Framing / Error Control

We are here

- OSI 7 layer model

Framing

프레임 단위로 에러를 확인하고
오류가 나오면 프레임을 버림

Framing

- Bits are transmitted continuously → bit stream

- 0101101100011001001001001001111101001011

- The receiver should understand the received data

- The first thing is to do is to divide the bit stream into meaningful block of data → "<u>frame</u>"

Reading by Punctuating

“Elizabeth, New Jersey, when my mother was being raised there in a flat over her father’s grocery store, was an industrial port a quarter the size of Newark, dominated by the Irish working class and their politicians and the tightly knit parish life that revolved around the town’s many churches, and though I never heard her complain of having been pointedly ill-treated in Elizabeth as a girl, it was not until she married and moved to Newark’s new Jewish neighborhood that she discovered the confidence that led her to become first a PTA “grade mother,” then a PTA vice president in charge of establishing a Kindergarten Mothers’ Club, and finally the PTA president, who, after attending a conference in Trenton on infantile paralysis, proposed an annual March of Dimes dance on January 30 – President Roosevelt’s birthday – that was accepted by most schools.”

- Philip Roth, “A plot against America”

Framing

- Fixed-size framing

- Variable-size framing

Fixed-size Framing

- Fixed number of bits

모든 프레임에 비트가 정해짐
수신자는 비트를 세서 처리하면 됨

나누는 패턴

- No delimiter necessary

- May cause inefficiency
  - Suppose the frame size is 1000 bits
  - If I have only 50 bits to send, the frame still has to be 1000 bits

프레임이 요구하는 비트 수 만큼 맞춰서 보내야한다
보내는 데이터가 프레임 보다 작으면 나머지 자리를 패딩으로 채움

Variable-size framing

- Each frame can have different number of bits
- Delimiter necessary to identify frames
  프레임의 시작과 끝을 알려줘야함
- Can build frames according to the data sent → efficient
  - If I have 50 bits to send → I can make the frame 50 bits

Variable-size Framing

- Flag: marker that points the beginning and the end of a frame
- Header: bits inserted before the data for control purposes
- Trailer: bits inserted after the data for control purposes
- Data/Payload: data to be sent

프레임의 구성
시스템마다 다르지만 예시

Frame Structure

- Defines size and meanings of fields in a frame

Frame Structure

- Start/end marker
  - e.g. 01111110

- What if "01111110" appears in the payload?
  - Suppose payload is "0010101111110100"
  - Assume there is no header/trailer
  - Then the frame is "<u>01111110</u>0010101111110100<u>01111110</u>"
  - What is the problem?

Bit Stuffing

데이터가 1이 연속으로 6번 이상 오게 하지 않겠다

- Problem
  - If the marker pattern appears in the payload, the receiver will think it as an end marker
  - leads to wrong framing

- Solution 프레임 flag인 0111110이 데이터에 나와서 flag랑 햇갈리지 않도록 데이터부분에 01111110이 안나오게함
  - Ensure the marker pattern does not appear in the payload
  - If “11111” is found, add a “0” after the pattern
    - Bit stuffing
  - The receiver removes “0” after “11111”, since it is the <u>stuffed bit</u>

지금 layer 2이기 때문에 1이 반복되는 clock drift, baseline wondering은 layer 1에서 해결하는 문제라 상관 없음

Bit Stuffing

Byte Stuffing

- The same purpose as bit stuffing
- Insert a byte after a marker byte pattern
- Used in systems that process data in unit of bytes

Byte Stuffing

Byte Stuffing

- TinyOS is an operating systems used for small embedded systems such as sensors

- TinyOS uses 7E as start/end marker
- If 7E appears in the payload, it is changed to 7D 5E
- If 7D appears in the payload, it is changed to 7D 5D

원래 데이터를 이렇게 바꾼다

- 7E → 7D 5E
- 7D → 7D 5D

원래 데이터 : 7E 7D
인코딩 : 7D 5E 7D 5D

45 28 36 7D 5E
7E 45 28 36 7D 5D 5E 7E
7E로 한 프레임을 구성

Error Control

Error Control

에러를 검출한 다음 뭘 할 것인가

- Error detected → What should the receiver do?

- For reliable communication,
  - The receiver should tell the sender to **repeat** frame transmission

재전송 해달라고 요청

- Automatic Repeat reQuest (ARQ)

ARQ

- ARQ Types
  - Stop-and-Wait
  - Go-back-N
  - Selective Request

Stop-and-Wait

ACK

수신자가 프레임을 받았다는 걸 알려주는 것

- Acknowledgement (ACK) frame
  - Receiver sends an ACK frame to the sender if a frame is successfully received
  - When sender receives the ACK frame, it knows that the sent frame is successfully received at the receiver

Normal case

Stop-and-Wait
Sender
Receiver
Frame
ACK
Frame
ACK
Frame
ACK
Frame
ACK
Time

Error case

- What happens if a frame has an error?
  - The receiver does not accept the frame
  - ACK is not sent

Error case

- It is also possible that the frame is correctly received, but the ACK is lost

- To the sender, the <u>two cases look the same</u>

sender 입장에서는 ack를 못받는 이 두 경우 중 어떤 것에 해당하는지 알 수 없음

Sender's action

- After sending a frame, wait for <u>T seconds</u>
  - T is called the "timeout period"
- If ACK comes in this duration, send the next frame
- If ACK does not arrive in this duration ("timeout"), the sender thinks the frame is lost
  → retransmit the frame

Retransmission cases

- Case 1: frame is lost

rtt보다 timeout을 길게 해야한다

Retransmission cases

- Case 2: frame not lost, but ACK is lost

Retransmission cases

- Case 3: frame not lost, ACK not lost, but ACK arrived later than the timeout
  - Problem: timeout period is too short!

timeout을 잘못 잡아서 다시 전송해버림

ack가 지금 프레임에 대한 반응인지, 그 전 프레임에 대한 반응인지 헷갈림

1번 프레임에 대한 ack가
2번 프레임의 timeout 시간에 와버림
-> sender는 2번 프레임에 대한 ack라고 생각

Timeout

- What is the proper timeout period?
- Timeout should be based on RTT (Round-Trip Time)
- RTT: Duration of time between the sender starts sending a frame and finishes receiving an ACK

  Transmission time of frame +
  Propagation time of frame +
  Transmission time of ACK +
  Propagation time of ACK +
  Processing time at sender and receiver

- Typical timeout period: <u>T = RTT + margin</u>
  - margin is to account for unpredicted delays (e.g. processing time)

Receiver's perspective

- A frame is arrived 프레임이 오면 붙여서 위쪽 레벨로 넘겨주는 게 layer 2
- Later, another frame is arrived
  - Is this the next frame? 받은 프레임이 재전송된 건지, 그 다음 프레임이 온건지 구분이 안됨
  - Or is this the same frame that has been retransmitted?

Cases

- Normal case

- S: sends frame
- R: receives frame, sends ACK
- S: receives ACK sends next frame 이 때의 프레임은 다음 프레임
- R: receives frame

- In this case, the frame is a new frame

Cases

- Retransmission case 1: frame is lost

- S: sends frame
- R: (does not receive frame)
- S: timeout called, retransmit frame
- R: receives frame, sends ACK
- S: receives ACK, sends next frame 이 때의 프레임은 재전송된 프레임

- In this case, the frame is a new frame

Cases

- Retransmission case 2: ACK is lost

- S: sends frame
- R: receives frame, sends ACK
- S: does not receive ACK, timeout, retransmit frame
- R: receives <u>frame</u>, sends ACK

- In this case, the <u>frame</u> is a retransmitted frame

Cases

- Retransmission case 3: ACK is arrived after timeout

- S: sends frame
- R: receives frame, sends ACK
- S: timeout called, retransmit frame
- S: receives ACK, sends next frame
- R: receives <u>frame</u>, sends ACK
- R: receives frame, sends ACK

- In this case, the <u>frame</u> is a retransmitted frame

Receiver's perspective: Summary

- A frame arrives, so sends an ACK
- Later, another frame arrives

- <u>The receiver does not know if it is a new frame or a retransmitted frame!</u>

Solution: Sequence Number

sequence number - 몇번째 프레임인지 알려주는 번호

- Include sequence number in the frame header
- Sequence number is incremented in each frame
- The range of sequence number: [0…N]
  - The next of frame(N) is frame(0)

n개의 프레임을 전송할 때, sequence number가 너무 크면 실제 데이터가 아닌 부분에 용량이 쓰이니까 효율적으로 sequence number에 몇 비트를 사용할 것인가

- The ACK also has a sequence number
  - It indicates the sequence number of "<u>next expected frame</u>"
  - If receiver receives frame(0), it sends ACK(1)

Solution: Sequence Number

- Sequence number for stop-and-wait ARQ: [0,1]

- frame(0) → frame(1) → frame(0) → frame(1) → …

Stop-and-Wait: Frame structure

- Frame(0) 1비트만 사용할 때

0	DATA	CRC

- Transmission sequence
  - S: sends frame(0)
  - R: sends ACK(1)
    - means the sender should send frame(1)
  - S: sends frame(1)
  - R: sends ACK(0)
    - means the sender should send frame(0)

Stop-and-Wait: Error case

- Retransmission case 1: frame lost
  - S: sends frame(0)
  - R: does not receive
  - S: timeout called, retransmits frame(0)
  - R: receives frame(0), sends ACK(1)
  - S: receives ACK(1)

- No problem

Stop-and-Wait: Error case

- Retransmission case 2: ACK lost
  - S: sends frame(0)
  - R: receives frame(0), sends ACK(1) → lost
  - S: timeout called, retransmits frame(0)
  - R: receives frame(0), sends ACK(1)
  - S: receives ACK(1)

- Problem solved

Stop-and-Wait: Error case

- Retransmission case 3: ACK arrives after timeout
  - S: sends frame(0)
  - R: receives frame(0), sends ACK(1)
  - S: timeout called, retransmits frame(0)
  - S: receives ACK(1), sends frame(1)
  - R: receives frame(0), sends ACK(1), discards frame(0)
    - Receiver already has frame(0) and is waiting for frame(1)
  - R: receives frame(1), sends ACK(0)
  - S: receives ACK(1), discards ACK(1)
    - Sender already received ACK(1)
  - S: receives ACK(0), sends frame(0)
    - frame(0) is a new frame that comes after frame(1)

- Problem solved

Stop-and-Wait: Error case

단점: 성능이 안좋음

Stop-and-Wait: Performance

- Suppose a system uses stop-and-wait ARQ. The link bandwidth is 1Mbps, and RTT is fixed at 20ms.

- Assuming no error and infinite timeout, what is the throughput? Suppose frame size is 1000 bits.

20ms마다 1000bit 전송 -> 50000bps
근데 회선이 1초에 보낼 수 있는 양은 1Mbps(bandwidth) -> 회선을 낭비하고 있음

- What is the ratio of throughput and link bandwidth?

  → bandwidth is wasted!

1 : 50

Stop-and-Wait: Performance

- Problem: When sender sends a frame, it has to wait for ACK before sending the next frame
  → resource waste

Go-Back-N

Stop-and-Wait → Go-Back-N

- It is wasteful to wait for ACK after sending a single frame

- Before waiting an ACK, send multiple frames in order to fill up the link pipe

한번에 1 프레임을 보내던 stop and wait에서
한번에 여러 프레임을 보내는 방식을 해야 효율적
대신 묶어서 보낸 프레임에 대해서 문제가 생겼을 때 적절히 대응해야함

- If an error occurs at a particular frame, disregard all frames sent after the lost frame, and restart from there → Go-Back-N

1~10 프레임을 묶어서 보내고,
문제가 생긴 프레임(n)이 어딘지 receiver가 알려주면
그 n번째 프레임부터 다시 전송

Go-Back-N

- How many frames should the sender send before receiving an ACK?
- What should be the sequence number range?
- What should we do when an error occurs?

Sliding Window

- Manages which frame to send/receive
- Sender and receiver each has a sliding window
  - Sender: send window
  - Receiver: receive window
- Sliding window has a fixed <u>size</u>

Send Window

묶어서 보낼 수 있는 프레임 사이즈

- Sender can only send frames inside the window
- When ACK is received, the window slides to the right

a. Send window before sliding

0,1,2 번이 제대로 왔다(0,1,2에 대한 ack가 왔다)
3번을 시작으로 window가 옮겨짐

b. Send window after sliding

Send Window

a. Send window before sliding

b. Send window after sliding

Send Window

- The numbers indicate frame sequence number

- In figure a,
  - Frames 13-15 (left): frame sent, ACK received
  - Frames 0-6: frames sent, ACK not received yet
    - called “outstanding frames”
  - Frames 7-14: frames can be sent, but not ready
  - Frame 15 (right) and after: cannot send yet

Send Window

- S_f: First outstanding frame
  - The first frame not to have received ACK yet
- S_n: Next frame to read
  - The first frame that is not sent yet
- S_{size}: Send window size

Send Window: Sliding

- When an ACK is received, the window slides to the right

- If ACKs are received for frames 0-2 in figure a, the sliding windows moves as figure b.

Receive Window

한번에 15프레임씩 보내기 때문에 sequence number가 15 이상이어야 함

a. Receive window

b. Window after sliding

Receive Window

- Receiver can only receive frames inside receive window

- In figure a,
  - frames 13-2: frames received and acknowledged
  - frames 3: frames that can be received
  - frames 4 and after: frames that cannot be received

Sequence Number Range

- What should be the sequence number range of Go-Back-N ARQ?

너무 크면 안좋고
너무 작으면 구분을 못한다

- If sequence number range is too wide, too many bits are needed to represent sequence number
  - If sequence number range is [0…N-1], then $\log_2 N$ bits are needed

- If sequence number range is too narrow, receiver cannot distinguish new frame from retransmitted frame

Sequence Number Range

- If send window size is N, then the sequence number range must be wider than N!
  - If sequence number uses m bits, the send window size must be less than 2^m

- Example: if send window size is 4, then sequence number range should be at least [0,4]

Window size vs. Sequence number range

send window = 3
sequence num = 3

프레임을 받았는데 ack가 소실된 상황

a. Window size < 2^m

send window = 4
sequence num = 3

텍스트

sender는 재전송하려고 0부터 다시 보내는데
receiver는 이 0번이 재전송된게 아니라 자연스럽게 돌아온 0의 순서라
생각해 받아서 문제가 됨
-> sequence number를 size window보다 크게 하면 0이 아니라 4를 기다리니까
sender가 0을 재전송하면 재전송된 내용은 버릴 수 있다

b. Window size = 2^m

프레임 0을 보낼때 타이머를 켬
ack가 오면 타이머를 리셋

Go-Back-N: Normal case

타이머 - 각 프레임에 대해서 또는 여기처럼 윈도우에 대해서 사용

ack2가 소실되어도 ack3이 도달하면 receiver가 2까지 받았다는 걸 알 수 있다

Go-Back-N: Timeout case

Stop-and-Wait vs. Go-Back-N

- Stop-and-Wait is a special case of Go-Back-N
- Go-Back-N with send window size = 1 is Stop-and-Wait

Go-Back-N: Problem

- When timeout occurs, we need to go back and restart from where the error occurred

- This is due to the fact that receive window size = 1

- Example
  - The sender sends frames 1-5
  - Frame 1 is lost, other packets are successfully received
  - Still, go-back-N requires all frames to be retransmitted

Selective Repeat

Go-Back-N → Selective Repeat

- Overcome problem of Go-Back-N
- Receive window size: N
- Typically, send window size = receive window size

Selective Repeat: send window

Selective Repeat: receive window

- Even frame 3 is not received, receiver accepts and saves subsequent frames 4, 7, 9
- This is possible because receiver window size is not 1

Selective Repeat: receive window

- When frame 3 is received, <u>the receiver sends back ACK(5) to request frame(5)</u>, since it already has frame(4)

Selective Repeat: sequence numbers

- If sequence number uses m bits, both send window and receive window should be less than or equal to 2^{m-1}

Receiver's action

- The receiver window looks like the figure

- Communication sequence
  - frame(3) is received → send ACK(4)
  - frame(4) is received → send ACK(7)
    - since frames 5,6 is already received

5,6은 받았는데 3,4를 재전송하는 상황

Selective Repeat: NAK mechanism

- NAK: Negative acknowledgement
- When the receivers sends NAK(1), it means that a subsequent frame was received without frame(1)
- Without NAK, the sender would have to wait until timeout

Selective Repeat: NAK mechanism

receiver의 window에서 가장 앞 프레임이 오면 ack를 보내고 다른 프레임이 오면 nak를 보냄

각 프레임에 대해 타이머를 쓰고 있음
한 프레임을 보낼 때 start, ack 받을 때 stop

타임아웃 걸리기 전에 조치하려고

첫 missing frame에만 nak를 보냄
nak가 계속가는 상황을 방지하기 위해

Selective Repeat: NAK mechanism

- When does the receiver send NAK?
  - When a frame is received, but a previous frame is still missing
  - Example: frame 3 is received, but frame 1 has not arrived yet → send NAK(1)

- NAK is sent only once for the same sequence number
  - Receiver does not receive frame(1)
  - Receiver receives frame(2) → send NAK(1)
  - Receiver receives frame(3) → <u>do nothing</u>

Selective Repeat: NAK mechanism

- When does the receiver send ACK?
  - When a frame is received, and receiver has all the previous frames

Selective Repeat: Example

- Receiver's action (frame(0) is the first frame)
  - receives frame(0) → sends ACK(1)
  - receives frame(1) → sends ACK(2)

receiver는 2번을 기다
리는데 3이옴 -> nak2

  - receives frame(3) → sends NAK(2)
  - receives frame(4) → do nothing
  - receives frame(5) → do nothing

  - receives frame(2) → sends ACK(6)

End of Chapter