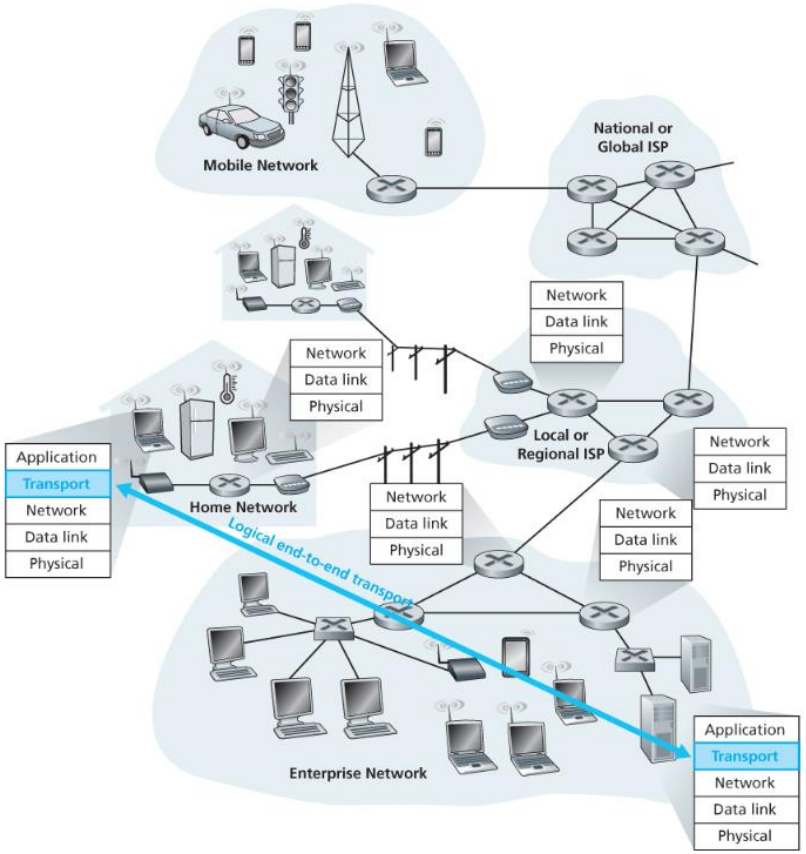
지능형 IoT 네트워크

- 3주차 트랜스포트계층 -

2024.09.18.

1. 트랜스포트 계층 서비스 및 개요

- **Transportation 애플리케이션:**
  - 각기 다른 호스트에서 동작하는 애플리케이션 프로세스 간의 논리적 통신(Logical communication)
  - 트랜스포트 계층 프로토콜은 네트워크 라우터가 아닌 종단 시스템에서 구현



① 송신 측의 트랜스포트 계층은 송신 애플리케이션 프로세스로부터 수신한 메시지를 트랜스포트 계층 패킷으로 변환한다. (Transport Layer Segment: L4-PDU)
- 애플리케이션 메시지를 작은 조각으로 분할
- 각각의 조각에 트랜스포트 계층 헤더를 추가

② 트랜스포트 계층은 송신 종단 시스템에 있는 네트워크 계층으로 세그먼트를 전달한다.

③ 수신 측에서 네트워크 계층은 데이터 그램으로부터 트랜스포트 계층 세그먼트를 추출하고 트랜스포트 계층으로 세그먼트를 보낸다.

① 트랜스포트 계층은 수신 애플리케이션에서 세그먼트 내부의 데이터를 이용할 수 있도록 수신된 세그먼트를 처리한다.

※ 하나이상의 트랜스포트 계층 프로토콜을 사용할 수 있음.
e. g. , TCP, UDP

1.1 트랜스포트 계층과 네트워크 계층 사이의 관계

- 트랜스포트 계층 프로토콜: 각기 다른 호스트에서 동작하는 프로세스들 사이의 논리적 통신을 제공
- 네트워크 계층 프로토콜은 호스트들 사이의 논리적 통신을 제공

•트랜스포트 계층 프로토콜은 종단 시스템에 존재하며, 애플리케이션 프로세스에서 네트워크 계층 사이에서 메시지를 운반하는 역할을 한다.

•메시지가 네트워크 계층 내부에서 어떻게 이동하는지는 언급하지 않는다.
즉, Application을 다루는(작성하거나 run 하는) 입장에서는 앞으로 이루어질 physical connection을 염려하지 않아도 된다.

•트랜스포트 계층이 제공할 수 있는 서비스는 하위 네트워크 계층 프로토콜의 서비스 모델에 의해 제약받는다.

- 네트워크 계층 프로토콜이 호스트 사이에서 전송되는 트랜스포트 계층 세그먼트에 대한 지연 보장이나 대역폭 보장을 제공할 수 없다면, 트랜스포트 계층 프로토콜은 프로세스끼리 전송하는 메시지에 대한 지연 보장이나 대역폭 보장을 제공할 수 없다.

•하위 네트워크 프로토콜이 상응하는 서비스를 제공하지 못할 때도, 특정 서비스는 트랜스포트 프로토콜에 의해 제공될 수 있다.

- 하위 네트워크 프로토콜이 비신뢰적 일 때, 트랜스포트 계층이 애플리케이션에게 신뢰적인 데이터 전송 서비스를 제공할 수 있다.

1.2 인터넷 트랜스포트 계층 개요

INDUSTRIAL AI RESEARCH CENTER

Transmission Control Protocol, TCP

- 신뢰적이고 연결지향형 서비스를 제공한다.(reliable data transfer)
- 혼잡제어(congestion control): 혼잡한 네트워크 링크에서 각 TCP연결이 링크의 대역폭을 공평하게 공유하여 통과하도록 해준다.

User Datagram Protocol, UDP

- 비신뢰적이고 비연결형인 서비스를 제공한다.
- UDP 트랜스포트 프로토콜을 사용하는 애플리케이션은 허용이 되는 한, 그것이 만족하는 어떤 속도로도 전송할 수 있다.

세그먼트와 데이터 그램

- 세그먼트(segment) : 트랜스포트 계층 패킷을 일컫는 말

➔TCP패킷을 세그먼트(segment), UDP패킷을 데이터 그램(datagram)

인터넷 프로토콜(Internet Protocol, IP)

- IP 서비스 모델은 호스트들 간에 논리적인 통신을 제공하는 최선형 전달 서비스(best-effort delivery service)

➔ IP가 통신하는 호스트들 간에 세그먼트 전달하기 위해 최대한 노력을 하지만, 어떤 보장도 하지 않는다.

➔ IP는 비신뢰적인 서비스(unreliable service)

UDP와 TCP의 서비스 모델

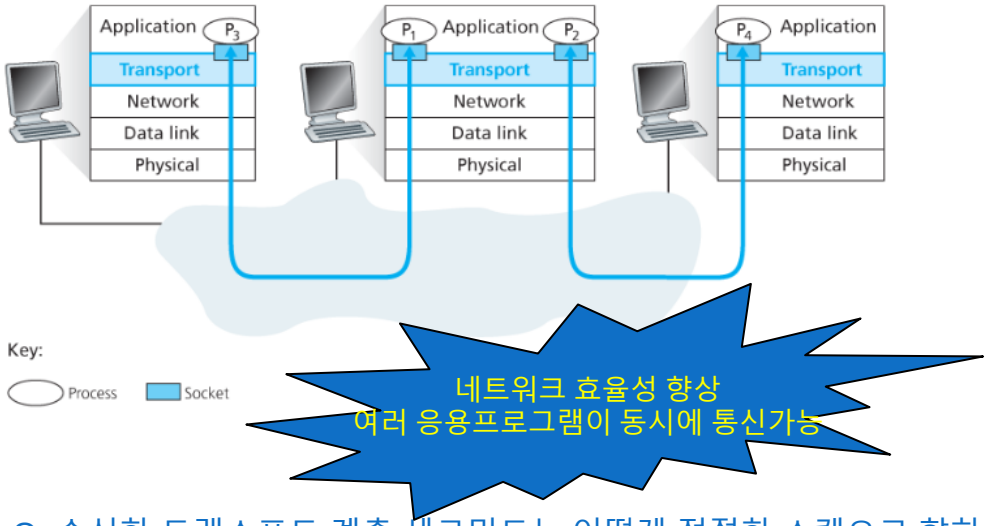
- 트랜스포트 계층 다중화(transport-layer multiplexing)과 역다중화(demultiplexing) : Host-to-Host ➔ Process to Process 전달
  - 종단 시스템 사이의 IP전달 서비스를 /종단 시스템에서 동작하는 두 프로세스 간의 전달 서비스로 확장한다.
- 헤더에 오류 검출 필드를 포함

2. 다중화와 역다중화

INDUSTRIAL AI RESEARCH CENTER

- 목적지 호스트에서의 트랜스포트 계층은 바로 아래 네트워크 계층으로부터 세그먼트를 수신한다.
  - 트랜스포트 계층은 호스트에서 동작하는 해당 애플리케이션 프로세스에게 이 세그먼트의 데이터를 전달하는 의무를 가진다.

- 트랜스포트 계층은 세그먼트(데이터)를 중간 매개자인 socket에게 전달한다.
  - 프로세스는 네트워크 애플리케이션의 한 부분으로서 소켓을 가지고 있다.
  - 이는 네트워크에서 프로세스로, 한 프로세스로부터 네트워크로 데이터를 전달하는 출입구 역할
  - 각각의 소켓은 하나의 유일한 식별자, Port number를 가진다.



Multiplexing:
여러 개의 응용 프로그램 또는 서비스가 하나의 통신 채널을 통해 데이터를 전송할 수 있도록 하는 과정이다. 즉 여러 데이터 스트림을 하나의 데이터 스트림으로 결합하여 효율적으로 전송하는 방법이다. 트랜스포트 계층에서는 주로 포트 번호를 사용하여 각 데이터 스트림을 식별한다. 예) hTTP는 80, FTP 21
(여러 데이터 스트림을 하나로 결합하여 전송)

Demultiplexing:
수신 측에서 멀티플렉싱된 데이터를 다시 개별 데이터 스트림으로 분리하는 과정입니다. 수신한 데이터 패킷의 포트 번호를 통해 어떤 응용 프로그램으로 데이터를 전달할지 결정합니다. 이렇게 함으로써, 동일한 물리적 채널을 통해 전송된 여러 응용 프로그램의 데이터가 각기 다른 목적지로 올바르게 전송됨
(수신한 데이터를 원래 여러 스트림으로 분리)

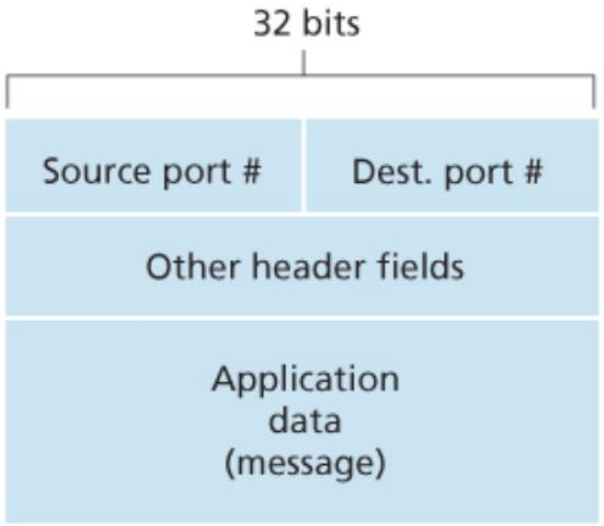
Q. 수신한 트랜스포트 계층 세그먼트는 어떻게 적절한 소켓으로 향하는가?

각각의 트랜스포트 계층 세그먼트는 세그먼트에 필드 집합을 가지고 있으며, 트랜스포트 계층은 수신 소켓을 식별하기 위해 이러한 필드를 검사한 후 해당 소켓으로 보낸다.

트랜스포트 계층 다중화의 두 가지 요구사항

- 소켓은 유일한 식별자를 가짐(= 포트 번호)
  - 각 세그먼트는 세그먼트가 전달될 적절한 소켓을 가리키는 특별한 필드를 갖는다
    - 출발지 포트 번호 필드(source port number field)
    - 목적지 포트 번호 필드(destination port number field)

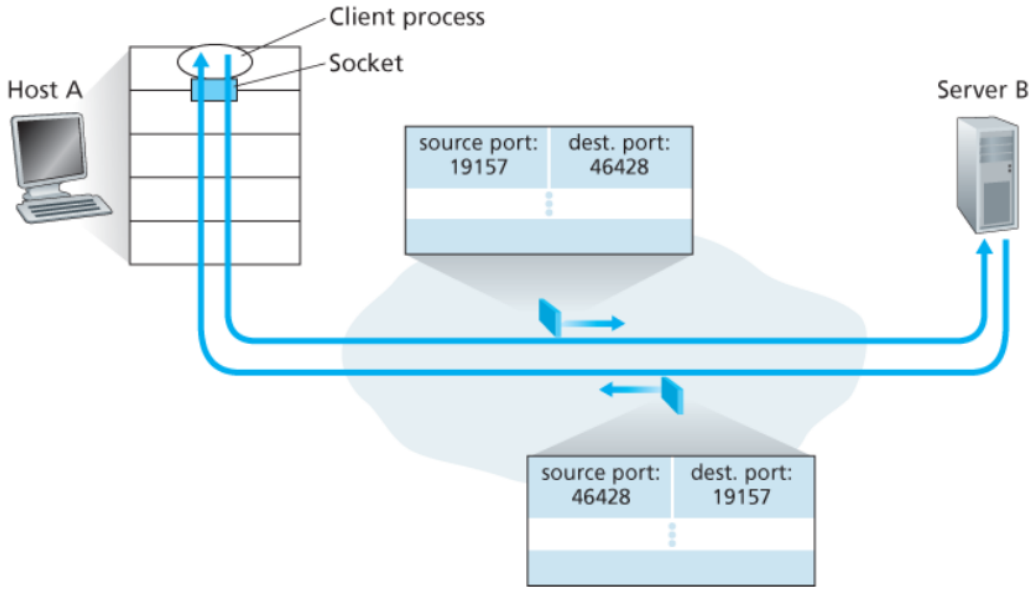
- 역다중화 서비스의 순서
  1. 호스트의 각 소켓은 포트 번호를 할당 받음
  2. 세그먼트가 호스트에 도착하면,
     - 트랜스포트 계층은 세그먼트 안의 목적지 포트 번호를 검사하고 그에 상응하는 소켓으로 세그먼트를 보낸다
  3. 세그먼트의 데이터는 소켓을 통해 해당되는 프로세스로 전달된다.



비연결형 다중화와 역다중화

UDP소켓은 목적지 IP주소와 목적지 포트 번호로 구성된 두 요소로 된 집합에 의해 식별된다.
→ 만약 2개의 UDP 세그먼트가 같은 목적지 IP주소와 목적지 포트 번호를 가진다면, 이 2개의 세그먼트는 같은 목적지 소켓을 통해 같은 프로세스로 도착

출발지 포트 번호는 회신 주소의 한 부분으로 사용됨.



연결지향형 다중화와 역다중화

TCP소켓은 4개 요소의 집합(four-tuple)에 의해 식별된다.

- 출발지 IP주소
- 출발지 포트번호
- 목적지 IP주소
- 목적지 포트번호

→ 특히, 다른 출발지 IP주소 또는 다른 출발지 포트 번호를 가지고 도착하는 2개의 TCP 세그먼트는 2개의 다른 소켓으로 향하게 된다.

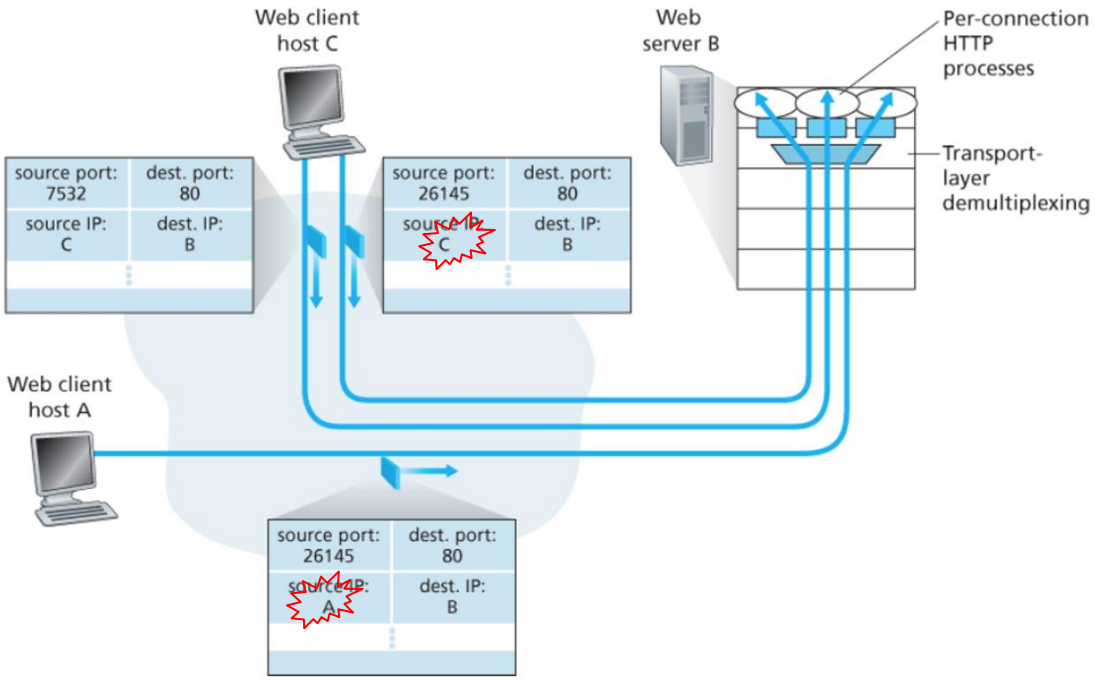
TCP연결 설정:

1) TCP 서버 애플리케이션은 환영 소켓을 갖고 있다.
2) TCP 클라이언트 소켓을 생성하고, TCP 헤더에 설정된 연결 설정 요청 세그먼트를 보낸다.
3) 서버 프로세스로 동작하는 컴퓨터의 호스트 운영체제가 목적지 포트를 포함하는 연결 요청 세그먼트를 수신하면, 이 세그먼트를 해당 포트 번호로 연결 수락을 기다리는 서버 프로세스로 보낸다.
4) 서버는 연결 요청 세그먼트의 4개 요소의 집합에 주목한다. 서버 호스트는 동시에 존재하는 많은 TCP 소켓을 지원할 수 있다.
    - 새롭게 생성된 연결 소켓은 4개 요소의 집합의 네 가지 값에 의해 식별된다.
    - 따라서 그다음에 도착하는 세그먼트의 출발지 포트, 출발지 IP주소, 목적지 포트, 목적지 IP주소가 전부 일치하면, 그 세그먼트는 이 소켓으로 역다중화 될것이다.

웹 서버와 TCP

서버는 각기 다른 클라이언트가 보낸 세그먼트를 출발지 IP주소와 출발지 번호로 구별한다.

같은 웹 서버 애플리케이션과 통신하기 위해 같은 목적지 포트(80)를 이용하는 두 클라이언트에 대한 예시:



Persistent & Non-persistent HTTP

INDUSTRIAL AI RESEARCH CENTER

지속적인(Persistent) HTTP

→ 지속적인 연결의 존속 기간에 클라이언트와 서버는 같은 서버 소켓을 통해 HTTP 메시지를 교환한다.

비지속적인(non-persistent) HTTP

→모든 요청/응답마다 새로운 TCP연결이 생성되고 종료된다.

3. 비연결형 트랜스포트: UDP

INDUSTRIAL AI RESEARCH CENTER

UDP는 트랜스포트 계층 프로토콜이 할 수 있는 최소 기능으로 동작

- 다중화/역다중화
- 간단한 오류 검사 기능 (이외에는 IP에 아무것도 추가하지 않음).

UDP동작순서

1. 애플리케이션 프로세스로부터 메시지를 가져와서 다중화/역다중화 서비스에 필요한 출발지 포트 번호 필드와 목적지 포트 번호 필드를 첨부한다.

2. 출발지 호스트의 IP주소 필드, 목적지 호스트 IP주소 필드를 추가한 후에 최종 트랜스포트계층 세그먼트를 네트워크 계층으로 넘겨준다

3. 네트워크 계층은 트랜스포트 계층 세그먼트를 IP 데이터그램으로 캡슐화하고, 세그먼트를 수신 호스트에게 전달한다.

4. 세그먼트가 수신 호스트에 도착한다면, UDP는 세그먼트의 데이터를 해당하는 애플리케이션 프로세스로 전달하기 위해 목적지 포트 번호를 사용한다.

- **DNS:**
  질의 호스트가 응답을 수신하지 못하면 질의를 다른 네임 서버로 송신하거나, 애플리케이션에게 응답을 수신할 수 없음을 통보

3. 비연결형 트랜스포트: UDP

INDUSTRIAL AI RESEARCH CENTER

UDP 장점:

1. 데이터 전송시 데이터 및 데이터 전송시간을 애플리케이션 레벨에서 정교한 제어 가능

   • UDP 하에서 애플리케이션 프로세스가 데이터를 UDP에 전달하자마자 UDP는 데이터를 UDP세그먼트로 만들고, 그 세그먼트를 즉시 네트워크 계층으로 전달.
   • 실시간 애플리케이션에서 UDP를 사용, 필요한 추가 기능을 구현

2. 연결 설정이 없다 (연결설정 지연 X)
   • 연결 상태가 없다
   -연결상태가 없고 연결상태에서 대한 파라미터도 기록하지 않는다.
   -전용서버는 애플리케이션 프로그램이 UDP에서 동작할 때 좀 더 많은 액티브 클라이언트를 수용 가능

   •작은 패킷 헤더 오버헤드
   (TCP head size = 20byte, UDP head size = 8byte)

TCP:
1.혼잡 제어 매커니즘이 존재한다.
즉, 목적지 호스트들과 출발지 호스트들 사이에서 하나 이상의 링크가 과도하게 혼잡해지면 트랜스포트 계층 TCP 송신자를 제한

2. 신뢰적인 전달이 얼마나 오래 걸리는지에 관계없이 목적지가 세그먼트의 수신 여부를 확인 응답할 때까지 데이터의 세그먼트 재전송을 계속.

UDP 단점:

혼잡제어(Congestion control)를 하지 않는다.

혼잡제어 없이 높은 비트의 비디오 스트리밍을 전송시,

- 라우터에 많은 패킷 오버플로가 발생
  - 소수의 UDP패킷만 출발지-목적지간의 경로를 무사히 통과

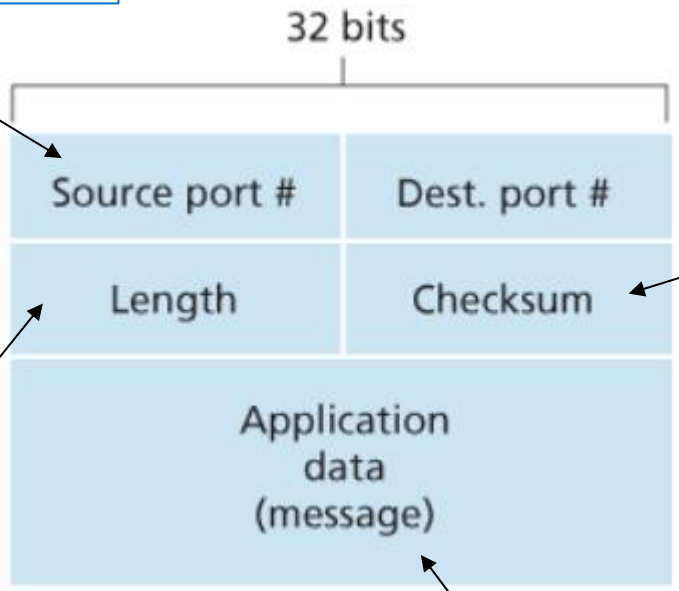
- 제어되지 않는 UDP 송신자에 의해 발생한 높은 손실율은 그 손실율을 감소시키기 위해 TCP 송신자들이 속도를 줄일 것
  → TCP 세션이 혼잡이 발생

UDP를 통한 신뢰적인 데이터 전송:

- UDP는 비신뢰적인 서비스를 제공하지만, 애플리케이션 자체에서 신뢰성을 제공한다면 UDP 사용하면 신뢰적 데이터 전송이 가능
  예) Chrome에서 QUIC(Quick UDP Internet Connection)

3.1 UDP 세그먼트 구조

목적지 호스트가 목적지 종단 시스템에서 동작하는 (역다중화 기능을 수행하는) 정확한 프로세스에게 애플리케이션 데이터 전달



세그먼트에 오류가 발생했는지 검사하기 위해 수신 호스트 사용

헤더를 포함하는 UDP 세그먼트의 길이를 바이트 단위 표시

UDP 데이터그램의 데이터 필드에 위치

3.2 UDP 체크섬

INDUSTRIAL AI RESEARCH CENTER

UDP 체크섬:

세그먼트가 출발지로부터 목적지로 이동했을 때, UDP 세그먼트 안의 비트에 대한 변경사항이 있는지 검사하여 오류 검출을 하기 위함.

1. 송신자 측에서 세그먼트 안에 있는 모든 16비트 워드의 합산에 대해 다시 1의 보수를 수행하며, 합산 과정에 발생하는 오버플로는 윤회식 자리올림(wrap around)을 한다

2. 이 결과값이 UDP 세그먼트의 체크섬 필드에 삽입된다.

3. 수신자에서는 체크섬을 포함한 모든 16비트 워드들이 더해진다.

4. 만약 패킷에 어떤 오류도 없다면 수신자에서의 합은 1111111111111111이 되며, 비트 중에 0이 하나라도 있다면 패킷에 오류가 발생했다는 것이다.

UDP 체크섬 적용 이유:

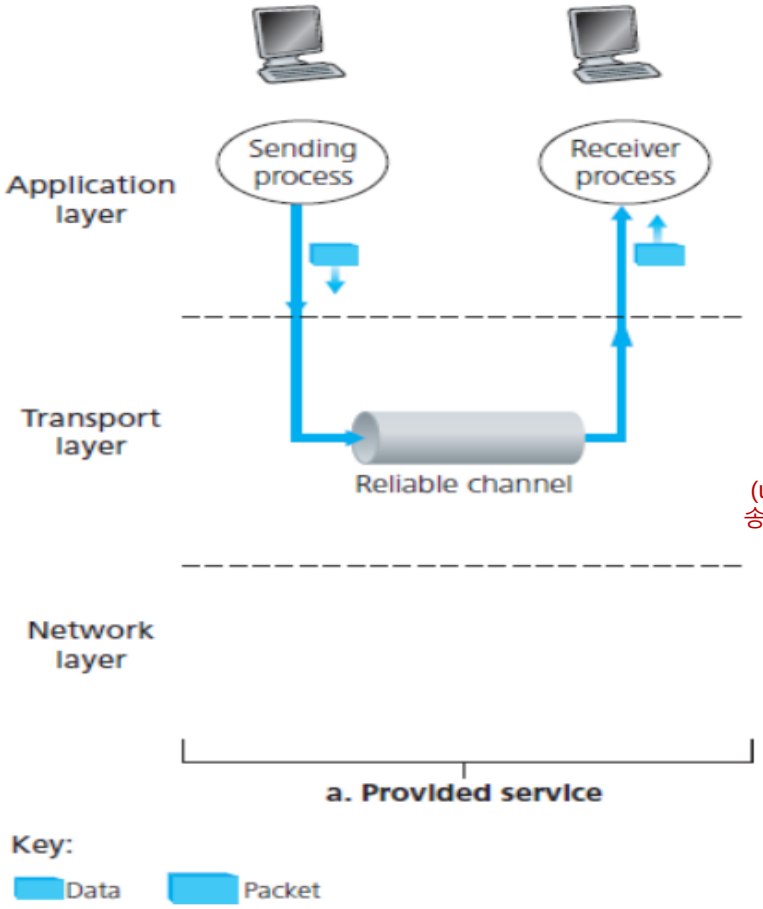
출발지와 목적지 사이의 모든 링크가 오류 검사를 제공한다는 보장이 없기 때문이다.
(세그먼트들이 정확하게 링크를 통해 전송되었을지라도, 세그먼트가 라우터의 메모리에 저장될 때 비트 오류가 발생가능)
※ **UDP는 오류 검사를 제공하지만, 오류를 회복하기 위한 어떤 일도 하지 않는다.** 손상된 세그먼트를 그냥 버리기도 하고 경고와 함께 손상된 세그먼트를 애플리케이션에게 넘기(처리방식은 구현에 따라서 다름)

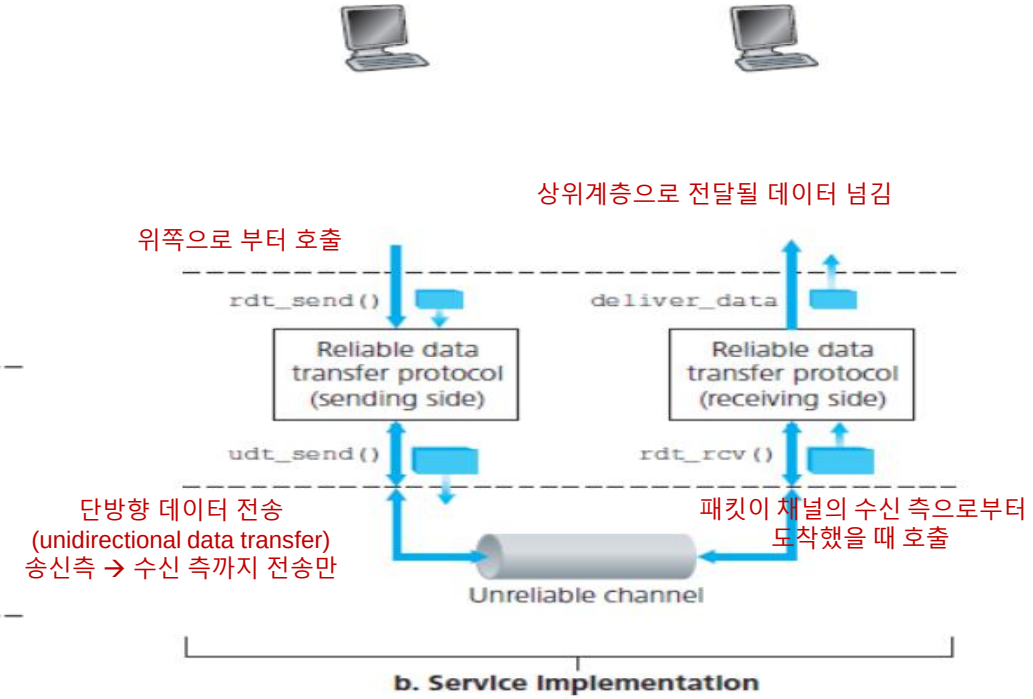
주어진 링크 간의 신뢰성과 메모리의 오류 검사가 보장되지도 않고, 종단 간의 데이터 전송 서비스가 오류 검사를 제공해야 한다면 **UDP는 종단 기반으로 트랜스포트 계층에서 오류검사를 제공 → End-to-End Principle**

End-to-End principle: 하위 레벨에 있는 기능들은 상위 레벨에서 이들은 제공하는 비용과 비교했을 때 중복되거나 거의 가치가 없을 수 있다.

4. 신뢰성 데이터 전송의 원리

신뢰적인 데이터 전송을 구현하는 문제는 트랜스포트 계층 뿐만 아니라 링크 계층과 애플리케이션 계층에서도 발생할 수 있는 문제이다.





상위 계층 객체에게 제공되는 서비스 추상화는 **데이터가 전송될 수 있는 신뢰적인 채널**의 서비스 추상화다.

- 신뢰적인 채널에서는 전송된 데이터가 손상되거나 손실되지 않으며, 모든 데이터는 전송된 순서 그대로 전달된다.
- 신뢰적인 데이터 전송 프로토콜의 의무는 신뢰적인 채널의 서비스 추상화를 구현하는 것이다.

4.1 신뢰적인 데이터 전송 프로토콜 구축

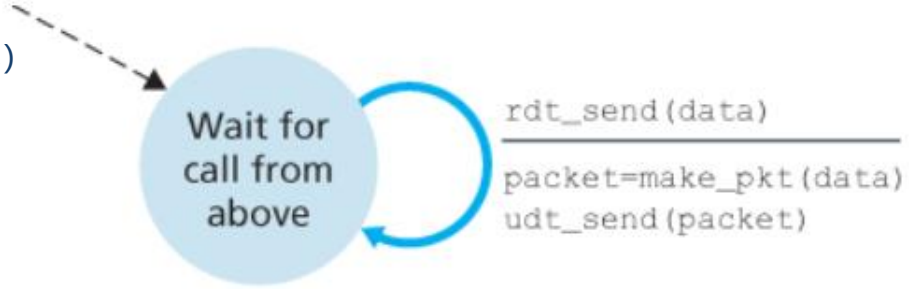
유한상태 머신(finite-state machine, FSM)에 대한 정의 화살표는 한 상태로부터 다른 상태로의 전이를 나타낸다. FSM의 초기 상태는 **점선 화살표**로 표시된다.

전이를 일으키는 이벤트(event)는 변화를 표기하는 가로선 위에 나타낸다. 이벤트가 발생했을 때 취해지는 행동, **액션(action)**은 가로선 아래에 나타낸다.

이벤트 발생 시 어떠한 행동도 취해지지 않거나, 어떠한 이벤트 발생 없이 행동이 취해질 때 **동작이나 이벤트가 없음**을 표시하기 위해 각각 가로선 아래나 위에 기호 **Λ**를 사용한다.

RDT 1.0 : 완벽하게 신뢰적인 채널

- dt_send(data) 이벤트에 의해 (이 이벤트는 상위 계층 애플리케이션의 프로시저 호출(e.g., rdt_send())에 의해 발생)
  - 상위 계층으로부터 데이터를 받아들이고
  - 데이터를 포함한 패킷을 생성한다.
- (make_pkt(data)) 그러고 난 후 패킷을 채널로 송신한다



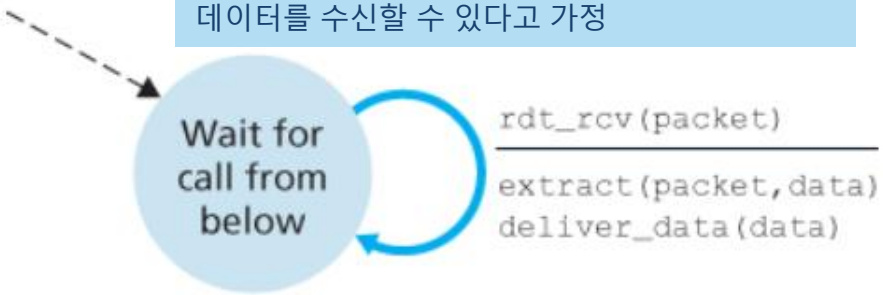
a. rdt1.0: sending side

완전히 신뢰적인 채널에서는 오류가 생길 수 없으므로 수신 측이 송신 측에게 어떤 feedback도 제공할 필요가 없다

수신자는 송신자가 데이터를 송신하자마자 데이터를 수신할 수 있다고 가정

rdt는 rdt_rcv(packet) 이벤트에 의해 하위의 채널로부터 패킷을 수신한다. : 이 이벤트는 하위 계층 프로토콜로부터의 프로시저 호출(e.g., rdt_rcv())에 의해 발생한다.

- 패킷으로부터 데이터를 추출하고 (extract(packet, data))
- 데이터를 상위 계층으로 전달한다. (deliver_data(data))



b. rdt1.0: receiving side

RDT 2.0 : 비트 오류가 있는 채널

패킷 안의 비트들이 하위 채널에서 손상되는 모델. 일반적으로 비트 오류는 패킷이 전송되거나 버퍼링 될 때 네트워크의 물리적인 구성요소에서 발생.

자동 재전송 요구(Automatic Repeat reQuest, ARQ) 프로토콜

- 긍정 확인응답(positive acknowledgment, "OK")
- 부정 확인응답(negative acknowledgment, "그것을 반복해주세요")

제어 메시지는 정확하게 수신되었는지 또는 잘못 수신되어 반복이 필요한지를 수신자가 송신자에게 공지 가능.

비트 오류를 처리하기 위해 ARQ 프로토콜의 3 가지 부가 프로토콜 기능:

오류 검출

비트 오류가 발생했을 때 수신자가 검출할 수 있는 기능이 필요 → checksum
- 수신자가 패킷 비트 오류를 검출하고 복구가능

수신자 피드백

송신자가 수신자의 상태를 알기 위한 유일한 방법은 수신자가 송신자에게 피드백을 제공
수신자의 상태 : 패킷이 정확하게 수신되었는지 아닌지 등

e.g., rdt2.0에서는 수신자로부터 송신자 쪽으로 ACK와 NAK 패킷들을

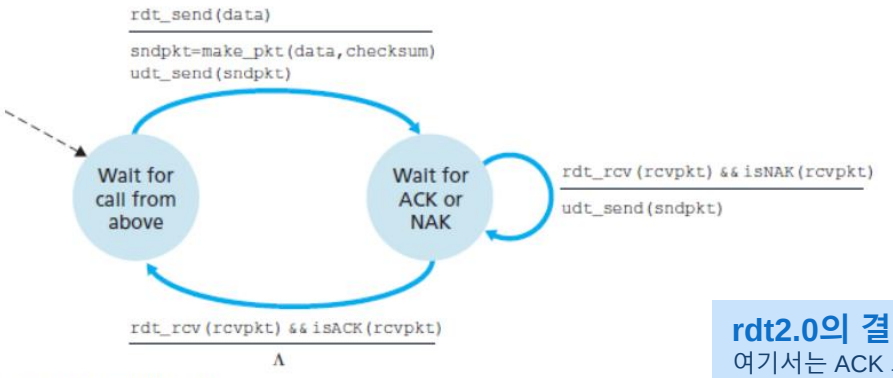
- 긍정 확인 응답(ACK)
- 부정 확인 응답(NAK)

→ 이러한 패킷은 단지 한 비트 길이면 된다. (0 또는 1)

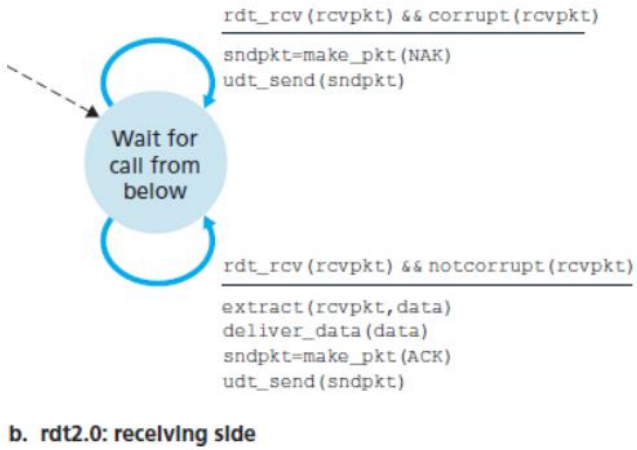
재전송

수신자에서 오류를 가지고 수신된 패킷은 송신자에 의해 재전송(retransmit).

RDT 2.0 FSM (Stop-and-wait)



a. rdt2.0: sending side



b. rdt2.0: receiving side

rdt2.0의 결함

여기서는 ACK 또는 NAK 패킷이 손상될 수 있다는 가능성을 고려하지 않았다.

만약 ACK 또는 NAK가 손상되었다면, 송신자는 수신자가 전송된 데이터의 마지막 부분을 올바르게 수신했는지를 알 방법이 없다

. 대안 1 : 송신자가 검출뿐만 아니라 비트 오류 로부터 회복할 수 있도록 충분한 체크섬 비트들을 추가한다.

대안 2 : 송신자가 왜곡된 ACK 또는 NAK 패킷을 수신할 때 현재 데이터 패킷을 단순히 다시 송신한다

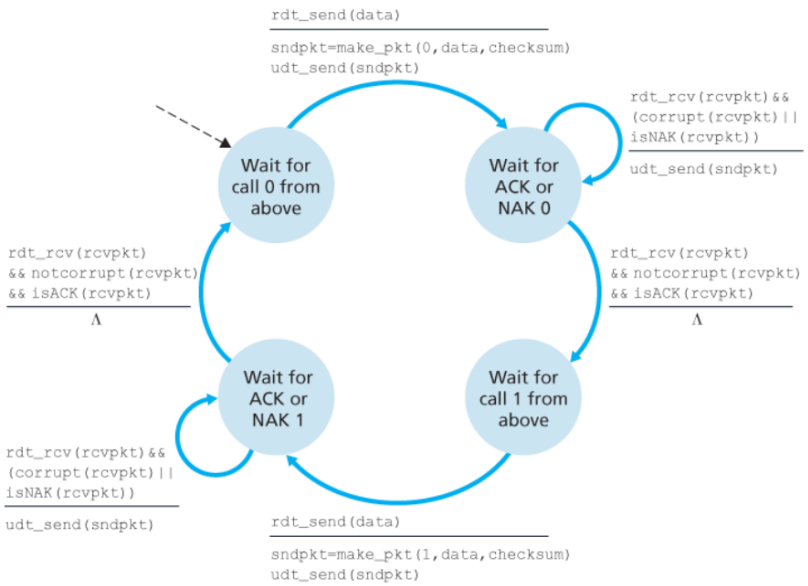
데이터 패킷에 새로운 필드를 추가하고 이 필드 안에 순서 번호(sequence number)를 삽입하는 방식으로 데이터 패킷에 송신자가 번호를 붙인다. 이는 현존하는 데이터 전송 프로토콜에 채택된 방법이다.

수신자는 수신된 패킷이 재전송인지를 결정할 때는 이 순서 번호만 확인하면 된다.

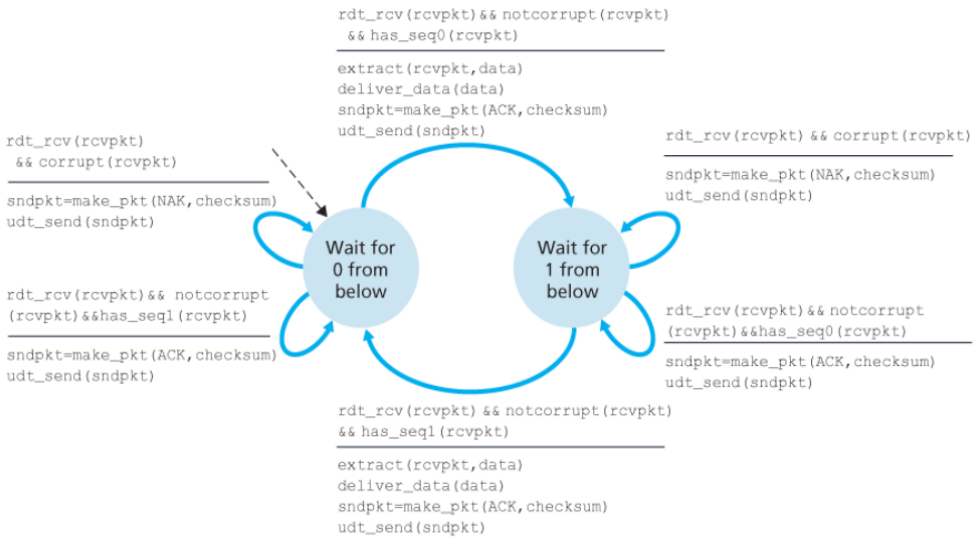
RDT2.1 FSM

INDUSTRIAL AI RESEARCH CENTER

TX



RX

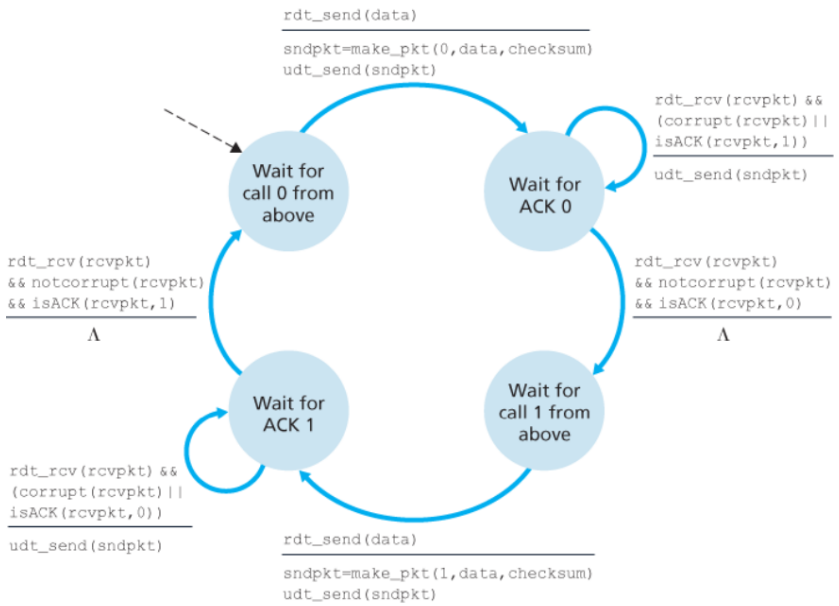


CHUNGBUK NATIONAL UNIVERSITY

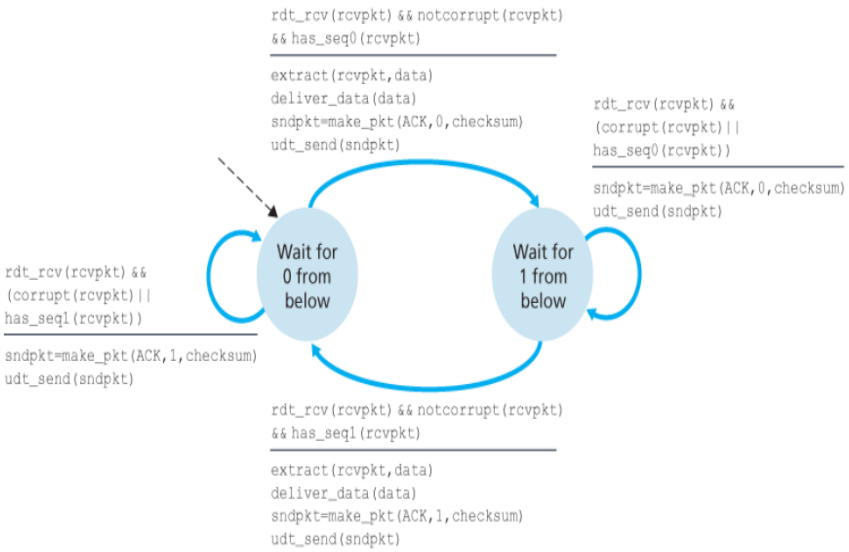
RTD 2.2: 비트 오류를 갖는 채널 w/o NAK

INDUSTRIAL AI RESEARCH CENTER

TX

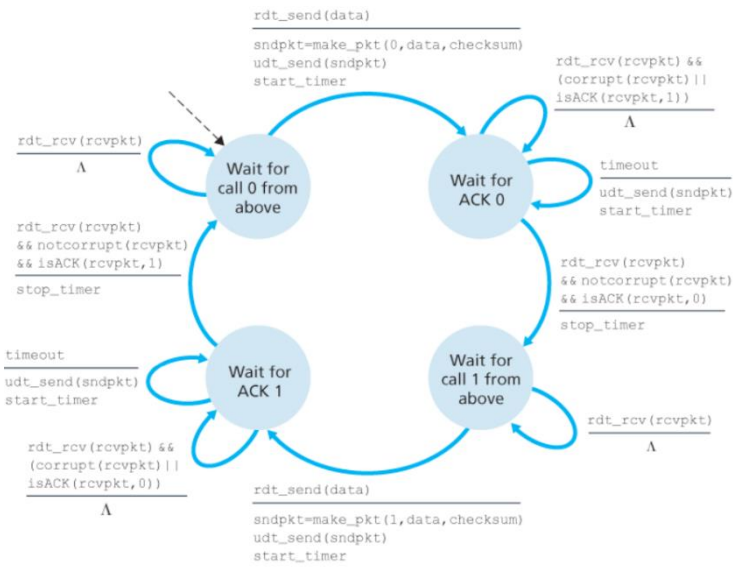


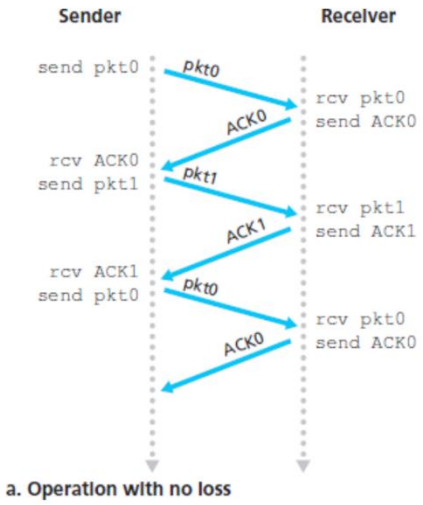
RX



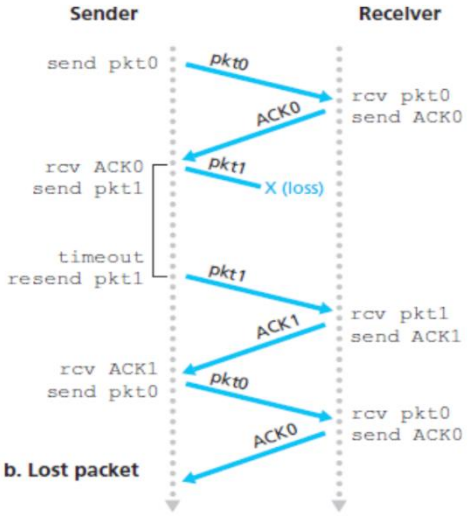
CHUNGBUK NATIONAL UNIVERSITY

RDT 3.0 비트오류와 손실 있는 채널

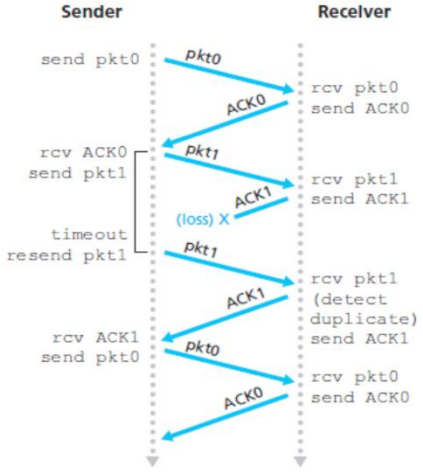




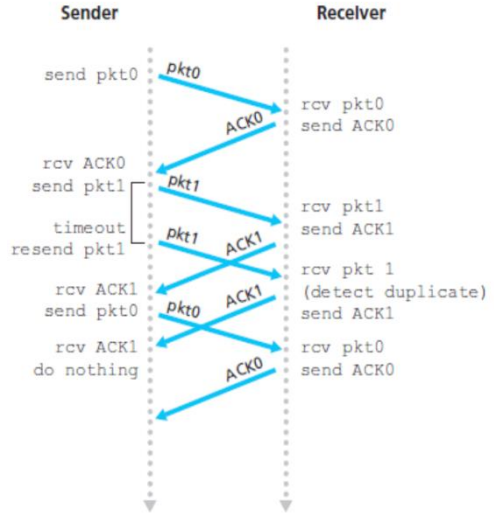
a. Operation with no loss



b. Lost packet



c. Lost ACK

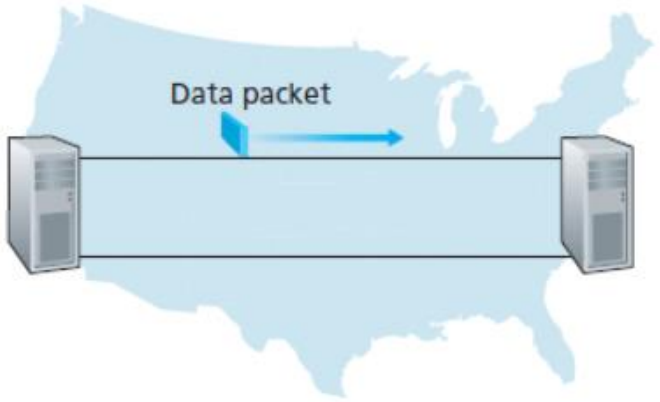


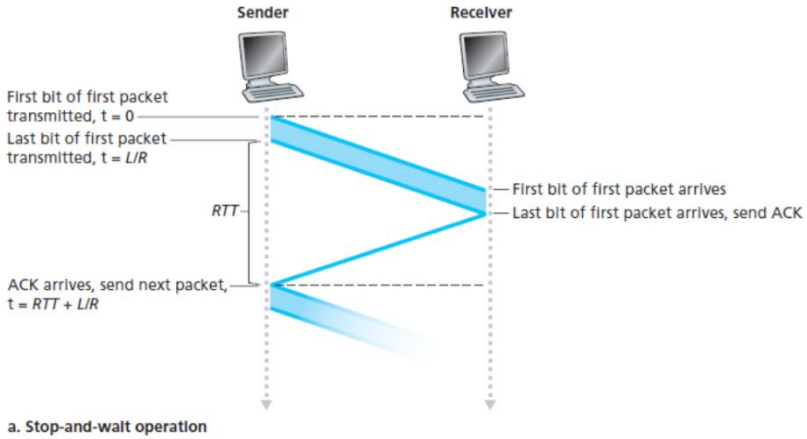
d. Premature timeout

4.2 파이프라이닝된 RDT 프로토콜

INDUSTRIAL AI RESEARCH CENTER

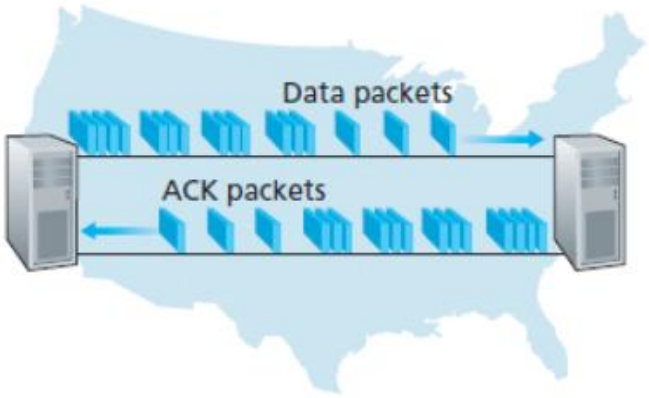
rdt3.0의 핵심적인 성능 문제점: 전송 후 대기(stop-and-wait) 프로토콜, 송신측 성능 저하(low utilization)

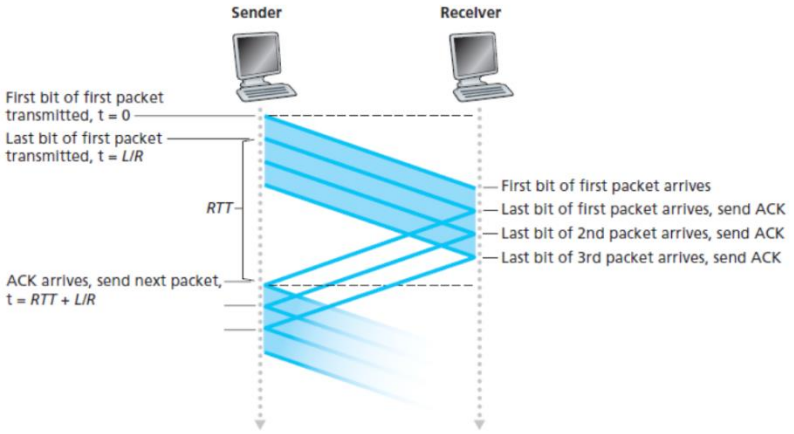




a. Stop-and-wait operation

해결책: 송신자에게 확인응답을 기다리기전에 송신을 전송 ➔ 파이프라이닝(pipelining)
즉 많은 전송 중인 송신자-수신자 패킷을 파이프라인에 채워 넣기





b. Pipelined operation

CHUNGBUK NATIONAL UNIVERSITY

파이프라이닝 방식

순서 번호의 범위 증가.
- 각각의 전송 중인 패킷은 유일한 순서 번호.
  Why? 전송 중인 확인응답(ACK)이 안 된 패킷이 여럿 있을지도 모름.

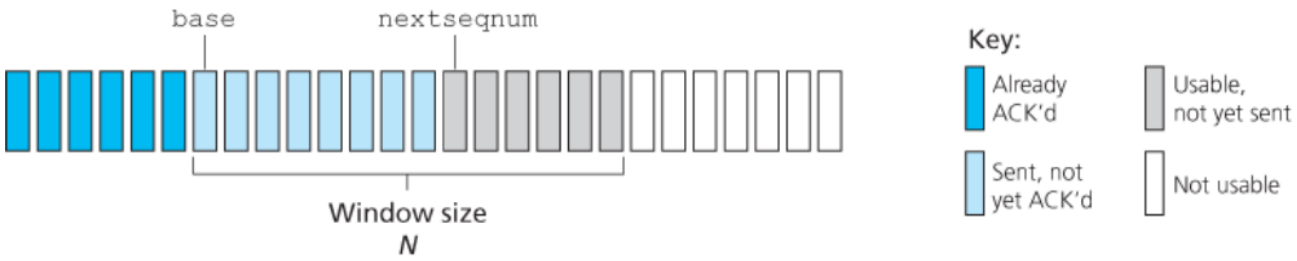
프토로콜의 송신 측과 수신 측은 패킷 하나 이상을 버퍼링.
- 최소한 '송신자는 전송되었으나 확인응답되지 않은 패킷'을 버퍼링.
- 정확하게 수신된 패킷의 버퍼링은 수신자에게서도 필요.

필요한 순서 번호의 범위와 버퍼링 조건은 데이터 전송 프로토콜이 손실 패킷과 손상 패킷 그리고 상당히 지연된 패킷들에 대해 응답하는 방식이 결정.

파이프라인 오류 회복의 2 가지 기본적인 접근 방법

- **GBN(Go-Back-N) :** N부터 반복
- **SR(Selective Repeat) :** 선택적 반복

4.3 파이프라인: GBN (Go-Back-N)



슬라이딩 윈도 프로토콜(sliding-window protocol)
전송되었지만 아직 확인응답이 안 된 패킷을 위해, 허용할 수 있는 순서 번호의 범위는 순서 번호의 범위상에서 크기가 N인 '윈도(window)'로 나타낸다.

프로토콜이 동작할 때, 이 윈도는 순서 번호 공간에서 오른쪽으로 이동(slide)된다. N = 윈도 크기(window size)

따라서 GBN 프로토콜은 슬라이딩 윈도 프로토콜(sliding-window protocol)이라고 부른다.

패킷의 순서 번호
실제로 패킷의 순서 번호는 패킷 헤더 안의 고정된 길이 필드에 포함된다.

만약 k가 패킷 순서 번호 필드의 비트 수라면, 순서 번호의 범위는 [0, 2^k - 1]
순서 번호의 제한된 범위에서, 순서 번호를 포함하는 모든 계산은 모듈로(module) 2^k 연산을 이용한다.

ACK 기반의 NAK 없는 확장된(extended) FSM
base와 nextseqnum 변수를 추가한다.

이러한 변수에서의 동작과 이러한 변수를 포함하는 조건부 동작을 추가한다.

GBN 프로토콜의 성능 문제

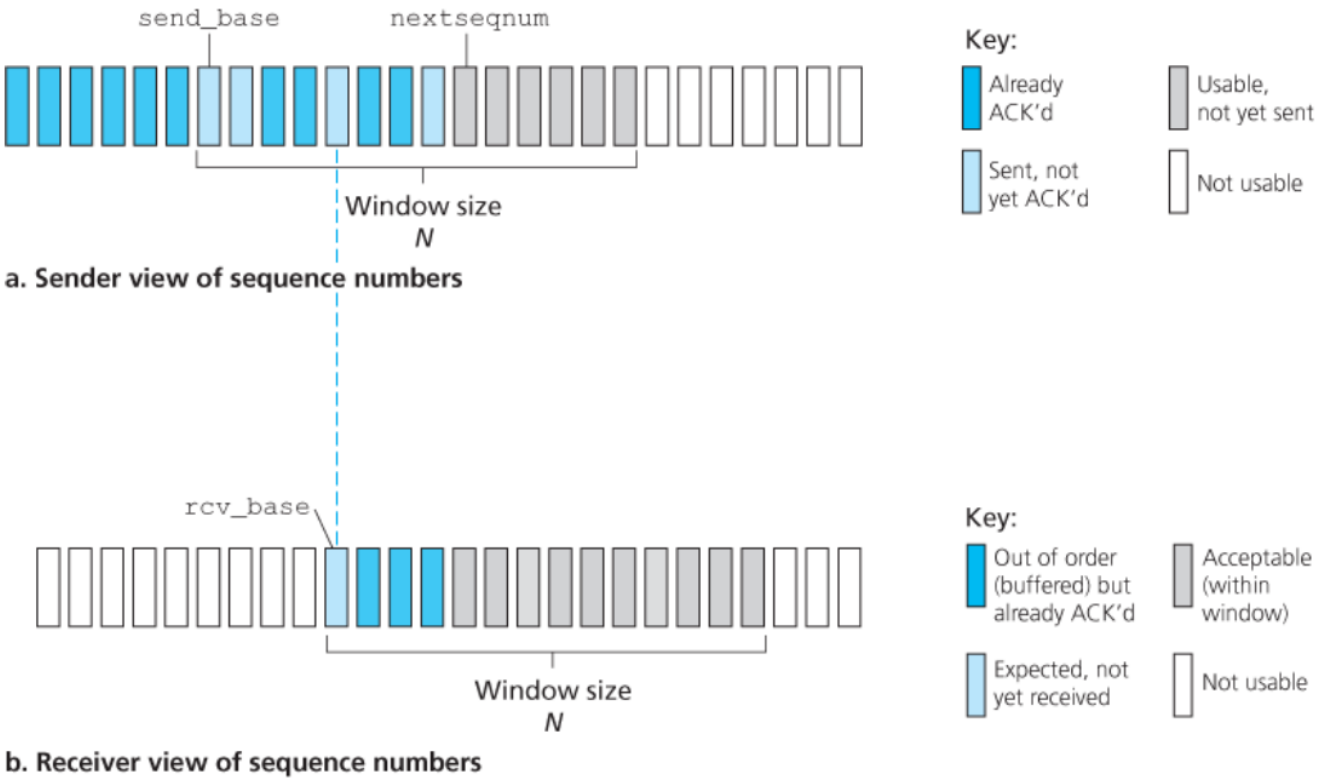
GBN 성능저하 문제점:

- 윈도 크기와 대역폭 지연(bandwidth-delay) 곱의 결과가 모두 클 때, 많은 패킷이 파이프라인에 존재.

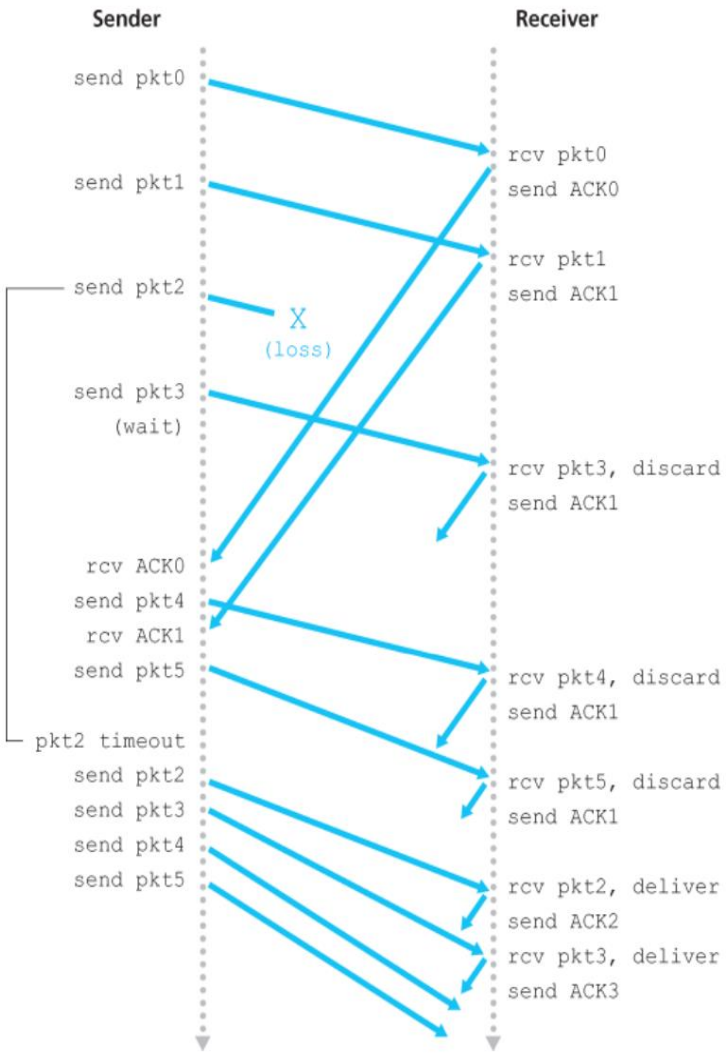
- GBN은 패킷 하나의 오류 때문에 많은 패킷을 재전송하므로, 많은 패킷을 불필요하게 재전송하는 경우가 발생.

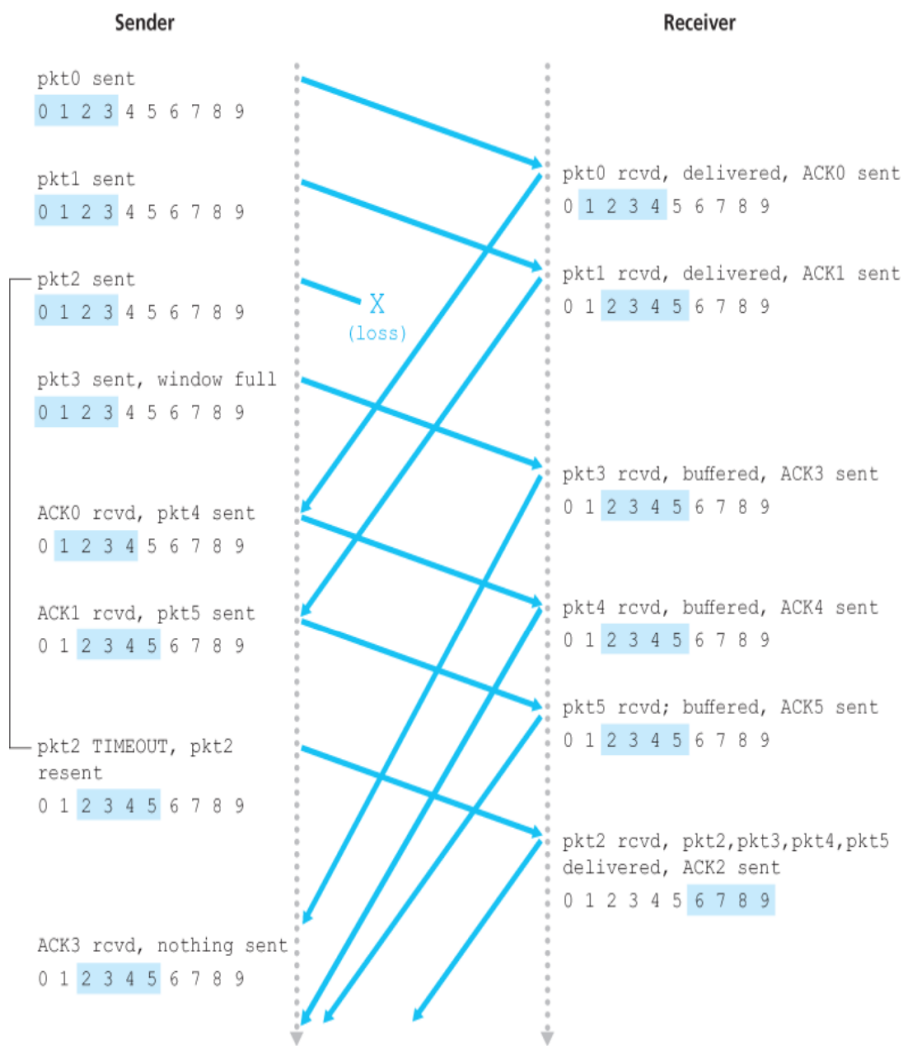
- 채널 오류의 확률이 증가할수록 파이프라인은 불필요한 재전송 데이터로 채워짐.

4.4 파이프라인: SR(Selective Repeat)



GBN동작 & SR동작


Sender
Receiver
send pkt0
send pkt1
send pkt2
X
(loss)
send pkt3
(wait)
rcv ACK0
send pkt4
rcv ACK1
send pkt5
pkt2 timeout
send pkt2
send pkt3
send pkt4
send pkt5
rcv pkt0
send ACK0
rcv pkt1
send ACK1
rcv pkt3, discard
send ACK1
rcv pkt4, discard
send ACK1
rcv pkt5, discard
send ACK1
rcv pkt2, deliver
send ACK2
rcv pkt3, deliver
send ACK3


Sender
Receiver
pkt0 sent
0 1 2 3 4 5 6 7 8 9
pkt1 sent
0 1 2 3 4 5 6 7 8 9
pkt2 sent
0 1 2 3 4 5 6 7 8 9
X
(loss)
pkt3 sent, window full
0 1 2 3 4 5 6 7 8 9
ACK0 rcvd, pkt4 sent
0 1 2 3 4 5 6 7 8 9
ACK1 rcvd, pkt5 sent
0 1 2 3 4 5 6 7 8 9
pkt2 TIMEOUT, pkt2
resent
0 1 2 3 4 5 6 7 8 9
ACK3 rcvd, nothing sent
0 1 2 3 4 5 6 7 8 9
pkt0 rcvd, delivered, ACK0 sent
0 1 2 3 4 5 6 7 8 9
pkt1 rcvd, delivered, ACK1 sent
0 1 2 3 4 5 6 7 8 9
pkt3 rcvd, buffered, ACK3 sent
0 1 2 3 4 5 6 7 8 9
pkt4 rcvd, buffered, ACK4 sent
0 1 2 3 4 5 6 7 8 9
pkt5 rcvd; buffered, ACK5 sent
0 1 2 3 4 5 6 7 8 9
pkt2 rcvd, pkt2,pkt3,pkt4,pkt5
delivered, ACK2 sent
0 1 2 3 4 5 6 7 8 9