블록체인 과제 DPoL

https://youtu.be/BmObqA8l3zg

DPoL

1. **양자컴퓨터 위협** 양자내성암호 적용 필요 → 키 사이즈 증가
2. **양자 내성 암호 + Delegation + PoL**
3. **위임 적용** → 합의에 소요되는 시간이 단축
4. **PoL : TEE 사용** → 행운증명, ASIC 저항

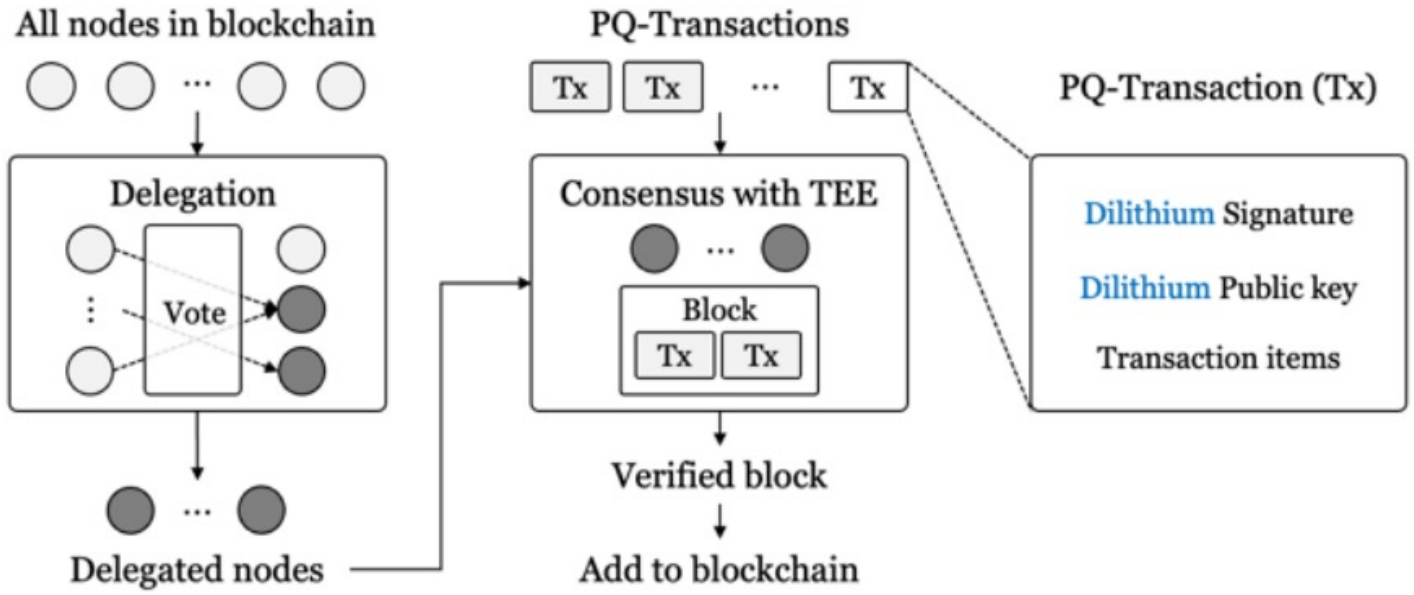


Fig. 1: Overview of PQ-DPoL.

DPoL

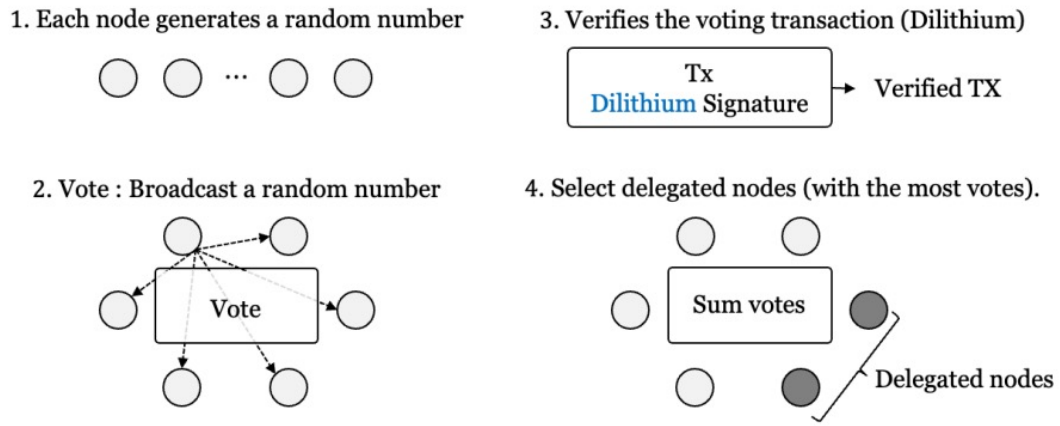


Fig. 2: Delegation phase of PQ-DPoL.

1. **난수 생성** : 각 노드는 **난수를 생성** 후,
   브로드캐스트(난수와 Dilithium의 서명 및 공개 키)

2. **서명 확인 :** 수신받은 다른 노드의 트렉젝션 검증

3. **위임 노드 선택** :가장 큰 난수값 상위 n개 노드가 선택

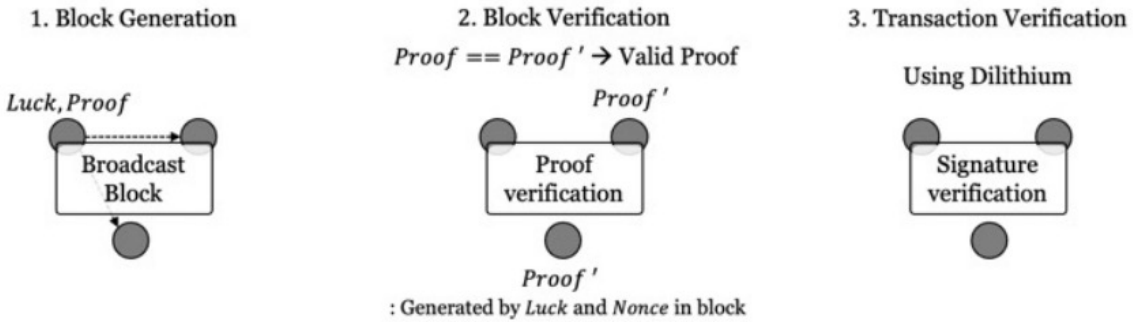
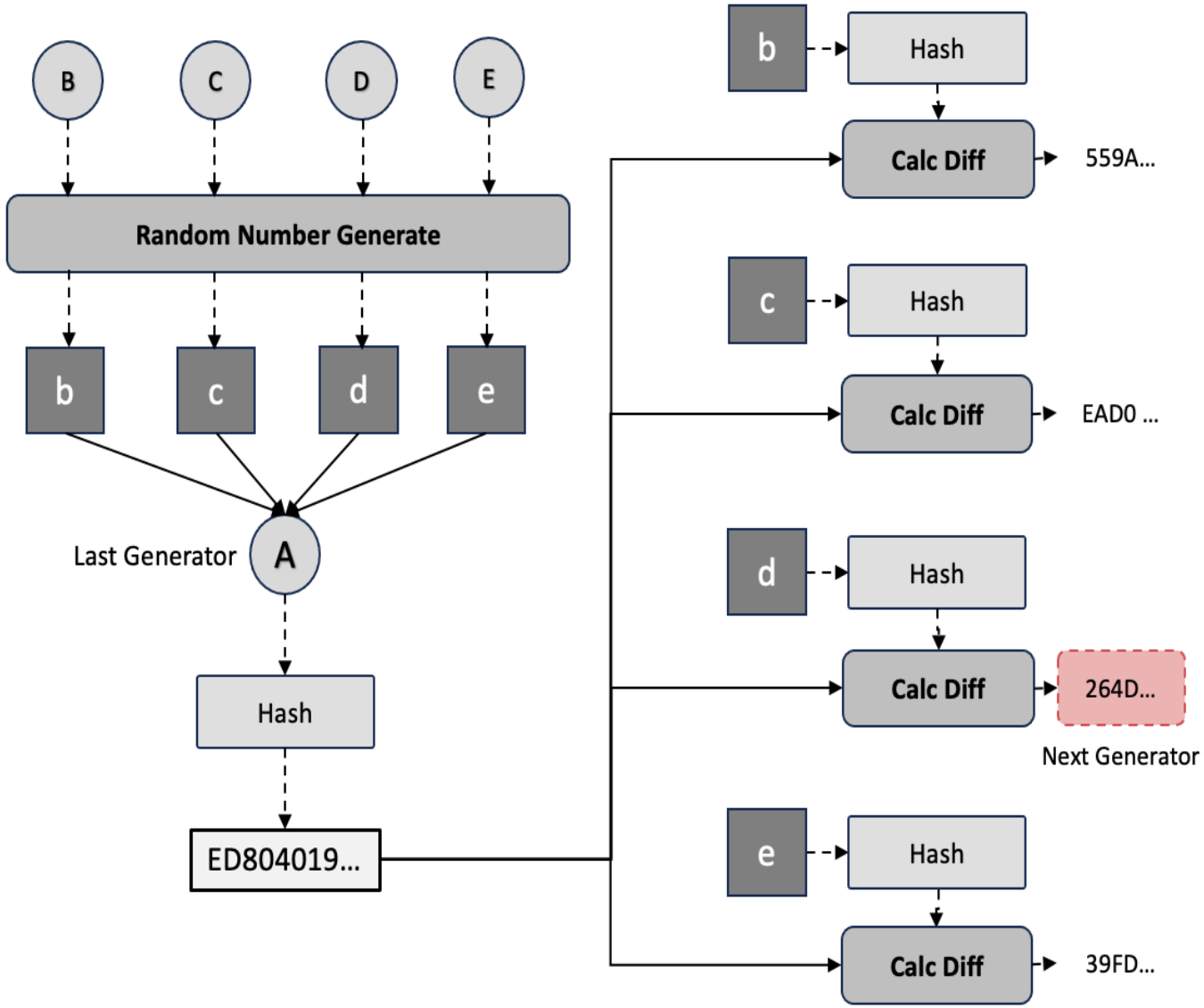


Fig. 3: Consensus phase of PQ-DPoL.

PoL 합의

의문점 :
다른 위임 합의 알고리즘은 선출된 대표자 노드에서
블록을 순차적으로 생성, **해당과정의 필요성?**

PQ-PoRR



3-1. 알고리즘

[그림 1]은 시스템의 전체적인 과정을 나타낸다. 우선 이전 라운드의 블록 생성자가 다른 노드들로부터 생성된 무작위 값을 수집한다. 그 후 수집한 무작위 값을 연접하여 해시 함수의 입력 값으로 사용하여 라운드 해시 값을 계산한다. 다른 노드들은 자신의 무작위 값을 해시하여 라운드 해시와의 차이를 계산한다. 차이가 가장 적은 해시를 생성한 노드가 다음 라운드의 블록 생성자로 선택된다. 이 때 이전에 블록을 생성한 적이 있는 노드는 선택되지 않으며 모든 노드가 동일한 수의 블록을 생성하였을 때 블록 생성자로 선택될 수 있다.

PBFT 기반 합의 알고리즘?

이전 Ns-3 구현

- 노드 간 송수신 문제

```
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;
            block.DeserializeBlock(msg);

            VerifyTxs(block);

            if(ZTest()) {
                Simulator::Cancel(m_eventId);

            }

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

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

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

            sleep(1);
        }
    }
}
```

초기단계

```
void
DPoLNode::StartApplication ()···
```

```
// Broadcast node id to peers
for (auto iter = m_peersAddresses.begin(); iter != m_peersAddresses.end(); ++iter) {
    TypeId tid = TypeId::LookupByName("ns3::UdpSocketFactory");
    Ptr<Socket> socketClient = Socket::CreateSocket(GetNode(), tid);
    socketClient->Connect(InetSocketAddress(*iter, 7071));
    m_peersSockets[*iter] = socketClient;

    uint8_t data[5];
    data[0] = intToChar(INIT);
    data[1] = intToChar((m_id >> 24) & 0xFF);
    data[2] = intToChar((m_id >> 16) & 0xFF);
    data[3] = intToChar((m_id >> 8) & 0xFF);
    data[4] = intToChar(m_id & 0xFF);

    Ptr<Packet> packet = Create<Packet>(data, sizeof(data));
    socketClient->Send(packet);
}

InitializeKey();
InitializeChain();
InitializeMempool();

Prevote();
```

단계 1

```
void
DPoLNode::Prevote(void){
    std::multiset<Item> min_heap2;
    min_heap.swap(min_heap2);
    uint64_t random_value;
    Ptr<Packet> p;
    uint8_t size = 13;
    uint8_t * data = GenerateRandom(random_value);
    m_rand = random_value;
    p = Create<Packet> (data, size);

    TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
    std::vector<Ipv4Address>::iterator iter = m_peersAddresses.begin();
    while(iter != m_peersAddresses.end()) {
        TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
        Ptr<Socket> socketClient = m_peersSockets[*iter];
        double delay = getRandomDelay();
        Simulator::Schedule(Seconds(delay), SendPacket, socketClient, p);
        iter++;
    }
    Item item;

    item.id = m_id;
    item.value = random_value;
    const int k = producer_num;
        min_heap.insert(item);

        if (min_heap.size() > k) {
            min_heap.erase(min_heap.begin());
        }
}
```

```
void
> DPoLNode::HandleRead (Ptr<Socket> socket)···
case PREVOTE:
{
    Item item;
    item.id = (static_cast<uint32_t>(charToInt(msg[1])) << 24) |
    (static_cast<uint32_t>(charToInt(msg[2])) << 16) |
    (static_cast<uint32_t>(charToInt(msg[3])) << 8) |
    static_cast<uint32_t>(charToInt(msg[4]));

    item.value = (static_cast<uint64_t>(charToInt(msg[5])) << 56) |
            (static_cast<uint64_t>(charToInt(msg[6])) << 48) |
            (static_cast<uint64_t>(charToInt(msg[7])) << 40) |
            (static_cast<uint64_t>(charToInt(msg[8])) << 32) |
            (static_cast<uint64_t>(charToInt(msg[9])) << 24) |
            (static_cast<uint64_t>(charToInt(msg[10])) << 16) |
            (static_cast<uint64_t>(charToInt(msg[11])) << 8) |
            static_cast<uint64_t>(charToInt(msg[12]));


    Find_Minimum(item);
    producer_vote++;

    if (CheckIdInHeap(m_id,min_heap) && producer_vote == node_num) {
        producer = 1;
        auto max_it = min_heap.begin();

        if(max_it->id == m_id){
            leader = m_id;
            vector<uint8_t> prevHash = GetBlockHash(m_chain.GetLatestBlock());
            vector<Tx> txs = GetPendingTxs();
            Block newBlock(prevHash, txs);
            Simulator::Schedule(Seconds(timeout), &DPoLNode::SendBlock, this, newBlock);

        }
    }


    break;
```

단계 2

```
case PREPARE:
{
    Block block;
    block.DeserializeBlock(msg);

    int dataSize = block.SerializeBlock().second;
    uint8_t *data = (uint8_t *)std::malloc(dataSize+8);

    data[0] = intToChar(PRECOMMIT);
    data[1] = intToChar(charToInt(msg[1]));      // v
    data[2] = intToChar(charToInt(msg[2]));      // n
    data[3] = intToChar(VerifyTxs(block));
    data[4] = intToChar(charToInt(msg[4]));
    data[5] = intToChar(charToInt(msg[5]));
    data[6] = intToChar(charToInt(msg[6]));
    data[7] = intToChar(charToInt(msg[7]));

    uint8_t *blockdata = network.Bytes2Packet(block.SerializeBlock().first, dataSize);

    std::memcpy(data + 8, blockdata, dataSize);
    Ptr<Packet> p;
    p = Create<Packet> (data, dataSize+8);
    free(data);

    TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
    std::vector<Ipv4Address>::iterator iter = m_peersAddresses.begin();

    while(iter != m_peersAddresses.end()) {···

    break;
}
```

단계 3

```
case PRECOMMIT:
{
    int index = charToInt(msg[2]);
    if (charToInt(msg[3]) == 1) {
        tx[index].prepare_vote++;
    }

    if (tx[index].prepare_vote == vote_num) {

        Block block;
        block.DeserializeBlock(msg);

        int dataSize = block.SerializeBlock().second;
        uint8_t *data = (uint8_t *)std::malloc(dataSize+8);

        data[0] = intToChar(COMMIT);
        data[1] = intToChar(charToInt(msg[1]));       // v
        data[2] = intToChar(charToInt(msg[2]));       // n
        data[3] = intToChar(0);
        data[4] = intToChar(charToInt(msg[4]));
        data[5] = intToChar(charToInt(msg[5]));
        data[6] = intToChar(charToInt(msg[6]));
        data[7] = intToChar(charToInt(msg[7]));

        uint8_t *blockdata = network.Bytes2Packet(block.SerializeBlock().first, dataSize);

        std::memcpy(data + 8, blockdata, dataSize);

        Send(data, dataSize+8);
        free(data);
    }
    break;
}
```

단계 4

```
case COMMIT:
{
    int index = charToInt(msg[2]);
    tx[index].commit_vote++;

    if(tx[index].commit_vote == producer_num-2){
        if((charToInt(msg[2])+1)%(producer_num+1)== 0){
            producer_vote=1;
            Prevote();
            break;
        }
        msg = getPacketContent(packet, from);
        Block block;
        block.DeserializeBlock(msg);
        m_chain.AddBlock(block);

        if(CheckIdInHeap(m_id,min_heap))
        {
            auto max_it = min_heap.begin();
            for(int i =0;i<(charToInt(msg[2]))%producer_num+1;i++){
                    ++max_it;
                }
            if(max_it->id == m_id){
                printf("BLOCK %d \n",charToInt(msg[2]));
                double currentTime = Simulator::Now().GetSeconds();
                NS_LOG_UNCOND("Current simulation time: " << currentTime << " seconds");
                leader = m_id;

                vector<uint8_t> prevHash = GetBlockHash(m_chain.GetLatestBlock());
                vector<Tx> txs = GetPendingTxs();
                Block newBlock(prevHash, txs);

                Simulator::Schedule(Seconds(timeout), &DPoLNode::SendBlock, this, newBlock);
            }

        }
    }
    break;
}
```

Q & A