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 $Operators$

**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**
$SendAddrMsg$:
$\quad channels[local\_peer\_id][remote\_peer\_id] := [$
$\quad\quad header \mapsto [command\_name \mapsto$ "addr"$],$
$\quad\quad payload \mapsto [$
$\quad\quad\quad address\_count \mapsto 1,$
$\quad\quad\quad\quad$ *Only a single address is supported.*
$\quad\quad\quad addresses \mapsto the\_network[local\_peer\_id].peer$
$\quad\quad ]$
$\quad ]$ **;**

```
    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!GetPeerTip(the_network[local_peer_id].peer_set[remote_peer_id].address)]
        ];
    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
    Incorporate:
        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
            )
        ] ;
        the_network[local_peer_id].blocks := the_network[local_peer_id].blocks ∪ ToSet(blocks_data) ;
    UpdateTip:
        the_network[local_peer_id].chain_tip := [
            height ↦ blocks_data[Len(blocks_data)].height,
            hash ↦ blocks_data[Len(blocks_data)].hash
        ] ;
    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 the_network[local_peer_index].chain_tip.height <
                the_network[local_peer_index].peer_set[remote_peer_index].tip then
                Request blocks.
                if the_network[local_peer_index].chain_tip.height = 0 then
                    call request_blocks(⟨⟩, local_peer_index, remote_peer_index) ;
                else
                    call request_blocks(
                        ⟨the_network[local_peer_index].chain_tip.hash⟩,
                        local_peer_index,
                        remote_peer_index
                    ) ;
                end if ;
            end if ;
        end with ;
    CheckSync:
        await the_network[local_peer_index].chain_tip.height > 0 ;
        if the_network[local_peer_index].chain_tip.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 ;
            print "Peer is in sync!" ;
        end if ;
end process ;

end algorithm ;
```

*BEGIN TRANSLATION* (*chksum*(*pcal*) = "6acb42eb" ∧ *chksum*(*tla*) = "4e4ceef9")
*Parameter local_peer_id of procedure announce at line* 40 *col* 20 *changed to local_peer_id_*
*Parameter remote_peer_id of procedure announce at line* 40 *col* 35 *changed to remote_peer_id_*

*Parameter local_peer_id of procedure addr at line 55 col 16 changed to local_peer_id_a*
*Parameter remote_peer_id of procedure addr at line 55 col 31 changed to remote_peer_id_a*
*Parameter local_peer_id of procedure version at line 70 col 19 changed to local_peer_id_v*
*Parameter remote_peer_id of procedure version at line 70 col 34 changed to remote_peer_id_v*
*Parameter local_peer_id of procedure verack at line 84 col 18 changed to local_peer_id_ve*
*Parameter remote_peer_id of procedure verack at line 84 col 33 changed to remote_peer_id_ve*
*Parameter local_peer_id of procedure getblocks at line 92 col 21 changed to local_peer_id_g*
*Parameter remote_peer_id of procedure getblocks at line 92 col 36 changed to remote_peer_id_g*
*Parameter local_peer_id of procedure request_blocks at line 134 col 34 changed to local_peer_id_r*
*Parameter remote_peer_id of procedure request_blocks at line 134 col 49 changed to remote_peer_id_r*
*Parameter local_peer_id of procedure inv at line 147 col 15 changed to local_peer_id_i*
*Parameter remote_peer_id of procedure inv at line 147 col 30 changed to remote_peer_id_i*

CONSTANT *defaultInitValue*

VARIABLES *the_network*, *channels*, *pc*, *stack*

*define statement*

LOCAL *Ops* $\triangleq$ INSTANCE *Operators*

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*$\rangle$

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

$Init \triangleq$ *Global variables*
$\wedge$ *the_network* $=$ *RunningBlockchain*
$\wedge$ *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*
$\land\ local\_peer\_index = [self \in PeerProcessDiffId + 1 \,..\, PeerProcessDiffId + Len(RunningBlockchai$
$\land\ best\_tip = [self \in PeerProcessDiffId + 1 \,..\, PeerProcessDiffId + Len(RunningBlockchain) \mapsto 0]$
$\land\ stack = [self \in ProcSet \mapsto \langle\rangle]$
$\land\ pc = [self \in ProcSet \mapsto \text{CASE } self \in 1 \,..\, Len(RunningBlockchain) \to \text{“Listening”}$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \square \quad self \in PeerProcessDiffId + 1 \,..\, PeerProcessDiffId + Len(Running$

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

]]

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

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

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

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

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

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

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

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

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

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

$HandleVerackMsg(self) \triangleq$ $\land pc[self] =$ "HandleVerackMsg"
$\land the\_network' = [the\_network$ EXCEPT $![local\_peer\_id\_ve[self]].peer\_set[rem$
$\land pc' = [pc$ EXCEPT $![self] = Head(stack[self]).pc]$
$\land local\_peer\_id\_ve' = [local\_peer\_id\_ve$ EXCEPT $![self] = Head(stack[self]).lo$
$\land remote\_peer\_id\_ve' = [remote\_peer\_id\_ve$ EXCEPT $![self] = Head(stack[self$
$\land stack' = [stack$ EXCEPT $![self] = Tail(stack[self])]$
$\land$ 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 $![self] = channels[local\_peer\_id\_g[se$
$\land block\_header\_hashes' = [block\_header\_hashes$ EXCEPT $![self] = channels$
$\land remote\_peer\_blocks' = [remote\_peer\_blocks$ EXCEPT $![self] = Ops!GetPe$
$\land$ IF $hash\_count'[self] = 0$
THEN $\land start\_height' = [start\_height$ EXCEPT $![self] = 1]$
ELSE $\land start\_height' = [start\_height$ EXCEPT $![self] = Ops!FindBlo$
$\land end\_height' = [end\_height$ EXCEPT $![self] = start\_height'[self] + (MaxG$
$\land found\_blocks' = [found\_blocks$ EXCEPT $![self] = Ops!FindBlocks(remote$
$\land pc' = [pc$ 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 $![local\_peer\_id\_g[self]][remote\_peer\_id\_g[self]] =$

]]

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

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

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

$\land\ pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]$
$\land\ hashes' = [hashes \text{ EXCEPT } ![self] = Head(stack[self]).hashes]$
$\land\ local\_peer\_id\_r' = [local\_peer\_id\_r \text{ EXCEPT } ![self] = Head(stack[self]).loca$
$\land\ remote\_peer\_id\_r' = [remote\_peer\_id\_r \text{ EXCEPT } ![self] = Head(stack[self]$
$\land\ stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$

$\land$ 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*] = "SendGetDataMsg"
$\land$ *channels*′ = [*channels* EXCEPT ![*local_peer_id_i*[*self*]][*remote_peer_id_i*[*self*

$\land$ *pc*′ = [*pc* EXCEPT ![*self*] = *Head*(*stack*[*self*]).*pc*]
$\land$ *local_peer_id_i*′ = [*local_peer_id_i* EXCEPT ![*self*] = *Head*(*stack*[*self*]).*local_*
$\land$ *remote_peer_id_i*′ = [*remote_peer_id_i* EXCEPT ![*self*] = *Head*(*stack*[*self*]).*r*
$\land$ *stack*′ = [*stack* EXCEPT ![*self*] = *Tail*(*stack*[*self*])]
$\land$ 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*)

*Incorporate*(*self*) $\triangleq$ $\land$ *pc*[*self*] = "Incorporate"
$\land$ *blocks_data*′ = [*blocks_data* EXCEPT ![*self*] = [*item* $\in$ 1 .. *Len*(*chan*
*Ops*!*FindBlockByHash*(
*Ops*!*GetPeerBlocks*(*the_net*
*channels*[*local_peer_id*[*self*]]
)
]]

$\land$ $the\_network' = [the\_network$ EXCEPT $![local\_peer\_id[self]].blocks = the\_network[l$
$\land$ $pc' = [pc$ EXCEPT $![self] =$ "UpdateTip"$]$
$\land$ UNCHANGED $\langle channels,\ stack,\ local\_peer\_id\_,$
$remote\_peer\_id\_,\ local\_peer\_id\_a,$
$remote\_peer\_id\_a,\ local\_peer\_id\_v,$
$remote\_peer\_id\_v,\ local\_peer\_id\_ve,$
$remote\_peer\_id\_ve,\ local\_peer\_id\_g,$
$remote\_peer\_id\_g,\ found\_blocks,$
$hash\_count,\ block\_header\_hashes,$
$remote\_peer\_blocks,\ start\_height,$
$end\_height,\ hashes,\ local\_peer\_id\_r,$
$remote\_peer\_id\_r,\ local\_peer\_id\_i,$
$remote\_peer\_id\_i,\ local\_peer\_id,$
$remote\_peer\_id,\ command,\ local\_peer\_index,$
$best\_tip\rangle$

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

$getdata(self) \triangleq Incorporate(self) \lor UpdateTip(self)$

$Listening(self) \triangleq$ $\land$ $pc[self] =$ "Listening"
$\land$ $Len(the\_network) \geq 2$
$\land$ $\exists\, remote\_peer\_index \in 1\,..\,Len(the\_network[self].peer\_set)$ :
IF $channels[self][remote\_peer\_index].header = defaultInitValue$
THEN $\land$ $pc' = [pc$ EXCEPT $![self] =$ "Listening"$]$
ELSE $\land$ $pc' = [pc$ EXCEPT $![self] =$ "Requests"$]$

$\land$ 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*$\rangle$

*Requests*(*self*) $\triangleq$ $\land$ *pc*[*self*] = "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* EXCEPT ![*self*] = *channels*[*self*][*remote_peer_index*].*hea*
$\land$ IF *command*$'$[*self*] = "addr"
THEN $\land$ $\land$ *local_peer_id_a*$'$ = [*local_peer_id_a* EXCEPT ![*self*] = *self*]
$\land$ *remote_peer_id_a*$'$ = [*remote_peer_id_a* EXCEPT ![*self*] = *remote*
$\land$ *stack*$'$ = [*stack* EXCEPT ![*self*] = $\langle$[*procedure* $\mapsto$ "addr",
*pc* $\mapsto$ "Listening",
*local_peer_id_a* $\mapsto$ *local_peer_*
*remote_peer_id_a* $\mapsto$ *remote_*
$\circ$ *stack*[*self*]]
$\land$ *pc*$'$ = [*pc* 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* EXCEPT ![*self*] =
$\land$ *remote_peer_id_v*$'$ = [*remote_peer_id_v* EXCEPT ![*se*
$\land$ *stack*$'$ = [*stack* EXCEPT ![*self*] = $\langle$[*procedure* $\mapsto$ "ver

$pc \mapsto$ "List
$local\_peer\_id\_v \mapsto$
$remote\_peer\_id\_v$
$\circ\ stack[self]]$

$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"HandleVersionMsg"}]$
$\land$ 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 $\land\ \land\ local\_peer\_id\_ve' = [local\_peer\_id\_ve$ EX
$\land\ remote\_peer\_id\_ve' = [remote\_peer\_id\_v$
$\land\ stack' = [stack \text{ EXCEPT } ![self] = \langle [proce$
$pc$
$local\_$
$remot$
$\circ\ stac$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"HandleVerackM}$
$\land$ 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 $\land\ \land\ local\_peer\_id\_g' = [local\_pee$
$\land\ remote\_peer\_id\_g' = [remot$

$\land\ stack' = [stack \text{ EXCEPT } ![se$

$\land\ found\_blocks' = [found\_blocks$
$\land\ hash\_count' = [hash\_count \text{ EX}$
$\land\ block\_header\_hashes' = [block\_$
$\land\ remote\_peer\_blocks' = [remote$
$\land\ start\_height' = [start\_height \text{ E}$
$\land\ end\_height' = [end\_height \text{ EX}$
$\land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“Ha}$
$\land \text{ UNCHANGED } \langle local\_peer\_id\_i,$
$remote\_peer\_id\_$
$local\_peer\_id,$
$remote\_peer\_id,$
$blocks\_data\rangle$
ELSE $\land \text{ IF } command'[self] = \text{“inv”}$
THEN $\land\ \land\ local\_peer\_id\_i'$
$\land\ remote\_peer\_id.$
$\land\ stack' = [stack$

$\land\ pc' = [pc \text{ EXCEPT}$
$\land \text{ UNCHANGED } \langle loca$
$rem$
$bloc$
ELSE $\land \text{ IF } command'[self]$
THEN $\land\ \land\ loc$
$\land\ ren$
$\land\ sta$

$\land\ blocks$

$\wedge\ pc' =$
ELSE $\wedge\ pc' =$
$\wedge$ UNCH

$\wedge$ UNCHANGED $\langle loca$
$rem$
$\wedge$ UNCHANGED $\langle local\_peer\_id\_g,$
$remote\_peer\_id\_$
$found\_blocks,$
$hash\_count,$
$block\_header\_ha$
$remote\_peer\_blo$
$start\_height,$
$end\_height\rangle$
$\wedge$ UNCHANGED $\langle local\_peer\_id\_ve,$
$remote\_peer\_id\_ve\rangle$
$\wedge$ UNCHANGED $\langle local\_peer\_id\_v,$
$remote\_peer\_id\_v\rangle$
$\wedge$ UNCHANGED $\langle local\_peer\_id\_a,$
$remote\_peer\_id\_a\rangle$
$\wedge$ UNCHANGED $\langle the\_network, channels, local\_peer\_id\_,$
$remote\_peer\_id\_, hashes, local\_peer\_id\_r,$
$remote\_peer\_id\_r, local\_peer\_index, best\_tip\rangle$

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

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

$$
\begin{array}{l}
Announce(self) \triangleq \\
\quad \land\ pc[self] = \text{"Announce"} \\
\quad \land\ Assert(Len(the\_network) \geq 2, \\
\qquad\qquad \text{"Failure of assertion at line 224, column 9."}) \\
\quad \land\ Len(the\_network[local\_peer\_index[self]].peer\_set) > 0 \\
\quad \land\ \exists\, remote\_peer\_index \in 1 \,..\, Len(the\_network[local\_peer\_index[self]].peer\_set) : \\
\qquad \land\ \land\ local\_peer\_id\_' = [local\_peer\_id\_ \text{ EXCEPT } ![self] = local\_peer\_index[self]] \\
\qquad\quad \land\ remote\_peer\_id\_' = [remote\_peer\_id\_ \text{ EXCEPT } ![self] = remote\_peer\_index \\
\qquad\quad \land\ stack' = [stack \text{ EXCEPT } ![self] = \langle [procedure \mapsto \text{"announce"}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad pc \mapsto \text{"RequestInventory"}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad local\_peer\_id\_ \mapsto local\_peer\_id\_[self], \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad remote\_peer\_id\_ \mapsto remote\_peer\_id\_[se \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \circ\ stack[self]] \\
\qquad \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{"SendAddrMsg"}] \\
\quad \land \text{ UNCHANGED } \langle the\_network,\ channels,\ local\_peer\_id\_a, \\
\qquad\qquad\qquad\qquad remote\_peer\_id\_a,\ local\_peer\_id\_v, \\
\qquad\qquad\qquad\qquad remote\_peer\_id\_v,\ local\_peer\_id\_ve, \\
\qquad\qquad\qquad\qquad remote\_peer\_id\_ve,\ local\_peer\_id\_g, \\
\qquad\qquad\qquad\qquad remote\_peer\_id\_g,\ found\_blocks,\ hash\_count, \\
\qquad\qquad\qquad\qquad block\_header\_hashes,\ remote\_peer\_blocks, \\
\qquad\qquad\qquad\qquad start\_height,\ end\_height,\ hashes, \\
\qquad\qquad\qquad\qquad local\_peer\_id\_r,\ remote\_peer\_id\_r, \\
\qquad\qquad\qquad\qquad local\_peer\_id\_i,\ remote\_peer\_id\_i, \\
\qquad\qquad\qquad\qquad local\_peer\_id,\ remote\_peer\_id,\ blocks\_data, \\
\qquad\qquad\qquad\qquad command,\ local\_peer\_index,\ best\_tip\rangle
\end{array}
$$

$$
\begin{array}{l}
RequestInventory(self) \triangleq \\
\quad \land\ pc[self] = \text{"RequestInventory"} \\
\quad \land\ \exists\, remote\_peer\_index \in 1 \,..\, Len(the\_network[local\_peer\_index[self]].peer\_set \\
\qquad \land\ the\_network[local\_peer\_index[self]].peer\_set[remote\_peer\_index].establis \\
\qquad \land \text{ IF } the\_network[local\_peer\_index[self]].peer\_set[remote\_peer\_index].tip > \\
\qquad\qquad \text{THEN } \land\ best\_tip' = [best\_tip \text{ EXCEPT } ![self] = the\_network[local\_pe \\
\qquad\qquad \text{ELSE } \land \text{ TRUE} \\
\qquad\qquad\qquad\quad \land \text{ UNCHANGED } best\_tip \\
\qquad \land\ \ channels[local\_peer\_index[self]][remote\_peer\_index].header = defaultI \\
\qquad\quad \land\ channels[local\_peer\_index[self]][remote\_peer\_index].payload = defaul \\
\qquad \land \text{ IF } the\_network[local\_peer\_index[self]].chain\_tip.height < \\
\qquad\qquad the\_network[local\_peer\_index[self]].peer\_set[remote\_peer\_index].tip \\
\qquad\qquad \text{THEN } \land \text{ IF } the\_network[local\_peer\_index[self]].chain\_tip.height = 0 \\
\qquad\qquad\qquad\qquad \text{THEN } \land\ \land\ hashes' = [hashes \text{ EXCEPT } ![self] = \langle\rangle] \\
\qquad\qquad\qquad\qquad\qquad\qquad \land\ local\_peer\_id\_r' = [local\_peer\_id\_r \text{ EXCEPT } \\
\qquad\qquad\qquad\qquad\qquad\qquad \land\ remote\_peer\_id\_r' = [remote\_peer\_id\_r \text{ EXC} \\
\qquad\qquad\qquad\qquad\qquad\qquad \land\ stack' = [stack \text{ EXCEPT } ![self] = \langle [procedure \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad pc \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad hashes \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad local\_peer \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad remote\_p
\end{array}
$$

$\circ\ stack[se$
$\wedge\ pc' = [pc \text{ EXCEPT } ![self] = \text{“SendGetBlocksMsg}$
ELSE $\wedge\ \wedge\ hashes' = [hashes \text{ EXCEPT } ![self] = \langle the\_ne$
$\wedge\ local\_peer\_id\_r' = [local\_peer\_id\_r \text{ EXCEPT }$
$\wedge\ remote\_peer\_id\_r' = [remote\_peer\_id\_r \text{ EXC}$
$\wedge\ stack' = [stack \text{ EXCEPT } ![self] = \langle [procedure$
$pc$
$hashes$
$local\_pee$
$remote\_p$
$\circ\ stack[se$
$\wedge\ pc' = [pc \text{ EXCEPT } ![self] = \text{“SendGetBlocksMsg}$
ELSE $\wedge\ pc' = [pc \text{ EXCEPT } ![self] = \text{“CheckSync”}]$
$\wedge$ UNCHANGED $\langle stack, hashes,$
$local\_peer\_id\_r,$
$remote\_peer\_id\_r\rangle$
$\wedge$ UNCHANGED $\langle the\_network, channels,$
$local\_peer\_id\_, remote\_peer\_id\_,$
$local\_peer\_id\_a, remote\_peer\_id\_a,$
$local\_peer\_id\_v, remote\_peer\_id\_v,$
$local\_peer\_id\_ve, remote\_peer\_id\_ve,$
$local\_peer\_id\_g, remote\_peer\_id\_g,$
$found\_blocks, hash\_count,$
$block\_header\_hashes,$
$remote\_peer\_blocks, start\_height,$
$end\_height, local\_peer\_id\_i,$
$remote\_peer\_id\_i, local\_peer\_id,$
$remote\_peer\_id, blocks\_data, command,$
$local\_peer\_index\rangle$

$CheckSync(self) \triangleq$ $\wedge\ pc[self] = \text{“CheckSync”}$
$\wedge\ the\_network[local\_peer\_index[self]].chain\_tip.height > 0$
$\wedge$ IF $the\_network[local\_peer\_index[self]].chain\_tip.height < best\_tip[self]$
THEN $\wedge\ pc' = [pc \text{ EXCEPT } ![self] = \text{“RequestInventory”}]$
ELSE $\wedge\ \exists\, remote\_peer\_index \in 1\,..\,Len(the\_network[local\_peer\_index[self]].pe$
$the\_network[local\_peer\_index[self]].peer\_set[remote\_peer\_index].est$
$\wedge\ channels[local\_peer\_index[self]][remote\_peer\_index].header = defa$
$\wedge\ channels[local\_peer\_index[self]][remote\_peer\_index].payload = def$
$\wedge\ PrintT(\text{“Peer is in sync!”})$
$\wedge\ pc' = [pc \text{ EXCEPT } ![self] = \text{“Done”}]$
$\wedge$ 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.*

$$Terminating \triangleq \land \forall\, self \in ProcSet : pc[self] = \text{“Done”} \land \text{UNCHANGED } vars$$

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

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

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

*END TRANSLATION*