-컴퓨터 네트워크-

Protocol Functions

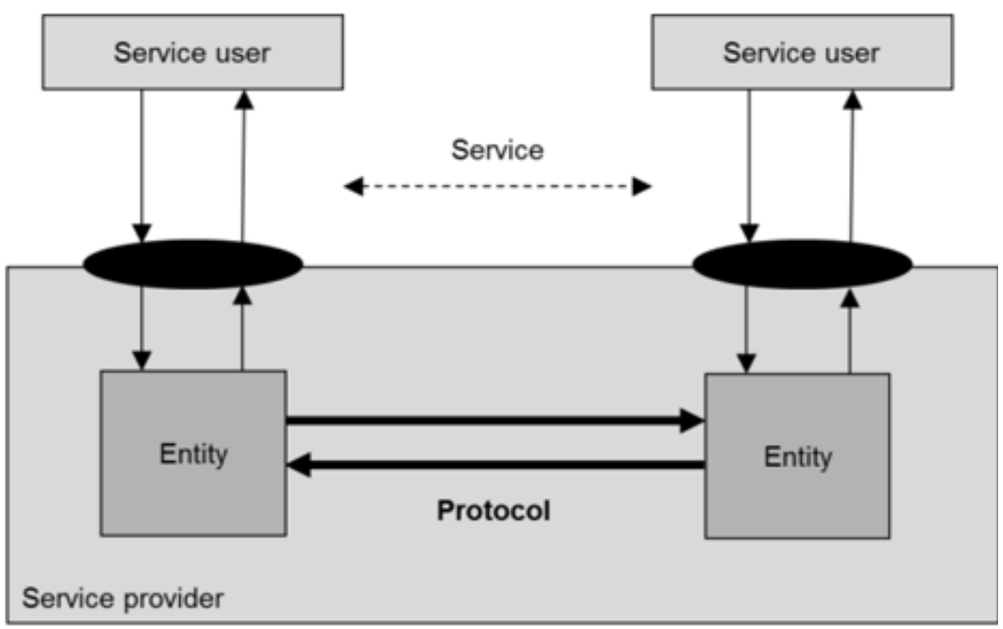
2022 Spring
Kyungseop Shin

Course Outline

- 다양한 Protocol function들에 대해 이해
  - Error control
  - ARQ

Protocol?

- 지난번에는 model 관점에서 protocol 개념에 대해 접근
    - 실루엣은 보이나, 실제 생김새는 아직 실감이 안감
    - 실제 생김새 = protocol function



Protocol Functions

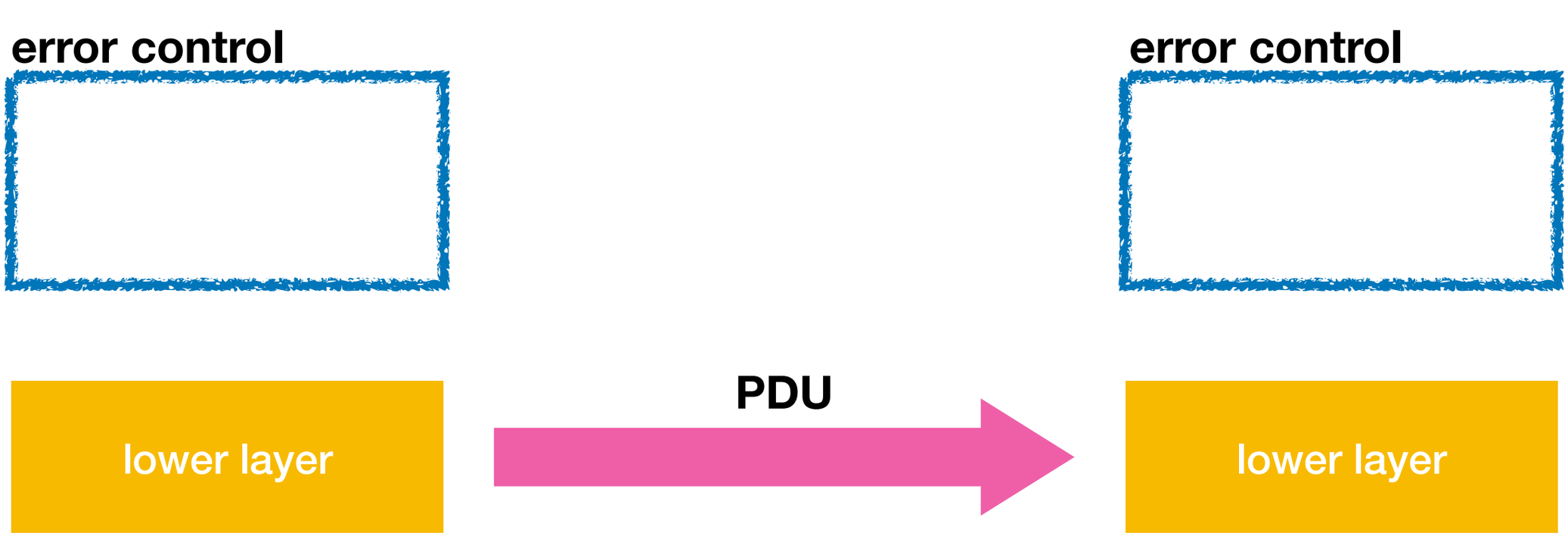
- 여러 protocol에서 두루 사용되는 특정 procedure / mechanism
  - 여기저기서 공통적으로 등장하는 주연/조연 같은 존재

Protocol Functions

- 예시 :
  - error control
    - PDU가 정상적으로 전달되지 않은 상황에서의 protocol entity의 동작
  - fragmentation, flow control
    - entity간 data를 서로 주고받는 속도 및 형태 조절

Error Control

- Connection-oriented communications protocol에게 주어지는 중요 역할 중 하나는
  - 상대방에게 확실하게 PDU 전달 (correctly/reliably)



Error Control

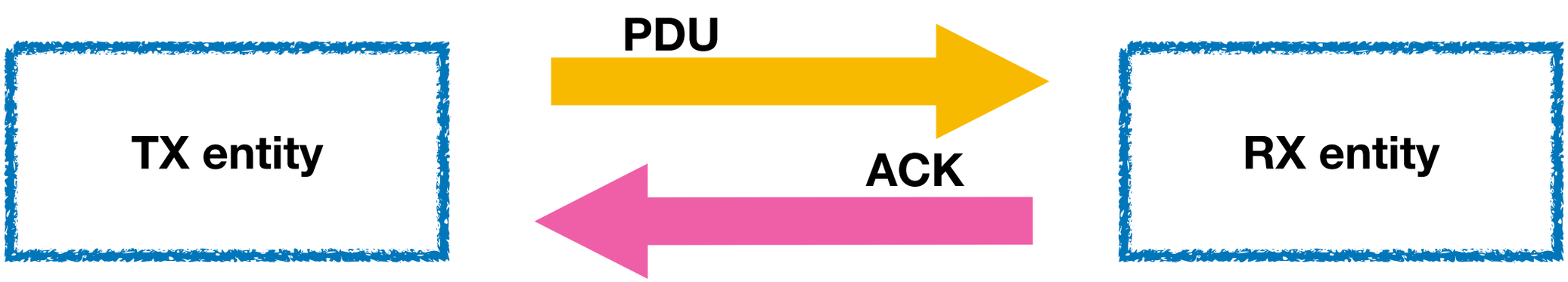
- Protocol에서 “확실한 전달”을 하려면 두가지 동작에 대한 정의가 필요함
    - Detect : 송신 측에서 error한 상황을 인지
    - react : error 상황에서 송신 측 동작

Error Control : Confirmation

- 한쪽 entity에서 PDU를 보내면 receiver entity에 그 PDU가 도달했는지 여부를 확인할 수 있어야 함

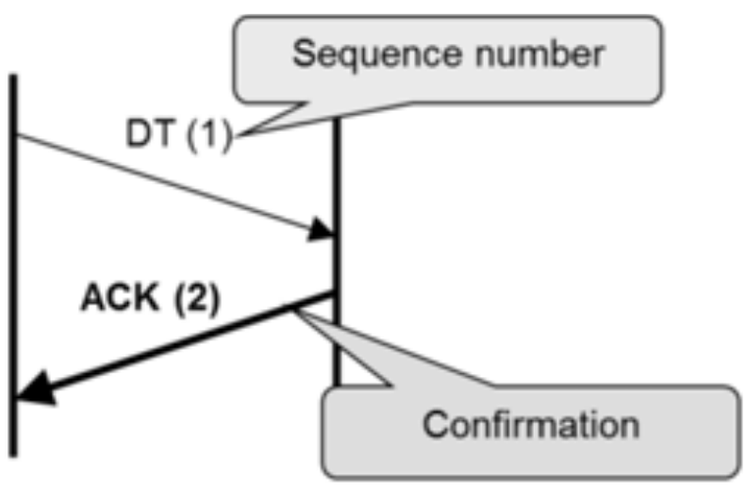
- Explicit acknowledgement : receiver entity가 PDU를 온전히 받았는지 여부를 다시 보내줌

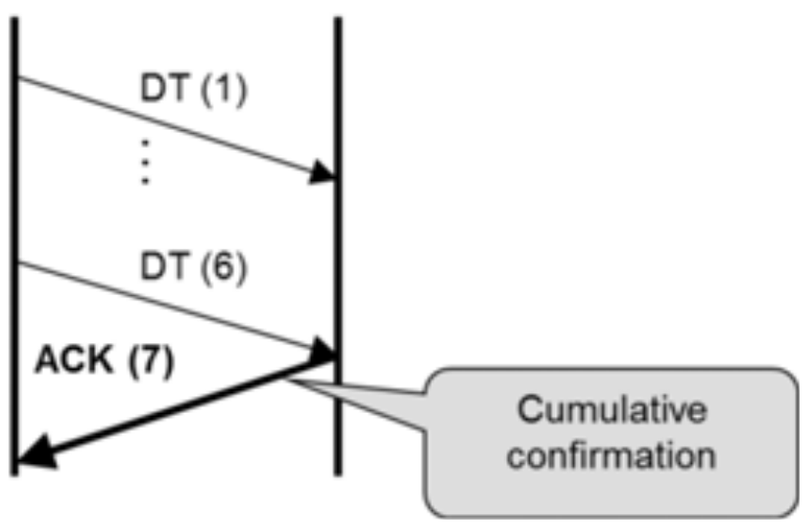
- 가장 단순하고도 확실하게 confirmation을 하는 방법



Error Control : Confirmation

- Explicit ACK는 기본적으로 두가지 세부 방법 존재
  - Positive ACK : 정상 수신된 PDU를 알려주는 방식
    - 지금까지 수신한 PDU의 다음 sequence number를 알려줌
    - network load를 줄이기 위해 cumulative ACK가 주로 사용됨



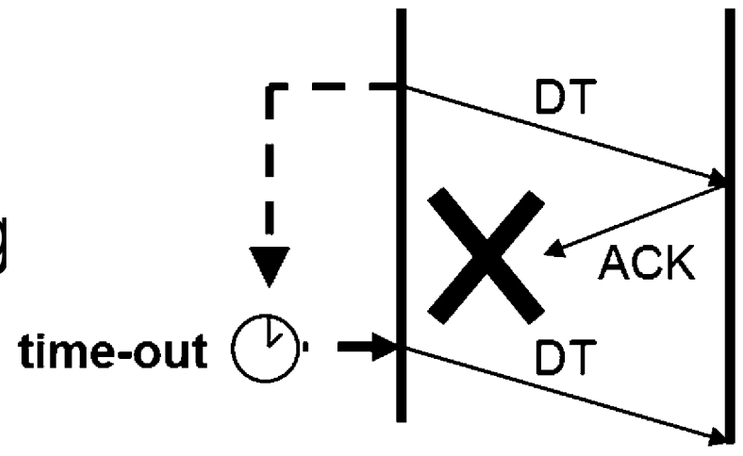


Error Control : Confirmation

- Negative ACK : 미수신 PDU를 알려주는 방식
    - faulty or outstanding sequence
    - active error control
- reaction은 곧 재전송이나, 세부 방법은 다양함
    - 대개 못받은 PDU를 재전송하거나 연결을 끊음
- Piggybacking : explicit한 ACK 메시지 전송이 아닌, PDU에 실어서 같이 ACK 정보 전송

Timer

- Acknowledgement 전송 역시 손실될 수 있음
  - TX entity 입장에서는 PDU를 보낸 뒤 acknowledgement를 무기한 기다릴 수는 없음
    - **Deadlock** situation
- 따라서 ACK에 대한 **기한**을 두고 기한을 넘기면 action을 취할 필요가 있음
  - timer에 의해 ACK 도달을 monitoring



Timer

- 보통 PDU를 보낼 때 timer 시작
- Timer 가동 시
  - timer interval은 ACK가 도달할 예상 시간을 감안하여 정의
  - ACK이 도달하지 않으면 timer expiry가 되고
    - time-out이 되면 기정의된 plan B를 수행히여 deadlock에 빠지지 않도록 함
  - ACK이 도달하면 timer를 reset함

Activity Timer

- 보통 protocol에서 timer는 peer entity에 대한 monitoring 용도로 활용
    - peer entity가 무슨 이유에 의해서 communication을 멈추었을 때 deadlock이 발생할 수 있으므로, 이를 방지하기 위해 timer 가동
        - peer partner의 inactivity 감지
- 항상 peer partner의 reaction 감지 시 멈춤
    - PDU 수신

- 보통 reaction 시간 보다 충분히 크게 timer interval을 잡음

Activity Timer

- 예시

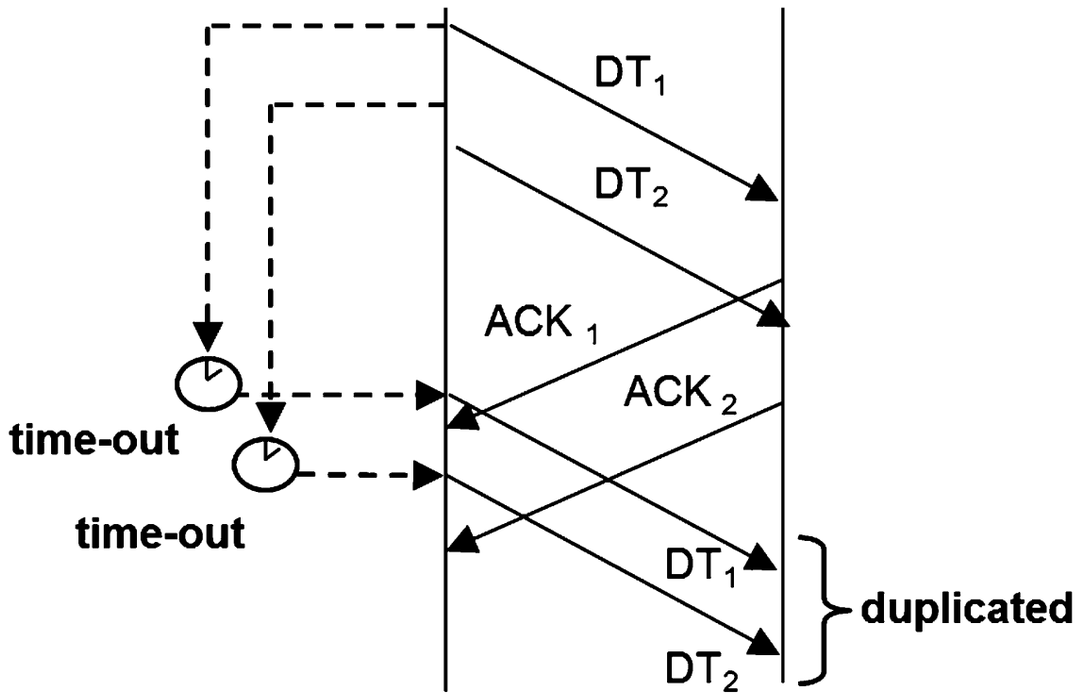
```
start t1                                          // Start activity timer
loop{
    start t2                                      // Start ACK monitoring
    wait event{                                   // Await ACK
        ACK ← receiver: reset t1                  // Reset activity timer
                reset t2                          // Reset ACK supervision
                . . .                             // ACK decoding
                start t1  |                       // Restart activity timer
            . . .
        timeout t1: respond XABORTind(conn)       // Receiver entity inactive
                set CONNECT                       // Sender closes connection
                sequ:=1; last:=0                  // Reset variables
                clear_queue(conn)
                exit ack_handler                  // Leaving protocol part
    }
}
```

Timer의 중요성

- Protocol specification(정의)에 있어서 핵심 내용 중 하나가 timer를 정의하는 것임
  - timer name / interval / start / stop / expiry
- Time interval은 적절한 길이로 설정되어야 함
  - 너무 짧으면 time-out이 빈번하게 일어나서 불필요한 reaction 발생
  - 너무 길면 reaction이 너무 늦어지면서 불필요한 deadlock 구간 발생

PDU Loss and Duplication

- Data transmission동안 PDU가 손실되거나 중복 수신이 될 수 있음
    - entity는 PDU sequence number에 의해 감지
- PDU Loss의 경우
    - PDU 가 time-out interval 이후에도 도달하지 않는 경우
    - 더 큰 sequence number PDU수신 시
    - TX는 다음 2가지를 통해 파악
        - ACK를 통해 빠진 sequence 인식
        - 중복된 ACK sequence 수신



PDU Loss and Duplication

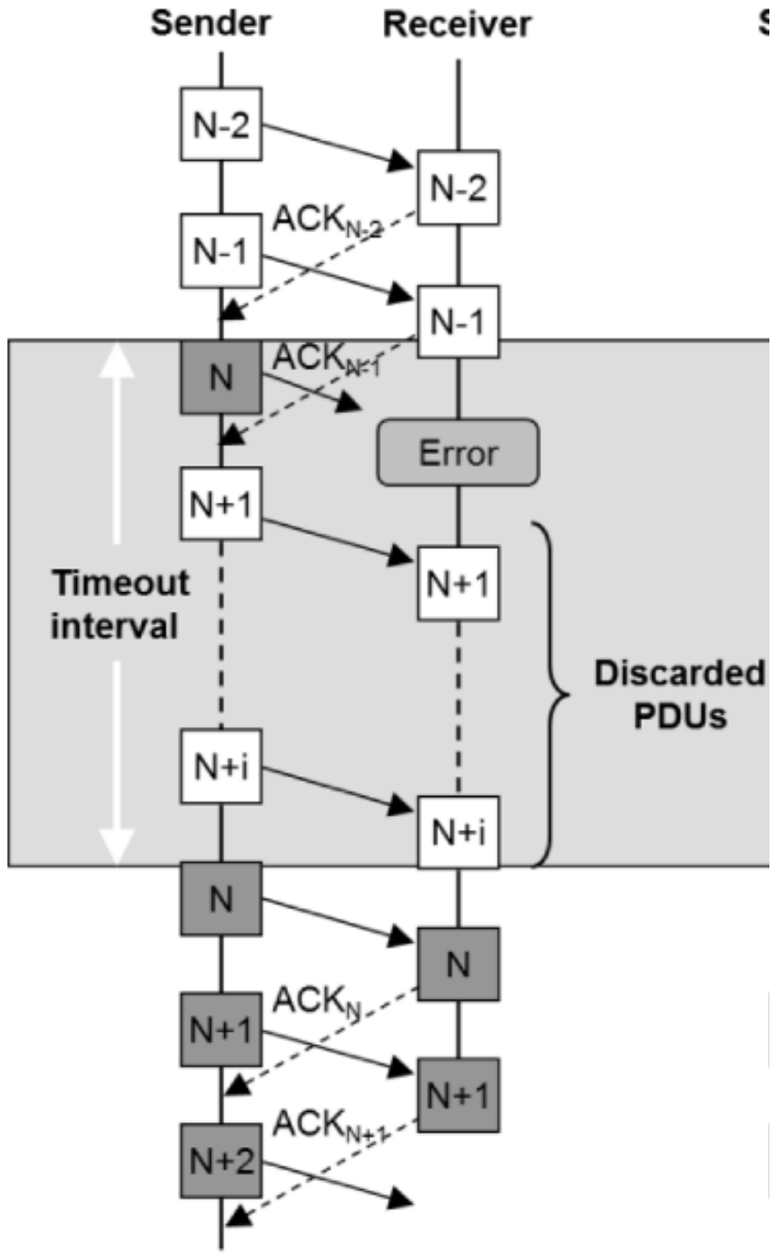
- PDU duplication의 경우
  - ACK가 손실되고 TX 측에서 time-out에 의해 PDU가 재전송 된 경우
  - 수신 측에서는 sequence number를 확인하고 discard(무시) 할 수 있음

Automatic Repeat reQuest (ARQ)

- ARQ : Acknowledgement + timer + **reactions**
  - 신뢰성 있는 PDU 전송을 위한 가장 보편적인 protocol function
  - TX/RX가 confirmation (ACK)를 받거나 time-out 까지 기다리는 동안 하는 일련의 동작 포함
- 재전송 방법에 따라 여러가지 방식 존재
  - stop and wait
  - go back N
  - selective repeat

Automatic Repeat reQuest (ARQ)

- Go back N : N으로 되돌아가기
  - N번째 unconfirmed PDU로 돌아가서 거기서부터 재전송함
  - TX entity는 acknowledged되기 전까지 보낸 PDU를 그대로 저장하고 있어야 함
  - RX entity는 N번째 PDU가 문제가 생기면 이후 PDU는 모두 버림
  - time-out의 경우 N번째 PDU부터 재전송

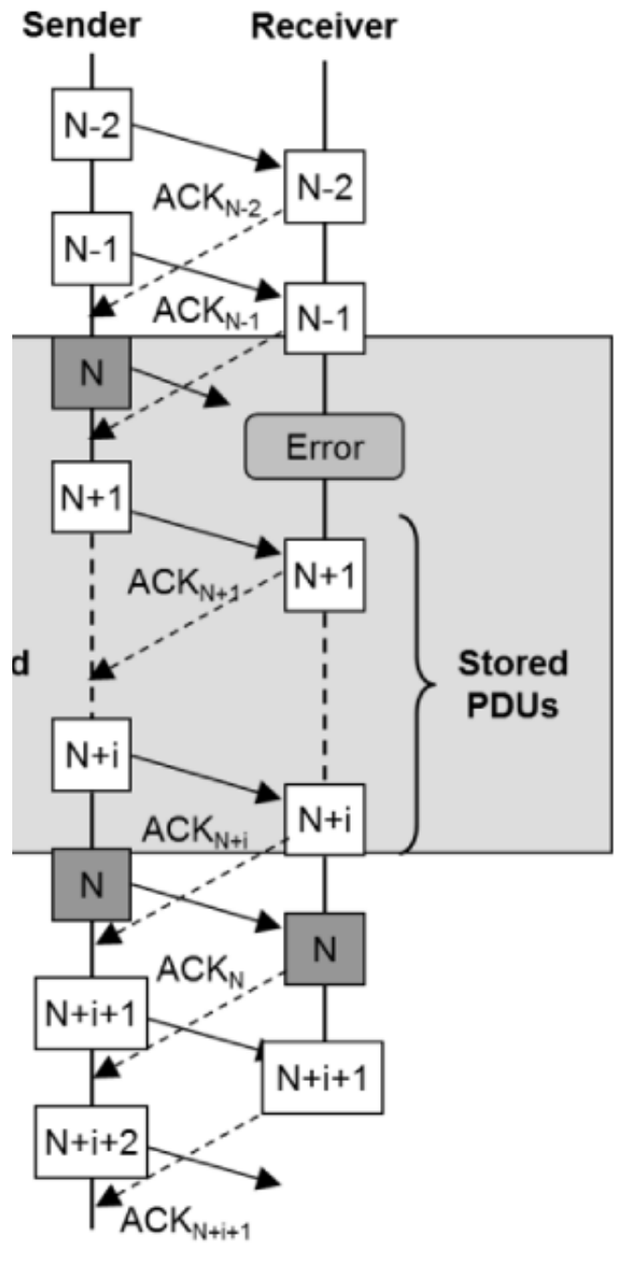


a) *Go back N*

b

Automatic Repeat reQuest (ARQ)

- Selective repeat
  - 손실된 PDU에 대해서만 골라서 재전송
  - RX entity는 제대로 받은 PDU을 저장
    - transmission order대로 다 받을때 까지 저장
- cumulative ACK를 통해 제대로 전달된 PDU를 TX entity에서 확인



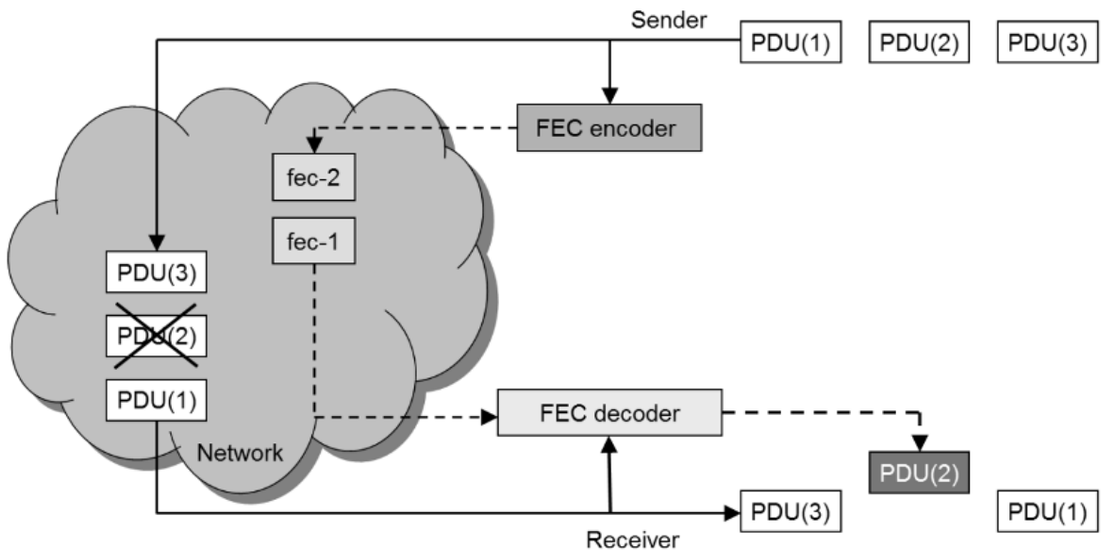
b) Selective repeat

Automatic Repeat reQuest (ARQ)

- Go back N vs. selective repeat
    - Selective repeat는 RX entity 측에 buffer 필요
    - Go back N는 TX entity 측에 buffer 필요
    - Go back N은 재전송 수신까지 시간이 걸리나
    - selective repeat은 재전송 수신이 상대적으로 시간이 덜 걸림

Forward Error Control (FEC)

- Coding theory : 자가치유 능력
  - 일부 bit 오류에 대해 정정이 가능하도록 data bit를 뻥튀기 해주는 이론
- FEC
  - coding theory 기반으로 PDU의 일부 오류를 정정
    - redundant data 추가



Forward Error Control (FEC)

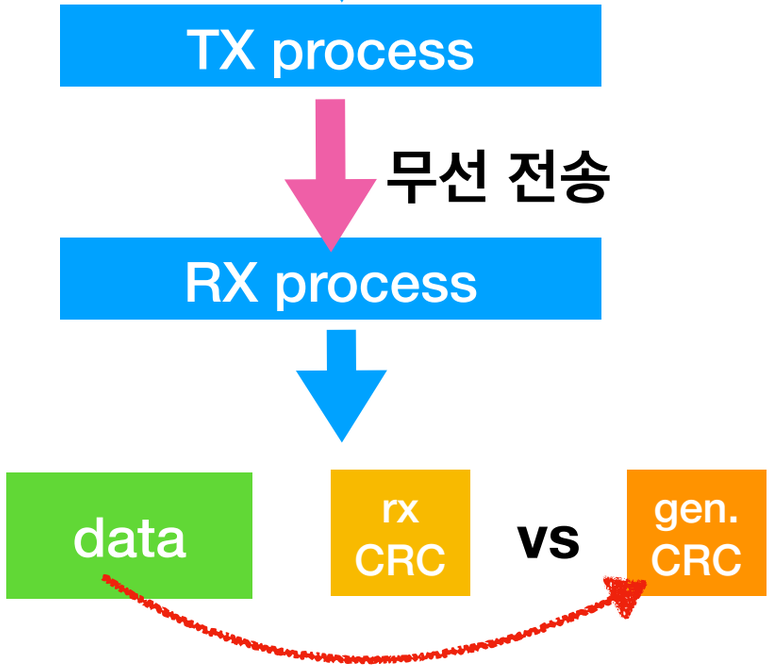
- 보통 계산량이 많음
- real-time traffic에 유리 (voice, video)
    - 시간이 지나면 수신 PDU가 의미가 없어지는 서비스
        - 재전송을 할 여유가 없으므로 받는 즉시 정정하는 것이 바람직함
- 오늘날 모든 무선통신 시스템에서 활용
    - 계산 능력의 향상
    - 무선 자원에 대한 효율적 사용

Cyclic Redundant Check (CRC)

- FEC가 모든 bit 오류를 다 고쳐주진 못함
    - bit 오류가 많거나, 연속적인 bit 오류는 완벽하게 정정하지 못함
- CRC의 활용 : Error detection
    - 아무리 고치려 시도해도 오류가 여전히 있으면 버려야 함
    - “오류가 여전히 있음”을 감지하기 위해서 CRC 활용
- CRC 원리
    - 부가 정보를 붙이고, 수신 측에서 부가 정보를 활용해서 깨졌는지 여부 확인

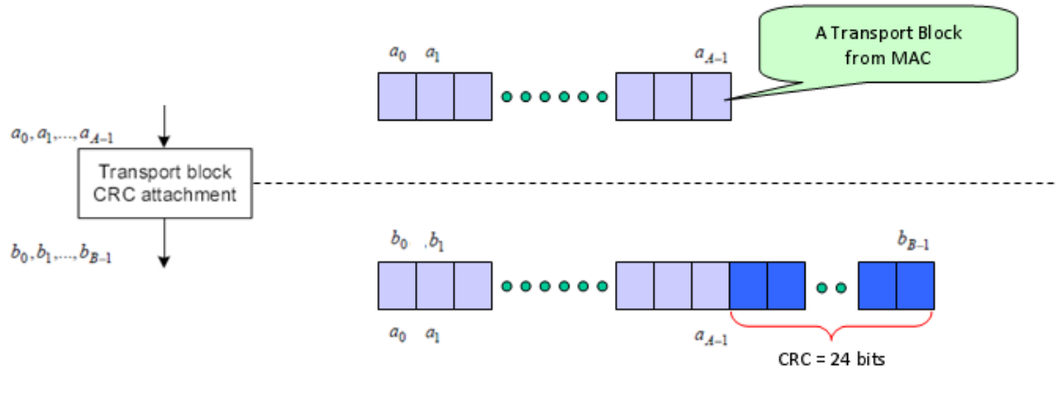
Cyclic Redundant Check (CRC)

- 원 데이터에 parity bit들을 꼬리에 추가
  - LTE의 경우 16/24bit 추가
  - 원 데이터 bit들을 입력으로 해서 parity bit 만듬
- 주로 FEC와 조합하여 쓰임
  - 수신 CRC vs. 다시 만들어본 CRC 가 같은지 확인
    - 같으면 error bit 가 없음
    - 같지 않으면 error bit 존재
  - channel coding이 원데이터를 완벽하게 복원했는지 확인하는 용도로 활용



Cyclic Redundant Check (CRC)

- CRC generation
  - input bit : $a_0, a_1, a_2, a_3, ..., a_{A-1}$
  - parity bit (L bits) : $p_0, p_1, p_2, p_3, ..., p_{L-1}$
  - cyclic generating polynomial



❖ B = A + 24

$g_{CRC24A}(D) = [D^{24} + D^{23} + D^{18} + D^{17} + D^{14} + D^{11} + D^{10} + D^7 + D^6 + D^5 + D^4 + D^3 + D + 1]$

$g_{CRC24B}(D) = [D^{24} + D^{23} + D^6 + D^5 + D + 1]$ for a CRC length $L = 24$ and;

$g_{CRC16}(D) = [D^{16} + D^{12} + D^5 + 1]$ for a CRC length $L = 16$.

$g_{CRC8}(D) = [D^8 + D^7 + D^4 + D^3 + D + 1]$ for a CRC length of $L = 8$.