경량 합의 알고리즘 설계 방향성

https://youtu.be/lddvGmr67ic

합의 알고리즘 서베이

경량 합의 알고리즘 설계 (TEE 기반)

PoL & PoET 구현

합의 알고리즘 서베이

PoET (Proof-of-Elapsed-Time)

- **PoET (Proof-of-Elapsed-Time)**[2]
  - 기존 **PoW의 경쟁적인 컴퓨팅 자원 소모 문제**를 해결하기 위한 합의 알고리즘
  - 모든 노드에게 **공평한 기회를 제공**하기 위하여 무작위로 생성된 대기 시간을 적용

    → 가장 짧은 대기 시간을 지닌 노드가 블록 생성

    → Intel SGX와 같은 TEE를 통해 대기 시간이 실제로 경과하였는지에 대하여 검증
  - 일정 분포에 따라 대기 시간을 무작위로 생성하였는지에 대해 검증하기 위한 통계적 테스트인 Z-Test 수행

PoL (Proof-of-Luck)

- **PoL (Proof-of-Luck)**[3]
  - 기존 **PoW의 무의미한 컴퓨팅 자원 소모 문제**를 해결하기 위한 합의 알고리즘
  - 모든 노드에게 무작위로 생성된 0~1 사이의 luck value 부여

    → 가장 높은 luck value를 갖는 노드가 블록 생성
  - PoW, PoO, PoT에 TEE를 적용함으로써 얻는 장점을 적용

    → PoW에 TEE를 적용함으로써 **ASIC 저항성을 만족**

    → PoO에 TEE를 적용함으로써 **다중 호출 방지**

    → PoT에 TEE를 적용함으로써 Sleep을 통한 **Cycle과 자원을 절약**
  - **Merging 과정을 통해 N개의 블록을 super block에 병합시킴으로써 공간 효율적으로 압축**

LPoS (Leased-PoS)

- **LPoS (Leased-PoS)**[7]
  - 기존 **PoS의 지분 독점 문제**를 해결하기 위한 합의 알고리즘
  - 지분이 적은 노드에게 일정 지분을 임대하여 블록 생성 확률을 증가시켜주는 합의 알고리즘

    → **공평한 블록 생성 기회 적용**

    → 지분의 분산화를 통해 **블록체인의 안정성 증대**
  - 만약, 임대를 받은 노드가 블록을 생성했을 경우 임대를 해준 노드에게 보상의 일정량 분배

PoI (Proof-of-Importance)

- **PoI (Proof-of-Importance)**[8]
  - 기존 **PoS의 지분 독점 문제**를 해결하기 위한 합의 알고리즘
  - PoI는 중요도에 따라 블록 생성 확률 증가

    → 기득 통화량, 코인 거래량, 노드간 상호 연결도에 따라 중요도가 결정됨

    → 중요도를 높이기 위해 통화량을 증가시킴으로써 **지분 분산화**

    → **공평한 블록 생성 기회 제공**

dBFT (delegated-BFT)

- **dBFT (delegated-BFT)**[10]
  - 기존 **PBFT의 확장성의 한계**를 해결하기 위한 합의 알고리즘
  - PBFT에 위임(delegate)의 개념을 추가하여 대표자 선출

    → 대표자로 선출된 일부 노드들만이 PBFT 합의 알고리즘 수행
  - 자신의 권한을 위임해 대표자를 선출한다는 점에서 **민주주의적**
  - 합의에 모든 노드가 참여하지 않음으로써 **확장성 향상**
  - 네트워크 크기에 비해 대표자 수가 적다면 **중앙집중화 문제**가 발생할 수 있음

Tendermint

- **Tendermint**[12]
  - PBFT의 **확장성의 한계 문제**를 해결하기 위한 합의 알고리즘
  - PBFT와 DPoS의 하이브리드 합의 알고리즘
    - 기존의 PBFT는 모든 노드가 동일한 투표권을 갖지만 Tendermint는 지분에 따라 투표를 함으로써 합의
    - Propose, Prevote, Precommit, Commit, NewHeight의 단계로 구성
  - 악의적인 노드의 참가를 전체 노드의 1/3 까지 허용
  - 투표에 참여한 지분을 동결시킴으로써 **이중 투표 문제 해결**
  - 만약 악의적인 행동을 시도할 경우, 지분을 뺏음으로써 **Nothing at Stake 문제 해결**

PoBT (Proof-of-Block & Trade)

- **PoBT (Proof-of-Block & Trade)**[13]
  - Hyperledger Fabric 프레임워크 기반의 **대규모 IoT 시스템**을 위한 경량 합의 알고리즘
  - 합의에 참여하는 노드의 수를 기반으로 피어 노드를 통합하는 알고리즘 활용

    → 검증에 필요한 **네트워크 오버헤드 및 시간 감소**

    → 리소스가 제한된 IoT 장치의 **트랜잭션 속도 증가**
  - 트랜잭션에 분산 피어 노드 시스템을 사용하여 **IoT 노드에 필요한 메모리 절감**
  - 원장에 커밋되기 전에 트랜잭션뿐만 아니라 추가적으로 블록을 검증하여 **보안성 및 변조 방지성 향상**

HPoC (Hierarchical Proof-of-Capability)

- **HPoC (Hierarchical Proof-of-Capability)**[14]
  - **저사양 IoT 디바이스 상에서의 블록체인**을 위한 합의 알고리즘
  - 계층적 구조는 1분, 10분, 하루마다 블록이 생성되는 체인으로 분류

    → 1분 체인은 매 분 블록을 생성 후 체인에 추가

    → 10분 체인은 10분동안의 블록을 병합 및 패킹하여 체인에 추가

    → 하루 체인은 클라우드에 업로드 된 하루 동안의 블록을 병합 및 패킹하여 체인에 추가
  - 10분 체인과 하루 체인에 비동기 PoW 매커니즘을 사용

    → 각 체인에 있는 **블록의 변조 방지성 향상**
  - 각 노드는 7일 간의 블록만을 저장하고 있어 **저장 공간의 부담 감소**
  - 클라우드를 사용한다는 점에서 **중앙화 및 단일 실패 지점 문제 발생 가능성** 존재

Feature / Consensus	Decentralization	Scalability	Throughput	Latency	Computing Overhead	Network Overhead	Storage Overhead
PoW	High	High	Low	High	High	Low	High
PoET	Medium	**High**	**High**	**Low**	**Low**	**Low**	High
PoL	**High**	**High**	N/A	Medium	**Low**	**Low**	Medium
PoS	High	High	Low	Medium	Medium	Low	High
DPoS	Medium	**High**	**High**	Medium	Medium	N/A	High
Roll-DPoS	Medium	**High**	**High**	Medium	Medium	High	**Low**
LPoS	High	High	Low	Medium	Medium	Low	High
PoI	High	High	High	Medium	Low	Low	High
PBFT	Medium	Low	High	Low	Low	High	High
dBFT	Medium	**High**	**High**	Medium	**Low**	High	High
SCP	High	High	High	Medium	Low	Medium	High
Tendermint	Medium	High	High	Low	Low	High	High
PoBT	Medium	High	Medium	Low	Low	Low	Low
HPoC	Low	High	High	Medium	Low	Medium	Low

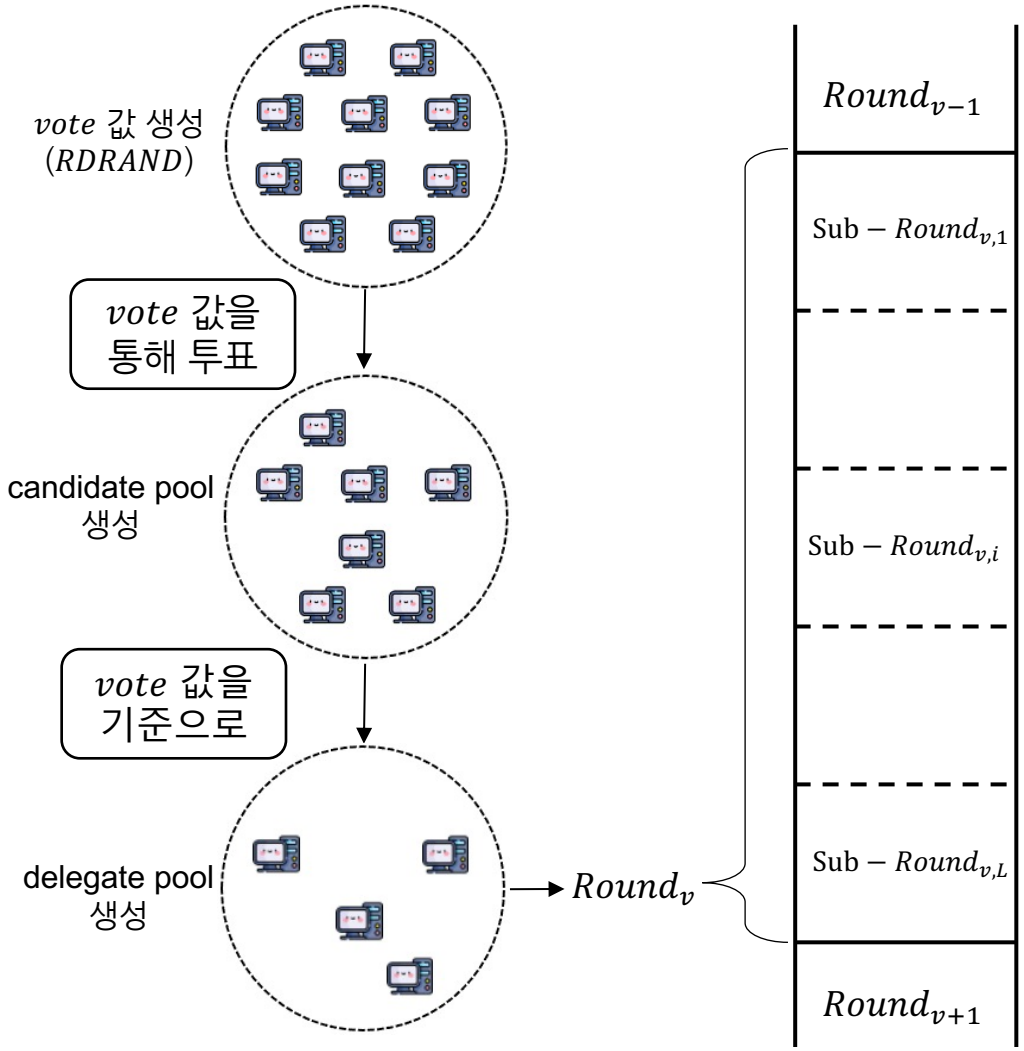
Zheng, Zibin, et al. "Blockchain challenges and opportunities: A survey." *International journal of web and grid services* 14.4 (2018): 352-375.

경량 합의 알고리즘 설계 1 (TEE 기반)

PoM_TEE (Proof of Merge_TEE)

- **Overview of the PoM_TEE**

1. 각 노드들이 **난수를 생성**하여 $vote$ 값으로 사용
2. $voter$들은 delegate로 선출하고자 하는 candidate에게 $vote$ 값 전송 (**투표**)
3. 각 candidate들은 전송받은 $vote$ 값을 모두 더함 (***sumVote***)
4. $sumVote$ 값이 가장 큰 상위 M명의 candidate선택 (**candidate pool**)
5. 각 candidate들이 생성했던 $vote$ 값에 따라 상위 N명을 delegate로 선정
6. N명의 delegate들이 순서대로 블록 생성
    1) sub-round마다 1명의 delegate가 블록 생성
    2) 다른 $delegate$들은 블록에 대한 검증 진행
    3) 검증이 완료되었을 경우 다음 delegate가 블록 생성
7. 모든 delegate들이 블록을 생성했을 경우 라운드(round) 종료



PoM_TEE (Proof of Merge_TEE)

1. 각 voter들이 난수를 생성하여 $vote$ 값으로 사용

- TEE의 $RDRAND$ 명령어를 사용하여 $vote$ 값 생성
  - → $0 < vote \leq 1$
  - → 변조가 불가능한 Uniform Distribution 상에서의 무작위 값을 생성
  - → 해당 값은 delegate를 선출할 때도 사용

2. voter들은 delegate로 선출하고자 하는 candidate에게 $vote$ 값 전송 (투표)

- voter들은 생성하고자 하는 블록의 헤더에 $vote$값을 포함
- 해당 블록의 헤더를 전송함으로써 투표 (트랜잭션의 형태, $voteTx$)

} **Merging을 위한 과정**

- 각 $voter$들은 라운드 당 1회의 투표만 가능
- 트랜잭션을 검증하기 위해 CRYSTALS-Dilithium을 통해 전자서명
  - → 전자서명을 함으로써 vote 값에 대한 조작 방지
  - → 개인키는 TEE의 enclave 영역에 저장
- Remote Attestation을 통한 신뢰 가능한 네트워크 통신

PoM_TEE (Proof of Merge_TEE)

3. **각 candidate들은 전송받은 헤더의 $vote$ 값을 모두 더함 ($sumVote$)**
   - 생성하고자 하는 블록에 $voteTx$과 $sumVote$를 포함시킨 후 브로드캐스팅

4. **$sumVote$ 값이 큰 상위 M명의 candidate 선택 (candidate pool)**
   - candidate들의 $sumLuck$ 값을 내림차순으로 정렬
   - 상위 M명을 candidate pool에 포함

5. **각 candidate들이 생성했던 $vote$ 값에 따라 상위 N명을 delegate로 선정**
   - 악의적인 노드들의 담합을 막기 위한 과정
   - TEE에 의해 생성된 $vote$ 값에 따라 delegate가 선정되므로 조작 방지
   - 상위 N명을 delegate pool에 포함

PoM_TEE (Proof of Merge_TEE)

6. N명의 delegate들이 순서대로 블록 생성

1) sub-round마다 1명의 delegate가 블록 생성

→ delegate에게 투표한 $voter$들의 블록 전체를 merging함으로써 블록 병합

2) 다른 delegate들은 블록에 대한 검증 진행

→ 블록 내의 Transaction이 변조되지 않았는지 검증

→ 블록을 생성한 delegate 의 블록 생성 횟수가 비정상적으로 많은지에 대해 Z-Test를 통해 검증

3) 검증이 완료되었을 경우 다음 delegate 가 블록 생성

7. 모든 delegate들이 블록을 생성했을 경우 라운드(round) 종료

PoM의 특징

- $vote$ 값을 무작위로 생성함으로써 **공평한 블록생성자 지정**

  → 블록 독점 생성 방지

- CRYSTALS-Dilithium을 통한 전자서명

  → **양자 내성 암호**

  → **트랜잭션 변조 방지**

  → **vote 값 변조 방지**

- Merging을 통해 하위 블록을 슈퍼 블록으로 병합함으로써 프로토콜의 **throughput과 liveness 향상**
- Delegate를 선출함으로써 **Scalability 향상 및 Network Overhead 감소**
- 네트워크의 크기에 따라 delegate의 수를 조절하여 **중앙집중화 방지 및 효율성 증대**
- 통계적 테스트인 Z-Test를 통해 노드가 비정상적으로 블록을 자주 생성하는지 검증

블록체인 합의 알고리즘 시뮬레이터 (PoL)

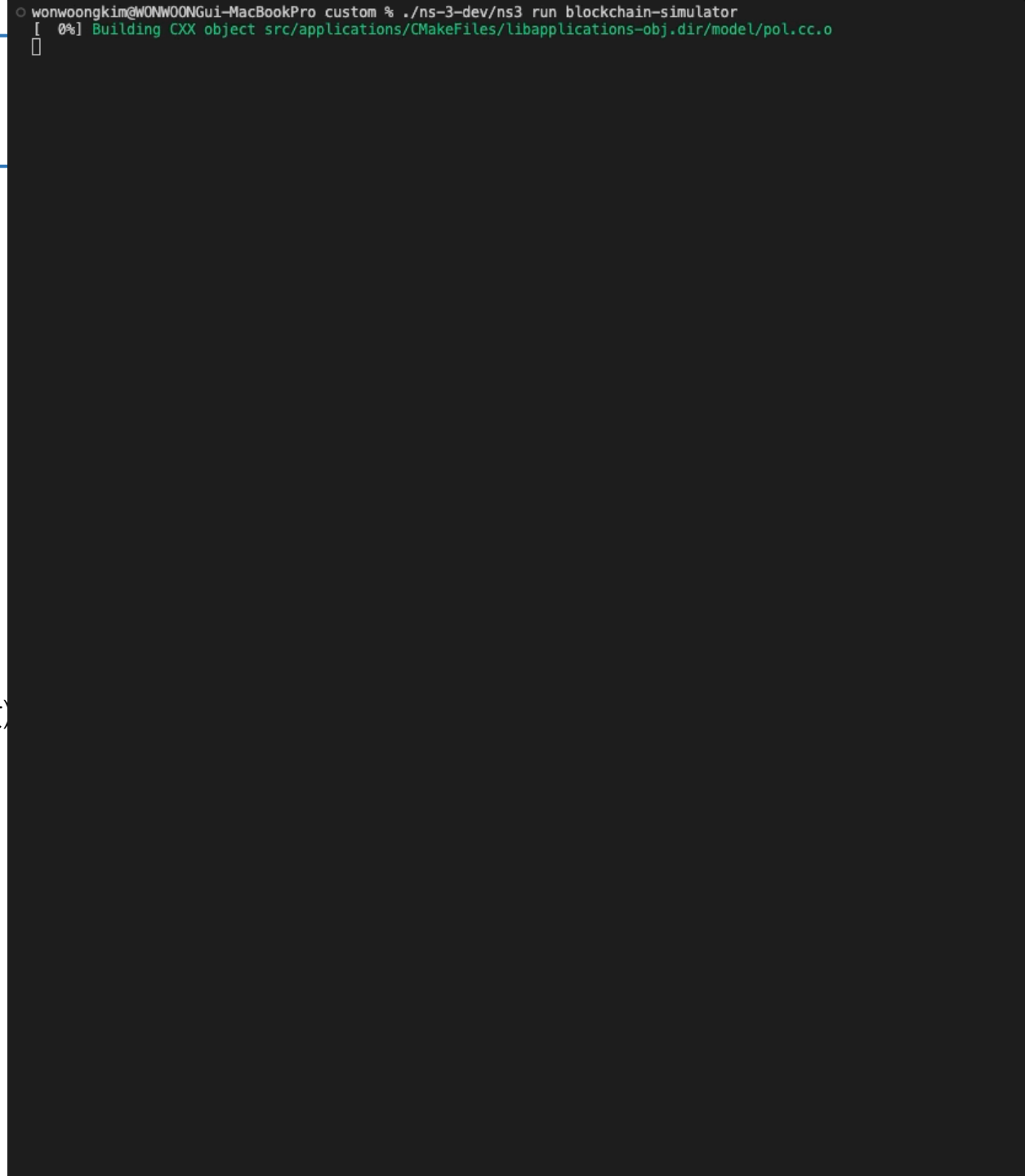
PoL Implementation

- **PoL 시연 영상**
  - **Node의 수**: 4
  - **ROUND_TIME**: 1
  - **GenesisBlock** {
    prevHash: 0000…0000
    Tx: 0000…0000
    Proof: iFiY4cAcuEq6c5Bx2QKnja3XGadagh
    m9zRcmtgTrgMQU6vRCr56xhUY5xA5gsZ6ZQ9
    6THB5LizwKNtWbVNsYY4sx1Pw2XUTzBh00
    (nonce와 luck이 0으로 초기화된 값을 Base58 encoding 한 값)
    }

prevHash: 이전 블록의 해시값

Tx: 트랜잭션

Proof: nonce와 luck값을 base58 encoding 한 값



PoL Implementation

1) **체인의 초기 상태 및 Monotonic Counter 설정**
   → pre-defined 된 초기 블록을 체인에 추가
   → 병렬 실행을 막기위한 Monotonic Counter 설정

```
// ===== 초기 상태 설정 ===== //
Block genesisBlock;                                                    // genesisBlock 생성
m_chain.AddBlock(m_chain, genesisBlock);                               // genesisBlock 추가
m_secureWorker.IncrementMonotonicCounter();                            // 초기 Monotonic Counter 설정

cout << "Node " << GetNode()->GetId() << "'s                          : "
<< m_secureWorker.ReadMonotonicCounter() << endl;
```

체인의 초기 상태 설정

Monotonic Counter의 초기 상태 설정

소스 코드

```
Node 0's current Monotonic Counter: 1
Node 1's current Monotonic Counter: 1
Node 2's current Monotonic Counter: 1
Node 3's current Monotonic Counter: 1
```

실행 결과

PoL Implementation

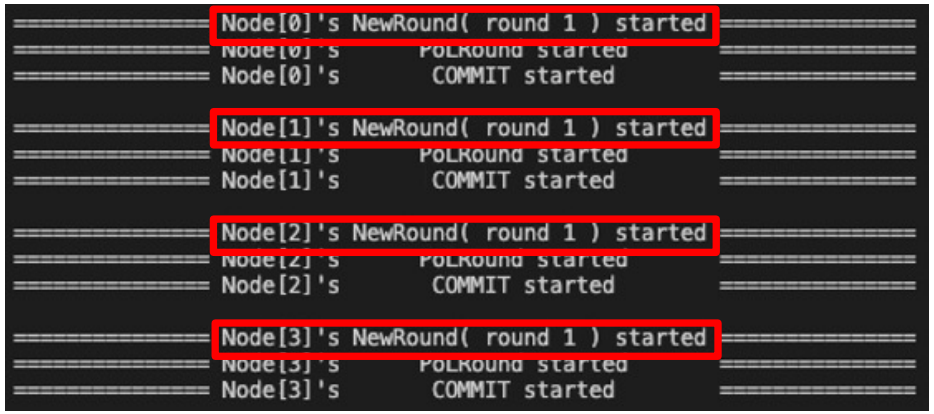
2) NewRound를 통해 블록 생성을 위한 라운드 시작

→ 가장 최신 블록을 roundBlock으로 설정

```
// ================ 새로운 라운드 시작 ================ //
void
PoL::NewRound(void) {
    round++;
    NS_LOG_INFO(endl << "================ Node[" << GetNode()->GetId() << "]'s NewRound( round " << round << " ) started ================");
    Block roundBlock;
    roundBlock = m_chain.GetLatestBlock();

    PoLRound(roundBlock);
}
```

소스 코드



실행 결과

PoL Implementation

3) PoLRound를 통해 TEE 준비 및 roundBlock, roundTime 설정

→ roundBlock은 이번 라운드에 채굴할 블록

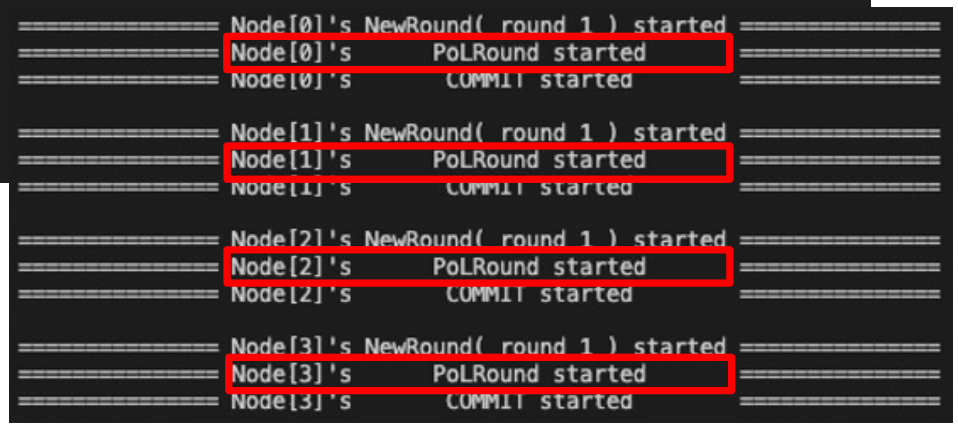
→ roundTime은 실제 대기시간을 보냈는지 검증하기 위한 값

```
// ================ TEE를 준비하기 위한 함수 ================ //
void
PoL::PoLRound(Block block) {
    NS_LOG_INFO("================ Node[" << GetNode()->GetId() << "]'s      PoLRound started       ================");

    m_roundBlock                = block;
    m_roundTime                 = m_secureWorker.GetTrustedTime();
    vector<uint8_t> newTxs      = GetPendingTxs();

    Commit(newTxs);
}
```

소스 코드



실행 결과

PoL Implementation

4) Commit을 통해 체인에 newBlock 추가 및 PoLMine을 동작시켜 proof 생성

```
// =============== 체인에 새로운 블록 추가 =============== //
void
PoL::Commit(vector<uint8_t> newTxs) {
    NS_LOG_INFO("=============== Node[" << GetNode()->GetId() << "]'s       COMMIT started       ===============");
    //NS_LOG_INFO("COMMIT started time: " <<  (Time)Simulator::Now().GetSeconds());

    Block previousBlock = m_chain.GetLatestBlock();

    Block::BlockHeader header;
    header.prevHash    = GetBlockHash(previousBlock);
    header.tx          = newTxs;

    /*
        PoLMine 실행 전 ROUND_TIME 만큼의 delay를 주기 위하여 Schedule 사용
        이때, Schedule 함수는 return이 불가능하여 PoLMine내에서 AddBlock까지 수행
    */
    Simulator::Schedule(Seconds(ROUND_TIME), &PoL::PoLMine, this, header, previousBlock);
}
```

소스 코드

```
=============== Node[0]'s NewRound( round 1 ) started ===============
=============== Node[0]'s       PoLRound started       ===============
=============== Node[0]'s       COMMIT started         ===============

=============== Node[1]'s NewRound( round 1 ) started ===============
=============== Node[1]'s       PoLRound started       ===============
=============== Node[1]'s       COMMIT started         ===============

=============== Node[2]'s NewRound( round 1 ) started ===============
=============== Node[2]'s       PoLRound started       ===============
=============== Node[2]'s       COMMIT started         ===============

=============== Node[3]'s NewRound( round 1 ) started ===============
=============== Node[3]'s       PoLRound started       ===============
=============== Node[3]'s       COMMIT started         ===============
```

실행 결과

PoL Implementation

5) PoLMine을 통해 블록 생성

→ PoLMine 과정중에 header와 Monotonic Counter에 대한 검증 수행

→ Remote Attestation을 통해 블록의 luck값이 변조되지 않았는지에 대해 검증하기 위한 proof 생성

소스 코드

```
// ================ 채굴 과정 ================ //
void
PoL::PoLMine(Block::BlockHeader header, Block previousBlock) {
    NS_LOG_INFO(endl << "================ Node[" << GetNode()->GetId() << "]'s PoLMine started ================");
    //NS_LOG_INFO("PoLMine started time: " <<  (Time)Simulator::Now().GetSeconds());

    Time now = m_secureWorker.GetTrustedTime();
    int newCounter = m_secureWorker.ReadMonotonicCounter();

    // ===== 검증 과정 ===== //
    if (header.prevHash != GetBlockHash(previousBlock)
        || previousBlock.header.prevHash != m_roundBlock.header.prevHash
        || now < m_roundTime + (Time)ROUND_TIME
        || m_secureWorker.m_counter != newCounter) {
            NS_LOG_INFO("validation failed");
            return;
    } else { NS_LOG_INFO("validation success"); }

    float luck = GetRandomLuck();
    sleep(GetWaitingTime(luck));

    vector<uint8_t> nonce = GetHeaderHash(header);
    vector<uint8_t> proof = m_secureWorker.RemoteAttestation(nonce, luck);

    Block newBlock(header.prevHash, header.tx, proof);
    m_chain.AddBlock(m_chain, newBlock);

    Simulator::Schedule(Seconds(0), &PoL::SendChain, this, m_chain);
}
```

검증 과정

무작위 luck값 생성

Proof 생성

체인에 newBlock 추가

체인 브로드캐스팅

실행 결과

```
================ Node[0]'s PoLMine started ================
validation success
Luck: 0.424242
Waiting for 1.28788s

================ Node[1]'s PoLMine started ================
validation success
Luck: 0.606061
Waiting for 1.19697s

================ Node[2]'s PoLMine started ================
validation success
Luck: 0.979798
Waiting for 1.0101s

================ Node[3]'s PoLMine started ================
validation success
.808081
 for 1.09596s
```

PoL Implementation

6) 브로드캐스팅 과정 (송신)

→ 각 노드들이 체인을 브로드캐스팅

→ proof에 luck값이 포함되어 있음

```
// ================ 체인 브로드캐스팅 ================ //
void
PoL::SendChain(Blockchain chain) {
    cout << "================================================================================" << endl;

    cout << endl << "================ Node[" << GetNode()->GetId() << "] BROADCAST chain ================" << endl;
    //NS_LOG_INFO("SendChain started time: " <<  (Time)Simulator::Now().GetSeconds());

    network.PrintChain("sended chain", chain);                                  // 브로드캐스팅할 체인 출력

    Ptr<Packet> p;

    uint8_t *data = network.Chain2Array(chain);                                 // 체인을 브로드캐스팅하기

    p = Create<Packet>(data, network.blockSize * chain.GetBlockchainHeight());  // 패킷 생성

    vector<Ipv4Address>::iterator iter = m_peersAddresses.begin();              // 시작 peer node부터
    while(iter != m_peersAddresses.end()) {                                     // 마지막 peer node까지
        Ptr<Socket> socketClient = m_peersSockets[*iter];                       // 소켓 생성
        Simulator::Schedule(Seconds(0), &PoL::SendPacket, this, socketClient, p);  // 패킷 송신
        iter++;                                                                 // 다음 peer node
    }

    Simulator::Schedule(Seconds(1), &PoL::NewRound, this);                      // 다음 라운드 스케줄링
    cout << "================================================================================" << endl;
    sleep(1);
}
```

소스 코드

```
================ Node[0] BROADCAST chain ================

sended chain's[0]th Block: {
   prevHash: 0000000000000000000000000000000000000000000000000000000000000000
   tx: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
   proof: iFiY4cAcuEq6c5Bx2QKnja3XGadaghm9zRcmtgTrgMQU6vRCr56xhUY5xA5gsZ6ZQ96THB5LizwKNtWbVNsYY4sx1Pw2XUTzBh00
}

sended chain's[1]th Block: {
   prevHash: 1acb2f94d24dcdcaeaaa46445e862113beac857d79ac5e316aab6b666551831d
   tx: 9fda737d78e62f1c2446b3ccac6f07a1299b876f7b812ea18a8debeb385f3c65fe9173018049ff5d9a01efdef39d
   proof: kpiabqJ32o4Fvf5RVNaEB87zoyCPzhtviS88L3d54keJYTCenSLs5qLZdcxS9T9kwstc81GbwFPvEsoc3wMmy4c92w2yvULgWG00
}

================================================================================
================================================================================
================ Node[1] BROADCAST chain ================

sended chain's[0]th Block: {
   prevHash: 0000000000000000000000000000000000000000000000000000000000000000
   tx: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
   proof: iFiY4cAcuEq6c5Bx2QKnja3XGadaghm9zRcmtgTrgMQU6vRCr56xhUY5xA5gsZ6ZQ96THB5LizwKNtWbVNsYY4sx1Pw2XUTzBh00
}

sended chain's[1]th Block: {
   prevHash: 1acb2f94d24dcdcaeaaa46445e862113beac857d79ac5e316aab6b666551831d
   tx: 7d7196bd6fa914496b1e7e9eaae574ce7daec0957b2a88c3f1ae8c6115af2342ff70ef06f9c33dca8e59bdb3133e
   proof: j7xWMyeA771HHsiuQbGp99PbQWgXH4eEztqg5qQU9PsdppqxNkWED4W4Nf9mXDFmYeBqECyrT9QQQx3uV5V3a7RUaTiLixMru00
}

================================================================================
================================================================================
================ Node[2] BROADCAST chain ================

sended chain's[0]th Block: {
   prevHash: 0000000000000000000000000000000000000000000000000000000000000000
   tx: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
   proof: iFiY4cAcuEq6c5Bx2QKnja3XGadaghm9zRcmtgTrgMQU6vRCr56xhUY5xA5gsZ6ZQ96THB5LizwKNtWbVNsYY4sx1Pw2XUTzBh00
}

sended chain's[1]th Block: {
   prevHash: 1acb2f94d24dcdcaeaaa46445e862113beac857d79ac5e316aab6b666551831d
   tx: 4fd962e73d030a86453ae21ffb7f711d2b8ac283fc432a160fe89fe53a5f3de9038351fb48494ead238128597aba
   proof: nYDe92t6vJnB6GLuPYk2w18UyHk4z52FkGNEJVffmqhvdsQEZD6TPw3BgdV4So2uaBTxFeT36RGtiXSAzdLYMFPfWgzicGMne100
}

================================================================================
================================================================================
================ Node[3] BROADCAST chain ================

sended chain's[0]th Block: {
   prevHash: 0000000000000000000000000000000000000000000000000000000000000000
   tx: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
   proof: iFiY4cAcuEq6c5Bx2QKnja3XGadaghm9zRcmtgTrgMQU6vRCr56xhUY5xA5gsZ6ZQ96THB5LizwKNtWbVNsYY4sx1Pw2XUTzBh00
}

sended chain's[1]th Block: {
   prevHash: 1acb2f94d24dcdcaeaaa46445e862113beac857d79ac5e316aab6b666551831d
   tx: bebfff349b057b62a13e59e49360e33329c6ad1132de03e813d7980f5b80990e41506154cb01d0a6fce201118438
   proof: mromYaqx3EkevsX4VWop33ETHUGQEfomTabaULPoZeu8VCeN3oEvm5opHikZDqr632PXsuatKVrRtW5ZF1kyRjsap2zeNpU1wb00
}

================================================================================
```

실행 결과

PoL Implementation

7) 브로드캐스팅 과정 (수신)

→수신한 체인을 검증

→해당 체인과 현재 체인의 Luck 비교

→ 수신한 체인의 Luck값이 클 경우

해당 체인으로 변경

소스 코드

```
// ================ 패킷 수신 후 실행되는 함수 ================ //
void
PoL::HandleRead(Ptr<Socket> socket) {
    cout << "================================================================================" << endl;
    cout << endl << "================ Node[" << GetNode()->GetId() << "] RECEIVED chain ================" << endl;
    //NS_LOG_INFO("HandleRead started time: " <<  (Time)Simulator::Now().GetSeconds());

    Ptr<Packet> packet;
    Address from;
    string msg;

    // ===== packet에 대한 동작 과정 ===== //
    while((packet = socket->RecvFrom(from))) {
        socket->SendTo(packet, 0, from);
        if (packet->GetSize() == 0) { break; }                          // 패킷이 비어있는지 검증

        if (InetSocketAddress::IsMatchingType(from)) {
            msg = GetPacketContent(packet, from);                       // 수신받은 패킷의 내용 Get
            Blockchain chain = network.Array2Chain(msg);                // 수신받은 패킷을 체인의 형태로 변경

            network.PrintChain("received chain", chain);                // 브로드캐스팅 받은 체인 출력

            Valid(chain);                                               // 브로드캐스팅 받은 체인 검증

            cout << "current chain's luck: " << Luck(m_chain) << endl;  // 기존 체인의 luck값 출력
            cout << "received chain's luck: " << Luck(chain) << endl;   // 브로드캐스팅 받은 체인의 luck값 출력
            cout << "================================================================================" << endl;

            sleep(1);

            // ===== luck값 비교 ===== //
            if (Luck(chain) > Luck(m_chain)) {                          // 브로드캐스팅 받은 체인의 luck값이 더 클 경우,
                cout << "================================================================================" << endl;
                cout << "* chain has changed! *"
                network.PrintChain("current chain", m_chain);           // 기존 체인 출력
                m_chain = chain;                                        // 체인 변경

                network.PrintChain                                      // 변경된 체인 출력
                cout << "================================================================================" << endl;

                sleep(1);
            }
        }
    }
}
```

수신한 체인 검증

현재 체인과 수신한 체인의 Luck값 비교

체인 변경

PoL Implementation

8) 수신한 체인의 블록이 브로드캐스팅 과정에서 위변조되지 않았는지에 대해 검증

→ 블록의 prevHash값이 이전 블록으로부터 생성되었는지

→ proof 내의 nonce 값이 올바르게 생성되었는지

→ nonce와 luck값으로부터 올바른 proof가 생성되었는지

```
// =============== 체인 검증 =============== //
void
PoL::Valid(Blockchain chain) {
    cout << "===============    VALID started    ===============" << endl;
    for (int i = 1; i < chain.GetBlockchainHeight() - 1; i++) {                                    /* Tx에 대한 검증 추가 예정 */
        if (chain.m_blocks[i].header.prevHash != GetBlockHash(chain.m_blocks[i - 1])               // 블록의 prevHash값이 이전 블록으로부터 생성되었는지
            || m_secureWorker.NonceInProof(chain.m_blocks[i].body.proof) != GetHeaderHash(chain.m_blocks[i].header)   // proof내의 nonce값이 올바르게 생성되었는지
            || chain.m_blocks[i].body.proof                                                        // nonce와 luck으로부터 올바른 proof가 생성되었는지
                != m_secureWorker.RemoteAttestation(m_secureWorker.NonceInProof(chain.m_blocks[i].body.proof), m_secureWorker.LuckInProof(chain.m_blocks[i].body.proof))) {
            NS_LOG_INFO("VALID failed");
            return;
        }
    }
    NS_LOG_INFO("VALID success");
}
```

소스 코드

PoL Implementation

9) 검증(VALID)에 통과하였을 경우 Base58 decoding을 통해 proof내에 포함된 Luck값 추출

→ 체인 내의 모든 블록에 대한 Luck값을 추출하여 합산한 후 리턴

```
// =============== 체인 내의 모든 블록들의 Luck값의 합 계산 =============== //
float
PoL::Luck(Blockchain chain) {
    float chainLuck = 0;

    for (int i = 0; i < chain.GetBlockchainHeight(); i++) {
        chainLuck += m_secureWorker.LuckInProof(chain.m_blocks[i].body.proof);
    }

    return chainLuck;
}
```

소스 코드

PoL Implementation

10) 브로드캐스팅 과정 (수신)

→ 체인이 변조되지 않았는지 검증 수행

→ Base58 decoding을 통해

받은 체인의 luck값을 계산하여 비교

→ 받은 체인이 더 클 경우 해당 체인으로 교체

→ 모든 노드가 동기화 된 후 다음 라운드 수행

```
========== Node[1] RECEIVED chain ==========

received chain's[0]th Block: {
   prevHash: 0000000000000000000000000000000000000000000000000000000000000000
   tx: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
   proof: iFiY4cAcuEq6c5Bx2QKnja3XGadaghm9zRcmtgTrgMQU6vRCr56xhUY5xA5gsZ6ZQ96THB5LizwKNtWbVNsYY4sx1Pw2XUTzBh00
}

received chain's[1]th Block: {
   prevHash: 1acb2f94d24dcdcaeaaa46445e862113beac857d79ac5e316aab6b666551831d
   tx: 9fda737d78e62f1c2446b3ccac6f07a1299b876f7b812ea18a8debeb385f3c65fe9173018049ff5d9a01efdef39d
   proof: kpiabqJ32o4Fvf5RVNaEB87zoyCPzhtviS88L3d54keJYTCenSLs5qLZdcxS9T9kwstc81GbwFPvEsoc3wMmy4c92w2yvULgWG00
}

==========    VALID started    ==========
VALID success
current chain's luck: 0.606061
received chain's luck: 0.424242
================================================================================
================================================================================

========== Node[2] RECEIVED chain ==========

received chain's[0]th Block: {
   prevHash: 0000000000000000000000000000000000000000000000000000000000000000
   tx: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
   proof: iFiY4cAcuEq6c5Bx2QKnja3XGadaghm9zRcmtgTrgMQU6vRCr56xhUY5xA5gsZ6ZQ96THB5LizwKNtWbVNsYY4sx1Pw2XUTzBh00
}

received chain's[1]th Block: {
   prevHash: 1acb2f94d24dcdcaeaaa46445e862113beac857d79ac5e316aab6b666551831d
   tx: 9fda737d78e62f1c2446b3ccac6f07a1299b876f7b812ea18a8debeb385f3c65fe9173018049ff5d9a01efdef39d
   proof: kpiabqJ32o4Fvf5RVNaEB87zoyCPzhtviS88L3d54keJYTCenSLs5qLZdcxS9T9kwstc81GbwFPvEsoc3wMmy4c92w2yvULgWG00
}

==========    VALID started    ==========
VALID success
current chain's luck: 0.979798
received chain's luck: 0.424242
================================================================================
================================================================================

========== Node[3] RECEIVED chain ==========

received chain's[0]th Block: {
   prevHash: 0000000000000000000000000000000000000000000000000000000000000000
   tx: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
   proof: iFiY4cAcuEq6c5Bx2QKnja3XGadaghm9zRcmtgTrgMQU6vRCr56xhUY5xA5gsZ6ZQ96THB5LizwKNtWbVNsYY4sx1Pw2XUTzBh00
}

received chain's[1]th Block: {
   prevHash: 1acb2f94d24dcdcaeaaa46445e862113beac857d79ac5e316aab6b666551831d
   tx: 9fda737d78e62f1c2446b3ccac6f07a1299b876f7b812ea18a8debeb385f3c65fe9173018049ff5d9a01efdef39d
   proof: kpiabqJ32o4Fvf5RVNaEB87zoyCPzhtviS88L3d54keJYTCenSLs5qLZdcxS9T9kwstc81GbwFPvEsoc3wMmy4c92w2yvULgWG00
}

==========    VALID started    ==========
VALID success
current chain's luck: 0.808081
received chain's luck: 0.424242
================================================================================
```

검증 결과 및 luck 값 비교

실행 결과

PoL Implementation

10) 브로드캐스팅 과정 (수신)

→기존 체인보다 Luck값이 큰 체인을 수신받았으므로 해당 체인으로 교체

```
========== Node[0] RECEIVED chain ==========

received chain's[0]th Block: {
   prevHash: 0000000000000000000000000000000000000000000000000000000000000000
   tx: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
   proof: iFiY4cAcuEq6c5Bx2QKnja3XGadaghm9zRcmtgTrgMQU6vRCr56xhUY5xA5gsZ6ZQ96THB5LizwKNtWbVNsYY4sx1Pw2XUTzBh00
}

received chain's[1]th Block: {
   prevHash: 1acb2f94d24dcdcaeaaa46445e862113beac857d79ac5e316aab6b666551831d
   tx: 7d7196bd6fa914496b1e7e9eaae574ce7daec0957b2a88c3f1ae8c6115af2342ff70ef06f9c33dca8e59bdb3133e
   proof: j7xWMyeA771HHsiuQbGp99PbQWgXH4eEztqg5qQU9PsdppqxNkWED4W4Nf9mXDFmYeBqECyrT9QQQx3uV5V3a7RUaTiLixMru00
}

==========    VALID started    ==========
VALID success
current chain's luck: 0.424242
received chain's luck: 0.606061
================================================================================
* chain has changed! *

current chain's[0]th Block: {
   prevHash: 0000000000000000000000000000000000000000000000000000000000000000
   tx: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
   proof: iFiY4cAcuEq6c5Bx2QKnja3XGadaghm9zRcmtgTrgMQU6vRCr56xhUY5xA5gsZ6ZQ96THB5LizwKNtWbVNsYY4sx1Pw2XUTzBh00
}

current chain's[1]th Block: {
   prevHash: 1acb2f94d24dcdcaeaaa46445e862113beac857d79ac5e316aab6b666551831d
   tx: 9fda737d78e62f1c2446b3ccac6f07a1299b876f7b812ea18a8debeb385f3c65fe9173018049ff5d9a01efdef39d
   proof: kpiabqJ32o4Fvf5RVNaEB87zoyCPzhtviS88L3d54keJYTCenSLs5qLZdcxS9T9kwstc81GbwFPvEsoc3wMmy4c92w2yvULgWG00
}

changed chain's[0]th Block: {
   prevHash: 0000000000000000000000000000000000000000000000000000000000000000
   tx: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
   proof: iFiY4cAcuEq6c5Bx2QKnja3XGadaghm9zRcmtgTrgMQU6vRCr56xhUY5xA5gsZ6ZQ96THB5LizwKNtWbVNsYY4sx1Pw2XUTzBh00
}

changed chain's[1]th Block: {
   prevHash: 1acb2f94d24dcdcaeaaa46445e862113beac857d79ac5e316aab6b666551831d
   tx: 7d7196bd6fa914496b1e7e9eaae574ce7daec0957b2a88c3f1ae8c6115af2342ff70ef06f9c33dca8e59bdb3133e
   proof: j7xWMyeA771HHsiuQbGp99PbQWgXH4eEztqg5qQU9PsdppqxNkWED4W4Nf9mXDFmYeBqECyrT9QQQx3uV5V3a7RUaTiLixMru00
}
```

수신한 체인

기존의 체인보다 luck값이 더 큼

변경 전 체인

변경 후 체인

실행 결과

블록체인 합의 알고리즘 시뮬레이터 (PoET)

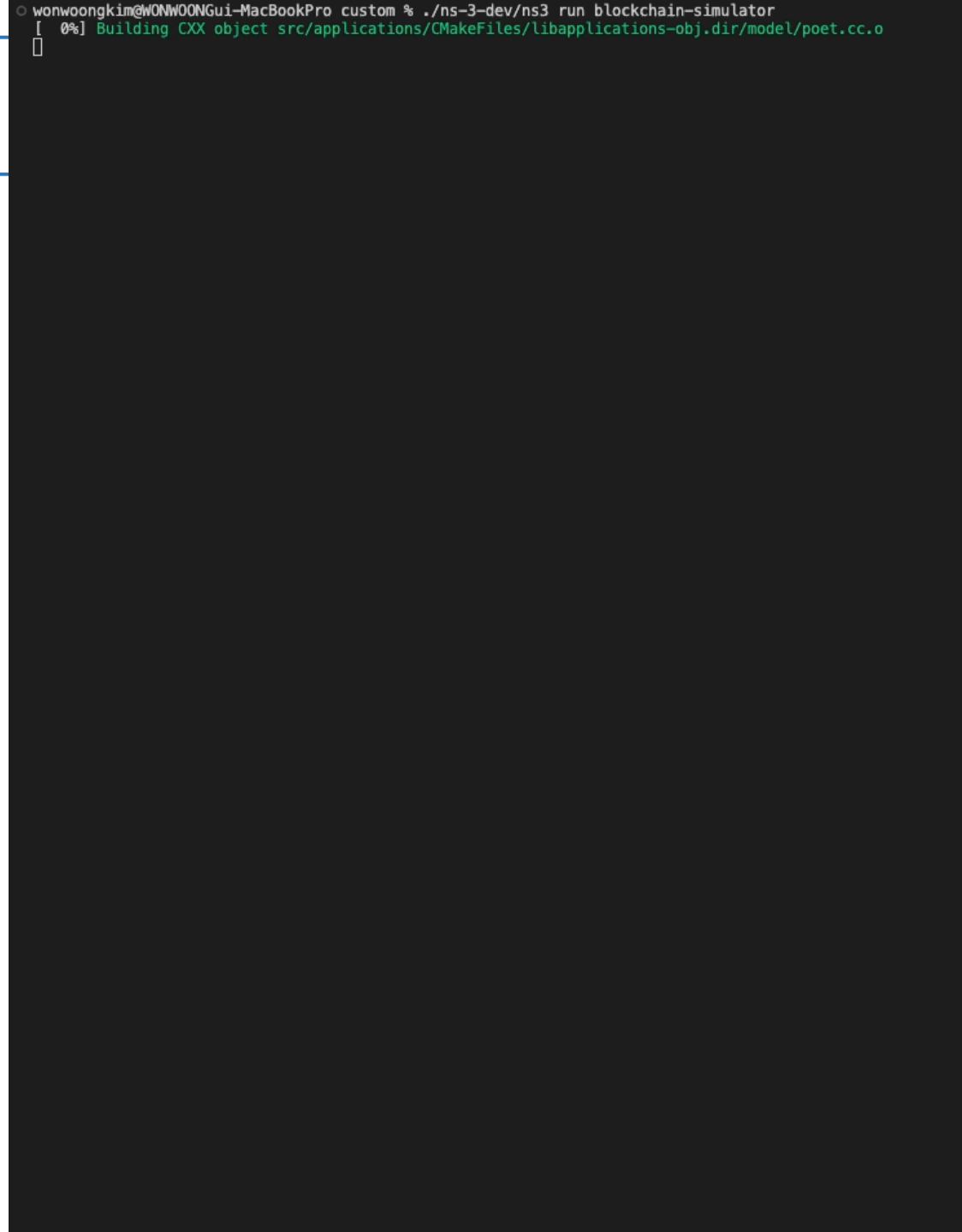
PoET Implementation

- **PoET 시연 영상**
  - **Node의 수**: 4

  - **GenesisBlock** {
    prevHash: 0000…0000
    Tx: 0000…0000
    }

prevHash: 이전 블록의 해시값

Tx: 트랜잭션



PoET Implementation

1) NewRound를 통해 채굴하고자 하는 newBlock 생성
   → 랜덤으로 생성된 Random Value에 따라 waitTime 생성
   → waitTime = minimumWait - localAverage * log(randomValue)

```
void
PoET::NewRound(void) {
    m_waitTime = GetWaitTime();

    cout << "Node[" << GetNode()->GetId() <<"]s waitTime: " << m_waitTime << endl;
    vector<uint8_t> prevHash = GetBlockHash(m_chain.GetLatestBlock());
    vector<uint8_t> tx = GetPendingTxs();
    Block newBlock(prevHash, tx);

    network.PrintBlock("newBlock", newBlock);
    m_eventId = Simulator::Schedule(Seconds(m_waitTime), &PoET::SendBlock, this, newBlock);

    cout << "==========================================================================================================" << endl;

    sleep(1);
}
```

GetWaitTime()을 통해 waitTime 생성

newBlock 생성

소스 코드

```
float
PoET::GetWaitTime(void) {
    float r = (float)dis(gen) / 99;                                    // r ∈ [0, 1]

    cout << "================================== Generate Random Value ==================================" << endl;


    cout << endl << "Random Value: " << r << endl;
    // return network.minimumWait - network.localAverage * log10(r);
    return network.minimumWait - network.localAverage * log(r);        // Random Wait Time 리턴
}                                                                      // Random Wait Time 리턴
```

Random Value 생성

waitTime 계산

```
================================== Generate Random Value ==================================

Random Value: 0.282828
Node[0]s waitTime: 15.9695
                                                                    0404d70577785316fa95218e8ba
                                                                    2451f77b701cf3d49c44d04ded4d8e9dfcec0b8c6958169eddfe
                                                                    02e8ca324a823ef9f1d701f6e51a237befe602392dcc69
================================== Generate Random Value ==================================

Random Value: 0.848485
Node[1]s waitTime: 6.42712
newBlock: {
   prevHash: 67f022195ee405142968ca1b53ae2513a8bab0404d70577785316fa95218e8ba
   tx: 25a4738be379978a7966314ff452434685f4e034a946c4d451f45db8cdb3c2af55bd17459ccb77fac4bac7c65365b87
576f1b622a2b1679c4d1e12e2abf54303e6bba12a928ee0f4cd9a03b30f11350c887d35c71df34da85749f7d7776581b
}
==========
==========

Random Value
Node[2]s waitTime: 6.12232
newBlock: {
   prevHash: 67f022195ee405142968ca1b53ae2513a8bab0404d70577785316fa95218e8ba
   tx: 30d25641d2ffb5524d1ba2883228875c3fc3e217acc543fcfa54ae35c928a98f8e3a7853c95d34e9551030d2419ec37
2e09229d693060efba6888d4edc88a1e49a1ea058176d0d59e54e94ad242bd1505843e7fd161630aa7388fe63044660a
}
==========
================================== Generate Random Value ==================================

Random Value: 0.646465
Node[3]s waitTime: 8.7891
newBlock: {
   prevHash: 67f022195ee405142968ca1b53ae2513a8bab0404d70577785316fa95218e8ba
   tx: 929d458d11ec2565272815cdb3d12fee54ef858d93fd2298295d5700c3442b795eea1ed6987a65eef11d00c30e2ff96
693b6948a1a612ba35ca62e5762836d64f89f3fb0beebe90e7edc39a00a03340f87c73bdc653e37cec2c52b8582c177a
}
==========
```

Random Value와 그에 따른 waitTime

이번 라운드에서 채굴하고자 하는 newBlock

실행 결과

PoET Implementation

2) 브로드캐스팅 과정 (송신)

소스 코드

```
// ============== 블록 브로드캐스팅 ============== //
void
PoET::SendBlock(Block block) {
    Ptr<Packet> p;

    uint8_t *data = network.Block2Array(block);                    // 블록을 브로드캐스팅하기 위해 패킷의 현태로 변경
    p = Create<Packet>(data, network.blockSize);                   // 패킷 생성

    network.fastestNode = GetNode()->GetId();

    cout << "========================================== Send Block ==========================================" << endl;
    cout << endl << "fastestNode: " << network.fastestNode << endl;
    network.PrintIntVector("winNum", network.winNum);

    network.winNum[network.fastestNode] += 1;
    if (ZTest())
        m_chain.Ad

        // ===== 모든 노드에게 브로드캐스팅 ===== //
        vector<Ipv4Address>::iterator iter = m_peersAddresses.begin();          // 시작 peer node부터
        while(iter != m_peersAddresses.end()) {                                 // 마지막 peer node까지
            Ptr<Socket> socketClient = m_peersSockets[*iter];
            Simulator::Schedule(Seconds(0), &PoET::SendPacket, this, socketClient, p);
            iter++;
        }

        Simulator::Schedule(Seconds(1), &PoET::NewRound, this);
    }

    else {
        network.winNum[network.fastestNode]--;
        cout << "Z-Test failed" << endl;
        return;
    }

    network.PrintChain("Fastest Chain", m_chain);
    cout << "================================================================================================"

    sleep(1);
}
```

winNum 증가

Z-Test 수행

블록 브로드캐스팅

→ 가장 작은 waitTime을 가진 노드가 자신의 newBlock 브로드캐스팅.

→ 승리 노드의 winNum 증가 (Z-Test에 사용)

→ Z-Test에 성공할 경우 블록 브로드캐스팅

실행 결과

```
========== Send Block ==========
fastestNode: 2
winNum: 0000
zScore: 1.73205
Fastest Chain's[0]th Block: {
   prevHash: 0000000000000000000000000000000000000000000000000000000000000000
   tx: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
}

Fastest Chain's[1]th Block: {
   prevHash: 67f022195ee405142968ca1b53ae2513a8bab0404d70577785316fa95218e8ba
   tx: 30d25641d2ffb5524d1ba2883228875c3fc3e217acc543fcfa54ae35c928a98f8e3a7853c95d34e9551030d2419ec37
2e09229d693060efba6888d4edc88a1e49a1ea058176d0d59e54e94ad242bd1505843e7fd161630aa7388fe63044660a
}
```

승리 노드의 ID와 Z-Test를 위한 Z-Score

승리 노드의 체인

PoET Implementation

- 브로드캐스트 (수신)

  → 송신자에 대한 Z-Test 수행

    → 송신자의 승리횟수에 따라 Z-Score 계산

    → 일정 임계값을 넘지 않았을 경우 통과

  → Z-Test에 통과한 경우 수신한 블록 추가

소스 코드

```
void
PoET::HandleRead(Ptr<Socket> socket) {
    Ptr<Packet> packet;
    Address from;
    string msg;

    // ===== packet에 대한 동작 과정 ===== //
    while((packet = socket->RecvFrom(from))) {
        socket->SendTo(packet, 0, from);
        if (packet->GetSize() == 0) { break; }                    // 패킷이 비어있는지 검증

        if (InetSocketAddress::IsMatchingType(from)) {
            msg = GetPacketContent(packet, from);                 // 수신받은 패킷의 내용 Get
            Block block = network.Array2Block(msg);

            if(ZTest()) {
                Simulator::Cancel(m_eventId);
                m_chain.AddBlock(block);
            }

            cout << "========================================= Receive Block ========================================" << endl;

            network.PrintChain("Node " + to_string(GetNode()->GetId()), m_chain);
            Simulator::Schedule(Seconds(0), &PoET::NewRound, this);

            cout << "================================================================================================" << endl;

            sleep(1);
        }
    }
}
```

Z-Test 수행 및 체인에 블록 추가

```
// ========== 노드가 비정상적으로 블록을 많이 생성했는지에 대해 검증하는 함수 ========== //
bool
PoET::ZTest(void) {
    int m = m_chain.GetBlockchainHeight();
    float p = 1.0 / network.numNodes;
    float zScore = (network.winNum[network.fastestNode] - m * p) / sqrt(m * p * (1 - p));

    NS_LOG_INFO("zScore: " << zScore);                            // z-score 출력

    if (zScore <= network.zMax) { return true; }
    else { return false; }
}
```

승리횟수에 따른 Z-Score 계산

Z-Score가 임계값을 넘지 않았을 경우 검증 통과

PoET Implementation

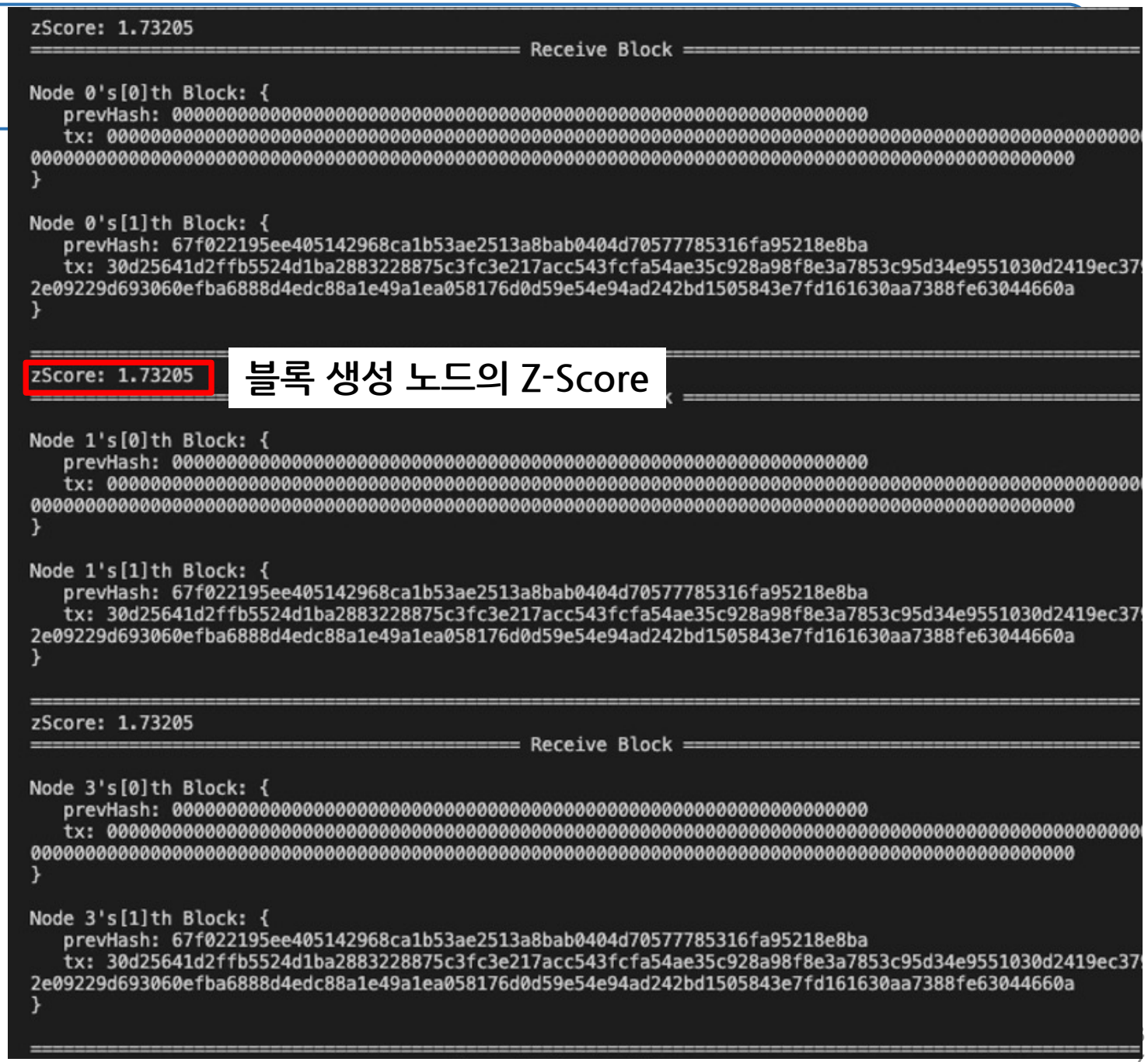
- 브로드캐스트 (수신)

  → 송신자에 대한 Z-Test 수행

    → 송신자의 승리횟수에 따라 Z-Score 계산

    → 일정 임계값을 넘지 않았을 경우 통과

  → Z-Test에 통과한 경우 수신한 블록 추가



실행 결과

향후 계획

- **트랜잭션 클래스화** (input, output, 코인 전송 내용, 전자서명 공개키)
- 전자서명: **CRYSTALS-Dilithium**
- 트랜잭션 발생
- **메모리풀** 구현
- 트랜잭션 **머클트리**
- **타임스탬프**
- **성능 측정** (PoW, PoS, PoL, PoET)
    - Throughput(TPS), Latency, 네트워크 크기에 따른 Delegate 수 등등..

참고문헌

[1] Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system." *Decentralized business review* (2008): 21260.

[2] Chen, Lin, et al. "On security analysis of proof-of-elapsed-time (poet)." *International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Springer, Cham, 2017.

[3] Milutinovic, Mitar, et al. "Proof of luck: An efficient blockchain consensus protocol." *proceedings of the 1st Workshop on System Software for Trusted Execution*. 2016.

[4] King, Sunny, and Scott Nadal. "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake." *self-published paper, August* 19.1 (2012).

[5] Zheng, Zibin, et al. "Blockchain challenges and opportunities: A survey." *International journal of web and grid services* 14.4 (2018): 352-375.

[6] Fan, Xinxin, and Qi Chai. "Roll-DPoS: a randomized delegated proof of stake scheme for scalable blockchain-based internet of things systems." *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. 2018.

[7] Begicheva, A., and A. Kofman. "Fair proof of stake." *Fair Block Delay Distribution, in Proof-of-Stake Project; Waves Platform: Moscow, Russia* (2018).

참고문헌

[8] https://nemproject.github.io/nem-docs/pages/Whitepapers/docs.en.html
[9] Castro, Miguel, and Barbara Liskov. "Practical byzantine fault tolerance." *OsDI*. Vol. 99. No. 1999. 1999.
[10] Zheng, Zibin, et al. "Blockchain challenges and opportunities: A survey." *International journal of web and grid services* 14.4 (2018): 352-375.
[11] Mazieres, David. "The stellar consensus protocol: A federated model for internet-level consensus." *Stellar Development Foundation* 32 (2015): 1-45.
[12] Kwon, Jae. "Tendermint: Consensus without mining." *Draft v. 0.6, fall* 1.11 (2014).
[13] Biswas, Sujit, et al. "PoBT: A lightweight consensus algorithm for scalable IoT business blockchain." *IEEE Internet of Things Journal* 7.3 (2019): 2343-2355
[14] Nie, Zixiang, Maosheng Zhang, and Yueming Lu. "HPoC: A Lightweight Blockchain Consensus Design fo r the IoT." *Applied Sciences* 12.24 (2022): 12866.

들어주셔서 감사합니다.