Final Proposal

2017.06.06

KONKUK UNIVERSITY ITCS

노은방, 심우진

CONTENTS

Introduction

Blockchain & Bitcoin

- Bitcoin은 중앙 집중형 시스템을 탈피하여 분산 시스템에서 주고 받는 정보의 무결성을 제공하는 블록체인을 활용한 가상 화폐
- 원본데이터(원장)의 분산을 통해 위・변조에 강한 내구성을 지님
- Bitcoin Protocol에서 블록체인은 블록 연결이 가장 긴 체인을 선택하여 해당 블록체인이 결정
- But, 공격자가 선택된 블록이 아닌 새로운 블록을 생성하여 기존 블록보다 길게 블록을 생성하면 공격자가 생성한 블록이 선택되어 거래가 이루어지는 문제가 발생.
- 이러한 공격을 막기 위해 Bitcoin의 GHOST(Greedy Heaviest-Observed Sub-Tree) Protocol을 활용한 블록 체인을 검증.

GHOST Bitcoin Protocol

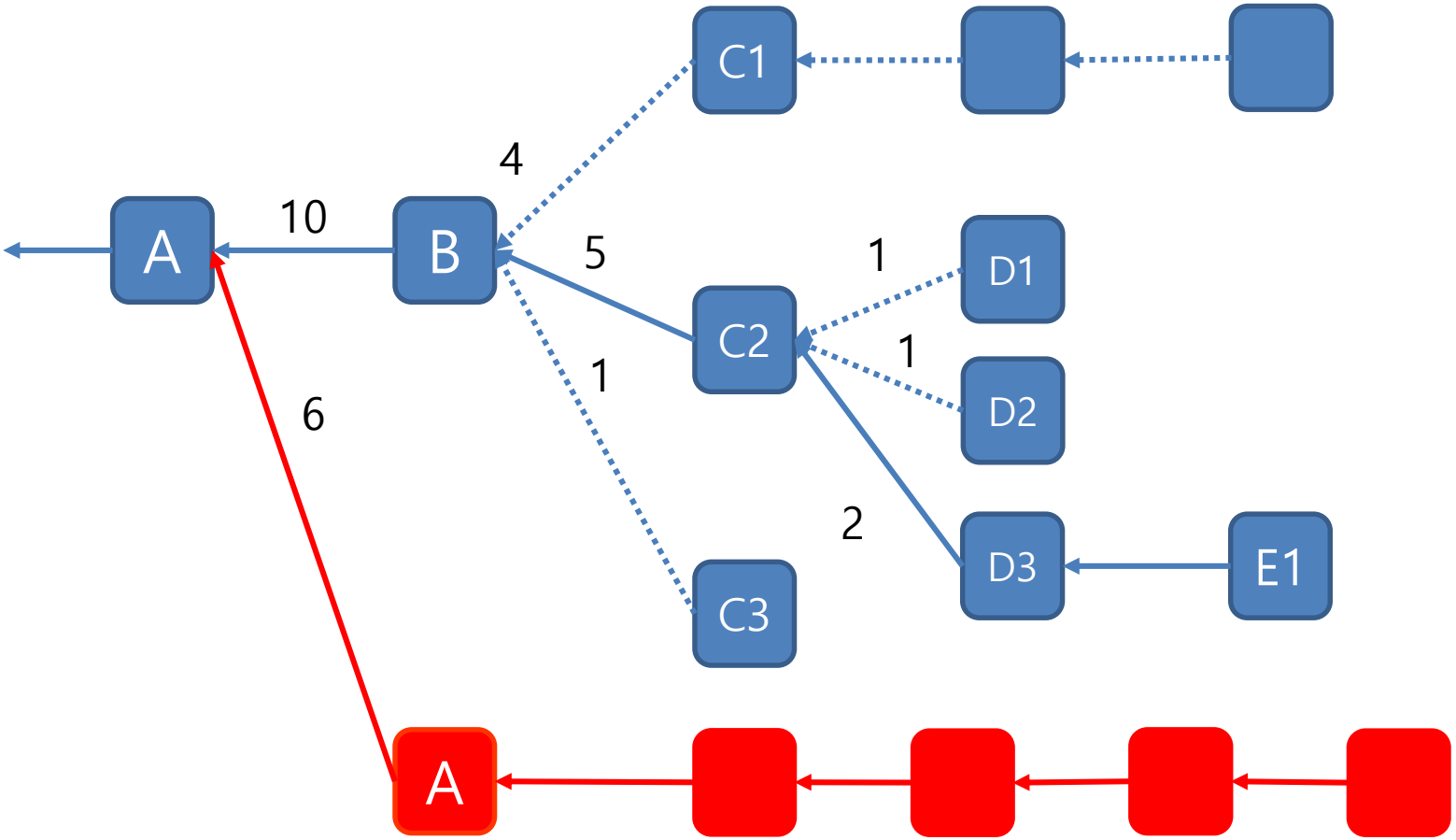
Input: Block tree T

1. *set* $B \leftarrow$ *Genesis Block*
2. *if* $Children_T(B) = \emptyset$ *then return*(B) *and exit*
3. *else update* $B \leftarrow \underset{C \in Children_T(B)}{\operatorname{argmax}} |subtree_T(C)|^8$
4. *goto line 2*

1. Blockchain의 첫 번째 블록인 Genesis Block 설정.
2. 자식 노드인 블록이 없으면 블록 반환 후 종료
3. 2가 아닌 경우 블록 업데이트를 수행하는데 $|subtree_T(C)|^8$가 최대값이 되도록 Block을 갱신
4. 수행 후 2로 이동

Protocol

GHOST Bitcoin Protocol



Modeling

Constraint

1. Blockchain의 time은 Spin으로 적용하기 어려움으로 인해 블록 생성 시 +1씩 증가하는 방향으로 모델링

2. 채굴 난이도를 결정할 시 바이트의 한계로 인해 채굴 난이도보다 작은 값이 아닌 동일한 값을 찾도록 모델링

Peer – Source_Code

```
peerInit:
    do
    //블록 새로 생성되는거 대기
    :: nempty(blockSolution) -> blockSolution?<recvBlock>;
        //지금까지 저장한 블록과 비교해봄
        if
        //새로 받은 블록을 검사, 0이면 제네시스 블록부터
        ::(recvBlock.prev==0 && curChain<recvBlock.cur) ->
                    blockChain[curChain+1].prev = recvBlock.prev;
                    blockChain[curChain+1].cur = recvBlock.cur;
                    blockChain[curChain+1].length = recvBlock.length;
                    blockChain[curChain+1].time = recvBlock.time;
                    curChain++;
        :: else ->
            if
            ::(recvBlock.prev==blockChain[0].cur && curChain<recvBlock.cur) ->
                    blockChain[curChain+1].prev = recvBlock.prev;
                    blockChain[curChain+1].cur = recvBlock.cur;
                    blockChain[curChain+1].length = recvBlock.length;
                    blockChain[curChain+1].time = recvBlock.time;
                    curChain++;
            ::(recvBlock.prev==blockChain[1].cur && curChain<recvBlock.cur) ->
                    blockChain[curChain+1].prev = recvBlock.prev;
                    blockChain[curChain+1].cur = recvBlock.cur;
                    blockChain[curChain+1].length = recvBlock.length;
                    blockChain[curChain+1].time = recvBlock.time;
                    curChain++;
```

Peer – Source_Code

```
//트렌젝션 수신
::(curPeerTx < curTx) ->
    for(i : curPeerTx .. curTx){
        if
        ::(TxPool[i].TxOutput.address==peerId) ->
            walletVal = TxPool[i].TxOutput.value + walletVal;
            txprevId=TxPool[i].Id
        ::else->skip
        fi
    }
```

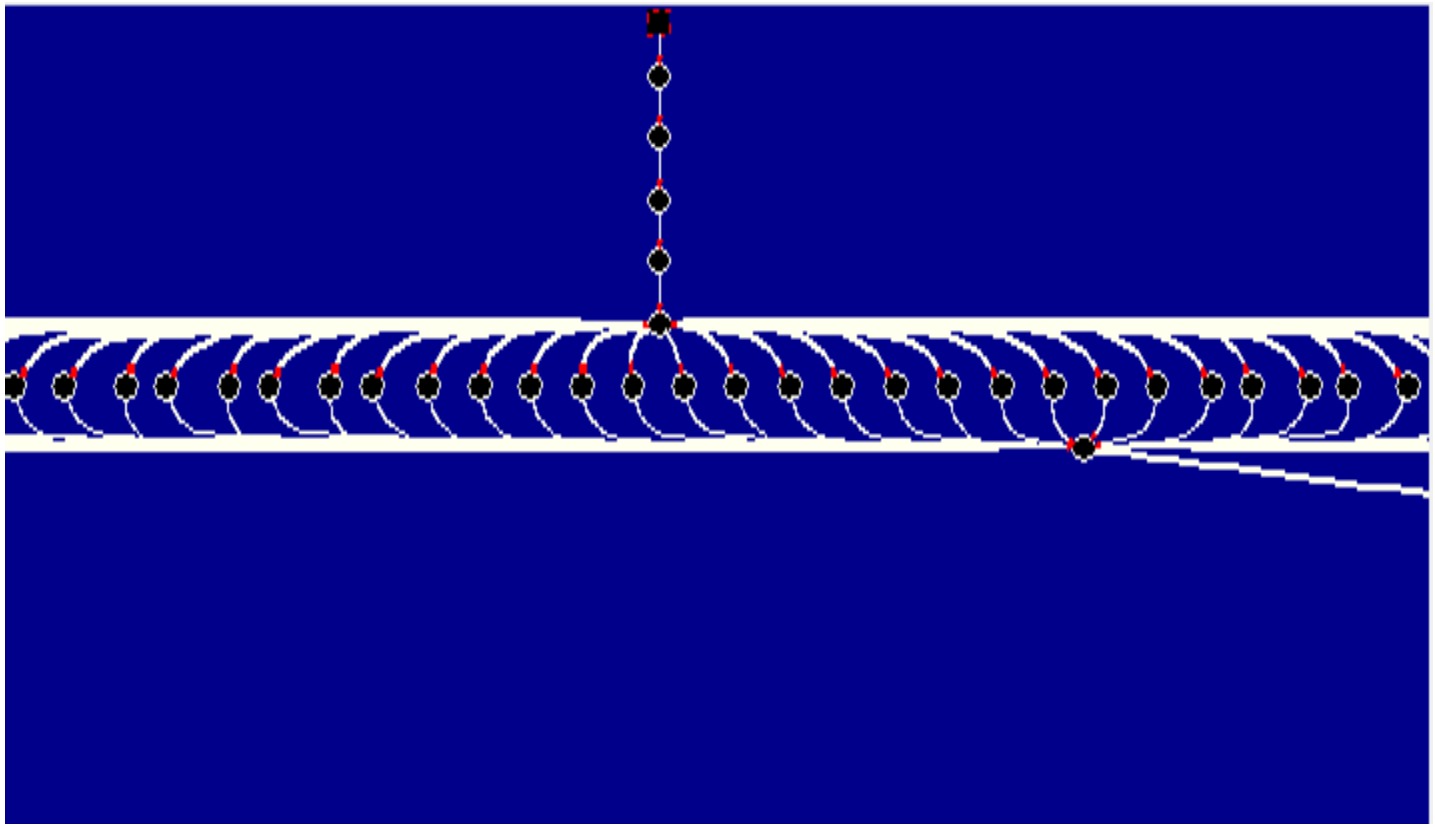
```
::(walletVal>=71) -> transactionVal=70; goto transactionSend
::(walletVal>=72) -> transactionVal=71; goto transactionSend
::(walletVal>=73) -> transactionVal=72; goto transactionSend
::(walletVal>=74) -> transactionVal=73; goto transactionSend
::(walletVal>=75) -> transactionVal=74; goto transactionSend
::(walletVal>=76) -> transactionVal=75; goto transactionSend
::(walletVal>=77) -> transactionVal=76; goto transactionSend
::(walletVal>=78) -> transactionVal=77; goto transactionSend
::(walletVal>=79) -> transactionVal=78; goto transactionSend
```

Peer – Source_Code

```
transactionSend:
//트랜잭션 전송
    curTx++;
    STIME++;

    TxPool[curTx].Id = curTx;
    TxPool[curTx].time = STIME;
    TxPool[curTx].TxInput.id = txprevId;
    if
    //PEERS
    //peer번호중 초기번호가 아니고, 지금 이 피어가 아닌 랜덤한 피어에게 트랜잭션
    ::(peerlist[0]!=0 && peerlist[0]!=peerId) -> TxPool[curTx].TxOutput.address=peerlist[0]
    ::(peerlist[1]!=0 && peerlist[1]!=peerId) -> TxPool[curTx].TxOutput.address=peerlist[1]
    ::(peerlist[2]!=0 && peerlist[2]!=peerId) -> TxPool[curTx].TxOutput.address=peerlist[2]
    ::(peerlist[3]!=0 && peerlist[3]!=peerId) -> TxPool[curTx].TxOutput.address=peerlist[3]
    ::(peerlist[4]!=0 && peerlist[4]!=peerId) -> TxPool[curTx].TxOutput.address=peerlist[4]
    ::(peerlist[5]!=0 && peerlist[5]!=peerId) -> TxPool[curTx].TxOutput.address=peerlist[5]
    fi
    TxPool[curTx].TxOutput.value = transactionVal;
    TxPool[curTx].status = UNCONFIRMED;
```

Peer – Automata



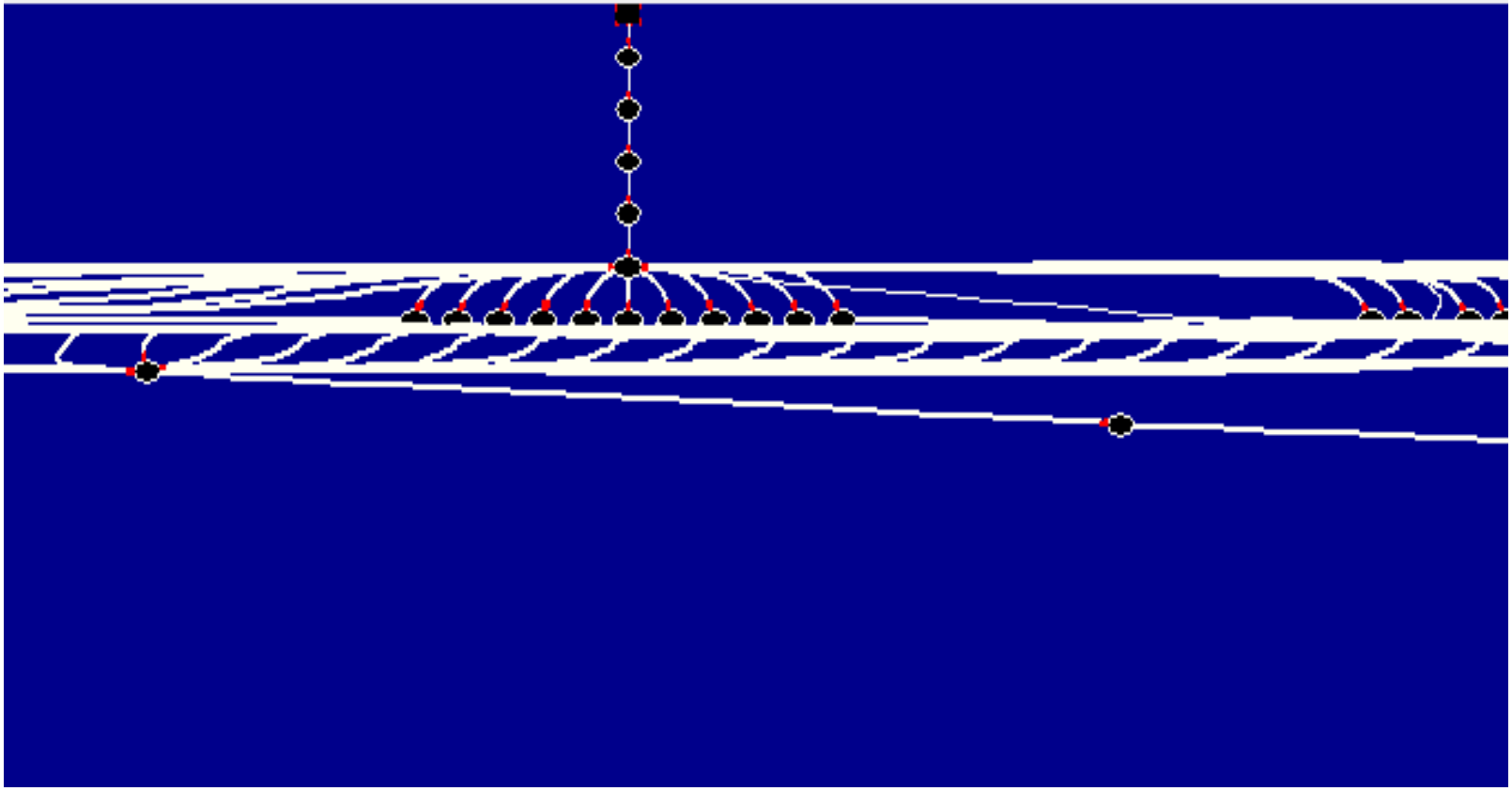
Pool – Source_Code

```
poolInit:
    do
    //블록 생성 대기
    :: nempty(blockSolution) -> blockSolution?<recvBlock>;
        //지금까지 저장한 블록과 비교해봄
        //fork 하는 기준을 어떻게 세울 것인가??
        //미리 생성하고 배포하는 블록 내부에서 이를 전달할 것인가? 이게 맞는듯
        if
        //새로 받은 블록을 검사, 0이면 제네시스 블록부터
        ::(recvBlock.prev==0 && curChain<recvBlock.cur) ->
                    blockChain[curChain+1].prev = recvBlock.prev;
                    blockChain[curChain+1].cur = recvBlock.cur;
                    blockChain[curChain+1].length = recvBlock.length;
                    blockChain[curChain+1].time = recvBlock.time;
                    curChain++;
        :: else ->
                if
                ::(recvBlock.prev==blockChain[0].cur && curChain<recvBlock.cur) ->
                        blockChain[curChain+1].prev = recvBlock.prev;
                        blockChain[curChain+1].cur = recvBlock.cur;
                        blockChain[curChain+1].length = recvBlock.length;
                        blockChain[curChain+1].time = recvBlock.time;
                        curChain++;
                ::(recvBlock.prev==blockChain[1].cur && curChain<recvBlock.cur) ->
                        blockChain[curChain+1].prev = recvBlock.prev;
                        blockChain[curChain+1].cur = recvBlock.cur;
                        blockChain[curChain+1].length = recvBlock.length;
                        blockChain[curChain+1].time = recvBlock.time;
                        curChain++;
```

Pool – Source_Code

```
::mining=4 ->
    if
    ::(mining==SOLUTION) -> goto solutionFind;
    ::else -> skip
    fi
::mining=5 ->
    if
    ::(mining==SOLUTION) -> goto solutionFind;
    ::else -> skip
    fi
::mining=6 ->
    if
    ::(mining==SOLUTION) -> goto solutionFind;
    ::else -> skip
    fi
::mining=7 ->
    if
    ::(mining==SOLUTION) -> goto solutionFind;
    ::else -> skip
    fi
::mining=8 ->
    if
    ::(mining==SOLUTION) -> goto solutionFind;
    ::else -> skip
    fi
```

Pool – Automata



Malicious Peer – Source_Code

```
proctype mal_peer(chan poolToPeer, peerid)
{
    //수신용
    BC recvBlock;
    Tx sendTx;

    byte peerId = peerid;
    byte curChain=0;
    byte curPeerTx=0;
    byte i=0;
    //지금 보관하는 비트코인 개수
    byte walletVal=0;
    //트랜잭션 양
    byte transactionVal=0;
    //이전에 트랜잭션의 id
    byte txprevId;
    //ghost protocol 임시변수
    byte temp_GHOST[MAX_BC];

    BC blockChain[MAX_BC];
    blockChain[0].cur = 0;
    blockChain[0].prev = 0;
    blockChain[0].length = 0;
    blockChain[0].time = 0;
```

Malicious Peer – Source_Code

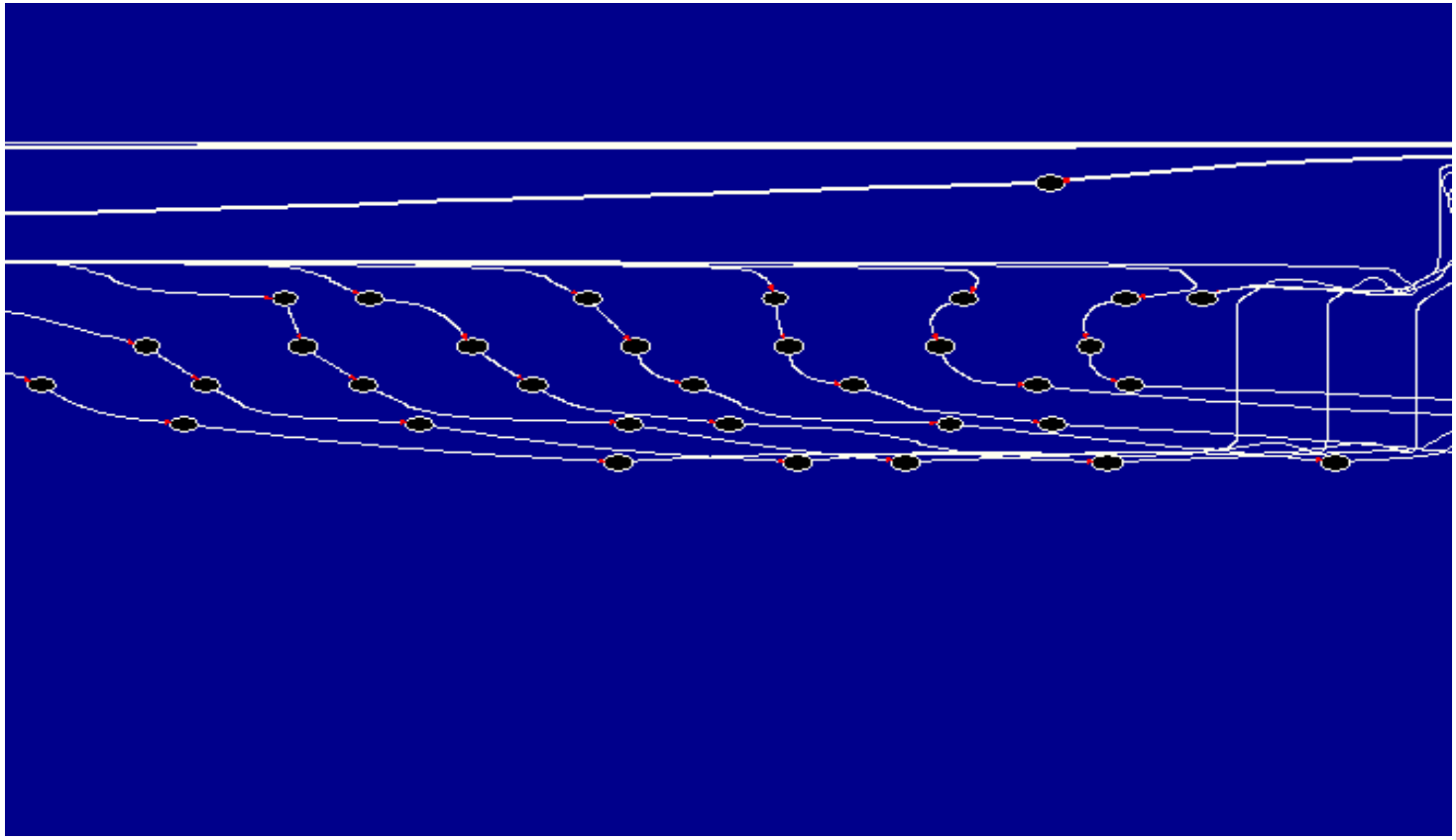
```
    //트랜잭션 수신
    ::(curPeerTx < curTx) ->
        for(i : curPeerTx .. curTx){
            if
            ::(TxPool[i].TxOutput.address==peerId) ->
                walletVal = TxPool[i].TxOutput.value + walletVal;
                txprevId=TxPool[i].Id
            ::else->skip
            fi
        }

        curPeerTx=curTx;
//  ::else->skip
    od

transactionSend:
//트랜잭션 전송
    curTx++;
    STIME++;

    TxPool[curTx].Id = curTx;
    TxPool[curTx].time = STIME;
    TxPool[curTx].TxInput.id = txprevId;
```

Malicious Peer – Automata

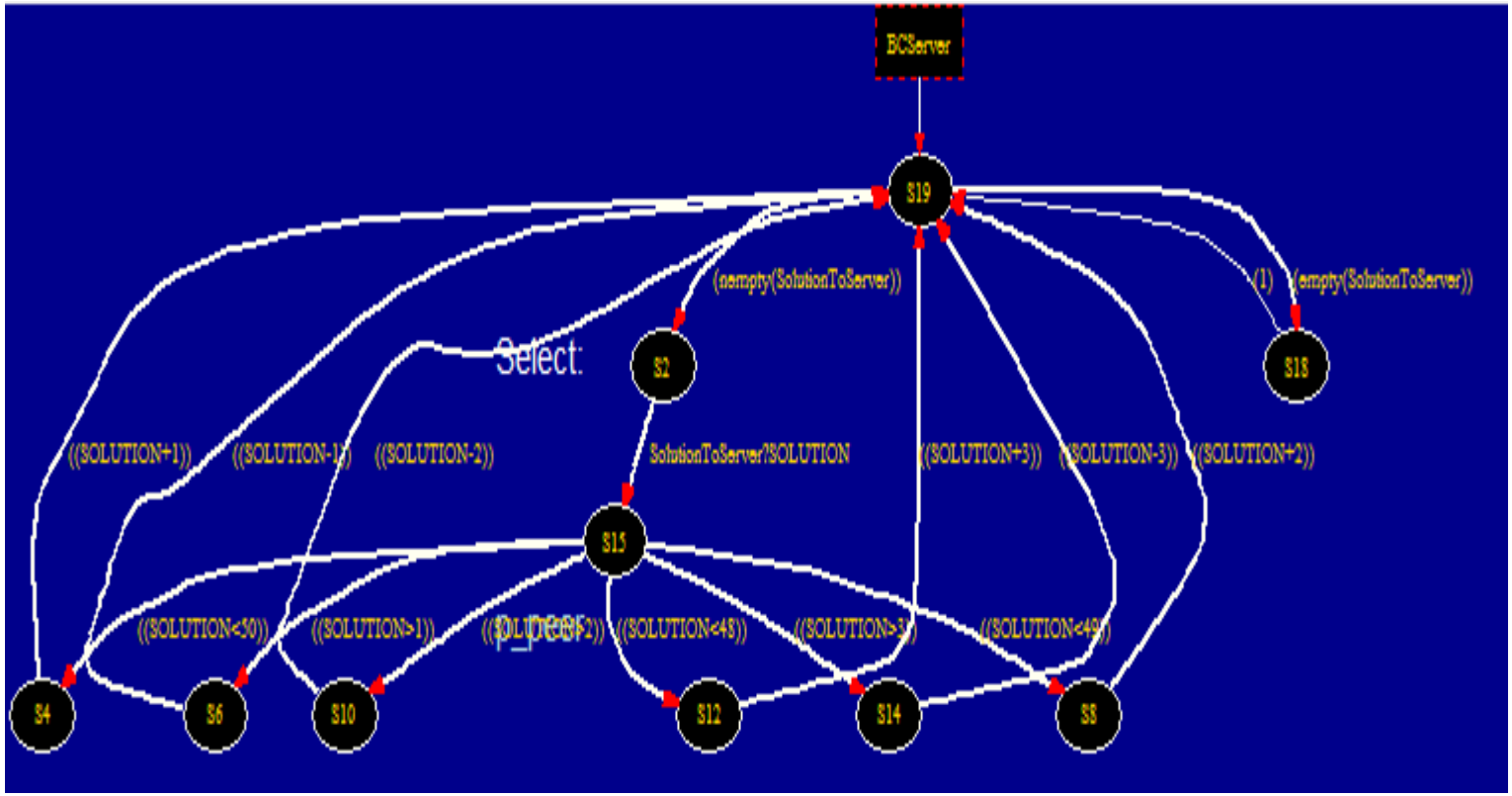


Server – Source_Code

```
proctype BCServer()
{
    do
    ::nempty(SolutionToServer) -> SolutionToServer?SOLUTION;
        if
        ::(SOLUTION<50) -> SOLUTION+1
        ::(SOLUTION>1) -> SOLUTION-1
        ::(SOLUTION<49) -> SOLUTION+2
        ::(SOLUTION>2) -> SOLUTION-2
        ::(SOLUTION<48) -> SOLUTION+3
        ::(SOLUTION>3) -> SOLUTION-3
        fi
    ::empty(SolutionToServer)-> skip;
    od
}
```

Modeling

Server – Automata



Property

- LTL p1 {[] <> (peer : recvBlock == CONFIRMED)}
  - 언제나 언젠간 블록의 상태는 CONFIRMED 되어야 한다.
  - 각 블록마다 트랜잭션이 일어나기 위해서는 블록의 상태가 CONFIRMED 되어 있어야 함.
  - True

- LTL p2 {[] <> (longestChain < longestChain_GHOST)}
  - 언제나 언젠간 longestChain은 longestChain_GHOST보다 길이가 길어야 한다.
  - 블록의 길이가 기존 블록체인보다 longestChain_GHOST보다 항상 길어야 함.
  - True

References

[1] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski and Łukasz, “Modeling Bitcoin Contracts by Timed Automata” University of Warsaw, 2014.

[2] Kaylash Chaudhary, Ansgar Fehnker, Jaco van de Pol, “Modeling and Verification of the Bitcoin Protocol” University of Twente, 2015.

THANK YOU