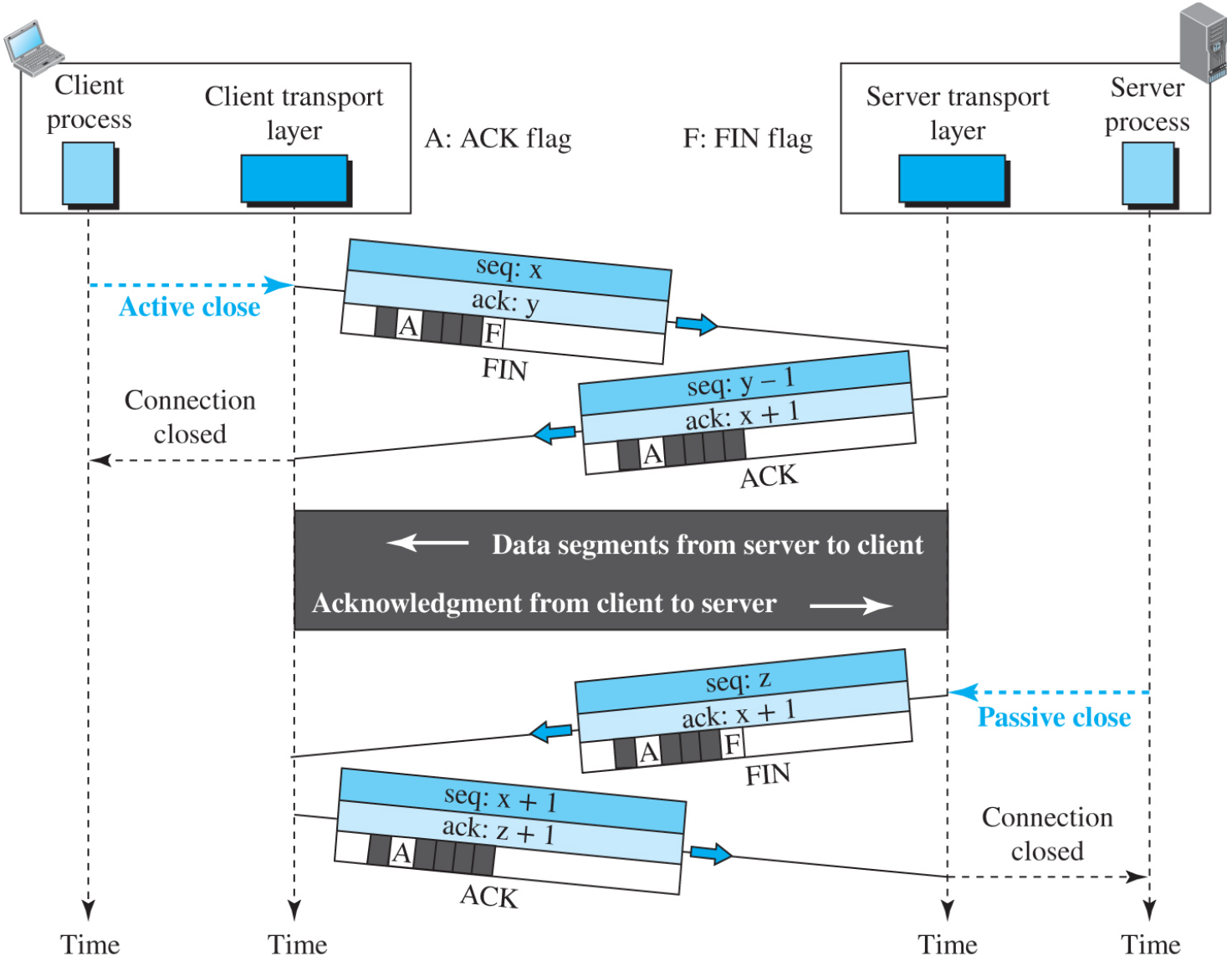


그림 9.27: 절반-연결 폐쇄 (half-close)



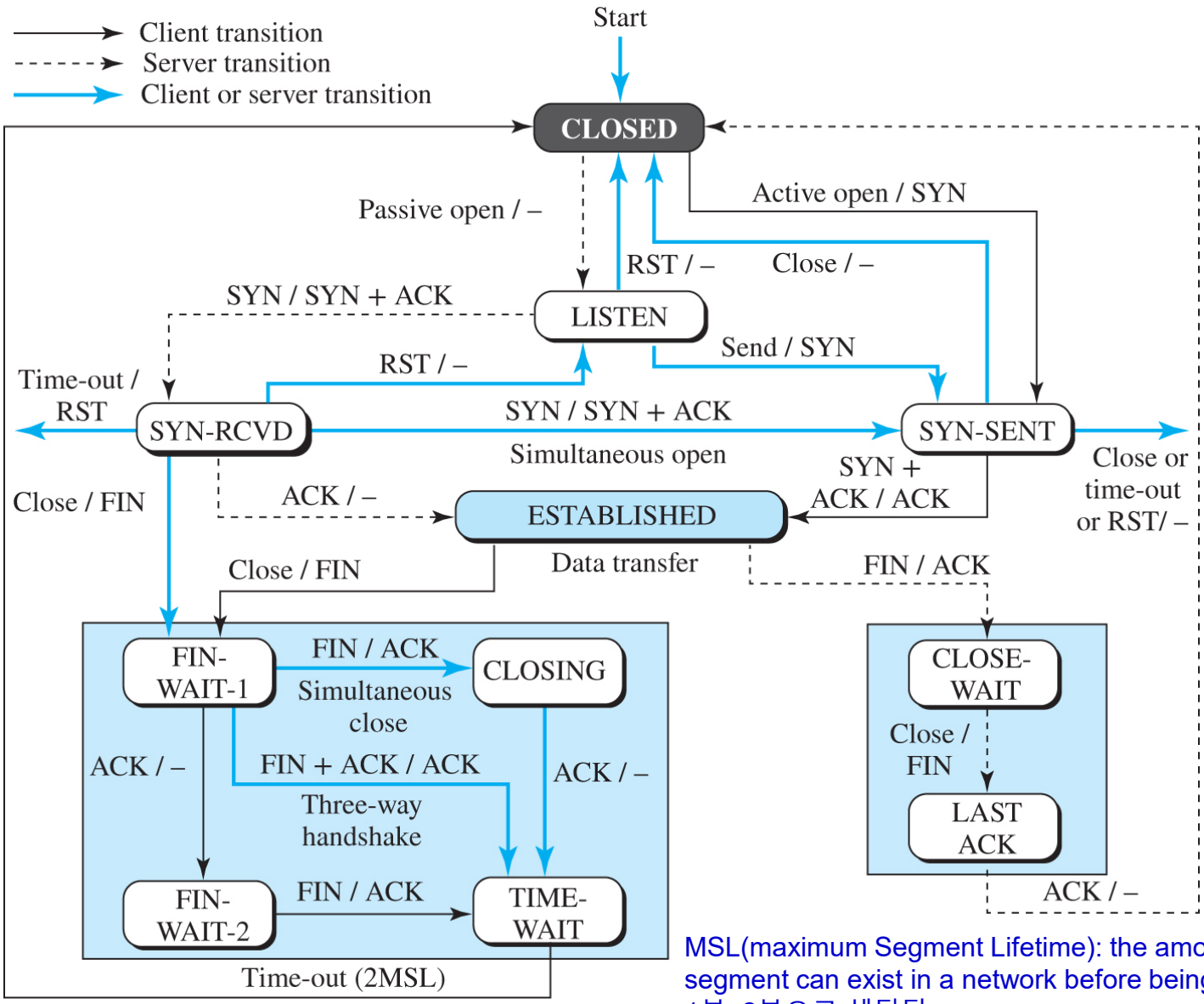
연결 리셋(Connection Reset)

- 한쪽 끝에 있는 TCP는 연결 요청을 거부하거나, 기존 연결을 중단하거나, 휴지상태 (idle state)에 있는 연결을 종료할 수 있다.
- 이 모든 것은 RST(재설정) 플래그로 수행된다.

9.4.5 상태 천이도

- 연결 설정, 연결 종료, 그리고 데이터 전송 동안 발생하는 모든 다른 이벤트를 추적하기 위하여, TCP는 그림 9.14처럼 유한 상태 기기(FSM)를 규정한다.

그림 9.14: 상태 천이도(state transition diagram)



MSL(maximum Segment Lifetime): the amount of time any segment can exist in a network before being discarded; 30초, 1분, 2분으로 셋팅됨

표 9.2: TCP의 상태

State	*Description*
CLOSED	No connection exists
LISTEN	Passive open received; waiting for SYN
SYN-SENT	SYN sent; waiting for ACK
SYN-RCVD	SYN+ACK sent; waiting for ACK
ESTABLISHED	Connection established; data transfer in progress
FIN-WAIT-1	First FIN sent; waiting for ACK
FIN-WAIT-2	ACK to first FIN received; waiting for second FIN
CLOSE-WAIT	First FIN received, ACK sent; waiting for application to close
TIME-WAIT	Second FIN received, ACK sent; waiting for 2MSL time-out
LAST-ACK	Second FIN sent; waiting for ACK
CLOSING	Both sides decided to close simultaneously

그림 9.31: 절반-폐쇄 연결 종료를 가진 상태 다이어그램
(Transition diagram with half-close connection termination)

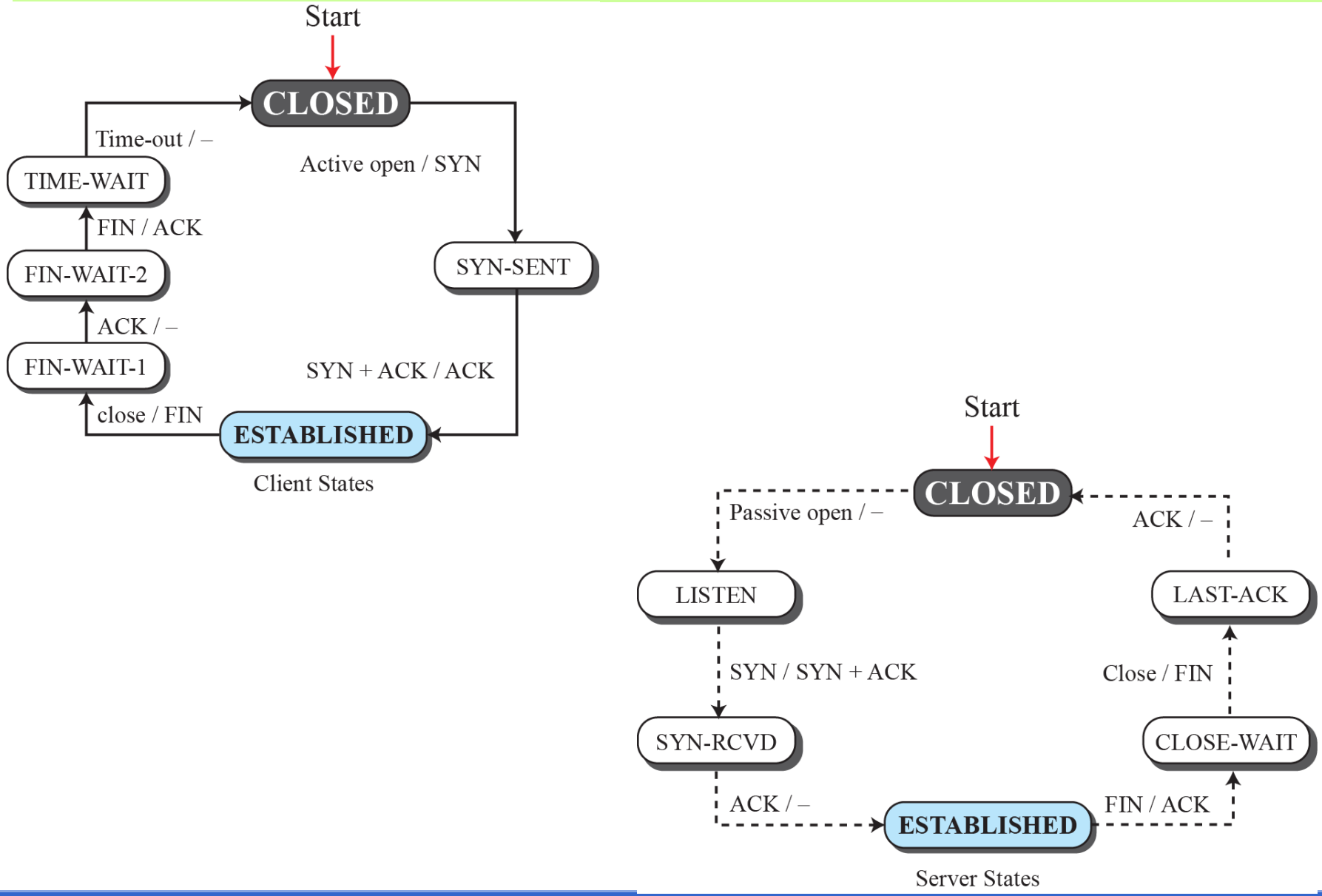
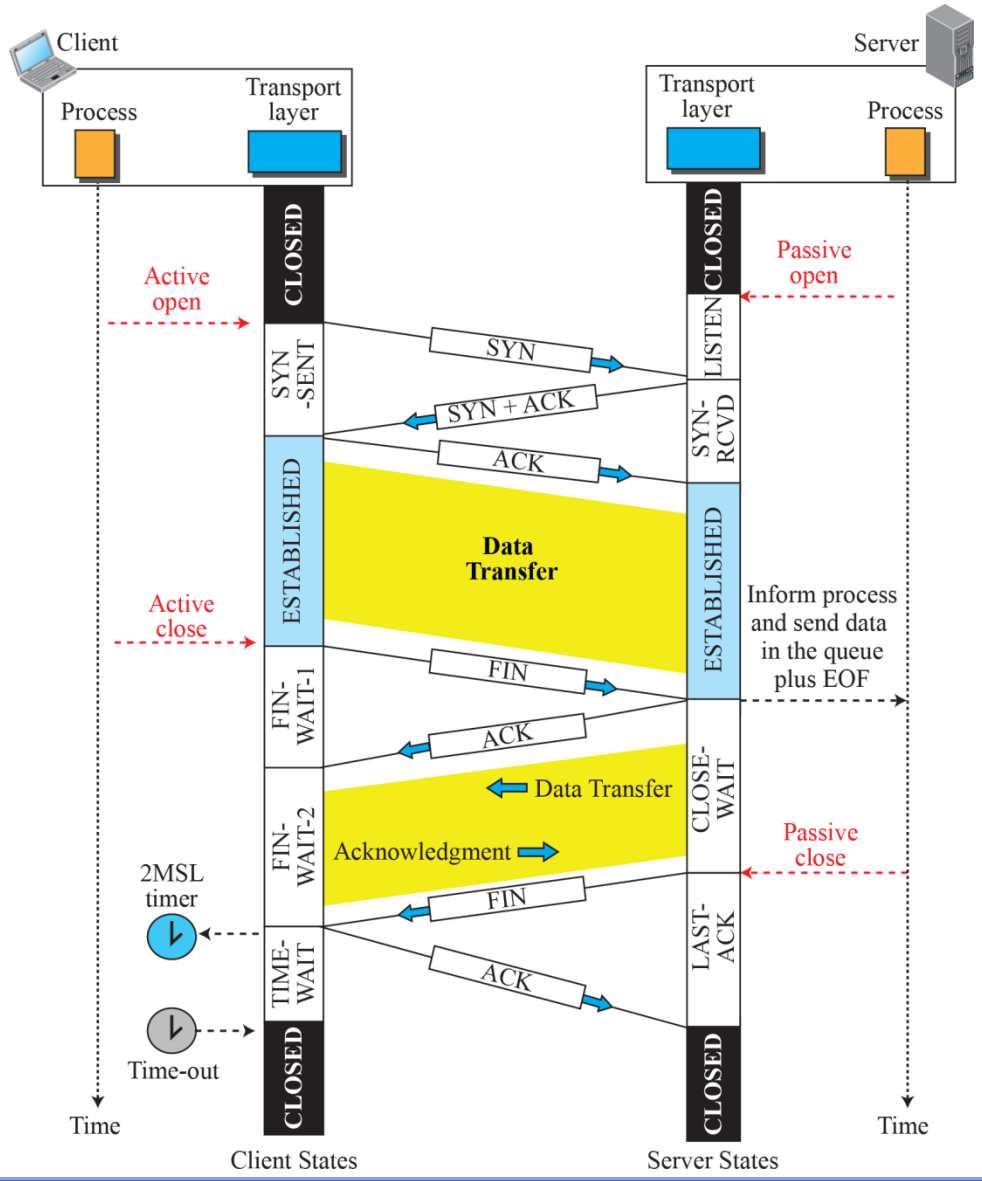


그림 9.32: 일반적인 시나리오(common scenario)의 시간축 다이어그램(time-line diagram)



9.3.6 TCP의 윈도

- TCP에서의 데이터 전송과 흐름 제어, 오류 제어, 그리고 혼잡 제어와 같은 문제를 다루기 전에 우선 TCP에서 사용되는 윈도에 대해 살펴보고자 한다.
- TCP는 데이터 전송을 위한 각 방향에 대해서 2개의 윈도(송신 윈도와 수신 윈도)를 사용하며 따라서 양방향 통신을 위하여 4개의 윈도가 필요하다.
- 하지만 간단한 설명을 위해 통신이 단방향(클라이언트로부터 서버로)으로 이루어지는 상황을 고려한다.
- 양방향 통신은 2개의 단방향 통신과 피지배깅(piggybacking)을 이용하면 유추할 수 있다.

그림 9.33: TCP의 송신 윈도(send window)

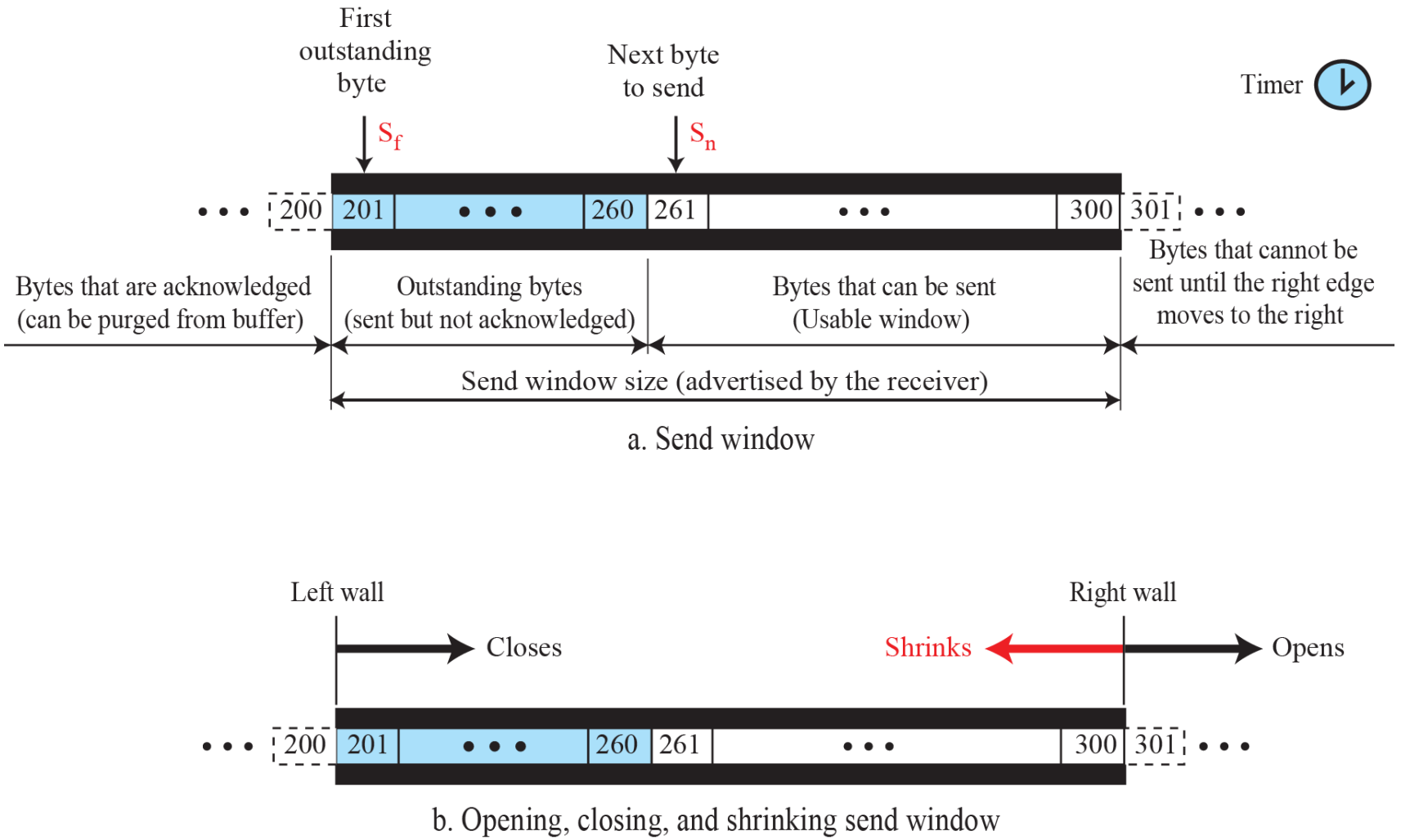
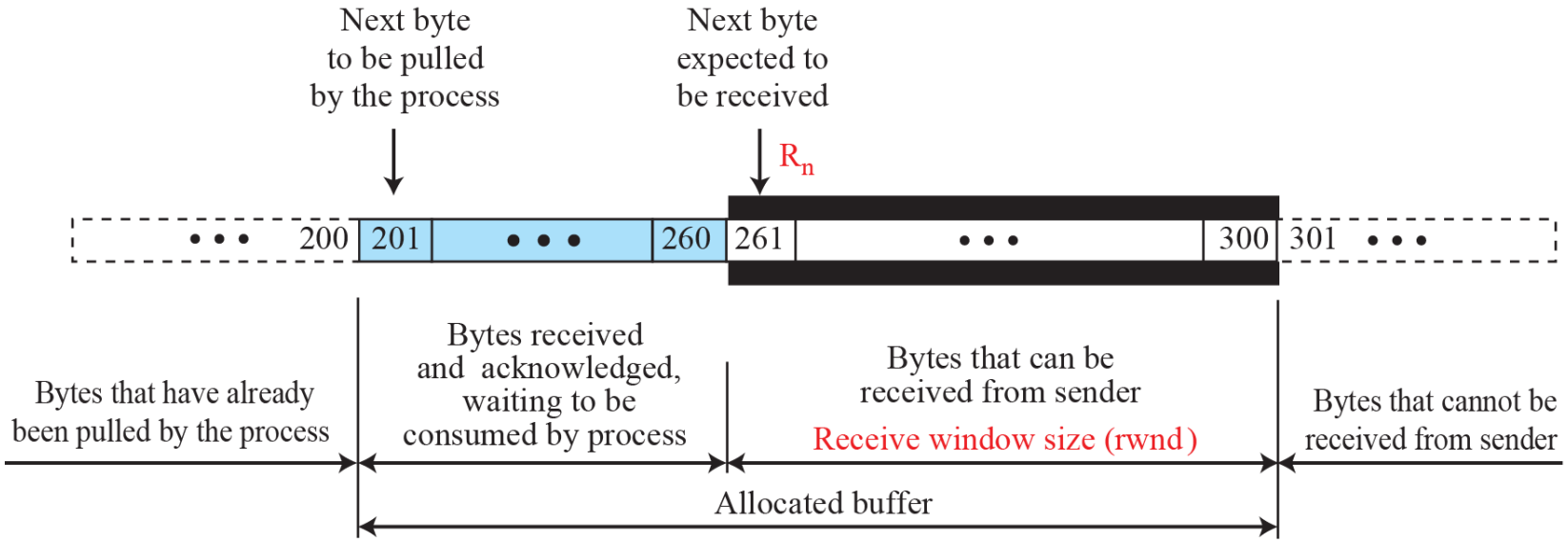
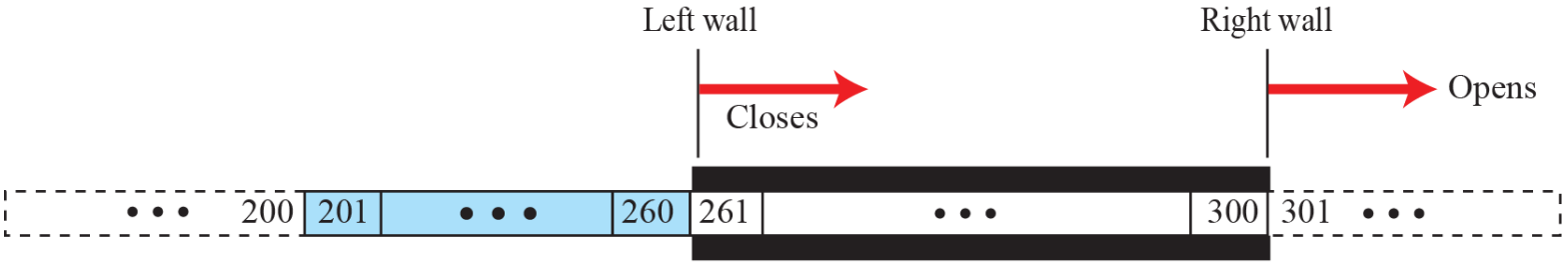


그림 9.34: TCP의 수신 윈도(receive window)



a. Receive window and allocated buffer



b. Opening and closing of receive window

9.4.7 흐름 제어

- 이전에 설명한 것과 같이 흐름 제어(flow control)는 생산자가 데이터를 만드는 속도와 소비자가 데이터를 사용하는 속도의 균형을 맞추는 것이다.
- TCP는 흐름 제어와 오류 제어를 구분한다.
- 이 절에서는 오류 제어를 무시하고 흐름 제어에 대해 설명한다. 송신과 수신 TCP 사이에 설정된 논리 채널은 오류가 없다고 가정한다.

그림 9.35: TCP에서의 데이터 흐름과 흐름 제어 피드백

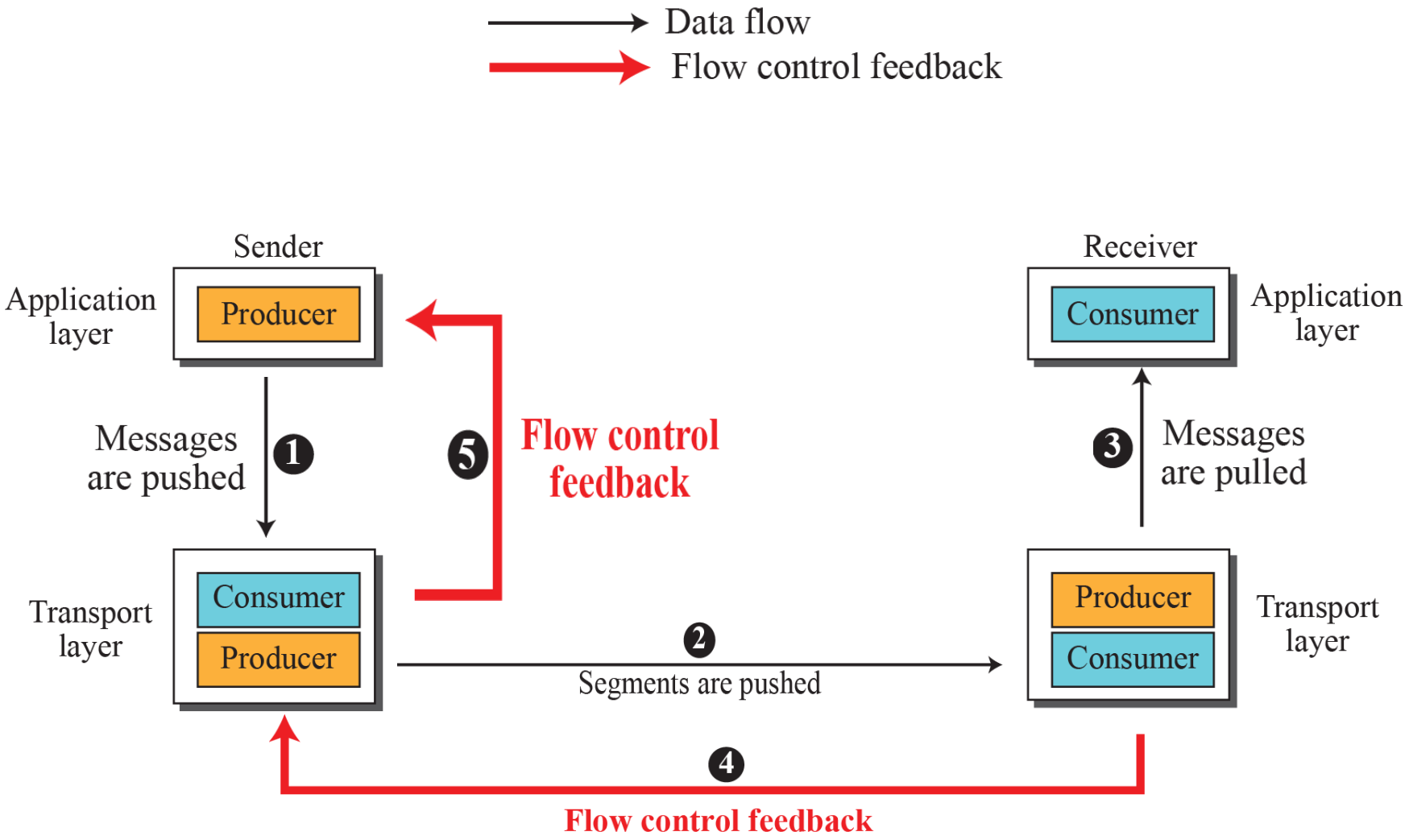
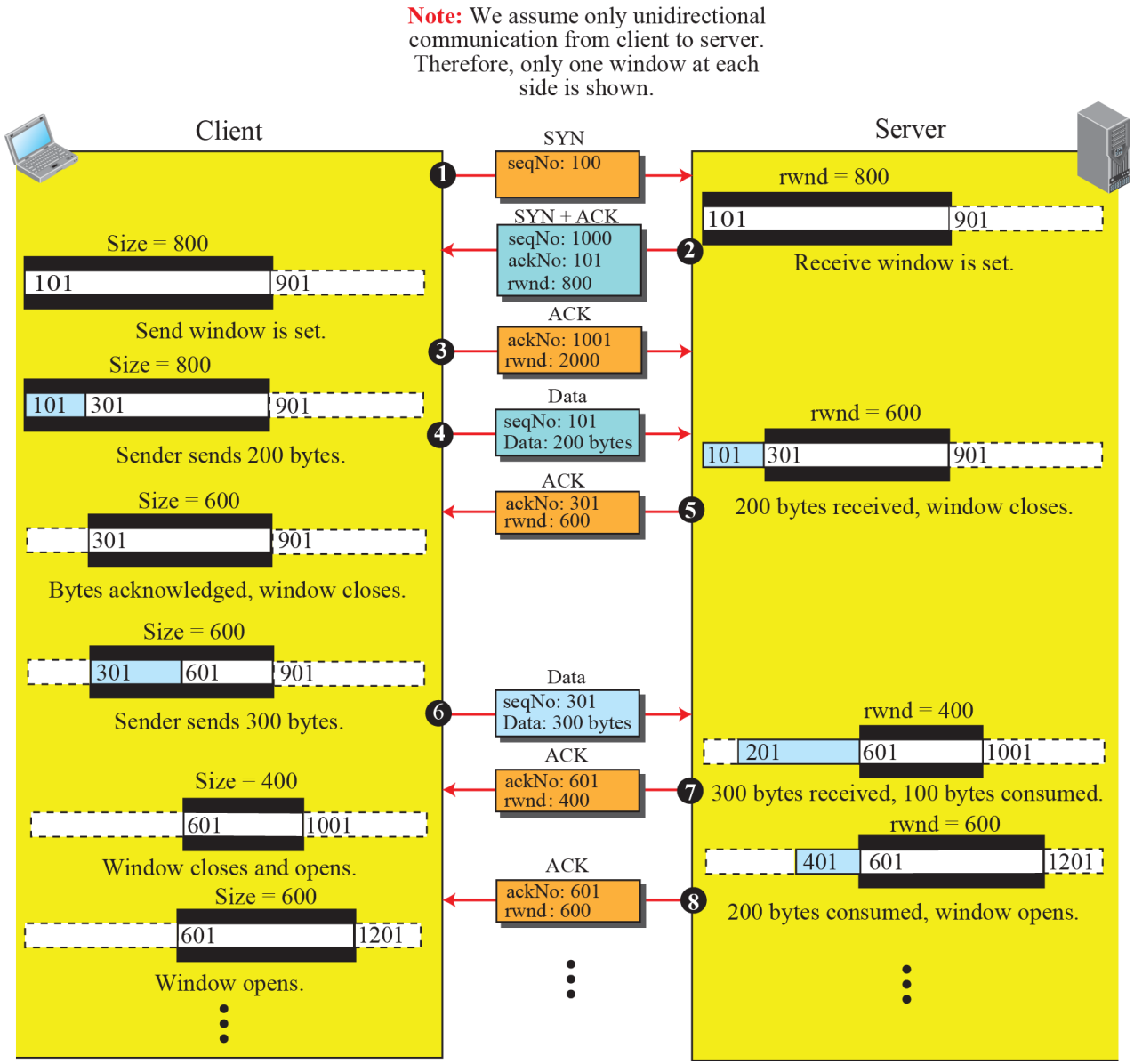


그림 9.36: 흐름 제어의 예



윈도우 축소

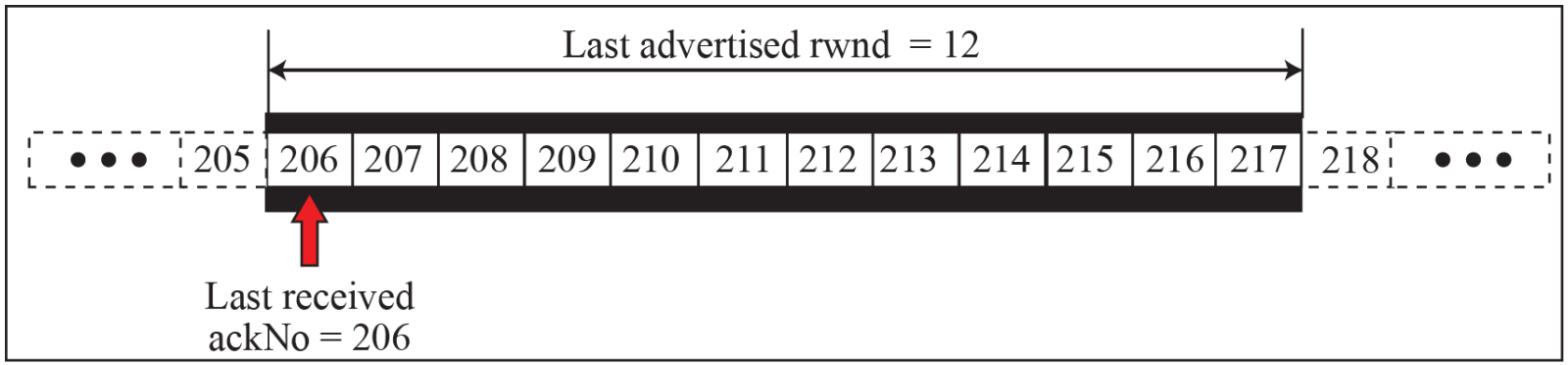
- 수신창(receive window)은 축소할 수 없다.
- 반면에 수신자가 rwnd 값을 정의하여 창을 축소하는 경우 송신창(send window)은 축소될 수 있다.
- 그러나 일부 구현(Implementation)에서는 송신창 축소를 허용하지 않는다.
- 송신창의 오른쪽 벽이 왼쪽으로 이동하는 것을 허용하지 않는 제한 사항이다.
- 즉, 수신자는 마지막 및 새 확인응답(last and new ack)과, 마지막 및 새 rwnd 값(last and new receive window value) 사이의 다음 관계를 유지하여 송신창의 축소를 방지해야 한다.

new ackNo + new *rwnd* >= last ackNo + last *rwnd*

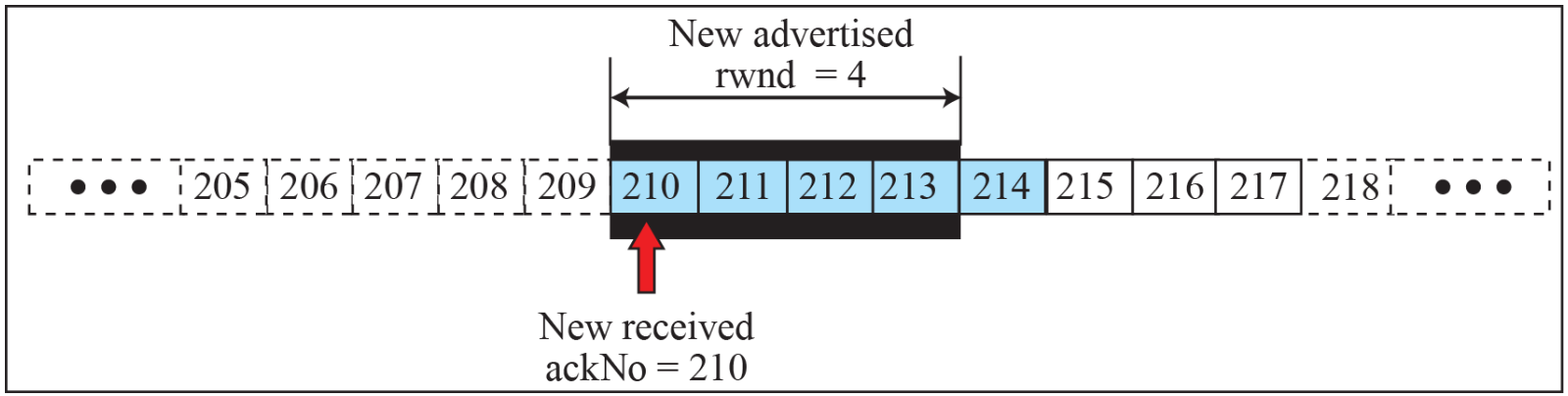
예제 9.9

- 그림 9.37은 필수사항에 대한 이유를 보여준다.
- 그림의 a는 마지막 확인응답과 rwnd 값을 보여준다. 그림 b는 송신측에서 206에서 214까지의 바이트를 전송한 경우를 보여준다.
- 206~209바이트는 확인응답되었고 제거되었다. 그렇지만 rwnd가 새로운 값인 4로 정의된다.
- 즉 210+4<206+12가 된다. 송신 윈도가 축소될 때 문제가 발생한다. 이미 전송된 바이트 214는 윈도의 외부에 있다.
- 수신측에서는 바이트 210부터 바이트 217 중에서 이미 전송된 것이 어떤 것인지 알 수 없기 때문에 앞에서 언급한 관계식은 수신측에서 윈도의 오른쪽 벽을 유지할 수 있도록 한다.
- 이 같은 상황에 대처하기 위해 수신측은 충분한 버퍼를 확보할 때까지 피드백을 연기하는 전략이 필요하다.

그림 9.37: 예제 9.9



a. The window after the last advertisement



b. The window after the new advertisement; window has shrunk

9.4.8 오류 제어

- TCP는 신뢰성 있는 전송층 프로토콜이다.
- 즉 데이터 스트림을 TCP로 전달하는 응용 프로그램은 TCP가 전체 스트림을 순서에 맞고 오류 없이, 또한 부분적인 손실이나 중복 없이 상대편에 있는 응용 프로그램에게 전달함을 확신하는 것을 의미한다.
- TCP는 오류 제어를 이용하여 신뢰성을 제공한다.
- 오류 제어는 훼손된 세그먼트의 감지 및 재전송, 손실 세그먼트의 재전송, 분실된 세그먼트가 도착하기 전까지 순서가 맞지 않는 세그먼트를 저장하고 중복 세그먼트의 감지 및 폐기를 위한 메커니즘을 포함한다.
- TCP에서의 오류 제어는 간단한 세 가지 도구인 검사합, 확인응답(acknowledgment) 그리고 타임아웃(timeout) 등을 통해 수행된다.

6가지 오류제어 규칙

- **Rule1**:종단 A가 종단 B로 데이터 세그먼트를 보낼 때 수신할 다음 시퀀스 번호를 제공하는 승인을 포함(piggyback)해야 한다. 이 규칙은 필요한 세그먼트 수를 줄여 트래픽을 줄인다.

- **Rule2**: 수신자가 보낼 데이터가 없고 순서대로(in-order) 세그먼트(예상한 _expected 시퀀스 번호 포함)를 수신하고, 이전 세그먼트가 이미 확인응답(ACKed)된 경우 수신자는 다른 세그먼트가 도착할 때까지 또는 일정 기간(일반적으로 500ms)이 지날때까지 기다린다. 즉, 수신자는 미해결 순서 세그먼트가 하나만 있는 경우 ACK 세그먼트 전송을 지연해야 한다. 이 규칙은 ACK 세그먼트 수를 줄인다.

6가지 오류제어 규칙

- **Rule3** : 수신자가 예상한 시퀀스 번호로 세그먼트가 도착하고 이전 순서대로 도착된 세그먼트가 승인되지 않은 경우 수신자는 즉시 ACK 세그먼트를 보냅니다. 다시 말해서, 순서대로 도착한 미확인 세그먼트는 한 번에 2개 이상 있어서는 안 된다. 이것은 네트워크에 혼잡을 일으킬 수 있는 세그먼트의 불필요한 재전송을 방지한다.
- **Rule4**: 세그먼트가 예상보다 높은 순서에 맞지 않는 번호로 도착하면 수신자는 즉시 다음 예상 (받기를 원하는) 세그먼트의 순서 번호를 알리는 ACK 세그먼트를 보낸다. 이것은 누락된 세그먼트의 빠른 재전송으로 이어진다.
- **Rule5**: 누락된 세그먼트가 도착하면 수신기는 ACK 세그먼트를 전송하여 예상되는 다음 시퀀스 번호를 알립니다. 이것은 누락된 세그먼트가 수신되었음을 수신자에게 알리는 것이다.
- **Rule6**: 중복된 세그먼트가 도착하면 수신자는 세그먼트를 버리지만 예상되는 다음 순서 세그먼트를 나타내는 확인응답(ACK)을 즉시 보낸다. 이것은 ACK 세그먼트 자체가 손실될 때 발생하는 문제를 해결한다.

그림 9.38: TCP 송신측의 단순화된 FSM

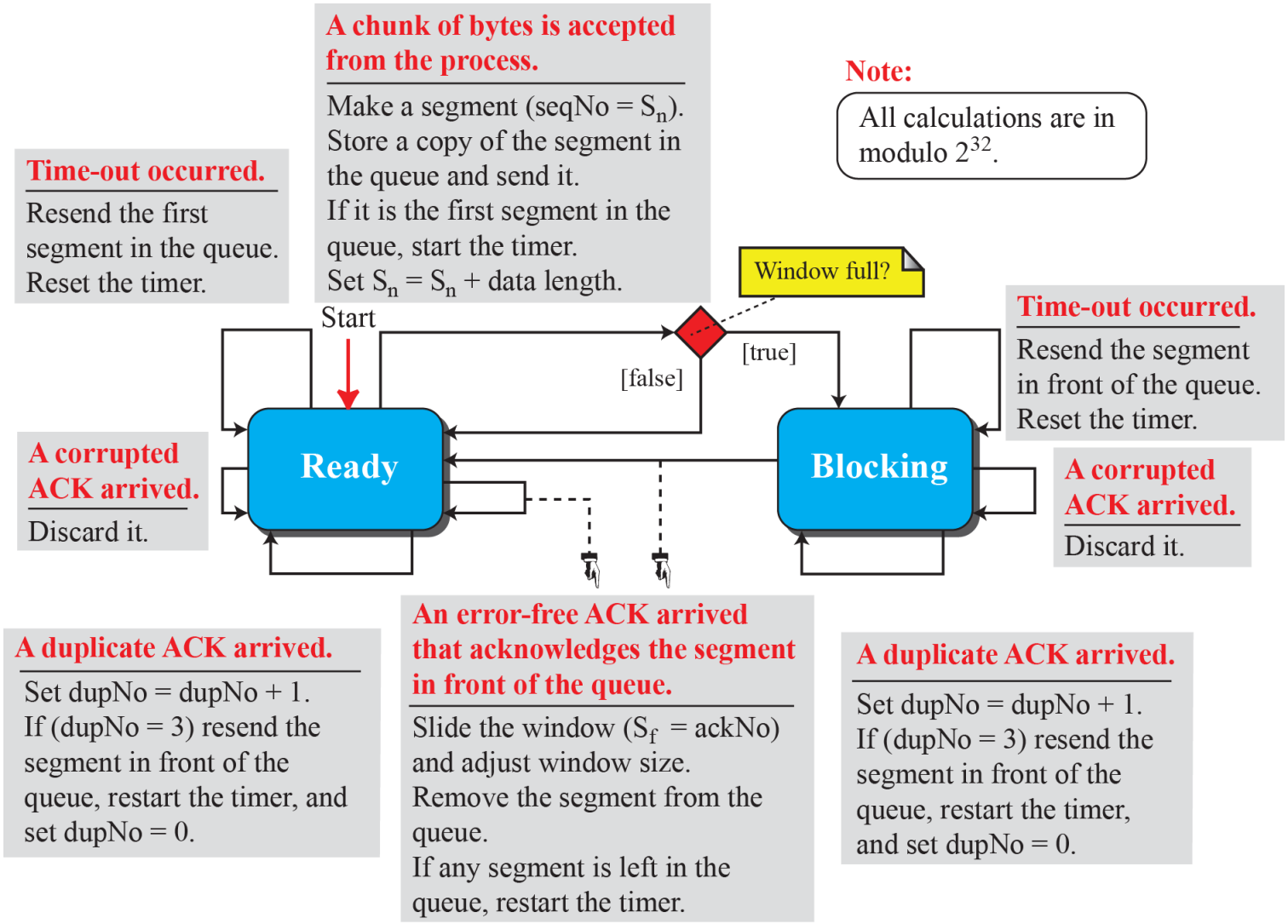


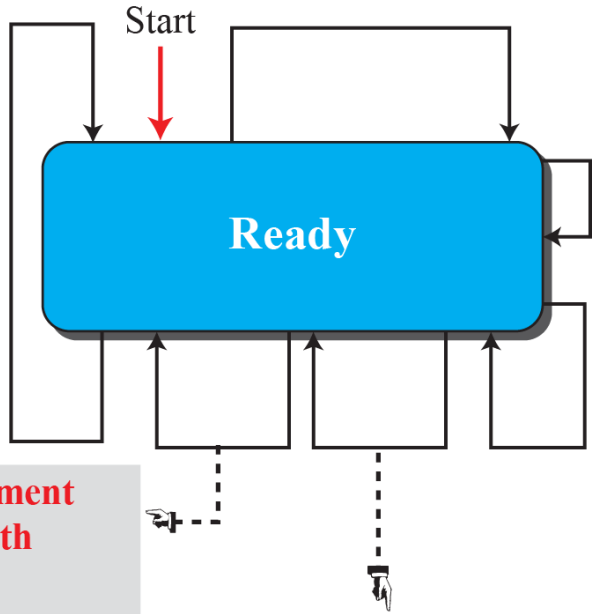
그림 9.39: TCP 수신측의 단순화된 FSM

An expected error-free segment arrived.

Buffer the message.
$R_n = R_n$ + data length.
If the ACK-delaying timer is running,
stop the timer and send a cumulative ACK.
Otherwise, start the ACK-delaying timer.

Note:

All calculations are in modulo 2^{32}.



A request for delivery of k bytes of data from process came.

Deliver the data.
Slide the window and adjust window size.

ACK-delaying timer expired.

Send the delayed ACK.

An error-free, but out-of order segment arrived.

Store the segment if not duplicate.
Send an ACK with ackNo equal to the sequence number of expected segment (duplicate ACK).

An error-free duplicate segment or an error-free segment with sequence number outside window arrived.

Discard the segment.
Send an ACK with ackNo equal to the sequence number of expected segment (duplicate ACK).

A corrupted segment arrived.

Discard the segment.

그림 9.40: 정상 동작(normal operation)

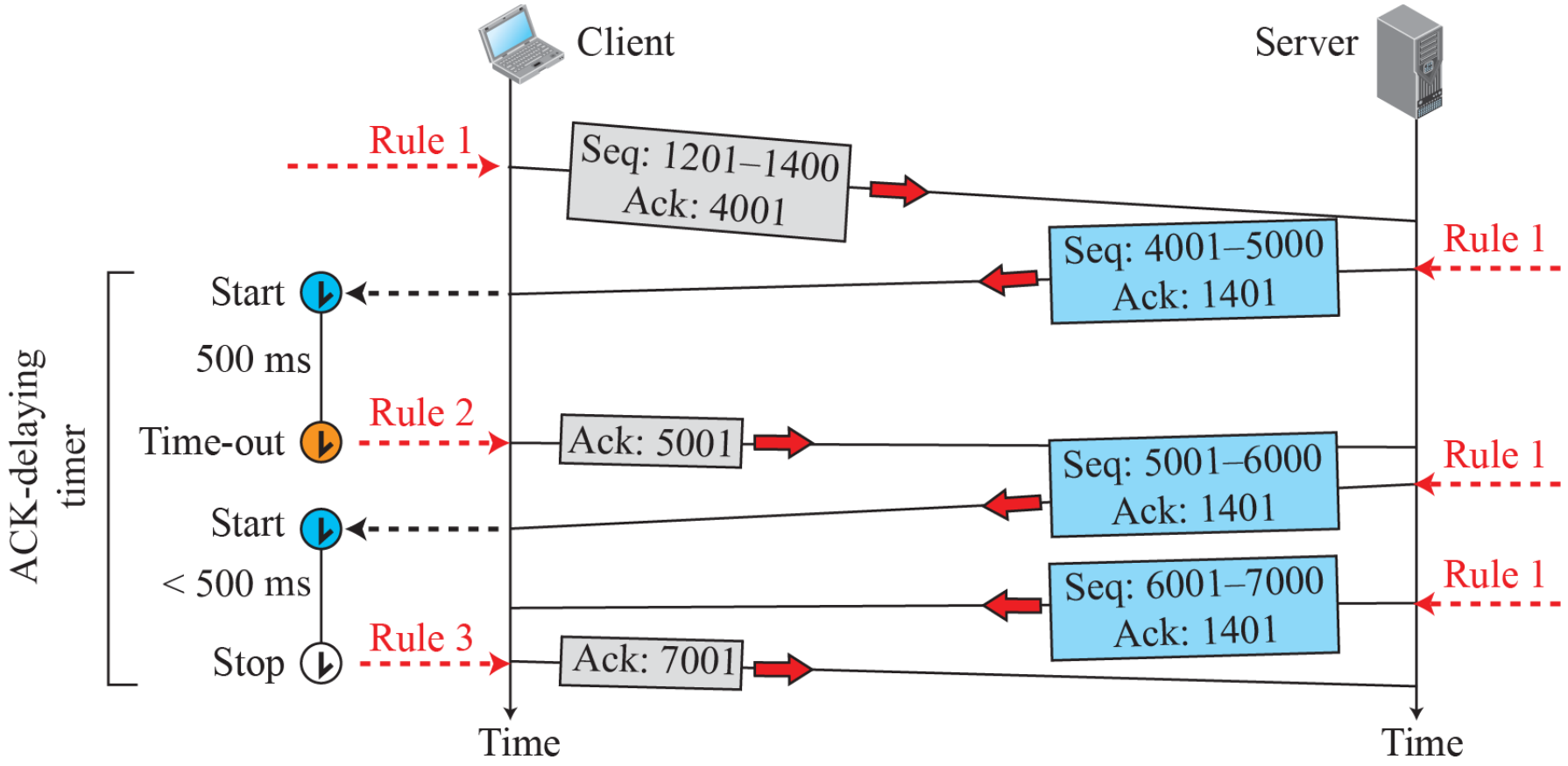


그림 9.41: 손실 세그먼트(lost segment)

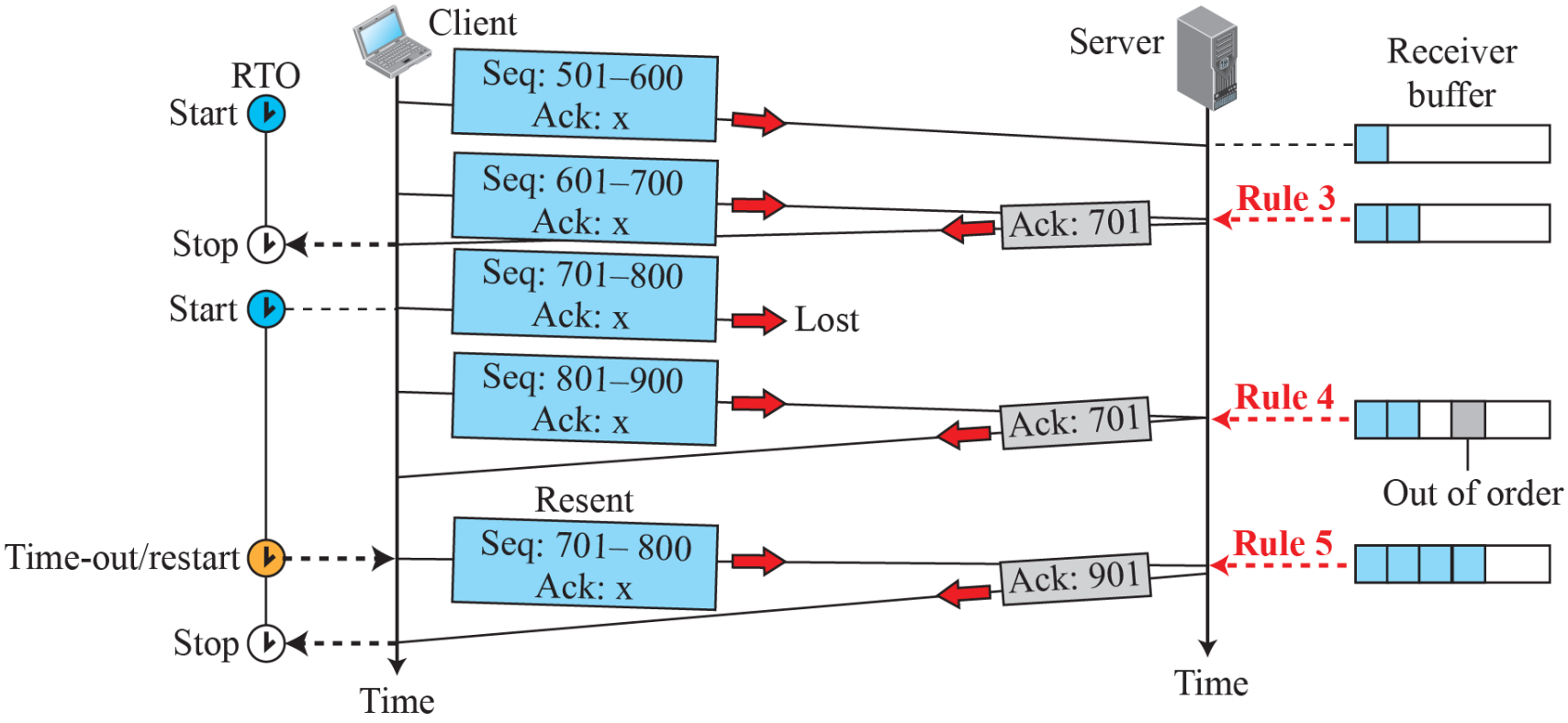


그림 9.42: 빠른 재전송(fast retransmission)

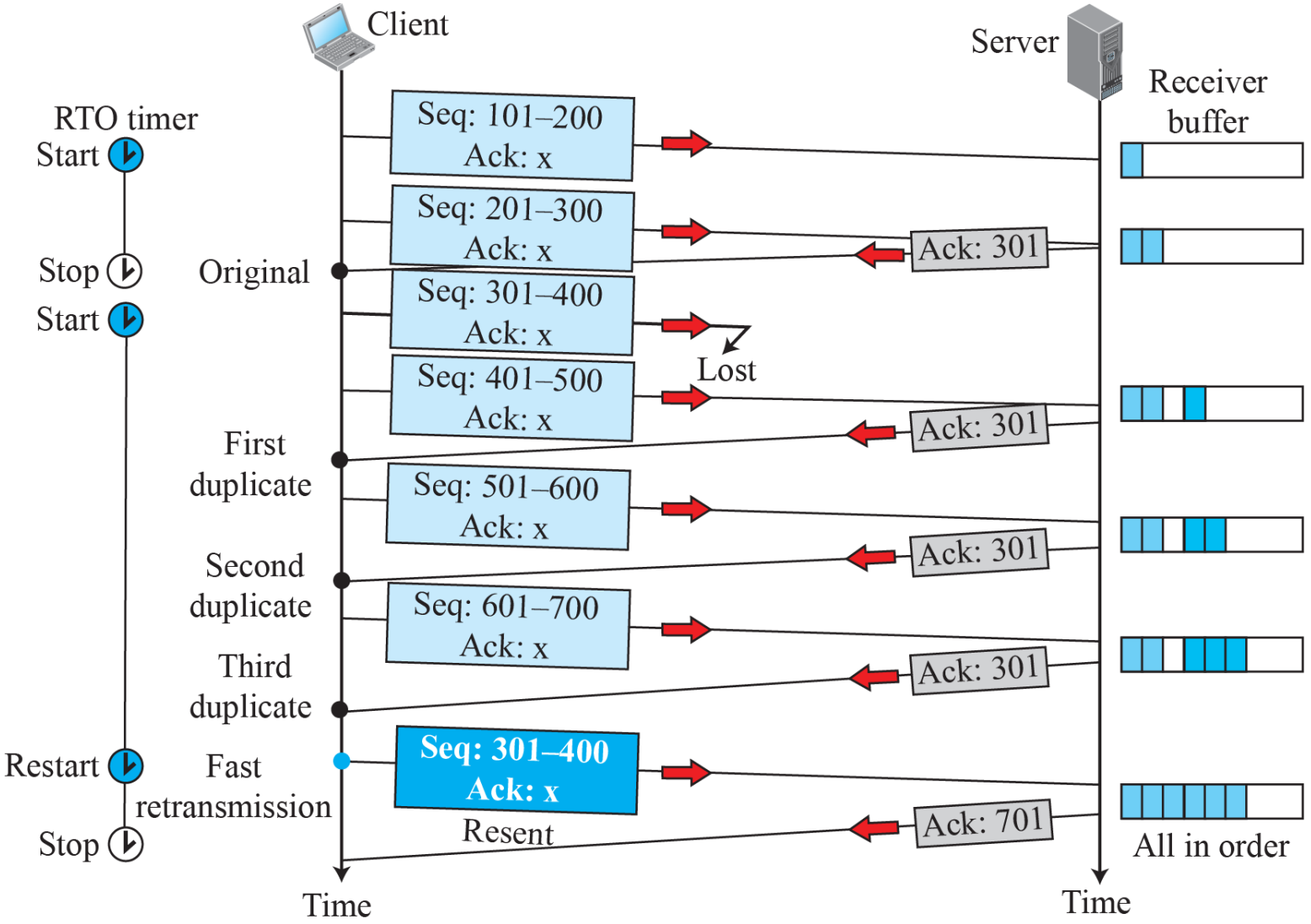


그림 9.43: 확인응답 분실(lost acknowledgment)

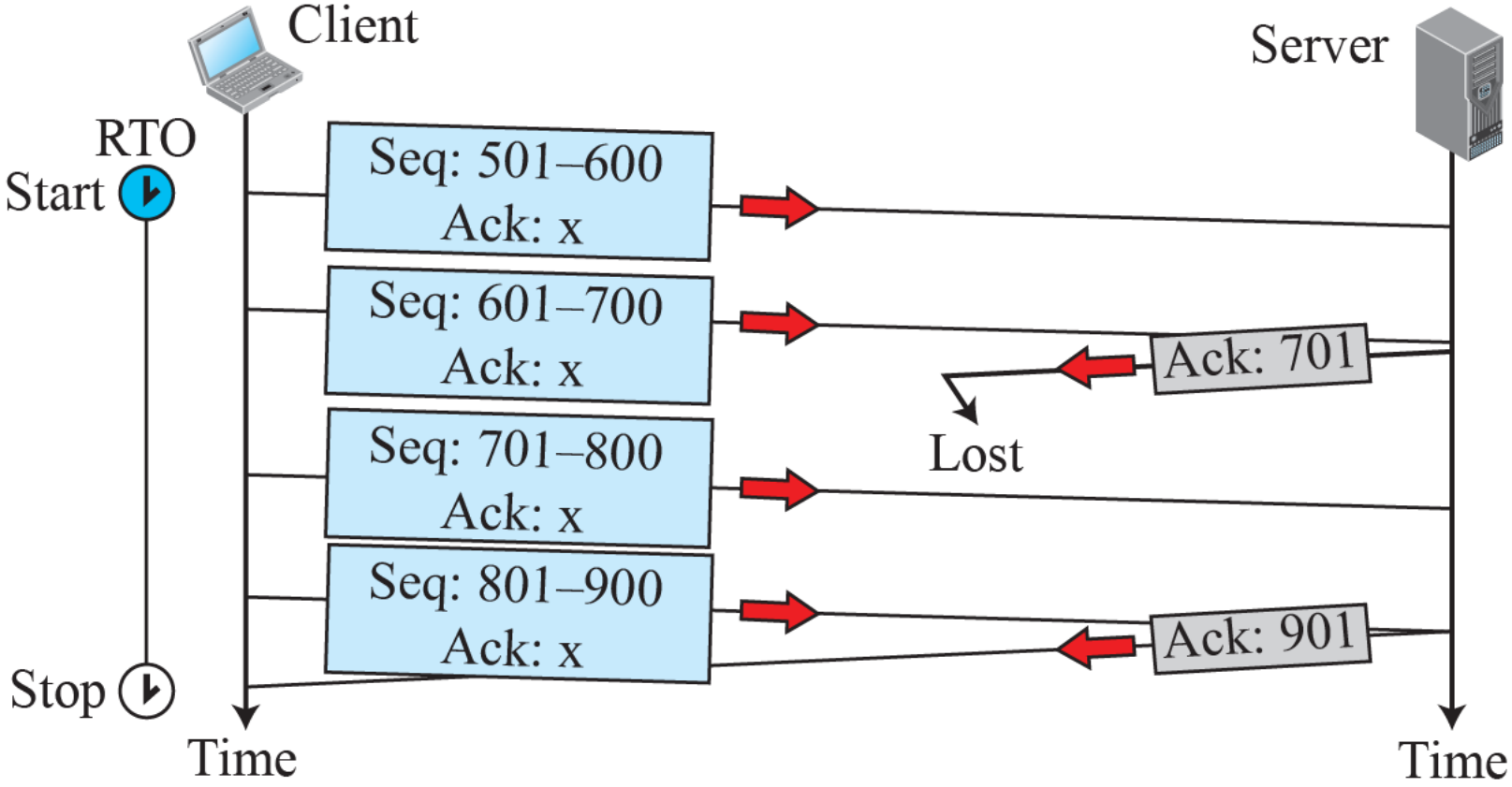
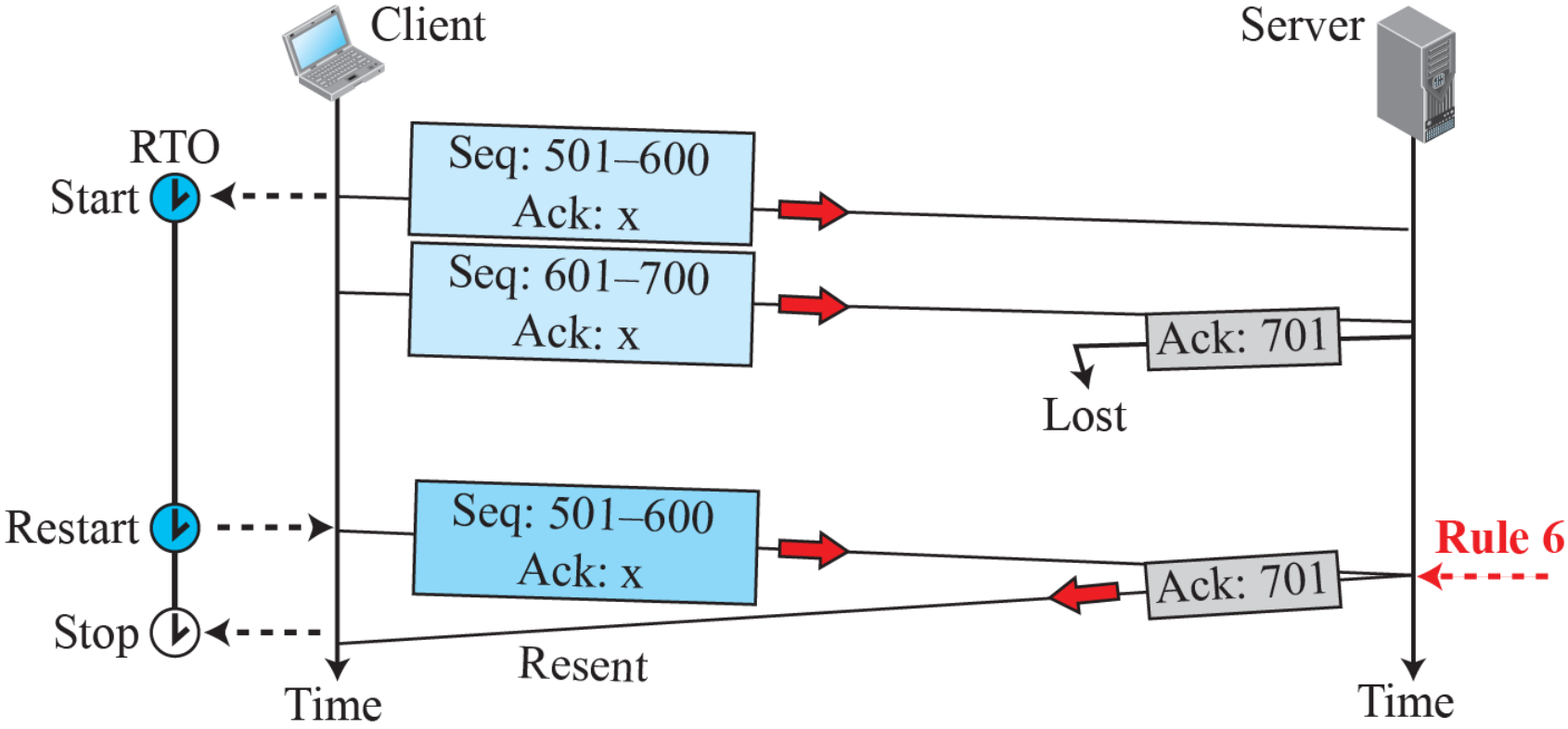


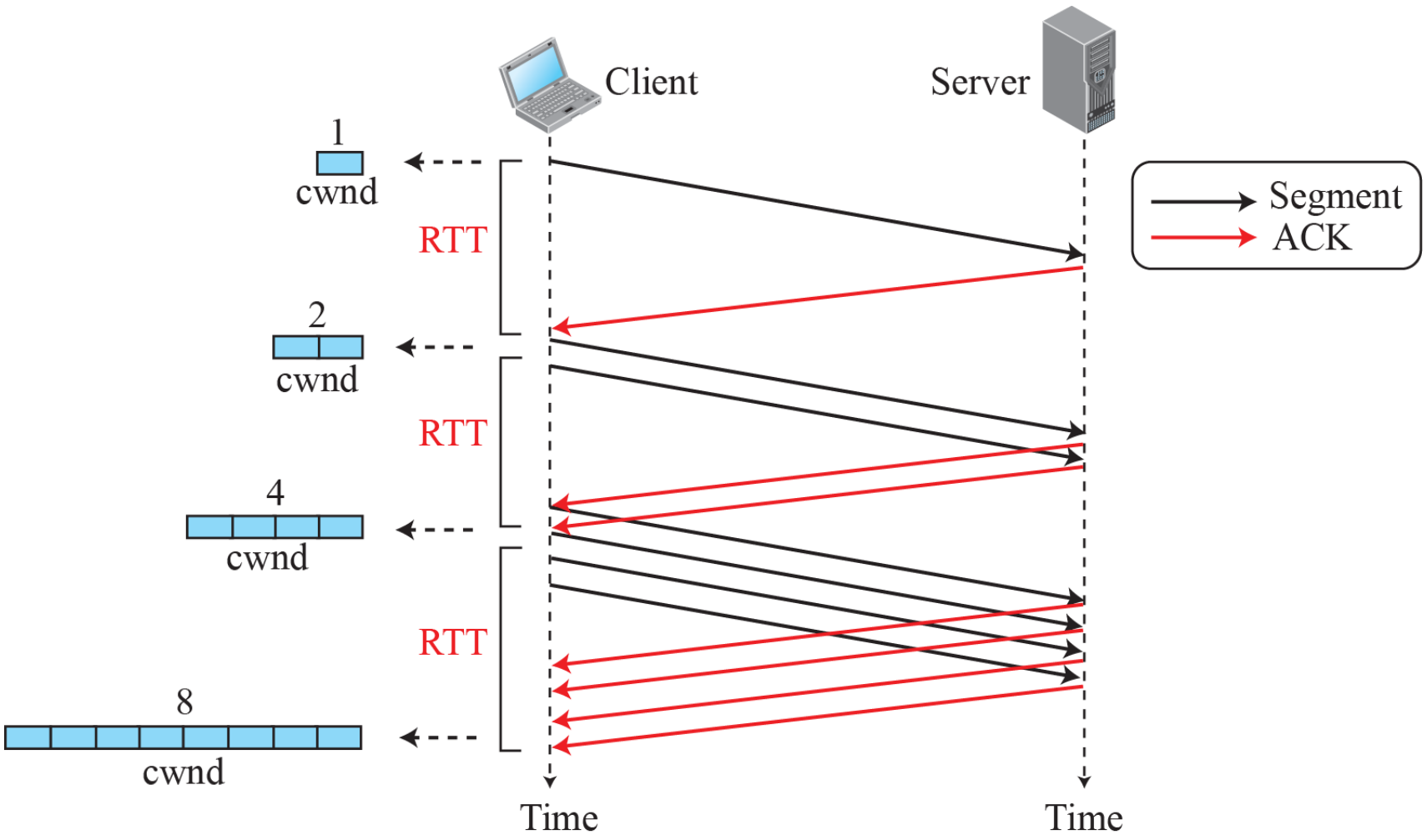
그림 9.44: 세그먼트를 재전송함으로써 교정되는 분실 확인응답
(Lost acknowledgment corrected by sending a segment)



9.4.9 TCP 혼잡 제어

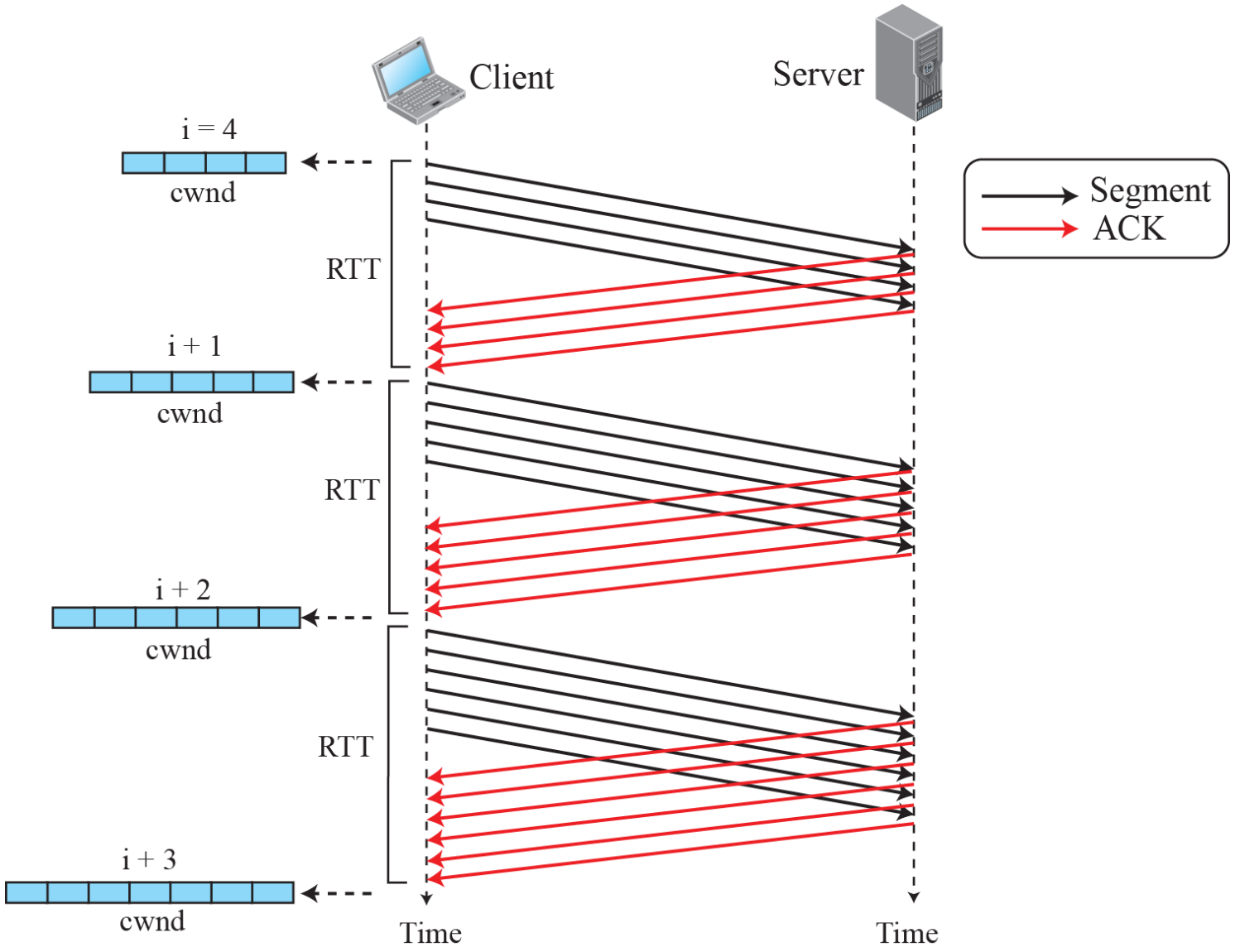
- 혼잡 윈도우 (Congestion Window)
  - Actual window size = minimum (rwnd, cwnd)
- TCP의 혼잡 제어를 위해 다른 정책
  - 느린 시작(slow start): 지수 증가(exponential increase)
  - 혼잡 회피(congestion avoidance): 가산 증가(additive increase)
  - 빠른 회복(fast recovery)

그림 9.45: 느린 시작, 지수 증가(Slow start, exponential increase)



RTT: Round Trip Time

그림 9.46: 혼잡 회피, 가산 증가(congestion avoidance, additive increase)



혼잡제어 정책 전환

- TCP의 세 가지 혼잡 정책에 대해 논의했다. 이제 문제는 이러한 각 정책이 사용되는 시기와 TCP가 한 정책에서 다른 정책으로 이동할 때이다. 이러한 질문에 답하기 위해 TCP의 세 가지 버전인 Taho TCP, Reno TCP 및 New Reno TCP를 참고해야 한다.

  - Taho TCP
  - Reno TCP
  - New Reno TCP

예제 9.10

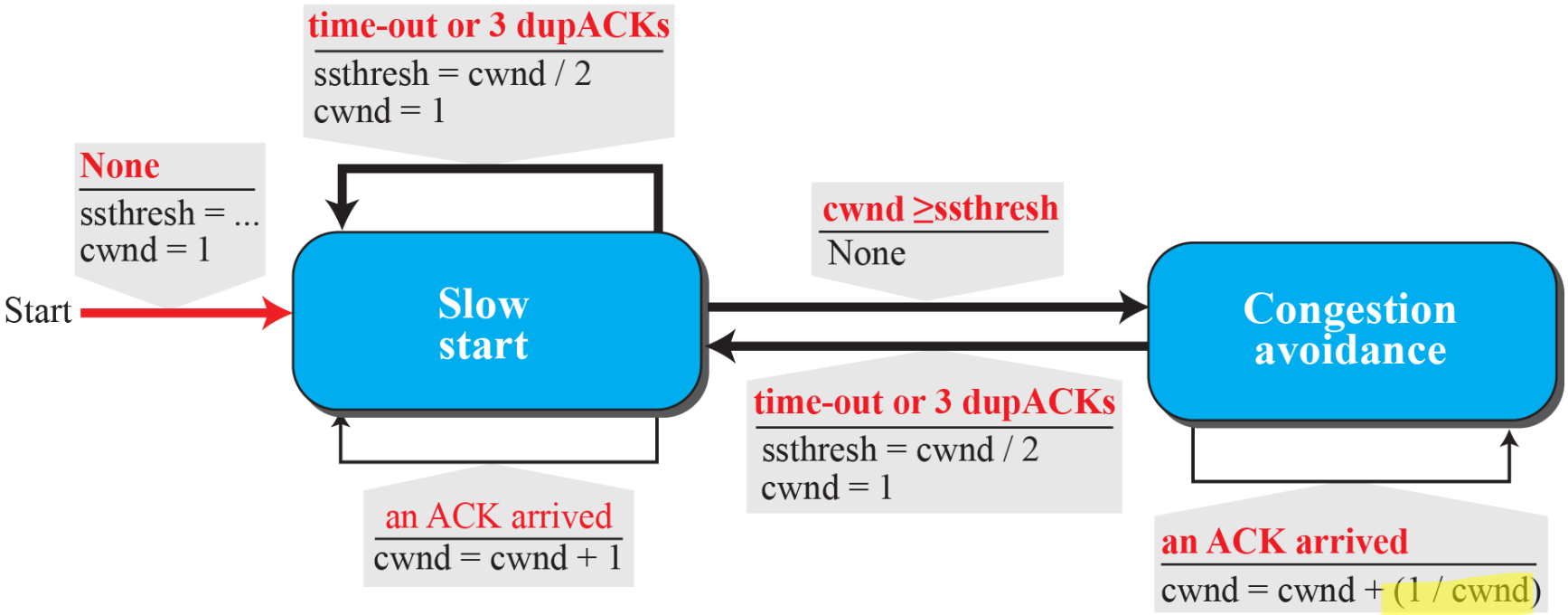
- 그림 9.48는 Taho TCP에서의 혼잡 제어를 보여준다. TCP는 데이터 전송을 시작하고, ssthresh 변수를 16MSS로 정한다.
- TCP는 cwnd=1에서 느린 출발(SS, slow-start)을 시작한다. 혼잡 윈도는 기하급수적으로 증가하지만 타임아웃은 세 번째 RTT(임계치에 도달하기 전에) 후에 발생한다.
- TCP는 네트워크에 혼잡이 있다고 가정한다. 즉시 새로운 ssthresh를 4MSS(현재 8로 설정된 cwnd의 반)로 설정한 다음 cwnd=1로 설정된 새로운 출발을 시작한다.
- 혼잡은 기하급수적으로 새로 설정된 임계치까지 증가하게 된다.
- TCP는 이제 혼잡 회피(CA: Congestion Avoidance) 상태로 변하고, 혼잡 윈도는 추가적(additive)으로 cwnd=12MSS에 도달할 때까지 증가한다.

* MSS: maximum segment size
* ssthresh: slow start threshold

예제 9.10 (계속)

- 그 순간 3개의 중복된 ACK가 도착하고, 또 다른 혼잡이 암시된다. TCP는 ssthresh 값을 반인 6MSS로 줄이고, 느린 시작 상태가 된다.
- cwnd의 지수적 증가는 계속된다. RTT 15인 후에 cwnd 크기의 크기는 4이다. 4개의 세그먼트를 보내고 오직 두 ACK를 받은 후에 윈도 크기가 ssthresh(6)에 도달한 TCP는 혼잡 회피 상태에 들어간다.
- 연결이 RTT 20후 끊어질 때까지 데이터 전송은 혼잡 회피(CA) 상태에서 계속된다.

그림 24.31: Taho TCP의 FSM



❖ 1/cwnd portion of MSS

그림 9.48: Taho TCP의 예

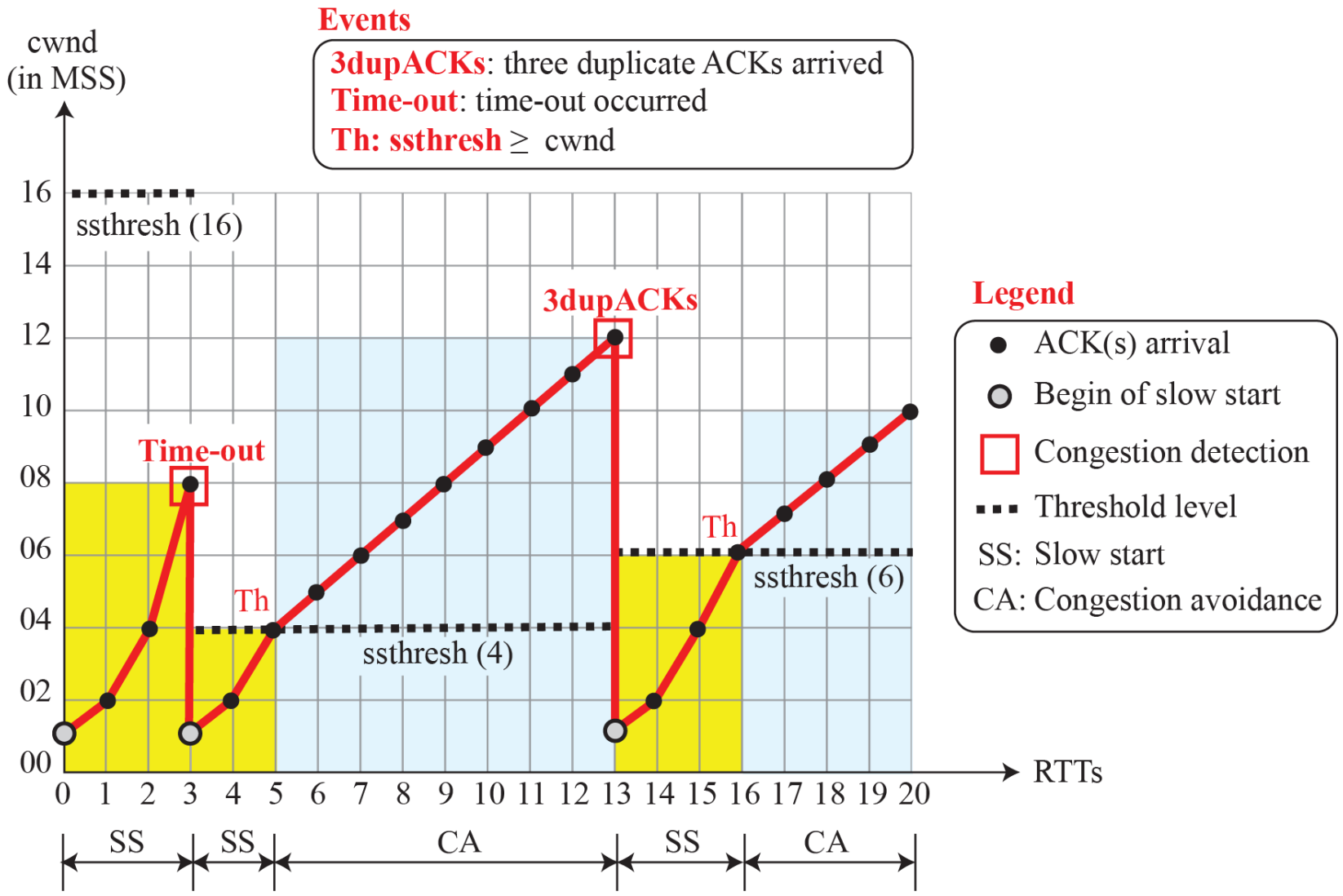
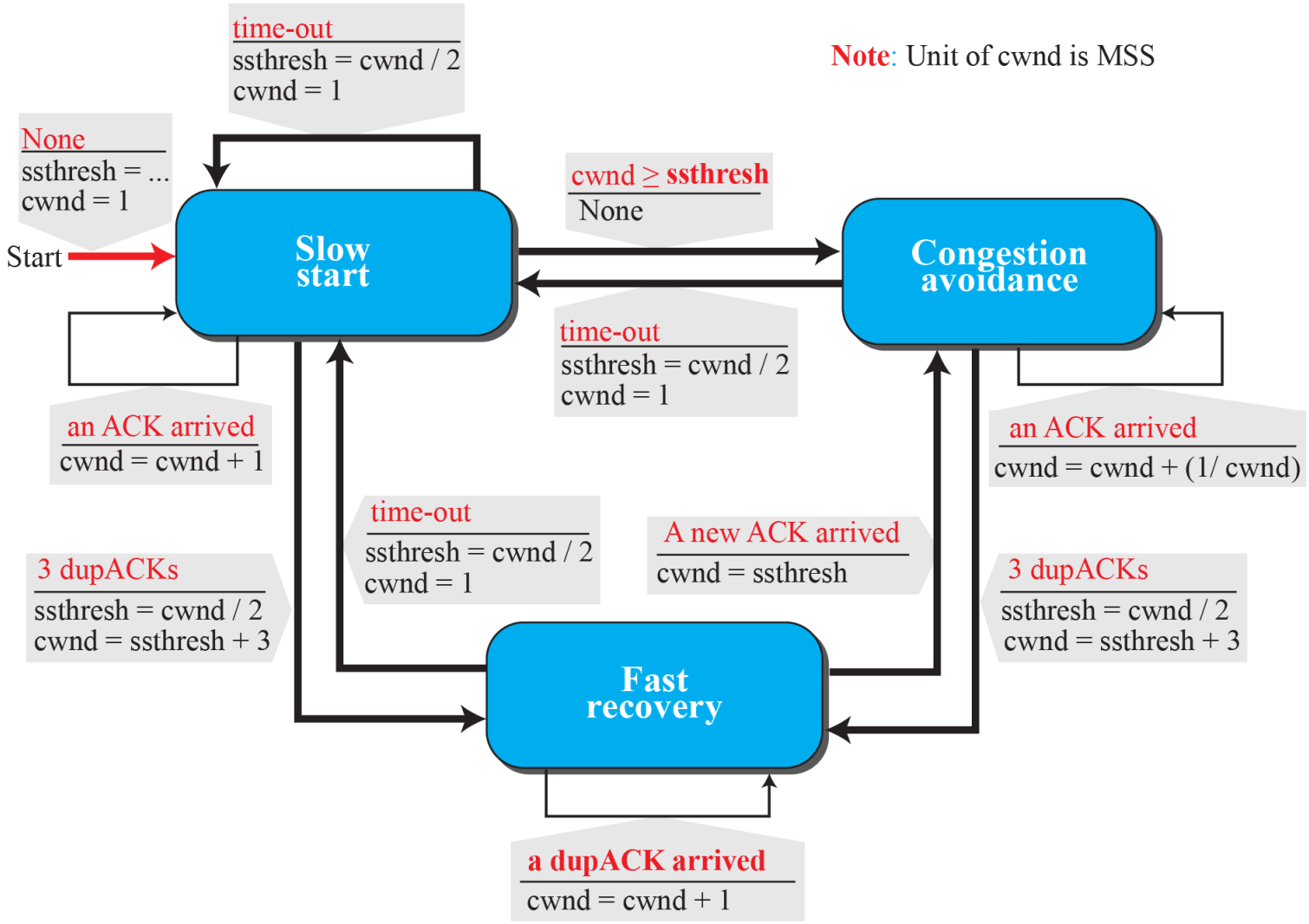


그림 9.49: Reno TCP의 FSM



예제 9.11

- 그림 9.50는 그림 9.48와 같은 상황을 보여주고 있지만 Reno TCP에 관한 내용이다. 3개의 중복 ACK들이 도착할 때 혼잡 윈도의 변화는 RTT 13까지 동일하다.
- 그 순간, Reno TCP는 ssthresh를 6MSS(Taho TCP로서 동일한 것)로 감소하지만, 대신 cwnd를 1MSS대신에 높은 값(ssthresh+3=9MSS)으로 설정한다.
- Reno TCP는 빠른 회복 상태로 전환한다. 추가적으로 2개의 중복 ACK가 RTT 15까지 도착하고 cwnd가 지수적으로 증가한다고 가정하고 있다.
- 그 순간 잃어버린 세그먼트의 수신을 알리기 위해 새로운 ACK(중복되지 않은)가 도착한다. Reno TCP는 혼잡 회피 상태로 전환하게 된다.
- 하지만 첫 번째로 빠른 회복(fast recovery) 상태를 무시하고 혼잡 윈도를 이전 트랙으로 다시 이동하는 것과 같이 6MSS로 줄인다.

그림 9.50: Reno TCP의 예제

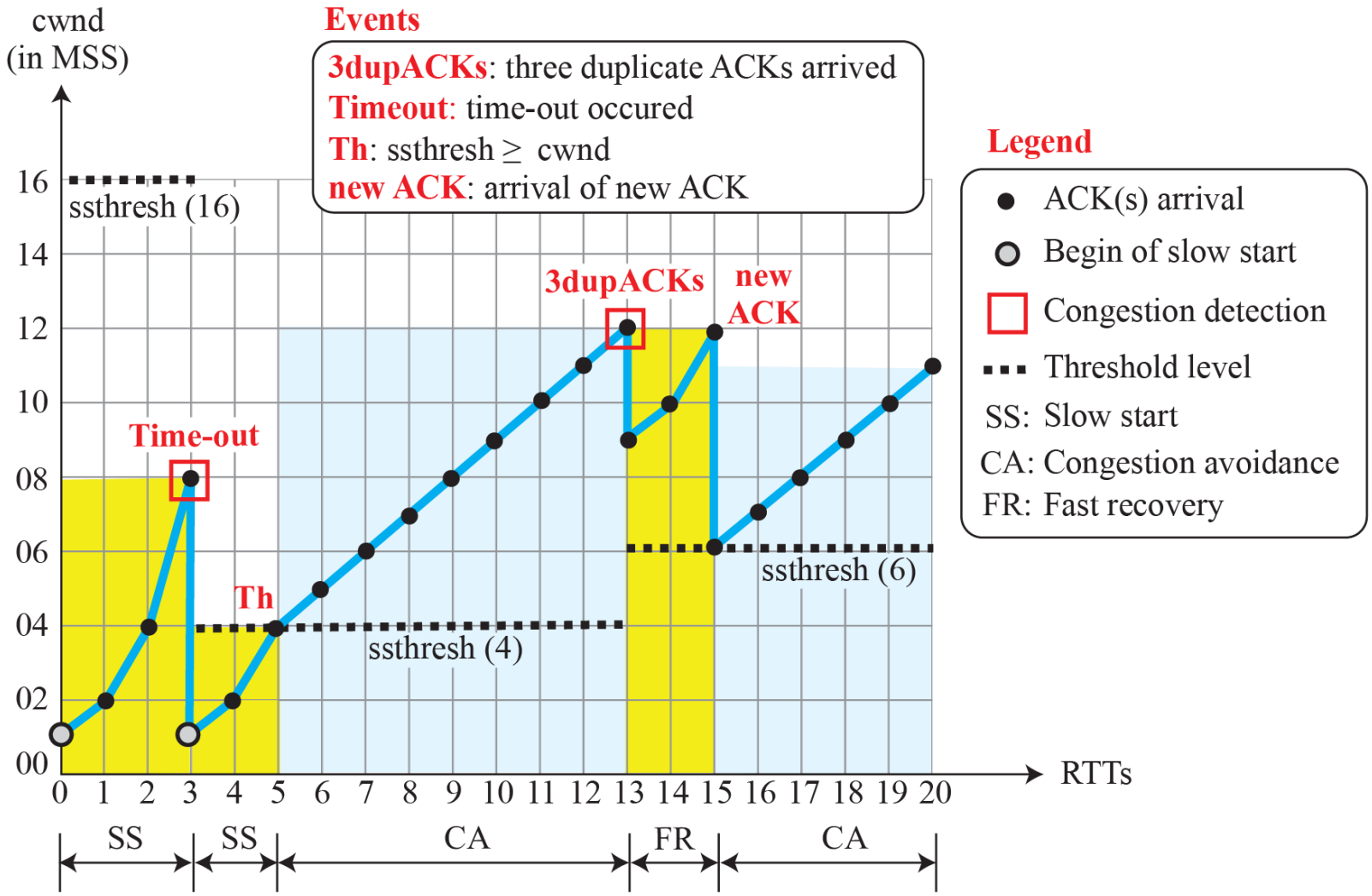
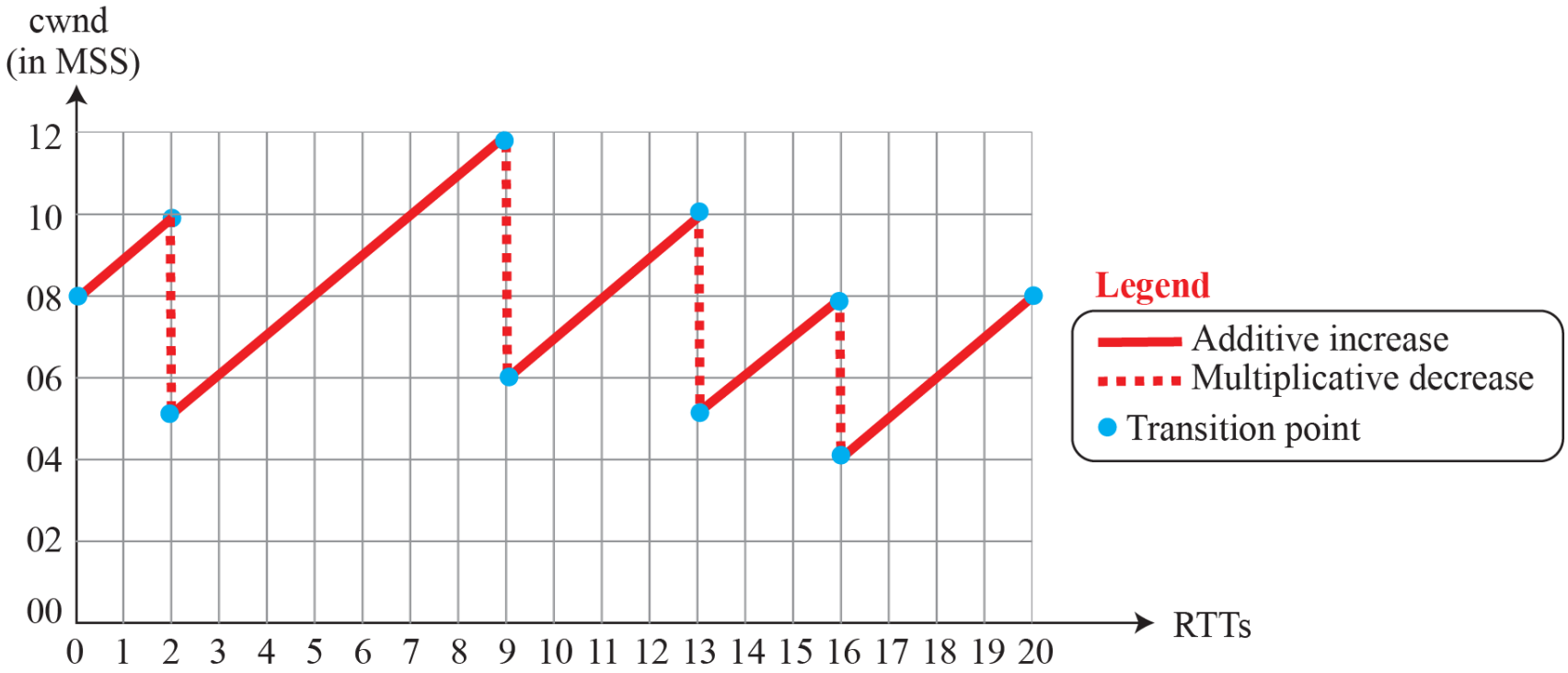


그림 9.51: 가산 증가(additive increase), 지수 감소(multiplicative decrease)



TCP 처리량(Throughput)

- 혼잡창(congestion window) 동작을 기반으로 하는 TCP의 처리량은 cwnd가 RTT의 상수 함수인 경우 쉽게 찾을 수 있다.
- 이 비현실적인 가정의 처리량은 처리량 = cwnd / RTT 이다.

Throughput = (0.75) Wmax / RTT

*** Wmax는 혼잡시의 평균 window크기이다.**

예제9.12

- 만약 그림 9.51와 같이 MSS=10KB이고, RTT=100ms이면, 아래와 같이 처리량을 계산할 수 있다.

$$W_{max} = (10 + 12 + 10 + 8 + 8) / 5 = 9.6 \text{ MSS}$$

$$\text{Throughput} = (0.75\ W_{max} / \text{RTT}) = 0.75 \times 960 \text{ kbps} / 100 \text{ ms} = 7.2 \text{ Mbps}$$

9.4.10 TCP 타이머

- TCP의 동작을 순조롭게 수행하기 위하여 대부분의 TCP 구현은 적어도 4개의 타이머, 즉 재전송(retransmission), 지속(persistence), 킵얼라이브(keepalive), 시간-기다림(TIME-WAIT)을 이용한다.
  - 재전송 타이머(Retransmission timer) : Retransmission time-out 운영
  - Persistence timer : deadlock상태를 정정하기 위해 사용; probe 세그먼트 사용
  - Keepalive timer : long idle connect을 방지하기 위해 사용 (보통 2시간)
  - Time-wait timer : 세그먼트가 네트워크상에 존재하는 시간(30초, 1분, 2분 중 하나로 셋팅

RTT

- Round Trip Time(RTT)
  - 재전송 time-out(RTO)을 계산하기 위해, 먼저 round-trip time(RTT)을 계산
  - Measured RTT (RTT_M) : 세그먼트전송후 확인응답(ACK) 수신까지 걸리는 시간
  - Smoothed RTT (RTT_S) : Weighted average of RTT_M and previous RTT_S

Original → No Value

After first measurement → $RTT_S = RTT_M$

After any other measurement → $RTT_S = (1- \alpha)\ RTT_S + \alpha \cdot RTT_M$

*value of α is implementation-dependent, but it is normally set to 1/8

RTT

- RTT Deviation (RTT_D)
  - Original → No Value
  - After first measurement → $RTT_D = RTT_M/2$
  - After any other measurement

    → $RTT_D = (1- \beta)\ RTT_D + \beta \cdot | RTT_S - RTT_M |$

    - The value of β is also implementation dependent, but is it is usually is sent to ¼.
- Retransmission Timeout(RTO)

  Original → Initial Value

  After any measurement → $RTO = RTT_S + 4\ RTT_D$

예제 9.13

그림 9.52은 연결의 일부분을 보여준다. 그림에서는 연결 설정과 데이터 전송 단계의 일부분을 보여준다.

1. SYN 세그먼트가 전송될 때에는 RTT_M, RTT_S, RTT_D 값은 설정되지 않는다. RTO의 값은(초기값인) 6.00초로 설정된다. 다음은 각 변수의 값들을 보여준다.

$$RTO = 6$$

예제9.13 (계속)

2. SYN+ACK 세그먼트가 도착하면 RTT_M이 측정되며 1.5초의 값을 가진다. 다음은 각 변수들의 값을 보여준다.

$$
\begin{aligned}
RTT_M &= 1.5 \\
RTT_S &= 1.5 \\
RTT_D &= (1.5) / 2 = 0.75 \\
RTO &= 1.5 + 4 \times 0.75 = 4.5
\end{aligned}
$$

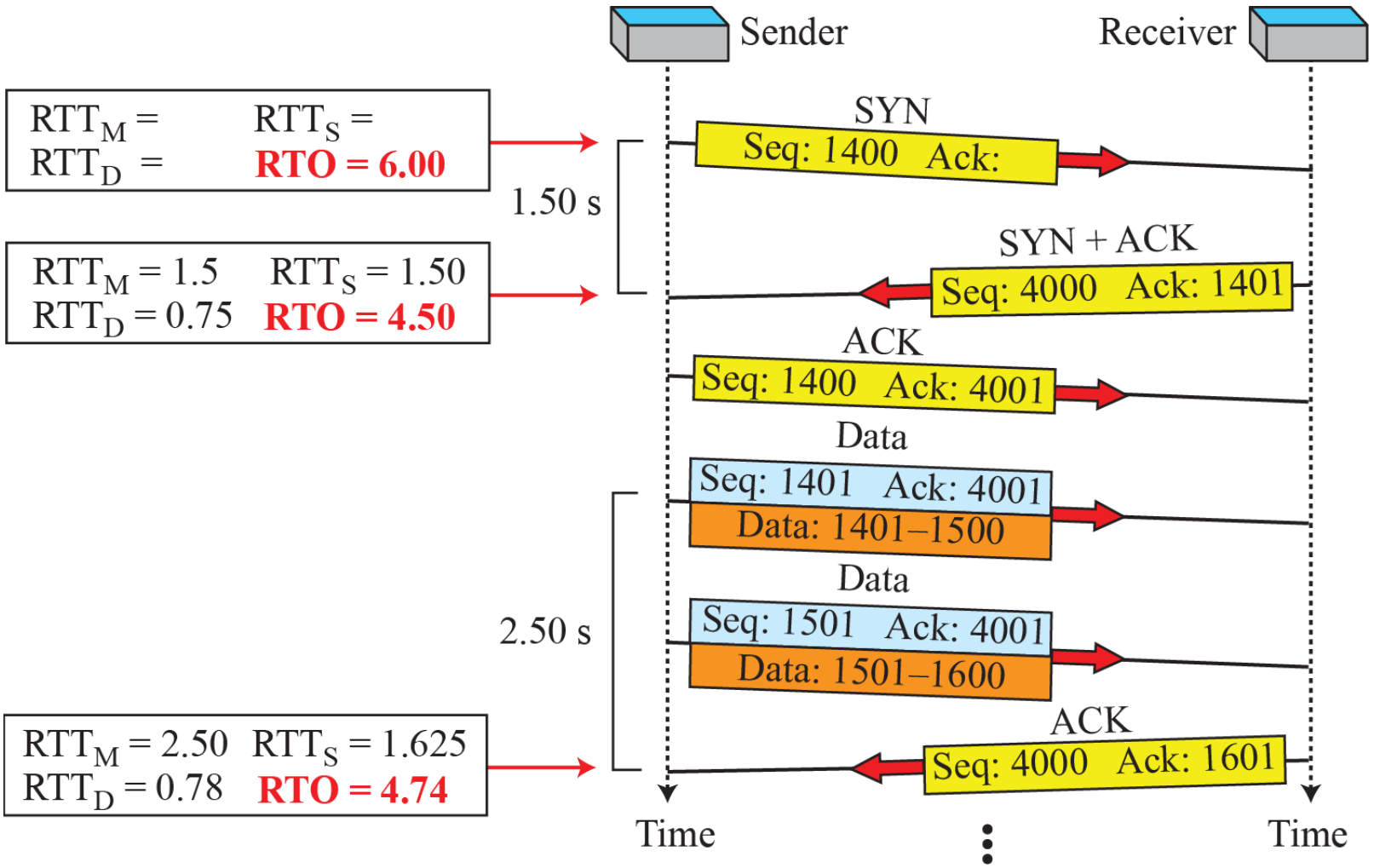
예제 9.13 (계속)

3.

- 첫 번째 데이터 세그먼트가 전송될 때 새로운 RTT 측정이 시작된다. 송신자가 ACK 세그먼트를 전송할 때에는 RTT 측정을 하지 않는다. 왜냐하면 ACK 세그먼트는 순서번호를 소비하지 않으며 또한 타임아웃 타이머도 설정되지 않기 때문이다.
- 이미 측정이 진행 중이기 때문에 두 번째 데이터 세그먼트에 대해서도 RTT 측정을 하지 않는다.
- ACK 세그먼트가 도착하면 RTT_M를 계산한다. 비록 ACK 세그먼트의 수신이 2개의 세그먼트를 누적해서 확인응답하지만, 이 세그먼트가 도착하면 첫 번째 세그먼트에 대한 RTT_M 값을 계산한다. 각 변수들의 값은 다음과 같이 계산된다.

$$RTT_M = 2.5$$
$$RTT_S = 7/8 \times 1.5 + (1/8) \times 2.5 = 1.625$$
$$RTT_D = 3/4\ (7.5) + (1/4) \times |1.625 - 2.5| = 0.78$$
$$RTO = 1.625 + 4 \times 0.78 = 4.74$$

그림 9.52: 예제 9.13

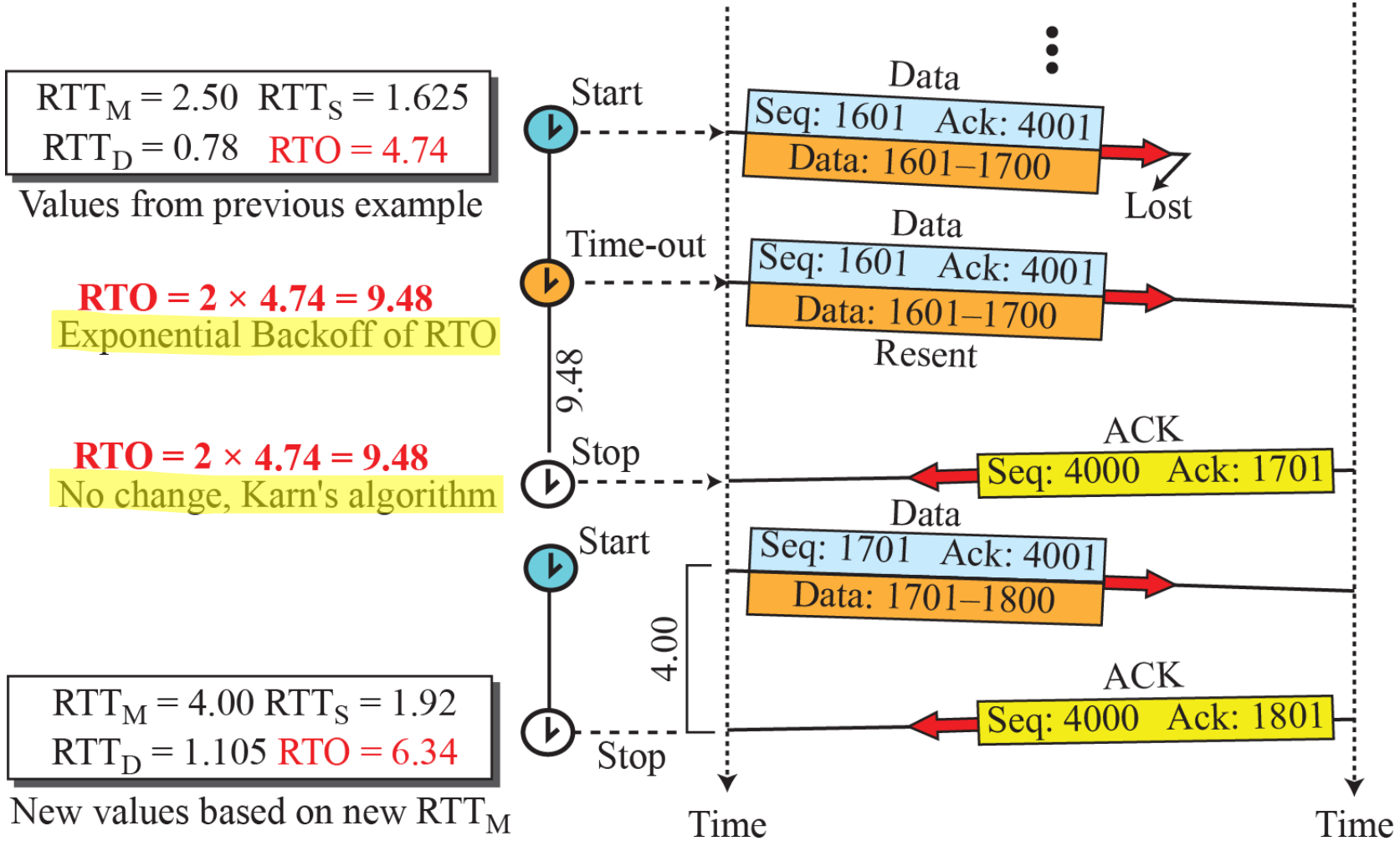

Sender
Receiver
RTT_M =
RTT_S =
RTT_D =
RTO = 6.00
SYN
Seq: 1400 Ack:
1.50 s
RTT_M = 1.5
RTT_S = 1.50
RTT_D = 0.75
RTO = 4.50
SYN + ACK
Seq: 4000 Ack: 1401
ACK
Seq: 1400 Ack: 4001
Data
Seq: 1401 Ack: 4001
Data: 1401–1500
2.50 s
Data
Seq: 1501 Ack: 4001
Data: 1501–1600
RTT_M = 2.50
RTT_S = 1.625
RTT_D = 0.78
RTO = 4.74
ACK
Seq: 4000 Ack: 1601
Time
Time

예제 9.14

- 그림 9.53에서는 앞선 예제의 연속으로 세그먼트가 재전송되어서 카른(Karn) 알고리즘*이 적용된 예를 보여준다.
- 이 그림에서 첫 번째 세그먼트는 전송 도중 손실되었다. RTO 타이머는 4.74초 이후에 만료된다. 세그먼트는 재전송되고, RTO 타이머는 앞선 RTO 값의 두 배인 9.48초로 설정된다.
- 시간 초과가 발생하기 전에 ACK를 수신하였다. 새로운 RTO 값을 계산하기 위해서는 새 세그먼트를 전송하고 ACK를 수신할 때까지 기다려야 한다.

* TCP는 새로운 RTO를 계산할 때 재전송된 세그먼트의 RTT를 고려하지 않는다.

그림 9.52: 예제 9.14


RTT_M = 2.50 RTT_S = 1.625
RTT_D = 0.78 RTO = 4.74
Values from previous example
RTO = 2 × 4.74 = 9.48
Exponential Backoff of RTO
RTO = 2 × 4.74 = 9.48
No change, Karn's algorithm
RTT_M = 4.00 RTT_S = 1.92
RTT_D = 1.105 RTO = 6.34
New values based on new RTT_M
Start
Data
Seq: 1601 Ack: 4001
Data: 1601–1700
Lost
Time-out
Data
Seq: 1601 Ack: 4001
Data: 1601–1700
Resent
9.48
Stop
ACK
Seq: 4000 Ack: 1701
Start
Data
Seq: 1701 Ack: 4001
Data: 1701–1800
4.00
ACK
Seq: 4000 Ack: 1801
Stop
Time
Time

9.4.11 옵션

- TCP 헤더에는 최대 40바이트의 옵션 정보가 있을 수 있다.
- 이러한 옵션 정보들은 목적지에게 부가 정보를 전달하거나 또는 다른 옵션의 정렬을 맞추기 위하여 사용된다.
- 이 옵션은 다른 참조를 위해 책 웹사이트에 포함된다.

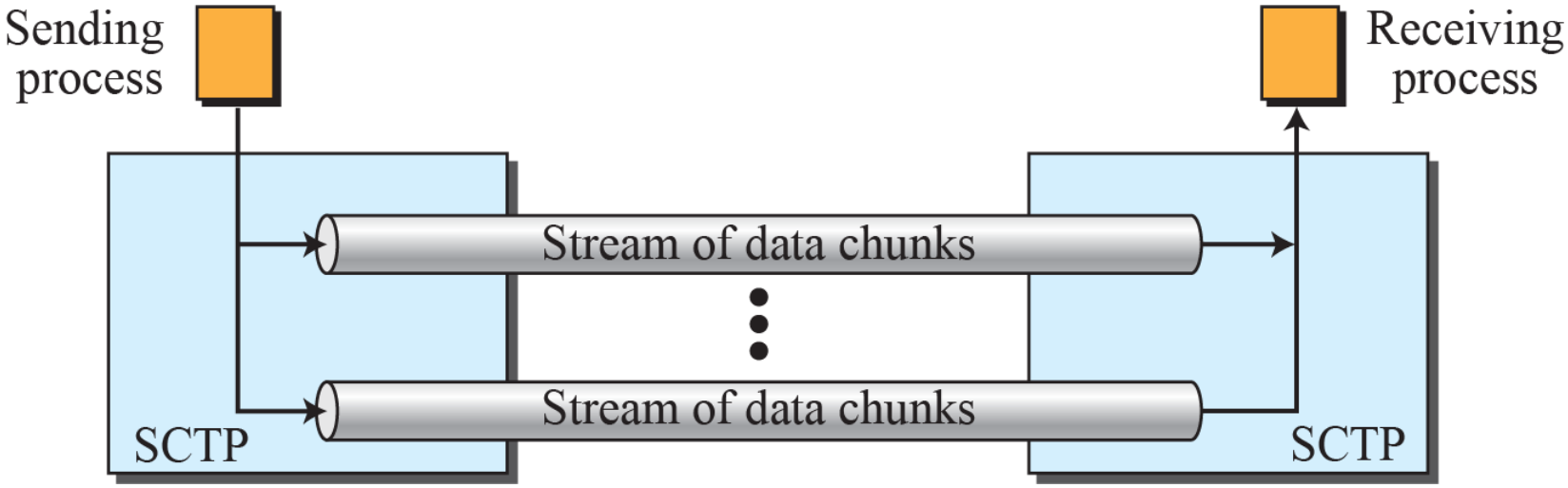
9.5 SCTP

- 스트림 제어 전송 프로토콜(SCTP, stream control transmission protocol)은 멀티미디어 통신을 위해 좀 더 좋은 프로토콜을 만들기 위해 UDP와 TCP의 일부 장점을 결합하여 설계된 새로운 전송층 프로토콜이다.

9.5.1 SCTP 서비스

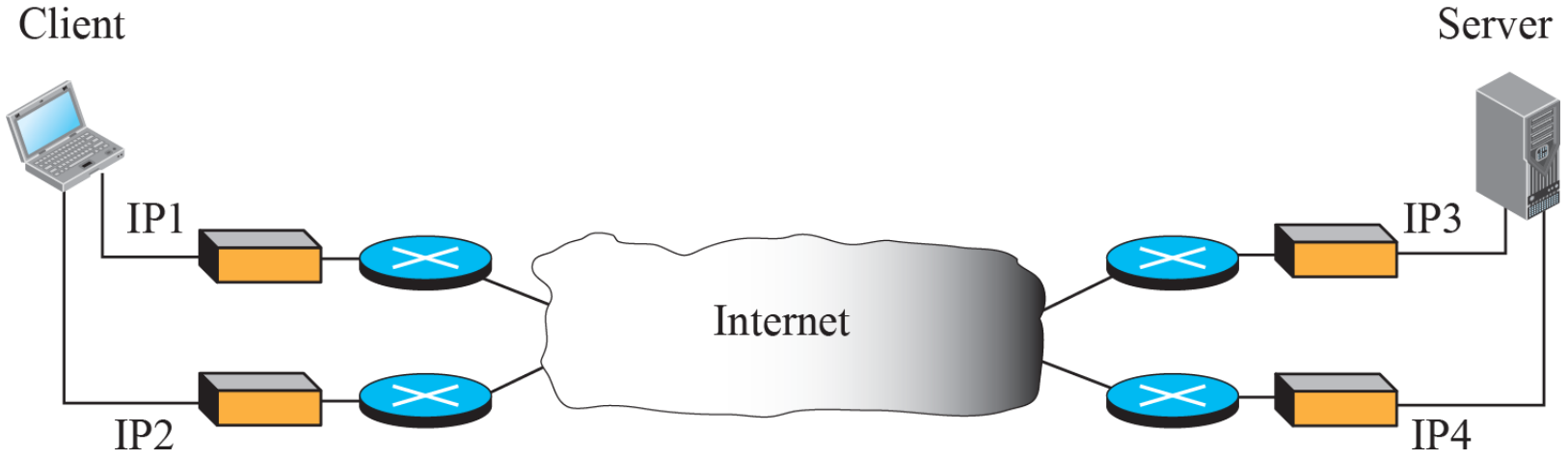
- SCTP 동작에 대하여 언급하기 전에 SCTP에 의해 응용층 프로세스에게 제공되는 서비스들에 대하여 설명한다.
- 프로세스-대-프로세스 통신
- 다중 스트림
- 멀티홈잉
- 전이중 통신
- 연결지향 서비스
- 신뢰성 있는 서비스

그림 9.54: 다중 스트림(Multiple Streams)의 개념



- 다중 스트림서비스: Association이라 부르며, 스트림중 1개가 블로킹 되더라도 다른 스트림들이 데이터를 전송할 수 있는 서비스를 의미

그림 9.55 : 멀티홈잉 (multihoming) 개념

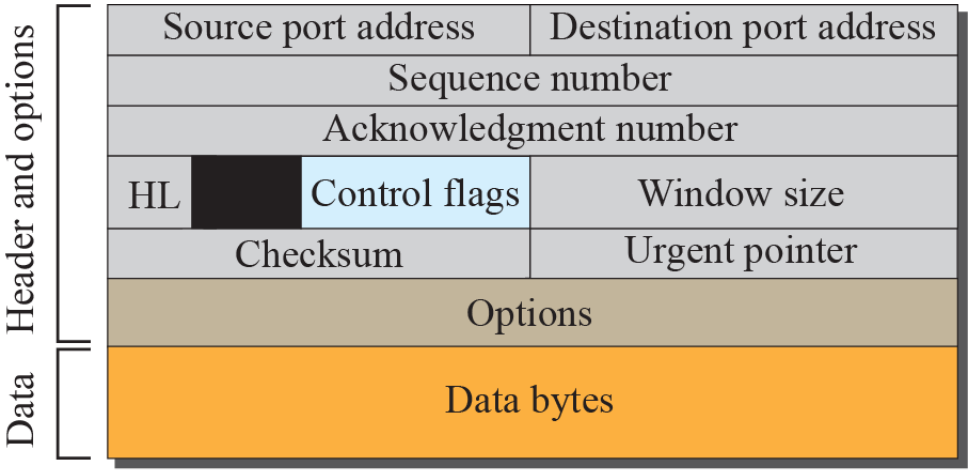


- 송신자와 수신자는 다중 IP주소를 갖고 있는 구성이다. 한 개 경로가 실패하더라도 다른 인터페이스는 다른 데이터 전송을 위해 사용될 수 있다.

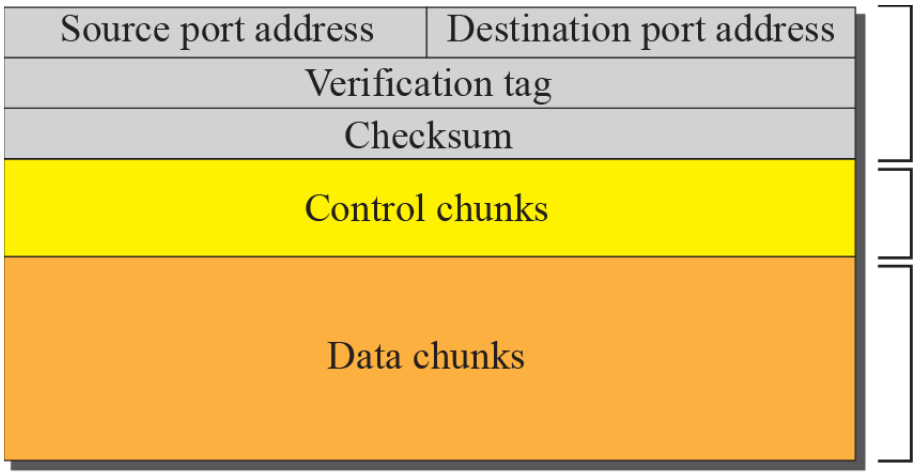
9.5.2 SCTP 특징

- SCTP의 일반적인 특징을 먼저 살펴보고 TCP의 특징과 비교해보자.
  - Transmission Sequence Number (TSN)
  - Stream Identifier (SI)
  - Stream Sequence Number (SSN)
    - 각 스트림의 데이터 청크를 표시
- SCTP에서 데이터 단위는 데이터 청크(Data Chunk)이다.

그림 9.56: TCP 세그먼트와 SCTP 패킷 비교

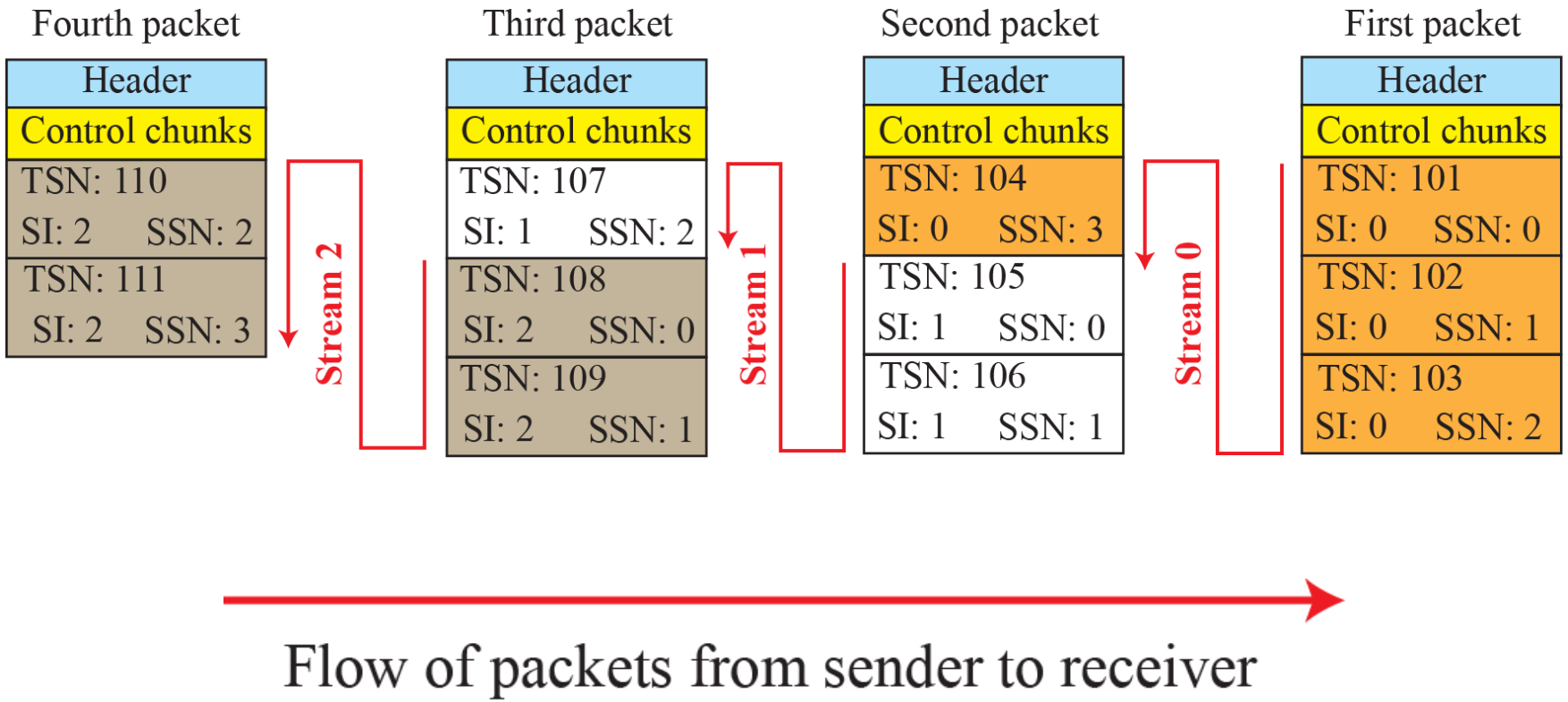


A segment in TCP



A packet in SCTP

그림 9.57: 패킷, 데이터 청크, 스트림



9.5.3 패킷 형식

- SCTP 패킷은 필수 항목의 일반 헤더와 청크로 불리는 블록 집합들을 가진다.
- 청크에는 제어 청크와 데이터 청크 두 가지 형태가 있다.
- 패킷에서 제어 청크는 데이터 청크 전에 온다.
- 그림 9.58는 일반적인 SCTP 패킷의 형식을 보여주고 있다.

그림 9.58 : SCTP 패킷 형식

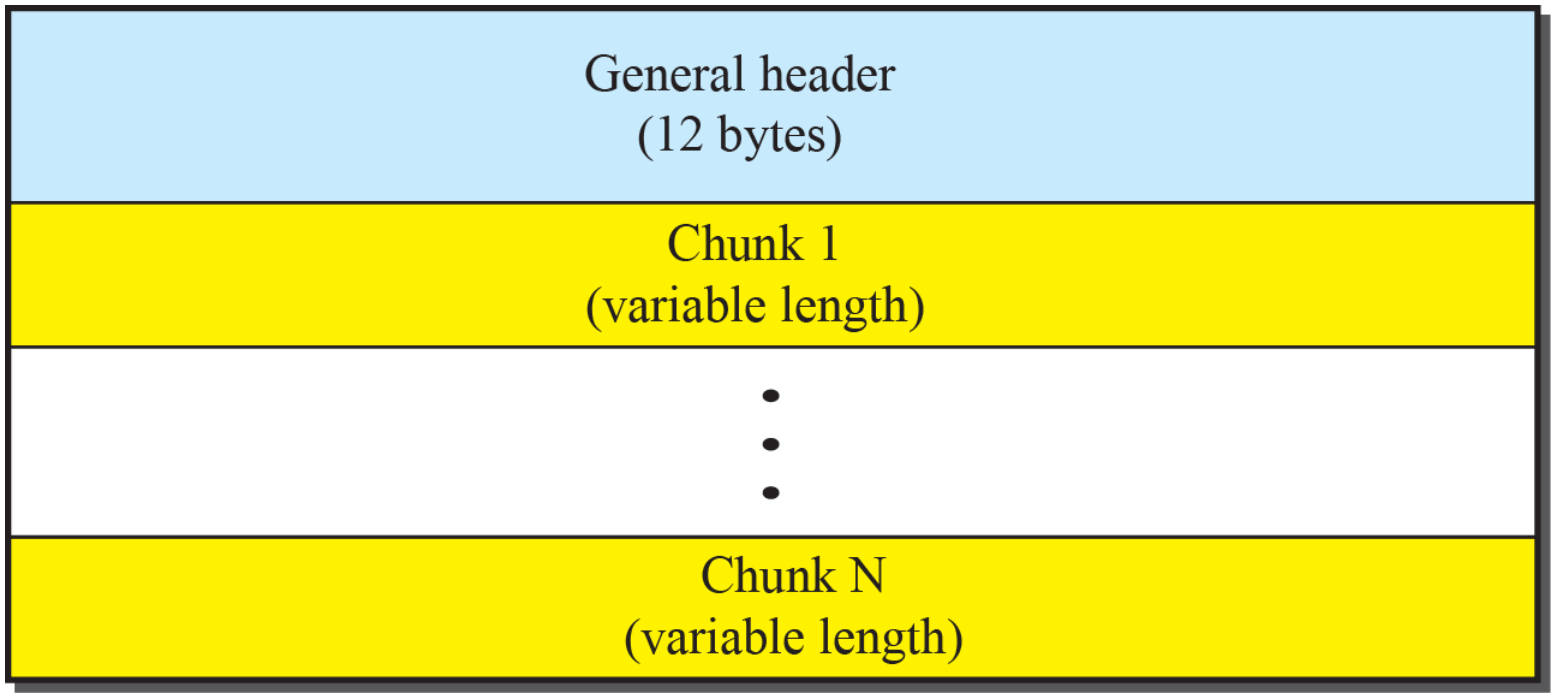
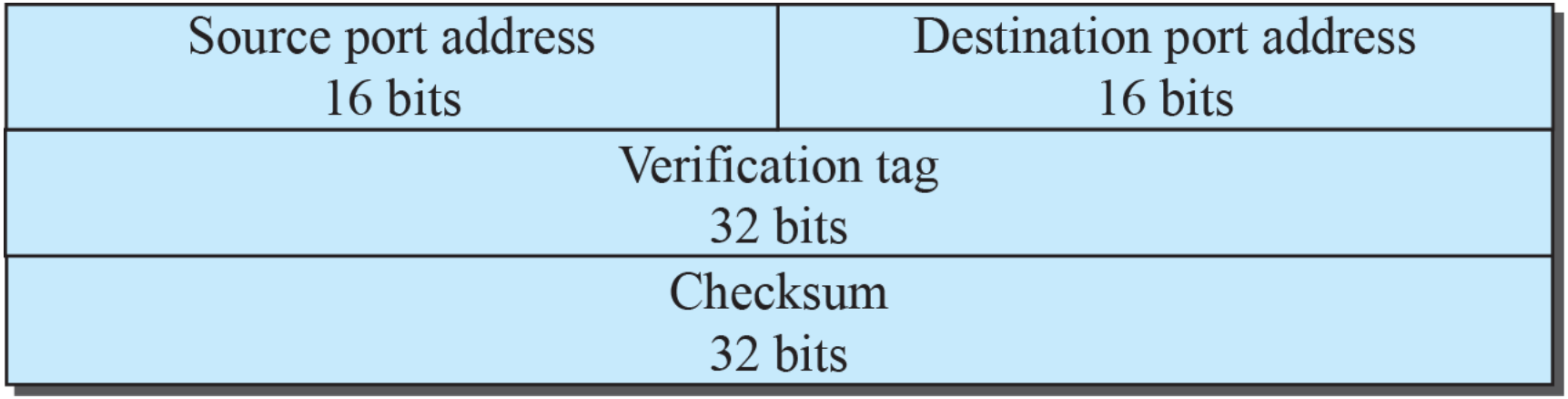
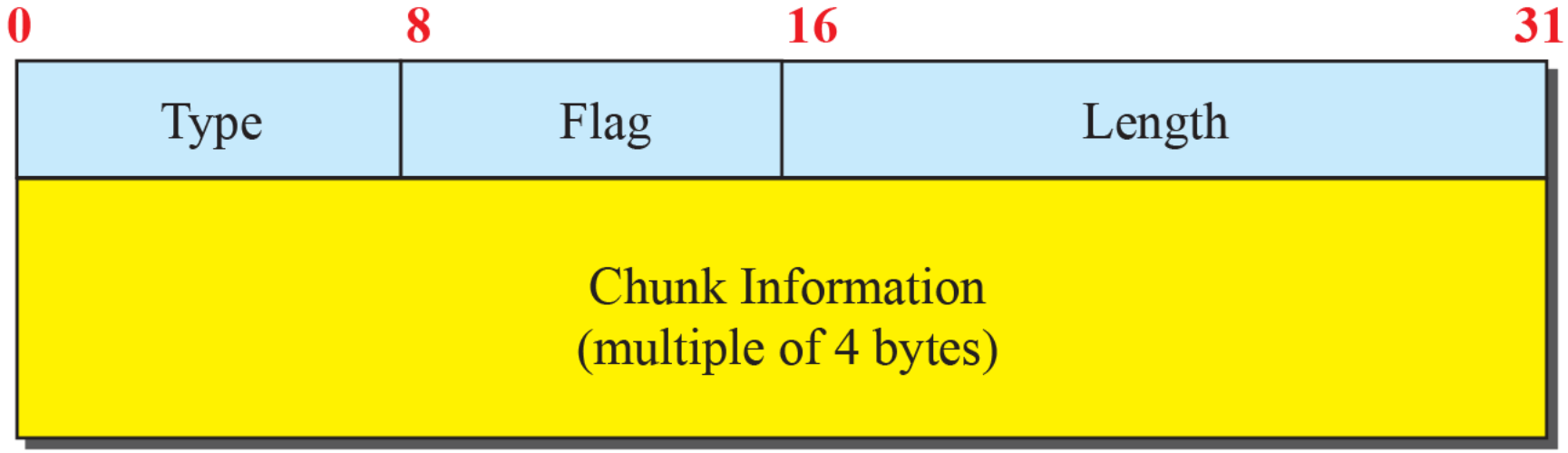


그림 9.59: 일반 헤더



- Verification tag: 특정 association에 패킷이 속함을 확인. 즉 association ID의 역할을 수행
- Checksum: CRC-32 사용

그림 9.60 : 청크의 공통 형식



■ 제어 정보 또는 사용자 데이터는 청크로 운반된다. 청크는 그림 9.60과 같이 공통 레이아웃을 가지고 있다. 처음 세 필드는 모든 청크에 공통이다. 정보 필드는 청크 유형에 따라 다르다. 유형 필드는 최대 256개의 청크 유형을 정의할 수 있습니다. 지금까지 정의된 것은 소수에 불과하다. 나머지는 향후 사용을 위해 예약되어 있다.

표 9.3: 청크

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	Aborts 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

9.5.4 SCTP 결합(Association)

- SCTP는 TCP처럼 연결지향 프로토콜이다.
- 그러나 SCTP에서는 멀티홈잉을 강조하기 위하여 결합(association)이라고 한다.

- 결합 설정
- 데이터 전송
  - ➲ 멀티홈잉 데이터 전송, 멀티스트림 전달, 단편화
- 결합 종료

그림 9.61: 네-방향 핸드쉐이킹(Four-way Handshaking)

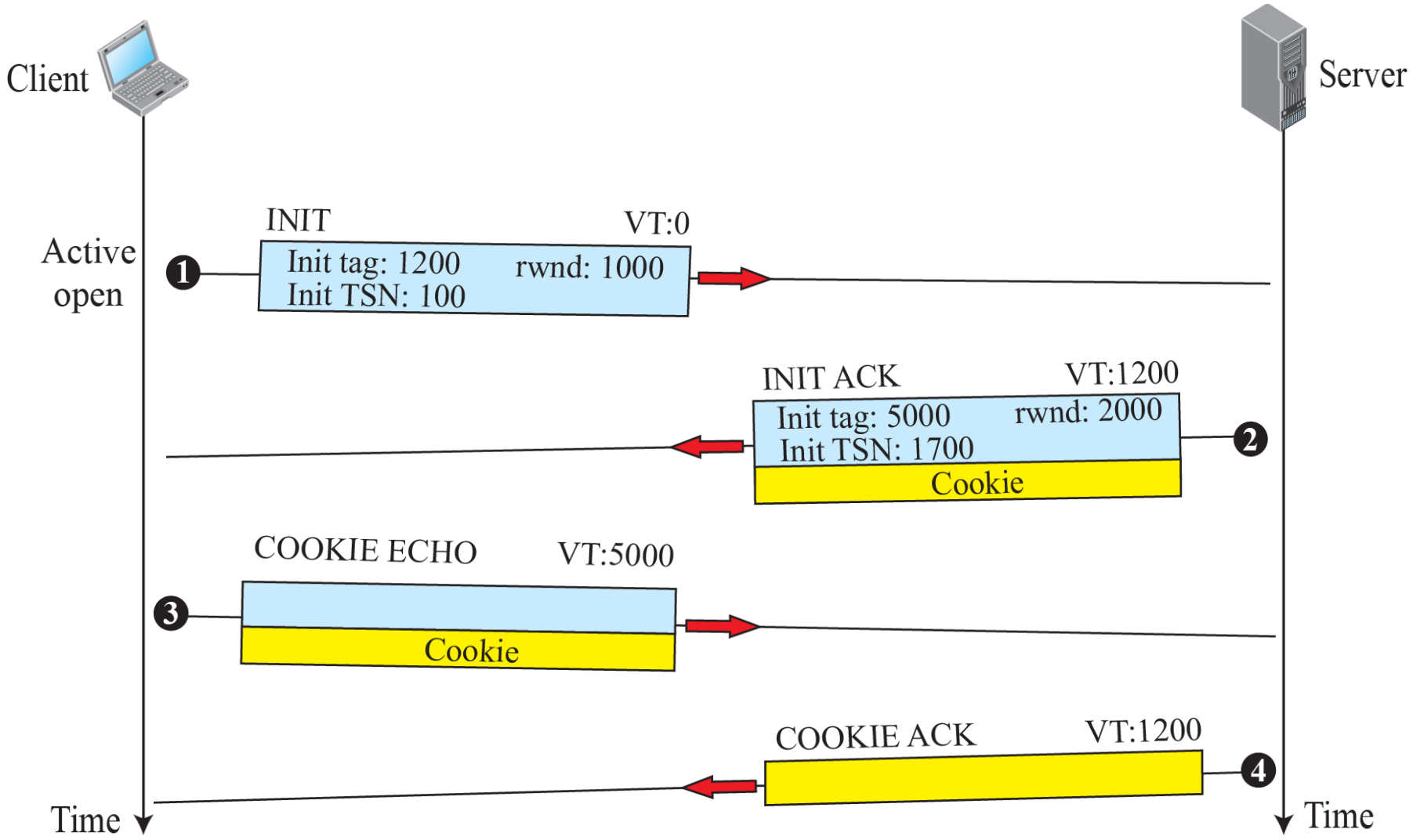
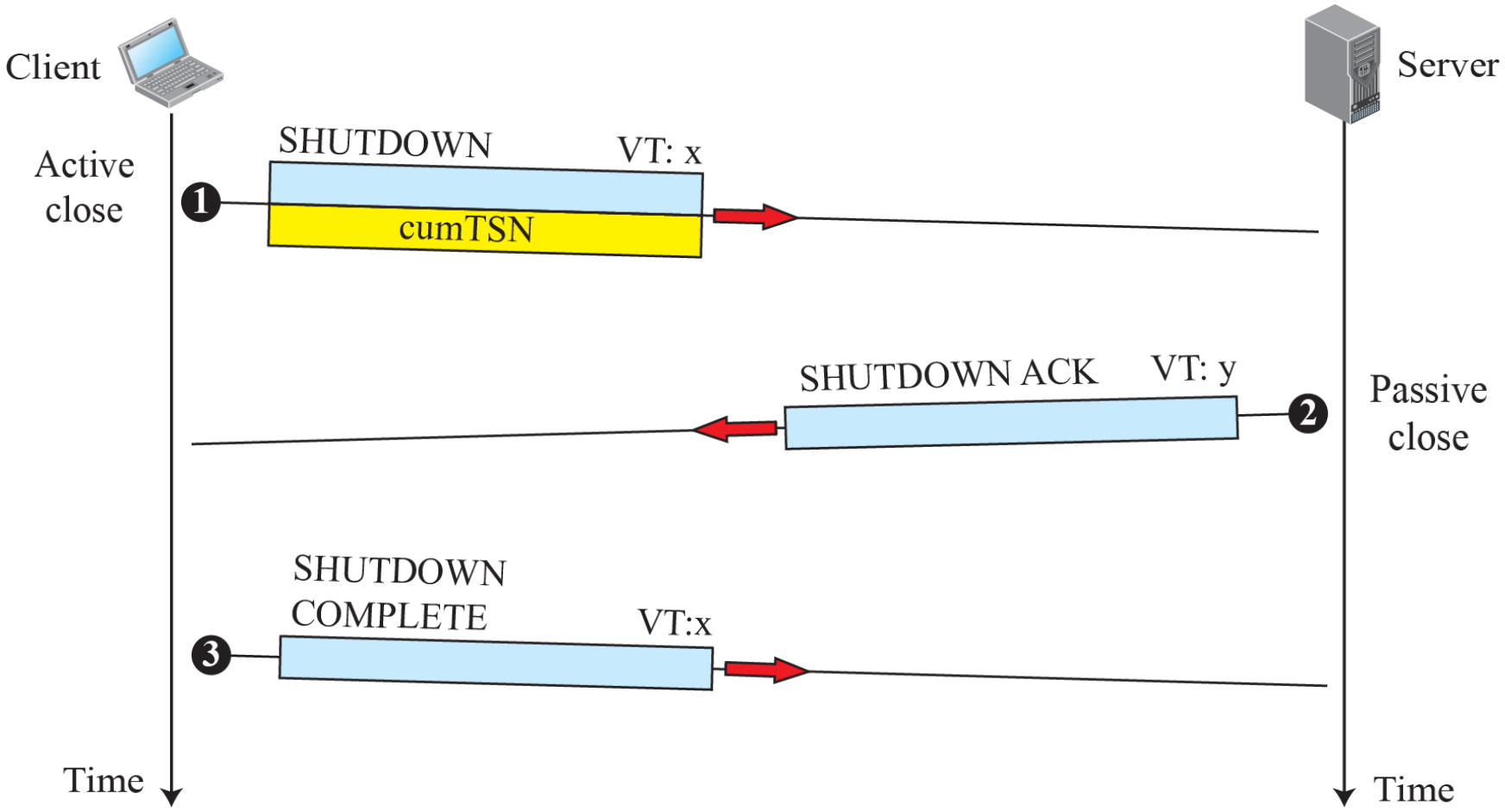


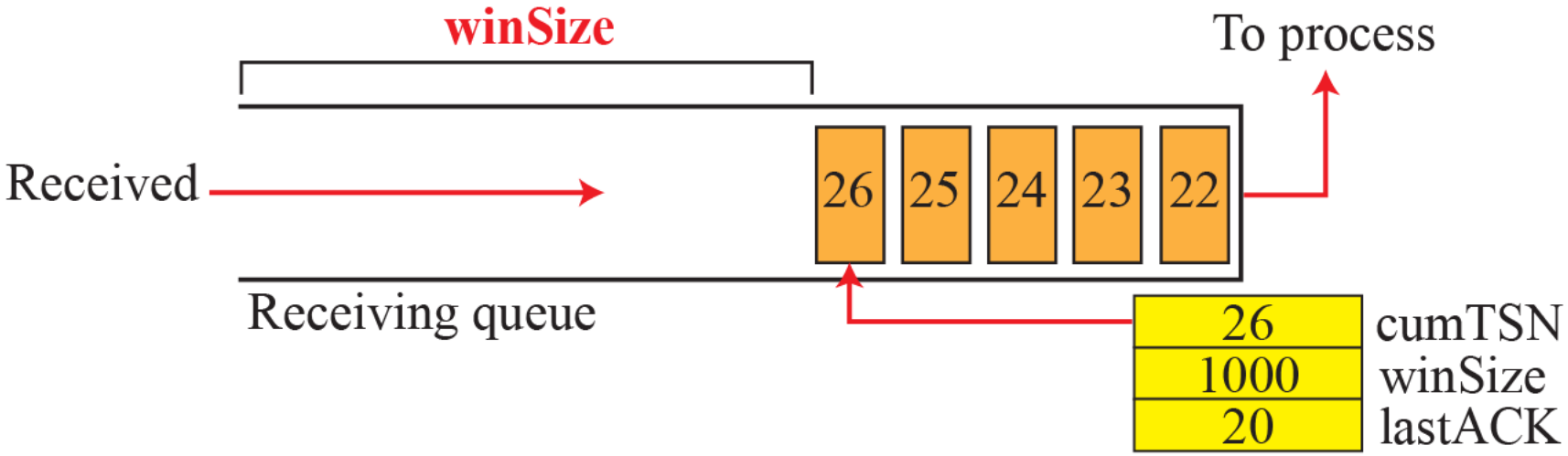
그림 9.62 : 결합 종료(Association Termination)



9.5.5 흐름 제어

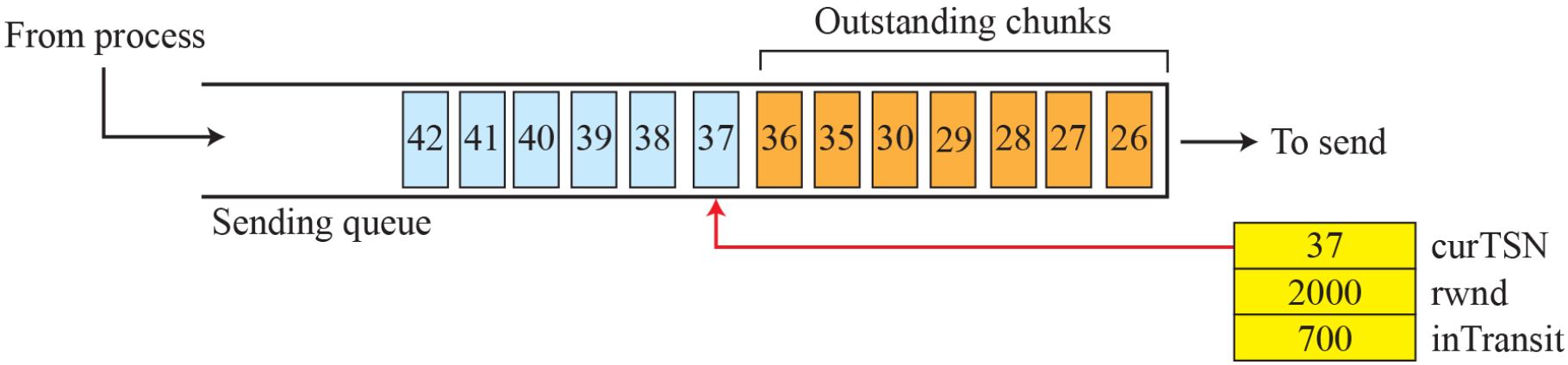
- SCTP에서의 흐름 제어(flow control)는 TCP와 유사하다.
- TCP에서는 하나의 데이터 단위, 즉 바이트를 다루어야 했다.
- SCTP에서는 2개의 데이터 단위인 바이트와 청크를 다루어야 한다. rwnd와 cwnd의 값은 바이트로 표현하고 TSN과 확인응답 값은 청크로 표현한다.
- 개념을 설명하기 위하여 실질적이지는 않지만 몇 가지 가정을 한다. 네트워크에 결코 혼잡이 없고 오류가 없다고 가정한다.

그림 9.63: 흐름 제어, 수신기 사이트



- 수신측에는 하나의 버퍼(대기열)와 세 개의 변수가 있다. 큐는 프로세스에서 아직 읽지 않은 수신된 데이터 청크를 보유한다. 첫 번째 변수는 마지막으로 수신된 TSN인 cumTSN을 보유한다. 두 번째 변수는 사용 가능한 버퍼 크기인 winSize를 보유한다. 세 번째 변수는 마지막 누적 확인응답인 lastACK를 보유한다. 그림 9.63은 수신자 사이트의 대기열과 변수를 보여준다.

그림 9.64: 흐름 제어, 송신기 사이트

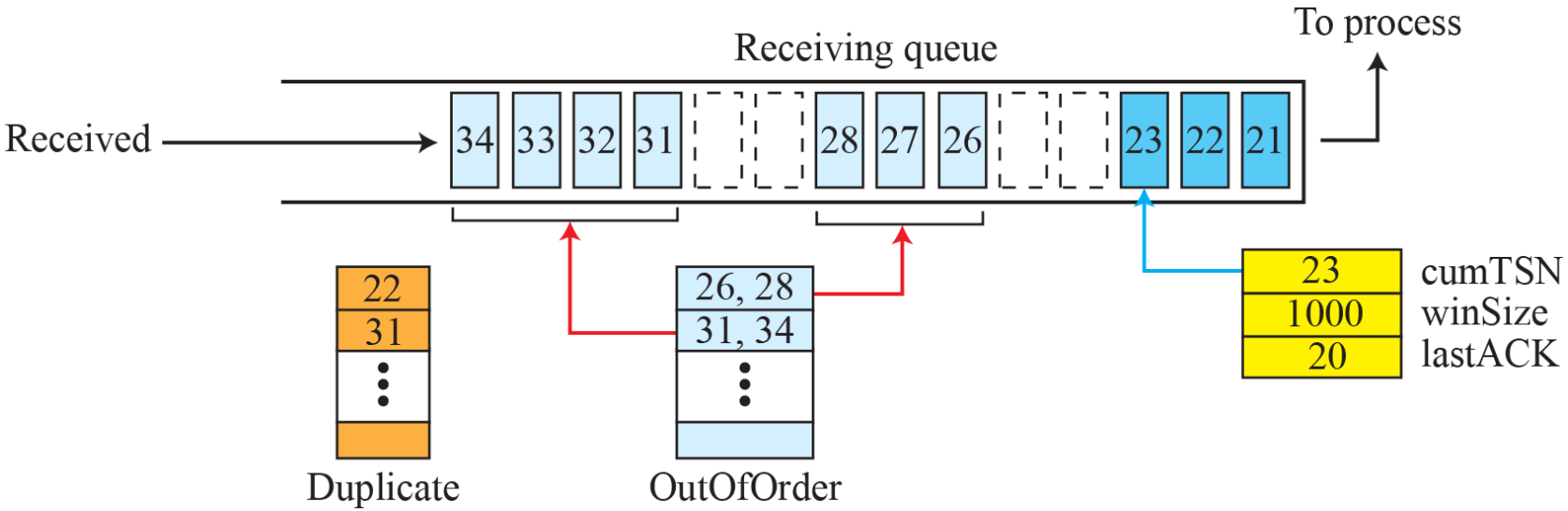


- 발신자는 하나의 버퍼(대기열)와 세 개의 변수(curTSN, rwnd 및 inTransit)를 가지고 있다(그림 9.64 참조). 각 청크의 길이가 100바이트라고 가정한다.

9.5.6 오류 제어

- SCTP는 TCP처럼 신뢰성 있는 전송층 프로토콜이다.
- 송신기에 수신기 버퍼의 상태를 보고하기 위하여 SACK (Selective ACK) 청크를 사용한다.
- 각 구현 방법에서는 수신기와 송신기 사이트를 위하여 서로 다른 형태의 엔티티와 타이머를 사용한다.
- 개념을 전달하기 위하여 간단한 디자인을 사용한다.

그림 9.65 : 오류 제어, 수신기 사이트



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

그림 9.66 : 오류 제어, 송신기 사이트

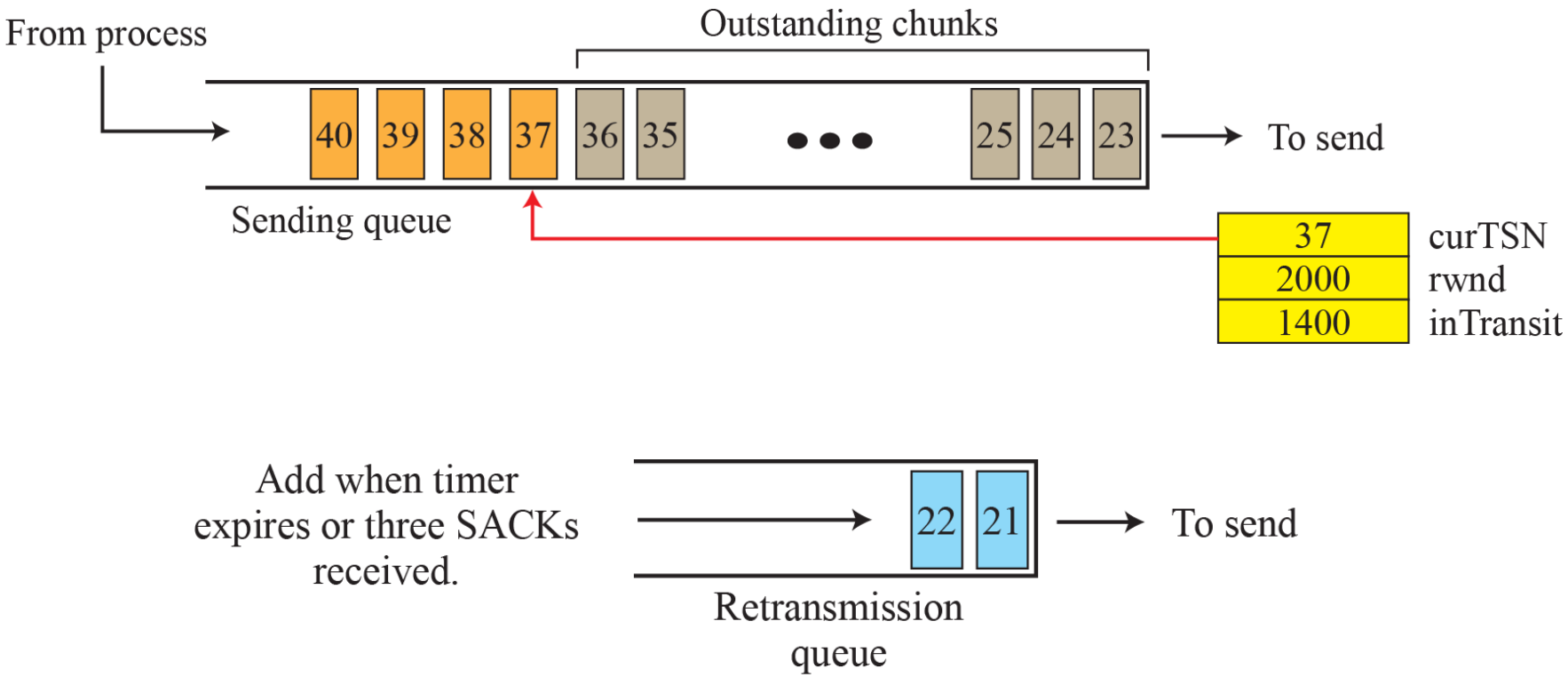
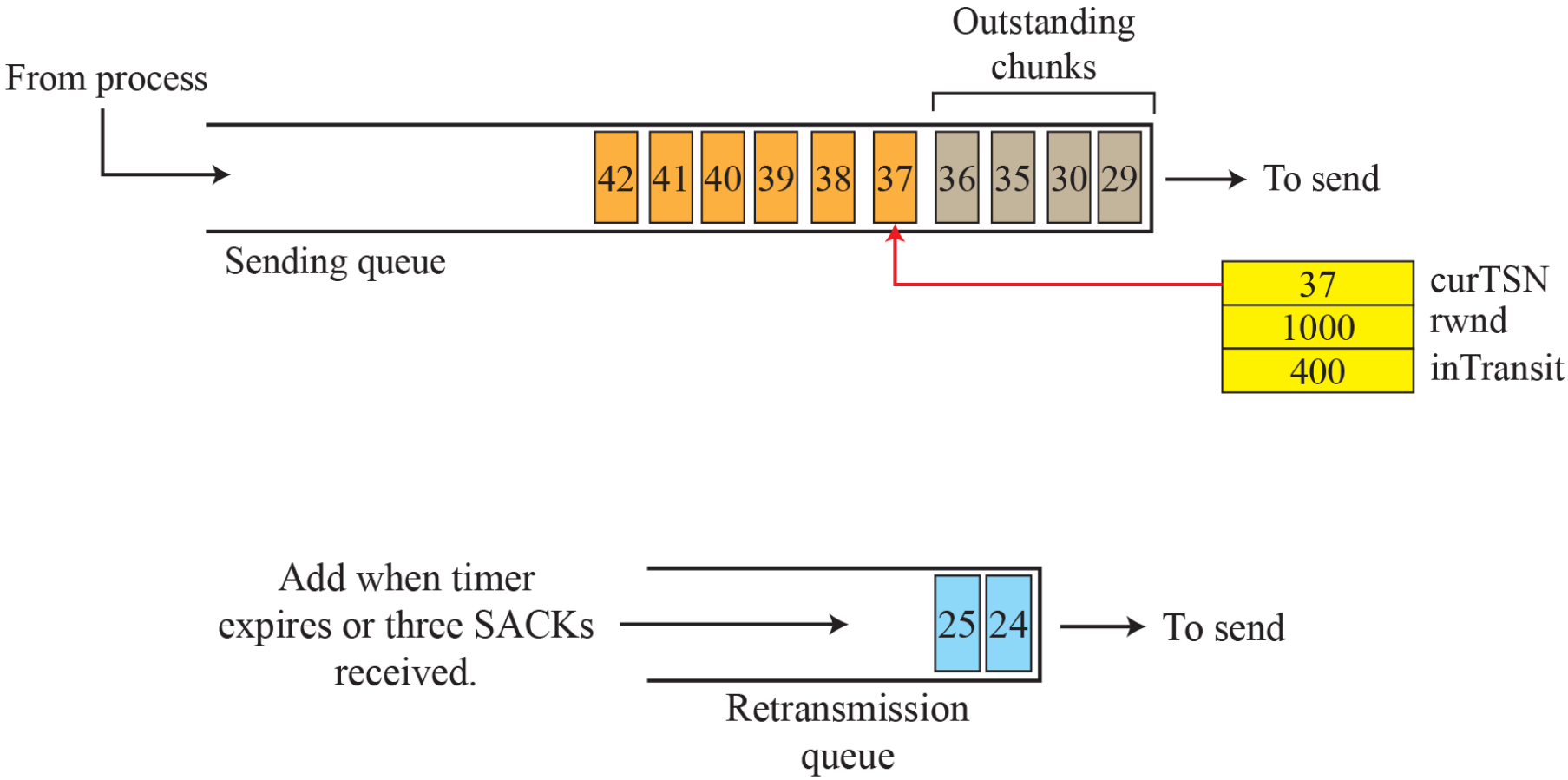


그림 9.67: SACK 청크를 수신한 후 송신 사이트의 새로운 상태



Q & A