ChatGPT

세션 2: 운영 신뢰성 설계 스터디 자료

강의 목표

(1) Broadcaster와 Confirmation Tracker를 왜 분리해야 하는지 설명할 수 있다.
(2) 재시도(retry)를 무작정 반복하는 것이 아니라 멱등성·재생 방지·백오프 정책으로 설계해야 하는 이유를 이해한다.
(3) 중복 안전성을 키 설계와 상태머신으로 구현하는 방법을 이해한다.
(4) 관측가능성을 "로그 많이 남기기"가 아니라 추적 가능한 식별자·지표·알람으로 구성할 수 있다.

핵심 메시지: **지갑 백엔드의 신뢰성은 "성공률"이 아니라 "중복과 재시도에 얼마나 안전한가"로 결정된다.**

1. 원칙 – Broadcaster와 Confirmation Tracker를 분리하는 이유

1.1 두 역할의 책임과 장애 패턴이 다르다

- **Broadcaster**는 전송자의 서명된 트랜잭션을 체인에 전파하는 일을 담당한다. 이 단계에서 발생하는 장애는 네트워크 타임아웃, RPC 요청 제한, 이미 메모풀에 존재하는 동일한 트랜잭션(txHash 중복) 오류, 네트워크 혼잡으로 인한 전파 지연, 같은 nonce를 가진 새 트랜잭션으로 기존 트랜잭션을 대체하는 등의 단기적인 실패가 많다. 이런 경우에는 빠른 재시도가 필요하다.
- **Confirmation Tracker**는 체인에 포함되었는지, 몇 번 컨펌되었는지, 안전(safe)·최종성(finalized)까지 도달했는지 판정한다. 이 단계에서 생기는 이슈는 블록 리오그(reorg)나 로그 removed, dropped·replaced·expired 트랜잭션 분류와 같이 시간이 걸리는 작업이다. 재시도는 "천천히, 꾸준히" 해야 하며, 금액이나 리스크 tier에 따라 완료 기준을 다르게 설정할 수 있다.

Ethereum의 트랜잭션은 주소마다 순차적인 **nonce** 를 사용한다. 특정 nonce의 트랜잭션이 실패하더라도 메모풀에 유지되기 때문에, 나중에 같은 nonce의 다른 트랜잭션이 포함되면 원래의 트랜잭션이 뒤늦게 포함되는 사례가 발생한다 [1]. 이런 레이스 컨디션 때문에 "발송"과 "완료 판정"은 분리된 책임이어야 한다.

1.2 분리하지 않았을 때의 문제

- Broadcaster 로직과 Tracker 로직이 섞여 있으면 네트워크 타임아웃을 "실패"로 간주해 같은 트랜잭션을 다시 발송하고 중복 출금 사고가 발생할 수 있다.
- 'send' 가 성공했다고 해서 완료했다고 착각하면 리오그나 dropped / replaced를 반영하지 못해 원장 불일치가 발생한다.
- 재시도 로직을 한 곳에서 다루면 언제 무엇을 재시도했는지 추적하기 어렵고, 장애 복구 시 원인을 파악하기 힘들다.

따라서 **Broadcaster는 "시도(attempt) 생성과 전파"만 담당하고, Confirmation Tracker는 "포함 여부 판단과 완료 선언"만 담당하는 구조로 분리해야 한다.**

2. 전략 – 재시도 정책의 3가지 요소

운영 신뢰성을 높이려면 재시도를 단순히 반복하는 것이 아니라 **(1) 멱등성(idempotency)**, **(2) 재생 방지(replay protection)**, **(3) 백오프(backoff)** 를 함께 설계해야 한다.

2.1 멱등성: "같은 요청은 같은 결과"

네트워크 타임아웃은 실패가 아니라 "모르는 상태"이다. 타임아웃이 발생하면 재시도가 필수인데, 동일한 요청을 여러 번 처리하는 과정에서 중복 처리 위험이 생긴다. 이를 해결하기 위해 두 수준의 멱등성 키를 설계한다.

(A) API 요청 레벨

서비스에 동일한 출금 요청이 두 번 들어오지 않도록 각 요청에 **idempotency key**를 부여한다. **AlgoMaster**의 idempotency 소개에서는 클라이언트가 고유한 식별자를 요청에 첨부하고 서버가 이를 데이터베이스에 저장해 중복 요청을 식별하는 방법을 설명한다 [2]. 또한 데이터베이스 수준에서는 **upsert**나 **unique constraint**를 활용해 동일한 idempotency key로 처리한 요청이 한 번만 성공하도록 강제해야 한다 [3]. 예를 들어 `withdrawal_id` 열에 UNIQUE 제약을 두고 동일한 idempotency key로 두 번 삽입되는 것을 방지한다.

(B) 체인 실행 레벨

EVM에서는 **from 주소와 nonce**가 사실상 체인 레벨 멱등성 키이다. 한 주소에서 전송하는 트랜잭션은 nonce가 증가해야 하며, 같은 nonce를 다시 제출하면 기존 트랜잭션이 대체되거나 dropped & replaced 상태로 분류된다 [4]. 이는 트랜잭션 재생 공격을 방지하는 역할도 한다. sendRawTransaction을 여러 번 호출할 때는 `(chain, from, nonce)`를 attempt의 기본 키로 사용하고, 체인 레벨 멱등성을 보장해야 한다.

2.2 재생 방지: "같은 서명이 다른 맥락에서 재사용되지 않게"

체인 간 리플레이: EIP-155는 트랜잭션 서명에 **chainId**를 포함해 다른 체인에서 동일한 서명을 재생할 수 없도록 한다. `Zokyo Auditing Tutorials`는 EIP-155가 여러 네트워크에서 서명된 메시지가 다른 체인에서 재생되는 공격을 방지한다고 설명한다 [5]. Chain ID는 서명에 포함되기 때문에 트랜잭션은 의도한 체인에서만 유효하다 [6].

시스템 내부 리플레이: 내부에서도 같은 승인 결정이나 서명을 반복 실행하는 것을 막아야 한다. 예를 들어 이미 승인된 withdrawal을 다시 signer에 보낼 경우 중복 처리 사고가 발생할 수 있다. ApprovalId, PolicyDecisionId, SignatureRequestId와 같은 식별자를 만들어 **한 번 사용한 결정은 다시 사용할 수 없게** 해야 한다. 이 식별자는 DB에서 UNIQUE 제약으로 관리한다.

2.3 백오프: "재시도를 질서 있게"

단순히 재시도 횟수를 늘리는 것보다, **시간 간격을 늘려가며 재시도**해야 시스템을 과부하에서 보호할 수 있다.

- **지수 백오프**: 재시도마다 대기 시간을 1 → 2 → 4 → 8 → 16 초처럼 기하급수적으로 늘린다. Better Stack의 분산 시스템 백오프 가이드는 지수 백오프가 네트워크 실패를 처리하는 데 효과적이며 처음에는 빠르게 재시도하고 점차 속도를 늦춘다고 설명한다 [7].
- **지터(jitter)**: 동일한 오류를 경험한 여러 클라이언트가 동시에 재시도하면 "떼 쓰는 문제(thundering herd)"가 발생할 수 있다. 랜덤한 지터를 대기 시간에 추가하면 각 클라이언트가 서로 다른 시간에 재시도해 폭주를 줄일 수 있다 [7]. 예를 들어, `actualDelay = random(0, exponentialDelay)`와 같이 구현한다 [8].
- **상한(cap)과 구분**: 백오프 간격은 최대로 60~120 초 정도로 제한한다. Broadcaster는 초·분 단위로 빠른 재시도가 필요하고, Tracker는 분·시간 단위로 느리게 반복해야 한다. 단기간의 네트워크 장애와 장기간의 체인 포함 지연의 특성이 다르기 때문이다.

3. 중복 안전성을 확보하는 구조

중복은 피할 수 없다. 사용자 재전송, API 타임아웃 후 재시도, 워커 장애 후 재기동, 메시지 큐의 at-least-once 전달, RPC 응답 누락 등으로 인해 동일한 요청/트랜잭션이 여러 번 처리될 수 있다. 시스템은 중복을 막는 것이 아니라 **중복을 받아들여도 결과가 안전**하도록 설계해야 한다.

3.1 중복 안전성을 위한 4대 장치

1. **상태머신의 방향성과 조건**: 트랜잭션 상태가 `BROADCASTED → INCLUDED → SAFE → FINALIZED`처럼 단방향으로 진행해야 한다. 이미 `INCLUDED` 상태인 요청을 다시 `BROADCASTED`로 되돌리는 전이는 금지한다. 상태 전이 조건을 명시적으로 정의해 로직 오류나 중복 처리 시에도 이전 상태로 회귀하지 않도록 한다.

2. **DB 유니크 키/제약조건**: 코드 수준에서 중복을 막는 것만으로는 장애나 재시작 상황에서 안전하지 않다. **AlgoMaster**의 idempotency 전략에서 언급한 것처럼, unique constraint나 upsert를 통해 중복 삽입을 차단해야 한다 [3]. 예를 들어 `(chain, from, nonce)`로 구성된 `nonce_key` 컬럼에 UNIQUE 제약을 두어 같은 nonce를 다시 예약(reserve)하지 못하게 한다.

3. **원장 이벤트는 append-only**: 전통적인 CRUD식으로 잔고를 덮어쓰면 사고 발생 시 복구가 불가능하다. **Baytech Consulting**의 이벤트 소싱 해설은 이벤트 저장소가 **append-only 로그**라는 점을 강조한다; 이벤트는 수정·삭제되지 않고 새로운 보상 이벤트로만 되돌린다 [9]. 잔고를 `RESERVE`, `COMMIT`, `RELEASE` 같은 이벤트로 기록하고 이를 합산해 현재 상태를 계산하면, 중복 처리나 재시도 시에도 로그를 재생하여 정확한 상태를 복원할 수 있다.

4. **외부 호출은 Outbox 패턴으로 분리**: 데이터베이스 업데이트와 외부 RPC 호출을 같은 트랜잭션에 묶으면 장애 시 불일치가 발생할 수 있다. `microservices.io`의 Transactional Outbox 패턴은 메시지를 DB에 저장한 후 별도 프로세스가 메시지를 브로커로 전송하도록 하여 2PC 없이도 데이터베이스 업데이트와 메시지 발송의 원자성을 확보한다고 설명한다 [10]. 이 패턴을 사용하면 워커가 실패 후 재시작해도 메시지를 다시 전송할 수 있으며, 소비자는 메시지를 idempotent하게 처리해야 한다.

4. 관측가능성 – 로그만으로는 부족하다

운영자는 **볼 수 있는 시스템**만 관리할 수 있다. 단순히 로그를 많이 남기는 것이 아니라, 요청과 트랜잭션을 끝까지 추적할 수 있는 식별자와 메트릭을 정의하고 알람을 설정해야 한다.

4.1 통일해야 하는 5개의 식별자

- **withdrawal_id** : 업무 단위(출금 요청) 식별자
- **attempt_id** : 실행 단위(시도) 식별자
- **idempotency_key** : API 요청 중복 방지 식별자
- **nonce_key (chain, from, nonce)** : 체인 레벨 멱등성 식별자
- **tx_hash** : 트랜잭션 해시(관찰 단서, 키 자체는 아님)

이 식별자는 로그의 MDC(Mapped Diagnostic Context), 분산 트레이스의 span attribute, DB 스키마 등에 동일하게 남겨야 한다. **Last9**의 글에서 설명하듯, **Correlation ID**는 요청이 여러 서비스를 통과할 때 같은 식별자를 유지해 로그를 연결해 주며 [11], Trace ID는 span을 계층 구조로 추적할 수 있게 해준다 [12]. 이러한 식별자 없이 로그만 쌓이면 장애 상황에서 어떤 요청이 어떻게 처리되었는지 파악하기 어렵다.

4.2 운영 KPI(지표)

- **Broadcaster 측정지표**

- `send 성공률/실패율`
- `RPC 오류율` (429 rate limit, timeout 등)
- `time_to_broadcast` : 출금 요청을 받았을 때 첫 attempt를 발송하기까지의 지연
- `replacement_count` : 수수료 상향을 위해 같은 nonce로 replacement한 횟수

- **Tracker 측정지표**

- `time_to_inclusion` : broadcast → 첫 블록 포함까지 걸린 시간
- `time_to_safe` , `time_to_finalized` : 포함 후 safe / finalized에 도달하기까지 걸린 시간. Ethereum Beacon Chain은 safe 블록과 finalized 블록을 구분하며, safe 블록은 두 분의 세 이상의 검증자 attestations를 받은 블록, finalized 블록은 한 epoch 뒤에 거의 되돌릴 수 없는 블록이다 [13] [14] .
- `dropped_count` , `reorged_out_count` : dropped / reorg로 분류된 거래 수
- `pending_age_p95` : 오래 걸리는 트랜잭션의 95번째 분위수 나이

- **시스템 지표**

- `큐 backlog` : 작업 큐에 쌓여 있는 메시지 수
- `워커 재시도 횟수 분포` : 각 워커가 몇 번 재시도했는지 분포
- `동일 idempotency_key 재요청 빈도` : 클라이언트가 같은 idempotency key를 다시 보내는 비율 (재시도 원인 분석)

4.3 알람

- 특정 금액 또는 리스크 tier에 따라 출금의 `pending_age` 가 30분 이상 증가하는 경우
- `dropped` / `replaced` 트랜잭션이 평소보다 급증하는 경우 (메모풀 혼잡 또는 replacement 정책 설정 문제)
- RPC 오류율이 일정 임계치를 넘는 경우
- `time_to_finalized` 가 급격히 늘어나는 경우 (네트워크 혼잡 또는 체인 리오그 가능성)
- 원장 상에 `RESERVE` 이벤트는 있지만 `COMMIT` 이벤트가 없는 상태가 증가하는 경우 (원장 정합성 문제)

알람은 SLA나 리스크 tier를 고려해 다단계로 설정하고, 모든 알람에 위에서 정의한 식별자를 포함해 원인 분석과 대응을 쉽게 한다.

5. 운영 신뢰성 체크리스트 (제출용 1 page)

아래 체크리스트는 설계 리뷰 시나 운영 준비 시 바로 사용할 수 있는 항목들이다.

영역	체크 항목
구조	Broadcaster와 Confirmation Tracker의 책임이 명확하게 분리되어 있는가?
멱등성	API 레벨 멱등성 키가 있는가? (동일 요청 → 동일 `withdrawal_id`)
	체인 레벨 멱등성 키 `(chain, from, nonce)` 가 정의되어 있고 UNIQUE 제약이 적용됐는가?
재생 방지	EIP-155 Chain ID를 사용해 체인 간 리플레이를 방지하고 있는가 [6] ?
	내부 리플레이 방지 식별자(ApprovalId/SignatureRequestId 등)가 있어 한 번 승인된 의사결정을 다시 실행하지 않는가?

영역	체크 항목
재시도 정책	재시도 로직이 지수 백오프 + 지터를 적용하며, Broadcaster/Tracker별로 다른 시간 스케일을 사용하고 있는가 [7] ?
중복 안전성	중복 처리 허용을 전제로 상태머신이 단방향으로 설계되어 있는가?
	`withdrawal_id`, `idempotency_key`, `nonce_key` 등 모든 키에 DB UNIQUE 제약이 있는가 [3] ?
	원장/감사 로그가 append-only 이벤트 모델로 구현되어 있는가 [9] ?
	외부 호출이 Outbox 패턴이나 작업 큐로 분리되어 재시도가 가능한가 [10] ?
관측가능성	로그·트레이스에 `withdrawal_id`, `attempt_id`, `nonce_key`, `tx_hash`가 항상 찍히도록 MDC/Span 속성이 통일돼 있는가?
	핵심 KPI(time_to_inclusion, dropped, reorg 등)와 알람이 정의돼 있고 대시보드에서 확인 가능한가?
	재시도 횟수, 백오프 대기 시간 등을 모니터링하고 있는가?

6. 강의 팁 – 자주 묻는 질문 대비

- **Q: "재시도는 몇 번 해야 하나요?"**
  A: 재시도는 횟수가 아니라 **멱등성 키**와 **백오프 전략**, **수동介入 경계**를 함께 설계해야 한다. 특정 횟수 이후에는 사람의介입이나 경보가 발생하도록 설계한다. 지수 백오프 + 지터를 사용해 장애를 증폭시키지 않도록 한다 [7] .

- **Q: "txHash로만 추적하면 안 되나요?"**
  A: EVM에서 같은 nonce의 트랜잭션은 다른 txHash로 대체(replacement)될 수 있으며, dropped & replaced 상태가 존재한다 [15] . 따라서 최소 `(chain, from, nonce)`를 키로 하는 attempt 모델이 필요하다. txHash는 관찰 단서로만 사용한다.

- **Q: "언제 완료를 선언해야 하나요?"**
  A: 완료는 단순히 트랜잭션이 포함(included)됐다고 해서 끝나지 않는다. Ethereum Beacon Chain에서는 **safe**와 **finalized**라는 추가적인 확정 레벨이 존재한다 [13] . 금액과 리스크 tier에 따라 **safe**만으로 완료를 선언할지, **finalized**까지 기다릴지 정책을 정의해야 한다.

- **Q: "중복 출금 사고를 방지하려면?"**
  A: 중복은 피할 수 없음을 인정하고, 멱등성 키와 상태머신·DB UNIQUE 제약·append-only 원장·Outbox 패턴을 조합해 중복 처리가 발생해도 결과가 변하지 않도록 설계한다.

본 자료는 백엔드 개발자 대상 강의를 위한 심화 학습 자료이다. 각 원칙과 전략에 대한 이론적 배경과 실제 설계 시 유의할 점을 이해하고, 돌발 질문이 나왔을 때 근거를 가지고 답변할 수 있도록 준비한다.

---

[1] Duplicate Ethereum Transaction Nonce article – Edge
https://support.edge.app/hc/en-us/articles/7073936846235-Duplicate-Ethereum-Transaction-Nonce-article

[2] [3] Idempotency | System Design | AlgoMaster.io
https://algomaster.io/learn/system-design/idempotency

4 15 Ethereum Transactions - Pending, Mined, Dropped & Replaced | Alchemy Docs
https://www.alchemy.com/docs/ethereum-transactions-pending-mined-dropped-replaced

5 6 Vulnerabilities of Missing EIP-155 Replay Attack Protection | Zokyo Auditing Tutorials
https://zokyo-auditing-tutorials.gitbook.io/zokyo-tutorials/tutorial-49-not-conforming-to-eip-standards/vulnerabilities-of-missing-eip-155-replay-attack-protection

7 8 Mastering Exponential Backoff in Distributed Systems | Better Stack Community
https://betterstack.com/community/guides/monitoring/exponential-backoff/

9 Event Sourcing Explained: The Pros, Cons & Strategic Use Cases for Modern Architects
https://www.baytechconsulting.com/blog/event-sourcing-explained-2025

10 Pattern: Transactional outbox
https://microservices.io/patterns/data/transactional-outbox.html

11 12 Trace ID vs Correlation ID: Understanding the Key Differences | Last9
https://last9.io/blog/correlation-id-vs-trace-id/

13 14 What are Ethereum commitment levels?
https://www.alchemy.com/overviews/ethereum-commitment-levels