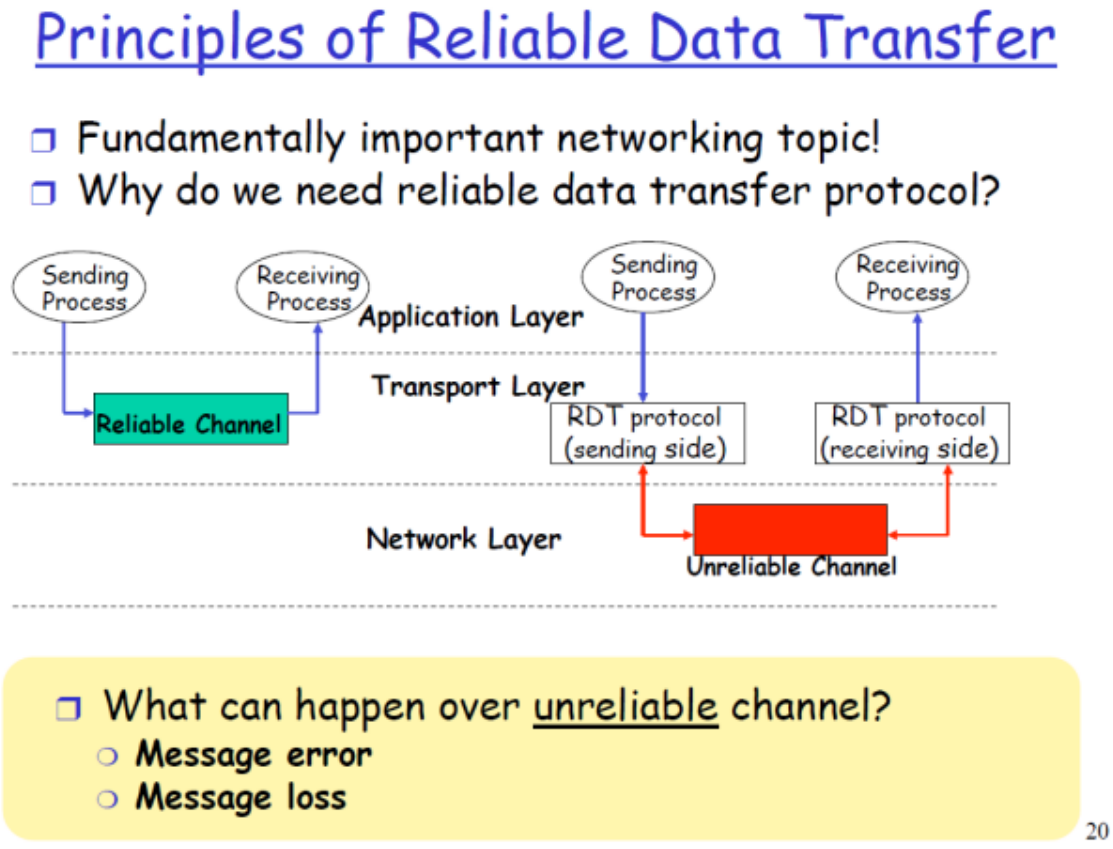
애플리케이션 계층 2

강의날짜	@2022/09/26
작성일시	@2022년 9월 26일 오후 11:38
편집일시	@2022년 9월 27일 오전 12:42
분야	네트워크
공부유형	스터디 그룹
복습	☐
태그	



- unreliable channel
  - message error
  - message loss
  - 두가지뿐임

Let's Build *simple* Reliable Data Transfer Protocol

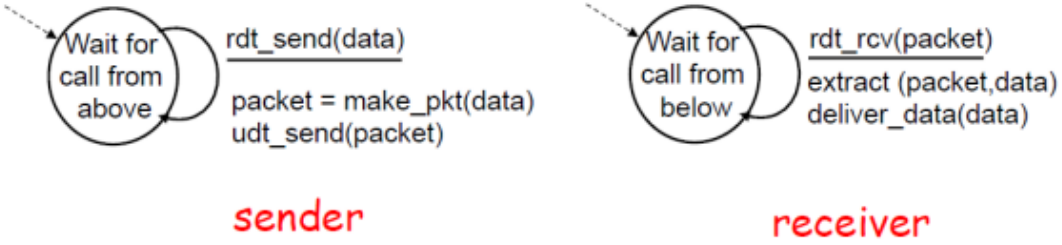
We'll:

- incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- consider only stop-and-wait protocol
- use finite state machines (FSM) to specify sender, receiver

21

Rdt1.0: Data Transfer over a Perfect Channel

- underlying channel is perfectly reliable
  - no packet errors
  - no packet loss
- What mechanisms do we need for reliable transfer?
  - Nothing! Underlying channel is reliable!



22

- underlying channel 완벽하면 에러 없음

Rdt2.0: channel with packet errors (no loss!)

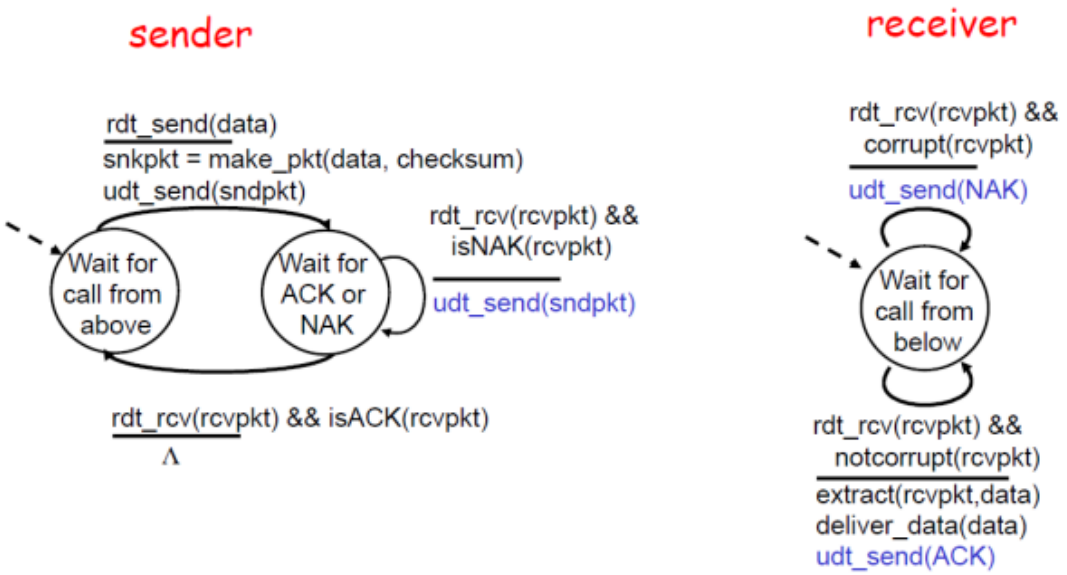
- What mechanisms do we need to deal with error?
  - Error detection
    - Add checksum bits
  - Feedback
    - *Acknowledgements (ACKs):* receiver explicitly tells sender that packet received correctly
    - *Negative acknowledgements (NAKs):* receiver explicitly tells sender that packet had errors
  - Retransmission
    - sender retransmits packet on receipt of NAK

- So, we need the following mechanisms:
  - Error detection, Feedback (ACK/NACK), Retransmission

23

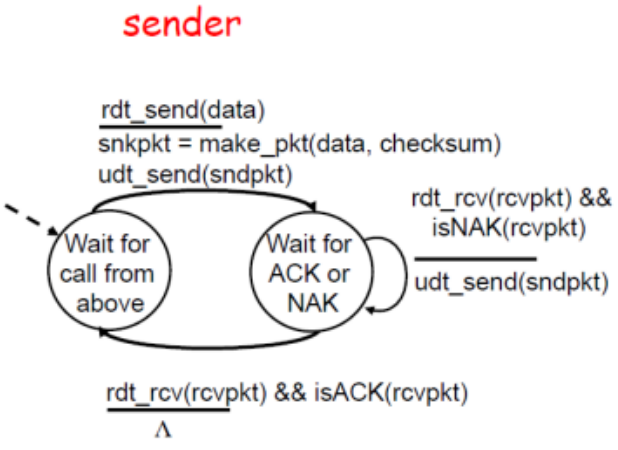
- 에러 탐지
- 지속적인 피드백
- 재전송

rdt2.0: FSM specification



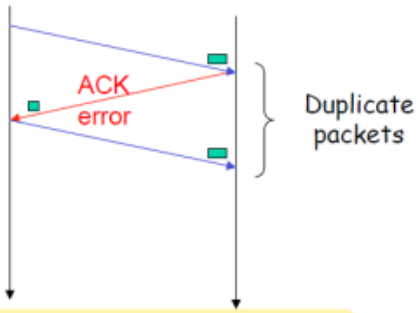
24

rdt2.0: Can this completely solve errors?



What happens when ACK or NAK has errors?

Approach: resend the current data packet?



The received packet is new or duplicate?

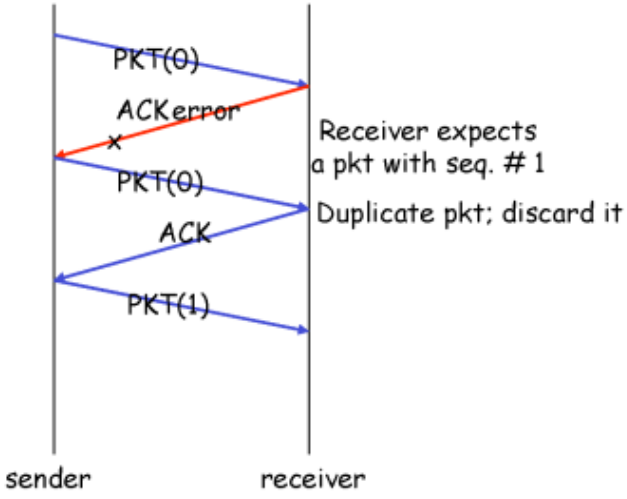
25

Handling Duplicate Packets

- Sender adds *sequence number* to each packet
- Sender retransmits current packet if ACK/ NAK garbled
- Receiver discards duplicate packet

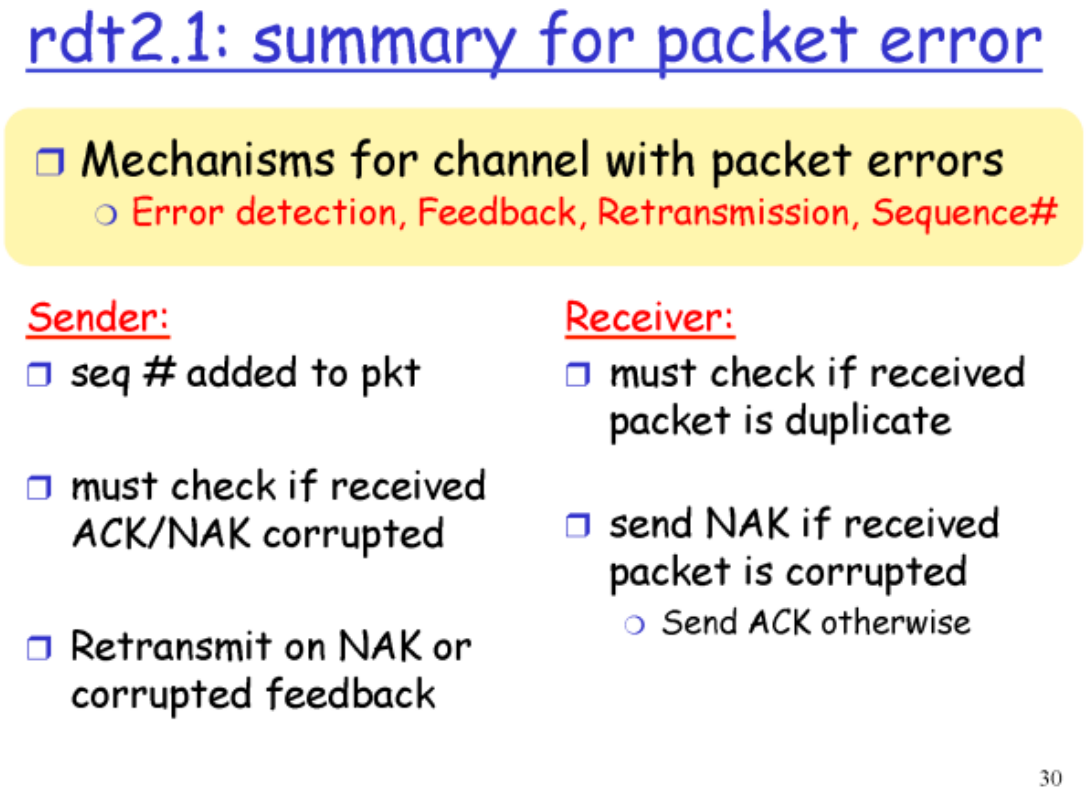
26

rtd2.1: examples



27

- Seq# 는 header에 들어감
- header는 최소한의 필드만 사용해야함



rdt2.2: a NAK-free protocol

- Same functionality as rdt2.1, using ACKs only
- Instead of NAK, receiver sends ACK for last correctly received packet
  - Receiver must *explicitly* include seq # of pkt being ACKed
- Duplicate ACK at sender results in same action as NAK: *retransmit current pkt*

31

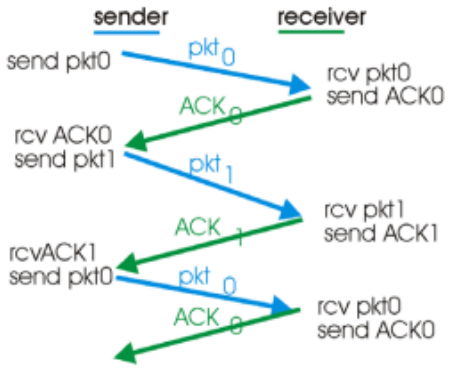
rdt3.0: channel with loss & packet errors

- What mechanisms do we need for packet loss?
  - Timer!
- Sender waits "reasonable" amount of time for ACK (a Time-Out)
- If packet (or ACK) is just delayed (not lost):
  - Retransmission will be duplicate, but use of seq. #'s already handles this

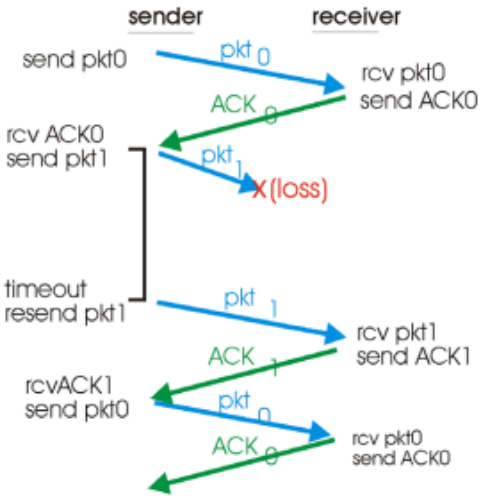
33

- 시간이 지나면 loss를 알게됨
- 센더는 패키지 보낼 때 마다 타이머를 같이 보냄

rdt3.0 in action



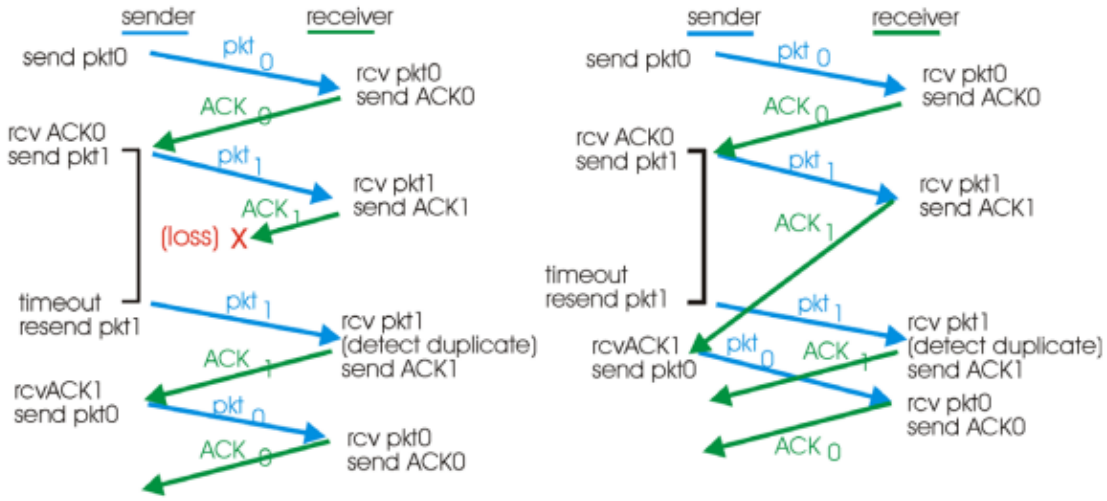
(a) operation with no loss



(b) lost packet

34

rdt3.0 in action



35

Recap: Principles of Reliable Data Transfer

- What can happen over unreliable channel?
  - Packet error, packet loss
- What mechanisms for **packet error**?
  - Error detection, feedback, retransmission, sequence#
- What mechanisms for **packet loss**?
  - Timeout!
- We built simple reliable data transfer protocol
  - Real-world protocol (e.g., TCP) is more complex, but *with same principles!*

36

Performance of rdt3.0

- rdt3.0 works, but performance stinks
- example: 1 Gbps link, 15 ms e-e prop. delay, 1KB packet:

$$T_{transmit} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8kb/pkt}{10^{**}9 \text{ b/sec}} = 8 \text{ microsec}$$

  - U_{sender}: utilization – fraction of time sender busy sending

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

  - 1KB pkt every 30 msec -> 33kB/sec thruput over 1 Gbps link
  - network protocol limits use of physical resources!

Pipelined protocols

Pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged pkts

  - range of sequence numbers must be increased
  - buffering at sender and/or receiver

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

- Two generic forms of pipelined protocols: *go-Back-N, selective repeat*

loss - timer

메카니즘은 구현되어잇음

packet의 tcp 헤더부분에 정보 필드

하지만 현재 배운 것은 RDT protocol은 너무 단순해서 한번에 하나의 패킷밖에 보내지 못함

물론 신뢰성 있는 통신이긴함