12. Modeling and Analyzing Transport Protocols

이 장은 **Maude를 이용해 신뢰성이 낮은 전송 매체 위에서 신뢰할 수 있고 순서가 보장된 통신 프로토콜을 모델링/분석하는 방법**을 다룬다.

IP는 메시지의 신뢰성/순서를 보장하지 않기 때문에, TCP처럼 상위 계층에서 별도의 메커니즘으로 신뢰성과 순서를 확보해야 한다.

먼저 12.1절에서는 **순서 번호와 확인 응답을 이용해 메시지 시퀀스를 신뢰적으로 전달하는 단순한 프로토콜**을 소개한다.

전송 인프라가 순서는 유지하지만 신뢰성이 낮은 경우에는 **두 개의 순서 번호만 사용하는 교대 비트 프로토콜(Alternating Bit Protocol)**로 단순화할 수 있다.

그러나 이는 효율성이 낮아, 다음 메시지를 보내기 전에 반드시 ACK을 기다려야 한다. 이에 따라 12.2절에서는 **슬라이딩 윈도우 프로토콜**을 통해 여러 메시지를 연속적으로 전송할 수 있도록 확장한다.

슬라이딩 윈도우는 TCP의 핵심 구조로 널리 사용된다.

12.3절에서는 **순서 보장이 없는 환경과 있는 환경 모두에서 적용 가능한 슬라이딩 윈도우 프로토콜의 일반화된 버전**을 설명하되, 실제 Maude 모델링과 분석은 독자가 수행할 과제로 남겨둔다.

12.1 Reliable Communication Using Sequence Numbers

12.1절은 **순서 번호(sequence number)**와 **확인 응답(ACK)**을 이용하여, 신뢰성이 낮고 메시지 손실/재정렬이 발생할 수 있는 채널에서도 **메시지를 완전하고 순서대로 전달하기 위한 기본 프로토콜**을 설명한다.

- **송신자(Sender)의 동작**
  - 각 메시지에 순서 번호를 부여한다(1, 2, 3, ...).
  - 메시지를 전송한 후, 해당 순서 번호에 대한 ACK을 기다린다.
  - 일정 시간 내에 ACK을 받지 못하면 동일 메시지를 재전송한다.
  - ACK을 받으면 다음 메시지로 진행한다.
- **수신자(Receiver)의 동작**
  - **새로운 순서 번호의 메시지**를 수신하면 저장(또는 애플리케이션에 전달)하고 ACK을 보낸다.
  - 이미 수신한 순서 번호의 메시지는 무시한다(중복 메시지 제거).
  - 가장 최근에 본 순서 번호에 대한 ACK을 주기적으로 재전송하여 송신자가 상태를 확인할 수 있도록 한다.
- **시간 개념을 제거한 추상화 모델**
  - 실제 시간(타임아웃)을 제거한 더 단순한 모델에서는:
    - 송신자는 ACK을 받을 때까지 동일한 메시지를 계속 전송한다.
    - 수신자는 가장 큰 순서 번호에 대한 ACK을 반복적으로 전송한다.

12.1.1 Maude Modeling

이 절에서는 **메시지 손실과 순서 뒤바뀜(out-of-order)이 발생할 수 있는 환경**에서 동작하는 간단한 신뢰적 전송 프로토콜을 Maude로 모델링하는 방법을 설명한다. 이를 위해 Section 11.2에서 소개된 **표준 메시지 래퍼 (envelope)** 모델을 사용하며, 모든 메시지는

```
msg content from sender to receiver
```

형태로 표현된다.

메시지 콘텐츠 모델링

- 송신자가 전송하려는 데이터는 문자열들의 리스트이며, 각 문자열은 시퀀스 번호(sequence number)와 함께 전송된다.
- 예시 형태:
  - `"great" withSeqNo 4`
  - `ack withSeqNo 3` (ACK 메시지도 동일하게 시퀀스 번호를 가짐)
- 모듈 `SEQNO-UNORDERED`는 다음 구성요소를 포함한다:
  - Content : 전송되는 메시지의 내용(문자열 또는 ack)
  - ack : ACK 메시지를 나타내는 특수 콘텐츠
  - withSeqNo : 콘텐츠와 시퀀스 번호를 묶어 **MsgContent**를 생성
  - StringList : 문자열 리스트 (송신해야 할 메시지 목록 또는 수신된 메시지 목록)

Sender 객체 모델링

- Sender는 다음 정보를 가진 객체이다:

속성	의미
`msgsToSend: StringList`	아직 전송되지 않은 문자열 리스트
`currentMsg: StringList`	현재 전송 중인 메시지(없을 경우 nil)
`currentSeqNo: Nat`	현재 메시지의 시퀀스 번호
`receiver: Oid`	수신자 ID

- 초기 준비 단계

  `currentMsg`가 비어 있으면 다음 메시지를 준비하고 `currentSeqNo = 1`로 설정한다.
- 메시지 전송 규칙

  현재 메시지를 다음과 같이 전송한다:

  ```
  msg (S withSeqNo N) from O to O'
  ```
- ACK 처리
  - **현재 시퀀스 번호에 대한 ACK를 받으면** 다음 메시지로 이동하고 시퀀스 번호를 증가시킨다.
  - **모든 메시지를 전송한 경우** `currentMsg`는 nil이 된다.
  - 낡은 ACK(N < currentSeqNo)* 무시한다.

Receiver 객체 모델링

- Receiver는 다음 정보를 가진다:

속성	의미
`greatestSeqNoRcvd: Nat`	지금까지 받은 가장 큰 시퀀스 번호
`sender: Oid`	송신자 ID
`msgsRcvd: StringList`	수신한 문자열 리스트

- ACK 전송

  Receiver는 자신이 본 가장 큰 시퀀스 번호에 대해 ACK를 주기적으로 보낸다:

  ```
  msg (ack withSeqNo NZ) from O to O'
  ```

- 새로운 메시지 수신

  새로운 시퀀스 번호(S withSeqNo N)가 현재보다 크면:

  - `msgsRcvd`에 추가한다. (메시지 저장)
  - `greatestSeqNoRcvd`를 N으로 갱신한다.

- 오래된 메시지 무시

  이미 본 메시지(N ≤ greatestSeqNoRcvd)는 무시한다.

12.1.2 Formal Analysis

- 프로토콜 검증을 위해 Maude에서 초기 상태(init)를 정의한다.
  - Alice는 문자열 리스트를 Bob에게 전송하며,
  - Bob은 수신한 메시지의 최대 시퀀스 번호를 기록한다.
- 단순 rewriting(`frew`)을 수행하면 정상적으로 모든 문자열이 순서대로 전달된 "좋은 실행"이 나온다.

  → 하지만 이는 **가능한 많은 실행 중 하나만** 확인한 것이다.
- 프로토콜의 모든 가능한 실행에서 오류가 발생하는지 확인하기 위해 `search`로 **bad state**(예: greatestSeqNoRcvd = 5 이지만 수신 리스트가 올바른 순서가 아님)를 탐색한다.
- 무한 상태 공간 때문에 일반 search는 종료하지 않는다.

  → bad state가 도달 불가능하기 때문에 Maude는 계속 탐색을 이어간다.
- **bounded search(예: search [1,25])**를 사용하면 일정 rewrite 단계 안에서 bad state가 있는지 빠르게 확인할 수 있다.
- bounded search에서 bad state가 발견되지 않더라도
  - 전체 상태공간을 다 탐색한 것은 아니므로
  - 프로토콜이 "완전히 안전하다"고 증명되는 것은 아니다.
- 또한 이 분석은 **하나의 구체적인 초기 상태**에 대해서만 수행되었다는 점에서,

  → 다른 초기 조건에서는 오류가 발생할 가능성이 남아 있다.

12.2 The Alternating Bit Protocol

배경: Ordered + Lossy 통신 환경

기반 통신 인프라가 **순서(order)는 보장되지만 손실(lossy)**이 발생할 수 있는 메시지 전달 환경이라고 가정한다.

이때 Section 12.1의 프로토콜을 그대로 적용할 수 있으나, 링크 객체를 사용하는 정도만 달라진다.

문제점: 큰 시퀀스 번호의 비효율성

많은 메시지를 보낼 때 시퀀스 번호(sequence number)가 크게 증가하는 것은 비효율적이다.

손실은 발생하더라도 **중복(duplicating)은 발생하지 않는** 환경에서는 다음 사실을 이용할 수 있다:

- 수신된 가장 큰 시퀀스 번호가 n일 때,
- 현재 링크에 있는 메시지들의 시퀀스 번호는 **n 또는 n-1** 뿐이다.

따라서 시퀀스 번호 전체가 필요하지 않고, **짝수/홀수 여부(parity, n mod 2)**만 있으면 충분하다.

핵심 아이디어: 시퀀스 번호 → 패리티 비트 (0 또는 1)

기존 프로토콜의 시퀀스 번호 n을 다음으로 대체한다:

- n mod 2
  - 1번 메시지 → 비트 1
  - 2번 메시지 → 비트 0
  - 3번 메시지 → 비트 1
  - ...

즉, 송신자는 메시지를 보낼 때마다 비트를 **교대로 전환(alternating)** 한다.

이 최적화된 프로토콜을 **Alternating Bit Protocol (ABP)** 이라고 한다.

ABP 요약 규칙

1. Section 12.1의 프로토콜을 사용하되, **lossy link**를 사용한다.
2. 모든 시퀀스 번호 n은 **n mod 2**, 즉 0 또는 1의 **비트**로 대체한다.

Maude 명세 구조 요약

- BIT 모듈
  - 비트 타입 정의: `0`과 `1`
  - not 연산 포함
    - not(0) → 1
    - not(1) → 0
- MESSAGES 모듈
  - 기존 메시지 구조와 동일하되, 비트를 포함하는 메시지 타입 구성
  - 메시지 형식:
    - `ack`
    - `content withBit B`
- ALTERNATING-BIT-PROTOCOL 모듈
  - STRING-LIST + MESSAGES + LOSSY-LINK 포함

- 송신자(Sender) 클래스 구조:
  - msgsToSend
  - currentMsg
  - currentBit
  - receiver

주요 규칙(Rules) 요약

- [start]
  - 전송할 메시지가 있으면:
    - currentMsg ← 다음 메시지
    - currentBit ← 1 (초기 비트 설정)
- [sendCurrentMsg]
  - 현재 메시지를 링크에 (currentBit과 함께) 전송
  - 링크 객체에 `S withBit currentBit` 추가
- [receiveCurrentAckNotLast]
  - 수신한 ACK 비트 B가 currentBit와 같다면:
    - currentBit ← not(B) (교대 비트 적용)
    - 다음 메시지를 currentMsg로 설정

12.3 The Sliding Window Protocol

슬라이딩 윈도우 프로토콜은 앞서 소개된 단순 Stop-and-Wait(1개씩 보내고 ACK 기다림) 방식보다 효율적으로 **여러 메시지를 동시에 전송**할 수 있도록 확장한 신뢰적 통신 프로토콜이다.

송신자와 수신자는 각각 **윈도우(window)** 라는 버퍼를 유지하며, 이 윈도우의 크기를 **k**라고 한다.

개념 요약

- **윈도우(window)**
  - 송신자: 아직 ACK를 받지 않아 재전송 가능성이 있는 메시지 집합
  - 수신자: 이미 받은 일부 메시지를 임시 저장하는 버퍼
- **핵심 아이디어**
  - 송신자는 ACK를 기다리느라 멈추지 않고 **윈도우 내의 여러 메시지를 연속적으로 전송 가능**
  - 수신자는 순서가 어긋나도 저장해 두었다가 **누락된 메시지가 도착하면 순서대로 상위 계층에 전달**

송신자(Sender) 프로토콜

- 기본 동작
  1. 처음에 메시지 **1 ~ k**를 윈도우에 넣는다.
  2. 윈도우에 있는 메시지 중 **아무 것이나 지속적으로 전송**한다.
  3. ACK를 받으면 다음과 같이 행동한다.

- ACK 처리 규칙
  - **윈도우 밖의 메시지에 대한 ACK:** 무시
  - **윈도우 안의 메시지 번호 n에 대한 ACK:**

    → 윈도우를 "슬라이드"함

    → 메시지 **n+1 ~ n+k**를 새로 윈도우에 추가

    (더 이상 보낼 메시지가 없다면 추가하지 않음)

수신자(Receiver) 프로토콜

- 상태 변수
  - **currentAck = q**

    → 수신자가 **1, ..., q**까지 모든 메시지를 정상적으로 받았음을 의미

    → 즉, 마지막으로 완전히 전달된 메시지 번호
- 기본 동작
  1. 길이 **k**의 수신 윈도우를 유지
  2. 항상 **currentAck 번호에 대한 ACK**를 반복적으로 송신
  3. 번호 **≤ currentAck** 메시지는 중복이므로 무시
  4. 번호 **> currentAck** 메시지가 오면 아래와 같이 처리
     - 메시지 i > currentAck 를 받은 경우
       - Case 1: `currentAck+1 ... i-1` 메시지가 모두 이미 수신 윈도우에 저장되어 있는 경우
         - 즉, **i가 현재 비어 있는 구멍(홀)을 메꾸는 순간**
         - 다음을 수행:
           1. 수신 윈도우에서 **currentAck+1 ~ j**(최대 연속 메시지)까지를 애플리케이션에 전달
           2. currentAck ← j 로 갱신
           3. 윈도우를 j+1 ~ j+k 로 슬라이드
       - Case 2: i보다 앞선 번호 중 아직 받지 않은 것이 있는 경우
         - i를 수신 윈도우에 저장만 하고 기다림

핵심 동작 흐름 예시 요약

- 송신자
  - 초기 윈도우: [12, 13, 14]
  - ACK(14)를 받으면 → 윈도우를 [15, 16, 17]로 이동
  - ACK(16)를 받으면 → 윈도우를 [17, 18, 19]로 이동
- 수신자
  - currentAck = 11
  - 메시지 14 먼저 도착 → 12, 13이 없으므로 14는 저장
  - 이후 13 도착 → 저장

- 이후 12 도착 → 12, 13, 14 모두 연속되므로
  - application에 12, 13, 14 전달
  - currentAck = 14
  - 윈도우를 [15, 16, 17]로 이동

12.3.1 Sliding Window with Links

손실(lost) 가능성이 있는 **lossy links** 환경에서 슬라이딩 윈도우 프로토콜을 사용할 때, **시퀀스 번호를 최적화할 수 있다**는 내용을 설명한다.

- 핵심 아이디어
  - 일반 슬라이딩 윈도우는 이론적으로 매우 큰 시퀀스 번호가 필요할 수 있음
  - 하지만 **lossy link(중복 없음)의 조건**에서는 **시퀀스 번호를 단지 2k개만 사용하면 충분하다**
    - 여기서 **k = window size**
- alternating bit protocol과 연결
  - k = 1 일 때 :
    - 필요한 시퀀스 번호는 2개 → {0,1}

      → 이는 **alternating bit protocol**과 동일
  - 즉, alternating bit protocol = sliding window(k=1)의 특수 사례
- 예시
  - 윈도우 크기 k = 3
  - 필요한 시퀀스 번호는 2k = 6개

    → 0, 1, 2, 3, 4, 5
  - 패킷 5 다음 패킷의 번호는 다시 0으로 돌아감 (mod 6)
- 결론

  손실 링크 환경에서는 시퀀스 번호를 무한히 늘릴 필요가 없고, **윈도우 크기의 두 배인 2k만큼만 순환 번호로 사용하면 충분**하다.