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.

$channels = [i \in 1 \,..\, Len(the\_network) \mapsto$
$\quad [j \in 1 \,..\, MaxConnectionsPerPeer \mapsto [$
$\quad\quad header \mapsto defaultInitValue,$
$\quad\quad payload \mapsto defaultInitValue$
$\quad]]$
$]$ **;**

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.

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

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.

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

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.

$ChainTipRespectsPeerSet \triangleq$
$\forall\, peer \in 1 \,..\, Len(RunningBlockchain) :$
$\forall\, remote\_peer \in 1 \,..\, Len(the\_network[peer].peer\_set) :$
$Ops!GetPeerTipByIndex(peer).height \leq Ops!GetPeerTipByIndex(remote\_peer).height$

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.

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

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.

$BlockOrdering \triangleq$
$\forall\, peer \in 1 \,..\, Len(RunningBlockchain) :$
$\forall\, block \in the\_network[peer].blocks :$
IF $block.height < 2$ THEN
TRUE
ELSE
$block.height =$
$(\text{CHOOSE } b \in the\_network[peer].blocks : b.height = block.height - 1).height + 1$

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.

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

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.

$ConnectionLimit \triangleq$
$\forall\, peer \in 1 \,..\, Len(RunningBlockchain) :$
$Len(the\_network[peer].peer\_set) \leq MaxConnectionsPerPeer$

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.

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

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 ↦ [*command_name* ↦ "addr"],
payload ↦ [
address_count ↦ 1,
Only a single address is supported.
addresses ↦ *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 ↦ [command_name ↦ "version"],
            payload ↦ [
                addr_recv ↦ the_network[local_peer_id].peer,
                addr_trans ↦ the_network[local_peer_id].peer_set[remote_peer_id].address,
                start_height ↦
                    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 ↦ [command_name ↦ "verack"],
            payload ↦ 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 ↦ [command_name ↦ "getdata"],
            payload ↦ 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 ∈ 1 .. Len(channels[local_peer_id][remote_peer_id].payload.inventory) ↦
            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 ∪ ToSet(blocks_data) ;
    return ;
end procedure ;

A set of listener process for each peer to listen to incoming messages and act accordingly.
process LISTENER ∈ 1 .. Len(RunningBlockchain)
variables command ;
begin
    Listening:
        await Len(the_network) ≥ 2 ;
        with remote_peer_index ∈ 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 ∈ 1 .. Len(the_network[self].peer_set) do
            await channels[self][remote_peer_index].header ≠ 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 $Operators$

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

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

$ChainTipRespectsPeerSet \triangleq$
$\forall\, peer \in 1 \,..\, Len(RunningBlockchain) :$
$\forall\, remote\_peer \in 1 \,..\, Len(the\_network[peer].peer\_set) :$
$Ops!GetPeerTipByIndex(peer).height \leq Ops!GetPeerTipByIndex(remote\_peer).height$

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

$$
\begin{array}{l}
BlockOrdering \triangleq \\
\quad \forall\, peer \in 1 \,..\, Len(RunningBlockchain) : \\
\qquad \forall\, block \in the\_network[peer].blocks : \\
\qquad\quad \text{IF } block.height < 2 \text{ THEN} \\
\qquad\qquad \text{TRUE} \\
\qquad\quad \text{ELSE} \\
\qquad\qquad block.height = \\
\qquad\qquad\quad (\text{CHOOSE } b \in the\_network[peer].blocks : b.height = block.height - 1).height + 1
\end{array}
$$

$$
\begin{array}{l}
SyncProgress \triangleq \\
\quad \forall\, peer \in 1 \,..\, Len(RunningBlockchain) : \\
\qquad \Diamond(Ops!GetPeerTipByIndex(peer).height \geq \\
\qquad\quad Ops!GetPeerTipByIndexAndNetwork(peer,\ RunningBlockchain).height)
\end{array}
$$

$$
\begin{array}{l}
ConnectionLimit \triangleq \\
\quad \forall\, peer \in 1 \,..\, Len(RunningBlockchain) : \\
\qquad Len(the\_network[peer].peer\_set) \leq MaxConnectionsPerPeer
\end{array}
$$

$$
\begin{array}{l}
ConsistentBlocksAcrossPeers \triangleq \\
\quad \forall\, peer1,\ peer2 \in 1 \,..\, Len(RunningBlockchain) : \\
\qquad peer1 \neq peer2 \Rightarrow \\
\qquad\quad \forall\, block1 \in the\_network[peer1].blocks,\ block2 \in the\_network[peer2].blocks : \\
\qquad\qquad block1.height = block2.height \Rightarrow \\
\qquad\qquad\quad block1.hash = block2.hash \wedge block1.block = block2.block
\end{array}
$$

$$
\begin{array}{l}
InductiveInvariant \triangleq \\
\quad \wedge\, ChainTipRespectsPeerSet
\end{array}
$$

$\land$ *ValidBlockPropagation*
$\land$ *BlockOrdering*
$\land$ *SyncProgress*
$\land$ *ConnectionLimit*
$\land$ *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*$\rangle$

$ProcSet \triangleq (1 \,..\, Len(RunningBlockchain)) \cup (PeerProcessDiffId + 1 \,..\, PeerProcessDiffId + Len(RunningBloc$

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

$\land\ local\_peer\_id\_g = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ remote\_peer\_id\_g = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ found\_blocks = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ hash\_count = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ block\_header\_hashes = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ remote\_peer\_blocks = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ start\_height = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ end\_height = [self \in ProcSet \mapsto defaultInitValue]$
Procedure request_blocks
$\land\ hashes = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ local\_peer\_id\_r = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ remote\_peer\_id\_r = [self \in ProcSet \mapsto defaultInitValue]$
Procedure inv
$\land\ local\_peer\_id\_i = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ remote\_peer\_id\_i = [self \in ProcSet \mapsto defaultInitValue]$
Procedure getdata
$\land\ local\_peer\_id = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ remote\_peer\_id = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ blocks\_data = [self \in ProcSet \mapsto defaultInitValue]$
Process LISTENER
$\land\ command = [self \in 1 \,..\, Len(RunningBlockchain) \mapsto defaultInitValue]$
Process SYNCHRONIZER
$\land\ local\_peer\_index = [self \in PeerProcessDiffId + 1 \,..\, PeerProcessDiffId + Len(RunningBlockchain) \mapsto$
$\land\ best\_tip = [self \in PeerProcessDiffId + 1 \,..\, PeerProcessDiffId + Len(RunningBlockchain) \mapsto 0]$
$\land\ stack = [self \in ProcSet \mapsto \langle\rangle]$
$\land\ pc = [self \in ProcSet \mapsto \text{CASE } self \in 1 \,..\, Len(RunningBlockchain) \to \text{“Listening”}$
$\square \quad self \in PeerProcessDiffId + 1 \,..\, PeerProcessDiffId + Len(RunningBlo$

$SendAddrMsg(self) \triangleq \land\ pc[self] = \text{“SendAddrMsg”}$
$\land\ channels' = [channels \text{ EXCEPT } ![local\_peer\_id\_[self]][remote\_peer\_id\_[self]] =$
he
pa
]
]]
$\land\ pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]$
$\land\ local\_peer\_id\_' = [local\_peer\_id\_ \text{ EXCEPT } ![self] = Head(stack[self]).local\_peer\_id\_]$
$\land\ remote\_peer\_id\_' = [remote\_peer\_id\_ \text{ EXCEPT } ![self] = Head(stack[self]).remote\_pe$
$\land\ stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$
$\land\ \text{UNCHANGED } \langle the\_network,\ local\_peer\_id\_a,$
$remote\_peer\_id\_a,\ local\_peer\_id\_v,$
$remote\_peer\_id\_v,\ local\_peer\_id\_ve,$
$remote\_peer\_id\_ve,\ local\_peer\_id\_g,$

$remote\_peer\_id\_g$, $found\_blocks$,
$hash\_count$, $block\_header\_hashes$,
$remote\_peer\_blocks$, $start\_height$,
end_height, $hashes$, $local\_peer\_id\_r$,
$remote\_peer\_id\_r$, $local\_peer\_id\_i$,
$remote\_peer\_id\_i$, $local\_peer\_id$,
$remote\_peer\_id$, $blocks\_data$, $command$,
$local\_peer\_index$, $best\_tip\rangle$

$announce(self) \triangleq SendAddrMsg(self)$

$SendVersionMsg(self) \triangleq \land pc[self] =$ "SendVersionMsg"
$\land channels' = [channels$ EXCEPT $![local\_peer\_id\_a[self]][remote\_peer\_id\_a[self]] =$

]
$\land pc' = [pc$ EXCEPT $![self] = Head(stack[self]).pc]$
$\land local\_peer\_id\_a' = [local\_peer\_id\_a$ EXCEPT $![self] = Head(stack[self]).local\_peer$
$\land remote\_peer\_id\_a' = [remote\_peer\_id\_a$ EXCEPT $![self] = Head(stack[self]).remo$
$\land stack' = [stack$ EXCEPT $![self] = Tail(stack[self])]$
$\land$ 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, $hashes$, $local\_peer\_id\_r$,
$remote\_peer\_id\_r$, $local\_peer\_id\_i$,
$remote\_peer\_id\_i$, $local\_peer\_id$,
$remote\_peer\_id$, $blocks\_data$, $command$,
$local\_peer\_index$, $best\_tip\rangle$

$addr(self) \triangleq SendVersionMsg(self)$

$HandleVersionMsg(self) \triangleq \land pc[self] =$ "HandleVersionMsg"
$\land the\_network' = [the\_network$ EXCEPT $![local\_peer\_id\_v[self]].peer\_set[remote\_$
$\land pc' = [pc$ EXCEPT $![self] =$ "SendVerackMsg"$]$
$\land$ 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, $hashes$, $local\_peer\_id\_r$,
$remote\_peer\_id\_r$, $local\_peer\_id\_i$,
$remote\_peer\_id\_i$, $local\_peer\_id$,
$remote\_peer\_id$, $blocks\_data$, $command$,
$local\_peer\_index$, $best\_tip\rangle$

$SendVerackMsg(self) \triangleq \land pc[self] = \text{“SendVerackMsg”}$
$\land channels' = [channels \text{ EXCEPT } ![local\_peer\_id\_v[self]][remote\_peer\_id\_v[self]] =$

]]

$\land pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]$
$\land local\_peer\_id\_v' = [local\_peer\_id\_v \text{ EXCEPT } ![self] = Head(stack[self]).local\_peer$
$\land remote\_peer\_id\_v' = [remote\_peer\_id\_v \text{ EXCEPT } ![self] = Head(stack[self]).remo$
$\land stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$
$\land \text{UNCHANGED } \langle the\_network, local\_peer\_id\_,$
$remote\_peer\_id\_, local\_peer\_id\_a,$
$remote\_peer\_id\_a, local\_peer\_id\_ve,$
$remote\_peer\_id\_ve, local\_peer\_id\_g,$
$remote\_peer\_id\_g, found\_blocks,$
$hash\_count, block\_header\_hashes,$
$remote\_peer\_blocks, start\_height,$
$end\_height, hashes, local\_peer\_id\_r,$
$remote\_peer\_id\_r, local\_peer\_id\_i,$
$remote\_peer\_id\_i, local\_peer\_id,$
$remote\_peer\_id, blocks\_data, command,$
$local\_peer\_index, best\_tip\rangle$

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

$HandleVerackMsg(self) \triangleq \land pc[self] = \text{“HandleVerackMsg”}$
$\land the\_network' = [the\_network \text{ EXCEPT } ![local\_peer\_id\_ve[self]].peer\_set[remote$
$\land pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]$
$\land local\_peer\_id\_ve' = [local\_peer\_id\_ve \text{ EXCEPT } ![self] = Head(stack[self]).local$
$\land remote\_peer\_id\_ve' = [remote\_peer\_id\_ve \text{ EXCEPT } ![self] = Head(stack[self]).r$
$\land stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$
$\land \text{UNCHANGED } \langle channels, local\_peer\_id\_,$
$remote\_peer\_id\_, local\_peer\_id\_a,$
$remote\_peer\_id\_a, local\_peer\_id\_v,$
$remote\_peer\_id\_v, local\_peer\_id\_g,$
$remote\_peer\_id\_g, found\_blocks,$
$hash\_count, block\_header\_hashes,$

$remote\_peer\_blocks$, $start\_height$,
end_height, $hashes$, $local\_peer\_id\_r$,
$remote\_peer\_id\_r$, $local\_peer\_id\_i$,
$remote\_peer\_id\_i$, $local\_peer\_id$,
$remote\_peer\_id$, $blocks\_data$, $command$,
$local\_peer\_index$, $best\_tip\rangle$

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

$HandleGetBlocksMsg(self) \triangleq$ $\land\ pc[self] =$ "HandleGetBlocksMsg"
$\land\ hash\_count' = [hash\_count$ EXCEPT $![self] = channels[local\_peer\_id\_g[self]]$[
$\land\ block\_header\_hashes' = [block\_header\_hashes$ EXCEPT $![self] = channels[loc$
$\land\ remote\_peer\_blocks' = [remote\_peer\_blocks$ EXCEPT $![self] = Ops!GetPeerB$
$\land$ IF $hash\_count'[self] = 0$
THEN $\land\ start\_height' = [start\_height$ EXCEPT $![self] = 1]$
ELSE $\land\ start\_height' = [start\_height$ EXCEPT $![self] = Ops!FindBlockB$
$\land\ end\_height' = [end\_height$ EXCEPT $![self] = start\_height'[self] + (MaxGetBl$
$\land\ found\_blocks' = [found\_blocks$ EXCEPT $![self] = Ops!FindBlocks(remote\_pee$
$\land\ pc' = [pc$ EXCEPT $![self] =$ "SendInvMsg"$]$
$\land$ UNCHANGED $\langle the\_network, channels, stack,$
$local\_peer\_id\_, remote\_peer\_id\_,$
$local\_peer\_id\_a, remote\_peer\_id\_a,$
$local\_peer\_id\_v, remote\_peer\_id\_v,$
$local\_peer\_id\_ve,$
$remote\_peer\_id\_ve, local\_peer\_id\_g,$
$remote\_peer\_id\_g, hashes,$
$local\_peer\_id\_r, remote\_peer\_id\_r,$
$local\_peer\_id\_i, remote\_peer\_id\_i,$
$local\_peer\_id, remote\_peer\_id,$
$blocks\_data, command,$
$local\_peer\_index, best\_tip\rangle$

$SendInvMsg(self) \triangleq$ $\land\ pc[self] =$ "SendInvMsg"
$\land\ channels' = [channels$ EXCEPT $![local\_peer\_id\_g[self]][remote\_peer\_id\_g[self]] =$
h
p
]
]]
$\land\ pc' = [pc$ EXCEPT $![self] = Head(stack[self]).pc]$

$\land\ found\_blocks' = [found\_blocks \text{ EXCEPT } ![self] = Head(stack[self]).found\_blocks]$
$\land\ hash\_count' = [hash\_count \text{ EXCEPT } ![self] = Head(stack[self]).hash\_count]$
$\land\ block\_header\_hashes' = [block\_header\_hashes \text{ EXCEPT } ![self] = Head(stack[self]).bloc$
$\land\ remote\_peer\_blocks' = [remote\_peer\_blocks \text{ EXCEPT } ![self] = Head(stack[self]).remot$
$\land\ start\_height' = [start\_height \text{ EXCEPT } ![self] = Head(stack[self]).start\_height]$
$\land\ end\_height' = [end\_height \text{ EXCEPT } ![self] = Head(stack[self]).end\_height]$
$\land\ local\_peer\_id\_g' = [local\_peer\_id\_g \text{ EXCEPT } ![self] = Head(stack[self]).local\_peer\_id\_$
$\land\ remote\_peer\_id\_g' = [remote\_peer\_id\_g \text{ EXCEPT } ![self] = Head(stack[self]).remote\_p$
$\land\ stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$
$\land\ \text{UNCHANGED } \langle the\_network,\ local\_peer\_id\_,$
$remote\_peer\_id\_,\ local\_peer\_id\_a,$
$remote\_peer\_id\_a,\ local\_peer\_id\_v,$
$remote\_peer\_id\_v,\ local\_peer\_id\_ve,$
$remote\_peer\_id\_ve,\ hashes,\ local\_peer\_id\_r,$
$remote\_peer\_id\_r,\ local\_peer\_id\_i,$
$remote\_peer\_id\_i,\ local\_peer\_id,$
$remote\_peer\_id,\ blocks\_data,\ command,$
$local\_peer\_index,\ best\_tip\rangle$

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

$SendGetBlocksMsg(self) \triangleq \land\ pc[self] = \text{“SendGetBlocksMsg”}$
$\land\ channels' = [channels \text{ EXCEPT } ![local\_peer\_id\_r[self]][remote\_peer\_id\_r[self]]$

$\land\ pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]$
$\land\ hashes' = [hashes \text{ EXCEPT } ![self] = Head(stack[self]).hashes]$
$\land\ local\_peer\_id\_r' = [local\_peer\_id\_r \text{ EXCEPT } ![self] = Head(stack[self]).local\_pe$
$\land\ remote\_peer\_id\_r' = [remote\_peer\_id\_r \text{ EXCEPT } ![self] = Head(stack[self]).re$
$\land\ stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$
$\land\ \text{UNCHANGED } \langle the\_network,\ local\_peer\_id\_,$
$remote\_peer\_id\_,\ local\_peer\_id\_a,$
$remote\_peer\_id\_a,\ local\_peer\_id\_v,$
$remote\_peer\_id\_v,\ local\_peer\_id\_ve,$
$remote\_peer\_id\_ve,\ local\_peer\_id\_g,$
$remote\_peer\_id\_g,\ found\_blocks,$
$hash\_count,\ block\_header\_hashes,$
$remote\_peer\_blocks,\ start\_height,$
$end\_height,\ local\_peer\_id\_i,$
$remote\_peer\_id\_i,\ local\_peer\_id,$
$remote\_peer\_id,\ blocks\_data,\ command,$
$local\_peer\_index,\ best\_tip\rangle$

$request\_blocks(self) \triangleq SendGetBlocksMsg(self)$

$SendGetDataMsg(self) \triangleq \land pc[self] = \text{“SendGetDataMsg”}$
$\land channels' = [channels \text{ EXCEPT } ![local\_peer\_id\_i[self]][remote\_peer\_id\_i[self]] =$

$]$

$\land pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]$
$\land local\_peer\_id\_i' = [local\_peer\_id\_i \text{ EXCEPT } ![self] = Head(stack[self]).local\_peer$
$\land remote\_peer\_id\_i' = [remote\_peer\_id\_i \text{ EXCEPT } ![self] = Head(stack[self]).remo$
$\land stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$
$\land \text{UNCHANGED } \langle the\_network, local\_peer\_id\_,$
$remote\_peer\_id\_, local\_peer\_id\_a,$
$remote\_peer\_id\_a, local\_peer\_id\_v,$
$remote\_peer\_id\_v, local\_peer\_id\_ve,$
$remote\_peer\_id\_ve, local\_peer\_id\_g,$
$remote\_peer\_id\_g, found\_blocks,$
$hash\_count, block\_header\_hashes,$
$remote\_peer\_blocks, start\_height,$
$end\_height, hashes, local\_peer\_id\_r,$
$remote\_peer\_id\_r, local\_peer\_id,$
$remote\_peer\_id, blocks\_data, command,$
$local\_peer\_index, best\_tip\rangle$

$inv(self) \triangleq SendGetDataMsg(self)$

$HandleGetDataMsg(self) \triangleq \land pc[self] = \text{“HandleGetDataMsg”}$
$\land blocks\_data' = [blocks\_data \text{ EXCEPT } ![self] = [item \in 1 \,..\, Len($
$Ops!FindBlockByHash($
$Ops!GetPeerBlocks(the\_$
$channels[local\_peer\_id[s$
$)$
$]]$
$\land pc' = [pc \text{ EXCEPT } ![self] = \text{“IncorporateBlocks”}]$
$\land \text{UNCHANGED } \langle the\_network, channels, stack,$
$local\_peer\_id\_, remote\_peer\_id\_,$
$local\_peer\_id\_a, remote\_peer\_id\_a,$
$local\_peer\_id\_v, remote\_peer\_id\_v,$
$local\_peer\_id\_ve, remote\_peer\_id\_ve,$
$local\_peer\_id\_g, remote\_peer\_id\_g,$
$found\_blocks, hash\_count,$
$block\_header\_hashes,$
$remote\_peer\_blocks, start\_height,$
$end\_height, hashes, local\_peer\_id\_r,$
$remote\_peer\_id\_r, local\_peer\_id\_i,$
$remote\_peer\_id\_i, local\_peer\_id,$

$remote\_peer\_id,\ command,$
$local\_peer\_index,\ best\_tip\rangle$

$IncorporateBlocks(self) \triangleq \land\ pc[self] = \text{“IncorporateBlocks”}$
$\land\ the\_network' = [the\_network \text{ EXCEPT } ![local\_peer\_id[self]].blocks = the\_networ$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]$
$\land\ blocks\_data' = [blocks\_data \text{ EXCEPT } ![self] = Head(stack[self]).blocks\_data]$
$\land\ local\_peer\_id' = [local\_peer\_id \text{ EXCEPT } ![self] = Head(stack[self]).local\_peer\_id$
$\land\ remote\_peer\_id' = [remote\_peer\_id \text{ EXCEPT } ![self] = Head(stack[self]).remote\_$
$\land\ stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$
$\land \text{UNCHANGED } \langle channels,\ local\_peer\_id\_,$
$remote\_peer\_id\_,\ local\_peer\_id\_a,$
$remote\_peer\_id\_a,\ local\_peer\_id\_v,$
$remote\_peer\_id\_v,\ local\_peer\_id\_ve,$
$remote\_peer\_id\_ve,\ local\_peer\_id\_g,$
$remote\_peer\_id\_g,\ found\_blocks,$
$hash\_count,\ block\_header\_hashes,$
$remote\_peer\_blocks,\ start\_height,$
$end\_height,\ hashes,\ local\_peer\_id\_r,$
$remote\_peer\_id\_r,\ local\_peer\_id\_i,$
$remote\_peer\_id\_i,\ command,$
$local\_peer\_index,\ best\_tip\rangle$

$getdata(self) \triangleq HandleGetDataMsg(self) \lor IncorporateBlocks(self)$

$Listening(self) \triangleq \land\ pc[self] = \text{“Listening”}$
$\land\ Len(the\_network) \geq 2$
$\land \exists\, remote\_peer\_index \in 1\,..\,Len(the\_network[self].peer\_set) :$
$\text{IF } channels[self][remote\_peer\_index].header = defaultInitValue$
$\text{THEN } \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“Listening”}]$
$\text{ELSE } \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“Requests”}]$
$\land \text{UNCHANGED } \langle the\_network,\ channels,\ stack,$
$local\_peer\_id\_,\ remote\_peer\_id\_,$
$local\_peer\_id\_a,\ remote\_peer\_id\_a,$
$local\_peer\_id\_v,\ remote\_peer\_id\_v,$
$local\_peer\_id\_ve,\ remote\_peer\_id\_ve,$
$local\_peer\_id\_g,\ remote\_peer\_id\_g,$
$found\_blocks,\ hash\_count,$
$block\_header\_hashes,\ remote\_peer\_blocks,$
$start\_height,\ end\_height,\ hashes,$
$local\_peer\_id\_r,\ remote\_peer\_id\_r,$
$local\_peer\_id\_i,\ remote\_peer\_id\_i,$
$local\_peer\_id,\ remote\_peer\_id,\ blocks\_data,$
$command,\ local\_peer\_index,\ best\_tip\rangle$

$Requests(self) \triangleq \land\ pc[self] = \text{“Requests”}$

$\land \exists\, remote\_peer\_index \in 1 \,..\, Len(the\_network[self].peer\_set) :$
$\land\ channels[self][remote\_peer\_index].header \neq defaultInitValue$
$\land\ command' = [command \text{ EXCEPT } ![self] = channels[self][remote\_peer\_index].header.$
$\land$ IF $command'[self] =$ "addr"
THEN $\land\ \land\ local\_peer\_id\_a' = [local\_peer\_id\_a \text{ EXCEPT } ![self] = self]$
$\land\ remote\_peer\_id\_a' = [remote\_peer\_id\_a \text{ EXCEPT } ![self] = remote\_pee$
$\land\ stack' = [stack \text{ EXCEPT } ![self] = \langle [procedure \mapsto$ "addr",
$pc \mapsto$ "Listening",
$local\_peer\_id\_a \mapsto local\_peer\_id\_$
$remote\_peer\_id\_a \mapsto remote\_peer$
$\circ\ stack[self]]$
$\land\ pc' = [pc \text{ EXCEPT } ![self] =$ "SendVersionMsg"]
$\land$ UNCHANGED $\langle local\_peer\_id\_v,$
$remote\_peer\_id\_v,$
$local\_peer\_id\_ve,$
$remote\_peer\_id\_ve,$
$local\_peer\_id\_g,$
$remote\_peer\_id\_g,$
$found\_blocks,\ hash\_count,$
$block\_header\_hashes,$
$remote\_peer\_blocks,$
$start\_height,\ end\_height,$
$local\_peer\_id\_i,$
$remote\_peer\_id\_i,$
$local\_peer\_id,$
$remote\_peer\_id,\ blocks\_data\rangle$
ELSE $\land$ IF $command'[self] =$ "version"
THEN $\land\ \land\ local\_peer\_id\_v' = [local\_peer\_id\_v \text{ EXCEPT } ![self] = self$
$\land\ remote\_peer\_id\_v' = [remote\_peer\_id\_v \text{ EXCEPT } ![self] =$
$\land\ stack' = [stack \text{ EXCEPT } ![self] = \langle [procedure \mapsto$ "version"
$pc \mapsto$ "Listeni
$local\_peer\_id\_v \mapsto lo$
$remote\_peer\_id\_v \mapsto$
$\circ\ stack[self]]$
$\land\ pc' = [pc \text{ EXCEPT } ![self] =$ "HandleVersionMsg"]
$\land$ UNCHANGED $\langle local\_peer\_id\_ve,$
$remote\_peer\_id\_ve,$
$local\_peer\_id\_g,$
$remote\_peer\_id\_g,$
$found\_blocks,$
$hash\_count,$
$block\_header\_hashes,$
$remote\_peer\_blocks,$
$start\_height,$
$end\_height,$

$local\_peer\_id\_i$,
$remote\_peer\_id\_i$,
$local\_peer\_id$,
$remote\_peer\_id$,
$blocks\_data\rangle$
ELSE $\land$ IF $command'[self] =$ "verack"
THEN $\land$ $\land$ $local\_peer\_id\_ve' = [local\_peer\_id\_ve$ EXCEP
$\land$ $remote\_peer\_id\_ve' = [remote\_peer\_id\_ve$ EX
$\land$ $stack' = [stack$ EXCEPT $![self] = \langle [procedure$
pc
$local\_peer$
$remote\_p$
$\circ\ stack[se$
$\land$ $pc' = [pc$ EXCEPT $![self] =$ "HandleVerackMsg"
$\land$ UNCHANGED $\langle local\_peer\_id\_g$,
$remote\_peer\_id\_g$,
$found\_blocks$,
$hash\_count$,
$block\_header\_hashes$,
$remote\_peer\_blocks$,
$start\_height$,
end_height,
$local\_peer\_id\_i$,
$remote\_peer\_id\_i$,
$local\_peer\_id$,
$remote\_peer\_id$,
$blocks\_data\rangle$
ELSE $\land$ IF $command'[self] =$ "getblocks"
THEN $\land$ $\land$ $local\_peer\_id\_g' = [local\_peer\_id$
$\land$ $remote\_peer\_id\_g' = [remote\_pe$
$\land$ $stack' = [stack$ EXCEPT $![self] =$

$\land$ $found\_blocks' = [found\_blocks$ EXC
$\land$ $hash\_count' = [hash\_count$ EXCEP
$\land$ $block\_header\_hashes' = [block\_hea$
$\land$ $remote\_peer\_blocks' = [remote\_pee$

$\land\ start\_height' = [start\_height$ EXCE
$\land\ end\_height' = [end\_height$ EXCEPT
$\land\ pc' = [pc$ EXCEPT $![self] =$ "Handl
$\land$ UNCHANGED $\langle local\_peer\_id\_i,$
$remote\_peer\_id\_i,$
$local\_peer\_id,$
$remote\_peer\_id,$
$blocks\_data\rangle$
ELSE $\land$ IF $command'[self] =$ "inv"
THEN $\land\ \land\ local\_peer\_id\_i' = [$
$\land\ remote\_peer\_id\_i' =$
$\land\ stack' = [stack$ EXC

$\land\ pc' = [pc$ EXCEPT $![se$
$\land$ UNCHANGED $\langle local\_pe$
$remote\_$
$blocks\_d$
ELSE $\land$ IF $command'[self] =$ '
THEN $\land\ \land\ local\_p$
$\land\ remote$
$\land\ stack' =$

$\land\ blocks\_da$
$\land\ pc' = [pc$
ELSE $\land\ pc' = [pc$
$\land$ UNCHANG

$\land$ UNCHANGED $\langle local\_pe$
$remote\_$
$\land$ UNCHANGED $\langle local\_peer\_id\_g,$
$remote\_peer\_id\_g,$
$found\_blocks,$
$hash\_count,$
$block\_header\_hashes$
$remote\_peer\_blocks,$
$start\_height,$

$end\_height\rangle$
$\land$ UNCHANGED $\langle local\_peer\_id\_ve,$
$remote\_peer\_id\_ve\rangle$
$\land$ UNCHANGED $\langle local\_peer\_id\_v,$
$remote\_peer\_id\_v\rangle$
$\land$ UNCHANGED $\langle local\_peer\_id\_a,$
$remote\_peer\_id\_a\rangle$
$\land$ UNCHANGED $\langle the\_network, channels, local\_peer\_id\_,$
$remote\_peer\_id\_, hashes, local\_peer\_id\_r,$
$remote\_peer\_id\_r, local\_peer\_index, best\_tip\rangle$

$ListenerLoop(self) \triangleq$ $\land pc[self] =$ "ListenerLoop"
$\land \exists\, remote\_peer\_index \in 1 \,..\, Len(the\_network[self].peer\_set) :$
$\land channels' = [channels$ EXCEPT $![self][remote\_peer\_index] = [header \mapsto defaultIn$
$\land pc' = [pc$ EXCEPT $![self] =$ "Listening"]
$\land$ UNCHANGED $\langle the\_network, stack, local\_peer\_id\_,$
$remote\_peer\_id\_, local\_peer\_id\_a,$
$remote\_peer\_id\_a, local\_peer\_id\_v,$
$remote\_peer\_id\_v, local\_peer\_id\_ve,$
$remote\_peer\_id\_ve, local\_peer\_id\_g,$
$remote\_peer\_id\_g, found\_blocks,$
$hash\_count, block\_header\_hashes,$
$remote\_peer\_blocks, start\_height,$
$end\_height, hashes, local\_peer\_id\_r,$
$remote\_peer\_id\_r, local\_peer\_id\_i,$
$remote\_peer\_id\_i, local\_peer\_id,$
$remote\_peer\_id, blocks\_data, command,$
$local\_peer\_index, best\_tip\rangle$

$LISTENER(self) \triangleq Listening(self) \lor Requests(self) \lor ListenerLoop(self)$

$Announce(self) \triangleq$ $\land pc[self] =$ "Announce"
$\land Assert(Len(the\_network) \geq 2,$
"Failure of assertion at line 335, column 9.")
$\land Len(the\_network[local\_peer\_index[self]].peer\_set) > 0$
$\land \exists\, remote\_peer\_index \in 1 \,..\, Len(the\_network[local\_peer\_index[self]].peer\_set) :$
$\land \land local\_peer\_id\_' = [local\_peer\_id\_$ EXCEPT $![self] = local\_peer\_index[self]]$
$\land remote\_peer\_id\_' = [remote\_peer\_id\_$ EXCEPT $![self] = remote\_peer\_index]$
$\land stack' = [stack$ EXCEPT $![self] = \langle [procedure \mapsto$ "announce",
$pc \mapsto$ "RequestInventory",
$local\_peer\_id\_ \mapsto local\_peer\_id\_[self],$
$remote\_peer\_id\_ \mapsto remote\_peer\_id\_[self]]\rangle$
$\circ\, stack[self]]$
$\land pc' = [pc$ EXCEPT $![self] =$ "SendAddrMsg"]
$\land$ UNCHANGED $\langle the\_network, channels, local\_peer\_id\_a,$
$remote\_peer\_id\_a, local\_peer\_id\_v,$

$remote\_peer\_id\_v$, $local\_peer\_id\_ve$,
$remote\_peer\_id\_ve$, $local\_peer\_id\_g$,
$remote\_peer\_id\_g$, $found\_blocks$, $hash\_count$,
$block\_header\_hashes$, $remote\_peer\_blocks$,
$start\_height$, end_height, $hashes$,
$local\_peer\_id\_r$, $remote\_peer\_id\_r$,
$local\_peer\_id\_i$, $remote\_peer\_id\_i$,
$local\_peer\_id$, $remote\_peer\_id$, $blocks\_data$,
$command$, $local\_peer\_index$, $best\_tip\rangle$

$RequestInventory(self) \triangleq$ $\land\ pc[self] = \text{“RequestInventory”}$
$\land\ \exists\, remote\_peer\_index \in 1\,..\,Len(the\_network[local\_peer\_index[self]].peer\_set) :$
$\land\ the\_network[local\_peer\_index[self]].peer\_set[remote\_peer\_index].established$
$\land$ IF $the\_network[local\_peer\_index[self]].peer\_set[remote\_peer\_index].tip > bes$
THEN $\land\ best\_tip' = [best\_tip$ EXCEPT $![self] = the\_network[local\_peer\_i$
ELSE $\land$ TRUE
$\land$ UNCHANGED $best\_tip$
$\land\quad channels[local\_peer\_index[self]][remote\_peer\_index].header = defaultInitV$
$\land\ channels[local\_peer\_index[self]][remote\_peer\_index].payload = defaultInit$
$\land$ IF $Ops!GetPeerTipByIndex(local\_peer\_index[self]).height <$
$the\_network[local\_peer\_index[self]].peer\_set[remote\_peer\_index].tip$
THEN $\land$ IF $Ops!GetPeerTipByIndex(local\_peer\_index[self]).height = 0$
THEN $\land\ \land\ hashes' = [hashes$ EXCEPT $![self] = \langle\rangle]$
$\land\ local\_peer\_id\_r' = [local\_peer\_id\_r$ EXCEPT $![se$
$\land\ remote\_peer\_id\_r' = [remote\_peer\_id\_r$ EXCEPT
$\land\ stack' = [stack$ EXCEPT $![self] = \langle[procedure \mapsto$
$pc \mapsto$
$hashes \mapsto$
$local\_peer\_id$
$remote\_peer\_$
$\circ\ stack[self]]$
$\land\ pc' = [pc$ EXCEPT $![self] = \text{“SendGetBlocksMsg”}]$
ELSE $\land\ \land\ hashes' = [hashes$ EXCEPT $![self] = \langle Ops!GetPe$
$\land\ local\_peer\_id\_r' = [local\_peer\_id\_r$ EXCEPT $![se$
$\land\ remote\_peer\_id\_r' = [remote\_peer\_id\_r$ EXCEPT
$\land\ stack' = [stack$ EXCEPT $![self] = \langle[procedure \mapsto$
$pc \mapsto$
$hashes \mapsto$
$local\_peer\_id$
$remote\_peer\_$
$\circ\ stack[self]]$
$\land\ pc' = [pc$ EXCEPT $![self] = \text{“SendGetBlocksMsg”}]$
ELSE $\land\ pc' = [pc$ EXCEPT $![self] = \text{“CheckSync”}]$
$\land$ UNCHANGED $\langle stack, hashes,$
$local\_peer\_id\_r,$

$remote\_peer\_id\_r\rangle$

$\land$ UNCHANGED $\langle the\_network,\ channels,$
$local\_peer\_id\_,\ remote\_peer\_id\_,$
$local\_peer\_id\_a,\ remote\_peer\_id\_a,$
$local\_peer\_id\_v,\ remote\_peer\_id\_v,$
$local\_peer\_id\_ve,\ remote\_peer\_id\_ve,$
$local\_peer\_id\_g,\ remote\_peer\_id\_g,$
$found\_blocks,\ hash\_count,$
$block\_header\_hashes,$
$remote\_peer\_blocks,\ start\_height,$
$end\_height,\ local\_peer\_id\_i,$
$remote\_peer\_id\_i,\ local\_peer\_id,$
$remote\_peer\_id,\ blocks\_data,\ command,$
$local\_peer\_index\rangle$

$CheckSync(self) \triangleq \land pc[self] = \text{"CheckSync"}$
$\land\ Ops!GetPeerTipByIndex(local\_peer\_index[self]).height > 0$
$\land$ IF $Ops!GetPeerTipByIndex(local\_peer\_index[self]).height < best\_tip[self]$
THEN $\land\ pc' = [pc$ EXCEPT $![self] = \text{"RequestInventory"}]$
ELSE $\land \exists\, remote\_peer\_index \in 1\,..\,Len(the\_network[local\_peer\_index[self]].peer\_s$
$the\_network[local\_peer\_index[self]].peer\_set[remote\_peer\_index].establis$
$\land\ channels[local\_peer\_index[self]][remote\_peer\_index].header = defaultI$
$\land\ channels[local\_peer\_index[self]][remote\_peer\_index].payload = default$
$\land\ pc' = [pc$ EXCEPT $![self] = \text{"Done"}]$
$\land$ UNCHANGED $\langle the\_network,\ channels,\ stack,$
$local\_peer\_id\_,\ remote\_peer\_id\_,$
$local\_peer\_id\_a,\ remote\_peer\_id\_a,$
$local\_peer\_id\_v,\ remote\_peer\_id\_v,$
$local\_peer\_id\_ve,\ remote\_peer\_id\_ve,$
$local\_peer\_id\_g,\ remote\_peer\_id\_g,$
$found\_blocks,\ hash\_count,$
$block\_header\_hashes,\ remote\_peer\_blocks,$
$start\_height,\ end\_height,\ hashes,$
$local\_peer\_id\_r,\ remote\_peer\_id\_r,$
$local\_peer\_id\_i,\ remote\_peer\_id\_i,$
$local\_peer\_id,\ remote\_peer\_id,\ blocks\_data,$
$command,\ local\_peer\_index,\ best\_tip\rangle$

$SYNCHRONIZER(self) \triangleq Announce(self) \lor RequestInventory(self)$
$\lor\ CheckSync(self)$

Allow infinite stuttering to prevent deadlock on termination.
$Terminating \triangleq \land \forall\, self \in ProcSet : pc[self] = \text{"Done"}$
$\land$ UNCHANGED $vars$

$Next \triangleq (\exists\, self \in ProcSet :\ \lor announce(self) \lor addr(self)$

$$
\begin{aligned}
&\lor\ version(self) \lor verack(self) \\
&\lor\ getblocks(self) \lor request\_blocks(self) \\
&\lor\ inv(self) \lor getdata(self)) \\
&\lor\ (\exists\, self \in 1\,..\,Len(RunningBlockchain) : LISTENER(self)) \\
&\lor\ (\exists\, self \in PeerProcessDiffId + 1\,..\,PeerProcessDiffId + Len(RunningBlockchain) : SYNCHRONI \\
&\lor\ Terminating
\end{aligned}
$$

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

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

END TRANSLATION