컴퓨터 네트워크

- 트랜스포트 계층 1 -

순천향대학교 사물인터넷학과

목표

- 트랜스포트 계층 서비스의 원리 이해
  - 다중화/역다중화multiplexing/demultiplexing
  - 신뢰적인 데이터 전송reliable data transfer
  - 흐름 제어flow control
  - 혼잡 제어congestion control
- 인터넷 트랜스포트 계층 프로토콜 학습
  - UDP: 비연결형 전송connectionless transport
  - TCP: 연결지향형 전송connection-oriented transport
  - TCP 혼잡 제어congestion control

트랜스포트 계층

3.1 트랜스포트 계층 서비스

3.2 다중화 / 역다중화

3.3 비연결형 전송: UDP

3.4 신뢰적 데이터 전송의 원리

3.5 연결지향형 전송: TCP

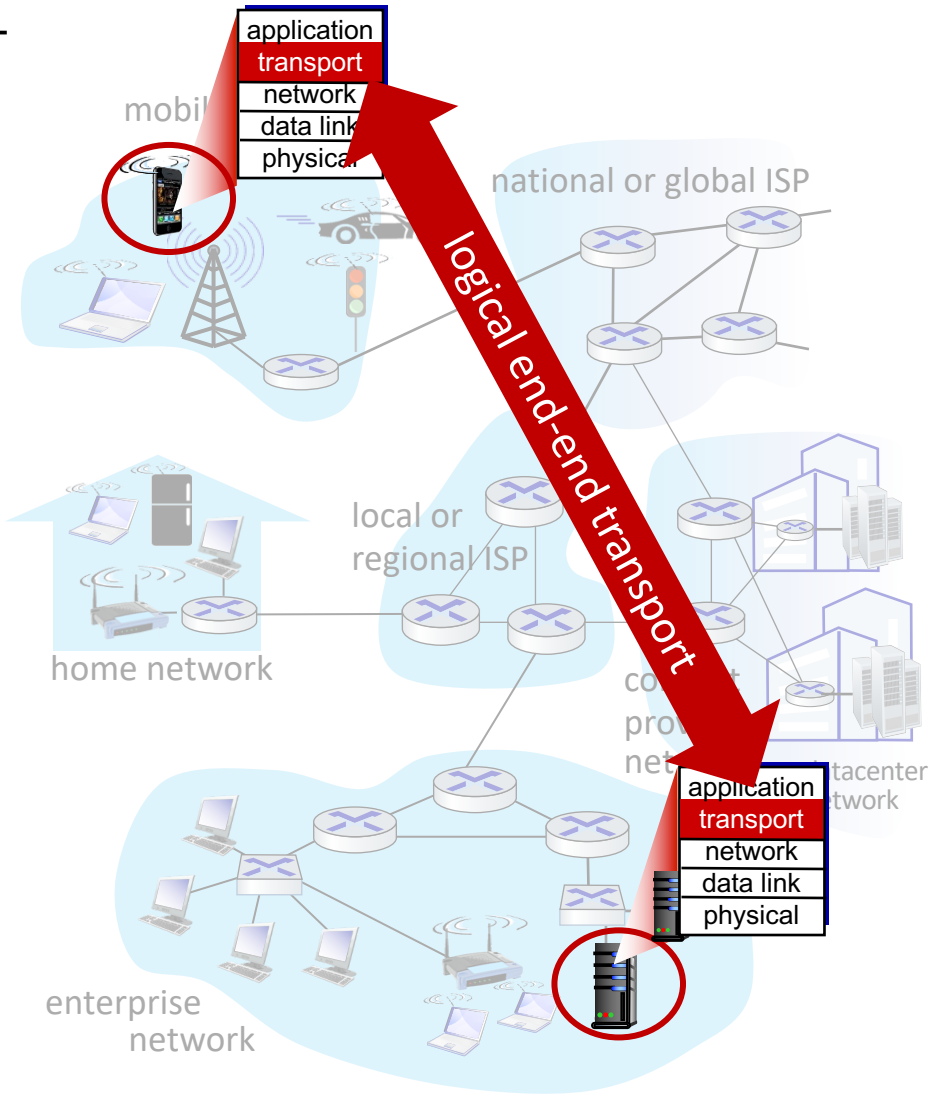
- 세그먼트 구조, 신뢰적 데이터 전송, 흐름 제어, 연결 관리

3.6 혼잡 제어의 원리

3.7 TCP 혼잡 제어

트랜스포트 계층 서비스 및 프로토콜

- 서로 다른 호스트에서 실행되는 애플리케이션 프로세스 간의 논리적 통신 제공
- 종단 시스템에 존재
  - 송신 측
    - ✓ 애플리케이션 메시지를 세그먼트(segment)로 분할하여 네트워크 계층으로 전달
  - 수신 측
    - ✓ 세그먼트를 메시지로 재조합하여 애플리케이션 계층으로 전달
- 인터넷 애플리케이션에 사용가능한 2개의 트랜스포트 계층 프로토콜
  - TCP, UDP



트랜스포트 계층 vs 네트워크 계층

- 네트워크 계층
  - 호스트 간 논리적 통신
- 트랜스포트 계층
  - 프로세스 간 논리적 통신
  - 네트워크 계층 서비스에 의존

편지 보내기 예제

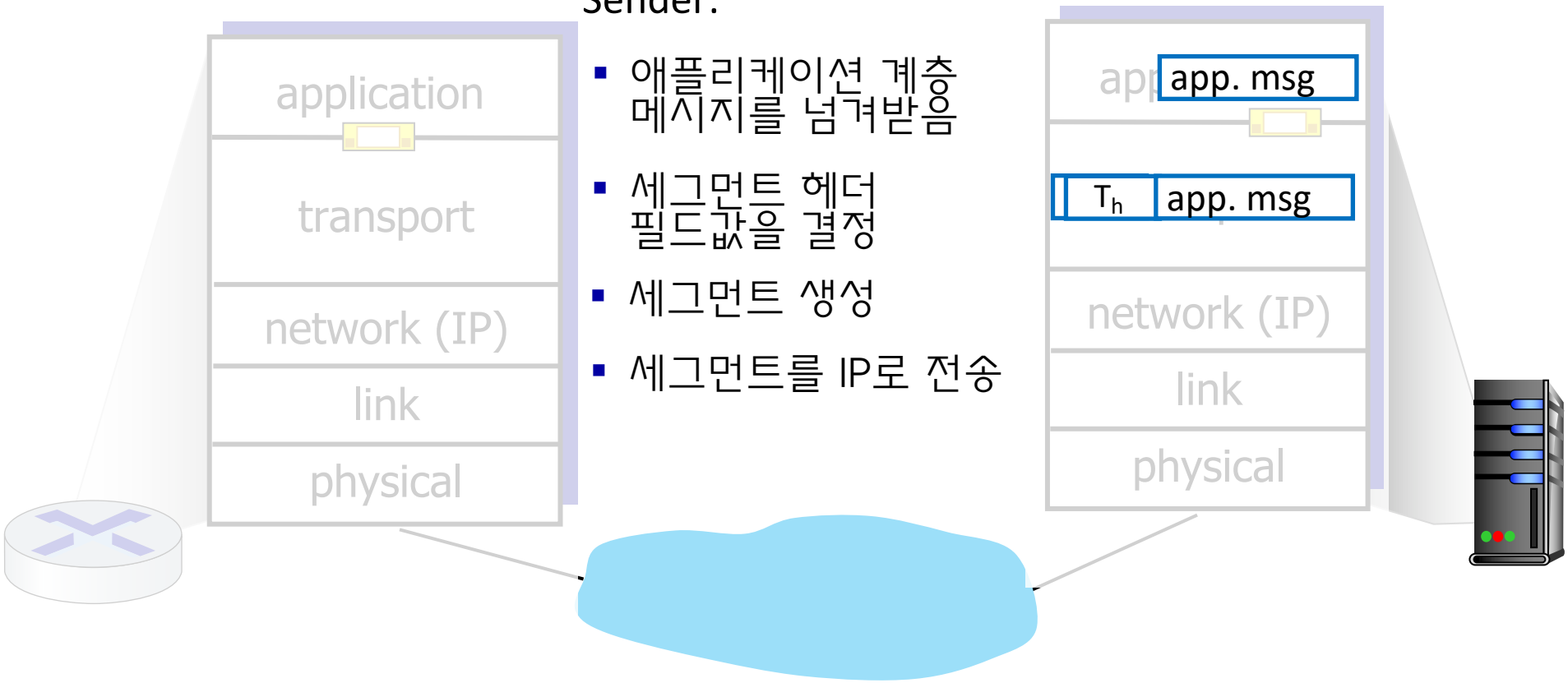
"앤"의 집의 12명의 아이가 "빌"의 집의 12명의 아이에게 편지를 보냄
"앤"과 "빌"이 각 집에서 편지를 아이들에게 나누어 줌

- 호스트 = 집
- 프로세스 = 아이들
- 애플리케이션 메시지 = 봉투 안의 편지
- 트랜스포트 프로토콜 = 아이들에게 편지를 나누어 주는 "앤"과 "빌"
- 네트워크 계층 프로토콜 = 우편 서비스

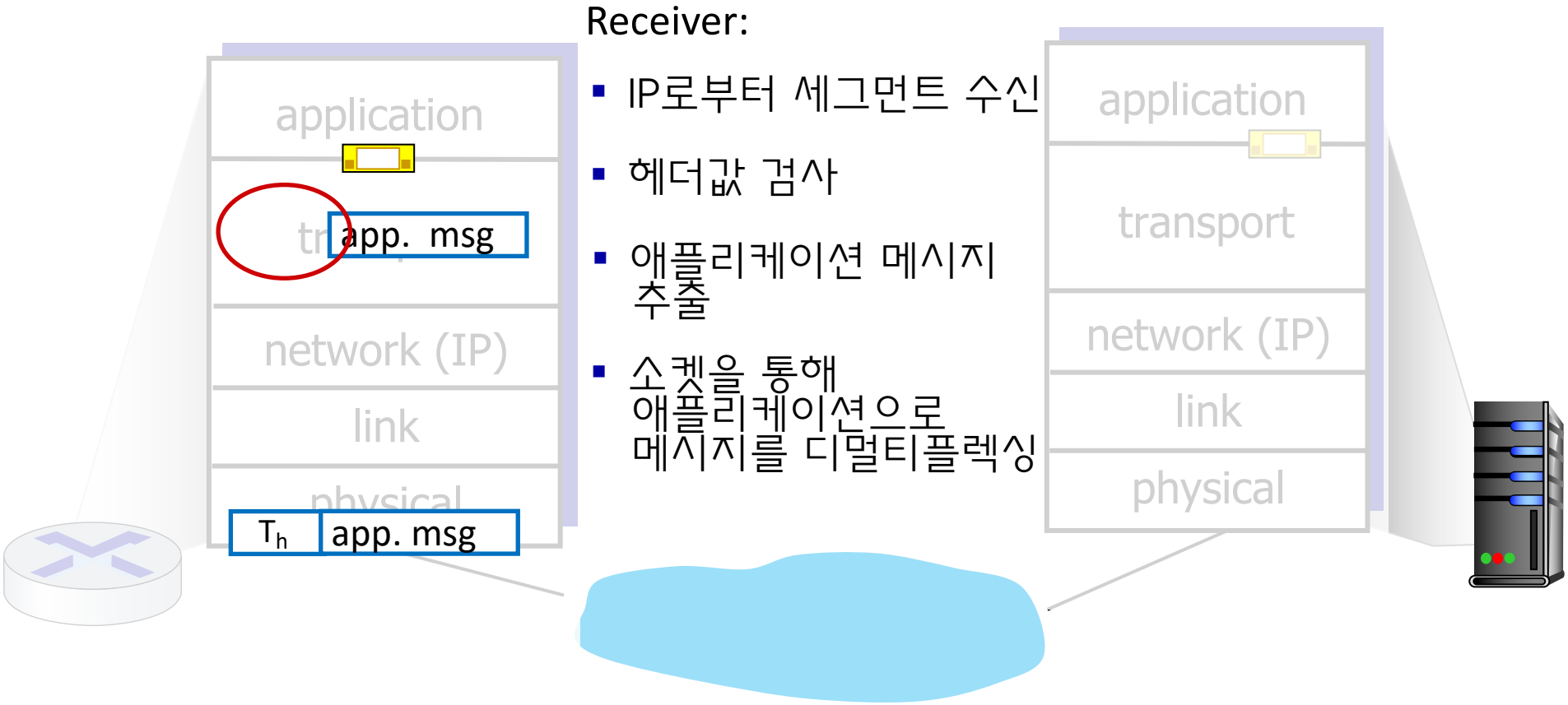
트랜스포트 계층 동작

Sender:

- 애플리케이션 계층 메시지를 넘겨받음
- 세그먼트 헤더 필드값을 결정
- 세그먼트 생성
- 세그먼트를 IP로 전송



트랜스포트 계층 동작



Receiver:

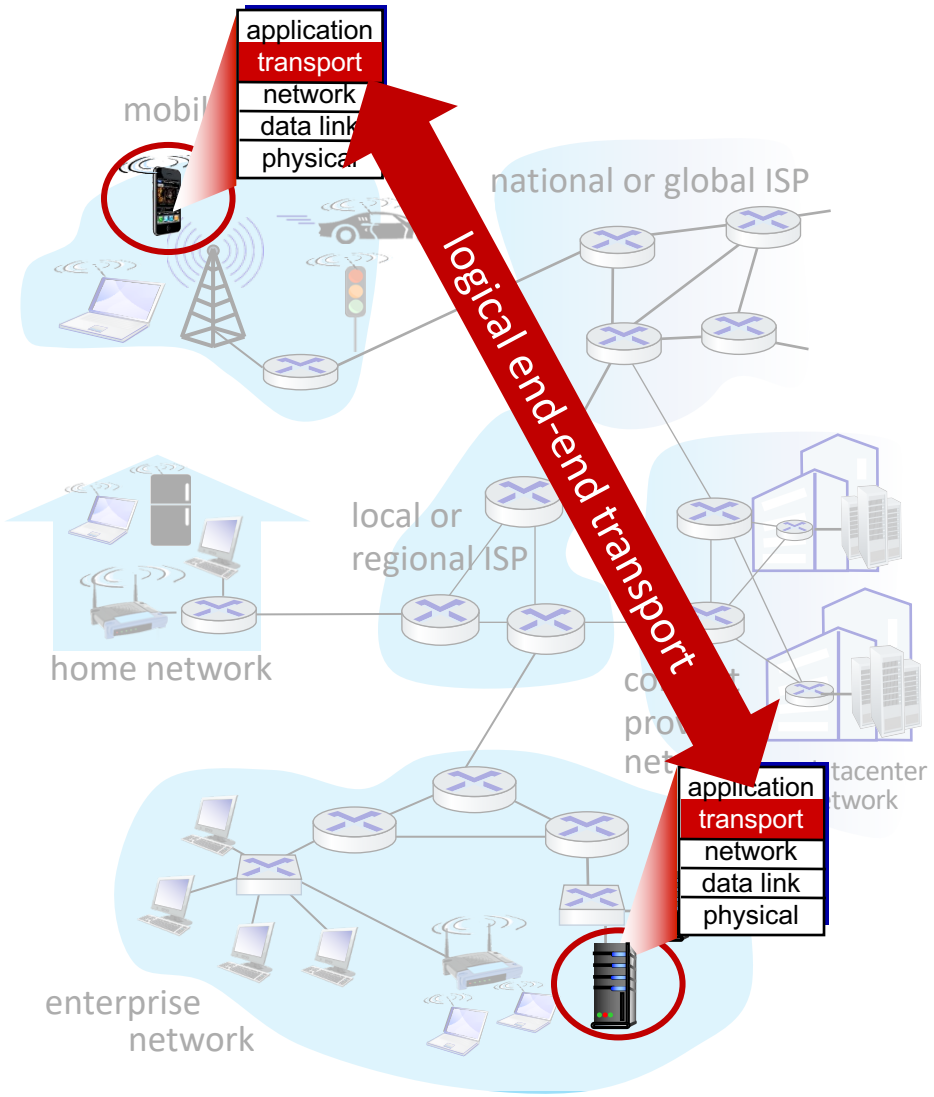
- IP로부터 세그먼트 수신
- 헤더값 검사
- 애플리케이션 메시지 추출
- 소켓을 통해 애플리케이션으로 메시지를 디멀티플렉싱

인터넷 트랜스포트 계층 프로토콜

- 신뢰적, 순차in-order 데이터 전달 (TCP)
  - 혼잡 제어congestion control
  - 흐름 제어flow control
  - 연결 설정connection setup

- 비신뢰적, 비순차out-of-order 데이터 전달 (UDP)
  - "최선(best-effort)" 서비스
    - ✓ 프로세스 간 데이터를 "최선(best-effort)"을 다해 전달하는 역할을 수행
      → 아무 것도 보장하지 않음

- 미제공 서비스
  - 지연 시간 보장
  - 대역폭(처리율) 보장



트랜스포트 계층

3.1 트랜스포트 계층 서비스

3.2 다중화 / 역다중화

3.3 비연결형 전송: UDP

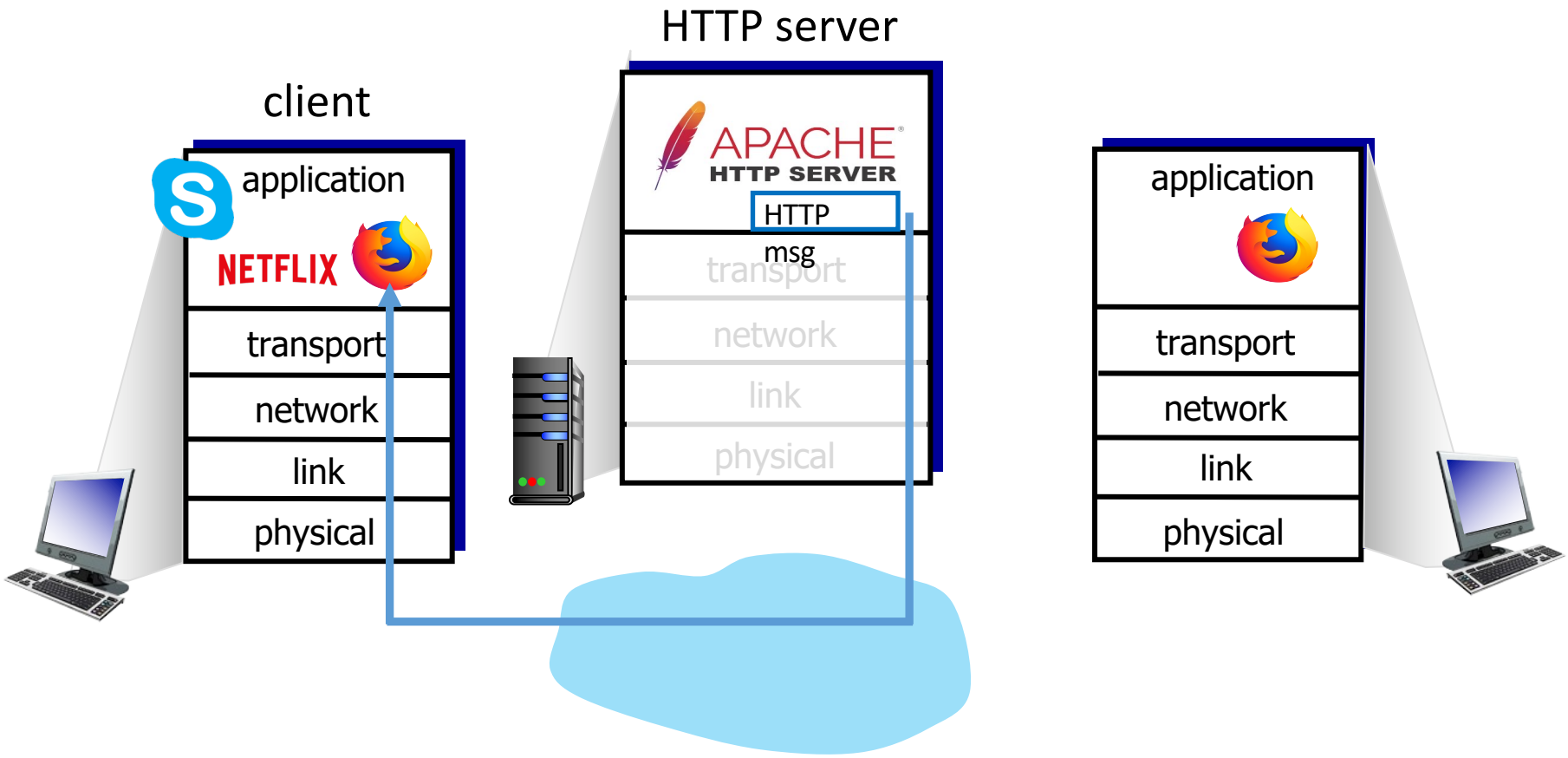
3.4 신뢰적 데이터 전송의 원리

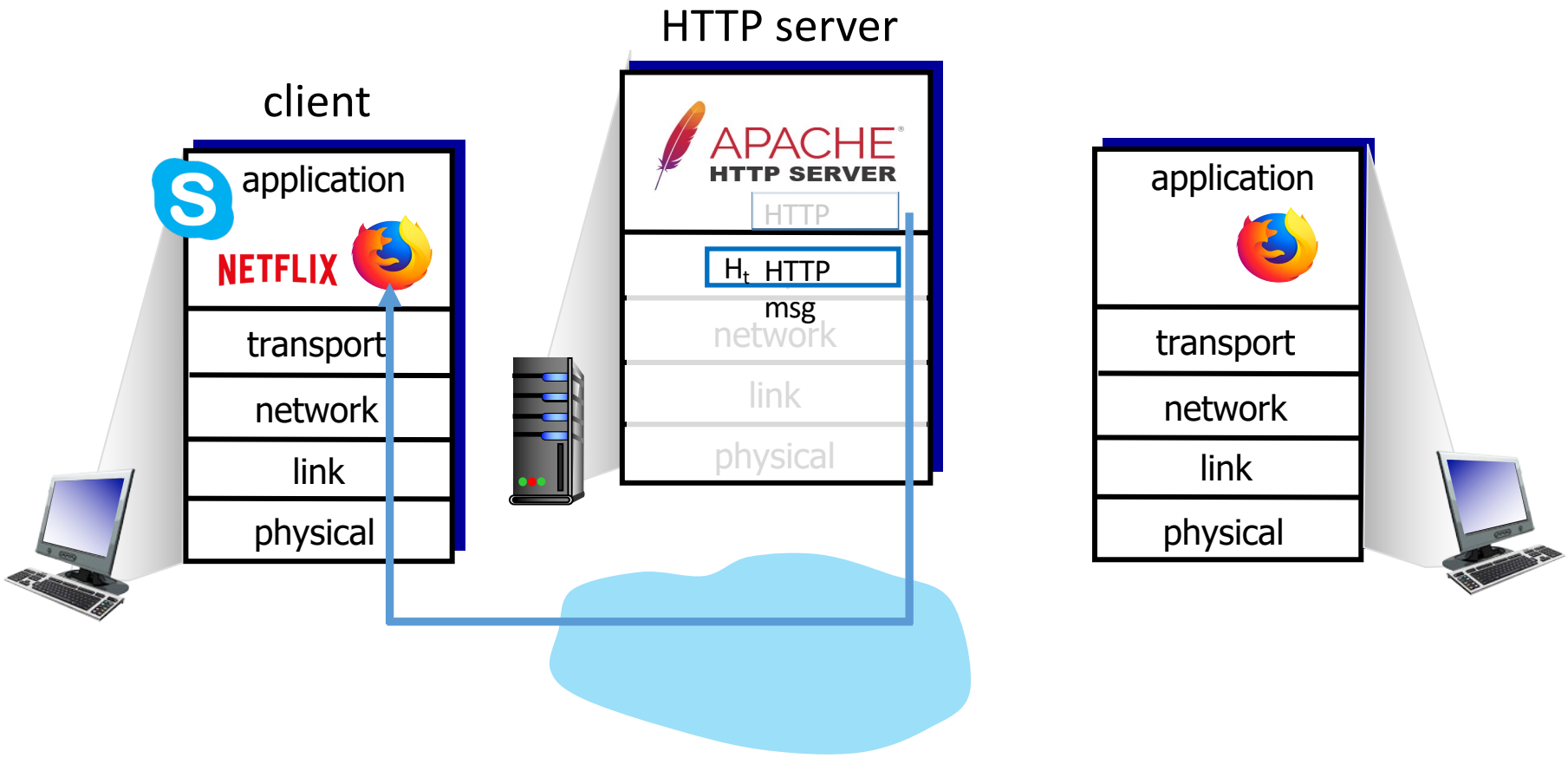
3.5 연결지향형 전송: TCP

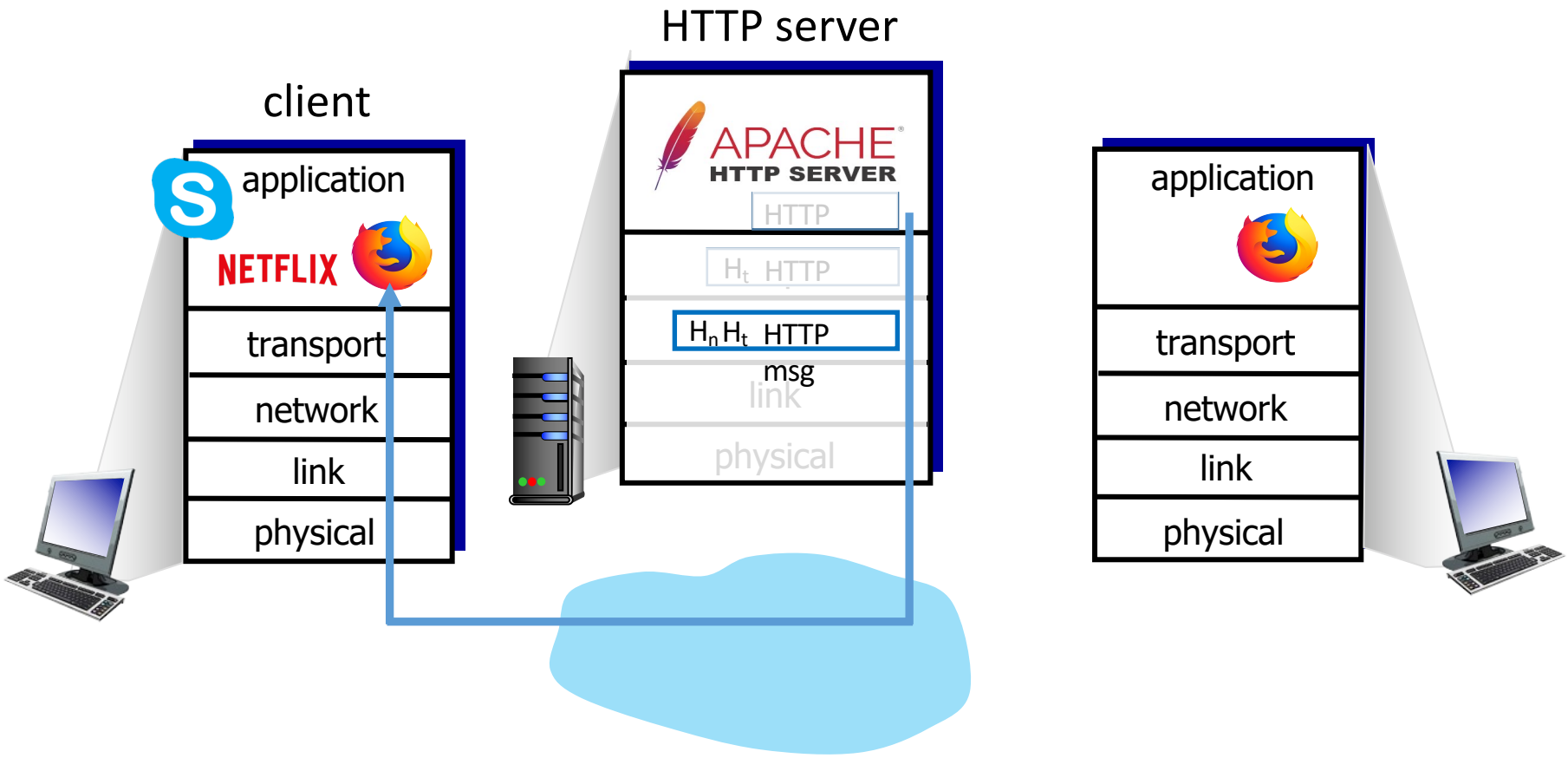
- 세그먼트 구조, 신뢰적 데이터 전송, 흐름 제어, 연결 관리

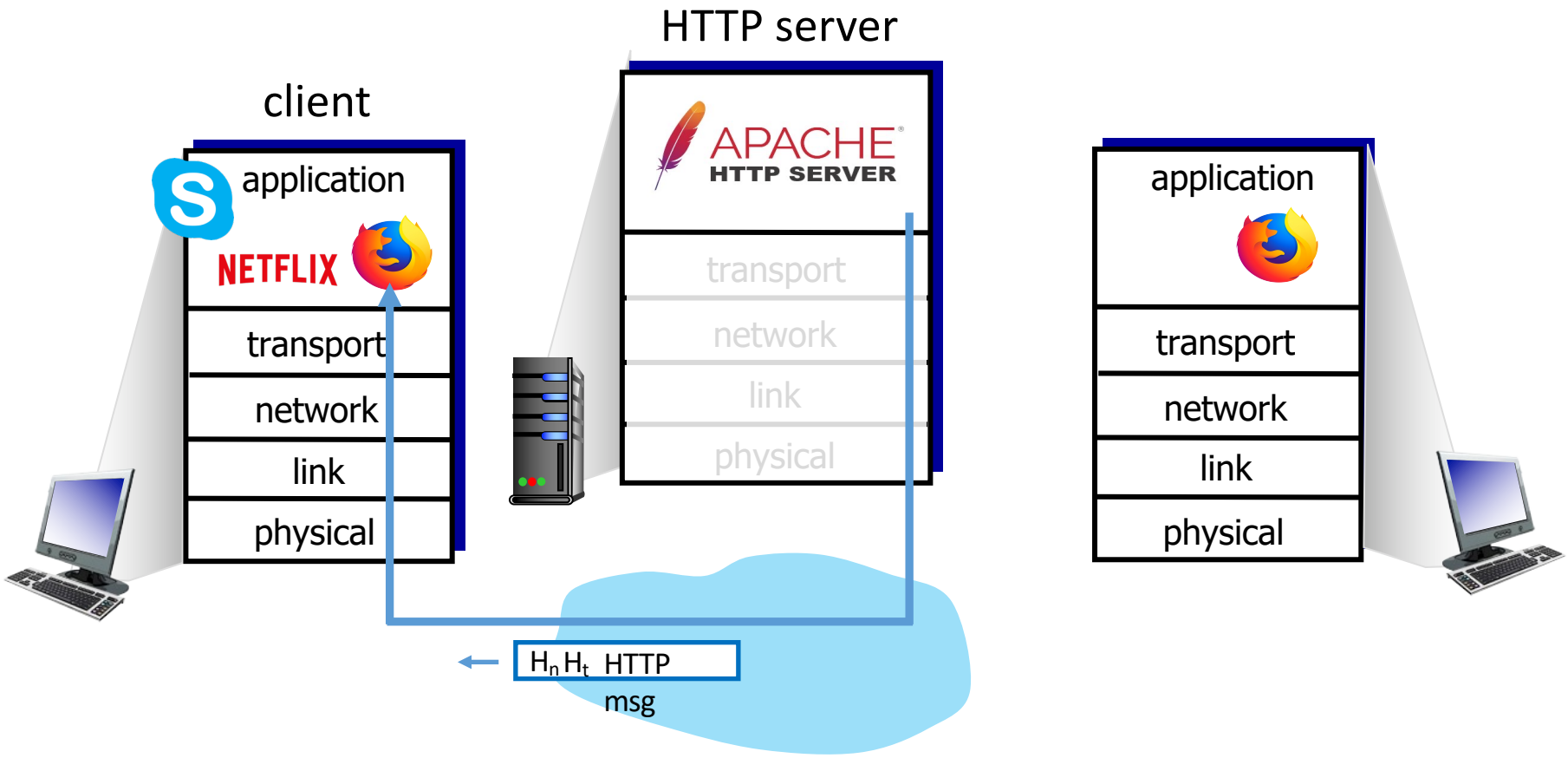
3.6 혼잡 제어의 원리

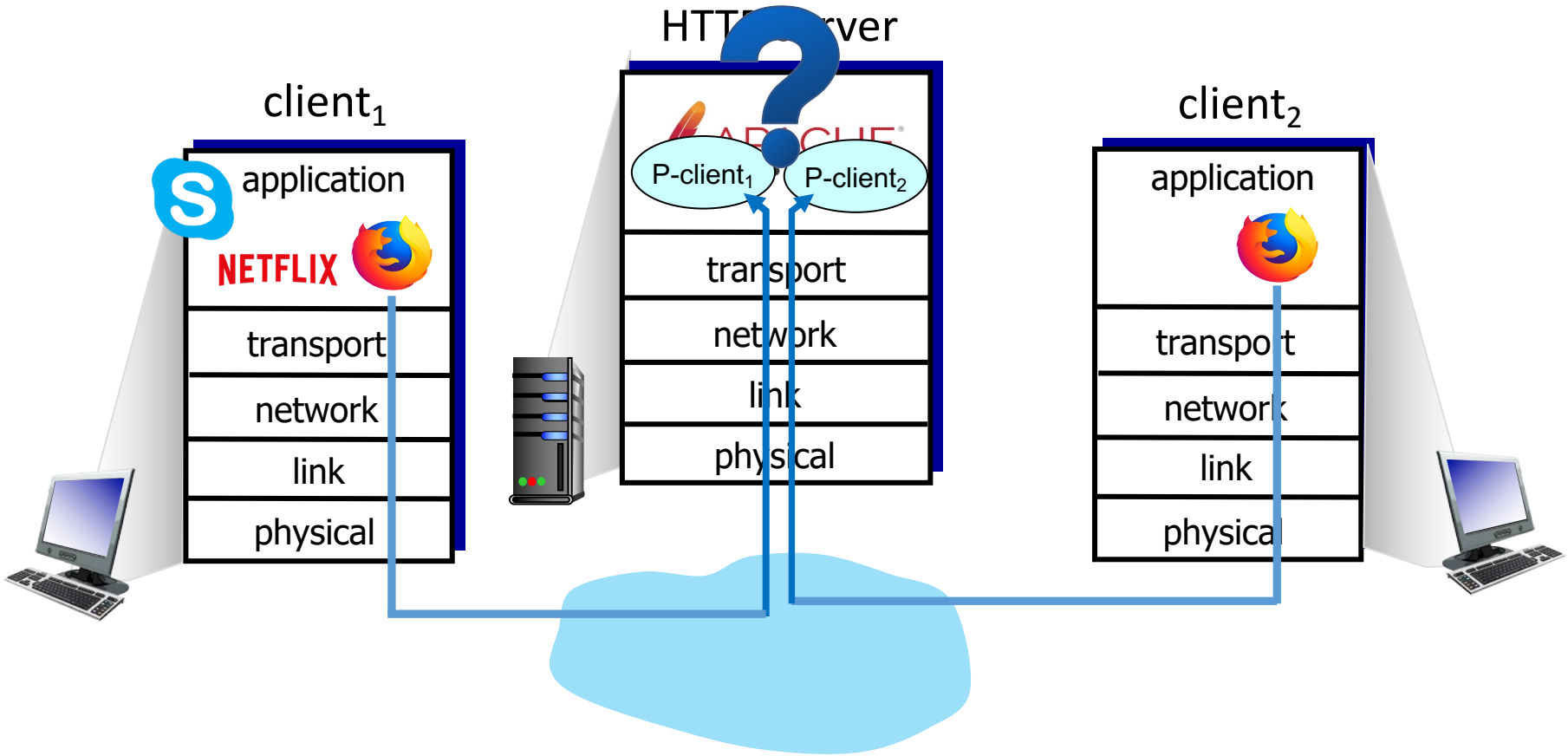
3.7 TCP 혼잡 제어


client
application
NETFLIX
transport
network
link
physical
HTTP server
APACHE
HTTP SERVER
HTTP
msg
transport
network
link
physical
application
transport
network
link
physical


HTTP server
client
application
NETFLIX
transport
network
link
physical
APACHE
HTTP SERVER
HTTP
H_t HTTP
msg
network
link
physical
application
transport
network
link
physical


HTTP server
client
application
NETFLIX
transport
network
link
physical
APACHE
HTTP SERVER
HTTP
H_t HTTP
H_n H_t HTTP
msg
link
physical
application
transport
network
link
physical


HTTP server
client
application
NETFLIX
transport
network
link
physical
APACHE
HTTP SERVER
transport
network
link
physical
application
transport
network
link
physical
H_n H_t HTTP
msg


client1
application
NETFLIX
transport
network
link
physical
HTTP server
P-client1
P-client2
transport
network
link
physical
client2
application
transport
network
link
physical

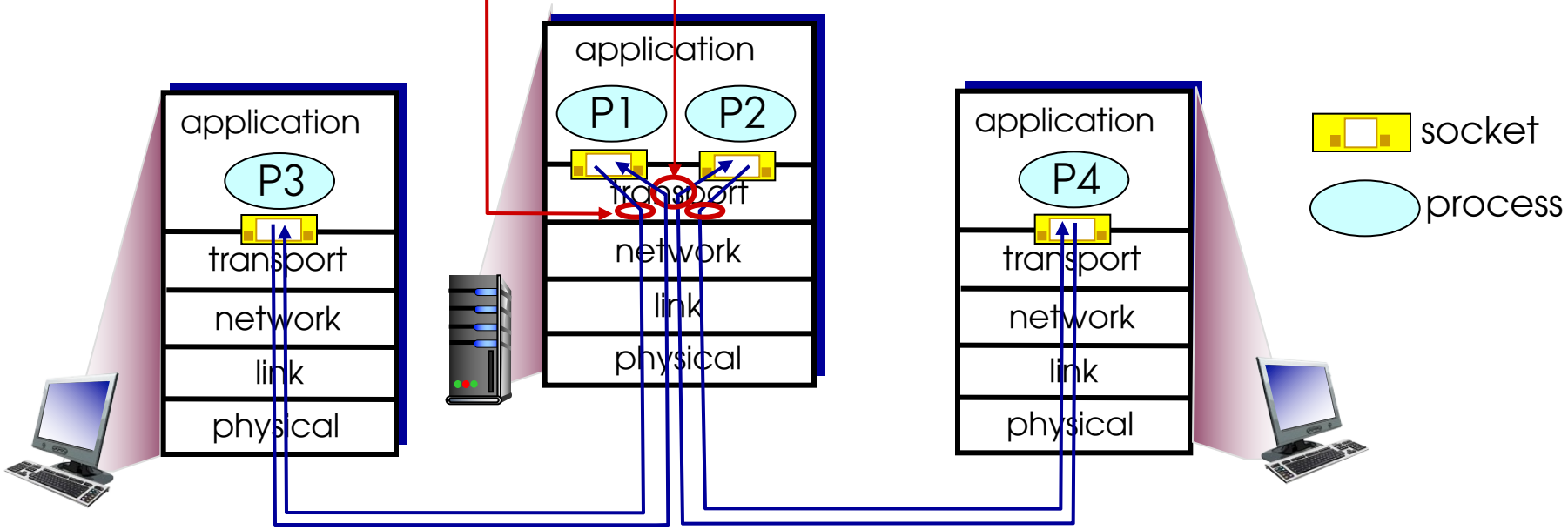
다중화/역다중화 multiplexing/demultiplexing

송신 측 멀티플렉싱

여러 소켓으로부터 온 데이터를 트랜스포트 계층 헤더를 붙여 네트워크 계층에 전달

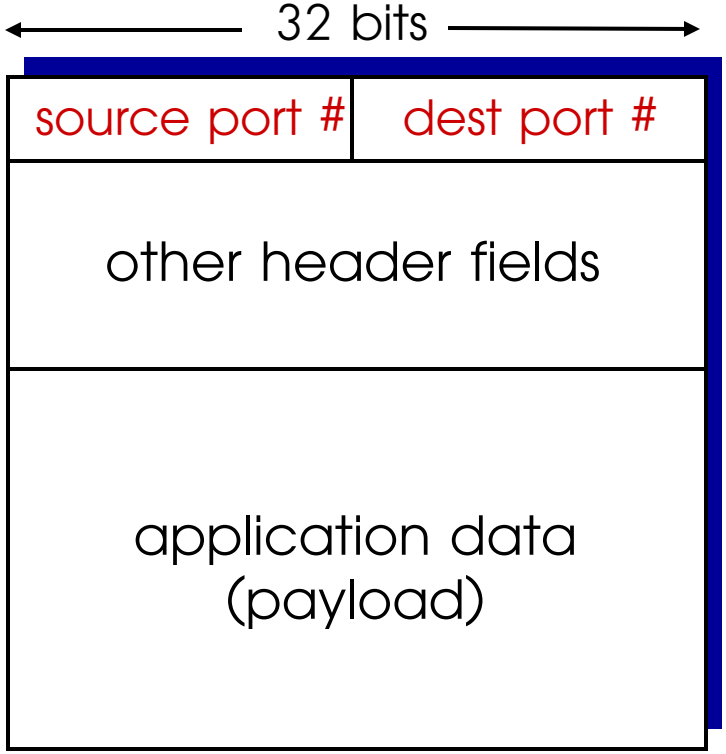
수신 측 디멀티플렉싱

수신한 세그먼트를 헤더 정보에 따라 해당 소켓으로 전달



역다중화 동작

- 호스트는 IP 데이터그램 수신
  - 각 데이터그램은 발신지 IP 주소, 목적지 IP 주소를 가짐
  - 각 데이터그램은 1개의 트랜스포트 계층 세그먼트를 가짐
  - 각 세그먼트는 발신지 포트 번호, 목적지 포트 번호를 가짐
- 호스트는 세그먼트를 해당 소켓으로 전달하기 위해 IP 주소와 포트번호를 사용



TCP/UDP segment format

비연결형 역다중화 Connectionless Demultiplexing

- 소켓을 생성할 때, 호스트 내에서 유일한 포트번호를 지정해야 함
  - `DatagramSocket mySocket1 = new DatagramSocket(12534);`
- UDP 소켓으로 데이터그램 전송 시 반드시 아래 정보를 기술하여야 함
  - 목적지 IP 주소
  - 목적지 포트 번호

- 호스트가 UDP 세그먼트를 수신하면
  - 세그먼트의 목적지 포트 번호 확인
  - UDP 세그먼트를 해당 포트 번호를 가진 소켓으로 전달

→

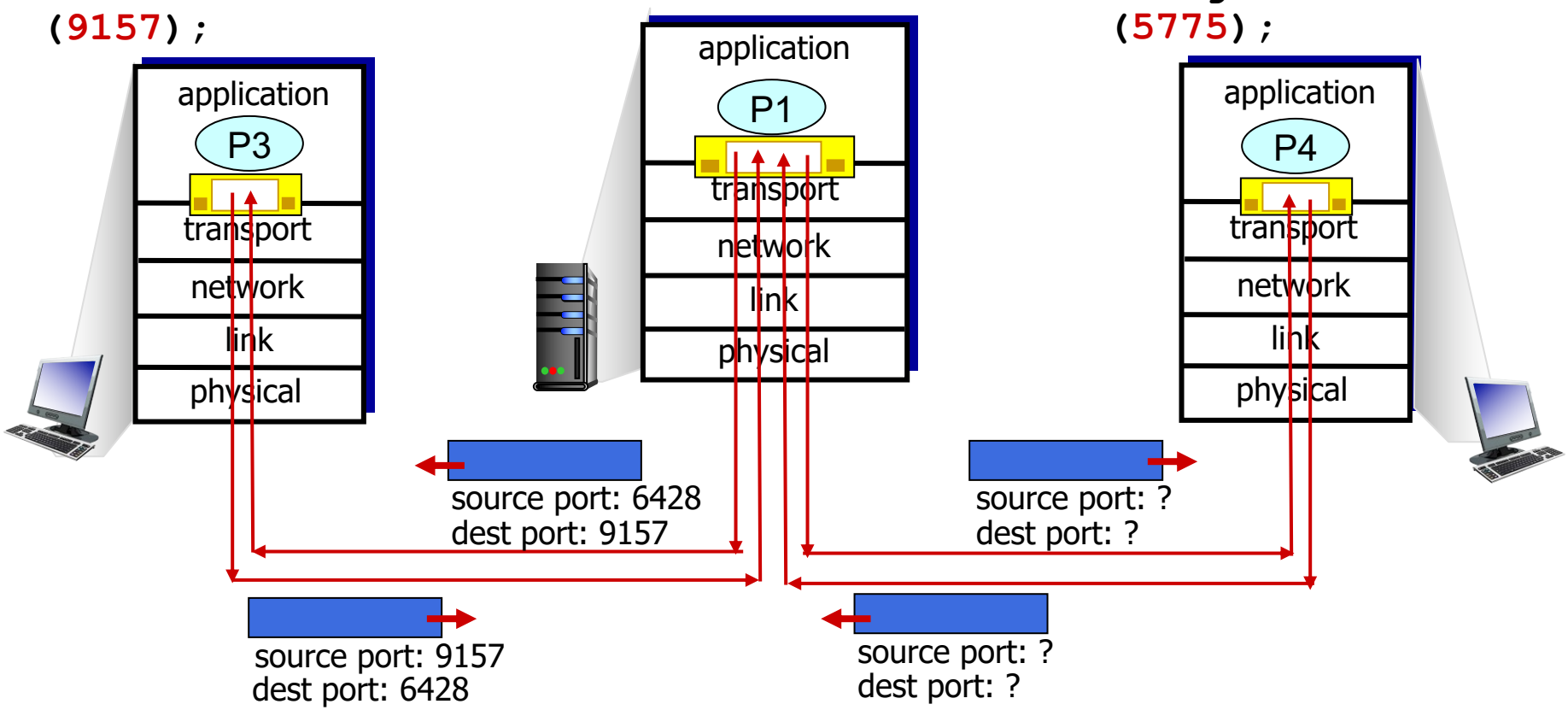
동일 목적 포트 번호를 가진 IP 데이터그램은 발신지 IP 주소나 포트 번호에 상관없이, 동일 소켓으로 전달됨

비연결형 역다중화 예제

```
DatagramSocket
 mySocket2 = new
 DatagramSocket
 (9157);
```

```
DatagramSocket
 serverSocket = new
 DatagramSocket
 (6428);
```

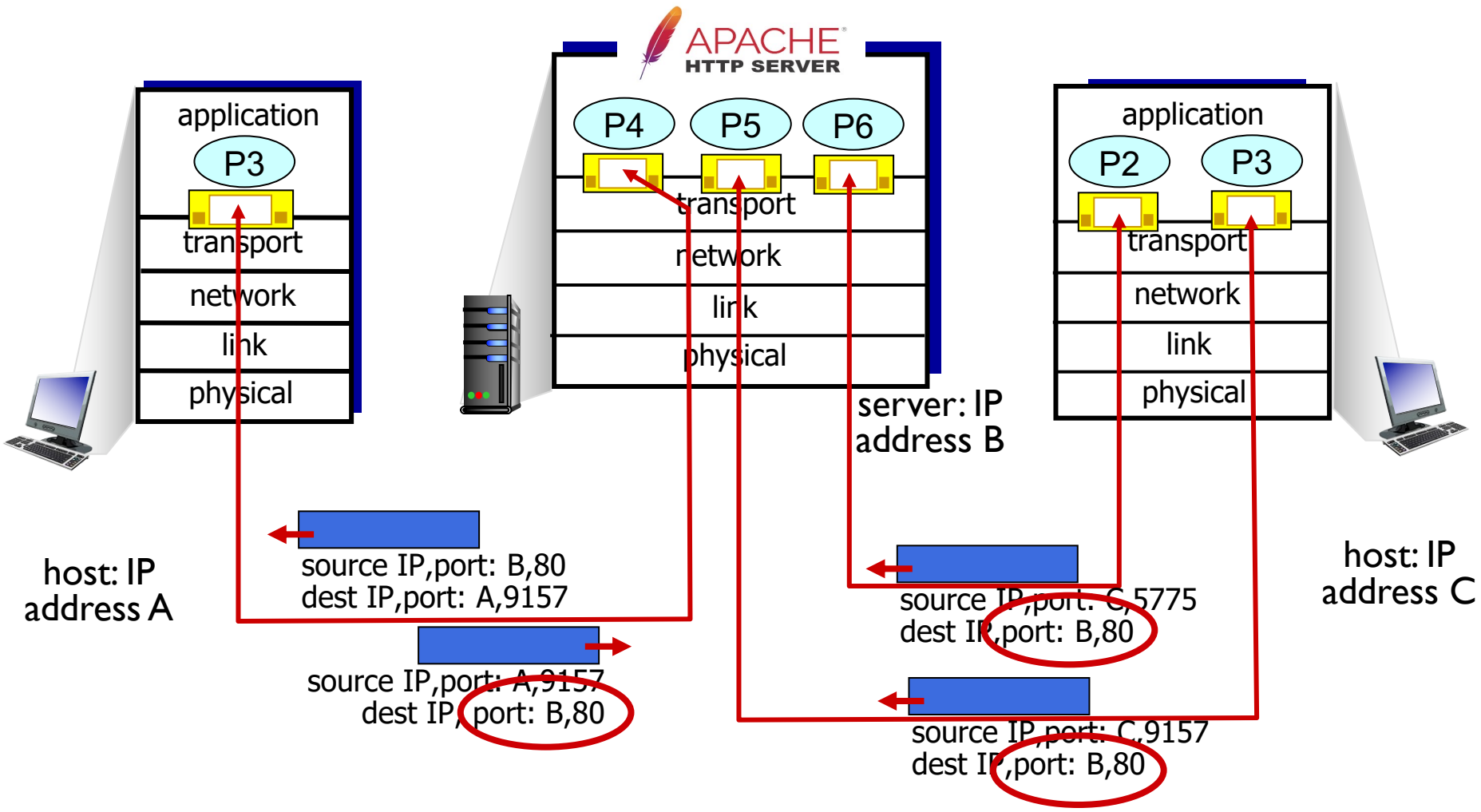
```
DatagramSocket
 mySocket1 = new
 DatagramSocket
 (5775);
```



연결 지향형 역다중화 Connection-Oriented Demultiplexing

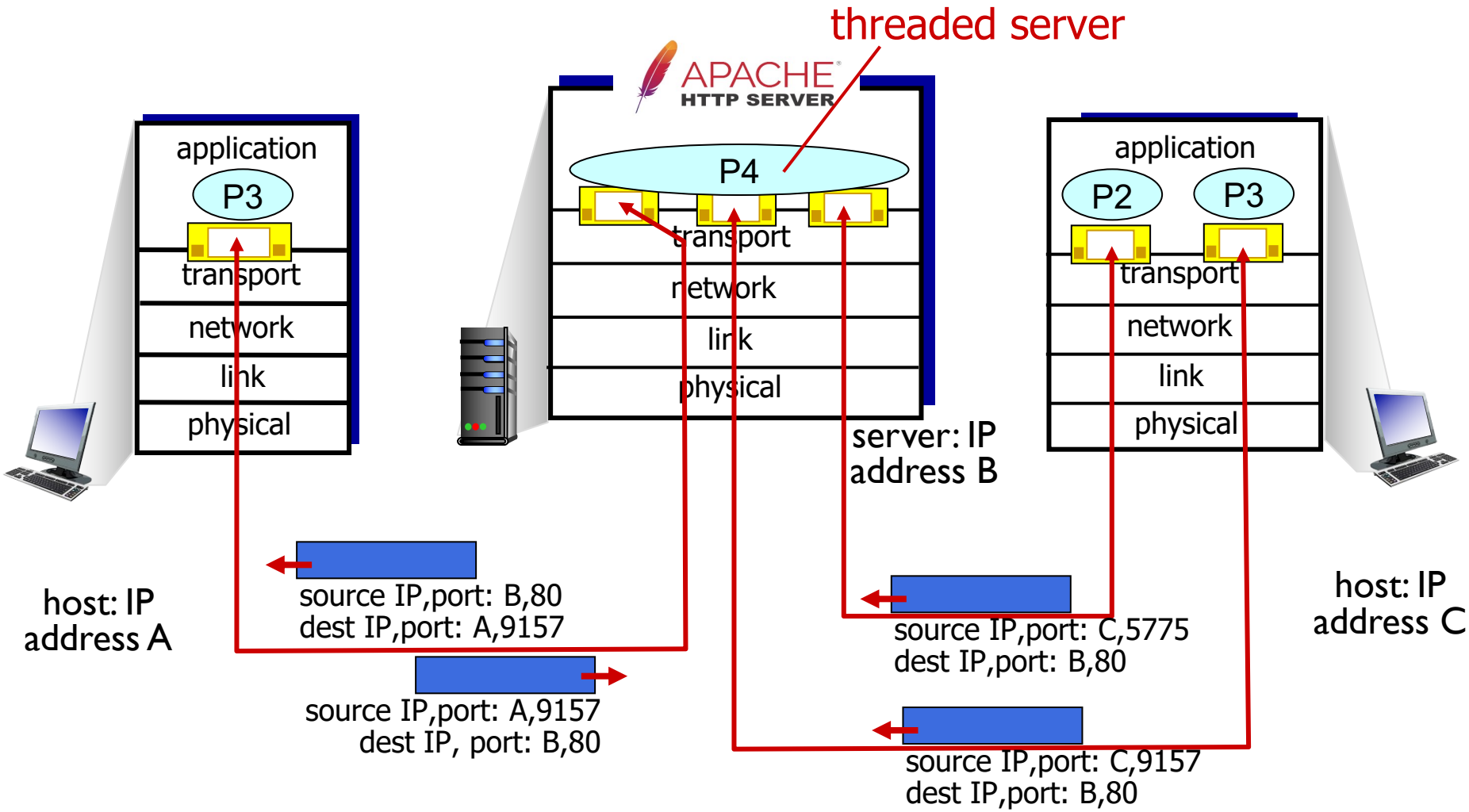
- TCP 소켓은 4-tuple로 구분
  - 발신지 IP 주소
  - 발신지 포트 번호
  - 목적지 IP 주소
  - 목적지 포트 번호
- 역다중화
  - 수신자는 4-tuple을 이용해서 해당 소켓으로 세그먼트 전달
- 서버는 동시에 여러 개의 TCP 소켓을 지원할 수 있음
  - 각 소켓은 4-tuple로 구분 가능
  - 각 소켓은 각각 다른 클라이언트를 나타냄
- 웹 서버는 각 클라이언트에 대해 다른 소켓을 사용
  - 지속 HTTP는 각 클라이언트에 대해 다른 소켓을 사용
  - 비지속 HTTP는 각 요청에 대해 다른 소켓을 사용

연결 지향형 역다중화 예제 (1)



3개의 세그먼트 모두 IP 주소 'B'로 전송됨
목적지 포트 번호 80이 서로 다른 소켓으로 역다중화됨

연결 지향형 역다중화 예제 (2)



일반적으로 서버는 1개의 프로세스로 되어 있으며,
각 클라이언트 요청에 대해 소켓을 생성한 후, 스레드thread가 처리함

트랜스포트 계층

3.1 트랜스포트 계층 서비스

3.2 다중화 / 역다중화

3.3 비연결형 전송: UDP

3.4 신뢰적 데이터 전송의 원리

3.5 연결지향형 전송: TCP

- 세그먼트 구조, 신뢰적 데이터 전송, 흐름 제어, 연결 관리

3.6 혼잡 제어의 원리

3.7 TCP 혼잡 제어

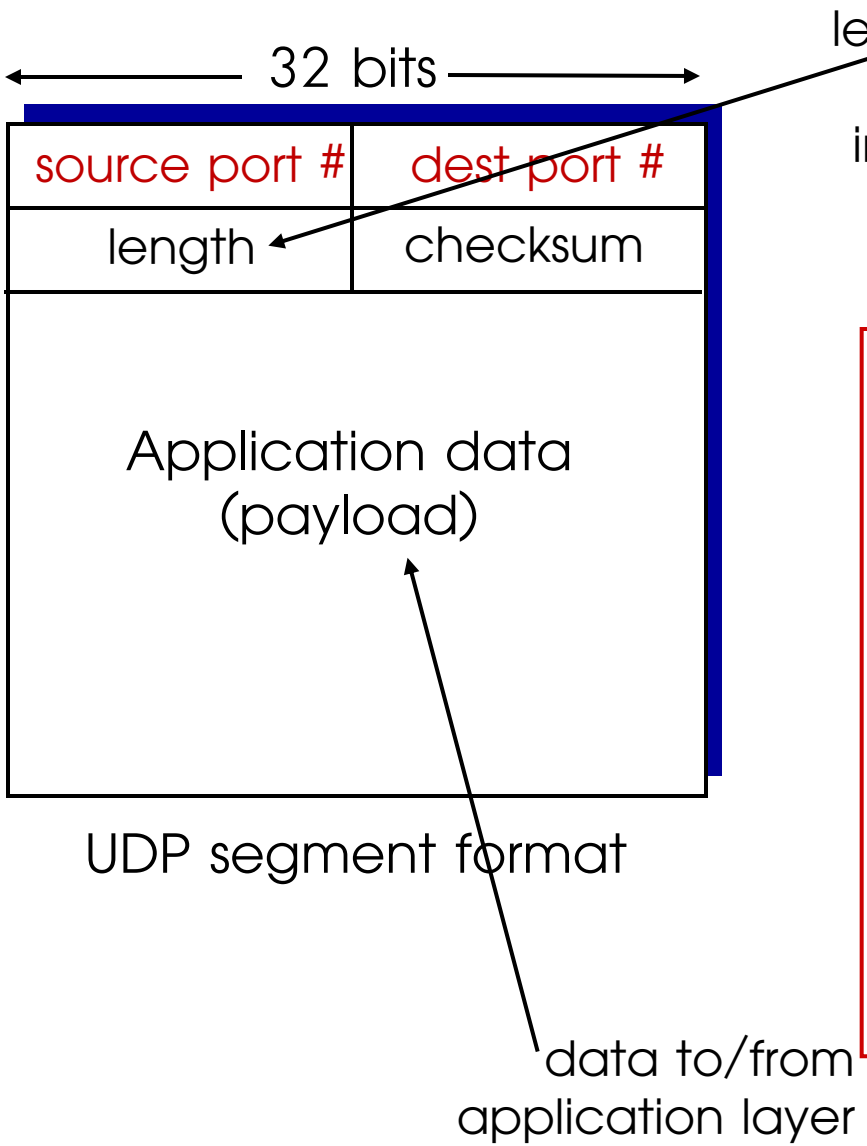
UDP [RFC 768] User Datagram Protocol

- IP 프로토콜에 "최소 기능"만 추가
  - 다중화/역다중화
  - 오류 검사
- "최선(Best effort)" 서비스
  - UDP 세그먼트는
    - ✓ 손실(loss)될 수 있고,
    - ✓ 순서에 맞지 않게 애플리케이션에 전달될 수 있음
- 비연결형connectionless
  - UDP 송신자/수신자 간 핸드셰이킹이 필요 없음
  - 각 UDP 세그먼트는 독립적으로 처리됨
- UDP는 아래 경우에 사용됨
  - 스트리밍 멀티미디어 애플리케이션
    - ✓ 손실 감내loss tolerant
    - ✓ 속도 민감bandwidth sensitive
  - DNS
  - SNMPSimple Network Management Protocol

Q. UDP 상에서 신뢰적 전송을 수행할 수 있는가?

- Yes. How?
- 애플리케이션 계층에서 신뢰성을 제공하여야 함
- 즉, 필요할 경우, 애플리케이션에서 자체적인 에러 복구를 수행하여야 함

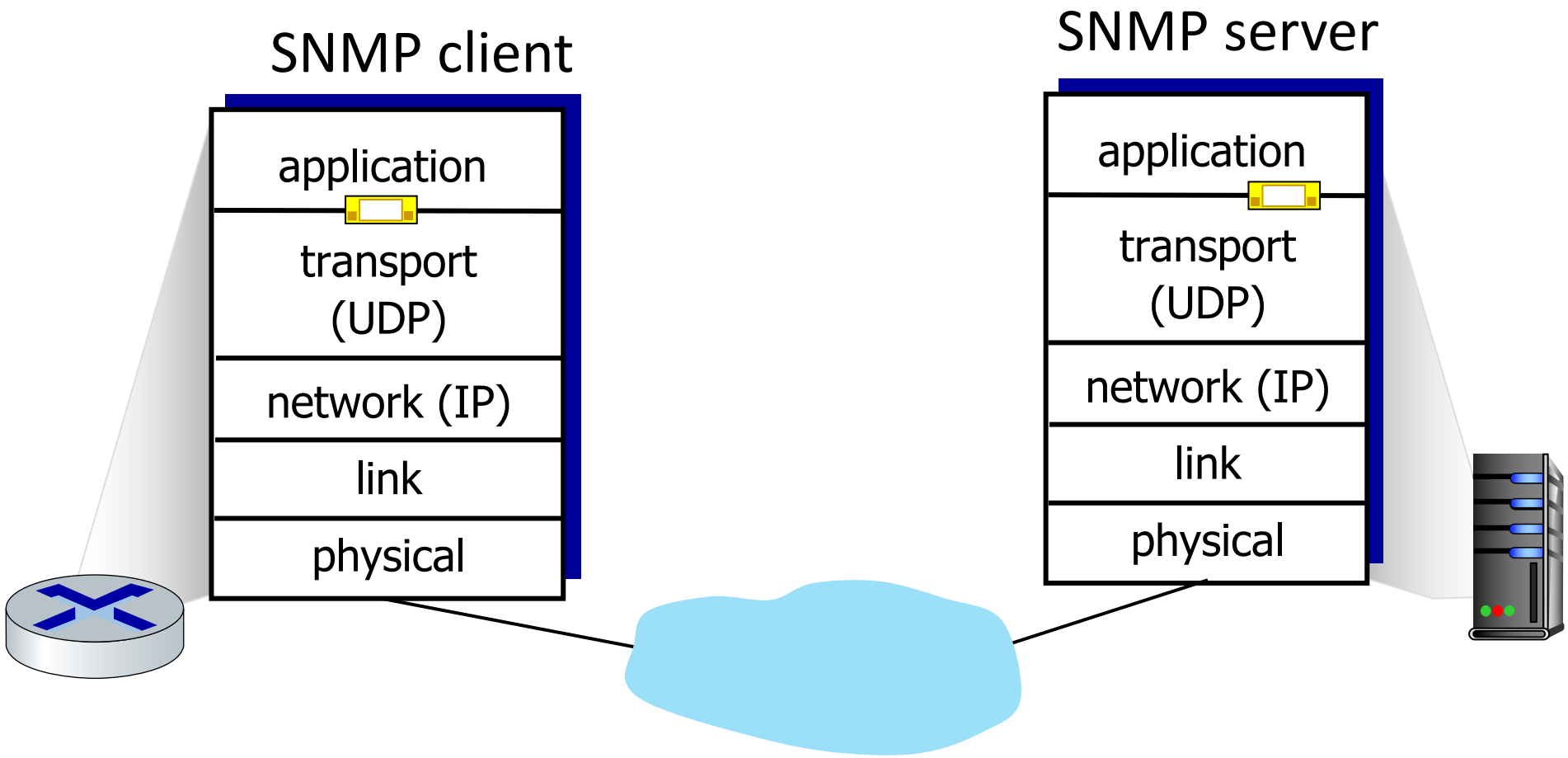
UDP: 세그먼트 헤더



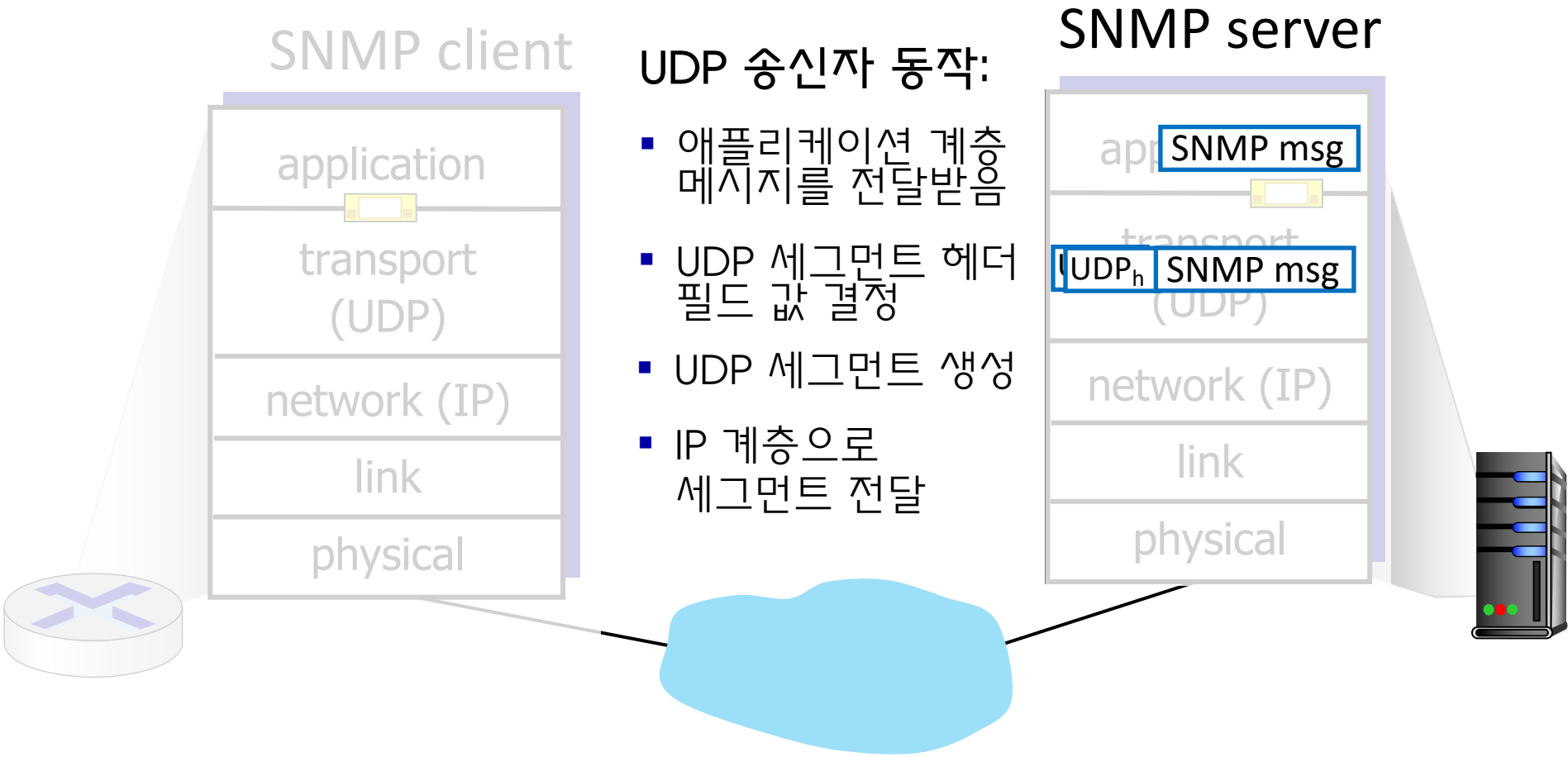
UDP는 왜 필요할까?

- 연결이 필요 없음 (RTT 지연시간이 필요없음)
- 간단함: 송신자/수신자 간 상태 유지가 필요 없음
- 작은 헤더 크기 (8bytes)
- 혼잡 제어 없음. 따라서 UDP는 원하는 만큼 최대한 빨리 전송할 수 있음

UDP: 트랜스포트 계층 동작



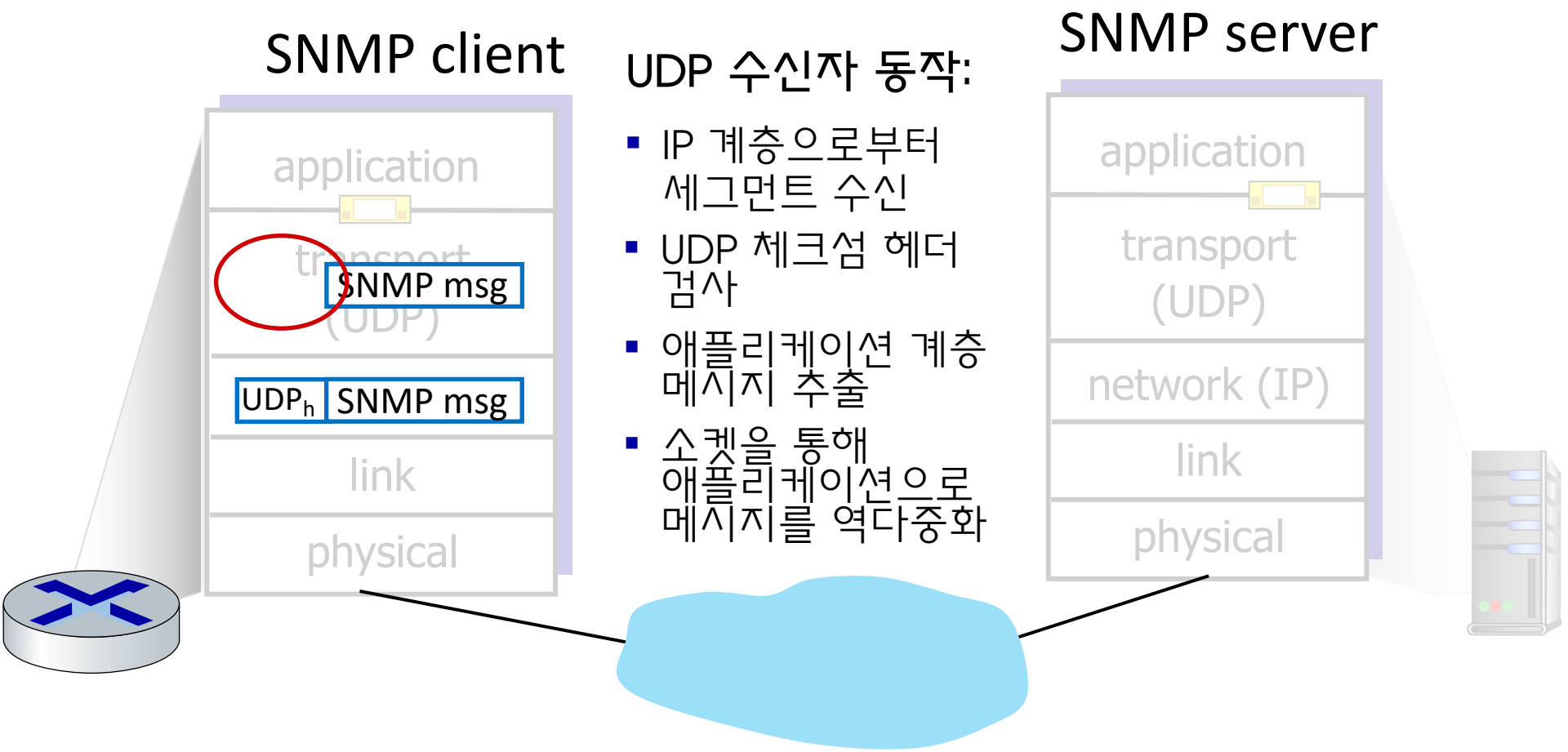
UDP: 트랜스포트 계층 동작



UDP 송신자 동작:

- 애플리케이션 계층 메시지를 전달받음
- UDP 세그먼트 헤더 필드 값 결정
- UDP 세그먼트 생성
- IP 계층으로 세그먼트 전달

UDP: 트랜스포트 계층 동작

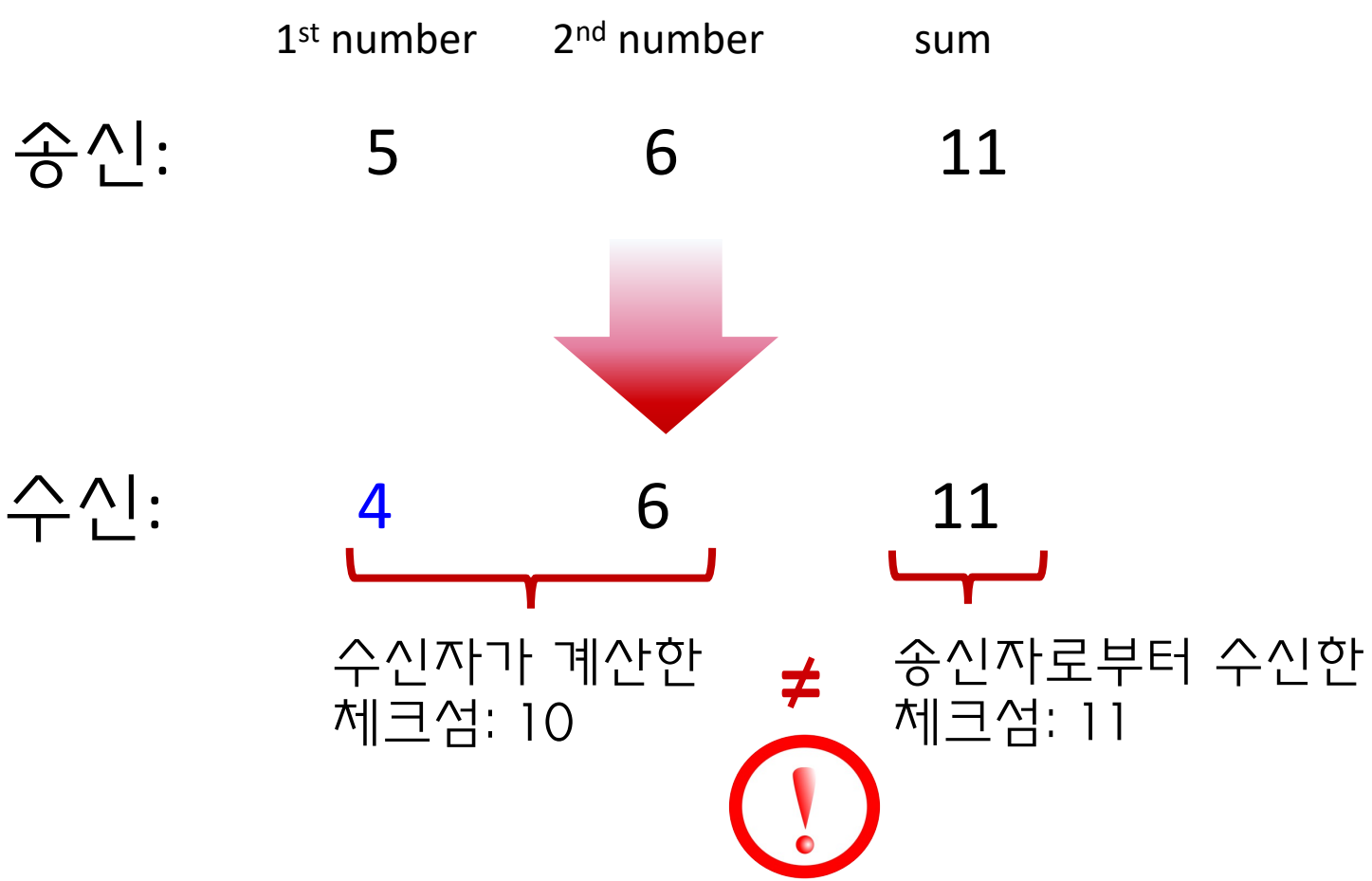


UDP 수신자 동작:

- IP 계층으로부터 세그먼트 수신
- UDP 체크섬 헤더 검사
- 애플리케이션 계층 메시지 추출
- 소켓을 통해 애플리케이션으로 메시지를 역다중화

UDP 체크섬 checksum

- 목적: 전송된 UDP 세그먼트에서 에러 검출



UDP 체크섬

- 목적: 전송된 세그먼트에서 에러 검출
  - 에러(또는 오류): 전송한 비트값과 수신한 비트값이 다른 경우
    - ✓ 예) 송신 측에서 "1111"을 전송하였는데, 수신 측에서 "1101"로 수신됨

송신 측

1. 헤더를 포함한 세그먼트 내용을 16비트 정수로 구분
2. 체크섬 계산
   세그먼트 내 각 16비트 정수의 합의 "1의 보수"
3. 계산한 체크섬을
   UDP "checksum" 필드에 기록

수신 측

1. 수신 세그먼트의 체크섬 계산
2. 계산한 체크섬이 수신한 체크섬과 일치하는 지 확인
3. No - 에러 발생
   Yes - 에러 없음

→ Yes 인 경우에도, 실제 에러 발생가능성 있음

UDP 체크섬

예제: 2개의 16비트 정수 더하기

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	(1)	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

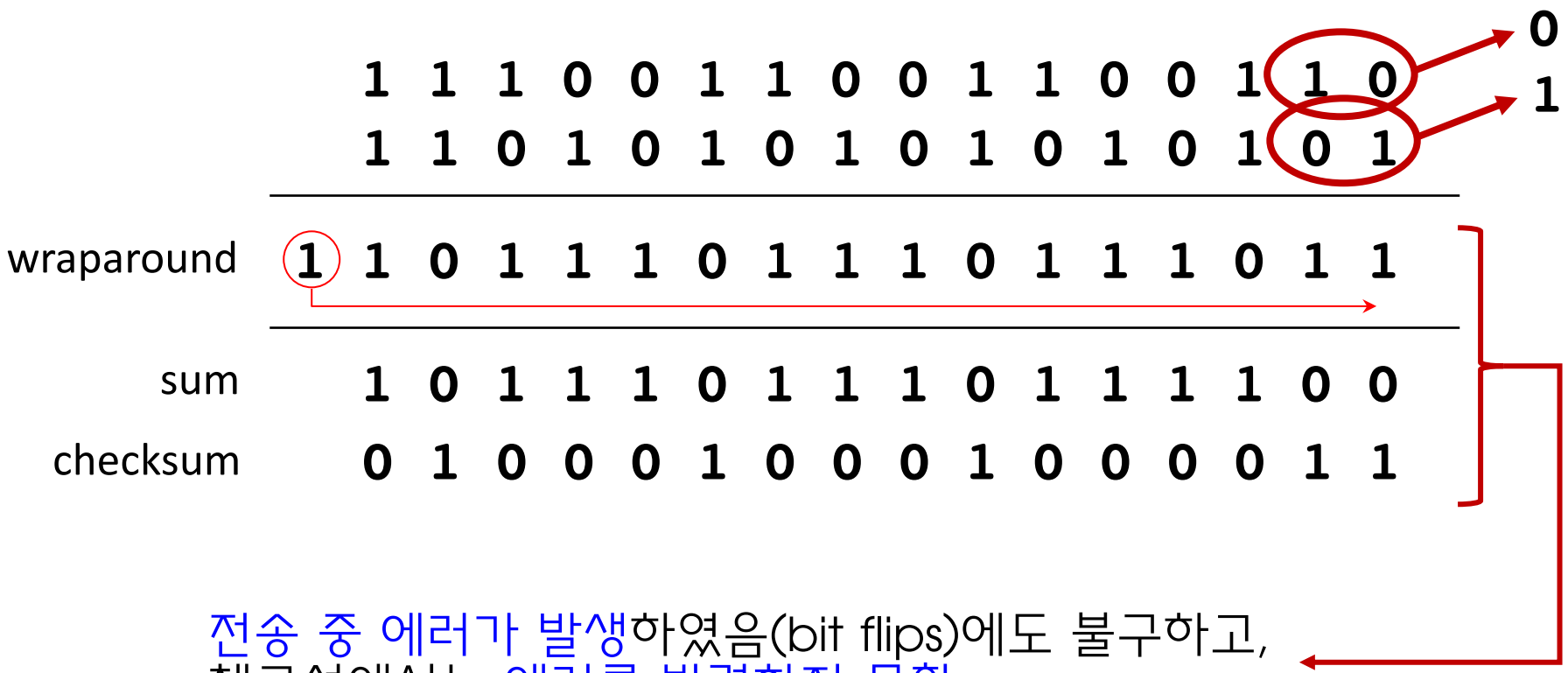
주의사항:

2개의 수를 더할 때, 최상의 비트의 캐리는 결과에 더해져야 함

UDP 체크섬: 에러 발견을 못하는 경우

예제: 2개의 16비트 정수 더하기

아래와 같이
바껴서 수신됨



전송 중 에러가 발생하였음(bit flips)에도 불구하고,
체크섬에서는 에러를 발견하지 못함

트랜스포트 계층

3.1 트랜스포트 계층 서비스

3.2 다중화 / 역다중화

3.3 비연결형 전송: UDP

3.4 신뢰적 데이터 전송의 원리

3.5 연결지향형 전송: TCP

- 세그먼트 구조, 신뢰적 데이터 전송, 흐름 제어, 연결 관리

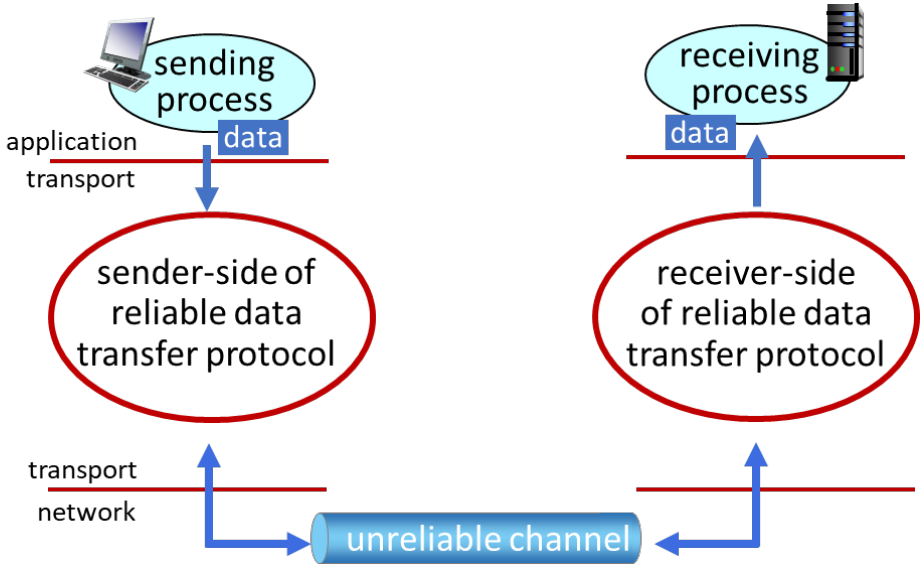
3.6 혼잡 제어의 원리

3.7 TCP 혼잡 제어

신뢰적 데이터 전송의 원리 Principles of Reliable Data Transfer (rdt)

- 신뢰적 데이터 전송 구현
  - 애플리케이션, 트랜스포트, 링크 계층 모두 중요함
  - 네트워크 상위 10위(Top-10) 안에 드는 중요한 이슈
- 비신뢰적 채널의 특성에 따라 신뢰적 데이터 전송reliable data transfer(rdt) 프로토콜의 복잡성이 결정됨

reliable service *abstraction*

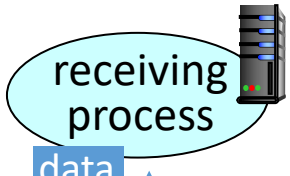


reliable service *implementation*

신뢰적 데이터 전송 프로토콜: 인터페이스

rdt_send(): 상위 계층에서 호출(e.g., by app.). 수신 측의 상위 계층으로 전송할 데이터를 전달

deliver_data(): **rdt**가 호출하여 상위 계층에 데이터 전달



rdt_send()

deliver_data()

우리가 지금부터 공부할 부분 (rdt)

sender-side *implementation* of rdt reliable data transfer protocol

data

packet

receiver-side *implementation* of rdt reliable data transfer protocol

udt_send()

Header data

Header data

rdt_rcv()

unreliable channel

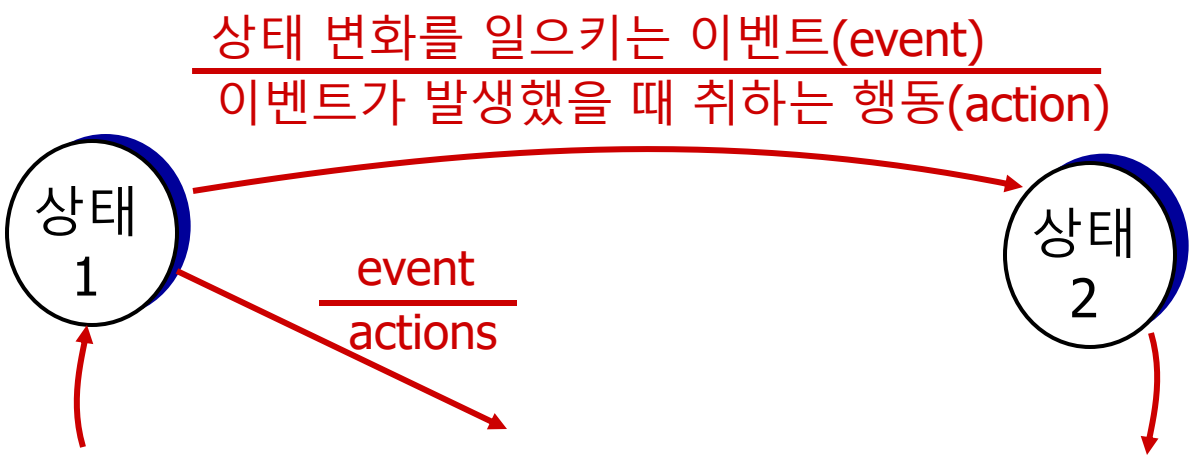
비신뢰적인 채널을 통한 쌍방향 통신

udt_send(): **rdt**가 호출. 비신뢰적 채널을 통해 패킷을 수신 측에 전송

rdt_rcv(): 패킷이 채널의 수신 측에 도착하였을 때 호출

신뢰적 데이터 전송 프로토콜: 시작하기

- 본 챕터에서는
  - 'rdt'의 송신 측/수신 측의 기능 개발을 단계적으로 소개할 예정임
  - "단방향 데이터 전송unidirectional data transfer" 만을 고려
    - ✓ 제어 정보는 양방향으로 전송됨
  - 송신자, 수신자의 동작을 기술하기 위해 유한상태기계finite state machines (FSM)을 사용함

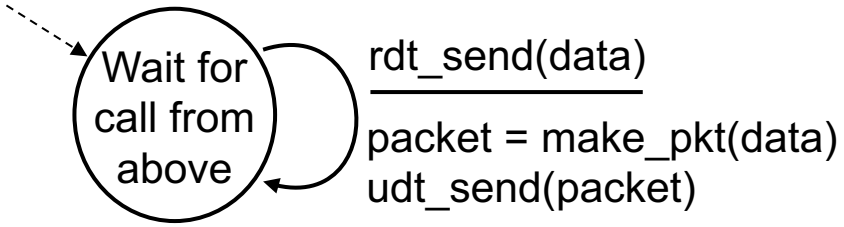


상태(state): "현재 상태"에서 "어떤 이벤트"에 의해 유일한 "다음 상태"가 결정됨

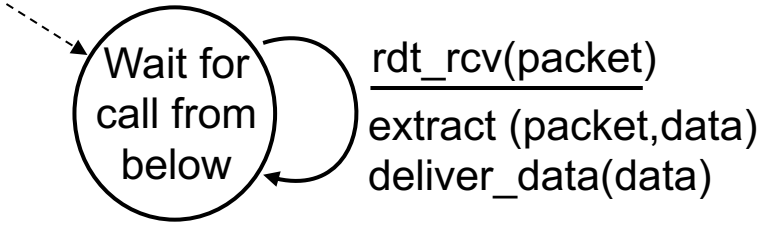
rdt 1.0: 신뢰적 채널 상에서 신뢰적 전송

- 하위 채널이 완전히 신뢰적임
  - 비트 에러 없음
  - 패킷 손실 없음

- 송신자, 수신자 각각에 대한 FSM
  - 송신자는 하위 채널로 데이터 전송
  - 수신자는 하위 채널로부터 데이터 수신



sender



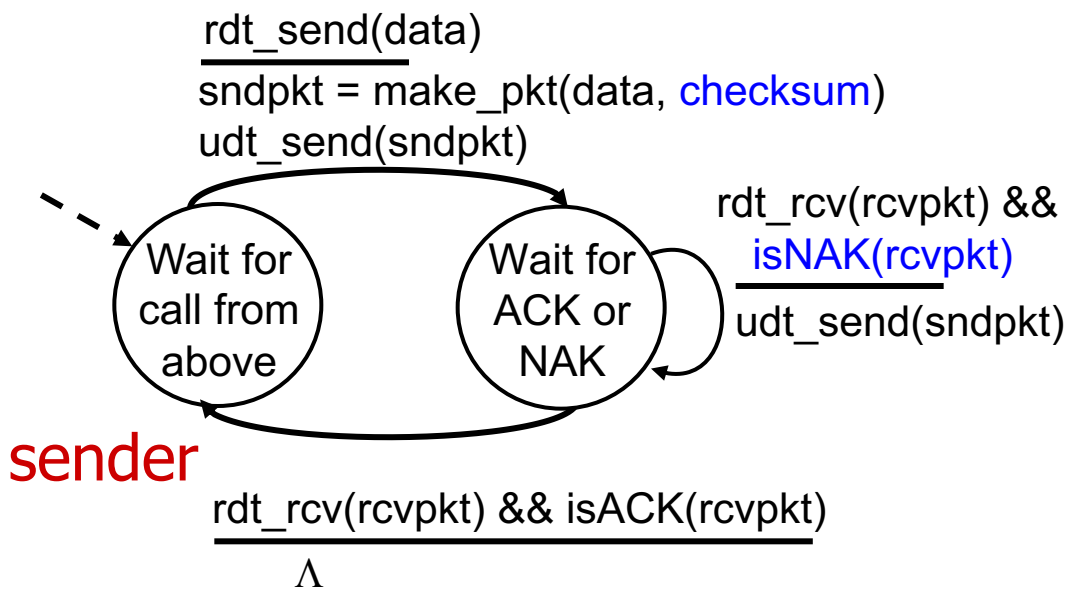
receiver

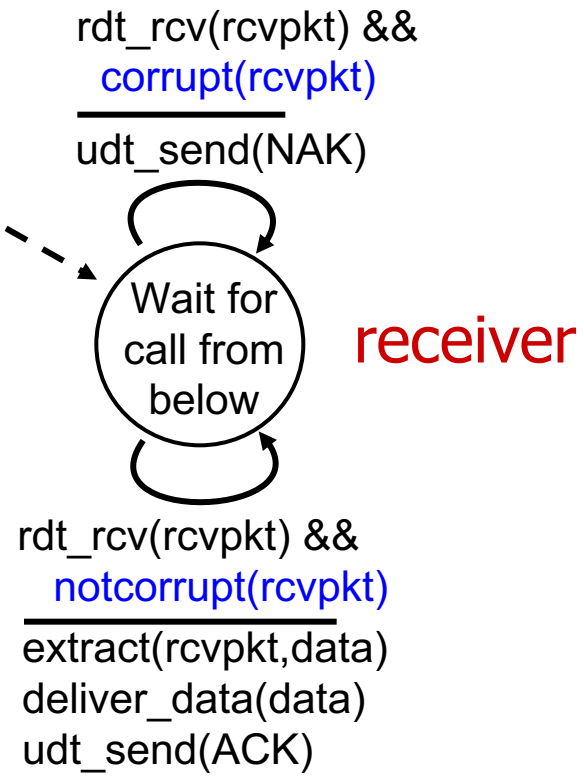
rdt 2.0: 비트 에러가 있는 채널

- 하위 채널에서 비트 에러가 발생 가능
  - 체크섬을 사용하여 비트 에러 검출
- Q: 어떻게 에러를 복구할 수 있는가?
  - ACK (acknowledgement)
    - ✓ 수신자는 수신된 패킷이 에러가 없음을 송신자에게 알려줌
  - NAK (negative acknowledgement)
    - ✓ 수신자는 수신한 패킷이 에러가 있음을 송신자에게 알려줌
  - 송신자는 NAK 수신 시, 패킷을 재전송함
- rdt 2.0의 추가 기능
  - 에러 발견
  - 피드백
    - ✓ 수신자는 제어 메시지(ACK, NAK)를 송신자로 전송
  - 재전송

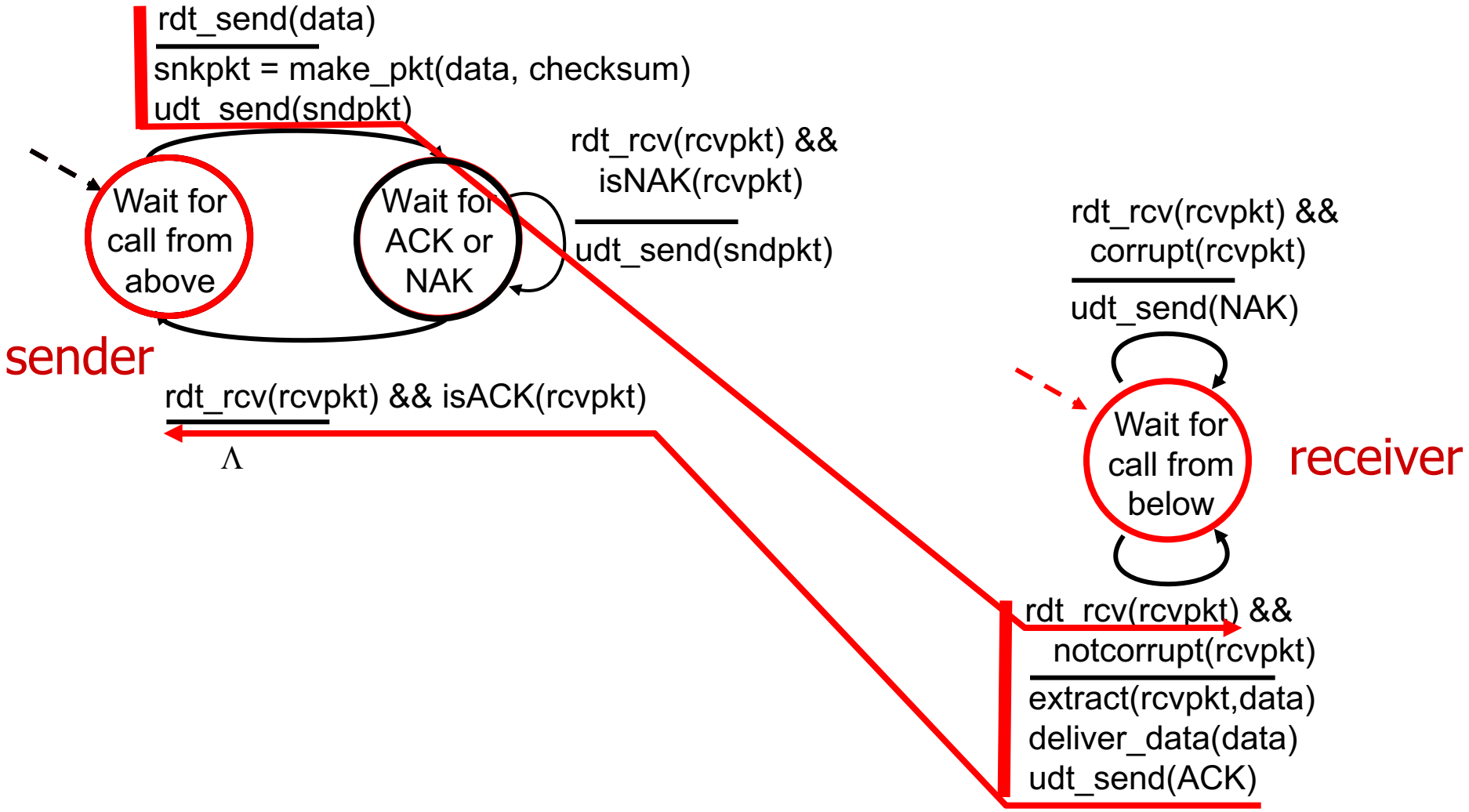
stop and wait
(전송 후 대기)
송신자는 1개의 패킷을 전송하고, 수신자로부터 응답을 받기를 기다림

rdt 2.0: FSM 동작 정의

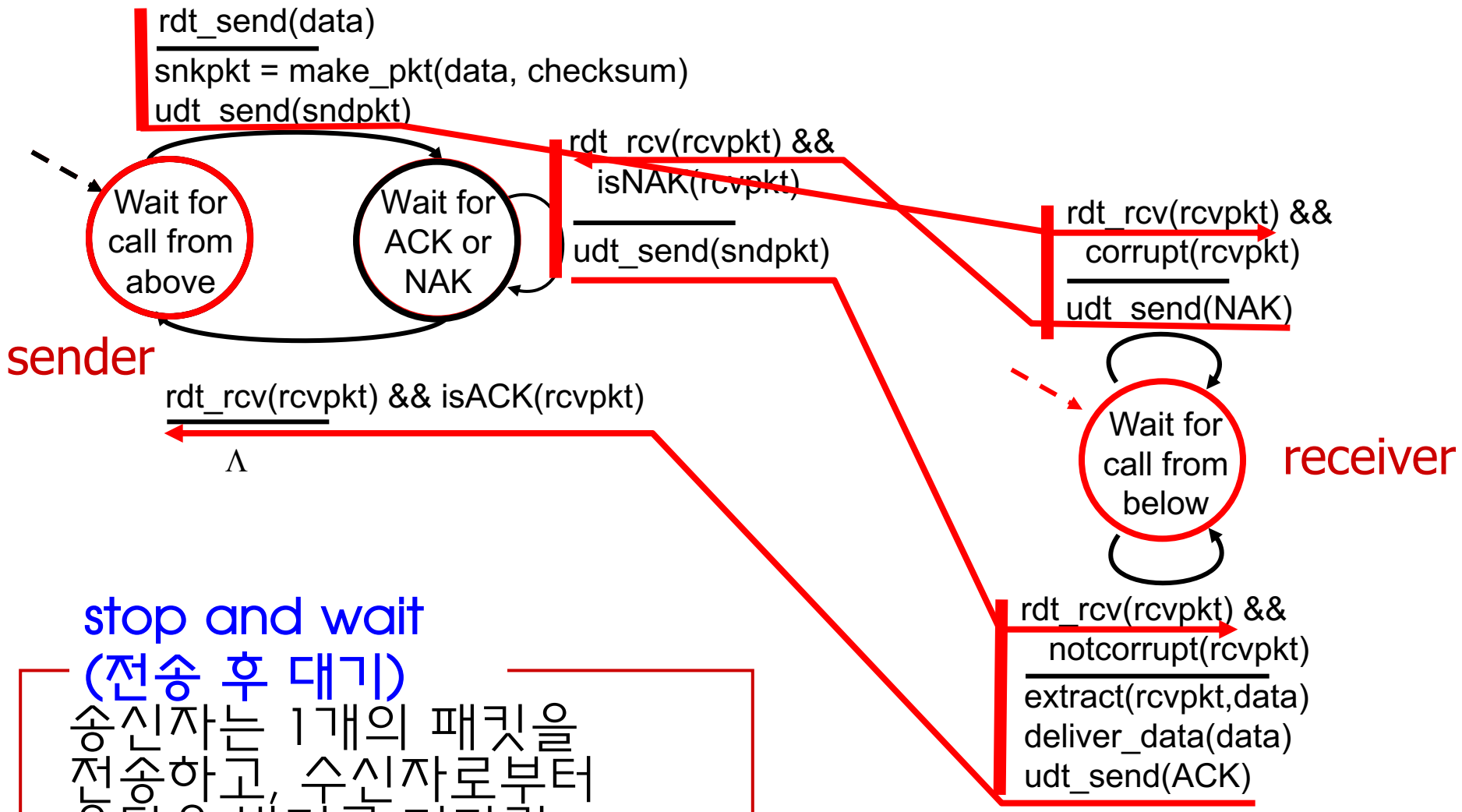




rdt 2.0: 에러가 없을 경우 동작



rdt 2.0: 에러가 발생했을 경우 동작



stop and wait
(전송 후 대기)

송신자는 1개의 패킷을 전송하고, 수신자로부터 응답을 받기를 기다림

rdt 2.0: 문제점

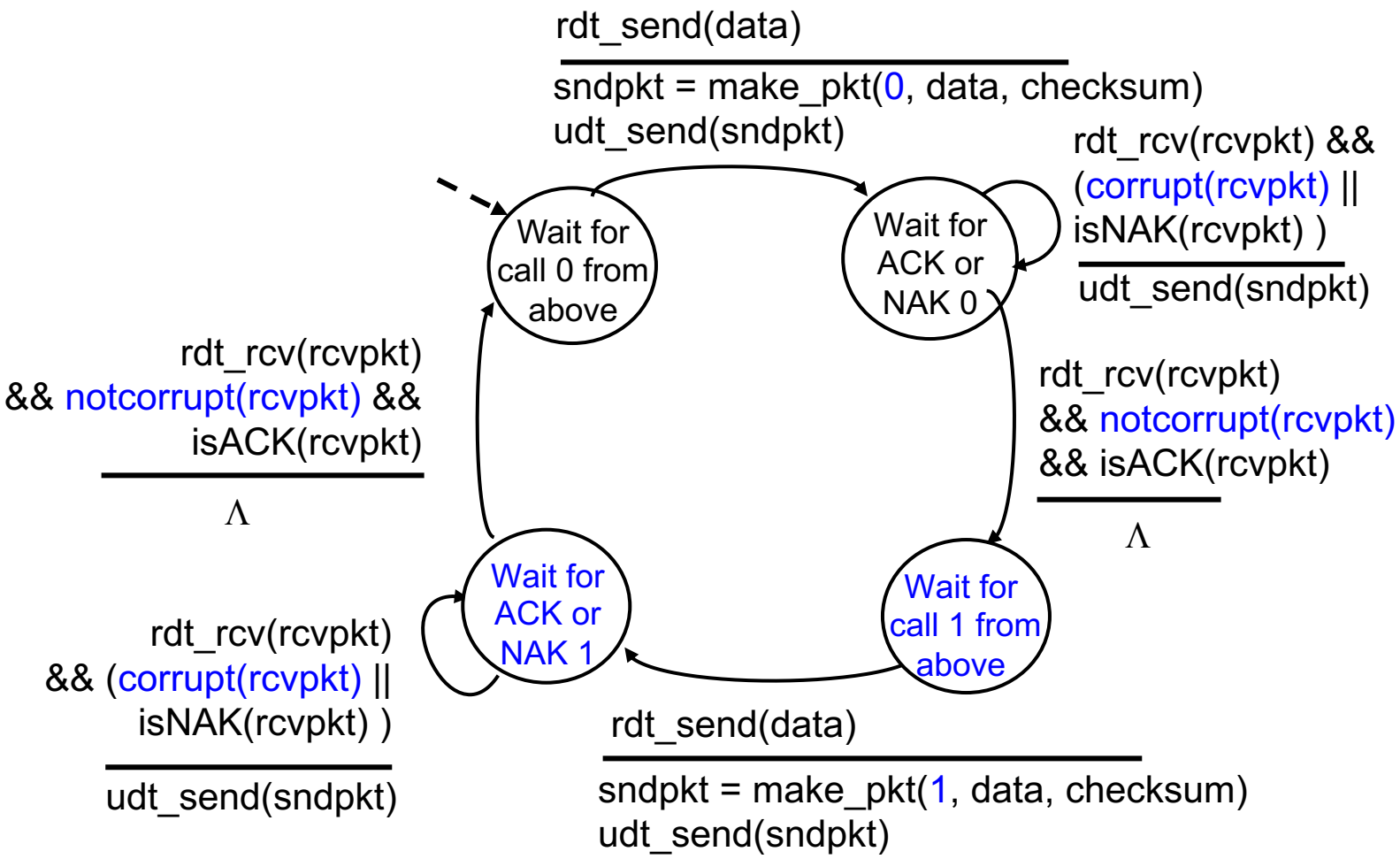
Q. ACK/NAK에 에러가 발생하면 어떻게 되는가?

- ACK/NAK 에러발생 시 (ACK/NACK을 구분할 수 없으므로) 송신자는 수신자가 패킷을 잘 받았는지 알 수 없음
- ACK/NAK 에러 발생 시, 수신자가 이전에 패킷을 잘 수신한 상황(ACK을 보낸 상황)에서 패킷을 그냥 재전송하면 수신자는 중복된 패킷을 수신하게 되나, 해당 패킷이 중복인지 아닌지 구분할 수 없음

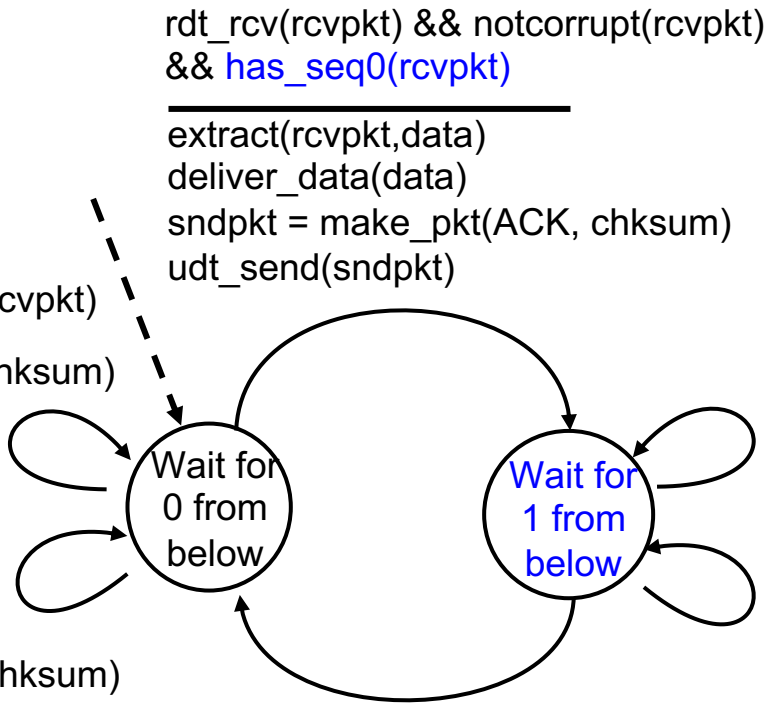
■ 중복 패킷duplicate packet 처리 방법

- 송신자는 각 패킷에 순서 번호sequence number를 붙임
- 송신자는 ACK/NAK에 에러가 발생하면 패킷을 재전송
- 수신자는 순서 번호를 이용해 중복 패킷은 버림
  - ✓ 예)
  1. 송신자가 순서 번호 0인 패킷을 전송
  2. 수신자는 패킷을 정상적으로 수신한 후, ACK을 전송
  3. ACK에 에러가 발생하여, 송신자는 순서 번호 0인 패킷을 재전송
  4. 수신자는 패킷의 순서 번호를 이용해 중복 패킷임을 확인하고 버림
  5. 수신자는 ACK을 전송

rdt 2.1: 송신자, ACK/NAK 에러 고려



rdt 2.1: 수신자, ACK/NAK 에러 고려



rdt 2.1: 정리

■ 송신자

- 패킷에 순서 번호를 붙임
  - ✓ 0과 1, 두 개의 순서 번호 사용
- 수신한 ACK/NAK의 에러 여부를 검사함
- rdt 2.0에 비해 2배의 상태 필요
  - ✓ rdt 2.0: 송신자 2개, 수신자 1개 / rdt 2.1: 송신자 4개, 수신자 2개
  - ✓ 다음 송신 패킷의 순서 번호가 0인지 1인지를 기억해야 함

■ 수신자

- 수신된 패킷이 중복되었는지 조사
  - ✓ 다음 수신 패킷의 순서 번호가 0인지 1인지를 기억해야 함
- 수신자는 ACK/NACK이 송신자로 잘 전달되었는지 알 수 없음

rdt 2.2: NAK 없는 프로토콜 (NAK-free protocol)

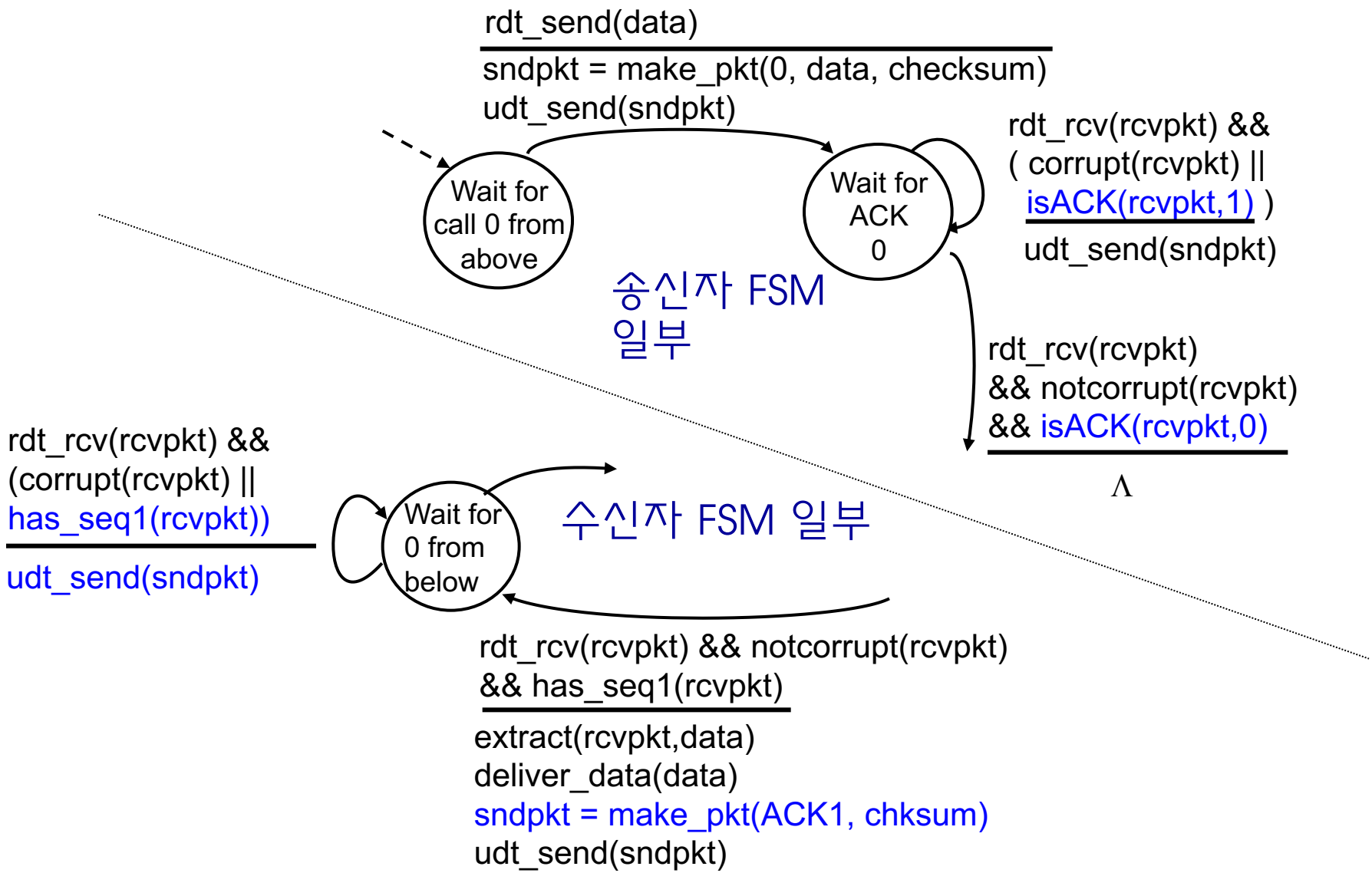
- NAK없이 ACK만 사용한다는 점을 제외하고는 rdt 2.1과 동일

- 수신자는 NAK 대신에 가장 최근에 성공적으로 수신한 패킷에 대한 ACK을 보냄
  - 수신자는 ACK 패킷에 순서 번호를 명시해야 함

- 송신자는 중복된 ACK를 수신하면, rdt 2.0에서 NAK을 수신한 것과 같이 현재 패킷을 재전송

- Note: TCP는 NAK 없는 프로토콜임

rdt 2.2: 송신자, 수신자 동작 일부



rdt 3.0: 에러와 "손실"이 있는 채널

■ 하위 채널에 대한 새로운 가정

- 하위 채널에서는 패킷(데이터, ACK)이 분실(packet loss)될 수 있음
  - ✓ "체크섬, 순서 번호, ACK, 재전송"은 도움은 되지만, 근본적인 분실 문제를 해결할 수 없음

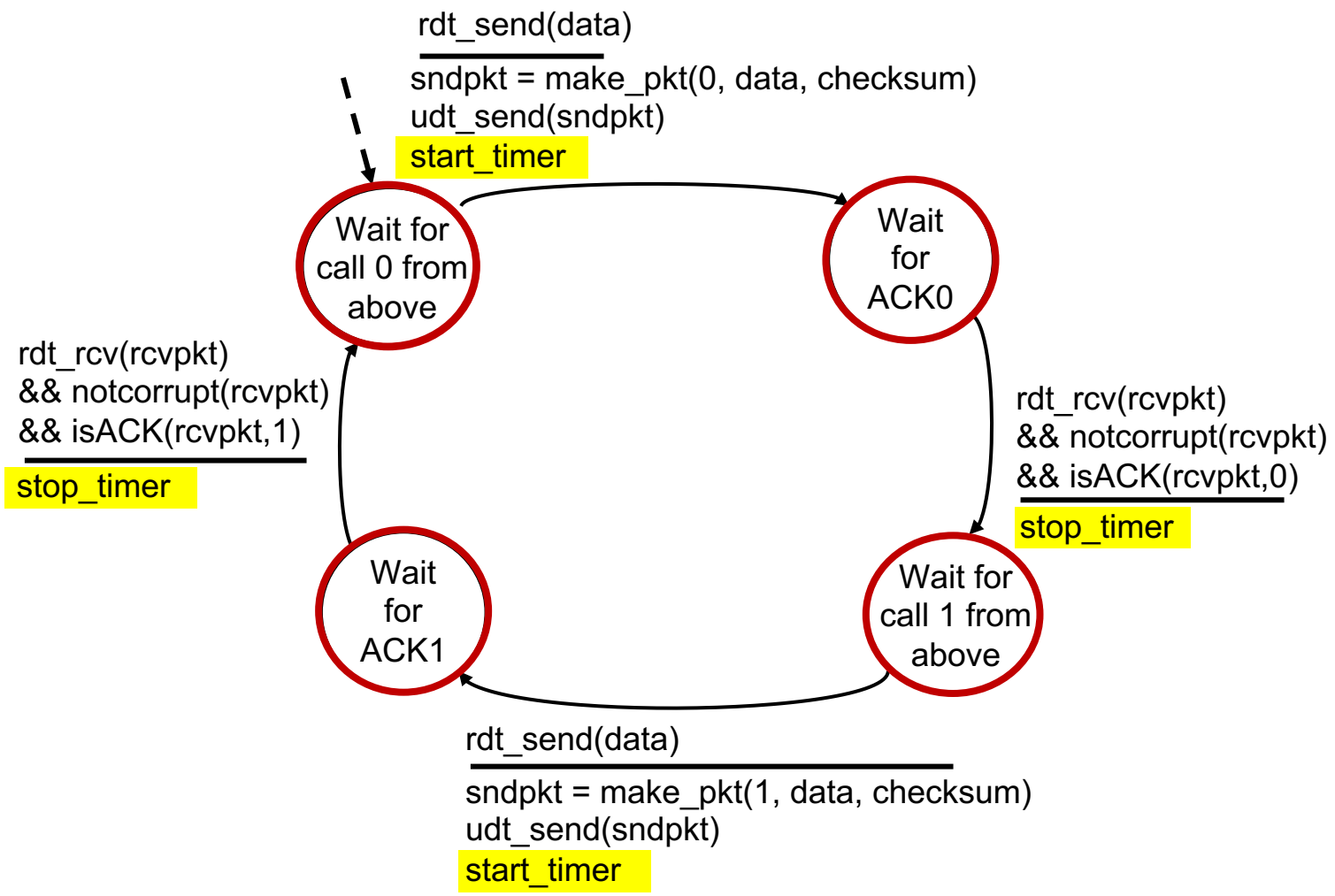
■ 해결 방법

- 송신자가 "충분한 시간" 동안 ACK 수신을 기다림
  - ✓ 지정된 시간 동안 ACK를 받지 못하면 재전송
  - ✓ "충분한 시간"을 측정할 "카운트다운 타이머countdown timer"가 필요함

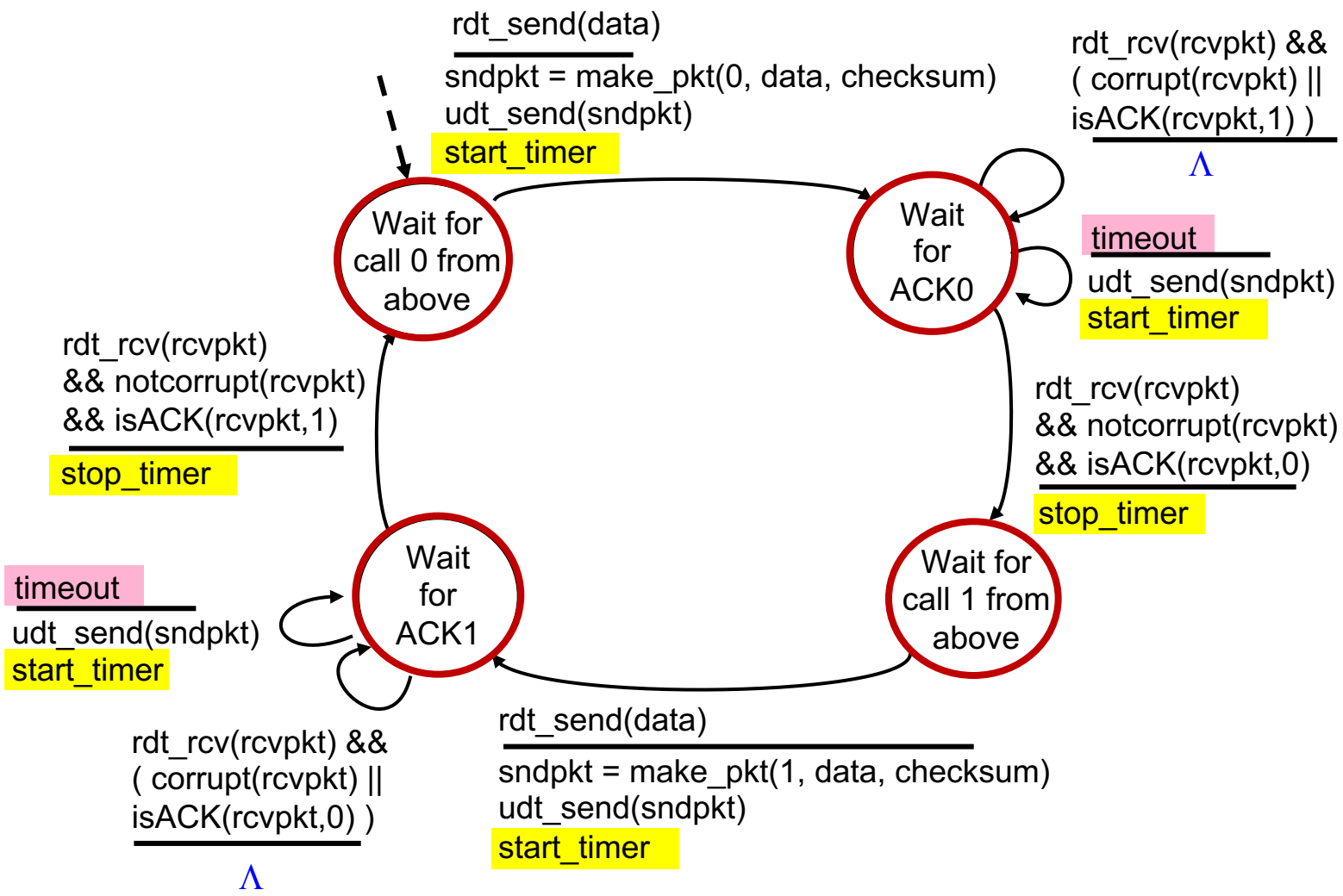
- 만약, 패킷(또는 ACK)이 손실된 것이 아니라, 지연된 경우라면
  - ✓ 재전송은 중복 패킷이 되지만, 순서 번호를 통해 해결됨

timeout

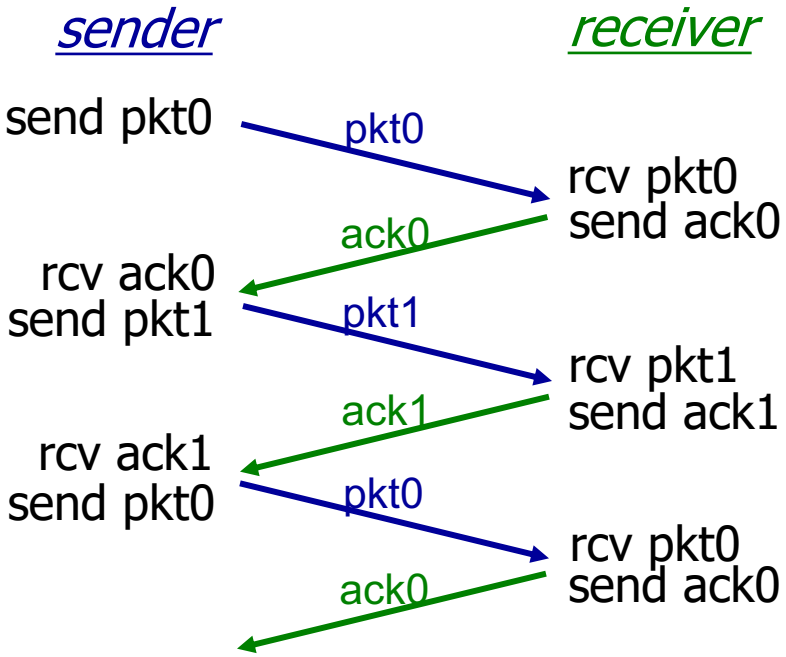
rdt 3.0: 송신자


rdt_send(data)
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
start_timer
Wait for call 0 from above
Wait for ACK0
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,0)
stop_timer
Wait for call 1 from above
Wait for ACK1
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,1)
stop_timer
rdt_send(data)
sndpkt = make_pkt(1, data, checksum)
udt_send(sndpkt)
start_timer

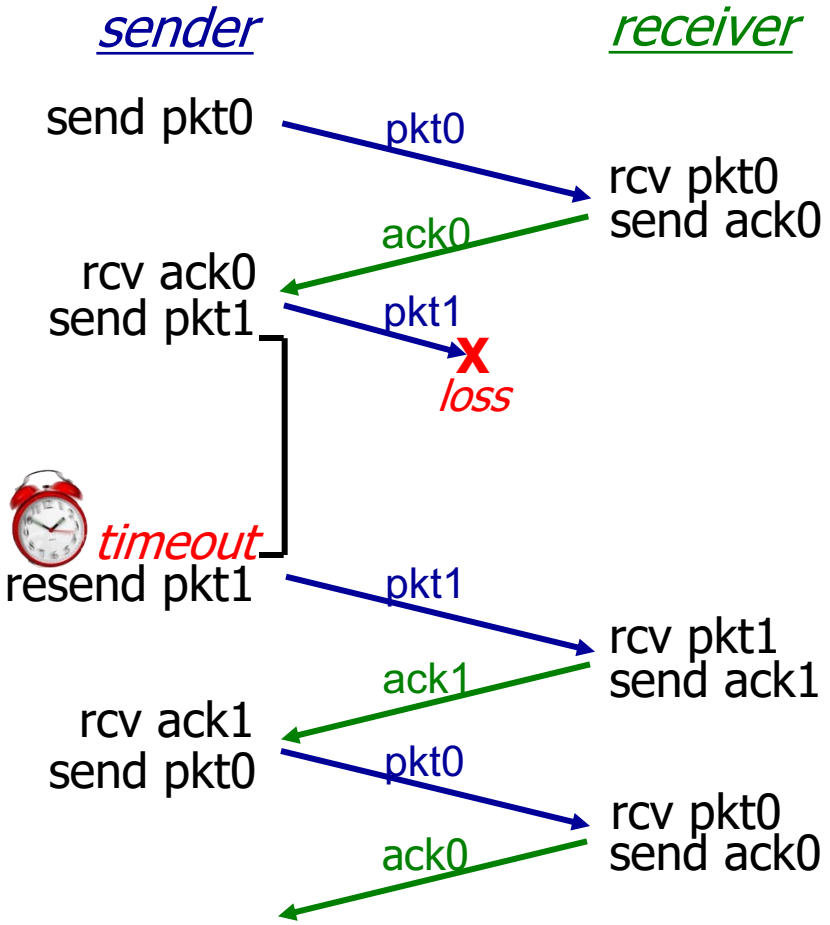
rdt 3.0: 송신자


rdt_send(data)
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
start_timer
Wait for call 0 from above
Wait for ACK0
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,1))
Λ
timeout
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,1)
stop_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,0)
stop_timer
Wait for ACK1
Wait for call 1 from above
timeout
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,0))
Λ
rdt_send(data)
sndpkt = make_pkt(1, data, checksum)
udt_send(sndpkt)
start_timer

rdt 3.0: 동작

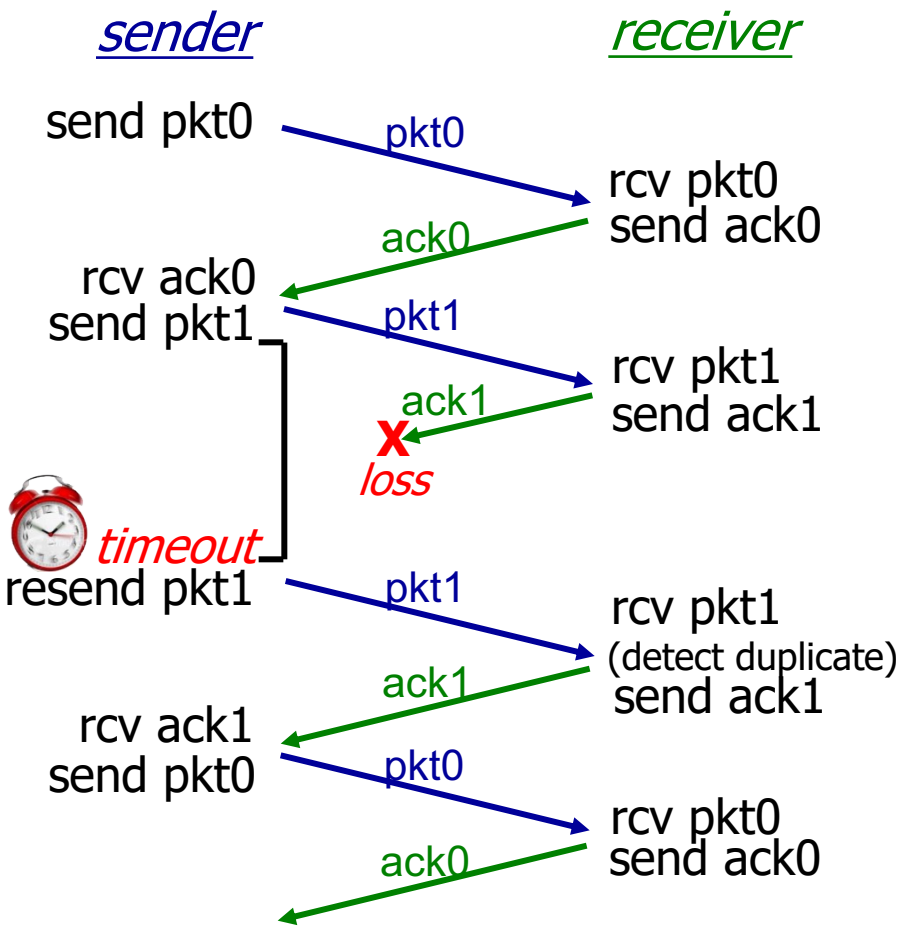


(a) 패킷 손실이 없는 경우

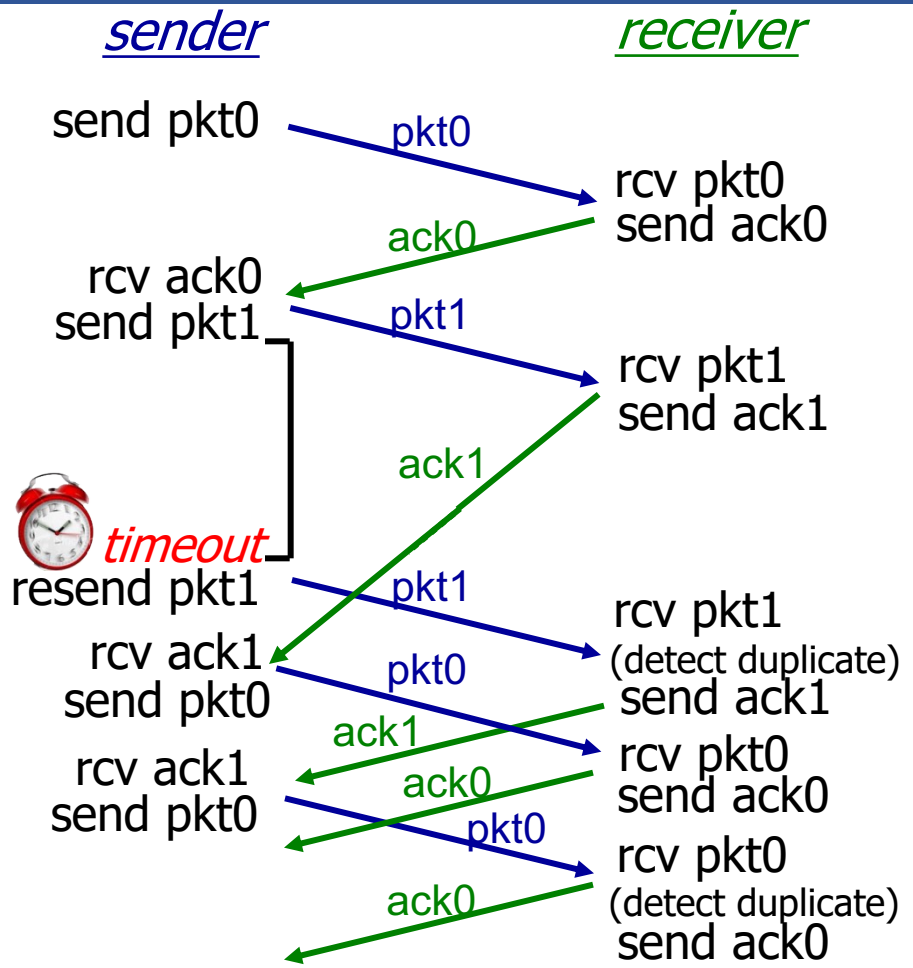


(b) 패킷 손실이 있는 경우

rdt 3.0: 동작



(c) ACK이 손실된 경우



(d) 지연된 ACK으로 인한 timeout 발생

rdt 3.0: 성능

- rdt 3.0은 기능적으로 잘 동작하지만, 성능이 떨어짐
  - "stop and wait(전송 후 대기)" 방식으로 동작하기 때문임
- 예제
  - 1Gbps 전송률(***R***)의 링크, RTT = 30ms, 데이터 패킷 크기(***L***) 8000비트
  - 1개의 데이터 패킷에 대한 전송 지연

$$D_{trans} = \frac{L}{R} = \frac{8000\ bits}{10^9\ bits/sec} = 8\ microsecs$$

  - 송신자 이용률utilization "U_{sender}": 송신자가 "데이터 패킷을 보내기 위해 걸린 전체 시간"에 대한 송신자가 "데이터 패킷을 실제로 보낸 시간"의 비율

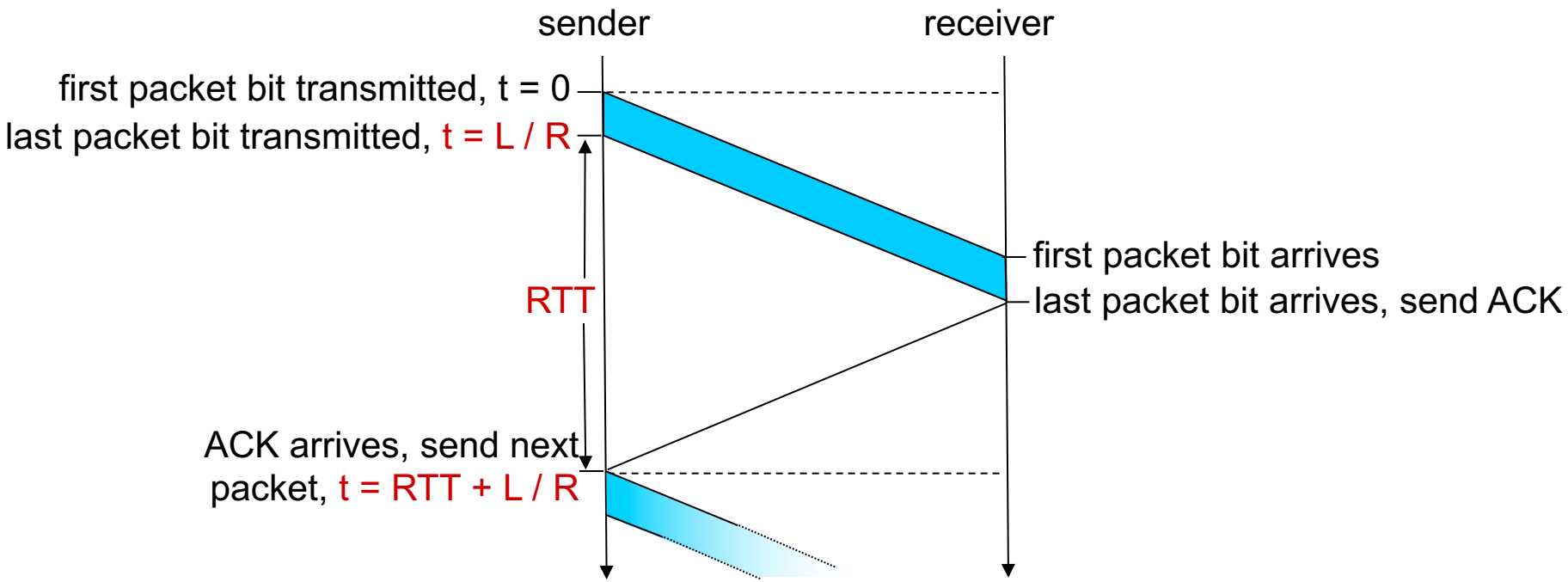
$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

여기서 RTT는 30ms로 가정

  - 송신자는 30.008ms 동안 8000 비트만 처리
    - ✓ 8000bit / 30.008ms = 266596bps = 약 267kbps 처리율
    - ✓ 1Gbps 링크에서 267kbps라니!!!!!!

→ 네트워크 프로토콜이 물리 자원의 사용을 제한함!

rdt 3.0: 전송 후 대기(Stop-and-Wait) 동작



$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

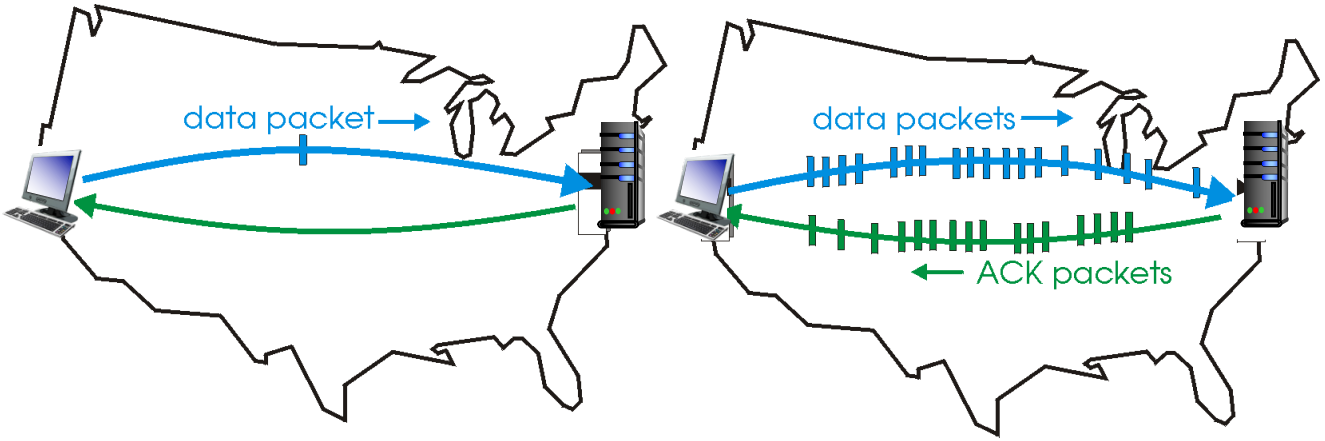
파이프라인 프로토콜 Pipelined Protocol

- 파이프라이닝pipelining

  : 송신자가 ACK을 기다리지 않고 여러 패킷을 전송

  ✓ 비교) stop-and-wait: 송신자는 ACK을 받은 후에 다음 패킷을 전송

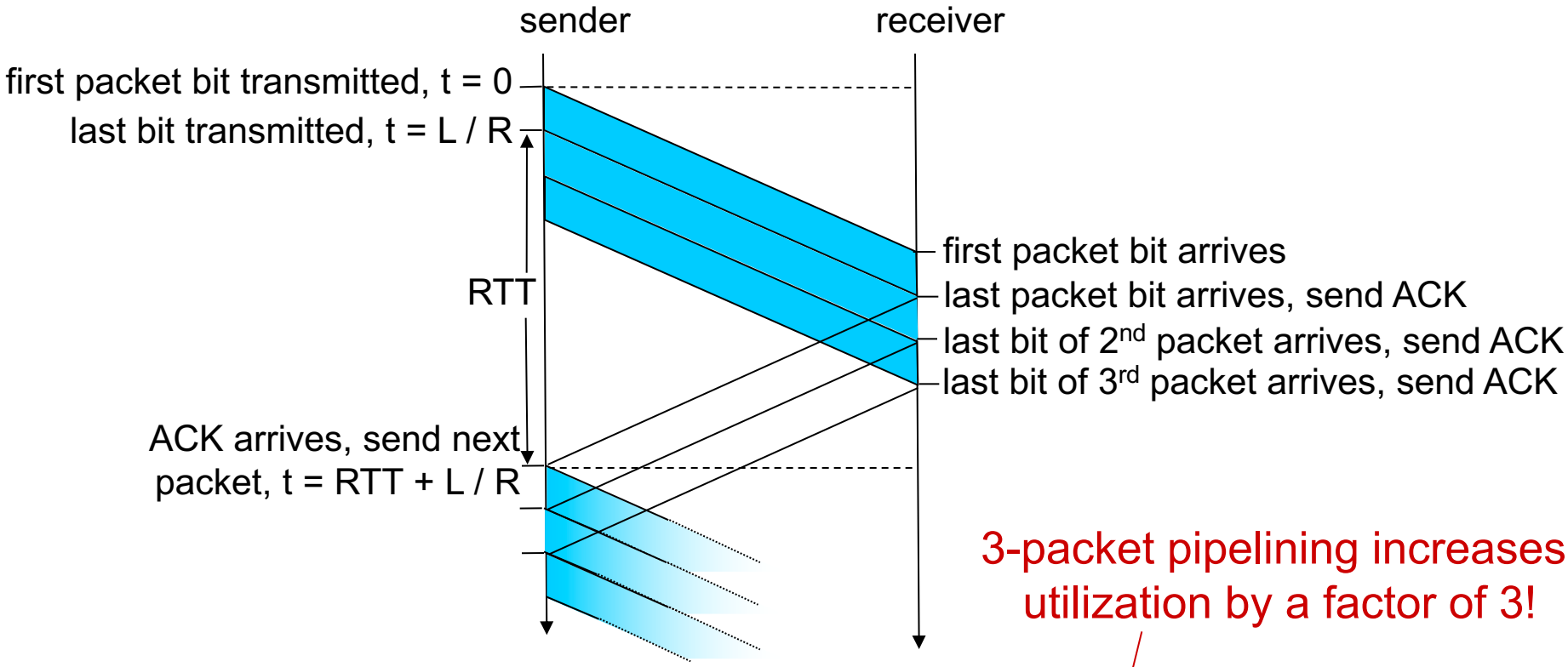
  - 순서 번호의 범위가 증가되어야 함
    ✓ 여러 패킷을 구분하기 위해
  - 송신 측과 수신 측이 패킷을 버퍼링해야 함
    ✓ 여러 패킷 중 일부 패킷이 에러 발생/손실되는 경우를 위해



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

- GBN(Go-Back-N)과 SR(Selective Repeat)이 대표적인 파이프라인 프로토콜임

파이프라이닝: 향상된 이용률



3-packet pipelining increases utilization by a factor of 3!

$$U_{sender} = \frac{3L/R}{RTT + L/R} = \frac{.0024}{30.008} = 0.00081$$

파이프라이닝: 개요

Go-Back-N

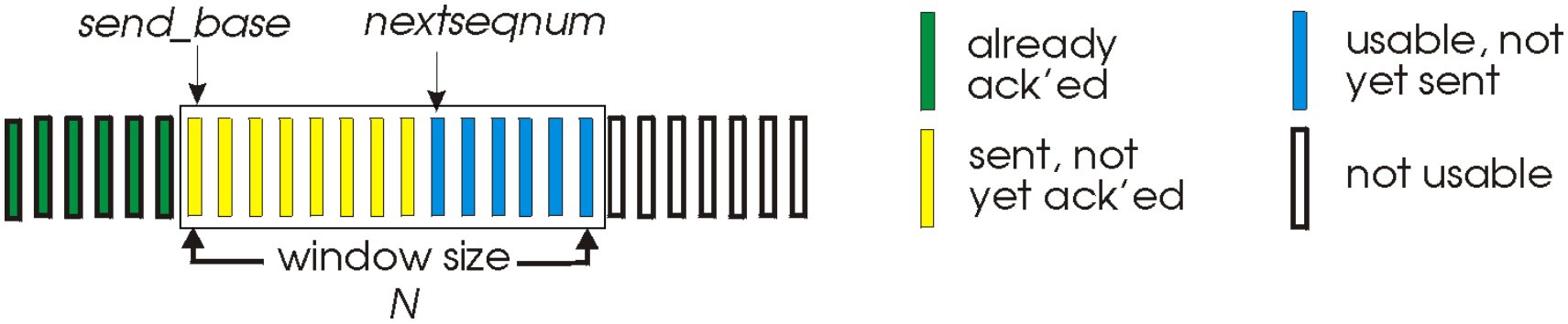
- 송신자는 ACK 수신 없이 최대 N개의 패킷을 전송할 수 있음
- 수신자는 누적 ACK(cumulative ack)만 전송
  - ✓ 특정 순서 번호까지 모든 패킷을 (누적해서) 잘 수신하였다는 것을 의미
  - ✓ 수신된 패킷들의 순서 번호에 "갭"이 있으면, ACK을 전송하지 않음
    - 예) 1, 2, 3, 5의 순서 번호를 가진 패킷을 받으면, 5에 대해 ACK을 전송하지 않음
- 송신자는 가장 오래된 "전송되었지만 ACK 수신 못한 패킷"에 대해서만 타이머를 가짐
  - ✓ 타이머가 만료되면, ACK을 수신하지 못한 모든 패킷을 재전송

Selective Repeat

- 송신자는 ACK 수신 없이 최대 N개의 패킷을 전송할 수 있음
- 수신자는 각 패킷에 대한 개별 ACK(individual ACK) 전송
- 송신자는 "전송되었지만 ACK 수신 못한 패킷" 각각에 대한 타이머를 가짐
  - ✓ 타이머가 만료되면, 해당 패킷만 재전송

Go-Back-N: 송신자

- 송신자
  - 패킷 헤더에 k 비트 순서 번호(seq #)
  - 최대 크기가 N인 윈도우window
    - ✓ ACK 수신하지 않은 연속적인 N개의 패킷을 버퍼링
    - ✓ 슬라이딩 윈도우sliding window



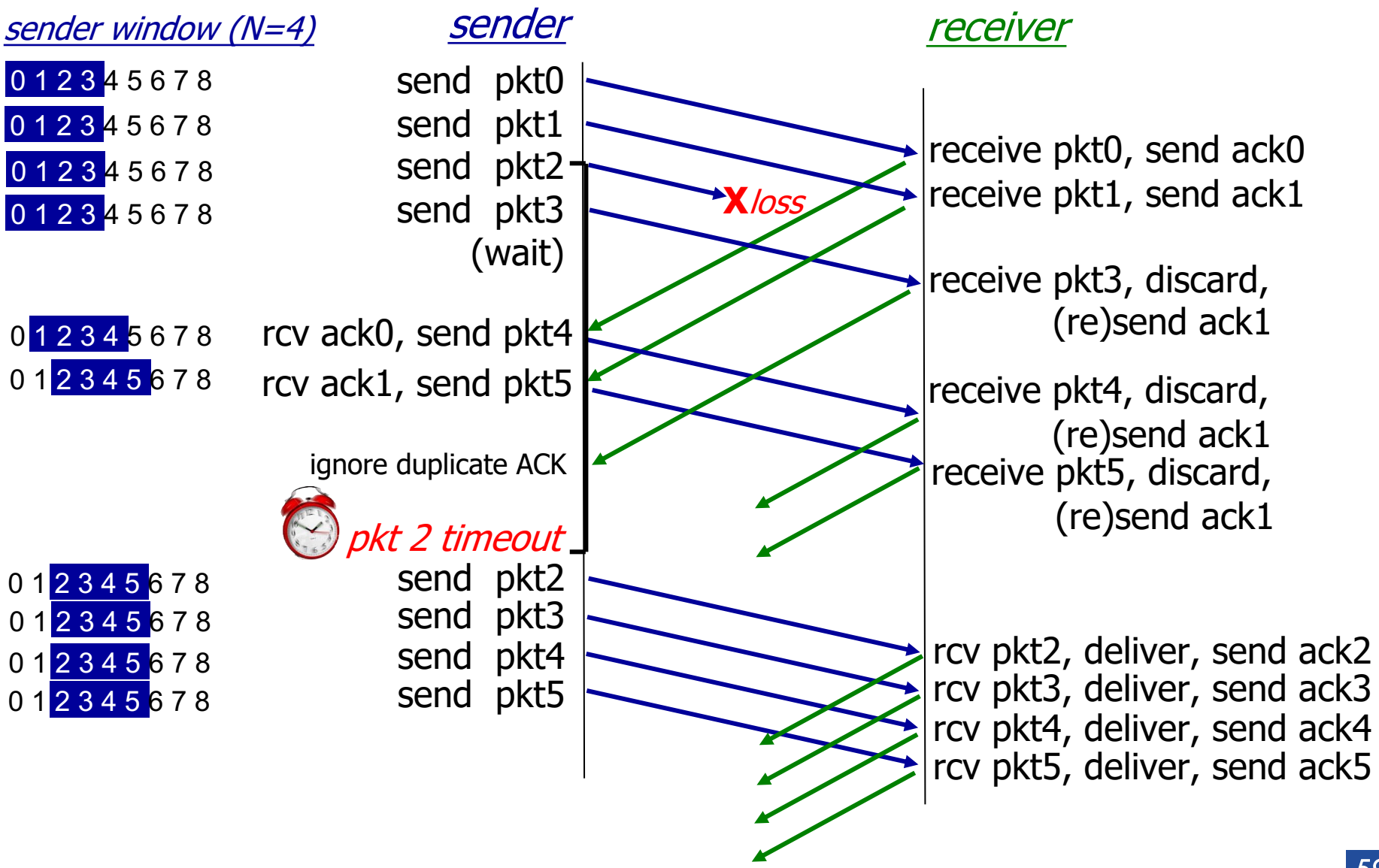
- ✓ ACK(n): 누적 ACK (cumulative ACK)
  - 수신자가 "순서 번호 n까지 정상적으로 수신"하였음을 의미함
  - 송신자는 중복 ACK을 수신할 수 있음
- ✓ 가장 오래된 ACK 수신 못한 패킷에 타이머 설정
- ✓ timeout(n): 순서 번호 n인 패킷에 대한 ACK을 정해진 시간 안에 받지 못한 경우
  - 윈도우에 있는 순서 번호 n 이상인 모든 패킷들을 재전송

Go-Back-N: 수신자

- ACK-only
  - 정상적으로 수신한 패킷 중 가장 큰 순차(in-order) 순서 번호를 가진 패킷에 대해 ACK을 보냄
    - ✓ 예) 1, 2, 3 수신 후, 5를 수신하면 ACK(3)을 보냄
  - 중복 ACK을 보낼 수 있음
    - ✓ 예) 위 상황에서 6을 수신하면 ACK(3)을 보냄
  - expectedseqnum(다음에 수신해야 할 패킷의 순서 번호) 값만 기억하면 됨

- 비순차(out-of-order) 패킷
  - 버퍼링하지 않고 버림
  - 가장 큰 순차 순서 번호로 ACK을 다시 보냄

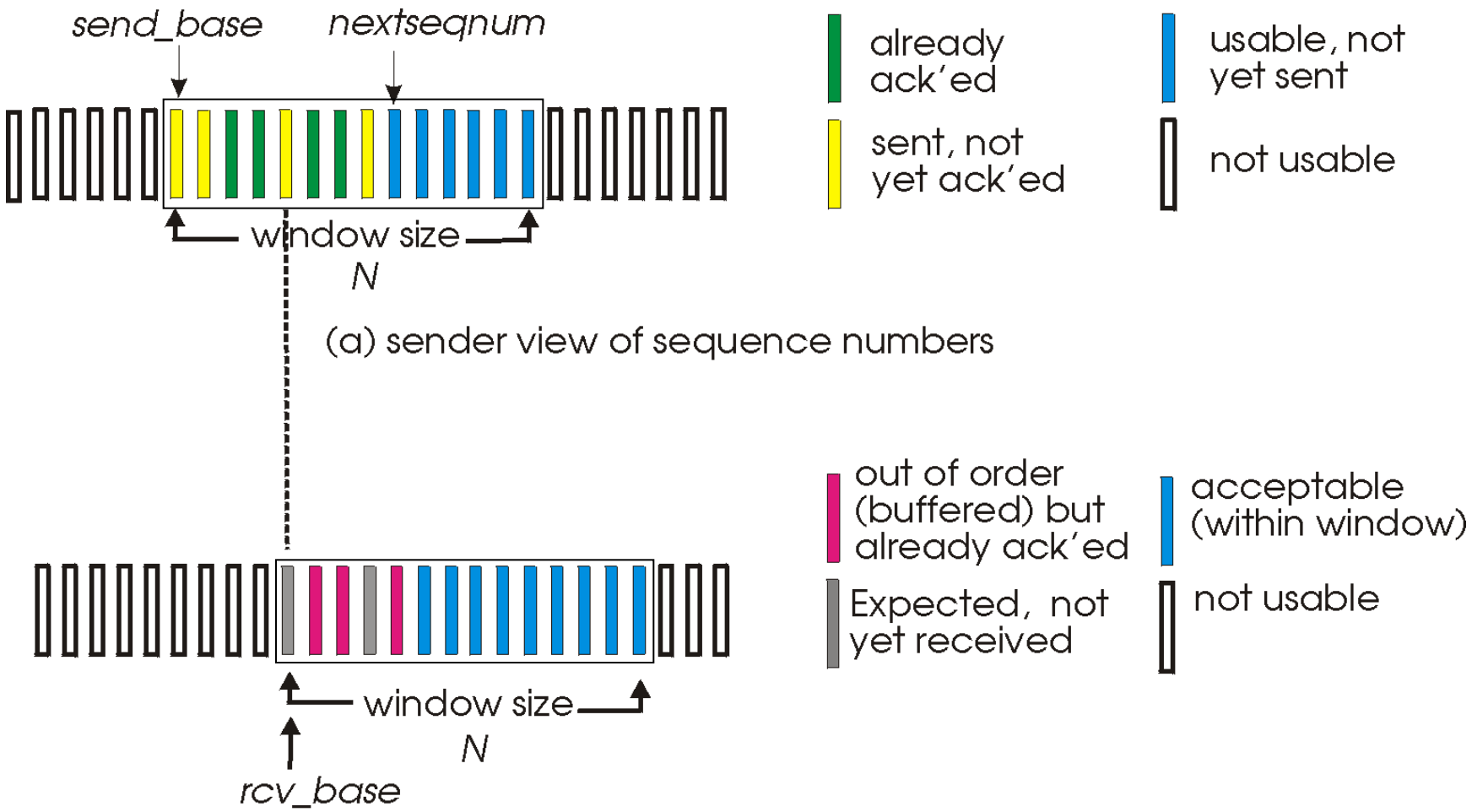
Go-Back-N: 동작 예제



Selective Repeat: 개요

- 수신자는 올바르게 수신된 모든 패킷들에 대해 개별적individually으로 ACK을 전송
  - 상위 계층에 순차적(in-order)으로 전달하기 위해 비순차(out-of-order) 패킷들을 버퍼링
- 송신자는 ACK 응답을 받지 못한 패킷들만 재전송
  - 각 패킷(전송되었으나 아직 ACK을 받지 못한)에 대해 타이머 설정
- 송신자 윈도우
  - N개의 연속적인 순서 번호를 가짐
  - (GBN과 마찬가지로) 윈도우 크기 N은 ACK 수신 못한 패킷들의 수를 제한함

Selective Repeat: 송신자, 수신자 윈도우



Selective Repeat: 송신자, 수신자의 동작

송신자

상위계층에서 메시지 수신:

- 윈도우에 가용한 다음 순서 번호가 있으면, 패킷 전송

timeout(n):

- 패킷 n을 재전송하고, 타이머를 재설정

ACK(n) in [sendbase,sendbase+N-1]:

- 패킷 n을 수신한 것으로 표시
- n이 ACK을 받지 않은 가장 작은 순서 번호이면, sendbase 를 다음 ACK을 받지 않은 순서 번호로 전진

수신자

패킷 n in [rcvbase, rcvbase+N-1]:

- ACK(n) 전송
- 비순차 패킷: 버퍼에 저장
- 순차 패킷: (버퍼에 저장된 이전 패킷들과 함께) 상위 계층으로 전달, 윈도우를 다음 수신하지 않은 순서 번호로 전진

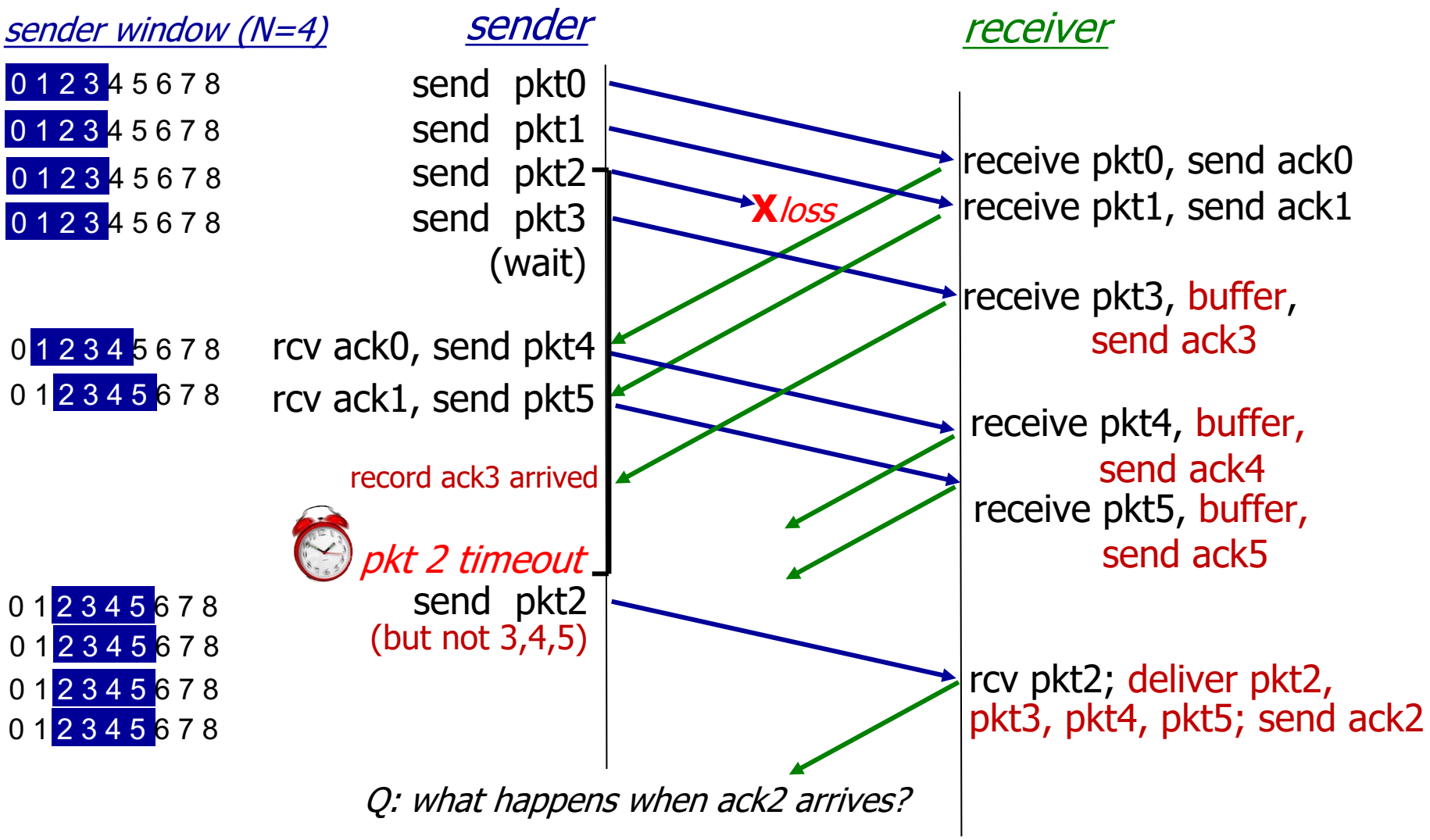
패킷 n in [rcvbase-N,rcvbase-1]:

- ACK(n)

이외의 경우:

- 무시

Selective Repeat: 동작 예제

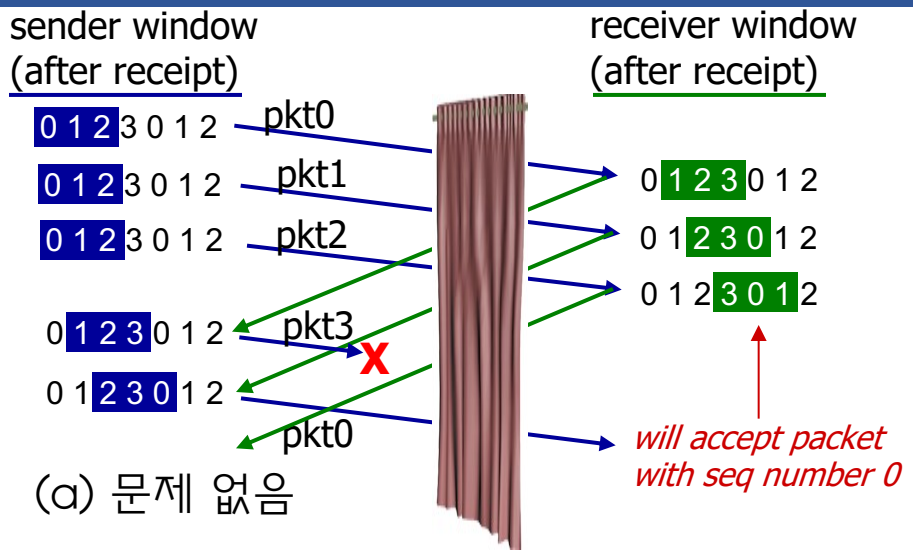


Selective Repeat : 문제점

- 예제
  - 순서번호: 0, 1, 2, 3
  - 윈도우 크기=3
  - 수신자는 두 시나리오의 차이점을 알 수 없음
  - (b) 시나리오에서는 송신자가 보낸 중복 패킷이 수신자에서는 새로운 패킷으로 인식됨

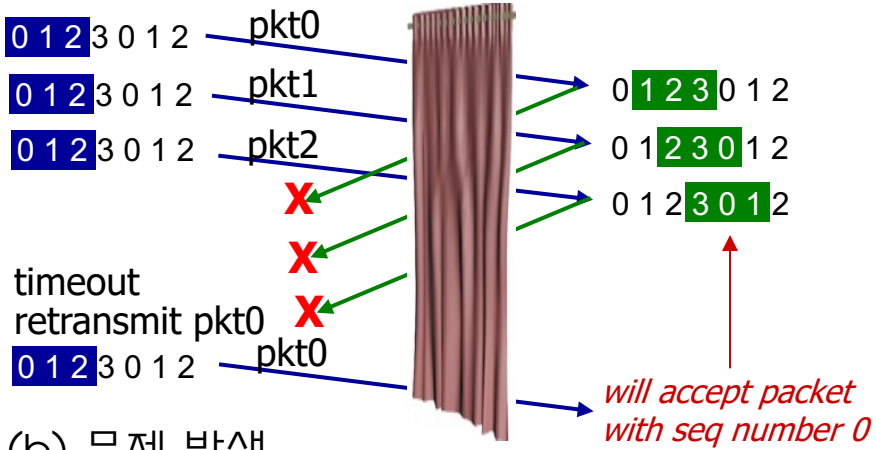
Q. (b)에서 문제점을 피하려면?

A. 윈도우 크기가 순서 번호 크기의 절반보다 작거나 같아야 함



(a) 문제 없음

수신자는 송신자를 볼 수 없음
두 경우에 수신자의 동작은 동일함
문제점 발생!!!



(b) 문제 발생

요약

- 멀티플렉싱, 디멀티플렉싱
  - TCP, UDP
- 신뢰적 데이터 전송의 원리
- 파이프라이닝
  - Go-Back-N
  - Selective Repeat