경량 합의 알고리즘 설계 방향성

https://youtu.be/JgTZBXrpl70

합의 알고리즘 서베이

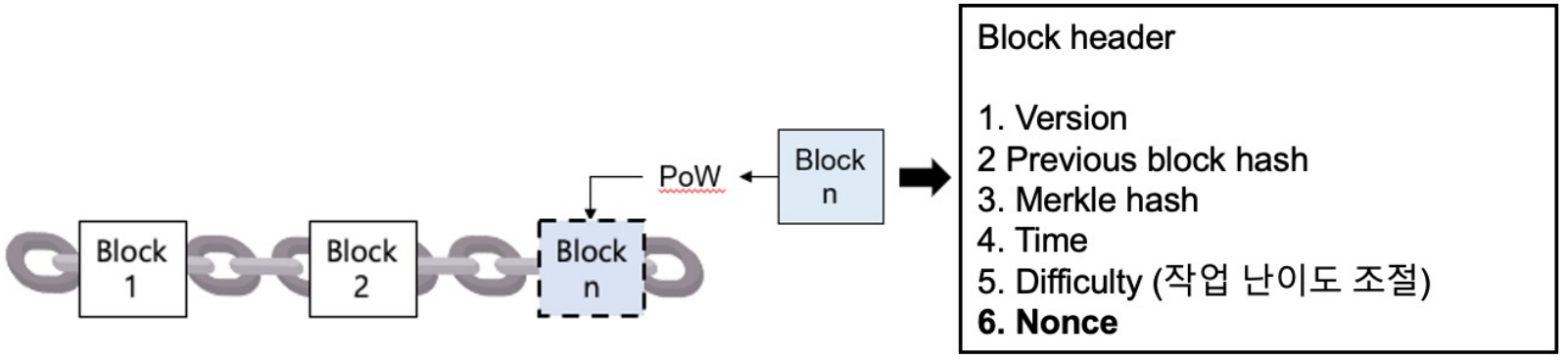
경량 합의 알고리즘 설계_PBFT 기반

PoW & PoS 구현

합의 알고리즘 서베이

PoW (Proof-of-Work)

- **PoW (Proof-of-Work)**[1]
  - 블록의 **특정 해시값**을 찾음으로써 합의에 도달하는 알고리즘
  - 블록 헤더에 속하는 6개의 정보 중, 특정 조건을 만족하는 유효한 **Nonce** 값을 찾는 문제
    → 채굴 (컴퓨팅 자원을 소모했다는 증명)
  - 블록 헤더의 Difficulty라는 요소로 **난이도 조절** 가능 → 블록의 해시값이 특정값 보다 작아야 함
  - 채굴자가 유효한 Nonce 값을 찾은 후에 다른 채굴자들이 해당 Nonce 값이 유효한지 검증
    → 이러한 검증 과정을 거친 후, 블록이 체인에 추가되고 채굴자는 보상을 받음
  - **채굴 과정에서 과도한 에너지 소비가 필요**



PoS (Proof-of-Stake)

- **PoS (Proof-of-Stake)**[4]
  - PoW의 **무의미한 컴퓨팅 자원소모 문제**를 해결하기 위한 합의 알고리즘
  - 노드가 시스템에 충분한 지분을 가지고 있다는 아이디어에 따라 작동

    → 해당 네트워크에 충분한 투자를 한 것이기 때문에, 악의적인 시도를 할 경우 이점보다 잃는 것이 더 많음
  - 작업증명(PoW)과 다르게 **채굴 과정 필요 X**

    → 에너지 소모 X
  - 지분율이 높은 노드가 블록을 생성할 가능성이 높기 때문에 **부익부 빈익빈 문제 야기**
  - **네트워크 내에 지분으로 사용할 암호화폐가 반드시 필요**

DPoS (Delegated-Proof-of-Stake)

- **DPoS (Delegated-Proof-of-Stake)**[5]
  - PoS를 기반으로 한 합의 알고리즘으로 투표를 통해 대표자를 선출함으로써 권한 위임
  - 대표로 선출된 노드들이 PoS 합의 알고리즘 수행

    → **속도가 빠르고 효율이 좋음**
  - 자신의 권한을 위임해 대표자를 선출한다는 측면에서 **민주주의적**

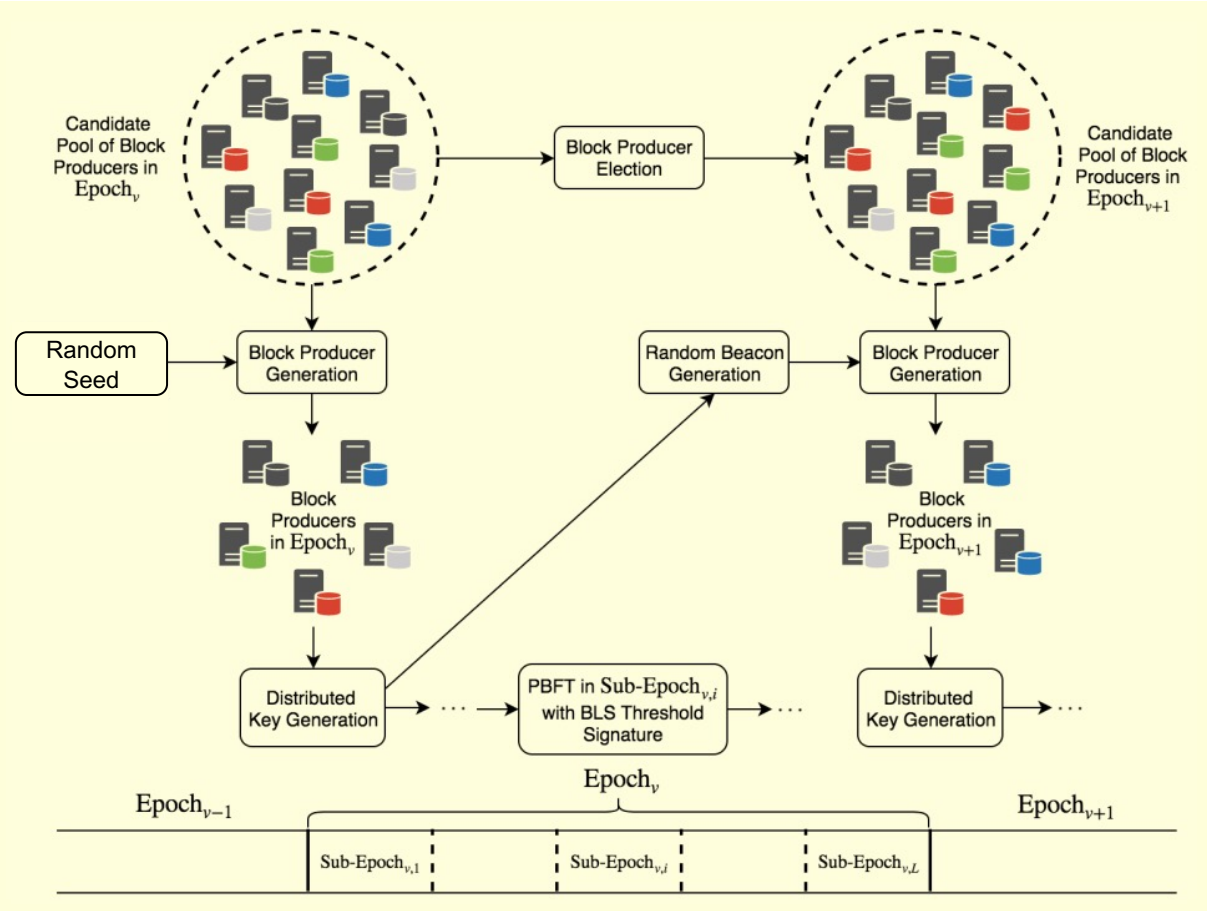
    → 코인을 많이 보유하지 않아도 대표자로 선출될 수 있기 때문에 PoS의 **부익부 빈익빈 문제가 발생하지 않음**
  - **네트워크의 크기에 비해 대표자 수가 적다면 중앙집중화 문제가 발생할 수 있음**

    → 소수의 블록 검증자들이 담합할 가능성 존재

Roll-DPoS

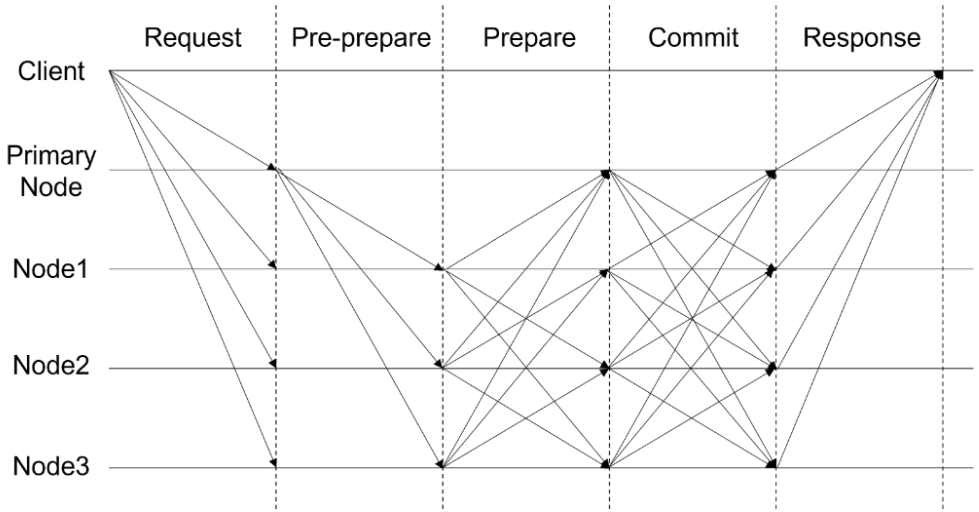
PBFT 합의 과정

- **Roll-DPoS**[6]
  1. 투표를 통해 블록을 생성하게 될 잠재적 블록 생성자 선출
     → **위임을 통한 확장성 향상**
  2. 난수를 통해 랜덤으로 Candidate Pool에서 블록 생성자 생성
     → **난수를 통해 공평한 블록생성자 지정**
  3. 블록 생성자들끼리 DKG 방식으로 키 생성
     → PBFT 합의 과정에서 서명할 때 사용
     → 다음 epoch에서 필요한 랜덤 seed를 생성할 때 사용
  4. BLS 임계값 서명을 통해 PBFT 방식으로 블록 생성 및 검증
     → 각 Sub-Epoch에서 블록 생성자들이 교대로 블록을 제안



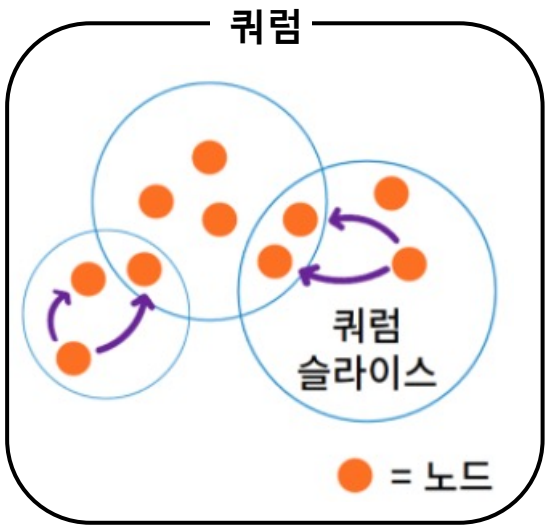
PBFT (Practical Byzantine Fault Tolerance)

- **PBFT (Practical Byzantine Fault Tolerance)**[9]
  - PBFT는 **비잔틴 장군 문제를 해결**하기 위한 합의 알고리즘
  - 2번의 브로드 캐스트를 통해 **악의적인 노드가 네트워크 내에 존재하여도 정상적인 합의에 도달**할 수 있는 알고리즘
    → $O(n^2)$의 시간복잡도를 가짐
  - 네트워크 내에 악의적인 노드가 f개일 경우 총 노드의 개수가 $3f + 1$개 이상일 경우 정상적으로 합의 가능
  - **네트워크의 크기가 커질 수록 합의에 도달하는 속도가 상대적으로 비효율적**
    → 2번의 브로드 캐스트로 인해 통신비용이 크게 증가하고 이에 따라 확장성 저하



SCP (Stellar Consensus Protocol)

- **SCP (Stellar Consensus Protocol)**[11]
  - FBA (Federated Byzantine Agreement)를 기반으로 만들어진 합의 알고리즘
    → FBA는 개인 노드 간이 아닌 노드의 집합 간(쿼럼 슬라이스 *) 합의를 이루는 방식
    → 노드가 누구를 신뢰할 것인지 직접 선택을 하고 이를 바탕으로 형성된 신뢰망을 이용하여 합의에 도달하는 방식
  - SCP는 Nomination protocol 단계와 Ballot protocol 단계로 나뉨
    1. **Nomination protocol 단계 :** 연합투표를 통해 네트워크에서 처리할 수 있는 후보 트랜잭션 집합을 선택하기 위한 프로토콜
    2. **Ballot protocol 단계 :** 후보 트랜잭션 집합에 대한 상태를 확실하게 합의하기 위한 프로토콜



쿼럼 : 합의를 이루기에 충분한 수의 노드 집합
쿼럼 슬라이스 : 각 노드들이 신뢰하는 노드들과 자기 자신을 의미 (노드들의 집합)

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

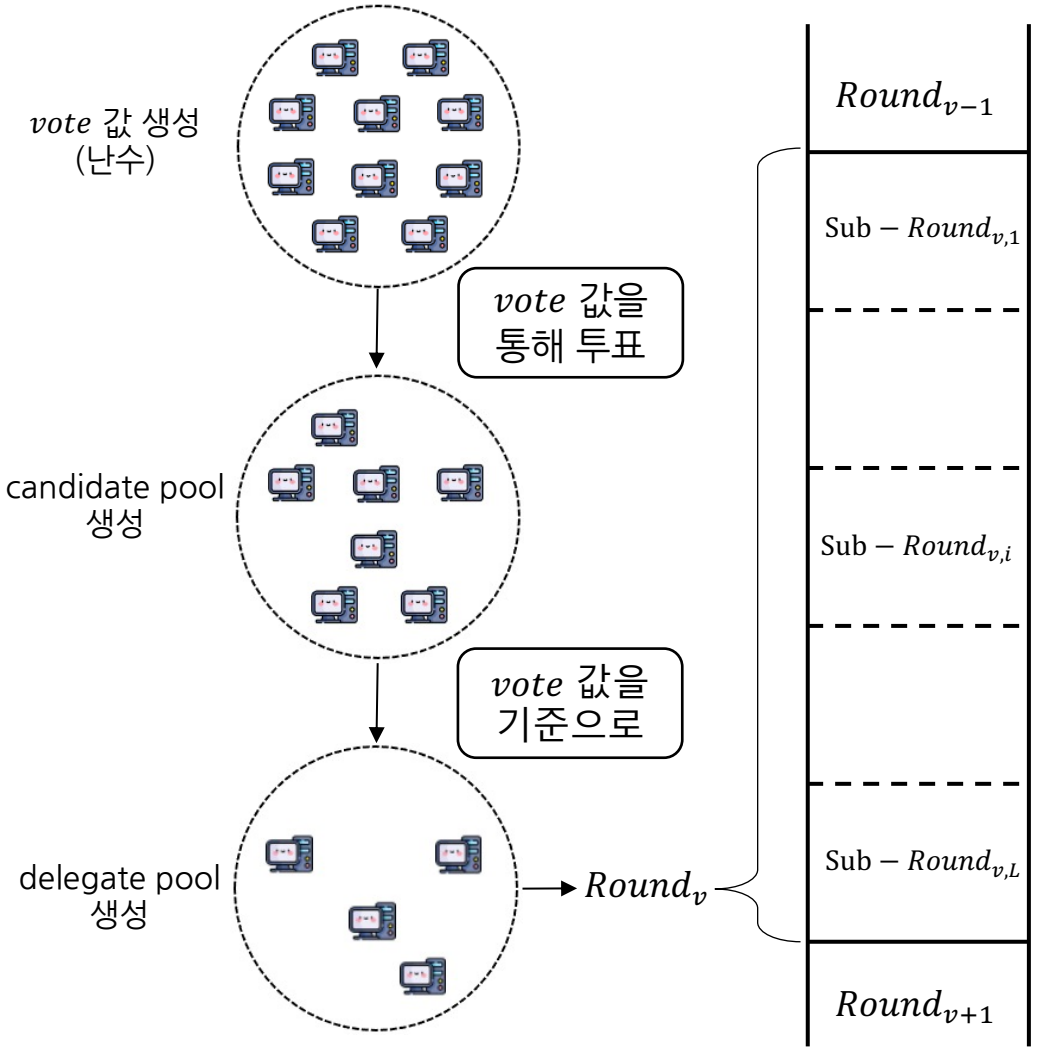
Zheng, Zibin, et al. "Blockchain challenges and opportunities: A survey." *International journal of web and grid services* 14.4 (2018): 352-375.

경량 합의 알고리즘 설계 2 (PBFT 기반)

PoM_PBFT (Proof of Merge_PBFT)

- **Overview of the PoM_PBFT**

1. 각 노드들이 **난수를 생성**하여 $vote$ 값으로 사용
2. delegate로 선출하고자 하는 candidate에게 $vote$ 값을 전송 (**투표**)
3. 각 candidate 들은 전송 받은 $vote$ 값을 모두 더함 (***sumVote***)
4. $sumVote$ 값이 큰 상위 M명의 candidate 선정 (**candidate pool**)
5. 각 candidatate들이 생성했던 $vote$에 따라 상위 N명을 delegate로 선정
6. N명의 delegate들이 순서대로 블록 생성
   1) sub-round마다 1명의 delegate가 **PBFT 방식으로 블록 생성**
   2) 다른 delegate들은 블록에 대한 검증 수행
   3) PBFT가 끝나면 다음 sub-round 진행
7. 모든 delegate들이 블록을 생성했을 경우 라운드(round) 종료



PBFT in $Sub - Round_{v,i}$
with CRYSTAL-Dilithium

PoM_PBFT (Proof of Merge_PBFT)

1. **Vote 값 생성 :** Verifiable Random Function을 통해 vote 값 생성

   → $0 < vote \leq 1$ (1에 가까울 수록 좋음)

   → 난수를 vote 값으로 사용함으로써 **중앙 집중화 완화**

   → 해당 값은 delegate를 선출할 때도 사용

VRF (Verifiable Random Function) : 생성된 **난수 값이 유효한지 검증 가능한 함수**

- **Keygen(r) → (VK, SK)** : 임의의 입력에서 키 생성 알고리즘은 검증 키 VK와 비밀 키 SK로 이루어진 키 쌍을 생성
- **Evaluate(SK, X) → (Y, ρ)** : 평가 알고리즘은 비밀 키 SK와 메시지 X를 입력하여 의사 난수 Y와 증명 ρ를 생성
- **Verify(VK, X, Y, ρ) → 0/1** : 검증 알고리즘은 검증 키 VK, 메시지 X, 난수 Y, 증명 ρ를 입력

  → Y가 SK와 X에 대한 평가 알고리즘에서 생성된 난수임이 검증된 경우에만 1이 출력

PoM_PBFT (Proof of Merge_PBFT)

2. Delegate 선출

1) 후보자 등록 : 블록 생성자가 되고 싶은 노드들은 candidate 등록을 하고 투표를 받기 위해 홍보

→ 예시) 자신의 홍보 웹사이트에 블록을 생성할 수 있는 SW/HW 스펙 명시

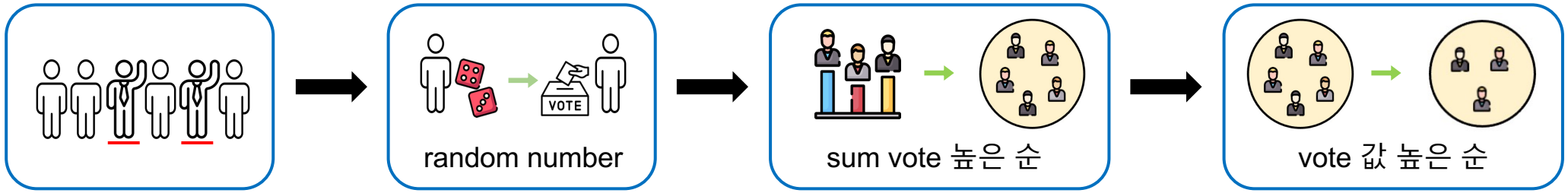
2) 투표 시작 : 생성하고자 하는 블록의 헤더를 전송함으로써 투표 (트랜잭션의 형태, $voteTx$)

→ voter들은 생성하고자 하는 블록의 헤더에 vote 값을 포함 (Merging을 위한 과정)

3) Candidate pool 생성 : 투표 시간이 끝나고 집계하여 $sumVote$ 값이 큰 상위 N명의 candidate들로 구성된 candidate pool 생성

→ $sumVote$ 값은 투표 받은 vote 값의 합

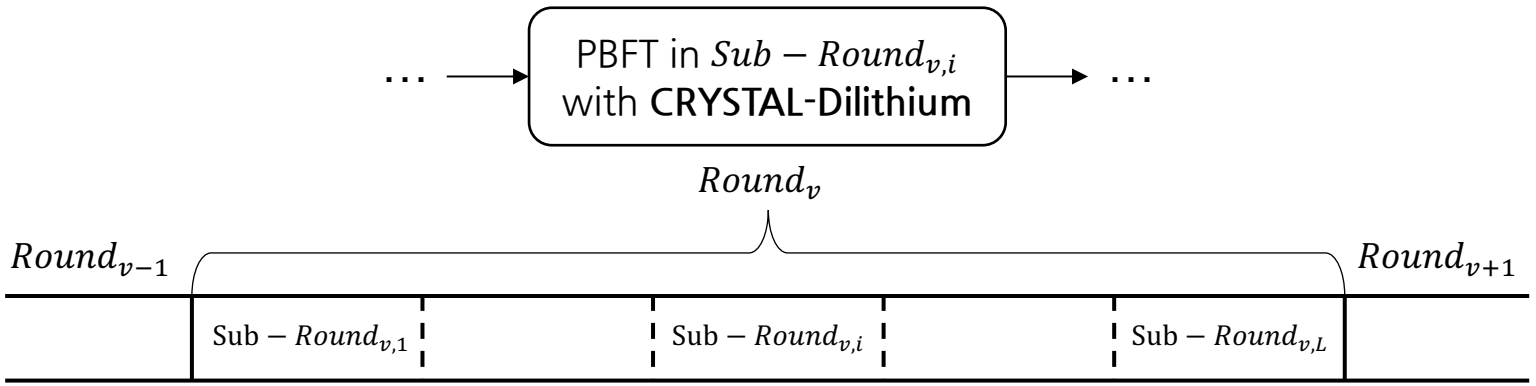
4) Delegate pool 생성 : 각 candidate들이 생성했던 $vote$에 따라 상위 N명을 delegate로 선정



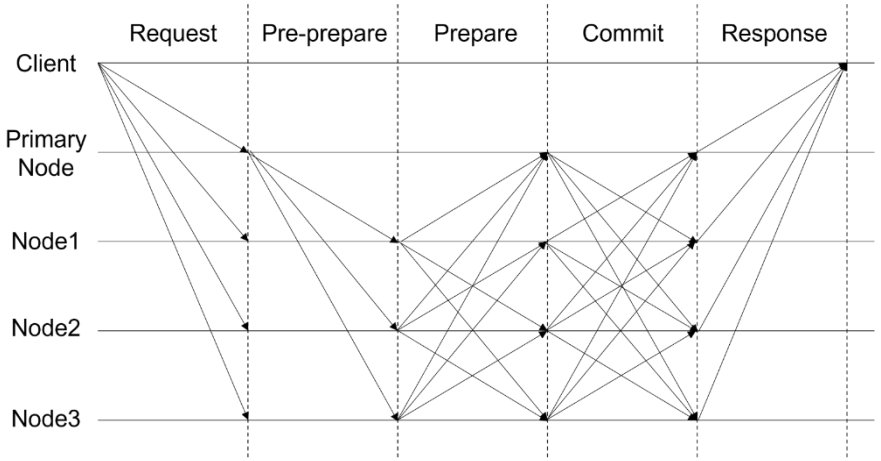
PoM_PBFT (Proof of Merge_PBFT)

3. CRYSTALS-Dilithium 서명을 통한 PBFT 합의

- Delegate는 각 sub-round에서 교대로 블록을 제안
  → 모든 delegate들이 블록을 생성했을 경우 라운드(round) 종료
  → voter들의 블록 전체를 merging함으로써 블록 병합
  → 각 sub-round마다 블록 생성자는 한 명이고, 다른 delegate는 블록 검증자가 됨
  → Delegate끼리의 PBFT(dBFT)를 통해 **Scalability 향상**
  → PBFT를 통한 **비잔틴 장군 문제 해결**



Dilithium 서명을 통한 PBFT



PBFT 합의과정

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 (PoW)

PoW 시연 영상

Difficulty : 3

PoW **채굴 난이도**에 따른 속도 비교

Difficulty : 4

블록의 헤더

블록의 바디
(트랜잭션)

블록의
메소드 및 멤버변수

```
class Block
    {
        public:
            // ===== 생성자 & 소멸자 ===== //
            Block(void);
            Block(int blockId, string prevHash, int difficulty, string nonce, vector<int> tx);
            virtual ~Block(void);

            class BlockHeader {
                public:
                    BlockHeader(void);
                    virtual ~BlockHeader(void);

                    // ===== 변수 ===== //
                    int             blockId;
                    string          prevHash;
                    int             difficulty;
                    string          nonce;
            };

            class BlockBody {
                public:
                    BlockBody(void);
                    virtual ~BlockBody(void);

                    // ===== 변수 ===== //
                    vector <int>    tx;
            };

            // ===== 함수 ===== //
            void            PrintBlock(void);
            int             GetBlockSize(void);
            string          GetBlockHash(void);

            // ===== 변수 ===== //
            BlockHeader     header;
            BlockBody       body;
    };
```

Block의 구조

```
class Blockchain {
    public:
        // ===== 생성자 & 소멸자 ===== //
        Blockchain(void);
        virtual ~Blockchain(void);

        // ===== 함수 ===== //
        void            AddBlock (Block newBlock);
        void            PrintBlockchain(int round);
        void            PrintChain(void);
        int             GetBlockChainHeight(void);

        // ===== 변수 ===== //
        vector<Block>   m_blocks;
};
```

Blockchain의 구조

PoW Implementation

제네시스 블록 생성 과정

제네시스 블록의 prevHash 초기화

```
노드 0 : Add GenesisBlock
노드 1 : Add GenesisBlock
노드 2 : Add GenesisBlock
노드 3 : Add GenesisBlock
```

round가 0일 경우 제네시스 블록 생성

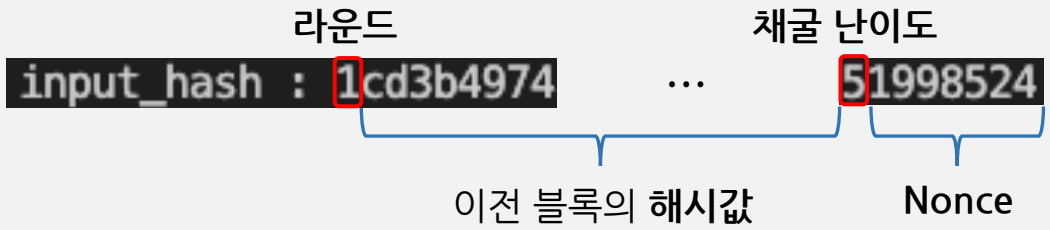
```
string Network::GetGenesisPrevHash (void)
{
  string Genesis_prevHash = "";

  for (int i = 0; i < 64; i++)
    {
      // 제네시스 블록은 이전 블록의 해시값이 존재하지 않기 때문에 임의로 0으로 초기화
      Genesis_prevHash = Genesis_prevHash + "0";
    }

  return Genesis_prevHash;
}
```

```
if(round == 0)
{
    NS_LOG_INFO("노드 " << m_id << " : Add GenesisBlock");
    int random_tx = 0;
    for (int i = 0; i < network.tx_size; i++)
    {
        vectTX.push_back (random_tx);
    }
    prevHash = network.GetGenesisPrevHash ();
}
```

PoW Implementation

PoW 채굴 과정

블록의 헤더 값을 연접하여 해시값 계산




```
inputHash = hRound + hPrevHash + hDifficulty + hNonce; // 헤더 연접 완성
bool checkmine = MiningBlock(inputHash, nonce, mineTime);

if(!checkmine)
{
    mineTime++;
    nonce++;
    m_miningId = Simulator::Schedule (Seconds(1),
                 &PoW::RunPoW, this, nonce, mineTime); // 채굴 시작
}
else
{
    return;
}
```

Nonce 값을 바꿔가며 반복해서 채굴

해시값이 목표값을 만족하는지 체크

난이도(4)를 만족하는 해시값 (앞에서부터 0의 개수)

```
bool Network::CheckHash (string output)
{
  for (int i = 0; i < network.difficulty; i++)
    {
      if (output[i] != '0')
        {
          return false;
        }
    }
  return true; // 채굴 성공
}
```

해시값이 목표값을 만족할 경우 블록 생성 후 브로드캐스트

```
Block block = GenerateBlock (nonce);
m_Blockchain.AddBlock (block); // 체인에 블록 입력

uint8_t *arr_block = network.Block2Arr (block);
SendBlock(arr_block, network.GetBlockSize(block));
```

PoW Implementation

채굴 성공 시 출력 결과
(difficulty : 4)

입력한 해시값이 목표값을 만족

블록 생성 후 브로드 캐스트

```
********** Round : 3 **********
< Node 1 : 채굴에 성공하였습니다. >
input_hash : 300007f0d953c93bf71a34ae175d64156004a5b54097181a3aa9c376d8ec724884116852
output_hash : 0000f4013d42349ba8c4b33a6fb0bb6018c30f36cec6ae6e4bc68a5068c0aa04
< Node 1 : 검증을 위해 브로드 캐스트 하였습니다. >
노드1이 전송한 블록(Round 3)
Block 3
{
   BlockId : 3
   PrevHash : 00007f0d953c93bf71a34ae175d64156004a5b54097181a3aa9c376d8ec72488
   Difficulty : 4
   Nonce : 116852
   Transaction : 0089884477
}
```

Nonce 값

라운드 / 채굴 난이도

input_hash : 300007f0d … 4116852

이전 블록의 **해시값** / Nonce

output_hash : 0000 … c0aa04

난이도(4)를 만족하는 해시값 (앞에서부터 0의 개수)

PoW Implementation

전송 받은 블록 검증

전송 받은 블록의 해시값이
목표값을 만족

전송 받은 블록

```
*** Node 2 : 검증한 결과 유효한 블록입니다. ***
input_Hash : 300007f0d953c93bf71a34ae175d64156004a5b54097181a3aa9c376d8ec724884116852
output_Hash: 0000f4013d42349ba8c4b33a6fb0bb6018c30f36cec6ae6e4bc68a5068c0aa04
Block
{
   BlockId : 3
   PrevHash : 00007f0d953c93bf71a34ae175d64156004a5b54097181a3aa9c376d8ec72488
   Difficulty : 4
   Nonce : 116852
   Transaction : 0089884477
}
```

Nonce 값

전송 받은 블록의 해시값이
목표값을 불만족

전송 받은 블록

```
*** Node 3 : 검증한 결과 유효하지 않은 블록입니다. ***
input_Hash : 14d9f3d0fb63010232919f8598e86dd63095bf4f5c19ffcf32a46b82286c2110747341411
output_Hash: 723d27b77da831695bbd971a3e23e79858e6fb6372ab1cdd1081bc573916b3f5
Block
{
   BlockId : 1
   PrevHash : 4d9f3d0fb63010232919f8598e86dd63095bf4f5c19ffcf32a46b82286c21107
   Difficulty : 4
   Nonce : 734141
   Transaction : 8718643873
}
```

Nonce 값

블록체인 합의 알고리즘 시뮬레이터 (PoS)

PoS Implementation

- **PoS 합의 알고리즘 동작 과정**

1. 각 노드들이 블록 생성자가 되어 보상을 얻기 위해 코인을 *스테이킹
2. **가장 큰 CoinAge를 보유하고 있는 노드가 블록 생성자가 됨**

→ CoinAge = 스테이킹한 코인 * 스테이킹한 시간

3. 블록 생성자는 블록을 생성 후 해당 블록을 브로드캐스트
4. 블록을 전송받은 노드가 체인에 블록 추가

→ 이때 블록 생성자의 **스테이킹한 시간은 0으로 초기화 됨**

*스테이킹 : 채굴을 위해 자신이 보유한 암호화폐를 예치하고 보상을 받음

PoS Implementation

- 각 노드에게 코인 할당 및 스테이킹 후, 코인나이 브로드캐스트

```
void PoS::CoinAllocation(void)
{

    // 1부터 100사이의 코인을 할당 받음
    coin_volume = dis(gen) + 1;

    // 스테이킹할 비율
    double coin_ratio = dis(gen) + 1;

    // 스테이킹할 코인 개수
    Staking_Volume = coin_volume * coin_ratio / 100;

    // 보유한 코인에서 스테이킹한 코인 빼기
    coin_volume = coin_volume - Staking_Volume;

    // 스테이킹 일수
    Staking_day = 1;

    CoinageCalc();
}
```

1부터 100 사이의 코인을 임의로 할당

코인 나이 계산 및 브로드캐스트

```
노드0 : 할당 받은 코인 개수 : 80
노드0 : 스테이킹한 코인 개수 : 35

노드1 : 할당 받은 코인 개수 : 98
노드1 : 스테이킹한 코인 개수 : 66

노드2 : 할당 받은 코인 개수 : 89
노드2 : 스테이킹한 코인 개수 : 69

노드3 : 할당 받은 코인 개수 : 40
노드3 : 스테이킹한 코인 개수 : 8
```

실행 결과

PoS Implementation

- 가장 큰 코인 나이를 가지고 있는 노드가 블록을 생성하고 브로드캐스트

블록 생성자는 코인나이가 0으로 되고
보상으로 5코인을 지급 받음

```
if (max_value == int_CoinAge)
  {
    // 여기서 블록 생성해서 블록 브로드캐스트
    uint8_t *block = generateBlock (int_CoinAge);
    SendBlock(block);

    // 블록생성자는 스테이킹 시간 0으로 초기화
    Staking_day = 0;

    // 블록 생성 보상
    coin_volume += 5;

    // 라운드 +1
    Round++;

    // 다시 코인 스테이킹
    Simulator::Schedule (Seconds (timeout), &PoSNode::StakingCoin, this);
  }
```

```
노드0 : 코인 나이(0)를 브로드캐스트 하였습니다.
노드2 : 코인 나이(124)를 브로드캐스트 하였습니다.
노드3 : 코인 나이(44)를 브로드캐스트 하였습니다.
노드1 : 코인 나이(20)를 브로드캐스트 하였습니다.
```

```
========== 블록 생성 ==========
블록 나이가 가장 높은 사람이 블록을 생성하였습니다.
노드2 : 블록을 생성하여 전송하였습니다.
노드2이 전송한 블록(Round 243)
Block 243
{
    BlockId : 243
    PrevHash : 3dd8474d3de2c76252bbd58e09e2d7314fb1a1
    CoinAge : 124
    Transaction : 8585175176
}
```

가장 큰 코인 나이 값을 가지고 있는 노드 2가
블록 생성자가 된 경우

PoS Implementation

- 브로드 캐스트 받은 노드들이 자신의 체인에 블록 추가

```
// 블록 검증하고 리턴값에 따라 블록체인에 추가하거나 무시하거나
Block receive_block = Arr2block(msg);

bool check = ValidationBlock(receive_block);


if(check){

  m_Blockchain.AddBlock(receive_block);
  m_Blockchain.printBlockchain(Round);

  Round ++;
  Staking_day += 1; // 라운드가 지남에 따라 Staking_day + 1
}
else {
  return;
}
```

전송 받은 블록 **검증 수행**

유효한 블록일 경우 **체인에 블록 추가**
&
스테이킹 시간이 증가함에 따라 코인 나이도 증가

유효하지 않은 블록일 경우 **체인에 추가 X**

PoS Implementation

전송 받은 블록 검증

유효한 블록일 경우
체인에 추가 O

```
========== 블록 수신 ==========
Round : 364
검증 결과 : 유효한 블록
노드0 : 전송 받은 블록을 체인에 추가하였습니다.
Block 364
{
   BlockId : 364
   PrevHash : f37be8c4e9d302a2c35c8a64260b9b7fcd0ad3198e65a3a1150b2a97ff7625c3
   CoinAge : 56
   Transaction : 5156851973
}
```

유효하지 않은 블록일 경우
체인에 추가 X

```
========== 블록 수신 ==========
Round : 10
검증 결과 : 유효하지 않은 블록
노드2 : 전송 받은 블록을 체인에 추가하지 않았습니다.
전송 받은 블록 {
   BlockId : 10
   PrevHash : f306a3103332db489eb3ee22ab781af555cf3f987d1e42fcf2110f3da52db4cb
   CoinAge : 50
}
```

향후계획

1. PoS를 **DPoS로 확장** 시켜 새로운 합의 알고리즘을 구현할 때 참고하여 구현할 예정
2. **메모리풀** 구현 (처리되지 않은 트랜잭션 저장소)
3. 전자서명 크리스탈 **딜리씨움**
4. 트랜잭션 클래스화
5. **포크** 구현을 어떻게 해야할지
6. **Z-test** 구현
7. 블록, 트랜잭션, 블록체인, 네트워크 클래스 헤더 파일 구현 파일 모두 나누기

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.