

질리카(Zilliqa)

해당 번역본은 이용자가 디지털자산 관련 정보를 편리하게 확인할 수 있도록
단순 참고용으로 번역된 것이며, 투자 권유를 목적으로 하지 않습니다.
그 내용이 정확하지 않을 수 있으므로 정확한 정보 습득을 위해서는
원문을 참고하시거나 원문 작성 주체 측에 문의하시기 바랍니다.
두나무는 해당 번역본 내용의 진실성 및 정확성을 보장하지 않고,
제공된 정보에 의한 투자 결과에 대하여 어떠한 책임도 지지 않으며,
투자 손실은 투자자에게 귀속됩니다.

질리카 2.0 백서

버전 1.0

초록 - 블록체인 기술이 다양한 영역에서 채택됨에 따라, 요구 사항의 이질성은 불가피해졌다. 획일화된 단일 솔루션은 사용자 기대를 충족하지 못하며, 단일 체인에서 변동하는 트랜잭션 가격을 바탕으로 블록 공간을 두고 경쟁할 필요 또한 충족되지 않는다. 모든 중앙화 정보 시스템의 니즈를 충족하는 이상적인 컴퓨팅 인프라가 존재하지 않듯이, 모든 탈중앙화 사용 사례가 요구하는 바를 완벽히 충족할 수 있는 단일 체인은 존재하지 않는다.

본 백서는 웹3 공간에서 기존 및 미래 프로젝트의 서로 다른 요구사항을 해결하기 위해 설계된 질리카의 새로운 블록체인 플랫폼을 제시한다. 이 플랫폼은 상태 샤딩(state sharding)이라는 아이디어에 기반한 새로운 확장성 개념을 도입한다. 상태(state)는 여러 개의 x-shard로 세분화되며, 각 x-shard의 속성은 그 위에서 실행되는 애플리케이션의 특정 니즈에 부합한다. 이들의 공통된 목표는 낮은 지연(latency)과 빠른 트랜잭션 최종성(finality)을 통해 우수한 사용자 및 개발자 경험을 제공하고, 탈중앙화 애플리케이션과 상호작용할 때 웹2와 유사한 사용자 친화성을 제공하며, x-shard 간, 더 나아가 질리카 생태계를 넘어 스마트 컨트랙트 간의 원활한 상호운용성을 제공하는 것이다.

I. 서론(Introduction)

2017년에 제안된 질리카 블록체인 플랫폼 [1] (이하 ZQ1)은 트랜잭션 샤딩(transaction sharding)과 병렬화를 가능하게 하기 위해 설계된 스마트 컨트랙트 언어인 Scilla를 도입해 대규모 확장을 최초로 달성한 블록체인 네트워크였다. 본 백서에서 제시하는 질리카 2.0 프로토콜 (이하 ZQ2)은 더 빠른 합의, 온체인 스마트 계정, 다른 EVM 체인과의 상호운용성 등 중요한 개선 사항을 포함하며, x-shard라 불리는 주권적이고 애플리케이션 특화된 샤드를 도입함으로써 샤딩의 패러다임을 정교화한다.

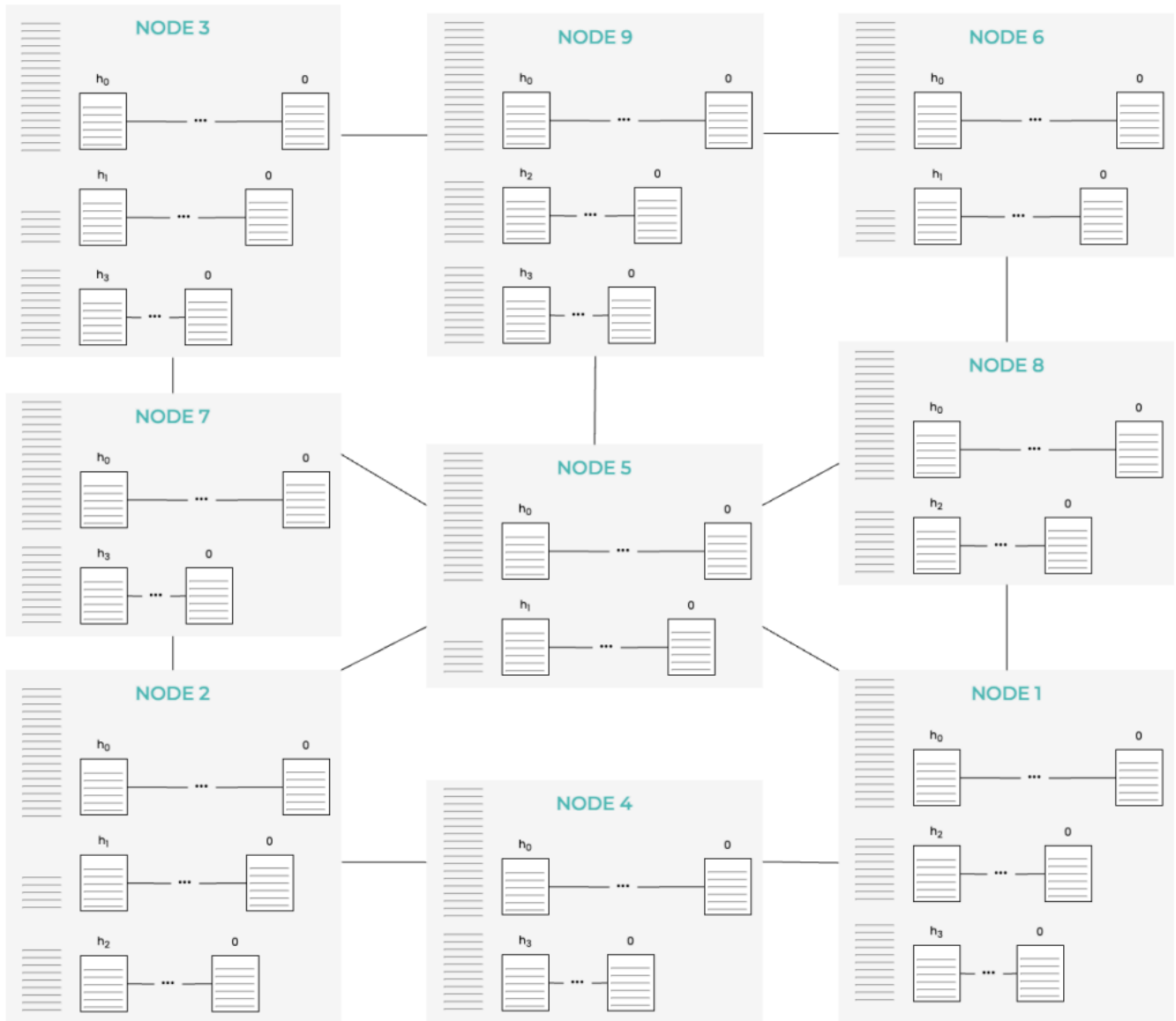


Fig. 1: The nodes of the network operate multiple x-shards, each having its own set of pending transactions (stacked lines on the left) and chain of blocks that starts at the genesis (rightmost rectangle).

x-shard는 논리적으로 분리되어 있으면서도 상호 연결된 체인으로, 그림 1에서 설명된 바와 같이 ZQ2 네트워크의 일부 노드 집합에 의해 운영된다. 어떤 노드가 어떤 x-shard에 참여할 수 있는지는 해당 노드의 권한에 따라 결정된다. 블록 크기 및 블록 시간, 가격 모델, 프라이버시 설정과 같은 x-shard의 속성은 사용자 요구에 맞게 조정 가능하며, 모든 x-shard의 근원이라 할 수 있는 ZQ2 메인넷에서 설정된다. 샤드가 생성한 마이크로블록을 병합하여 최종 블록을 구성했던 ZQ1과 달리, ZQ2 프로토콜은 x-shard들이 서로를 기다릴 필요가 없다. 각 x-shard는 자체 검증자 집합을 보유하며, 이들은 독립적인 합의 인스턴스를 자체 속도로 실행하고, 해당 x-shard의 상태를 관리하며 트랜잭션을 처리한다. 가장 중요한 점은, 서로 다른 x-shard에 배포된 스마트 컨트랙트가 서로의 상태에 접근할 수 있으며, 검증자에 의해 전달되는 샤드 간 메시지를 통해 상호작용할 수 있다는 것이다. 이러한 새로운 특성은 ZQ2의 짧은 블록 간격, 빠른 최종성, 라이트 클라이언트 지원 덕분에 가능해졌으며, 이는 조합 가능한 샤드 간 애플리케이션의 핵심 요소이다.

본 백서는 다음과 같이 구성된다. 제 II장은 시스템 환경과 적대적 가정을 소개한다. 제 III장은 ZQ2에서 사용되는 암호학적 기초 개념을 설명한다. 제 IV장은 풀 노드가 저장하는 데이터인 계정 기반 상태 모델, 트랜잭션, 블록을 개괄하며, 라이트 클라이언트가 모든 데이터를 저장하지 않고 필요한 데이터에 접근하는 방법을 설명한다. 제 V장은 검증자 노드가 합의를 통해 블록을 생성하고 최종 확정하는 과정을 설명한다. 제 VI장은 p2p 네트워크의 노드가 x-shard를 운영하고, 상호간 혹은 다른 EVM 체인과 연결하는 방법을 설명한다. 제 VII장은 스마트 컨트랙트에 초점을 맞추며, 프리컴파일(precompile)로 구현된 새로운 기초 개념을 소개한다. 제 VIII장은 ZQ2의 토크노믹스와 거버넌스 모델을 설명하며 결론을 맺는다.

II. 시스템 설정 및 가정(System Settings and Assumptions)

ZQ2에서는 ZQ1과 마찬가지로 사용자와 노드 두 가지 유형의 개체가 존재한다. 합의에 참여하는 노드를 검증자(validator)라고 부른다. 현재 블록을 제안하는 검증자는 리더(leader)로도 알려져 있다.

ZQ2는 제네시스(최초) 상태에서 ZQ1의 토큰 잔액을 가져옴으로써 ZQ1로부터 네이티브 토큰인 ZIL을 계승한다. 2021년에 스테이킹 프로그램이 시작된 이후, 스테이킹 시드 노드(staked seed node)가 운영하는 스테이킹 풀은 위임된 지분을 축적하여 ZIL 토큰 공급량의 30%를 초과하는 물량을 점유하게 됐다. ZQ2에서는 이러한 스테이킹 시드 노드가 메인넷 검증자의 역할을 맡음으로써 작업 증명(PoW)이 지분 증명(PoS)으로 대체된다. x-shard는 권한 증명(PoA)을 사용하는 별도의 검증자를 선택할 수 있는 옵션을 가진다.

ZQ1에서 작업 증명 기반 검증자 선택 알고리즘의 적대적 모델은 네트워크 해시레이트의 1/4 미만이 악의적 주체에 의해 통제된다고 가정했다. 이 모델은 샤드 위원회가 커야 할 필요성을 낳았다. 즉, 작업 증명에 따라 무작위로 선택된 노드를 위원회에 많이 추가할수록 위원회에 1/3 이상의 비잔틴 노드가 포함될 확률이 낮아졌다. 그러나 위원회가 커질수록 PBFT [2] 합의는 느려졌다. ZQ2에서 사용되는 작업 증명 기반 Fast-HotStuff [3] 합의에서는 절대 다수(supermajority)가 검증자의 수가 아닌 그들의 지분 총합으로 결정된다. 조정된 적대적 모델은 ZIL 스테이킹 총량의 1/4 미만이 적대자에 의해 통제된다고 가정한다. 검증자는 무작위로 선택되지 않고, 스테이커에 의해 위임된다. 새로운 적대적 가정 하에서, 전체 지분의 75% 이상을 포함하는 어떤 검증자 집합도 1/3 이상의 악의적 주체가 통제하는 지분을 포함할 수 없다는 것을 쉽게 증명할 수 있다.

프로토콜은 x-shard가 검증자 집합을 구성하기 위한 또 다른 옵션으로 권한 증명을 제공한다. 지분 증명과 달리, 권한 증명은 검증자 노드를 운영하는 알려진 개체들의 실제 세계 평판에 의존한다. 그들의 선택은 개별 x-shard의 구체적인 기준에 따라 이루어진다. PBFT와 마찬가지로, 검증자의 투표 권한은 가중치가 없으며, 절대 다수는 전체 검증자의 2/3로 정의된다.

III. 암호화 계층(Cryptographic Layer)

secp256k1 곡선으로 구현된 ECDSA [4]는 ZQ1 출시 이래로 사용자 인증에 사용되어 왔으며, ZQ2에서도 변경되지 않는다. ZQ1 프로토콜은 p2p 네트워킹 계층에서 독자적인 gossip 구현과 함께 Schnorr 서명을 사용하였다. ZQ2는 libp2p [5]를 사용하며, 노드들은 Ed25519 서명 방식 [6]을 기반으로 인증된다.

ZQ2의 합의 계층은 선형성을 달성하기 위해 Schnorr 서명 [8] 대신 BLS 서명 [7]을 사용하며, 이를 통해 서명 집계는 더욱 효율적으로 이루어진다. Schnorr는 공동 서명자들이 여러 라운드에 참여해야 하지만, BLS 서명은 단일 라운드에서 단순히 집계될 수 있다. 원래의 HotStuff 논문 [9]에서 제안된 것과 달리, ZQ2 합의는 임계치 서명(threshold signatures)을 사용하지 않으며, 이를 통해 공동 서명자를 식별하고 이중 서명(equivocation)을 시도하는 검증자를 탐지할 수 있다. 따라서 ZQ1에서 사용된 Schnorr 다중서명과 마찬가지로, ZQ2의 BLS 다중서명 또한 공동 서명자 목록을 동반하며, 이는 검증자 집합의 비트 벡터(bit vector)로 저장된다.

IV. 데이터 계층(Data Layer)

x-shard를 운영하는 모든 풀 노드가 유지하는 데이터는 상태(state), 상태를 변경하는 트랜잭션(transaction), 그리고 합의를 거친 트랜잭션의 전체 순서를 저장하는 블록(block)으로 구성된다.

A. 상태(State)

프로토콜은 계정 기반 상태 모델(account-based state model)을 사용한다. ZQ1은 EOA와 스마트 컨트랙트를 구분하였다. EOA는 ZIL 잔액과 논스만 보유했으나, 스마트 컨트랙트의 상태는 코드 해시와 스토리지 루트 해시도 포함하였다. Scilla 프로그래밍 언어로 작성된 스마트 컨트랙트를 위한 ZQ1의 API와의 호환성을 유지하기 위해, ZQ2의 스마트 컨트랙트 상태는 EVM 코드 해시와 선택적인 Scilla 코드 해시를 모두 포함한다.

ZQ2는 자체적으로(네이티브) 계정 추상화(account abstraction)를 구현하여 EOA를 스마트 계정으로 변환한다. 계정 추상화는 다중 인증, 일일 한도에 따른 트랜잭션 승인, 화이트리스트(whitelist) 및 기타 규칙, 다차원 논스, 가스비 보조(sponsored gas fees), ERC-20 토큰으로 지불되는 가스비 등 다양한 기능을 사용자 계정이 활용할 수 있게 함으로써 보안성과 사용자 경험을 향상시킨다. ZQ2의 네이티브 계정 추상화는 ERC-4337 [10]과 호환되며, 이를 통해 제3자가 구현한 위 기능들을 활용할 수 있다. ZQ2는 검증자를 번들러로 사용하고, 엔트리 포인트를 프리컴파일(precompile)로 구현하여 한 데 모인(bundled) 사용자 연산(user operation)의 병렬 실행을 가능하게 한다. ZQ1에서 가져온 EOA는 자동으로 스마트 계정으로 변환되며, 그 외 EOA는 트랜잭션에 처음 등장하는 시점에서 스마트 계정으로 변환된다. 프로토콜은 관련된 EOA의 코드 해시를 기본 스마트 계정 구현을 가리키도록 설정하고, 초기값으로 스토리지를 채운다. 기본 스마트 계정 구현은 EOA 동작을 모방하지만, 계정 소유자가 필요할 경우 언제든지 더 정교한 기능으로 업그레이드할 수 있다. 또한 ZQ2 프로토콜은 제출된 트랜잭션을 사용자 연산으로 변환하여, 엔트리 포인트를 통해 추가 처리할 수 있도록 한다.

스마트 컨트랙트의 스토리지에 데이터를 저장하거나 조회하는 트랜잭션은 해당 작업에 대해 가스를 지불한다. 그러나 데이터가 한 번 저장되면, 이후 트랜잭션이 그것에 접근하지 않더라도 풀 노드의 끊임없이 커지는 영속적 저장소에 보관되어야 한다. 컨트랙트를 값싼 데이터 저장소로 악용하는 것을 방지하기 위해, x-shard는 컨트랙트가 사용하는 스토리지 공간에 대해 임대료(rent)를 부과할 수 있는 옵션을 가진다. 각 컨트랙트에 대해 임대료 스폰서(rent sponsor)를 등록하도록 컨트랙트 소유자에게 요구할 수 있다. 스폰서는 컨트랙트 자체일 수도 있고, 다른 계정일 수도 있으며, 정의된 에포크 수마다 그 계정에서 임대료가 차감된다. 스폰서의 잔액이 너무 작으면, x-shard는 해당 컨트랙트와 관련된 트랜잭션을 거부하거나 트랜잭션 발신자에게 더 높은 가스비를 부과할 수 있다. x-shard가 보조(subsidize)하고자 하는 컨트랙트는 임대료에서 면제될 수 있다.

B. 트랜잭션(Transactions)

ZQ2는 레거시 트랜잭션, EIP-2930 [11]에서 정의된 타입-1 트랜잭션, 그리고 EIP-1559 [12]에서 정의된 타입-2 트랜잭션을 지원한다. 레거시 트랜잭션은 송신자의 논스, 가스 가격, 가스 한도, 목적지 주소, 금액, 입력 데이터, 그리고 송신자의 서명 (r, s) 과 패리티 v 를 저장하는 (v, r, s) 튜플로 구성되며, 이는 EIP-155 [13]에서 정의된 체인 아이디와 결합될 수 있다. 타입-1 트랜잭션은 추가적인 접근 목록(access list)을 포함하며, 체인 아이디는 서명의 v 요소와 별도로 저장된다. EIP-2930에서 도입된 선택적 접근 목록은 거래가 실행 시 접근할 상태의 일부를 명시할 수 있게 한다. 접근 목록을 사용하면 해당 연산 코드(opcode)에 부과되는 가스비에 대한 할인 혜택이 제공된다. 타입-2 트랜잭션은 타입-1 트랜잭션의 가스 가격을 EIP-1559에서 정의된 최대 수수료(max fee)와 최대 우선 수수료(max priority fee)로 대체해 블록 활용도를 기반으로 한 수수료 시장을 형성한다. 특정한 경우, ZQ2 EVM 트랜잭션은 위에서 설명한 표준 이더리움(Ethereum) 트랜잭션과 다르다. 예를 들어, VI. B 절에서 설명되는 샤드 간 트랜잭션(intershard transaction)은 송신자 주소를 복구할 수 있는 서명을 포함하지 않으므로, 송신자 계약의 주소와 x-shard를 별도의 필드에 저장한다. VII. A 절에서 설명되는 즉각적 샤드 간 상태 읽기(instant intershard state read)를 사용하는 트랜잭션은 다른 x-shard의 사전 로드된 상태 값(prefetched state values), 머클 증명(Merkle proofs), 블록 번호를 포함한다.

스마트 컨트랙트를 호출하지 않는 트랜잭션은 21,000 가스를 소비하며, 스마트 컨트랙트를 호출하는 경우 해당 컨트랙트가 사용하는 가스가 추가된다. 모든 트랜잭션에는 가스 한도가 존재한다. 하나의 블록에서 모든 트랜잭션이 사용할 수 있는 총 가스는 블록 한도에 의해 결정된다. x-shard의 블록 한도는 사용자 정의가 가능하며, 이는 x-shard 컨트랙트에 저장된 매개변수의 일부이다. ZQ2 메인넷의 블록 한도는 8,400만 가스로, 이는 4,000건의 단순 송금 트랜잭션과 평균 50,000 가스를 요구하는 1,680건의 컨트랙트 호출을 처리할 수 있는 수준이다.

x-shard가 트랜잭션 수수료를 처리하는 방법에는 여러 가지가 있다. 일부는 블록에서 발생한 가스 수수료를 블록 제안자에게 지급하고, 다른 일부는 블록의 QC에 기여한 검증자들에게 분배하거나 ZQ2 메인넷처럼 수수료를 소각한다. 가스 가격 책정에도 여러 옵션이 존재한다. 일부 x-shard에서는 트랜잭션 송신자가 가스 가격을 자유롭게 정하며, 다른 x-shard는 수수료 시장을 사용하여 블록 공간의 활용도에 따라 최소 가스 가격이 결정된다. 몇몇 x-shard는 가스 가격 경매를 구현할 수 있으며, 또 다른 경우에는 중요한 스마트 컨트랙트의 가스 가격을 보조하거나, 특정 개수의 이전 블록 동안 일정량의 가스를 사용한 사용자에게 할인을 제공할 수도 있다. 이러한 옵션들은 시스템 수준 스마트 컨트랙트와 네이티브 계정 추상화를 통해 쉽게 구현 가능하다.

ZQ2의 또 다른 편의 기능은 트랜잭션이 취소됐을 때 발생한 인코딩된 오류를 트랜잭션 영수증(transaction receipt)에 포함하는 것으로, 라이트 클라이언트가 주장된 오류가 실제로 발생했음을 검증할 수 있도록 한다.

C. 블록(Blocks)

ZQ2 블록은 헤더, 아래에서 더 자세히 설명할 쿼럼 증명(QC) 또는 집계된 QC, 그리고 트랜잭션 목록으로 구성된다. 블록 헤더에는 블록 높이, 뷰 번호, 블록 해시, 부모 해시, 유닉스 타임스탬프, RANDAO 리빌(reveal), 추가 데이터, 가스 한도와 사용된 가스, 발생한 로그의 블룸 필터, 상태 루트(state root), 트랜잭션 루트(transaction root), 영수증 루트(receipt root), 그리고 블록 제안자의 서명이 포함된다. 블룸 필터와 루트 해시는 이 절에서 후에 소개될 라이트 클라이언트에 필수적이다.

QC와 집계 QC는 ZQ2 합의에서 핵심적이다. QC는 블록에 대해 검증자 절대 다수(supermajority)의 투표를 나타낸다. 이는 검증자들이 투표한 블록 해시와 뷰 번호, 그들의 다중 서명(multisignature), 그리고 검증자 집합의 어떤 구성원이 투표했는지를 나타내는 비트 벡터로 구성된다. 집계 QC는 절대 다수의 검증자 인덱스, 뷰 번호, 그리고 최고 QC에 대한 서명을 집계한 것으로, 절대 다수가 집계 QC에 포함된 것보다 더 높은 QC를 보유하지 않음을 증명하는 역할을 한다.

라이트 클라이언트 프로토콜은 제한된 자원을 가진 노드가 블록 헤더만 처리하여 어떤 블록이 최종 확정되었는지를 결정할 수 있게 한다. 이는 신뢰 관계가 없는 RPC 노드로부터 직접 계산할 수 없는 암호학적으로 검증 가능한 데이터를 요청하는 과정을 수반하는데, 라이트 노드의 경우 풀 노드와 달리 직접 트랜잭션을 저장하거나 실행하지 않기 때문이다.

최종 확정된 블록 헤더를 판별하기 위해 라이트 클라이언트는 합의에서 정의된 커밋 규칙을 따라야 하며, 해당 x-shard의 검증자 집합을 기반으로 블록 헤더 내의 QC와 집계 QC를 검증해야 한다. 검증자 집합은 ZQ2 메인넷의 스마트 컨트랙트에 의해 관리되므로, 라이트 클라이언트는 메인넷 RPC 노드에 **eth_getProof** 메서드를 사용해 질의해야 하며, 이는 해당 컨트랙트의 스토리지 값과 머클 증명을 반환하여 라이트 클라이언트가 그 유효성을 검증할 수 있도록 한다. 검증자 집합 업데이트에 필요한 RPC 호출 횟수를 줄이기 위해, 검증자 집합은 VIII. B 절에서 설명하는 검증자 삭감에 따른 즉각적 제거를 제외하면 에포크 경계에서만 변경될 수 있다.

라이트 클라이언트는 트랜잭션 해시, 이벤트 로그, 블록에 포함된 계정 상태의 유효성을 블록 헤더에 포함된 트랜잭션 루트, 영수증 루트, 상태 루트를 사용해 검증할 수 있다. 검증은 머클 증명 [14]에 기반하며, 이는 해당 요소들을 리프(leaf)로 저장하는 머클 트리의 루트 해시를 계산하고 이를 블록 헤더의 루트 해시와 비교할 수 있게 한다. 리프의 머클 증명은 머클 트리의 각 레벨에서 그 리프의 형제 노드 해시들로 구성된다. 계산은 리프를 해싱하는 것으로 시작한다. 해시를 형제 노드의 해시와 합친(concatenate) 뒤 다시 해싱하여 트리에서 한 레벨 위의 부모 해시를 얻는다. 이 과정을 트리의 루트에 도달할 때까지 반복한다.

ZQ2 노드가 제공하는 **eth_getProof RPC** 메서드를 통해 라이트 클라이언트는 컨트랙트 스토리지를 포함한 계정 상태를 질의하고, 결과에 대한 머클 증명을 받을 수 있다. 그러나 이벤트 로그와 그에 대한 머클 증명을 함께 가져오는 RPC 메서드는 존재하지 않는다. 대신, 라이트 클라이언트는 알고리즘 1을 실행하여 RPC 노드가 반환한 이벤트 로그가 유효한지 검증할 수 있다. 원하는 로그가 포함된 블록을 찾기 위해, 라이트 클라이언트는 수신한 블록 헤더의 **logsBloom**을 확인할 수 있다. 블룸 필터 [15]에 의해 검색된 로그 토픽

은 거짓 양성(false positive)일 수 있으나, 필터에 해당 토픽이 없다면 라이트 클라이언트는 해당 블록의 영수증을 가져올 필요가 없음을 확실히 알 수 있다.

알고리즘 1: Generic event log verification

```
# input: logs from blocks whose logsBloom
# indicates positives

1 for log in logs:
2 rcpts = rpc.eth_getBlockReceipts(
  log.blockHash)
3
4 # check validity of receipts
5 if log.blockNumber not in rcptRoots:
6 rcptRoots[log.blockNumber] =
  merkle_tree(rcpts).root
7 if block[log.blockNumber].receiptRoot
  != rcptRoots[log.blockNumber]:
8 raise RootMismatch
9
10 # verify if log included in receipt
11 if log not in
  rcpts[log.transactionIndex].logs:
12 raise InvalidLogs
13
14 # search for withheld logs
15 moreLogs = [l for l in sum([r.logs for
  r in rcpts], []) if l not in logs]
16 if any(l.address in filter.address
  and filter.topics.match(l.topics)
  for l in moreLogs):
17 raise MissingLogs
```

새롭게 합류한 노드가 빠르게 동기화할 수 있도록, ZQ2는 약한 주관성 체크포인트(weak subjectivity checkpoint)를 지원한다. 이는 제네시스 블록과 동일한 방식으로 신뢰할 수 있고 되돌릴 수 없는 블록을 나타낸다. 마지막 체크포인트의 공개된 상태와 블록 해시는 해당 블록 헤더의 상태 루트와 일치해야 하며, 노드는 시작 시 이를 다운로드한다. 그 상태에 저장된 검증자 집합을 기반으로 노드는 이후 블록 헤더를 검증하고, 그 트랜잭션을 실행하며, 자신의 상태를 네트워크의 나머지 부분과 동기화할 수 있다. 이 메커니즘은 라이트 클라이언트를 동기화하는 데에도 유용하다. 라이트 클라이언트는 체크포인트 블록의 공개된 상태를 다운로드하지 않고, **eth_getProof RPC** 메서드를 사용해 그 안에 저장된 검증자 집합을 요청한 뒤 체크포인트 헤더에 대해 증명을 검증한다.

V. 합의 계층(Consensus Layer)

ZQ1은 합의 리더를 통해 메시지를 라우팅하고 검증자 서명을 집계함으로써 인증자 복잡성을 줄인 개선된 PBFT 알고리즘을 구현하였다. ZQ2는 PBFT를 완전히 대체하여, HotStuff [9]의 최적화된 변형인 Pipelined Fast-HotStuff를 채택하였다. 실패 시 리더를 교체하기 위해 비용이 많이 드는 뷰 변경 프로토콜이 필요한 PBFT나, 리더 교체 전에 최대 네트워크 지연을 기다리는 텐더민트(Tendermint)와 달리, HotStuff는 선형성(linearity)과 응답성(responsiveness)을 동시에 보장한다. 두 속성을 동시에 달성하는 대가로 2라운드 대신 3라운드의 투표가 필요하지만, 파이프라인화된 버전인 Chained HotStuff는 이를 일반화된 투표 라운드로 통합하여, 동일한 메시지를 사용해 여러 뷰의 다양한 단계를 동시에 표현한다.

A. 블록 생성 및 최종 확정(Block Production and Finalization)

Fast-HotStuff [3]는 타임아웃으로 인한 뷰 변경 시 허용 가능한 오버헤드를 감수하는 대신, 라운드 수를 2로 줄여 HotStuff를 개선한다.

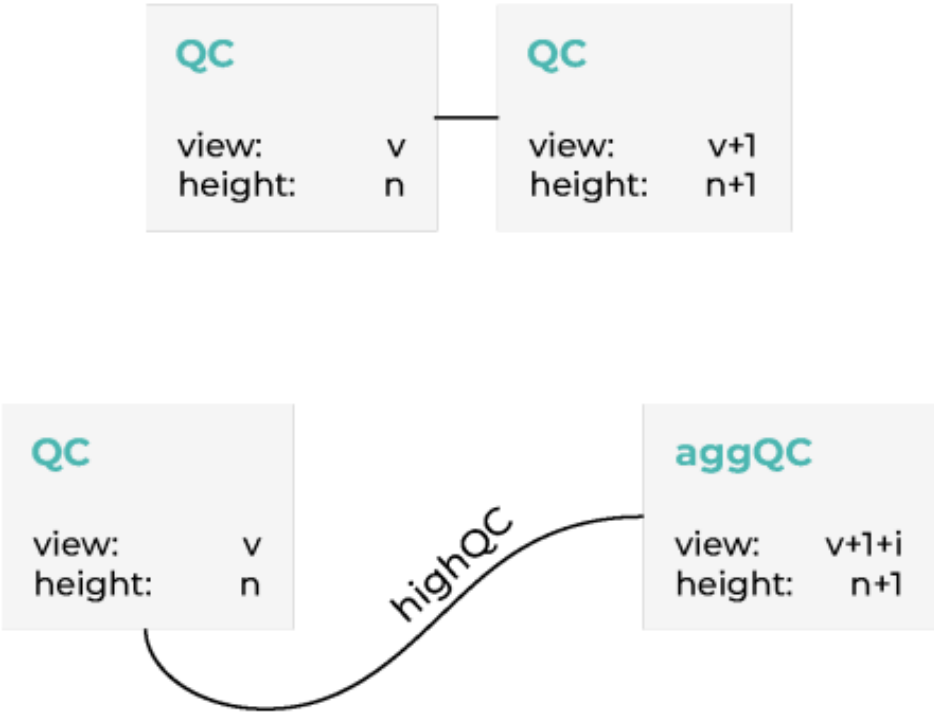


Fig. 2: The Pipelined Fast-HotStuff safe proposal rule. The new block is safe either if its parent was proposed in the previous view (top) or is referred to by the highQC among its aggQCs (bottom).

ZQ2에서 사용되는 Pipelined Fast-HotStuff에서는, 현재 리더가 새로운 블록 제안과 함께 부모 블록의 QC를 전파한다. 제안된 블록의 뷰 번호가 해당 QC의 뷰 번호보다 정확히 1 크고, 검증자의 로컬 뷰 번호보다 크거나 같다면, 검증자는 제안된 블록을 안전하다고 간주하고 서명된 블록 해시 및 뷰 번호와 함께 자신의 투표를 다음 리더에게 전송한다. 이 경우는 그림 2의 상단에 묘사되어 있다.

새로운 리더는 절대 다수로부터 이러한 투표를 수집하여 새로운 QC를 형성하고, 다음 블록을 제안한다. 검증자가 타임아웃이 만료되기 전에 안전한 블록 제안을 받지 못하면, 그들은 자신의 로컬 뷰 번호와 지금까지 받은 최고 QC에 대한 서명을 포함한 *new-view* 메시지를 다음 리더에게 보낸다. 다음 리더는 절대 다수로부터 *new-view* 메시지를 수집하여 그들의 집계 서명을 제안된 블록에 포함한다. 제안된 블록을 수신한 검증자는 그 블록이 집계 QC 중 최고 QC에서 연장된 것이라면 안전하다고 간주한다. 이 경우는 그림 2의 하단에 묘사되어 있다.

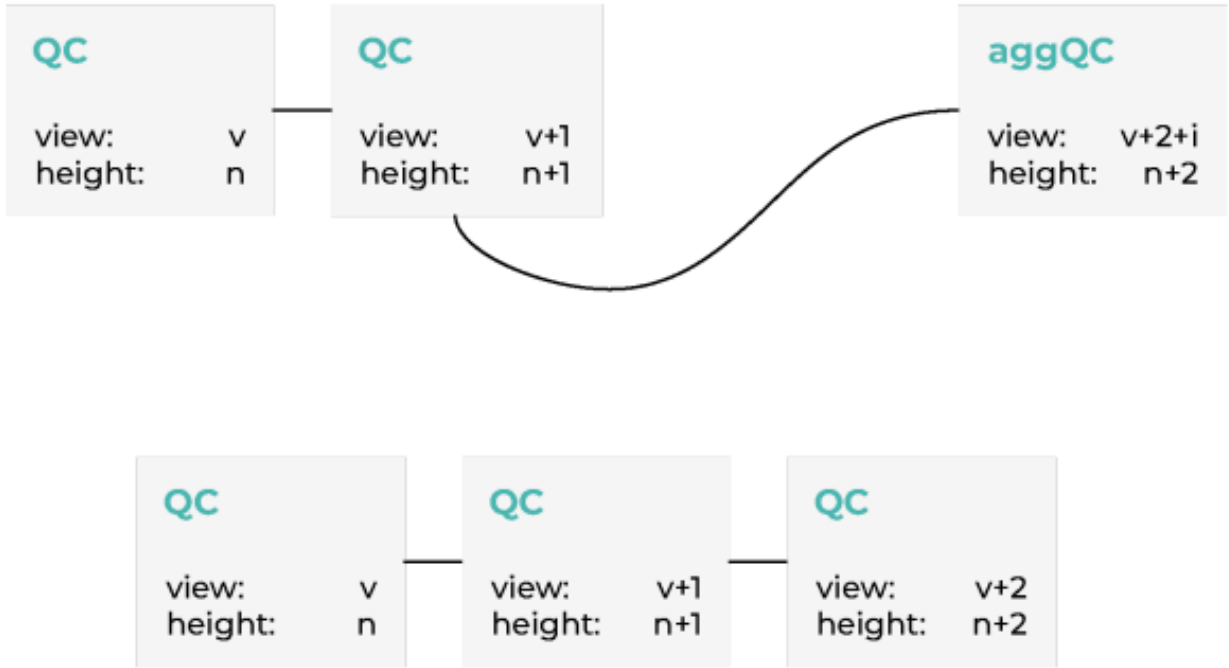


Fig. 3: The Fast-HotStuff commit rule. The block from view v is finalized when its child from view $v + 1$ becomes extended from by a new block in a later view. The two-direct-chain at the bottom illustrates the case when the extending block is from view $v + 2$.

Pipelined Fast-HotStuff의 커밋 규칙은 단순하다. 부모 블록과 조부모 블록 사이에 직접 체인이 형성된 2-체인이 도달되면, 조부모 블록과 그 조상 블록 모두를 실행할 수 있다. 결과적으로, 일반적인 경우 블록은 연속된 두 개의 블록 이후 최종 확정된다. 평균 블록 시간이 약 2초일 때, 블록을 최종 확정하는 데 약 5초가 걸린다. RPC 메서드에서 사용되는 블록 태그인 *safe*와 *finalized*는 각각 RPC 노드의 로컬 highQC가 참조하는 블록과, Pipelined Fast-HotStuff 커밋 규칙에 해당하는 가장 큰 번호의 블록을 의미한다는 점을 주목할 만하다.

B. 절대 다수 및 리더의 정의(Supermajority and Leader Definition)

x-shard는 검증자 집합의 구성에 따라 권한 증명 모드 또는 지분 증명 모드에서 Pipelined Fast-HotStuff를 사용할 수 있다. 권한 증명 모드에서는 검증자 투표 수가 검증자 집합 크기의 $\frac{2}{3}$ 이상일 때 절대 다수에 도달한 것으로 정의된다. 지분 증명모드에서는 투표한 검증자의 지분 합이 검증자 집합이 보유한 총 지분의 $\frac{2}{3}$ 를 초과할 때 절대 다수에 도달한다. 권한 증명은 검증자 투표가 검증자의 지분이 아닌 단순히 1로 가중되는 지분 증명의 특수한 경우로 간주될 수 있다.

지분 증명 검증자가 리더로 선택되는 빈도는 검증자의 지분이 검증자 집합 총 지분에서 차지하는 비율에 비례해야 한다. ZQ2의 리더 선택 알고리즘은 이더리움 비콘 체인에서 알려진 RANDAO 난수 축적기 (randomness accumulator) [16]를 사용한다. 이 방식은 블록 제안자가 자신들의 블록 안에서 현재 뷰 번호에 대한 BLS 서명을 공개하도록 요구한다. 새로운 블록을 수신한 노드는 제안된 블록에 공개된 BLS 서명의 해시를 이전 값과 XOR 연산하여 RANDAO 믹스를 갱신한다. 그 결과는 난수 소스(source of randomness)를 필요로 하는 스마트 컨트랙트가 다음 블록에서 전역 변수 **block.prevrando**를 통해 사용할 수 있으며, 또한 각 에포크 시작 시 노드들이 난수 생성기의 시드로 사용하여 다음 에포크의 블록 제안자를 결정하는 데 활용된다. 블록 수로 정의되는 에포크 길이는 x-shard의 구성 설정의 일부로서 보안 측면에서 상충 관계(tradeoff)가 존재한다. 에포크 길이가 길수록, 누구나 몇몇 블록 제안자를 미리 알 수 있어 그들을 매수하거나 공격할 위험이 있다. 반면, 에포크가 길면 악의적인 블록 제안자가 에포크 끝에서 자신의 BLS 서명을 공개하거나 블록을 제안하지 않음으로써 이를 숨길 수 있는 선택지를 가지더라도, 리더 선택의 편향성(biasability)에 의한 영향은 줄어든다.

지분 증명 검증자가 검증자 집합에 합류하거나 혹은 검증자 집합에서 떠나려면, 다음 에포크가 끝날 때까지 기다려 해당 집합에 실제로 추가되거나 혹은 해당 집합에서 제거되어야 한다. 다만, 떠난 후에도 그들의 지분은 언스테이킹/언본딩(unbonding) 기간이 끝날 때까지 잠긴 상태로 유지된다.

Pipelined Fast-HotStuff는 Chained HotStuff가 갖는 포킹 공격 취약성을 제거한다. 악의적인 리더가 조부모, 부모, 자식 블록으로 구성된 2-체인을 확인했을 때, 자식 블록에 대한 새로운 QC를 형성하는 대신 조부모를 참조하는 QC를 사용해 새로운 블록을 제안할 수 있다. 제안된 블록은 Chained HotStuff 규칙에 따라 안전한 것으로 간주되고, 다음 리더는 그 블록에서 연장하여 블록을 생성하게 된다. 그러나 이 경우 악의적인 리더는 부모와 자식 블록이라는 두 블록을 무효화하여, 그 안의 모든 거래를 지연시키고 이후 블록에 다시 포함되도록 만든다. Pipelined Fast-HotStuff는 리더가 이전 뷰에서 제안된 블록에서 연장하거나, 또는 오래된 뷰의 블록에서 연장하는 경우 그 블록이 검증자 절대 다수가 보유한 최고 QC와 일치함을 증명하는 집계 QC를 반드시 포함하도록 요구함으로써 포킹 공격을 방지한다. 따라서 악의적인 리더는 오래된 QC를 악용하여 다른 블록을 무효화하고 자신의 지분 점유율을 늘리는 행위를 할 수 없다.

VI. 네트워크 계층(Network Layer)

ZQ1은 지정된 시드 노드(seed node)를 사용하여 요청을 처리하고 RPC 인터페이스를 통해 제출된 트랜잭션을 수집하였다. 별도의 조회 노드(lookup node)가 트랜잭션을 검증자 노드에 전달하면, 해당 검증자 노드가 자신에게 할당된 샤드의 트랜잭션을 처리하였다. ZQ2의 노드는 RPC 요청을 처리하고, 블록 및 트랜잭션을 처리하며, 동시에 여러 x-shard의 합의에 검증자로 참여할 수 있다. 또한, 신뢰가 수반되지 않는 풀 노드를 통해 라이트 클라이언트 프로토콜을 사용하여 x-shard와 상호작용할 수도 있다.

A. 공개 및 비공개 x-shard(Public and Private X-Shards)

ZQ2는 ZQ1의 밀접하게 결합된 샤드 방식에서, x-shard라 불리는 새로운 패러다임으로 확장 방식을 변경하였다. x-shard는 합의 매개변수, 프라이버시 설정, 가격 정책을 맞춤 설정할 수 있는 공유 p2p 네트워크 상의 주권적 체인이다. 이를 통해 네트워크에 배포된 애플리케이션들은 블록 공간을 두고 서로 경쟁하지 않고, 자신의 니즈에 따라 운영될 수 있다. x-shard는 ZQ2 메인넷에서 관리된다. 새로운 x-shard의 제네시스 검증자 집합을 형성하는 노드를 설정한 후, 메인넷에서 트랜잭션을 제출함으로써 해당 x-shard를 등록할 수 있다. x-shard 레지스트리는 개별 x-shard를 관리하는 컨트랙트와 그들의 샤드 간 통신 링크를 유지한다. x-shard 컨트랙트는 해당 x-shard의 제네시스 해시와 합의 매개변수를 저장하고, 검증자와 활성 검증자 집합을 관리하며, 다른 x-shard에서 오는 샤드 간 통신 링크를 승인한다. 등록된 모든 x-shard는 ZQ2 메인넷과 통신 링크를 가진다.

프라이버시 옵션을 사용하는 x-shard의 경우, 모든 트랜잭션은 해당 x-shard 구성원 간에만 공유되는 대칭 키로 AES256 암호화된다. 라이트 클라이언트가 이를 검증할 수 있도록 비공개 x-shard의 블록 헤더만 암호화되지 않은 상태로 남겨진다. 새로운 노드가 비공개 x-shard에 합류할 때, 그들은 ZQ2 메인넷에 저장된 샤드 컨트랙트에서 검증자들에게 암호화 키의 이력을 요청한다. 새로운 노드가 제네시스 블록부터 모든 트랜잭션을 해독하고 재실행할 수 있도록, 과거 키를 공유하는 것이 필요하다. 키를 공유하기 전에 검증자는 새로운 노드가 실제로 해당 x-shard의 유효한 구성원인지 확인한다. 이 검증은 비공개 x-shard에 배포된 스마트 컨트랙트를 통해 수행되며, 기존 구성원 노드만 호출할 수 있다. 노드가 비공개 x-shard에서 떠날 때, 남아 있는 구성원들은 떠난 노드가 향후 트랜잭션을 해독하지 못하도록 새로운 암호화 키에 합의해야 한다. 새로운 키는 떠나는 노드가 실제로 x-shard에서 제외되기 전에 충분히 남은 노드가 새 키를 받을 수 있도록, x-shard 매개변수에 지정된 블록 수 전에 리더에 의해 생성된다. 리더는 Diffie-Hellman 키 교환 [17]을 사용하여 새 키를 공유하며, p2p 네트워크를 통해 해당 메시지를 전송한다.

비공개 x-shard의 RPC 노드는 요청을 인증하여 권한 없는 당사자에게 개인 정보가 유출되는 것을 방지해야 한다. 이는 다른 x-shard의 검증자들이 사용하는 라이트 클라이언트에도 적용되며, 이들은 샤드 간 호출을 나타내는 이벤트에 대해 반드시 통지받아야 한다.

B. 샤드 간 통신(Intershards Communication)

ZQ1의 샤드와 달리, x-shard는 더 느슨하게 결합되어 있다. 각 x-shard는 자체 상태와 트랜잭션을 가지며, 동일한 x-shard에 배포된 스마트 컨트랙트를 호출할 수 있다. 또한 네이티브 샤드 간 메시징 프로토콜을 사용하여 다른 x-shard에 배포된 컨트랙트와 상호작용할 수 있다.

로컬 x-shard의 모든 검증자는 해당 로컬 x-shard가 연결된 원격 x-shard의 라이트 클라이언트 역할을 한다. 그들은 원격 x-shard의 합의 규칙(지분 증명 또는 권한 증명)과 검증자 집합을 알고 있기 때문에, 원격 x-shard 블록이 최종 확정되는 시점을 판별할 수 있다. 원격 x-shard에서 최종 확정된 각 블록에서, 검증자들은 원격 x-shard에서 발생한 샤드 간 트랜잭션을 나타내는 이벤트 로그를 검색한다. 라이트 클라이언트는 원격 x-shard의 블록 헤더만 알 수 있으므로, 원격 x-shard의 전체 노드에서 받은 로그를 블록 헤더에 포함된 영수증 루트에 대해 검증해야 한다. 원격 x-shard에서 발생한 샤드 간 통신 이벤트는 샤드 간 호출을 요청한 송신자 컨트랙트 주소, 해당 호출이 전송되어야 하는 로컬 x-shard의 대상 컨트랙트 주소, 그리고 호출 데이터, 가스 한도, 가스 가격을 포함한다.

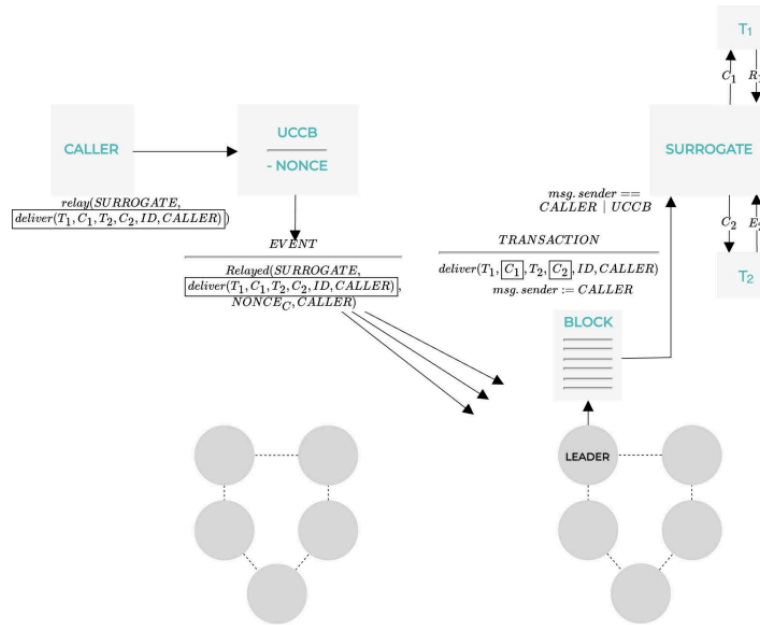


Fig. 4: The **CALLER** contract requests an intershard call to the **SURROGATE** contract's `deliver()` function with the target addresses **T₁** and **T₂** and call data **C₁** and **C₂** to be used by the **SURROGATE** as arguments. The target x-shard's consensus **LEADER** includes a transaction in its proposed block that calls the **SURROGATE** contract's `deliver()` function.

그림 4에서 보듯이, 로컬 x-shard의 리더가 새로운 블록을 제안할 때, 원격 x-shard의 최종 확정된 블록에서 가져온 이벤트 로그로부터 구성된 샤드 간 트랜잭션을 포함한다. 블록은 로컬 x-shard의 검증자들에게 전파되며, 이들은 블록에 포함된 샤드 간 트랜잭션을 승인하는 합의 투표를 전송하고, 블록 내의 일반 트랜잭션과 함께 이를 실행한다. 샤드 간 트랜잭션은 원격 x-shard에서 발생한 이벤트에 지정된 호출 데이터, 가스 한도, 가스 가격을 사용하여 대상 컨트랙트를 호출한다.

호출의 메시지 송신자(sender)는 샤드 간 호출을 요청한 원격 x-shard에 배포된 송신자 컨트랙트의 주소로 설정된다. 가스 수수료는 로컬 x-shard의 네이티브 토큰으로 된 송신자의 잔액에서 차감된다. 로컬 x-shard의 대상 컨트랙트 다시 원격 x-shard의 송신자 컨트랙트로 샤드 간 호출을 요청하여, 로컬 x-shard에서 실행된 샤드 간 트랜잭션 결과를 통지할 수 있다.

C. 크로스체인 통신(Cross-Chain Communication)

다른 EVM 체인과의 상호운용성은 ZQ2 플랫폼의 또 다른 핵심 속성이다. 토큰 브릿지(token bridge)나 아토피믹 스왑(atomic swap)과 같은 크로스체인 애플리케이션은 ZQ2 프로토콜에 통합된 UCCB(Universal Cross-Chain Broker)에 의해 지원된다. 이는 서로 다른 EVM 체인에 배포된 스마트 컨트랙트가 비동기 메시지를 주고받을 수 있도록 하여 상호작용을 가능하게 한다. 크로스체인 메시지와 그 결과는 검증자 절대 다수의 서명된 확인(acknowledgement)을 수집한 후, ZQ2 검증자에 의해 출발지 체인과 목적지 체인 사이에서 중계된다. 목적지 체인을 지정하기 위해 전역적으로 고유한 체인 ID를 사용함으로써, 스마트 컨트랙트는 UCCB 네트워크에 속한 어느 체인에서든 서로 호출할 수 있다.

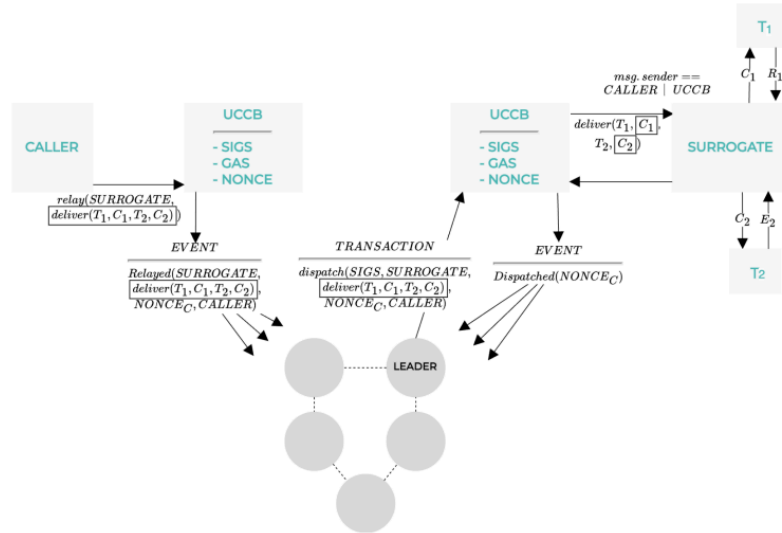


Fig. 5: The **CALLER** contract requests a cross-chain call to the **SURROGATE** contract's **deliver()** function with the target addresses T_1 and T_2 and call data C_1 and C_2 to be used by the **SURROGATE** as arguments. The validators collect the supermajority's signatures **SIGS** and the consensus **LEADER** submits a transaction to the **UCCB** contract, passing the **SIGS** and the requested call to its **dispatch()** function.

컨트랙트가 크로스체인 메시지를 전송하면, 출발지 체인에서 UCCB 이벤트가 발생한다. 이벤트에는 출발지 체인 및 도착지 체인 상 컨트랙트 주소, 호출 데이터(call data), 출발지 및 도착지 체인 ID가 인코딩된다. 검증자는 출발지 체인에서 최종 확정된 블록을 추적하며, 블록에서 발생한 모든 UCCB 이벤트를 가져온다. 검증자들은 이 이벤트들로부터 머클 트리를 구성하고, 그 루트 해시에 서명하며, 해당 이벤트를 승인하기 위해 ZQ2 메인넷의 p2p 네트워크에 자신의 서명을 전파한다. 현재 리더가 검증자 절대 다수의 서명을 확인하면, 머클 트리의 리프에 인코딩된 크로스체인 메시지를 전송하기 시작한다. 리더는 목적지 체인에 트랜잭션을 제출하여, 인코딩된 크로스체인 메시지와 함께 해당 머클 증명 및 검증자 절대 다수가 서명한 머클 루트를 인자로 포함시켜 UCCB 컨트랙트의 **dispatch** 함수를 호출한다. 리더는 목적지 체인의 가스 한도에 따라 가능한 한 많은 크로스체인 메시지를 전송한다. 리더 임기 동안 모든 크로스체인 메시지를 전송하지 못한 경우, 다음 리더가 대기 중인 메시지의 전송을 이어간다.

UCCB 컨트랙트는 서명을 검증하고, 인코딩된 대상 주소와 호출 데이터를 출발지 체인 상 컨트랙트에서 지정한 대리 컨트랙트(surrogate contract)에 전달한다. UCCB 컨트랙트는 대리 컨트랙트를 대신하여 예치금(deposit)을 관리하며, 예치금에 따라 대리 컨트랙트가 사용할 수 있는 가스를 제한한다. 또한 대리 컨트랙트의 가스 사용량을 측정하고, 리더의 트랜잭션 비용을 해당 예치금에서 환불한다. 나아가, 크로스체인 메시지에 서명한 검증자들에게 보상을 제공하기 위해 대리 컨트랙트의 예치금에서 추가 수수료를 공제한다. 또한, UCCB 컨트랙트는 크로스체인 메시지에 포함된 UCCB 논스 nonce를 관리하여 재실행 공격(replay attack)을 방지한다.

대리 컨트랙트는 대상 주소와 호출 데이터를 디코딩한 뒤 요청된 호출을 수행한다. 크로스체인 메시지에 여러 호출이 인코딩되어 있으면, 대리 컨트랙트는 이를 지정된 순서대로 실행한다. 대리 컨트랙트는 실행 중 발생한 성공 플래그(success flag), 반환값 또는 오류를 수집하여 인코딩한 뒤, UCCB 컨트랙트에 다시 크로스체인 메시지를 출발지 체인 상 컨트랙트로 전송할 것을 요청하여 그 결과를 알릴 수 있다. 크로스체인 메시지에는 애플리케이션 특화 고유 식별자(application-specific unique identifier)를 포함할 수 있으므로, 출발지 체인 상 컨트랙트는 수신한 결과를 자신이 보낸 크로스체인 메시지와 쉽게 매칭할 수 있다.

D. 공통점과 응용(Commonalities and Applications)

샤드 간 혹은 크로스체인 트랜잭션을 지원하는 검증자들은 연결된 체인과 x-shard의 라이트 클라이언트를 실행하여, 각 최종 확정된 블록에서 발생한 UCCB 이벤트 로그를 가져온다. 이들은 각 블록 헤더의 블록 필터를 확인하고, 필터가 긍정 결과를 나타낼 때만 RPC 노드에 영수증(receipt)을 요청한다. RPC 노드가 반환하는 모든 로그는 동일한 블록 번호를 가지므로, 검증자들은 단일 **eth_getBlockReceipts** RPC 호출을 사용하여 해당 블록의 모든 영수증을 가져오고, 알고리즘 2에서 제시된 것처럼 그 중 어떤 것이 UCCB 이벤트 로그를 포함하는지 로컬에서 확인할 수 있다.

프로그래머 인터페이스, 즉 스마트 컨트랙트가 원격 호출을 요청하기 위해 UCCB에 해당 이벤트를 발생시키도록 요구하는 방식은 샤드 간 트랜잭션과 크로스체인 트랜잭션 모두 동일하다. 주요 차이는 요청된 원격 호출을 중계하는 방식에 있다.

샤드 간 통신은 ZQ2 검증자 프로토콜에 네이티브로 통합되어 있어, 관련 x-shard의 합의를 활용하여 샤드 간 트랜잭션에 대한 절대 다수의 승인을 얻는다. 반면, 크로스체인 통신은 ZQ2 검증자가 절대 다수의 서명을 수집해야 하며, 목적지 체인에 배포된 UCCB 컨트랙트가 서명을 검증한 후 대리 컨트랙트(surrogate contract)를 호출해야 한다. 이로 인해 크로스체인 트랜잭션은 추가 가스 오버헤드가 발생하지 않는 샤드 간 트랜잭션보다 비용이 더 크다.

알고리즘 2: UCCB event log verification

```
# input: blockNum, logs

1 rcpts = rpc.eth_getBlockReceipts(blockNum)
2 # check validity of receipts
3 if block[blockNum].receiptRoot
   != merkle_tree(rcpts).root:
4   raise RootMismatch
5
6 # verify if logs included in receipts
7 if any(log for log in logs if log not
   in rcpts[log.transactionIndex].logs):
8   raise InvalidLogs
9
10 # search for withheld logs
11 moreLogs = [l for l in sum([r.logs for
   r in rcpts], []) if l not in logs]
12 if any(l.address in filter.address
   and filter.topics.match(l.topics)
   for l in moreLogs):
13   raise MissingLogs
```

UCCB와 샤드 간 통신 모두 출발지 체인에서 최종성(finality)에 도달하는 시간으로 인해 최소한의 지연이 발생한다. 두 방식의 신뢰성은 ZQ2 합의의 보안성에 기반한다. 합의가 안전성(safety)과 생존성(liveness)을 보장하는 한, ZQ2 메인넷 검증자를 사용하는 크로스체인 및 샤드 간 트랜잭션 역시 동일한 보장을 받는다.

ZQ2의 크로스체인 및 샤드 간 통신은 ERC-20 토큰과 NFT를 지원하는 정규 브릿지(canonical bridge)의 기반을 제공한다. 주목할 점은 ZRC-2 토큰과 ZRC-6 NFT 또한 ERC-20 및 ERC-721 컨트랙트로 래핑(wrapping)할 수 있기 때문에 브릿징이 가능하다는 것이다. 우리는 ZQ2와 다른 EVM 체인 간의 ERC-721 브릿지를 예시로 들어 구현이 얼마나 단순한지를 설명한다. 브릿지는 ZQ2에 배포된 에스크로(escrow) 컨트랙트와 다른 체인에 배포된 대리 컨트랙트로 구성된다. 에스크로 컨트랙트는 **ERC721Receiver** 인터페이스를 구현한다. 토큰 ID가 이 컨트랙트로 전송되면, 그 **onERC721Received** 함수가 다른 체인의 대리 컨트랙트에 원격 호출을 수행하여 동일한 토큰 ID로 래핑된 NFT를 발행하도록 요청한다. 이후, 래핑된 NFT가 다른 체인의 대리 컨트랙트로 전송되면, 그 **onERC721Received** 함수가 이를 소각(burn)하고, ZQ2의 에스크로 컨트랙트에 원래 NFT를 래핑된 NFT를 반환한 주소로 전송할 것을 요청한다.

VII. 스마트 컨트랙트 계층(Smart Contract Layer)

ZQ2는 러스트(Rust) 기반의 EVM 구현체를 통합하여, 이더리움의 하드 포크를 거치며 추가된 프리컴파일(precompile)뿐 아니라 ZQ2에만 독점적으로 제공되는 다른 프리컴파일도 지원한다.

EVM 호환 스마트 컨트랙트 개발 언어, 프레임워크, 도구 외에도 ZQ2는 Scilla를 지원한다. 초기에는 ZQ1과 마찬가지로 인터프리터를 통해 실행되며, 이후에는 Scilla 소스 코드를 EVM 바이트코드로 컴파일하는 방식으로 발전한다. ZQ2는 네이티브 Scilla와 EVM 컨트랙트 간의 완전한 상호운용성을 제공하여, ZRC-2 대체 가능 토큰이 ERC-20 프록시를 가질 수 있으며 그 반대도 가능하게 한다. 또한, ZQ2는 ZQ1의 전체 상태를 가져오므로, Scilla 컨트랙트를 포함한 ZQ1 계정은 ZQ1에서와 동일한 주소에 ZQ2에서도 저장된다. 지갑 및 기타 도구와의 하위 호환성을 유지하고, 가져온 Scilla 컨트랙트와 원활한 상호작용을 보장하기 위해 ZQ2는 ZQ1에서 제공하던 API도 지원한다.

더 나아가, ZQ2는 스마트 컨트랙트뿐만 아니라 사용자 차원에서도 Scilla와 EVM 간의 상호운용성을 개선한다. ZQ1 사용자들은 Scilla 트랜잭션과 EVM 트랜잭션을 서명하기 위해 두 개의 별도 지갑이 필요했으나, ZQ2의 계정 추상화 덕분에 이제는 단일 계정을 Scilla 지갑과 EVM 지갑 모두로 제어할 수 있다.

A. 즉각적 샤드 간 상태 읽기(Instant Intershard State Reads)

두 x-shard 간에 샤드 간 통신이 설정되면, 한 x-shard의 스마트 컨트랙트가 다른 x-shard의 스마트 컨트랙트 상태(스토리지 값 포함)를 읽고 이를 트랜잭션을 실행할 때 즉시 사용할 수 있다. 이를 위해 스마트 컨트랙트는 프리컴파일을 호출하여 다른 x-shard의 주소와 가져오려는 스토리지 슬롯을 지정해야 한다. 또한, 특정 블록 높이의 값을 원할 경우 다른 x-shard의 블록 번호를 지정할 수도 있다.

그림 6에서 보듯이, 송신자의 노드는 트랜잭션 실행을 시뮬레이션하여 프리컴파일의 주소와 스토리지 슬롯을 확인해야 한다. 송신자는 다른 x-shard의 RPC 노드에 **eth_getProof**를 사용해 지정된 스토리지 슬롯 값과 해당 머클 증명을 요청한다. 이때, 프리컴파일에 전달된 블록 번호나 해당 x-shard에서 가장 최근에 최종 확정된 블록 번호를 사용한다. 마지막으로, 송신자는 가져온 값과 머클 증명, 그리고 블록 번호를 트랜잭션의 추가 데이터로 포함시킨다.

블록 제안자가 자신의 블록에 이 트랜잭션을 포함시키면, 그는 다른 x-shard의 라이트 클라이언트를 실행하므로, 트랜잭션에 포함된 값과 머클 증명이 지정된 블록 번호에 대해 유효한지 확인한다. 검증자들이 블록을 수신하면 동일한 검증을 수행하며, 값이나 머클 증명 중 하나라도 유효하지 않으면 해당 블록을 거부한다.

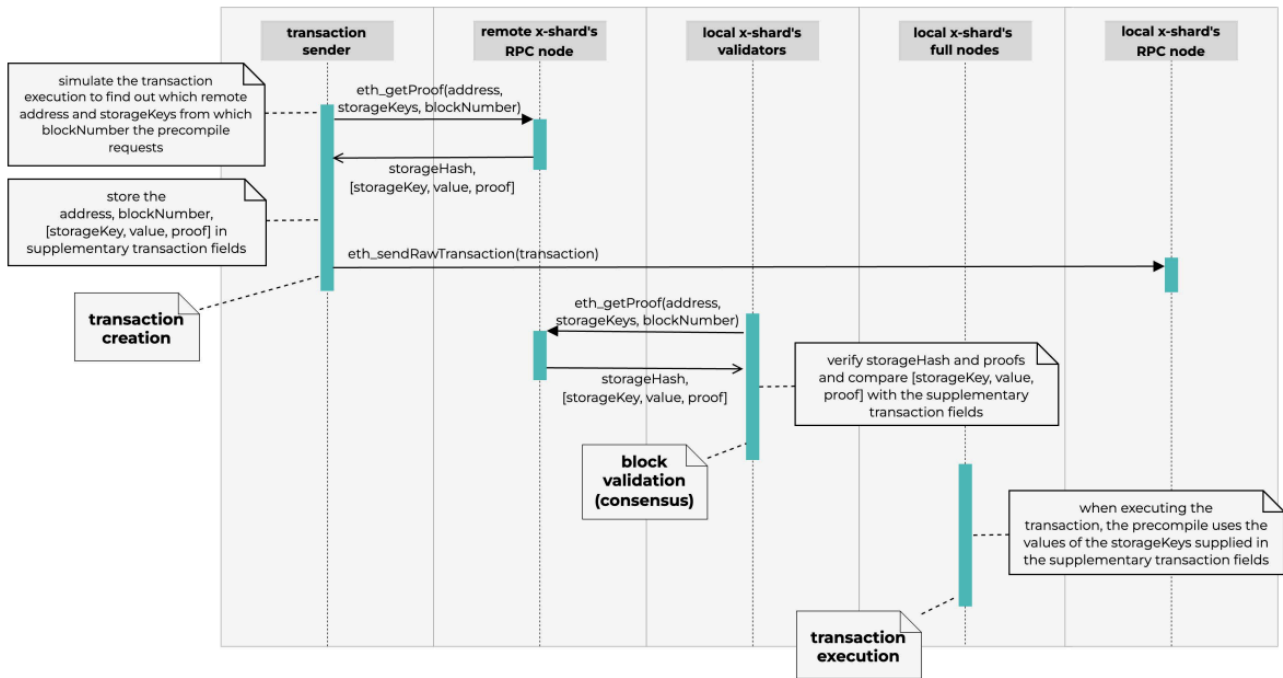


Fig. 6: Handling of intershard state reads by the transaction sender, the x-shard's validators proposing and validating the block of the transaction and the full nodes executing the transaction

비검증 노드가 블록의 일부로 트랜잭션을 실행할 때, 프리컴파일은 트랜잭션에 저장된 값을 사용한다. 이 노드는 값과 머클 증명을 직접 검증할 필요가 없으며, 이미 이를 검증한 검증자 절대 다수의 결과를 신뢰할 수 있다.

즉각적 샤드 간 상태 읽기는 다른 x-shard에 저장된 토큰 잔액을 기반으로 한 투표와 같은 사용 사례를 구현하는 우아한 방식을 제공한다. 투표 컨트랙트가 투표를 받을 때, 단순히 다른 x-shard의 고정된 “스냅샷” 블록 높이에서 유권자의 토큰 잔액을 조회하면 된다. 즉각적 샤드 간 상태 읽기의 또 다른 사용 사례는 다른 x-shard의 오라클이 제공하는 가격 피드이다. 컨트랙트는 해당 x-shard에 저장된 가격을 가장 최근 최종 확정된 블록 높이에서 관련 스토리지 슬롯을 조회하여 접근할 수 있다.

B. 예약 컨트랙트 연속 실행(Scheduled Contract Continuations)

임계값에 아직 도달하지 않았거나, 오라클에 의해 데이터가 아직 제공되지 않았거나, 다른 조건들이 충족되지 않은 상황에서는 스마트 컨트랙트가 실행해야 할 모든 단계를 완료하지 못할 수 있다.

이 문제를 해결하기 위해서는 트랜잭션 로직을 여러 단계로 나누고, 사용자가 장애물이 제거되는 시점을 직접 추적하여 다음 단계를 진행하는 별도의 트랜잭션을 제출해야 한다.

예약 컨트랙트 연속 실행은 스마트 컨트랙트가 지정된 조건이 충족될 때 자동으로 호출되어, 이전에 제출된 사용자 트랜잭션의 대기 중인 단계를 다시 실행(resume)할 수 있도록 하는 개념이다.

어떤 컨트랙트든 프리컴파일을 호출하여 트리거 조건의 인코딩된 호출 데이터와 연속 실행의 호출 데이터, 최소값 및 최대값 블록 높이, 그리고 연속 실행을 가능하게 하는 검증자에게 지급될 보상을 전달함으로써 재실행을 요청할 수 있다. 최소값과 최대값 블록 높이 사이의 블록 제안자는 트리거 조건을 평가하여 재실행 트랜잭션을 블록에 포함시키려 시도할 수 있다. 성공하면 보상을 얻고, 실패하면 다음 블록 제안자가 이를 시도할 기회를 갖는다. 블록 제안자가 트리거 조건이 false를 반환했음에도 재실행 트랜잭션을 블록에 포함시키면, 정직한 검증자들이 해당 블록을 거부한다. 트리거 조건이 끝내 true를 반환하지 않으면, 프리컴파일을 호출하는 트랜잭션을 제출한 사용자가 페널티를 받으며, 이는 프리컴파일에 의해 부과된 가스로 지불되어 보상에 사용된다.

VIII. 인센티브 및 거버넌스 계층(Incentive and Governance Layer)

A. 토큰 공급(Token Supply)

ZQ2 메인넷은 네이티브 통화로 ZIL을 사용한다. 수수료와 보상은 ZIL로 지급되며, 메인넷 검증자와 그 위임자(delegate)는 네트워크를 보호하기 위해 ZIL을 스테이킹한다. ZIL은 ERC-20 인터페이스로 네이티브하게 래핑되어, 디파이(DeFi) 애플리케이션에서 대체 가능 자산으로 일관되게 취급될 수 있다.

x-shard는 검증자 보상을 분배하고 소각된 트랜잭션 수수료를 수집하는 계정에 제네시스 공급량을 할당하는 방식으로 부트스트랩된다. 이들은 네이티브 토큰 선택에 있어 여러 옵션을 가진다. 네이티브 토큰과 제네시스 공급량은 메인넷에서 브릿징된 ZIL일 수 있다. 또는 메인넷에 이미 배포되어 새로운 x-shard로 브릿징된 다른 ERC-20 토큰의 일정량일 수도 있다. 여기에는 스테이블코인도 포함되며, 이를 네이티브 토큰으로 사용하면 트랜잭션 수수료와 검증자 보상이 가격 변동에 영향을 받지 않도록 할 수 있다. 또 다른 옵션은, x-shard를 완전히 새로운 네이티브 토큰으로 시작하는 것으로, 이 경우 해당 토큰은 초기에는 오직 x-shard에만 존재한다.

B. 스테이킹 보상(Staking Rewards)

ZQ2의 지분 증명 합의에는 ZIL 스테이킹이 필요하다. 사용자는 스테이킹 컨트랙트에 ZIL을 예치하고 이를 검증자에게 위임할 수 있다. 사용자는 검증자의 수수료가 차감된 보상의 몫을 즉시 청구할 수 있지만, 위임된 지분을 인출하거나 다른 검증자에게 이전하는 것은 ZQ1에서 이어진 언스테이킹/언본딩(unbonding) 기간이 지나야 가능하다. 대부분의 검증자는 스테이킹 풀에 의해 운영되지만, ZQ2는 위임자가 없는 단독 스테이커(solo staker)도 허용한다.

x-shard는 자체 검증자 인센티브를 자유롭게 사용할 수 있지만, 지분 증명 메인넷의 토크노믹스는 프로토콜에 내장되어 있다. 그 목표는 스테이킹 수익을 얻을 수 있는 지속 가능하면서도 매력적인 보상 모델을 확립하는 동시에, 스테이킹 비율을 희생하지 않고 네트워크 보안에 필수적인 ZIL 스테이킹 비율을 유지하면서 한정된 ZIL 공급량의 고갈을 방지하기 위해 인플레이션을 줄이는 것이다.

이를 달성하기 위해 검증자 보상은 블록 공간의 활용도에 따라 정기적으로 조정된다.

모든 검증자가 함께 얻는 보상의 총액을 계산하는 공식은 그림 7에 정의되어 있으며, 여기서 $\$c\{S\}$, $\$m\{S\}$, $\$x\{S\}$ 는 각각 현재, 최대, 목표 ZIL 공급량이고, $\$c\{R\}$ 과 $\$x\{R\}$ 은 각각 현재 및 목표 스테이킹 비율, $\$c\{F\}$ 는 소각된 수수료 양이다. 검증자는 기대된 블록 수를 생성할 경우 매시간 이 금액을 얻으며, 이후 자신의 보상 몫을 위임자의 스테이킹 ZIL 비율에 따라 분배한다.

$$Z_h = \left(Z \cdot \left(\frac{1 - \frac{c_s}{m_s}}{1 - \frac{x_s}{m_s}} \right)^T + c_F \right) \cdot \left(c_R \cdot B^{\frac{x_R - c_R}{x_R}} \right)$$

Fig. 7: Total staking rewards the validators are expected to earn per hour. The right hand side of the product calculates an amount based on the transaction fees and the depletion level of the ZIL reserve, and the left hand side adjusts it to encourage staking up to the target staking ratio.

매개변수 $\$Z\$, \$B\$, \$T\$$ 는 다음과 같이 정의된다. $\$Z\$$ 는 총 보상 계산의 기준이 되는 ZIL 토큰 양이다. 초기 값은 ZQ1에서 ZQ2로의 스테이킹 보상 전환이 원활하게 이루어지도록 선택되었다. $\$B\$$ 와 $\$T\$$ 는 두 개의 양의 상수이다. 이들의 효과를 이해하려면, 그림 7의 곱(product)의 양쪽 항을 먼저 설명해야 한다.

좌변은 최대 공급량이 전부 스테이킹되었을 경우 받을 총 보상액이다. 이 가상의 경우, 각 ZIL 토큰은 좌변의 $\$1 / 210억\$$ 을 받는다. 이 금액에는 소각된 트랜잭션 수수료와, 최대 공급량이 얼마나 고갈되었는지에 따라 달라지는 $\$Z\$$ 의 일부가 포함된다. 공급량이 목표를 초과하면, $\$Z\$$ 의 비율은 빠르게 줄어들며 최대 공급량에 가까워질수록 0에 수렴한다. $\$T\$$ 값이 클수록, 총 보상에 포함되는 $\$Z\$$ 의 비율이 더 많이 제한되어, 최대 공급량의 완전한 고갈을 막는 긴급 제동 장치 역할을 한다.

우변은 좌변의 금액을 현재 스테이킹 비율에 맞춰 조정하는 함수이다. 스테이킹 비율이 높을수록, 즉 더 많은 ZIL이 스테이킹될수록 총 보상이 지분에 의해 나누어지기 때문에 각 ZIL이 받는 보상은 줄어든다. 현재 스테이킹 비율이 목표보다 낮을 경우, $\$B > 1\$$ 을 사용하는 공식은 좌변의 금액을 지분에 비례하지 않게 증가시켜 더 많은 스테이킹을 장려한다. 스테이킹 비율이 목표에 가까워질수록 증가폭은 줄어들며, 이는 각 스테이킹된 ZIL의 보상이 점점 줄어드는 것을 의미한다. 스테이킹 비율이 목표를 초과하면 공식은 반대 효과를 발휘하여, 각 스테이킹된 ZIL의 보상은 비율이 더 높아질수록 과도하게 감소해 추가 스테이킹을 억제한다. $\$B\$$ 값이 클수록 ZIL 보유자에게는 목표 비율까지 스테이킹을 할 더 강한 유인이 생기지만, 목표를 넘어서는 것은 억제된다. 목표 도달 전 스테이커에게 더 높은 $\$B\$$ 가 주는 이익은 ZIL 준비금의 더 빠른 소진이라는 비용을 수반한다. 신중히 선택된 $\$T\$$ 값은 공급량이 최대에 너무 가까워질 경우 소진 속도를 늦추는 반대 효과를 제공한다.

그림 7에 제시된 공식은 여러 시뮬레이션에서 스트레스 테스트를 거쳤다. 이 공식은 검증자 보상과 가스 가격을 매월 조정하는 기준으로 사용된다. 시스템의 장기적인 지속 가능성을 위해 필요하다면, 질리카는 조정을 제안하며 최종 결정은 탈중앙화 거버넌스에 의해 내려진다.

메인넷은 블록이 생성되는 데 걸리는 시간과 관계없이 매 블록마다 검증자 보상을 발행한다. 블록 생성 속도가 예상보다 느려지면, 검증자가 아래와 같이 정의된 보상을 얻는 데 더 오랜 시간이 걸린다.

$$Z_b = \frac{Z_h}{E_h \cdot B_e}$$

- Z_h 는 그림 7의 총 보상 금액,
- E_h 는 시간당 예상 에포크 수,
- B_e 는 에포크 길이, 즉 에포크당 블록 수를 나타낸다.

보상은 다음과 같이 검증자에게 배분된다:

- **제안자 보상(Proposer rewards):** 50%는 블록 제안자에게 지급된다.
- **공동서명자 보상(Cosigner rewards):** 50%는 블록에 포함된 QC 또는 집계 QC에 기여한 검증자들에게 분배된다.

공동서명자 보상은 검증자의 지분 비율에 따라 배분된다. 즉, 두 명의 검증자가 QC 또는 집계 QC에 서명을 기여했을 때, 한 검증자가 다른 검증자보다 두 배의 지분을 보유하고 있다면, 더 많은 지분을 가진 검증자는 다른 검증자의 두 배에 해당하는 공동서명자 보상 몫을 받는다. 반면 제안자 보상은 블록을 제안한 모든 검증자에게 동일하게 지급되는데, 이는 리더로 선택되는 빈도 자체가 이미 검증자의 지분에 의해 가중되기 때문이다.

QC나 집계 QC에 포함되기 위해 자신의 투표 또는 new-view 메시지를 충분히 빠르게 제출하지 못한 검증자라도, 자신이 리더인 뷰에서 유효한 블록을 제안하고 해당 블록이 최종 확정되면 최대 검증자 보상의 50%까지 얻을 수 있는 기회를 가진다.

C. 검증자 페널티(Validator Penalties)

잘못된 행동을 한 검증자는 다음과 같이 처벌된다.

- 안전성을 훼손하면 삭감(slashed)
- 생존성을 저해하면 정지(suspended)

활성 검증자 집합을 관리하는 시스템 컨트랙트는 블록 제안자로서 검증자의 성능을 추적한다. 현재 검증자 집합을 반환하라는 요청을 받으면, 해당 컨트랙트는 프리컴파일을 호출하여, 검증자가 리더였음에도 불구하고 체인에 블록이 없는 뷰를 기준으로 검증자의 점수를 계산한다. 이 점수는 검증자가 정지될지 여부와 정지되는 블록 수를 결정한다. 점수는 다음 공식으로 계산된다:

$$s(i) = \frac{\sum_{v=v_p-m}^{v_p} (v - v_p + m + 1) \cdot l(i, v) \cdot f(v) \cdot \max(1 - f(v + 1), l(i, v + 1))}{m}$$

여기서

- i 는 검증자 집합에서 검증자의 인덱스,
- v_p 는 로컬 뷰 번호에서 1을 뺀 값,
- m 은 v_p 이전에 고려할 뷰의 수,
- $l(i, v)$ 는 인덱스 i 의 검증자가 뷰 v 의 리더였다면 1, 그렇지 않으면 0,
- $f(v)$ 는 체인에 뷰 번호 v 의 블록이 있으면 0, 없으면 1.

$s(i)$ 가 설정된 임계값을 초과하면, 인덱스 i 의 검증자는 $s(i) \cdot B_e$ 를 반올림한 블록 수만큼 정지된다. 여기서 B_e 는 블록 단위의 에포크 길이이다. 정지된 검증자는 블록 제안에 대한 투표나 new-view 메시지를 다음 리더에게 보낼 수는 있지만, 블록을 제안할 수 없으므로 제안자 보상을 포기해야 한다.

검증자는 단일 블록에 대해서는 정지되지 않으며, 체인에 포함되는 블록을 반복적으로 생성하지 못할 때만 처벌을 받는다. 공식은 또한 단순히 다음 리더가 실패했기 때문에 검증자가 처벌받는 일을 방지한다. 이전 뷰에서 누락된 블록도 현재 점수 계산에 반영되지만, 최신 블록보다 가중치가 낮다. 검증자가 총 지분에서 차지하는 비중이 클수록 더 많은 뷰를 책임지게 되며, 노드가 중단될 경우 더 빨리 정지된다. 이는 스테이킹 풀의 중앙화를 억제하기 위함이다.

만약 검증자가 리더가 동일한 뷰에서 여러 개의 다른 블록을 제안하는 이중 서명을 발견하거나, 리더가 검증자가 동일한 뷰에서 여러 개의 다른 투표 또는 new-view 메시지를 보낸 것을 발견하면, 이들은 증거를 모든 검증자와 공유하고, 리더는 해당 증거를 온체인에 기록하기 위해 삭감 트랜잭션을 포함시킨다. 삭감 트랜잭션은 삭감 컨트랙트에 제출되며, 이 컨트랙트는 프리컴파일을 호출해 증거를 검증한다. 증거가 해당 검증자의 잘못을 입증하면, 삭감 컨트랙트는 활성 검증자 집합을 관리하는 컨트랙트를 호출하여 검증자를 제거한다. 또한 삭감 컨트랙트는 스테이킹 컨트랙트를 호출하여 검증자의 지분에서 삭감된 부분을 몰수하고, 삭감 보상을 차감해 삭감 트랜잭션의 발신자(일반적으로 블록 제안자)에게 지급한다. 삭감은 검증자뿐 아니라 해당 검증자에게 위임한 모든 계정도 적용되며, 언스테이킹을 요청한 계정도 포함된다. 삭감되는 지분의 비율은 최근 에포크에서 누적된 삭감 양에 따라 증가한다:

$$\max \left(\frac{9}{100}, \min \left(1, \left(\frac{3 \cdot \sum_{i \in S} st(i)}{\sum_{i \in A} st(i)} \right)^2 \right) \right)$$

여기서 A 는 활성 검증자 집합이고, $S \subset A$ 는 삭감된 검증자 집합이며, $st(i)$ 는 검증자 i 의 지분을 나타낸다. 검증자가 활성 검증자 집합의 총 지분에서 10%를 초과하는 지분을 통제할수록, 같은 에포크에서 다른 검증자가 삭감되지 않더라도 더 큰 비율이 삭감된다. 삭감이 추가될 때마다, 비율은 갱신되어 해당 에포크 시작 이후 삭감된 모든 검증자에게 적용된다.

만약 프리컴파일이 검증한 증거가 잘못된 것으로 판명되면, 삭감 트랜잭션은 취소되지만, 프리컴파일은 잘못된 삭감 트랜잭션 발신자를 처벌하기 위해 매우 높은 가스를 부과한다.

삭감 컨트랙트는 어떤 검증자가 삭감되었는지를 나타내는 이벤트를 발생시킨다. 노드가 삭감 이벤트를 포함하는 블록을 수신하면, 에포크 시작 시점에 가져온 활성 검증자 집합에서 해당 검증자를 제거한다. 이는 라이트 클라이언트에도 적용되며, 이들은 블록 헤더의 블룸 필터를 사용해 블록에 삭감 이벤트가 포함되었는지 확인한다.

정지되거나 삭감된 검증자가 블록 제안자가 될 차례인 뷰의 경우, 리더 선택 알고리즘은 에포크 시작 시점에 초기화된 난수 생성기를 사용해 다른 검증자를 결정적으로 선택한다.

D. 거버넌스 모델(Governance Model)

ZQ2 메인넷은 거버넌스 토큰 gZIL과 ZQ1에서 확립된 거버넌스 프레임워크를 사용한다. 거버넌스는 매우 탈중앙화되어 있으며, 8만6천 명이 넘는 gZIL 토큰 보유자가 참여한다.

거버넌스 규칙에 의해 정해진 최소 잔액 이상을 보유한 토큰 보유자는 제안을 제출할 수 있다. 제안을 공식 투표에 제출하기 전에, 반드시 Discourse [18]의 거버넌스 포럼에 게시하여 커뮤니티가 논의하고, 피드백을 제공하며, 수정안을 제안할 수 있도록 해야 한다. 최종 버전의 제안은 최소 3일 동안 열려 있는 투표에서 찬성표가 25%를 초과해야 공식 투표 자격을 얻는다.

공식 투표는 Snapshot [19]에서 진행된다. 최소 7일 동안 열려 있어야 하며, 직전 3번의 투표 참여율에 따라 결정된 정족수를 충족하고, 50%를 초과하는 찬성을 얻어야 통과된다. 모든 gZIL 보유자는 참여할 수 있으며, 그들의 투표는 공식 투표가 생성될 때 지정된 블록 높이에서의 gZIL 잔액에 따라 가중된다. 그 블록 높이 이전에는 gZIL 보유자가 자신의 투표권 일부 또는 전부를 하나 이상의 대리인에게 위임할 수 있다.

하드 포크를 필요로 하거나 현재 프로토콜과 호환되지 않는 수정을 제안하는 경우, 제안은 Github [20]에 게시되는 질리카 개선 제안(ZIP, Zilliqa Improvement Proposal)의 형태로 추가 명시된다. x-shard는 자체 거버넌스 토큰을 사용하거나, gZIL을 채택하되 다른 인증 메커니즘과 결합하여 투표 자격 그룹을 제한할 수 있다. 투표권을 gZIL 보유량에 따라 결정하면, 투표자가 gZIL을 축적하여 자신이 속한 x-shard의 의사결정에서 더 큰 영향력을 행사하고 동시에 메인넷 거버넌스에도 참여할 수 있도록 장려한다. 특히 x-shard가 네이티브 토큰으로 ZIL을 사용할 경우, 메인넷 거버넌스가 ZIL 인플레이션 및 ZIL 가격에 직접적으로 영향을 미치는 기타 결정을 통제하기 때문에 이는 합리적이다.

IX. 향후 연구(Future Research)

ZQ2 출시 이후, 연구 및 개발은 다음 영역에 중점을 두고 지속될 것이다.

A. 포스트 양자 암호학(Post-Quantum Cryptography)

양자 컴퓨터는 현재 블록체인 시스템에서 사용되는 비대칭 암호에 위협을 가한다. ZQ2는 플러그형 암호학(pluggable cryptography)을 구현하여, 초기 사용된 암호 체계를 미래에 교체할 수 있도록 한다.

NIST에서 표준화한 양자 이후 서명 방식은 서명 및 검증 시간이 트랜잭션과 합의 메시지 인증에 적합하지만, 서명 크기가 너무 커 오늘날에는 실용적이지 않다. 특히, 본 합의 알고리즘에 필수적인 효율적인 서명 집계 방법이 존재하지 않는다.

대규모 컨트랙트 배포를 제외한 EVM 트랜잭션은 수백 바이트 크기이다. ECDSA 대신 서명 크기가 2.7 KB 인 격자 기반(lattice-based) Dilithium [21]을 사용할 경우, 트랜잭션 크기는 한 자릿수 차원(order of magnitude) 증가한다. Fast-HotStuff 블록 헤더에는 QC 또는 집계 QC가 포함되는데, QC는 블록 해시와 뷰 번호에 대한 BLS 다중서명으로 구성되며, 집계 QC는 절대 다수의 검증자별 QC와 집계 서명으로 구성된다. Dilithium을 사용하면 절대 다수에 속한 모든 검증자의 서명을 개별적으로 저장하고 검증해야 한다. 위원회에 31명의 검증자가 있는 경우, 우리는 21개의 서명을 얻게 되며, 이는 크기가 56.7 KB에 달한다. 이는 BLS 서명의 96바이트에 비해 2자릿수 차원 증가이다. 상황은 집계 QC에서는 더 악화되는데, 21개의 QC와 21개의 서명을 별도로 포함해야 하므로 집계가 불가능하다. 이 경우 오버헤드는 블록 헤더당 약 2.2 KB인 BLS 대비 1.25 MB가 되며, 이는 3자릿수 차원 증가이다.

서명 길이에 따른 상충 관계와 서명 집계 부재는 격자 기반 방식에 국한되지 않는다. 상태 기반 해시 서명 방식인 XMSS [22]와 LMS [23] 또한 1 KB가 넘는 서명을 사용하며, 무상태 해시 기반 NIST 최종 후보 SPHINCS+ [24]는 수십 KB 크기의 서명을 생성한다. 전 NIST 후보였던 Rainbow [25]는 작은 서명을 생성했으나, 수백 KB 크기의 공개키를 필요로 했고 키 복구 공격에 취약한 것으로 드러났다 [26].

B. 병렬화를 통한 확장(Scaling through Parallelism)

병렬 스마트 컨트랙트 실행은 질리카 프로토콜 초기부터 핵심적인 역할을 해왔다. 주목할 만한 결과물로는 트랜잭션 실행의 병렬 스케줄링에 필요한 속성을 추론하기 위한 정적 코드 분석 도구인 CoSplit [27]이 있다. 이는 트랜잭션이 각각 겹치지 않는(disjoint) 상태의 부분(조각)에서 작동하거나 연산이 교환 가능(commutative)하다면, 실행 순서에 관계없이 동일한 결과를 낸다는 통찰에 기반하였다. ZQ1을 위해 설계된 CoSplit은 트랜잭션을 동시에 동작하는 샤드에 할당한 후 상태 변화를 병합하는 방식이었다. 반면, ZQ2는 각 x-shard의 블록에 포함된 트랜잭션을 멀티스레드 방식으로 실행하기 위해 병렬화가 필요하다.

오늘날 일부 비-EVM 체인은 트랜잭션 실행 시 사용할 상태의 부분을 명시적으로 지정하도록 요구한다. 이를 통해 합의는 상태가 겹치지 않는 트랜잭션의 경우 전체 순서를 강제할 필요가 없다. 예를 들어, 솔라나(Solana)에서는 겹치지 않는 상태의 부분에 영향을 미치는 트랜잭션들이 동시에 실행된다. 수이(Sui)에서는 소유된 객체만 접근하는 트랜잭션은 해당 객체를 사용하지 않는 트랜잭션과 병렬로 실행되며, 이 과정에서 합의를 거치지 않는다.

질리카의 병렬 트랜잭션 실행 연구는 초기에 Scilla 컨트랙트에 초점을 맞췄으나, 현재 연구는 애플토스(Aptos)와 폴리곤(Polygon)에 통합된 Block-STM [28]에서 시작된 병렬 EVM에 집중되어 있다. Block-STM은 낙관적 동시성 제어(optimistic concurrency control)를 사용한다. 실행 대기 트랜잭션과 검증 대기 트랜잭션을 각각 저장하는 두 개의 순서 집합을 유지하며, 해당 작업들을 멀티스레드 방식으로 수행한다. 사전에 정의된 직렬화 순서는 블록 내 인덱스에 따라 두 집합의 트랜잭션을 우선순위화한다. 각 트랜잭션은 한 번 또는 여러 번 실행될 수 있다. 실행 중에는 읽고 쓰는 모든 상태 요소가 기록되어 트랜잭션의 읽기/쓰기 집합(read/write set)을 형성한다. 쓰기 집합이 이전 실행 이후 변경되지 않았다면 해당 트랜잭션만 검증 대상으로 예약된다. 변경된 경우, 대기 중이거나 실행 중이지 않은 더 높은 인덱스의 트랜잭션도 검증 대상으로 예약된다.

검증 과정에서는 트랜잭션의 읽기 집합에 있는 상태 요소의 현재 값이 실행 중 기록된 값과 비교된다. 만약 더 낮은 인덱스 트랜잭션의 쓰기 집합으로 인해 값이 일치하지 않으면, 검증 중인 트랜잭션은 중단되고 재실행으로 예약되며, 더 높은 인덱스의 모든 비대기/비실행 트랜잭션도 검증 대상으로 예약된다. 또한, 중단된 트랜잭션의 쓰기 집합은 무효화되지만, 재실행 이후 예상되는 쓰기 집합 요소를 추정하기 위해 보관된다. 실행 중인 트랜잭션이 추정으로 표시된 요소를 읽으면, 해당 요소를 기록하는 트랜잭션이 재실행을 끝낼 때까지 중단되고 이후 다시 실행된다. Block-STM과 유사한 낙관적 병렬화는 세이(Sei) V2에서도 채택되었다. Monad는 정적 코드 분석기를 통해 예측된 트랜잭션 의존성을 활용하여 낙관적 실행을 보완하며, BNB 스마트 체인에 통합된 NodeReal의 Parallel EVM 2.0은 블록 제안자가 아닌 노드의 실행 속도를 향상시키는 데 사용된다.

멀티스레딩 외에도, ZQ2에서의 병렬화 사례로 멀티블록 실행(multiblock execution)이 있다. 이는 EVM 연산의 복잡성으로 인해 블록 한도를 초과할 수 있는 트랜잭션 실행을 중단하고 이후 블록에서 다시 이어서 실행할 수 있도록 한다. 멀티블록 실행에는 원자성(atomicity)과 격리성(isolation)이 요구된다. Dfinity의 Deterministic Time Slicing은 원자성을 보장하지만, 멀티블록 연산이 완료되기 전까지는 동시 상태 조회만 허용하는 제한이 있으며, 이는 ZQ2가 극복하려는 한계이다.

C. 합의 개선(Consensus Improvements)

ZQ2 합의의 향후 개선 사항에는 비밀 리더 선출(secret leader election), 검증자 책임성(validator accountability), 프런트러닝(frontrunning) 방지가 포함된다.

ZQ2에 구현된 무작위 리더 선출은 리더의 신원을 사전에 드러내어 공격에 취약하게 만든다. 폴카닷(Polkadot)의 Sassafras [29]와 이더리움의 Whisk [30]와 같은 제안은 비밀 리더 선출이 기존 프로토콜에 어떻게 통합될 수 있는지를 보여준다. Whisk는 이더리움 검증자가 암호학적 커밋먼트(commitment)를 게시하도록 요구한다. 매 256 에포크(약 27시간)마다, RANDAO를 사용하여 16,384개의 커밋먼트가 후보로 선택된다. 이후 8,191 에포크 동안 모든 블록 제안자는 무작위로 선택된 128개의 커밋먼트를 섞고(shuffle) 유효한 셔플링 증명을 제공한다. 셔플링이 발생하지 않는 쿨다운(cooldown) 단계라 불리는 추가 1 에포크가 지난 후, 검증자들은 8,192개의 커밋먼트를 블록 제안자로 선택한다. 선택된 검증자는 자신이 어느 슬롯에서 리더로 선택되었는지를 알고 있으며, 제안 블록에서 자신의 커밋먼트를 공개함으로써 이를 증명할 수 있다. 다른 이들은 리더가 셔플링 이전에 후보였지만 셔플링 이후 블록 제안자로 선택되지 않은 검증자 중 최소 절반 이상이 포함된 익명 집합에 속한다는 것만 알 수 있다.

책임성은 정직한 검증자가 프로토콜에서 이탈한 악의적 검증자를 암호학적 증거로 입증할 수 있도록 한다. [31]에서 제시된 포렌식 지원은 PBFT와 HotStuff에 약간의 수정만 가해도, 안전성 위반을 탐지한 누구든 단일 정직한 검증자의 기록(transcript)을 기반으로 $3f + 1$ 검증자 중 최대 $2f$ 악의적 다수에서 $f + 1$ 명의 범인을 식별할 수 있도록 한다. 이러한 결과는 ZQ2에서 사용되는 Pipelined Fast-HotStuff에도 적용할 수 있다.

프런트러닝은 블록 제안자가 자신의 트랜잭션을 다른 사용자의 트랜잭션보다 앞에 삽입하여, 다른 사용자에게 피해를 주면서 MEV 증가와 같은 이득을 얻는 것이다. 프런트러닝 방지를 위해 세이는 일부 AMM이 사용하는 기법인 주문 배치(order batching)를 사용한다. 이는 들어오는 주문을 수집하여 사전 정의된 시간 간격마다 평균 가격으로 실행한다. 주문이 배치 내에서 처음이든 마지막이든 상관없다. Submarine commitments [32]는 프런트러닝 저항을 위해 제안된 커밋-공개(commit-reveal) 방식의 예시이다. 사용자는 나중에 조건부(counterfactual)로 배포될 컨트랙트 주소에 예치금을 송금하는데, 이는 원래 대상 컨트랙트에 송금할 금액과 동일하다. 이를 커밋 트랜잭션(commit transaction)이라 하며, 새 주소로의 최초 토큰 전송과 구분되지 않는다. 이때 사용된 솔트(salt)는 사용자의 주소와, 프런트러닝 공격자로부터 숨기고 싶은 정보(예: 매수/매도 주문의 호출 데이터)를 연결하여 해시한 값이다. 이후 사용자는 블록 번호 및 커밋 트랜잭션의 머클 증명과 함께 이 정보를 공개하는 트랜잭션을 제출한다. 이를 공개 트랜잭션(reveal transaction)이라 한다. 대상 컨트랙트(예: DEX)는 커밋 트랜잭션에서 사용된 주소에 새로운 컨트랙트를 조건부로 배포하고, 이를 호출해 예치금을 회수한 뒤 공개된 정보에 따라 실제 로직(매수/매도 주문)을 실행한다. 또한 커밋 트랜잭션이 공개 트랜잭션에 명시된 블록에 실제 포함되었는지, 해당 블록 번호가 충분히 오래된 것인지를 검증한다. 커밋과 공개 트랜잭션 사이에는 최소한의 블록 간격이 요구되는데, 이는 악의적 블

록 제안자가 커밋 트랜잭션이 포함된 블록을 재구성하거나 공개 트랜잭션의 포함을 지연시킨 뒤 자신들의 커밋·공개 트랜잭션을 삽입하는 것을 방지하기 위함이다.

Fino [33]는 Narwhal [34]과 Bullshark [35] 같은 DAG 기반 시스템과 다른 커밋-공개 방식을 결합한다. 사용자는 자신의 트랜잭션을 암호화하여 공정한(즉, 블라인드) 순서를 보장한다. 이들은 $\$f + 1\$ / \$3f + 1\$$ 샤미르 비밀분산(Shamir secret sharing) [36] 방식을 사용해 검증자들에게 복호화 키의 조각을 분배한다. 여기서 $\$f\$$ 는 악의적 검증자의 최대 개수이다. 순수하게 스마트 컨트랙트 기반인 submarine commitments와 달리, Fino의 방식은 합의 프로토콜에 통합이 필요하다. 다른 접근 방식으로는 프런트러닝 방지를 위해 임계치 암호화(threshold encryption) [37], 신원 기반 암호화(identity-based encryption) [38], 신뢰 실행 환경(TEE, Trusted Execution Environment) [39]을 사용하는 방안이 제안되기도 한다.

D. 영지식 증명(Zero-Knowledge Proofs)

질리카 연구의 핵심 영역 중 하나는 영지식 증명(zk-proofs) [40]이며, 특히 zk-SNARK와 zk-STARK는 ZQ2 블록체인 플랫폼 맥락에서 다양한 사용 사례를 가진다.

첫 번째 사례는 x-shard가 영지식 롤업(zk-rollup) 또는 밸리둠(validium)으로 동작할 수 있는 옵션이다. 간단히 말해, 롤업은 블록을 데이터 가용성(data availability)을 위해 기본 레이어(base layer)라 불리는 다른 블록체인에 게시하는 체인이다. 롤업 트랜잭션은 기본 레이어에 저장되지만 실행되지는 않는다. 정산(settlement)을 위해서는 트랜잭션 실행 및 이에 따른 상태 전이에 대한 유효성 증명(validity proof)이 기본 레이어에 제출되어 검증되어야 한다. 사용자는 시퀀서(sequencer)에게 트랜잭션을 전송하면, 시퀀서는 이를 블록으로 정리하고 실행한 후, zkEVM 프루버(prover)가 생성한 유효성 증명과 함께 기본 레이어에 전송한다. 풀 노드를 실행하는 사용자는 기본 레이어 블록에서 롤업 트랜잭션을 가져와 실행하여 상태를 재구성할 수 있으며, 라이트 클라이언트는 기본 레이어에 저장된 롤업 상태 루트에 의존할 수 있다. 따라서 영지식 롤업 기반 x-shard는 자체 검증자 집합과 합의가 필요하지 않다. 단일 주체에 의해 운영될 수도 있지만, 탈중앙화된 무허가(permissionless) 설정이 여전히 바람직하다. 롤업과 밸리둠의 차이는 트랜잭션을 어디에 저장하는지에 있다. 롤업 트랜잭션은 기본 레이어 블록에 저장되는 반면, 밸리둠 트랜잭션은 데이터 가용성 위원회(data availability committee)와 같은 오프체인 저장소를 사용하며, 이들은 기본 레이어에 증명서를 게시한다. 일부 구현에서는 데이터 가용성 위원회가 데이터를 제공하지 않을 경우 삭감되는 보증금을 제공해야 한다. 볼리션(volition)은 롤업과 밸리둠을 결합한 형태로, 사용자가 트랜잭션 단위로 데이터 가용성 모드를 선택할 수 있게 한다.

영지식 증명을 사용하더라도 영지식 롤업은 프라이버시를 제공하지 않는다. 트랜잭션이 공개되어 있으며, 스마트 컨트랙트 코드 포함 상태는 트랜잭션 실행으로 재구성될 수 있기 때문이다. 상태, 트랜잭션 입력, 트랜잭션 송신자를 노드 운영자로부터 비밀로 유지하려면 프로토콜이 프라이빗 연산(private computation)을 지원해야 한다. 전통적인 블록체인에서는 사용자가 입력이 포함된 트랜잭션을 평문으로 전송하고 모든 전체 노드가 스마트 컨트랙트를 실행해 상태를 갱신한다. 그러나 아즈텍(Aztec), 미나(Mina), 알레오(Aleo)와 같은 프라이버시 프로토콜에서는 스마트 컨트랙트가 사용자 제공 프라이빗 입력으로 오프체인에서 실행된다. 온체인에 제출된 트랜잭션에는 간결한 증명만 포함되며, 온체인에서 실행되는 프라이빗 스마트 컨트랙트는 단순히 증명 검증자(proof verifier) 역할을 한다. 프라이빗 상태 값도 오프체인에 저장된다. 온체인에 공개되는 정보는 상태 커밋먼트(state commitment)와 널리파이어(nullifier)뿐이다. 계정 인증은 트랜잭

선 간 연계성(unlinkability)을 보장하기 위해 프라이빗 연산의 일부로 수행된다.

비대화형 영지식 증명의 또 다른 유망한 응용 분야는 신뢰 최소화(trust-minimized) 영지식 브릿지이다. 많은 브릿지는 스마트 컨트랙트에 라이트 클라이언트 프로토콜을 구현하여 다른 체인의 블록 헤더를 추적하고, 사용자가 머클 증명을 제출해 블록 헤더의 루트 해시와 대조하여 검증할 수 있도록 한다. 그러나 브릿지 컨트랙트가 다른 체인의 합의를 검증해야 하므로 가스 비효율적이다. 텔레파시(Telepathy)와 폴리헤드라(Polyhedra)와 같은 영지식 브릿지는 이러한 검증을 오프체인에서 수행하고, 합의에 대한 영지식 증명을 브릿지 컨트랙트에 제출한다. Pipelined Fast-HotStuff에 대한 영지식 증명을 구현하면, 다른 체인에 배포된 브릿지 컨트랙트가 ZQ2 블록 헤더를 UCCB가 사용하는 ZQ2 검증자 절대 다수를 필요로 하지 않고도 효율적으로 처리할 수 있게 된다. 신뢰 최소화 영지식 브릿지 개념은 하이퍼 오라클(Hyper Oracle) [41]과 같은 신뢰 최소화 영지식 오라클로 확장될 수 있으며, 이들은 단순히 다른 체인의 블록 헤더뿐만 아니라 오프체인 데이터를 영지식 증명과 함께 제공할 수 있다.

마지막으로, 사용자는 Reclaim 프로토콜 [42]이 보여주듯 zk-SNARK를 활용해 신원을 증명하거나, Privacy Pools [43]에서 구현된 것처럼 익명성을 달성할 수도 있다.

참고 자료

[1] <https://docs.zilliqa.com/whitepaper.pdf>

[2] <https://pmg.csail.mit.edu/papers/osdi99.pdf>

[3] <https://arxiv.org/pdf/2010.11454.pdf>

[4] https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm

[5] <https://github.com/libp2p/specs/>

[6] <https://en.wikipedia.org/wiki/EdDSA>

[7] https://en.wikipedia.org/wiki/BLS_digital_signature

[8] https://en.wikipedia.org/wiki/Schnorr_signature

[9] <https://arxiv.org/pdf/1803.05069.pdf>

[10] <https://eips.ethereum.org/EIPS/eip-4337>

[11] <https://eips.ethereum.org/EIPS/eip-2930>

[12] <https://eips.ethereum.org/EIPS/eip-1559>

- [13] <https://eips.ethereum.org/EIPS/eip-155>
- [14] <https://decentralizedthoughts.github.io/2020-12-22-what-is-a-merkle-tree/>
- [15] https://en.wikipedia.org/wiki/Bloom_filter
- [16] https://eth2book.info/capella/part2/building_blocks/randomness/
- [17] https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange
- [18] <https://gov.zilliqa.com/>
- [19] <https://vote.zilliqa.com/#/gzil>
- [20] <https://github.com/Zilliqa/ZIP>
- [21] <https://eprint.iacr.org/2017/633.pdf>
- [22] <https://eprint.iacr.org/2011/484.pdf>
- [23] <https://datatracker.ietf.org/doc/html/rfc8554>
- [24] <https://sphincs.org/data/sphincs+-paper.pdf>
- [25] http://archive.ymcsc.tsinghua.edu.cn/pacm_download/672/12596-dingjt10.pdf
- [26] <https://eprint.iacr.org/2022/214.pdf>
- [27] <https://ilyasergey.net/assets/pdf/papers/cosplit-pldi21.pdf>
- [28] <https://arxiv.org/pdf/2203.06871.pdf>
- [29] <https://eprint.iacr.org/2023/031.pdf>
- [30] <https://ethresear.ch/t/whisk-a-practical-shuffle-based-ssle-protocol-for-ethereum/11763>
- [31] <https://arxiv.org/pdf/2010.06785.pdf>
- [32] <https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-breide>

[nbach.pdf](#)

[33] <https://arxiv.org/pdf/2208.00940.pdf>

[34] <https://arxiv.org/pdf/2105.11827.pdf>

[35] <https://arxiv.org/pdf/2201.05677.pdf>

[36] https://en.wikipedia.org/wiki/Shamir%27s_secret_sharing

[37] <https://eprint.iacr.org/2022/898.pdf>

[38] <https://eprint.iacr.org/2022/1066>

[39] <https://vukolic.com/Fairy-CR.pdf>

[40] https://en.wikipedia.org/wiki/Non-interactive_zero-knowledge_proof

[41] <https://ethresear.ch/t/defining-zkoracle-for-ethereum/15131>

[42] https://drive.google.com/file/d/1wmfdtIGPaN9uJBI1DHqN903tP9c_aTG2/view

[43] <https://github.com/ameensol/privacy-pools>

