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, $Sequences$, $Naturals$, $FiniteSets$, $Utils$, $Blockchain$

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

$MaxGetBlocksInvResponse \triangleq 3$

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

$PeerProcessDiffId \triangleq 1000$

Define the network to be used by the algorithm.

$RunningBlockchain \triangleq BLOCKCHAIN5$

--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. Each peer can establish a max of 3 connections.

$channels = [i \in 1 \,..\, Len(the\_network) \mapsto$
$[j \in 1 \,..\, 3 \mapsto [header \mapsto defaultInitValue,\ payload \mapsto defaultInitValue]]]$ **;**

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
$SendAddrMsg$:
$channels[local\_peer\_id][remote\_peer\_id] := [$
$header \mapsto [command\_name \mapsto$ "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 ↦ [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.
        await Len(the_network) ≥ 2;

        The peer set size must be at least 1.
        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) = "89735483" ∧ chksum(tla) = "d42aa36a")
  Parameter local_peer_id of procedure announce at line 33 col 20 changed to local_peer_id_
  Parameter remote_peer_id of procedure announce at line 33 col 35 changed to remote_peer_id_
  Parameter local_peer_id of procedure addr at line 48 col 16 changed to local_peer_id_a
  Parameter remote_peer_id of procedure addr at line 48 col 31 changed to remote_peer_id_a
  Parameter local_peer_id of procedure version at line 63 col 19 changed to local_peer_id_v
  Parameter remote_peer_id of procedure version at line 63 col 34 changed to remote_peer_id_v
  Parameter local_peer_id of procedure verack at line 77 col 18 changed to local_peer_id_ve
  Parameter remote_peer_id of procedure verack at line 77 col 33 changed to remote_peer_id_ve
  Parameter local_peer_id of procedure getblocks at line 85 col 21 changed to local_peer_id_g
```

Parameter remote_peer_id of procedure getblocks at line 85 *col* 36 *changed to remote_peer_id_g*
Parameter local_peer_id of procedure request_blocks at line 127 *col* 34 *changed to local_peer_id_r*
Parameter remote_peer_id of procedure request_blocks at line 127 *col* 49 *changed to remote_peer_id_r*
Parameter local_peer_id of procedure inv at line 140 *col* 15 *changed to local_peer_id_i*
Parameter remote_peer_id of procedure inv at line 140 *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*
$\land\ the\_network = RunningBlockchain$
$\land\ channels = [i \in 1 \,..\, Len(the\_network) \mapsto [j \in 1 \,..\, 3 \mapsto [header \mapsto defaultInitValue,\ payload \mapsto defaultInitValue]]]$
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
$\land\ local\_peer\_id\_ve = [self \in ProcSet \mapsto defaultInitValue]$
$\land\ remote\_peer\_id\_ve = [self \in ProcSet \mapsto defaultInitValue]$

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

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

$]]$
$\wedge\ pc' = [pc \text{ EXCEPT } ![self] = Head(stack[self]).pc]$
$\wedge\ local\_peer\_id\_' = [local\_peer\_id\_ \text{ EXCEPT } ![self] = Head(stack[self]).local\_peer\_$
$\wedge\ remote\_peer\_id\_' = [remote\_peer\_id\_ \text{ EXCEPT } ![self] = Head(stack[self]).remot$
$\wedge\ stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$
$\wedge\ \text{UNCHANGED } \langle the\_network,\ local\_peer\_id\_a,$
$\qquad\qquad\qquad\qquad\qquad\qquad remote\_peer\_id\_a,\ local\_peer\_id\_v,$
$\qquad\qquad\qquad\qquad\qquad\qquad remote\_peer\_id\_v,\ local\_peer\_id\_ve,$

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

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

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

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

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

$hash\_count$, $block\_header\_hashes$,
$remote\_peer\_blocks$, $start\_height$,
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 $![self] = channels[local\_peer\_id\_g[se$
$\land\ block\_header\_hashes' = [block\_header\_hashes$ 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 $![local\_peer\_id\_g[self]][remote\_peer\_id\_g[self]] =$

]]

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

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

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

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

$local\_peer\_index,\ best\_tip\rangle$

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

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

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

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

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

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

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

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

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

Requests(self) ≜ ∧ pc[self] = "Requests"
                 ∧ ∃ remote_peer_index ∈ 1 .. Len(the_network[self].peer_set) :
                     ∧ channels[self][remote_peer_index].header ≠ defaultInitValue
                     ∧ command' = [command EXCEPT ![self] = channels[self][remote_peer_index].hea
                     ∧ IF command'[self] = "addr"
                         THEN ∧ ∧ local_peer_id_a' = [local_peer_id_a EXCEPT ![self] = self]
                                ∧ remote_peer_id_a' = [remote_peer_id_a EXCEPT ![self] = remote
                                ∧ stack' = [stack EXCEPT ![self] = ⟨[procedure ↦ "addr",
                                                                     pc ↦ "Listening",
                                                                     local_peer_id_a ↦ local_peer
                                                                     remote_peer_id_a ↦ remote_
                                                                     ∘ stack[self]]
                              ∧ pc' = [pc EXCEPT ![self] = "SendVersionMsg"]
                              ∧ UNCHANGED ⟨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⟩
                         ELSE ∧ IF command'[self] = "version"
                                  THEN ∧ ∧ local_peer_id_v' = [local_peer_id_v EXCEPT ![self] =
                                         ∧ remote_peer_id_v' = [remote_peer_id_v EXCEPT ![se
                                         ∧ stack' = [stack EXCEPT ![self] = ⟨[procedure ↦ "ver
                                                                              pc ↦ "List
                                                                              local_peer_id_v ↦
                                                                              remote_peer_id_v
                                                                              ∘ stack[self]]
                                       ∧ pc' = [pc EXCEPT ![self] = "HandleVersionMsg"]
                                       ∧ UNCHANGED ⟨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
$\land$ $remote\_peer\_id\_ve' = [remote\_peer\_id\_v$
$\land$ $stack' = [stack$ EXCEPT $![self] = \langle[proce$
pc
$local\_$
$remot$
$\circ$ $stac$
$\land$ $pc' = [pc$ EXCEPT $![self] =$ "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 $\land$ $\land$ $local\_peer\_id\_g' = [local\_pee$
$\land$ $remote\_peer\_id\_g' = [remot$
$\land$ $stack' = [stack$ EXCEPT $![se$

$\land\ found\_blocks' = [found\_blocks$
$\land\ hash\_count' = [hash\_count$ EX
$\land\ block\_header\_hashes' = [block\_$
$\land\ remote\_peer\_blocks' = [remote$
$\land\ start\_height' = [start\_height$ E
$\land\ end\_height' = [end\_height$ EXC
$\land\ pc' = [pc$ EXCEPT $![self] =$ "H
$\land$ UNCHANGED $\langle local\_peer\_id\_i,$
$remote\_peer\_id\_$
$local\_peer\_id,$
$remote\_peer\_id,$
$blocks\_data\rangle$

ELSE $\land$ IF $command'[self] =$ "inv"

THEN $\land\ \land\ local\_peer\_id\_i'$
$\land\ remote\_peer\_id$
$\land\ stack' = [stack$

$\land\ pc' = [pc$ EXCEPT
$\land$ UNCHANGED $\langle loca$
rem
$bloc$

ELSE $\land$ IF $command'[self]$

THEN $\land\ \land\ loc$
$\land\ ren$
$\land\ sta$

$\land\ blocks$
$\land\ pc' =$

ELSE $\land\ pc' =$
$\land$ UNCH

$\land$ UNCHANGED $\langle loca$
rem
$\land$ 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 $![self][remote\_peer\_index] = [header \mapsto defau$
$\wedge\ pc' = [pc$ 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, $hashes$, $local\_peer\_id\_r$,
$remote\_peer\_id\_r$, $local\_peer\_id\_i$,
$remote\_peer\_id\_i$, $local\_peer\_id$,
$remote\_peer\_id$, $blocks\_data$, $command$,
$local\_peer\_index$, $best\_tip\rangle$

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

$Announce(self) \triangleq$ $\wedge\ pc[self] =$ "Announce"
$\wedge\ Len(the\_network) \geq 2$
$\wedge\ Len(the\_network[local\_peer\_index[self]].peer\_set) > 0$
$\wedge\ \exists\, remote\_peer\_index \in 1\,..\,Len(the\_network[local\_peer\_index[self]].peer\_set)$:
$\wedge\ \wedge\ local\_peer\_id\_' = [local\_peer\_id\_$ EXCEPT $![self] = local\_peer\_index[self]]$
$\wedge\ remote\_peer\_id\_' = [remote\_peer\_id\_$ EXCEPT $![self] = remote\_peer\_index$
$\wedge\ stack' = [stack$ EXCEPT $![self] = \langle[procedure \mapsto$ "announce",
$pc \mapsto$ "RequestInventory",
$local\_peer\_id\_ \mapsto local\_peer\_id\_[self]$,
$remote\_peer\_id\_ \mapsto remote\_peer\_id\_[se$
$\circ\ stack[self]]$

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

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

```
                                ELSE  ∧ pc' = [pc EXCEPT ![self] = "CheckSync"]
                                      ∧ UNCHANGED ⟨stack, hashes,
                                                   local_peer_id_r,
                                                   remote_peer_id_r⟩
                           ∧ UNCHANGED ⟨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⟩

CheckSync(self) ≜  ∧ pc[self] = "CheckSync"
                   ∧ the_network[local_peer_index[self]].chain_tip.height > 0
                   ∧ IF the_network[local_peer_index[self]].chain_tip.height < best_tip[self]
                        THEN  ∧ pc' = [pc EXCEPT ![self] = "RequestInventory"]
                        ELSE  ∧ ∃ remote_peer_index ∈ 1 .. Len(the_network[local_peer_index[self]].pe
                                   the_network[local_peer_index[self]].peer_set[remote_peer_index].est
                                   ∧ channels[local_peer_index[self]][remote_peer_index].header = defa
                                   ∧ channels[local_peer_index[self]][remote_peer_index].payload = def
                              ∧ PrintT("Peer is in sync!")
                              ∧ pc' = [pc EXCEPT ![self] = "Done"]
                   ∧ UNCHANGED ⟨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⟩

SYNCHRONIZER(self) ≜  Announce(self) ∨ RequestInventory(self)
                      ∨ CheckSync(self)
```

Allow infinite stuttering to prevent deadlock on termination.

$$
\begin{array}{l}
Terminating \triangleq \ \land \forall\, self \in ProcSet : pc[self] = \text{“Done”} \\
\qquad\qquad\quad \land \text{UNCHANGED } vars \\
\\
Next \triangleq \ (\exists\, self \in ProcSet : \ \lor announce(self) \lor addr(self) \\
\qquad\qquad\qquad\qquad\qquad\quad \lor version(self) \lor verack(self) \\
\qquad\qquad\qquad\qquad\qquad\quad \lor getblocks(self) \lor request\_blocks(self) \\
\qquad\qquad\qquad\qquad\qquad\quad \lor inv(self) \lor getdata(self)) \\
\qquad\quad \lor (\exists\, self \in 1 \,..\, Len(RunningBlockchain) : LISTENER(self)) \\
\qquad\quad \lor (\exists\, self \in PeerProcessDiffId + 1 \,..\, PeerProcessDiffId + Len(RunningBlockchain) : SYNCHRO \\
\qquad\quad \lor Terminating \\
\\
Spec \triangleq \ Init \land \Box[Next]_{vars} \\
\\
Termination \triangleq \ \Diamond(\forall\, self \in ProcSet : pc[self] = \text{“Done”}) \\
\\
\textit{END TRANSLATION}
\end{array}
$$