
MODULE *p2p*

This module defines a simple peer-to-peer network protocol that allows peers to connect, exchange blocks, and synchronize their chains.

EXTENDS *TLC*, *Sequences*, *Naturals*, *FiniteSets*, *Utils*, *Blockchain*

Define the network to be used by the algorithm.

CONSTANT *RunningBlockchain*

Maximum number of blocks to be retrieved in a single *getblocks* response.

CONSTANT *MaxGetBlocksInvResponse*

Maximum number of outbound connections a peer can have.

CONSTANT *MaxConnectionsPerPeer*

Difference in the *SYNCHRONIZER* process id so that it does not conflict with the *LISTENER* one.

PeerProcessDiffId \triangleq 1000

--algorithm *p2p*

variables

Represent the whole universe of peers in the network with all of their data.

the_network = *RunningBlockchain* ;

Each peer has a channel to communicate with other peers. Number of connections is limited.

channels = [*i* ∈ 1 .. *Len(the_network)* ↦
 [*j* ∈ 1 .. *MaxConnectionsPerPeer* ↦ [
 header ↦ *defaultInitValue*,
 payload ↦ *defaultInitValue*
]]] ;

define

Import the operators used in the algorithm.

LOCAL *Ops* \triangleq INSTANCE *Operators*

This property checks for the existence of at least one execution path in which all peers eventually have the same chain tip. It ensures that there is a scenario in which synchronization occurs, but does NOT guarantee that synchronization will happen in every possible execution. This makes it an existential check, not a liveness property.

ExistsSyncPath \triangleq
 $\exists \text{ peer1, peer2 } \in 1 \dots \text{Len}(\text{RunningBlockchain}) :$
 $\Diamond (\text{Ops!GetPeerTipByIndex}(\text{peer1}).\text{height} = \text{Ops!GetPeerTipByIndex}(\text{peer2}).\text{height})$

Liveness: Eventually, all peers will have the same chain tip. This property ensures that synchronization will happen in every possible path.

Note: This property is not guaranteed to hold in the current implementation.

$Liveness \triangleq$
 $\forall peer1, peer2 \in 1 \dots Len(RunningBlockchain) :$
 $\Diamond (Ops! GetPeerTipByIndex(peer1).height = Ops! GetPeerTipByIndex(peer2).height)$

Ensures that no peer in the network has a chain tip that is higher than the chain tip of any peer it is connected to. This guarantees that peers do not “advance” their chain beyond the knowledge of their connected peers, ensuring consistent progress across the network.

$ChainTipRespectsPeerSet \triangleq$
 $\forall peer \in 1 \dots Len(RunningBlockchain) :$
 $\forall remote_peer \in 1 \dots Len(the_network[peer].peer_set) :$
 $Ops! GetPeerTipByIndex(peer).height \leq Ops! GetPeerTipByIndex(remote_peer).height$

Ensures that each block in a peer’s local blockchain has a height less than or equal to the peer’s chain tip. This prevents peers from including invalid blocks that exceed the current chain tip.

$ValidBlockPropagation \triangleq$
 $\forall peer \in 1 \dots Len(RunningBlockchain) :$
 $\forall block \in the_network[peer].blocks :$
 $block.height \leq Ops! GetPeerTipByIndex(peer).height$

Ensures that the blocks within each peer’s blockchain are correctly ordered by height. For blocks with height greater than 1, each block must directly follow the block with height $block.height - 1$. This prevents gaps or misordering within a peer’s chain.

$BlockOrdering \triangleq$
 $\forall peer \in 1 \dots Len(RunningBlockchain) :$
 $\forall block \in the_network[peer].blocks :$
 IF $block.height < 2$ THEN
 TRUE
 ELSE
 $block.height =$
 $(CHOOSE b \in the_network[peer].blocks : b.height = block.height - 1).height + 1$

Ensures that each peer eventually reaches a chain tip that is at least as high as the initial chain tip it started with. This ensures that peers make progress in synchronizing their chains over time.

$SyncProgress \triangleq$
 $\forall peer \in 1 \dots Len(RunningBlockchain) :$
 $\Diamond (Ops! GetPeerTipByIndex(peer).height \geq$
 $Ops! GetPeerTipByIndexAndNetwork(peer, RunningBlockchain).height)$

Ensures that no peer exceeds the maximum number of connections allowed, ensuring that the network respects its maximum connection constraints. This prevents any peer from overloading its connection capacity.

$ConnectionLimit \triangleq$
 $\forall peer \in 1 \dots Len(RunningBlockchain) :$
 $Len(the_network[peer].peer_set) \leq MaxConnectionsPerPeer$

Ensures that for any two peers, if they have blocks at the same height, the blocks must be identical in both content and hash. This guarantees that no two peers hold blocks with the same height but different content, preventing inconsistencies and potential forks in the blockchain.

$ConsistentBlocksAcrossPeers \triangleq$

$\forall peer1, peer2 \in 1 \dots Len(RunningBlockchain) :$
 $peer1 \neq peer2 \Rightarrow$
 $\forall block1 \in the_network[peer1].blocks, block2 \in the_network[peer2].blocks :$
 $block1.height = block2.height \Rightarrow$
 $block1.hash = block2.hash \wedge block1.block = block2.block$

The overall inductive invariant that combines several sub-properties to ensure safety and correctness in the peer-to-peer protocol :

- *ChainTipRespectsPeerSet* ensures chain tip consistency between peers.
- *ValidBlockPropagation* ensures peers propagate valid blocks.
- *BlockOrdering* ensures correct block order within each peer's chain.
- *SyncProgress* ensures that peers continue to progress toward synchronization.
- *ConnectionLimit* ensures peers respect connection limits.

$InductiveInvariant \triangleq$

$\wedge ChainTipRespectsPeerSet$
 $\wedge ValidBlockPropagation$
 $\wedge BlockOrdering$
 $\wedge SyncProgress$
 $\wedge ConnectionLimit$
 $\wedge ConsistentBlocksAcrossPeers$

end define ;

Announce the intention of a peer to connect with another in the network by sending an *addr* message.

procedure *announce*(*local_peer_id*, *remote_peer_id*)
begin
 SendAddrMsg:
 channels[*local_peer_id*][*remote_peer_id*] := [
 header \mapsto [*command_name* \mapsto "addr"],
 payload \mapsto [
 address_count \mapsto 1,
 Only a single address is supported.
 addresses $\mapsto the_network[local_peer_id].peer$
]
];
 return;
end procedure ;

Given that an *addr* message is received, send a *version* message from the remote peer to start the connection.

procedure *addr*(*local_peer_id*, *remote_peer_id*)
begin
 SendVersionMsg:
 channels[*local_peer_id*][*remote_peer_id*] := [
 header \mapsto [*command_name* \mapsto "version"],
 payload \mapsto [
 version \mapsto 1,
 services \mapsto 0,
 timestamp \mapsto 0,
 address \mapsto [
 addr \mapsto [*local_peer_id*],
]
]
];
 return;
end procedure ;

```

    header  $\mapsto$  [command_name  $\mapsto$  "version"],
    payload  $\mapsto$  [
        addr_recv  $\mapsto$  the_network[local_peer_id].peer,
        addr_trans  $\mapsto$  the_network[local_peer_id].peer_set[remote_peer_id].address,
        start_height  $\mapsto$ 
            Ops! GetPeerTipByAddress(the_network[local_peer_id].peer_set[remote_peer_id].address).h
    ];
    return;
end procedure ;

```

Given a version message is received, send verack to acknowledge the connection.

```

procedure version(local_peer_id, remote_peer_id)
begin
    HandleVersionMsg:
        the_network[local_peer_id].peer_set[remote_peer_id].tip :=
            channels[local_peer_id][remote_peer_id].payload.start_height ;
    SendVerackMsg:
        channels[local_peer_id][remote_peer_id] := [
            header  $\mapsto$  [command_name  $\mapsto$  "verack"],
            payload  $\mapsto$  defaultInitValue
        ];
    return;
end procedure ;

```

Given a verack message is received, establish the connection.

```

procedure verack(local_peer_id, remote_peer_id)
begin
    HandleVerackMsg:
        the_network[local_peer_id].peer_set[remote_peer_id].established := TRUE ;
    return;
end procedure ;

```

Given a getblocks message is received, send an inv message with the blocks requested.

```

procedure getblocks(local_peer_id, remote_peer_id)
variables
    found_blocks, hash_count, block_header_hashes, remote_peer_blocks, start_height, end_height ;
begin
    HandleGetBlocksMsg:
        Retrieve necessary values from the channel payload
        hash_count := channels[local_peer_id][remote_peer_id].payload.hash_count ;
        block_header_hashes := channels[local_peer_id][remote_peer_id].payload.block_header_hashes ;

        Fetch the blocks of the remote peer
        remote_peer_blocks :=
            Ops! GetPeerBlocks(the_network[local_peer_id].peer_set[remote_peer_id].address) ;

```

```

    Determine the range of blocks to retrieve
    if hash_count = 0 then
        start_height := 1;
    else
        Assuming the hashes are in order, the height of the first hash should be the tip, ignore the rest.
        start_height :=
            Ops!FindBlockByHash(remote_peer_blocks, block_header_hashes[1]).height + 1;
    end if;
    end_height := start_height + (MaxGetBlocksInvResponse - 1);

    Find the blocks within the specified range.
    found_blocks := Ops!FindBlocks(remote_peer_blocks, start_height, end_height);
    SendInvMsg:
        channels[local_peer_id][remote_peer_id] := [
            header ↦ [command_name ↦ "inv"],
            payload ↦ [
                count ↦ Cardinality(found_blocks),
                inventory ↦ [
                    h ∈ 1 .. Cardinality(found_blocks) ↦ [
                        type_identifier ↦ "MSG_BLOCK",
                        hash ↦ SetToSeq({s.hash : s ∈ found_blocks})[h]
                    ]
                ]
            ]
        ];
    return;
end procedure;

Request blocks from the remote peer by sending a getblocks message with local hashes.
procedure request_blocks(hashes, local_peer_id, remote_peer_id)
begin
    SendGetBlocksMsg:
        channels[local_peer_id][remote_peer_id] := [
            header ↦ [command_name ↦ "getblocks"],
            payload ↦ [
                hash_count ↦ Len(hashes),
                block_header_hashes ↦ hashes
            ]
        ];
    return;
end procedure;

Given an inv message is received, send a getdata message to request the blocks.
procedure inv(local_peer_id, remote_peer_id)
begin
    SendGetDataMsg:
        channels[local_peer_id][remote_peer_id] := [

```

```

        header  $\mapsto$  [command_name  $\mapsto$  "getdata"],
        payload  $\mapsto$  channels[local_peer_id][remote_peer_id].payload
    ];
    return ;
end procedure ;

Incorporate data to the local peer from the inventory received.
procedure getdata(local_peer_id, remote_peer_id)
variables blocks_data ;
begin
    HandleGetDataMsg:
        blocks_data := [item  $\in$  1 .. Len(channels[local_peer_id][remote_peer_id].payload.inventory)  $\mapsto$ 
            Ops! FindBlockByHash(
                Ops! GetPeerBlocks(the_network[local_peer_id].peer_set[remote_peer_id].address,
                    channels[local_peer_id][remote_peer_id].payload.inventory[item].hash
            )
        ];
    IncorporateBlocks:
        the_network[local_peer_id].blocks := the_network[local_peer_id].blocks  $\cup$  ToSet(blocks_data) ;
    return ;
end procedure ;

A set of listener process for each peer to listen to incoming messages and act accordingly.
process LISTENER  $\in$  1 .. Len(RunningBlockchain)
variables command ;
begin
    Listening:
        await Len(the_network)  $\geq$  2 ;
        with remote_peer_index  $\in$  1 .. Len(the_network[self].peer_set) do
            if channels[self][remote_peer_index].header = defaultInitValue then
                goto Listening ;
            end if ;
        end with ;
    Requests:
        with remote_peer_index  $\in$  1 .. Len(the_network[self].peer_set) do
            await channels[self][remote_peer_index].header  $\neq$  defaultInitValue ;
            command := channels[self][remote_peer_index].header.command_name ;
            if command = "addr" then
                call addr(self, remote_peer_index) ;
                goto Listening ;
            elsif command = "version" then
                call version(self, remote_peer_index) ;
                goto Listening ;
            elsif command = "verack" then
                call verack(self, remote_peer_index) ;

```

```

    elsif command = "getblocks" then
        call getblocks(self, remote_peer_index);
        goto Listening;
    elsif command = "inv" then
        call inv(self, remote_peer_index);
        goto Listening;
    elsif command = "getdata" then
        call getdata(self, remote_peer_index);
    end if ;
end with ;
ListenerLoop:
    with remote_peer_index ∈ 1 .. Len(the_network[self].peer_set) do
        channels[self][remote_peer_index] :=
            [header ↦ defaultInitValue, payload ↦ defaultInitValue];
        goto Listening;
    end with ;
end process ;

```

A set of processes to synchronize each peer with the network.

```

process SYNCHRONIZER ∈ PeerProcessDiffId + 1 .. PeerProcessDiffId + Len(RunningBlockchain)
variables local_peer_index = self - PeerProcessDiffId, best_tip = 0;
begin

```

Announce:

The network must have at least two peer.

```

assert Len(the_network) ≥ 2;

```

The peer set size must be at least 1, ignoring the peers that are seeders only.

```

await Len(the_network[local_peer_index].peer_set) > 0;

```

Connect to each available peer we have.

```

with remote_peer_index ∈ 1 .. Len(the_network[local_peer_index].peer_set) do
    call announce(local_peer_index, remote_peer_index);
end with ;

```

RequestInventory:

```

with remote_peer_index ∈ 1 .. Len(the_network[local_peer_index].peer_set) do

```

Make sure the connection is established before requesting any block from this peer.

```

await the_network[local_peer_index].peer_set[remote_peer_index].established = TRUE;

```

Find the best tip among all peers.

```

if the_network[local_peer_index].peer_set[remote_peer_index].tip > best_tip then
    best_tip := the_network[local_peer_index].peer_set[remote_peer_index].tip;
end if ;

```

Wait for the peer channel to be empty before requesting new blocks.

```

await channels[local_peer_index][remote_peer_index].header = defaultInitValue
    ∧ channels[local_peer_index][remote_peer_index].payload = defaultInitValue ;

```

```

    Check if the local peer is behind the remote peer.
    if Ops!GetPeerTipByIndex(local_peer_index).height <
        the_network[local_peer_index].peer_set[remote_peer_index].tip then
        Request blocks.
        if Ops!GetPeerTipByIndex(local_peer_index).height = 0 then
            call request_blocks(⟨⟩, local_peer_index, remote_peer_index);
        else
            call request_blocks(
                ⟨Ops!GetPeerTipByIndex(local_peer_index).hash⟩,
                local_peer_index,
                remote_peer_index
            );
        end if ;
    end if ;
end with ;
CheckSync:
await Ops!GetPeerTipByIndex(local_peer_index).height > 0;
if Ops!GetPeerTipByIndex(local_peer_index).height < best_tip then
    goto RequestInventory;
else
    Make sure all connections are still established and all communication channels are empty
    with remote_peer_index ∈ 1 .. Len(the_network[local_peer_index].peer_set) do
        await the_network[local_peer_index].peer_set[remote_peer_index].established = TRUE
            ∧ channels[local_peer_index][remote_peer_index].header = defaultInitValue
            ∧ channels[local_peer_index][remote_peer_index].payload = defaultInitValue;
    end with ;
end if ;
end process ;
end algorithm ;

```

BEGIN TRANSLATION(chksum(pcal) = "fb1948d7" ∧ chksum(tla) = "bceb3955")
Parameter local_peer_id of procedure announce at line 155 col 20 changed to local_peer_id_
Parameter remote_peer_id of procedure announce at line 155 col 35 changed to remote_peer_id_
Parameter local_peer_id of procedure addr at line 170 col 16 changed to local_peer_id_a
Parameter remote_peer_id of procedure addr at line 170 col 31 changed to remote_peer_id_a
Parameter local_peer_id of procedure version at line 185 col 19 changed to local_peer_id_v
Parameter remote_peer_id of procedure version at line 185 col 34 changed to remote_peer_id_v
Parameter local_peer_id of procedure verack at line 199 col 18 changed to local_peer_id_ve
Parameter remote_peer_id of procedure verack at line 199 col 33 changed to remote_peer_id_ve
Parameter local_peer_id of procedure getblocks at line 207 col 21 changed to local_peer_id_g
Parameter remote_peer_id of procedure getblocks at line 207 col 36 changed to remote_peer_id_g
Parameter local_peer_id of procedure request_blocks at line 249 col 34 changed to local_peer_id_r
Parameter remote_peer_id of procedure request_blocks at line 249 col 49 changed to remote_peer_id_r
Parameter local_peer_id of procedure inv at line 262 col 15 changed to local_peer_id_i
Parameter remote_peer_id of procedure inv at line 262 col 30 changed to remote_peer_id_i

CONSTANT *defaultInitValue*
 VARIABLES *the_network*, *channels*, *pc*, *stack*
define statement
 LOCAL *Ops* \triangleq INSTANCE *Operators*

ExistsSyncPath \triangleq
 $\exists \text{peer1}, \text{peer2} \in 1 \dots \text{Len}(\text{RunningBlockchain}) :$
 $\diamond (\text{Ops!GetPeerTipByIndex}(\text{peer1}).\text{height} = \text{Ops!GetPeerTipByIndex}(\text{peer2}).\text{height})$

Liveness \triangleq
 $\forall \text{peer1}, \text{peer2} \in 1 \dots \text{Len}(\text{RunningBlockchain}) :$
 $\diamond (\text{Ops!GetPeerTipByIndex}(\text{peer1}).\text{height} = \text{Ops!GetPeerTipByIndex}(\text{peer2}).\text{height})$

ChainTipRespectsPeerSet \triangleq
 $\forall \text{peer} \in 1 \dots \text{Len}(\text{RunningBlockchain}) :$
 $\forall \text{remote_peer} \in 1 \dots \text{Len}(\text{the_network}[\text{peer}].\text{peer_set}) :$
 $\text{Ops!GetPeerTipByIndex}(\text{peer}).\text{height} \leq \text{Ops!GetPeerTipByIndex}(\text{remote_peer}).\text{height}$

ValidBlockPropagation \triangleq
 $\forall \text{peer} \in 1 \dots \text{Len}(\text{RunningBlockchain}) :$
 $\forall \text{block} \in \text{the_network}[\text{peer}].\text{blocks} :$
 $\text{block.height} \leq \text{Ops!GetPeerTipByIndex}(\text{peer}).\text{height}$

$$\begin{aligned}
\text{BlockOrdering} &\triangleq \\
&\forall \text{peer} \in 1 \dots \text{Len}(\text{RunningBlockchain}) : \\
&\quad \forall \text{block} \in \text{the_network}[\text{peer}].\text{blocks} : \\
&\quad \quad \text{IF } \text{block.height} < 2 \text{ THEN} \\
&\quad \quad \quad \text{TRUE} \\
&\quad \quad \text{ELSE} \\
&\quad \quad \quad \text{block.height} = \\
&\quad \quad \quad (\text{CHOOSE } b \in \text{the_network}[\text{peer}].\text{blocks} : b.\text{height} = \text{block.height} - 1).\text{height} + 1
\end{aligned}$$

$$\begin{aligned}
\text{SyncProgress} &\triangleq \\
&\forall \text{peer} \in 1 \dots \text{Len}(\text{RunningBlockchain}) : \\
&\quad \diamond (\text{Ops!GetPeerTipByIndex}(\text{peer}).\text{height} \geq \\
&\quad \quad \text{Ops!GetPeerTipByIndexAndNetwork}(\text{peer}, \text{RunningBlockchain}).\text{height})
\end{aligned}$$

$$\begin{aligned}
\text{ConnectionLimit} &\triangleq \\
&\forall \text{peer} \in 1 \dots \text{Len}(\text{RunningBlockchain}) : \\
&\quad \text{Len}(\text{the_network}[\text{peer}].\text{peer_set}) \leq \text{MaxConnectionsPerPeer}
\end{aligned}$$

$$\begin{aligned}
\text{ConsistentBlocksAcrossPeers} &\triangleq \\
&\forall \text{peer1}, \text{peer2} \in 1 \dots \text{Len}(\text{RunningBlockchain}) : \\
&\quad \text{peer1} \neq \text{peer2} \Rightarrow \\
&\quad \quad \forall \text{block1} \in \text{the_network}[\text{peer1}].\text{blocks}, \text{block2} \in \text{the_network}[\text{peer2}].\text{blocks} : \\
&\quad \quad \quad \text{block1.height} = \text{block2.height} \Rightarrow \\
&\quad \quad \quad \text{block1.hash} = \text{block2.hash} \wedge \text{block1.block} = \text{block2.block}
\end{aligned}$$

$$\begin{aligned}
\text{InductiveInvariant} &\triangleq \\
&\quad \wedge \text{ChainTipRespectsPeerSet}
\end{aligned}$$

\wedge *ValidBlockPropagation*
 \wedge *BlockOrdering*
 \wedge *SyncProgress*
 \wedge *ConnectionLimit*
 \wedge *ConsistentBlocksAcrossPeers*

VARIABLES *local_peer_id_*, *remote_peer_id_*, *local_peer_id_a*, *remote_peer_id_a*,
local_peer_id_v, *remote_peer_id_v*, *local_peer_id_ve*,
remote_peer_id_ve, *local_peer_id_g*, *remote_peer_id_g*, *found_blocks*,
hash_count, *block_header_hashes*, *remote_peer_blocks*, *start_height*,
end_height, *hashes*, *local_peer_id_r*, *remote_peer_id_r*,
local_peer_id_i, *remote_peer_id_i*, *local_peer_id*, *remote_peer_id*,
blocks_data, *command*, *local_peer_index*, *best_tip*

vars \triangleq \langle *the_network*, *channels*, *pc*, *stack*, *local_peer_id_*, *remote_peer_id_*,
local_peer_id_a, *remote_peer_id_a*, *local_peer_id_v*,
remote_peer_id_v, *local_peer_id_ve*, *remote_peer_id_ve*,
local_peer_id_g, *remote_peer_id_g*, *found_blocks*, *hash_count*,
block_header_hashes, *remote_peer_blocks*, *start_height*, *end_height*,
hashes, *local_peer_id_r*, *remote_peer_id_r*, *local_peer_id_i*,
remote_peer_id_i, *local_peer_id*, *remote_peer_id*, *blocks_data*,
command, *local_peer_index*, *best_tip* \rangle

ProcSet \triangleq $(1 \dots \text{Len}(\text{RunningBlockchain})) \cup (\text{PeerProcessDiffId} + 1 \dots \text{PeerProcessDiffId} + \text{Len}(\text{RunningBlockchain}))$

Init \triangleq *Global variables*
 \wedge *the_network* = *RunningBlockchain*
 \wedge *channels* = $[i \in 1 \dots \text{Len}(\text{the_network}) \mapsto$
 $[j \in 1 \dots \text{MaxConnectionsPerPeer} \mapsto [$
 $\text{header} \mapsto \text{defaultInitValue},$
 $\text{payload} \mapsto \text{defaultInitValue}$
 $]]$
 $]$
Procedure announce
 \wedge *local_peer_id_* = $[self \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
 \wedge *remote_peer_id_* = $[self \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
Procedure addr
 \wedge *local_peer_id_a* = $[self \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
 \wedge *remote_peer_id_a* = $[self \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
Procedure version
 \wedge *local_peer_id_v* = $[self \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
 \wedge *remote_peer_id_v* = $[self \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
Procedure verack
 \wedge *local_peer_id_ve* = $[self \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
 \wedge *remote_peer_id_ve* = $[self \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
Procedure getblocks

$\wedge \text{local_peer_id_g} = [\text{self} \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
 $\wedge \text{remote_peer_id_g} = [\text{self} \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
 $\wedge \text{found_blocks} = [\text{self} \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
 $\wedge \text{hash_count} = [\text{self} \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
 $\wedge \text{block_header_hashes} = [\text{self} \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
 $\wedge \text{remote_peer_blocks} = [\text{self} \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
 $\wedge \text{start_height} = [\text{self} \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
 $\wedge \text{end_height} = [\text{self} \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
Procedure request_blocks
 $\wedge \text{hashes} = [\text{self} \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
 $\wedge \text{local_peer_id_r} = [\text{self} \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
 $\wedge \text{remote_peer_id_r} = [\text{self} \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
Procedure inv
 $\wedge \text{local_peer_id_i} = [\text{self} \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
 $\wedge \text{remote_peer_id_i} = [\text{self} \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
Procedure getdata
 $\wedge \text{local_peer_id} = [\text{self} \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
 $\wedge \text{remote_peer_id} = [\text{self} \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
 $\wedge \text{blocks_data} = [\text{self} \in \text{ProcSet} \mapsto \text{defaultInitValue}]$
Process LISTENER
 $\wedge \text{command} = [\text{self} \in 1 \dots \text{Len}(\text{RunningBlockchain}) \mapsto \text{defaultInitValue}]$
Process SYNCHRONIZER
 $\wedge \text{local_peer_index} = [\text{self} \in \text{PeerProcessDiffId} + 1 \dots \text{PeerProcessDiffId} + \text{Len}(\text{RunningBlockchain}) \mapsto 0]$
 $\wedge \text{best_tip} = [\text{self} \in \text{PeerProcessDiffId} + 1 \dots \text{PeerProcessDiffId} + \text{Len}(\text{RunningBlockchain}) \mapsto 0]$
 $\wedge \text{stack} = [\text{self} \in \text{ProcSet} \mapsto \langle \rangle]$
 $\wedge \text{pc} = [\text{self} \in \text{ProcSet} \mapsto \text{CASE } \text{self} \in 1 \dots \text{Len}(\text{RunningBlockchain}) \rightarrow \text{"Listening"}$
 $\quad \square \quad \text{self} \in \text{PeerProcessDiffId} + 1 \dots \text{PeerProcessDiffId} + \text{Len}(\text{RunningBlockchain}) \rightarrow \text{"Listening"}$

$\text{SendAddrMsg}(\text{self}) \triangleq \wedge \text{pc}[\text{self}] = \text{"SendAddrMsg"}$
 $\wedge \text{channels}' = [\text{channels} \text{ EXCEPT } ![\text{local_peer_id_}[\text{self}]][\text{remote_peer_id_}[\text{self}]] =$

$\wedge \text{pc}' = [\text{pc} \text{ EXCEPT } ![\text{self}] = \text{Head}(\text{stack}[\text{self}]).\text{pc}]$
 $\wedge \text{local_peer_id_}' = [\text{local_peer_id_} \text{ EXCEPT } ![\text{self}] = \text{Head}(\text{stack}[\text{self}]).\text{local_peer_id_}]$
 $\wedge \text{remote_peer_id_}' = [\text{remote_peer_id_} \text{ EXCEPT } ![\text{self}] = \text{Head}(\text{stack}[\text{self}]).\text{remote_peer_id_}]$
 $\wedge \text{stack}' = [\text{stack} \text{ EXCEPT } ![\text{self}] = \text{Tail}(\text{stack}[\text{self}])]$
 $\wedge \text{UNCHANGED } \langle \text{the_network}, \text{local_peer_id_a},$
 $\quad \text{remote_peer_id_a}, \text{local_peer_id_v},$
 $\quad \text{remote_peer_id_v}, \text{local_peer_id_ve},$
 $\quad \text{remote_peer_id_ve}, \text{local_peer_id_g},$

remote_peer_id_g, *found_blocks*,
hash_count, *block_header_hashes*,
remote_peer_blocks, *start_height*,
end_height, *hashes*, *local_peer_id_r*,
remote_peer_id_r, *local_peer_id_i*,
remote_peer_id_i, *local_peer_id*,
remote_peer_id, *blocks_data*, *command*,
local_peer_index, *best_tip*

announce(*self*) \triangleq *SendAddrMsg*(*self*)

SendVersionMsg(*self*) \triangleq \wedge *pc*[*self*] = "SendVersionMsg"
 \wedge *channels*' = [*channels* EXCEPT ![*local_peer_id_a*[*self*]][*remote_peer_id_a*[*self*]] =

\wedge *pc*' = [*pc* EXCEPT ![*self*] = *Head*(*stack*[*self*])).*pc*
 \wedge *local_peer_id_a*' = [*local_peer_id_a* EXCEPT ![*self*] = *Head*(*stack*[*self*])).*local_peer_id_a*
 \wedge *remote_peer_id_a*' = [*remote_peer_id_a* EXCEPT ![*self*] = *Head*(*stack*[*self*])).*remote_peer_id_a*
 \wedge *stack*' = [*stack* EXCEPT ![*self*] = *Tail*(*stack*[*self*])]
 \wedge UNCHANGED \langle *the_network*, *local_peer_id_*,
remote_peer_id_, *local_peer_id_v*,
remote_peer_id_v, *local_peer_id_ve*,
remote_peer_id_ve, *local_peer_id_g*,
remote_peer_id_g, *found_blocks*,
hash_count, *block_header_hashes*,
remote_peer_blocks, *start_height*,
end_height, *hashes*, *local_peer_id_r*,
remote_peer_id_r, *local_peer_id_i*,
remote_peer_id_i, *local_peer_id*,
remote_peer_id, *blocks_data*, *command*,
local_peer_index, *best_tip* \rangle

addr(*self*) \triangleq *SendVersionMsg*(*self*)

HandleVersionMsg(*self*) \triangleq \wedge *pc*[*self*] = "HandleVersionMsg"
 \wedge *the_network*' = [*the_network* EXCEPT ![*local_peer_id_v*[*self*]].*peer_set*[*remote_peer_id_v*]
 \wedge *pc*' = [*pc* EXCEPT ![*self*] = "SendVerackMsg"]
 \wedge UNCHANGED \langle *channels*, *stack*, *local_peer_id_*,
remote_peer_id_, *local_peer_id_a*,
remote_peer_id_a, *local_peer_id_v*,
remote_peer_id_v, *local_peer_id_ve*,

$remote_peer_id_ve, local_peer_id_g,$
 $remote_peer_id_g, found_blocks,$
 $hash_count, block_header_hashes,$
 $remote_peer_blocks, start_height,$
 $end_height, hashes, local_peer_id_r,$
 $remote_peer_id_r, local_peer_id_i,$
 $remote_peer_id_i, local_peer_id,$
 $remote_peer_id, blocks_data, command,$
 $local_peer_index, best_tip\rangle$

$SendVerackMsg(self) \triangleq \wedge pc[self] = \text{"SendVerackMsg"}$
 $\wedge channels' = [channels \text{ EXCEPT } ![local_peer_id_v[self]][remote_peer_id_v[self]] =$

$\wedge pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]$
 $\wedge local_peer_id_v' = [local_peer_id_v \text{ EXCEPT } ![self] = Head(stack[self]).local_peer_id_v]$
 $\wedge remote_peer_id_v' = [remote_peer_id_v \text{ EXCEPT } ![self] = Head(stack[self]).remote_peer_id_v]$
 $\wedge stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$
 $\wedge \text{UNCHANGED } \langle the_network, local_peer_id_-,$
 $remote_peer_id_-, local_peer_id_a,$
 $remote_peer_id_a, local_peer_id_ve,$
 $remote_peer_id_ve, local_peer_id_g,$
 $remote_peer_id_g, found_blocks,$
 $hash_count, block_header_hashes,$
 $remote_peer_blocks, start_height,$
 $end_height, hashes, local_peer_id_r,$
 $remote_peer_id_r, local_peer_id_i,$
 $remote_peer_id_i, local_peer_id,$
 $remote_peer_id, blocks_data, command,$
 $local_peer_index, best_tip\rangle$

$version(self) \triangleq HandleVersionMsg(self) \vee SendVerackMsg(self)$

$HandleVerackMsg(self) \triangleq \wedge pc[self] = \text{"HandleVerackMsg"}$
 $\wedge the_network' = [the_network \text{ EXCEPT } ![local_peer_id_ve[self]].peer_set[remote_peer_id_ve[self]]]$
 $\wedge pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]$
 $\wedge local_peer_id_ve' = [local_peer_id_ve \text{ EXCEPT } ![self] = Head(stack[self]).local_peer_id_ve]$
 $\wedge remote_peer_id_ve' = [remote_peer_id_ve \text{ EXCEPT } ![self] = Head(stack[self]).remote_peer_id_ve]$
 $\wedge stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$
 $\wedge \text{UNCHANGED } \langle channels, local_peer_id_-,$
 $remote_peer_id_-, local_peer_id_a,$
 $remote_peer_id_a, local_peer_id_v,$
 $remote_peer_id_v, local_peer_id_g,$
 $remote_peer_id_g, found_blocks,$
 $hash_count, block_header_hashes,$

$remote_peer_blocks, start_height,$
 $end_height, hashes, local_peer_id_r,$
 $remote_peer_id_r, local_peer_id_i,$
 $remote_peer_id_i, local_peer_id,$
 $remote_peer_id, blocks_data, command,$
 $local_peer_index, best_tip\rangle$

$verack(self) \triangleq HandleVerackMsg(self)$

$HandleGetBlocksMsg(self) \triangleq \wedge pc[self] = \text{"HandleGetBlocksMsg"}$
 $\wedge hash_count' = [hash_count \text{ EXCEPT } ![self] = channels[local_peer_id_g[self]]]$
 $\wedge block_header_hashes' = [block_header_hashes \text{ EXCEPT } ![self] = channels[local_peer_id_g[self]]]$
 $\wedge remote_peer_blocks' = [remote_peer_blocks \text{ EXCEPT } ![self] = Ops!GetPeerBlocks]$
 $\wedge \text{IF } hash_count'[self] = 0$
 $\quad \text{THEN } \wedge start_height' = [start_height \text{ EXCEPT } ![self] = 1]$
 $\quad \text{ELSE } \wedge start_height' = [start_height \text{ EXCEPT } ![self] = Ops!FindBlockB]$
 $\wedge end_height' = [end_height \text{ EXCEPT } ![self] = start_height'[self] + (MaxGetBL$
 $\wedge found_blocks' = [found_blocks \text{ EXCEPT } ![self] = Ops!FindBlocks(remote_peer$
 $\wedge pc' = [pc \text{ EXCEPT } ![self] = \text{"SendInvMsg"}]$
 $\wedge \text{UNCHANGED } \langle the_network, channels, stack,$
 $\quad local_peer_id_l, remote_peer_id_l,$
 $\quad local_peer_id_a, remote_peer_id_a,$
 $\quad local_peer_id_v, remote_peer_id_v,$
 $\quad local_peer_id_ve,$
 $\quad remote_peer_id_ve, local_peer_id_g,$
 $\quad remote_peer_id_g, hashes,$
 $\quad local_peer_id_r, remote_peer_id_r,$
 $\quad local_peer_id_i, remote_peer_id_i,$
 $\quad local_peer_id, remote_peer_id,$
 $\quad blocks_data, command,$
 $\quad local_peer_index, best_tip\rangle$

$SendInvMsg(self) \triangleq \wedge pc[self] = \text{"SendInvMsg"}$
 $\wedge channels' = [channels \text{ EXCEPT } ![local_peer_id_g[self]][remote_peer_id_g[self]] =$

h
 pc

$]$
 $]]$

$\wedge pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]$

$$\begin{aligned}
& \wedge found_blocks' = [found_blocks \text{ EXCEPT } ![self] = Head(stack[self]).found_blocks] \\
& \wedge hash_count' = [hash_count \text{ EXCEPT } ![self] = Head(stack[self]).hash_count] \\
& \wedge block_header_hashes' = [block_header_hashes \text{ EXCEPT } ![self] = Head(stack[self]).block_header_hashes] \\
& \wedge remote_peer_blocks' = [remote_peer_blocks \text{ EXCEPT } ![self] = Head(stack[self]).remote_peer_blocks] \\
& \wedge start_height' = [start_height \text{ EXCEPT } ![self] = Head(stack[self]).start_height] \\
& \wedge end_height' = [end_height \text{ EXCEPT } ![self] = Head(stack[self]).end_height] \\
& \wedge local_peer_id_g' = [local_peer_id_g \text{ EXCEPT } ![self] = Head(stack[self]).local_peer_id_g] \\
& \wedge remote_peer_id_g' = [remote_peer_id_g \text{ EXCEPT } ![self] = Head(stack[self]).remote_peer_id_g] \\
& \wedge stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])] \\
& \wedge \text{UNCHANGED } \langle the_network, local_peer_id_-, \\
& \quad remote_peer_id_-, local_peer_id_a, \\
& \quad remote_peer_id_a, local_peer_id_v, \\
& \quad remote_peer_id_v, local_peer_id_ve, \\
& \quad remote_peer_id_ve, hashes, local_peer_id_r, \\
& \quad remote_peer_id_r, local_peer_id_i, \\
& \quad remote_peer_id_i, local_peer_id, \\
& \quad remote_peer_id, blocks_data, command, \\
& \quad local_peer_index, best_tip \rangle
\end{aligned}$$

$$getblocks(self) \triangleq HandleGetBlocksMsg(self) \vee SendInvMsg(self)$$

$$\begin{aligned}
SendGetBlocksMsg(self) \triangleq & \wedge pc[self] = \text{"SendGetBlocksMsg"} \\
& \wedge channels' = [channels \text{ EXCEPT } ![local_peer_id_r[self]][remote_peer_id_r[self]]]
\end{aligned}$$

$$\begin{aligned}
& \wedge pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc] \\
& \wedge hashes' = [hashes \text{ EXCEPT } ![self] = Head(stack[self]).hashes] \\
& \wedge local_peer_id_r' = [local_peer_id_r \text{ EXCEPT } ![self] = Head(stack[self]).local_peer_id_r] \\
& \wedge remote_peer_id_r' = [remote_peer_id_r \text{ EXCEPT } ![self] = Head(stack[self]).remote_peer_id_r] \\
& \wedge stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])] \\
& \wedge \text{UNCHANGED } \langle the_network, local_peer_id_-, \\
& \quad remote_peer_id_-, local_peer_id_a, \\
& \quad remote_peer_id_a, local_peer_id_v, \\
& \quad remote_peer_id_v, local_peer_id_ve, \\
& \quad remote_peer_id_ve, local_peer_id_g, \\
& \quad remote_peer_id_g, found_blocks, \\
& \quad hash_count, block_header_hashes, \\
& \quad remote_peer_blocks, start_height, \\
& \quad end_height, local_peer_id_i, \\
& \quad remote_peer_id_i, local_peer_id, \\
& \quad remote_peer_id, blocks_data, command, \\
& \quad local_peer_index, best_tip \rangle
\end{aligned}$$

remote_peer_id, *command*,
local_peer_index, *best_tip*)

IncorporateBlocks(*self*) \triangleq $\wedge pc[self] = \text{"IncorporateBlocks"}$
 $\wedge the_network' = [the_network \text{ EXCEPT } ![local_peer_id[self]].blocks = the_network]$
 $\wedge pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]$
 $\wedge blocks_data' = [blocks_data \text{ EXCEPT } ![self] = Head(stack[self]).blocks_data]$
 $\wedge local_peer_id' = [local_peer_id \text{ EXCEPT } ![self] = Head(stack[self]).local_peer_id]$
 $\wedge remote_peer_id' = [remote_peer_id \text{ EXCEPT } ![self] = Head(stack[self]).remote_peer_id]$
 $\wedge stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$
 $\wedge \text{UNCHANGED } \langle channels, local_peer_id_-,$
 $remote_peer_id_-, local_peer_id_a,$
 $remote_peer_id_a, local_peer_id_v,$
 $remote_peer_id_v, local_peer_id_ve,$
 $remote_peer_id_ve, local_peer_id_g,$
 $remote_peer_id_g, found_blocks,$
 $hash_count, block_header_hashes,$
 $remote_peer_blocks, start_height,$
 $end_height, hashes, local_peer_id_r,$
 $remote_peer_id_r, local_peer_id_i,$
 $remote_peer_id_i, command,$
 $local_peer_index, best_tip \rangle$

getdata(*self*) \triangleq *HandleGetDataMsg*(*self*) \vee *IncorporateBlocks*(*self*)

Listening(*self*) \triangleq $\wedge pc[self] = \text{"Listening"}$
 $\wedge Len(the_network) \geq 2$
 $\wedge \exists remote_peer_index \in 1 \dots Len(the_network[self].peer_set) :$
 $\text{IF } channels[self][remote_peer_index].header = defaultInitValue$
 $\text{THEN } \wedge pc' = [pc \text{ EXCEPT } ![self] = \text{"Listening"}]$
 $\text{ELSE } \wedge pc' = [pc \text{ EXCEPT } ![self] = \text{"Requests"}]$
 $\wedge \text{UNCHANGED } \langle the_network, channels, stack,$
 $local_peer_id_-, remote_peer_id_-,$
 $local_peer_id_a, remote_peer_id_a,$
 $local_peer_id_v, remote_peer_id_v,$
 $local_peer_id_ve, remote_peer_id_ve,$
 $local_peer_id_g, remote_peer_id_g,$
 $found_blocks, hash_count,$
 $block_header_hashes, remote_peer_blocks,$
 $start_height, end_height, hashes,$
 $local_peer_id_r, remote_peer_id_r,$
 $local_peer_id_i, remote_peer_id_i,$
 $local_peer_id, remote_peer_id, blocks_data,$
 $command, local_peer_index, best_tip \rangle$

Requests(*self*) \triangleq $\wedge pc[self] = \text{"Requests"}$

$$\begin{aligned}
& \wedge \exists \textit{remote_peer_index} \in 1 \dots \textit{Len}(\textit{the_network}[\textit{self}].\textit{peer_set}) : \\
& \quad \wedge \textit{channels}[\textit{self}][\textit{remote_peer_index}].\textit{header} \neq \textit{defaultInitValue} \\
& \quad \wedge \textit{command}' = [\textit{command} \text{ EXCEPT } ![\textit{self}] = \textit{channels}[\textit{self}][\textit{remote_peer_index}].\textit{header}. \\
& \quad \wedge \text{IF } \textit{command}'[\textit{self}] = \text{"addr"} \\
& \quad \quad \text{THEN } \wedge \wedge \textit{local_peer_id_a}' = [\textit{local_peer_id_a} \text{ EXCEPT } ![\textit{self}] = \textit{self}] \\
& \quad \quad \quad \wedge \textit{remote_peer_id_a}' = [\textit{remote_peer_id_a} \text{ EXCEPT } ![\textit{self}] = \textit{remote_peer_id_a}] \\
& \quad \quad \quad \wedge \textit{stack}' = [\textit{stack} \text{ EXCEPT } ![\textit{self}] = \langle [\textit{procedure} \mapsto \text{"addr"}, \\
& \quad \quad \quad \quad \quad \quad \textit{pc} \mapsto \text{"Listening"}, \\
& \quad \quad \quad \quad \quad \quad \textit{local_peer_id_a} \mapsto \textit{local_peer_id_a}, \\
& \quad \quad \quad \quad \quad \quad \textit{remote_peer_id_a} \mapsto \textit{remote_peer_id_a}, \\
& \quad \quad \quad \quad \quad \quad \circ \textit{stack}[\textit{self}]] \rangle] \\
& \quad \quad \quad \wedge \textit{pc}' = [\textit{pc} \text{ EXCEPT } ![\textit{self}] = \text{"SendVersionMsg"}] \\
& \quad \quad \quad \wedge \text{UNCHANGED } \langle \textit{local_peer_id_v}, \\
& \quad \quad \quad \quad \textit{remote_peer_id_v}, \\
& \quad \quad \quad \quad \textit{local_peer_id_ve}, \\
& \quad \quad \quad \quad \textit{remote_peer_id_ve}, \\
& \quad \quad \quad \quad \textit{local_peer_id_g}, \\
& \quad \quad \quad \quad \textit{remote_peer_id_g}, \\
& \quad \quad \quad \quad \textit{found_blocks}, \textit{hash_count}, \\
& \quad \quad \quad \quad \textit{block_header_hashes}, \\
& \quad \quad \quad \quad \textit{remote_peer_blocks}, \\
& \quad \quad \quad \quad \textit{start_height}, \textit{end_height}, \\
& \quad \quad \quad \quad \textit{local_peer_id_i}, \\
& \quad \quad \quad \quad \textit{remote_peer_id_i}, \\
& \quad \quad \quad \quad \textit{local_peer_id}, \\
& \quad \quad \quad \quad \textit{remote_peer_id}, \textit{blocks_data} \rangle \\
& \quad \quad \text{ELSE } \wedge \text{IF } \textit{command}'[\textit{self}] = \text{"version"} \\
& \quad \quad \quad \text{THEN } \wedge \wedge \textit{local_peer_id_v}' = [\textit{local_peer_id_v} \text{ EXCEPT } ![\textit{self}] = \textit{self}] \\
& \quad \quad \quad \quad \wedge \textit{remote_peer_id_v}' = [\textit{remote_peer_id_v} \text{ EXCEPT } ![\textit{self}] = \textit{remote_peer_id_v}] \\
& \quad \quad \quad \quad \wedge \textit{stack}' = [\textit{stack} \text{ EXCEPT } ![\textit{self}] = \langle [\textit{procedure} \mapsto \text{"version"}, \\
& \quad \quad \quad \quad \quad \quad \textit{pc} \mapsto \text{"Listening"}, \\
& \quad \quad \quad \quad \quad \quad \textit{local_peer_id_v} \mapsto \textit{local_peer_id_v}, \\
& \quad \quad \quad \quad \quad \quad \textit{remote_peer_id_v} \mapsto \textit{remote_peer_id_v}, \\
& \quad \quad \quad \quad \quad \quad \circ \textit{stack}[\textit{self}]] \rangle] \\
& \quad \quad \quad \quad \wedge \textit{pc}' = [\textit{pc} \text{ EXCEPT } ![\textit{self}] = \text{"HandleVersionMsg"}] \\
& \quad \quad \quad \quad \wedge \text{UNCHANGED } \langle \textit{local_peer_id_ve}, \\
& \quad \quad \quad \quad \quad \textit{remote_peer_id_ve}, \\
& \quad \quad \quad \quad \quad \textit{local_peer_id_g}, \\
& \quad \quad \quad \quad \quad \textit{remote_peer_id_g}, \\
& \quad \quad \quad \quad \quad \textit{found_blocks}, \\
& \quad \quad \quad \quad \quad \textit{hash_count}, \\
& \quad \quad \quad \quad \quad \textit{block_header_hashes}, \\
& \quad \quad \quad \quad \quad \textit{remote_peer_blocks}, \\
& \quad \quad \quad \quad \quad \textit{start_height}, \\
& \quad \quad \quad \quad \quad \textit{end_height}, \\
& \quad \quad \quad \quad \quad \textit{local_peer_id_i}, \\
& \quad \quad \quad \quad \quad \textit{remote_peer_id_i}, \\
& \quad \quad \quad \quad \quad \textit{local_peer_id}, \\
& \quad \quad \quad \quad \quad \textit{remote_peer_id}, \textit{blocks_data} \rangle
\end{aligned}$$

```

        local_peer_id_i,
        remote_peer_id_i,
        local_peer_id,
        remote_peer_id,
        blocks_data)
ELSE  ∧ IF command'[self] = “verack”
      THEN  ∧ ∧ local_peer_id_ve' = [local_peer_id_ve EXCEPT
        ∧ remote_peer_id_ve' = [remote_peer_id_ve EXCEPT
        ∧ stack' = [stack EXCEPT ![self] = ⟨[procedure
          pc
          local_peer_id
          remote_peer_id
          found_blocks
          hash_count
          block_header_hashes
          remote_peer_blocks
          start_height
          end_height
          local_peer_id_i
          remote_peer_id_i
          local_peer_id
          remote_peer_id
          blocks_data⟩
        ∧ pc' = [pc EXCEPT ![self] = “HandleVerackMsg”
        ∧ UNCHANGED ⟨local_peer_id_g,
          remote_peer_id_g,
          found_blocks,
          hash_count,
          block_header_hashes,
          remote_peer_blocks,
          start_height,
          end_height,
          local_peer_id_i,
          remote_peer_id_i,
          local_peer_id,
          remote_peer_id,
          blocks_data⟩
      ELSE  ∧ IF command'[self] = “getblocks”
            THEN  ∧ ∧ local_peer_id_g' = [local_peer_id_g EXCEPT
              ∧ remote_peer_id_g' = [remote_peer_id_g EXCEPT
              ∧ stack' = [stack EXCEPT ![self] =

```

```

    ∧ found_blocks' = [found_blocks EXCEPT
    ∧ hash_count' = [hash_count EXCEPT
    ∧ block_header_hashes' = [block_header_hashes EXCEPT
    ∧ remote_peer_blocks' = [remote_peer_blocks EXCEPT

```

$$\begin{aligned}
& \wedge start_height' = [start_height \text{ EXCEPT } \\
& \wedge end_height' = [end_height \text{ EXCEPT } \\
& \wedge pc' = [pc \text{ EXCEPT } ![self] = \text{"Handle"} \\
& \wedge \text{UNCHANGED } \langle local_peer_id_i, \\
& \quad remote_peer_id_i, \\
& \quad local_peer_id, \\
& \quad remote_peer_id, \\
& \quad blocks_data \rangle \\
\text{ELSE } & \wedge \text{IF } command'[self] = \text{"inv"} \\
& \quad \text{THEN } \wedge \wedge local_peer_id_i' = [\\
& \quad \wedge remote_peer_id_i' = [\\
& \quad \wedge stack' = [stack \text{ EXCEPT }
\end{aligned}$$

$$\begin{aligned}
& \wedge pc' = [pc \text{ EXCEPT } ![self] \\
& \wedge \text{UNCHANGED } \langle local_peer_id_i, \\
& \quad remote_peer_id_i, \\
& \quad blocks_data \rangle \\
\text{ELSE } & \wedge \text{IF } command'[self] = \text{"prop"} \\
& \quad \text{THEN } \wedge \wedge local_peer_id_i' = [\\
& \quad \wedge remote_peer_id_i' = [\\
& \quad \wedge stack' = [stack \text{ EXCEPT }
\end{aligned}$$

$$\begin{aligned}
& \wedge blocks_data' = [blocks_data \text{ EXCEPT } \\
& \wedge pc' = [pc \text{ EXCEPT } ![self] \\
\text{ELSE } & \wedge pc' = [pc \text{ EXCEPT } ![self] \\
& \wedge \text{UNCHANGED } \langle local_peer_id_i,
\end{aligned}$$

$$\begin{aligned}
& \wedge \text{UNCHANGED } \langle local_peer_id_i, \\
& \quad remote_peer_id_i, \\
& \wedge \text{UNCHANGED } \langle local_peer_id_g, \\
& \quad remote_peer_id_g, \\
& \quad found_blocks, \\
& \quad hash_count, \\
& \quad block_header_hashes, \\
& \quad remote_peer_blocks, \\
& \quad start_height,
\end{aligned}$$

$$\begin{aligned}
& \text{end_height}\rangle \\
& \wedge \text{UNCHANGED } \langle \text{local_peer_id_ve}, \\
& \quad \text{remote_peer_id_ve}\rangle \\
& \wedge \text{UNCHANGED } \langle \text{local_peer_id_v}, \\
& \quad \text{remote_peer_id_v}\rangle \\
& \wedge \text{UNCHANGED } \langle \text{local_peer_id_a}, \\
& \quad \text{remote_peer_id_a}\rangle \\
& \wedge \text{UNCHANGED } \langle \text{the_network}, \text{channels}, \text{local_peer_id_}, \\
& \quad \text{remote_peer_id_}, \text{hashes}, \text{local_peer_id_r}, \\
& \quad \text{remote_peer_id_r}, \text{local_peer_index}, \text{best_tip}\rangle \\
\text{ListenerLoop}(self) & \triangleq \wedge pc[self] = \text{"ListenerLoop"} \\
& \wedge \exists \text{remote_peer_index} \in 1 \dots \text{Len}(\text{the_network}[self].\text{peer_set}) : \\
& \quad \wedge \text{channels}' = [\text{channels} \text{ EXCEPT } ![self][\text{remote_peer_index}] = [\text{header} \mapsto \text{defaultIn}]] \\
& \quad \wedge pc' = [pc \text{ EXCEPT } ![self] = \text{"Listening"}] \\
& \wedge \text{UNCHANGED } \langle \text{the_network}, \text{stack}, \text{local_peer_id_}, \\
& \quad \text{remote_peer_id_}, \text{local_peer_id_a}, \\
& \quad \text{remote_peer_id_a}, \text{local_peer_id_v}, \\
& \quad \text{remote_peer_id_v}, \text{local_peer_id_ve}, \\
& \quad \text{remote_peer_id_ve}, \text{local_peer_id_g}, \\
& \quad \text{remote_peer_id_g}, \text{found_blocks}, \\
& \quad \text{hash_count}, \text{block_header_hashes}, \\
& \quad \text{remote_peer_blocks}, \text{start_height}, \\
& \quad \text{end_height}, \text{hashes}, \text{local_peer_id_r}, \\
& \quad \text{remote_peer_id_r}, \text{local_peer_id_i}, \\
& \quad \text{remote_peer_id_i}, \text{local_peer_id}, \\
& \quad \text{remote_peer_id}, \text{blocks_data}, \text{command}, \\
& \quad \text{local_peer_index}, \text{best_tip}\rangle \\
\text{LISTENER}(self) & \triangleq \text{Listening}(self) \vee \text{Requests}(self) \vee \text{ListenerLoop}(self) \\
\text{Announce}(self) & \triangleq \wedge pc[self] = \text{"Announce"} \\
& \wedge \text{Assert}(\text{Len}(\text{the_network}) \geq 2, \\
& \quad \text{"Failure of assertion at line 335, column 9."}) \\
& \wedge \text{Len}(\text{the_network}[\text{local_peer_index}[self]].\text{peer_set}) > 0 \\
& \wedge \exists \text{remote_peer_index} \in 1 \dots \text{Len}(\text{the_network}[\text{local_peer_index}[self]].\text{peer_set}) : \\
& \quad \wedge \wedge \text{local_peer_id_}' = [\text{local_peer_id_} \text{ EXCEPT } ![self] = \text{local_peer_index}[self]] \\
& \quad \wedge \text{remote_peer_id_}' = [\text{remote_peer_id_} \text{ EXCEPT } ![self] = \text{remote_peer_index}] \\
& \quad \wedge \text{stack}' = [\text{stack} \text{ EXCEPT } ![self] = \langle [\text{procedure} \mapsto \text{"announce"}, \\
& \quad \quad \text{pc} \mapsto \text{"RequestInventory"}, \\
& \quad \quad \text{local_peer_id_} \mapsto \text{local_peer_id_}[self], \\
& \quad \quad \text{remote_peer_id_} \mapsto \text{remote_peer_id_}[self]] \rangle \\
& \quad \quad \circ \text{stack}[self]] \\
& \quad \wedge pc' = [pc \text{ EXCEPT } ![self] = \text{"SendAddrMsg"}] \\
& \wedge \text{UNCHANGED } \langle \text{the_network}, \text{channels}, \text{local_peer_id_a}, \\
& \quad \text{remote_peer_id_a}, \text{local_peer_id_v},
\end{aligned}$$

$remote_peer_id_v, local_peer_id_ve,$
 $remote_peer_id_ve, local_peer_id_g,$
 $remote_peer_id_g, found_blocks, hash_count,$
 $block_header_hashes, remote_peer_blocks,$
 $start_height, end_height, hashes,$
 $local_peer_id_r, remote_peer_id_r,$
 $local_peer_id_i, remote_peer_id_i,$
 $local_peer_id, remote_peer_id, blocks_data,$
 $command, local_peer_index, best_tip$

$RequestInventory(self) \triangleq \wedge pc[self] = \text{"RequestInventory"}$
 $\wedge \exists remote_peer_index \in 1 \dots Len(the_network[local_peer_index[self]].peer_set) :$
 $\wedge the_network[local_peer_index[self]].peer_set[remote_peer_index].established$
 $\wedge \text{IF } the_network[local_peer_index[self]].peer_set[remote_peer_index].tip > best_tip$
 $\quad \text{THEN } \wedge best_tip' = [best_tip \text{ EXCEPT } ![self] = the_network[local_peer_index[self]].tip]$
 $\quad \text{ELSE } \wedge \text{TRUE}$
 $\quad \wedge \text{UNCHANGED } best_tip$
 $\wedge channels[local_peer_index[self]][remote_peer_index].header = defaultInitV$
 $\wedge channels[local_peer_index[self]][remote_peer_index].payload = defaultInitV$
 $\wedge \text{IF } Ops!GetPeerTipByIndex(local_peer_index[self]).height <$
 $\quad the_network[local_peer_index[self]].peer_set[remote_peer_index].tip$
 $\quad \text{THEN } \wedge \text{IF } Ops!GetPeerTipByIndex(local_peer_index[self]).height = 0$
 $\quad \quad \text{THEN } \wedge \wedge hashes' = [hashes \text{ EXCEPT } ![self] = \langle \rangle]$
 $\quad \quad \wedge local_peer_id_r' = [local_peer_id_r \text{ EXCEPT } ![self] = \langle \rangle]$
 $\quad \quad \wedge remote_peer_id_r' = [remote_peer_id_r \text{ EXCEPT } ![self] = \langle \rangle]$
 $\quad \quad \wedge stack' = [stack \text{ EXCEPT } ![self] = \langle [procedure \mapsto$
 $\quad \quad \quad pc \mapsto$
 $\quad \quad \quad hashes \mapsto$
 $\quad \quad \quad local_peer_id \mapsto$
 $\quad \quad \quad remote_peer_id \mapsto$
 $\quad \quad \quad \circ stack[self]]]$
 $\quad \quad \wedge pc' = [pc \text{ EXCEPT } ![self] = \text{"SendGetBlocksMsg"}]$
 $\quad \text{ELSE } \wedge \wedge hashes' = [hashes \text{ EXCEPT } ![self] = \langle Ops!GetPeerTipByIndex(local_peer_index[self]).tip \rangle]$
 $\quad \quad \wedge local_peer_id_r' = [local_peer_id_r \text{ EXCEPT } ![self] = \langle \rangle]$
 $\quad \quad \wedge remote_peer_id_r' = [remote_peer_id_r \text{ EXCEPT } ![self] = \langle \rangle]$
 $\quad \quad \wedge stack' = [stack \text{ EXCEPT } ![self] = \langle [procedure \mapsto$
 $\quad \quad \quad pc \mapsto$
 $\quad \quad \quad hashes \mapsto$
 $\quad \quad \quad local_peer_id \mapsto$
 $\quad \quad \quad remote_peer_id \mapsto$
 $\quad \quad \quad \circ stack[self]]]$
 $\quad \quad \wedge pc' = [pc \text{ EXCEPT } ![self] = \text{"SendGetBlocksMsg"}]$
 $\text{ELSE } \wedge pc' = [pc \text{ EXCEPT } ![self] = \text{"CheckSync"}]$
 $\quad \wedge \text{UNCHANGED } \langle stack, hashes,$
 $\quad \quad local_peer_id_r,$

$$\begin{aligned}
& \text{remote_peer_id_r}) \\
& \wedge \text{UNCHANGED } \langle \text{the_network}, \text{channels}, \\
& \quad \text{local_peer_id_}, \text{remote_peer_id_}, \\
& \quad \text{local_peer_id_a}, \text{remote_peer_id_a}, \\
& \quad \text{local_peer_id_v}, \text{remote_peer_id_v}, \\
& \quad \text{local_peer_id_ve}, \text{remote_peer_id_ve}, \\
& \quad \text{local_peer_id_g}, \text{remote_peer_id_g}, \\
& \quad \text{found_blocks}, \text{hash_count}, \\
& \quad \text{block_header_hashes}, \\
& \quad \text{remote_peer_blocks}, \text{start_height}, \\
& \quad \text{end_height}, \text{local_peer_id_i}, \\
& \quad \text{remote_peer_id_i}, \text{local_peer_id}, \\
& \quad \text{remote_peer_id}, \text{blocks_data}, \text{command}, \\
& \quad \text{local_peer_index} \rangle \\
\text{CheckSync}(self) & \triangleq \wedge pc[self] = \text{"CheckSync"} \\
& \wedge \text{Ops! GetPeerTipByIndex}(\text{local_peer_index}[self]).\text{height} > 0 \\
& \wedge \text{IF } \text{Ops! GetPeerTipByIndex}(\text{local_peer_index}[self]).\text{height} < \text{best_tip}[self] \\
& \quad \text{THEN } \wedge pc' = [pc \text{ EXCEPT } ![self] = \text{"RequestInventory"}] \\
& \quad \text{ELSE } \wedge \exists \text{remote_peer_index} \in 1 \dots \text{Len}(\text{the_network}[\text{local_peer_index}[self]].\text{peer_set}) \\
& \quad \quad \text{the_network}[\text{local_peer_index}[self]].\text{peer_set}[\text{remote_peer_index}].\text{established} \\
& \quad \quad \wedge \text{channels}[\text{local_peer_index}[self]][\text{remote_peer_index}].\text{header} = \text{defaultHeader} \\
& \quad \quad \wedge \text{channels}[\text{local_peer_index}[self]][\text{remote_peer_index}].\text{payload} = \text{defaultPayload} \\
& \quad \wedge pc' = [pc \text{ EXCEPT } ![self] = \text{"Done"}] \\
& \wedge \text{UNCHANGED } \langle \text{the_network}, \text{channels}, \text{stack}, \\
& \quad \text{local_peer_id_}, \text{remote_peer_id_}, \\
& \quad \text{local_peer_id_a}, \text{remote_peer_id_a}, \\
& \quad \text{local_peer_id_v}, \text{remote_peer_id_v}, \\
& \quad \text{local_peer_id_ve}, \text{remote_peer_id_ve}, \\
& \quad \text{local_peer_id_g}, \text{remote_peer_id_g}, \\
& \quad \text{found_blocks}, \text{hash_count}, \\
& \quad \text{block_header_hashes}, \text{remote_peer_blocks}, \\
& \quad \text{start_height}, \text{end_height}, \text{hashes}, \\
& \quad \text{local_peer_id_r}, \text{remote_peer_id_r}, \\
& \quad \text{local_peer_id_i}, \text{remote_peer_id_i}, \\
& \quad \text{local_peer_id}, \text{remote_peer_id}, \text{blocks_data}, \\
& \quad \text{command}, \text{local_peer_index}, \text{best_tip} \rangle \\
\text{SYNCHRONIZER}(self) & \triangleq \text{Announce}(self) \vee \text{RequestInventory}(self) \\
& \quad \vee \text{CheckSync}(self) \\
& \text{Allow infinite stuttering to prevent deadlock on termination.} \\
\text{Terminating} & \triangleq \wedge \forall self \in \text{ProcSet} : pc[self] = \text{"Done"} \\
& \quad \wedge \text{UNCHANGED vars} \\
\text{Next} & \triangleq (\exists self \in \text{ProcSet} : \vee \text{announce}(self) \vee \text{addr}(self))
\end{aligned}$$

$$\begin{aligned}
& \vee \text{version}(\text{self}) \vee \text{verack}(\text{self}) \\
& \vee \text{getblocks}(\text{self}) \vee \text{request_blocks}(\text{self}) \\
& \vee \text{inv}(\text{self}) \vee \text{getdata}(\text{self})) \\
& \vee (\exists \text{self} \in 1 \dots \text{Len}(\text{RunningBlockchain}) : \text{LISTENER}(\text{self})) \\
& \vee (\exists \text{self} \in \text{PeerProcessDiffId} + 1 \dots \text{PeerProcessDiffId} + \text{Len}(\text{RunningBlockchain}) : \text{SYNCHRONIZED}(\text{self})) \\
& \vee \text{Terminating}
\end{aligned}$$

$$\text{Spec} \triangleq \text{Init} \wedge \Box[\text{Next}]_{\text{vars}}$$

$$\text{Termination} \triangleq \Diamond(\forall \text{self} \in \text{ProcSet} : \text{pc}[\text{self}] = \text{"Done"})$$

END TRANSLATION