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

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:
        $channels[local\_peer\_id][remote\_peer\_id] := [$
            $header \mapsto [command\_name \mapsto$ "inv"$]$,
            $payload \mapsto [$
                $count \mapsto Cardinality(found\_blocks)$,
                $inventory \mapsto [$
                    $h \in 1 \,..\, Cardinality(found\_blocks) \mapsto [$
                        $type\_identifier \mapsto$ "MSG_BLOCK",
                        $hash \mapsto SetToSeq(\{s.hash : s \in found\_blocks\})[h]$
                    ]
                ]
            ]
        ] ;
    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:
        $channels[local\_peer\_id][remote\_peer\_id] := [$
            $header \mapsto [command\_name \mapsto$ "getblocks"$]$,
            $payload \mapsto [$
                $hash\_count \mapsto Len(hashes)$,
                $block\_header\_hashes \mapsto hashes]$
        ] ;
    return ;
end procedure ;

  Given an inv message is received, send a getdata message to request the blocks.
procedure $inv(local\_peer\_id,\ remote\_peer\_id)$
begin
    $SendGetDataMsg$:
        $channels[local\_peer\_id][remote\_peer\_id] := [$
            $header \mapsto [command\_name \mapsto$ “getdata”$],$
            $payload \mapsto channels[local\_peer\_id][remote\_peer\_id].payload$
        ] ;
    return ;
end procedure ;

  Incorporate data to the local peer from the inventory received.
procedure $getdata(local\_peer\_id,\ remote\_peer\_id)$
variables $blocks\_data$ ;
begin
    $Incorporate$:
        $blocks\_data := [item \in 1 \mathrel{..} Len(channels[local\_peer\_id][remote\_peer\_id].payload.inventory) \mapsto$
            $Ops\,!\,FindBlockByHash($
                $Ops\,!\,GetPeerBlocks(the\_network[local\_peer\_id].peer\_set[remote\_peer\_id].address),$
                $channels[local\_peer\_id][remote\_peer\_id].payload.inventory[item].hash$
            )
        ] ;
        $the\_network[local\_peer\_id].blocks := the\_network[local\_peer\_id].blocks \cup ToSet(blocks\_data)$ ;
    $UpdateTip$:
        $the\_network[local\_peer\_id].chain\_tip := [$
            $height \mapsto blocks\_data[Len(blocks\_data)].height,$
            $hash \mapsto blocks\_data[Len(blocks\_data)].hash$
        ] ;
    return ;
end procedure ;

  A set of listener process for each peer to listen to incoming messages and act accordingly.
process $LISTENER \in 1 \mathrel{..} Len(RunningBlockchain)$
variables $command$ ;
begin
    $Listening$:
        await $Len(the\_network) \geq 2$ ;
        with $remote\_peer\_index \in 1 \mathrel{..} Len(the\_network[self].peer\_set)$ do
            if $channels[self][remote\_peer\_index].header = defaultInitValue$ then
                goto $Listening$ ;
            end if ;
        end with ;
    Requests:
        with remote_peer_index $\in 1 \,..\, Len(the\_network[self].peer\_set)$ do
            await $channels[self][remote\_peer\_index].header \neq defaultInitValue$ ;
            $command := channels[self][remote\_peer\_index].header.command\_name$ ;
            if $command =$ "addr" then
                call $addr(self, remote\_peer\_index)$ ;
                goto Listening ;
            elsif $command =$ "version" then
                call $version(self, remote\_peer\_index)$ ;
                goto Listening ;
            elsif $command =$ "verack" then
                call $verack(self, remote\_peer\_index)$ ;
            elsif $command =$ "getblocks" then
                call $getblocks(self, remote\_peer\_index)$ ;
                goto Listening ;
            elsif $command =$ "inv" then
                call $inv(self, remote\_peer\_index)$ ;
                goto Listening ;
            elsif $command =$ "getdata" then
                call $getdata(self, remote\_peer\_index)$ ;
            end if ;
        end with ;
    ListenerLoop:
        with remote_peer_index $\in 1 \,..\, Len(the\_network[self].peer\_set)$ do
            $channels[self][remote\_peer\_index] :=$
                $[header \mapsto defaultInitValue, payload \mapsto defaultInitValue]$ ;
            goto Listening ;
        end with ;
end process ;

  A set of processes to synchronize each peer with the network.
process $SYNCHRONIZER \in PeerProcessDiffId + 1 \,..\, PeerProcessDiffId + Len(RunningBlockchain)$
variables $local\_peer\_index = self - PeerProcessDiffId,\ best\_tip = 0$ ;
begin
    Announce:
          The network must have at least two peer.
        assert $Len(the\_network) \geq 2$ ;

          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 $\in 1 \,..\, Len(the\_network[local\_peer\_index].peer\_set)$ do
            call $announce(local\_peer\_index, remote\_peer\_index)$ ;
        end with ;
  $RequestInventory$:
    with $remote\_peer\_index \in 1 \,..\, Len(the\_network[local\_peer\_index].peer\_set)$ do
        Make sure the connection is established before requesting any block from this peer.
      await $the\_network[local\_peer\_index].peer\_set[remote\_peer\_index].established = \text{TRUE}$ ;

        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 $channels[local\_peer\_index][remote\_peer\_index].header = defaultInitValue$
          $\land\ channels[local\_peer\_index][remote\_peer\_index].payload = defaultInitValue$ ;

        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 $request\_blocks(\langle\rangle, local\_peer\_index, remote\_peer\_index)$ ;
        else
          call $request\_blocks($
              $\langle the\_network[local\_peer\_index].chain\_tip.hash\rangle$,
              $local\_peer\_index$,
              $remote\_peer\_index$
          $)$ ;
        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 $remote\_peer\_index \in 1 \,..\, Len(the\_network[local\_peer\_index].peer\_set)$ do
        await $the\_network[local\_peer\_index].peer\_set[remote\_peer\_index].established = \text{TRUE}$
            $\land\ channels[local\_peer\_index][remote\_peer\_index].header = defaultInitValue$
            $\land\ channels[local\_peer\_index][remote\_peer\_index].payload = defaultInitValue$ ;
      end with ;
      print "Peer is in sync!" ;
    end if ;
end process ;

end algorithm ;
  BEGIN TRANSLATION$(chksum(pcal) = $ "6acb42eb" $\land\ 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 $\triangleq$ $(1 \,..\, Len(RunningBlockchain)) \cup (PeerProcessDiffId + 1 \,..\, PeerProcessDiffId + Len(Running$

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

$\land\ command = [self \in 1 \mathrel{..} Len(RunningBlockchain) \mapsto defaultInitValue]$

Process SYNCHRONIZER

$\land\ local\_peer\_index = [self \in PeerProcessDiffId + 1 \mathrel{..} PeerProcessDiffId + Len(RunningBlockchai$

$\land\ best\_tip = [self \in PeerProcessDiffId + 1 \mathrel{..} PeerProcessDiffId + Len(RunningBlockchain) \mapsto 0]$

$\land\ stack = [self \in ProcSet \mapsto \langle\rangle]$

$\land\ pc = [self \in ProcSet \mapsto \text{CASE } self \in 1 \mathrel{..} Len(RunningBlockchain) \rightarrow \text{“Listening”}$

$\qquad\qquad\qquad\qquad \Box\ \ self \in PeerProcessDiffId + 1 \mathrel{..} PeerProcessDiffId + Len(Running$

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

$\qquad\qquad\qquad\qquad\ \land\ channels' = [channels \text{ EXCEPT } ![local\_peer\_id\_[self]][remote\_peer\_id\_[self]] =$

```
                                                                                ]]
                              ∧ pc′ = [pc EXCEPT ![self] = Head(stack[self]).pc]
                              ∧ local_peer_id_′ = [local_peer_id_ EXCEPT ![self] = Head(stack[self]).local_peer_
                              ∧ remote_peer_id_′ = [remote_peer_id_ EXCEPT ![self] = Head(stack[self]).remot
                              ∧ stack′ = [stack EXCEPT ![self] = Tail(stack[self])]
                              ∧ UNCHANGED ⟨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⟩

announce(self) ≜ SendAddrMsg(self)

SendVersionMsg(self) ≜ ∧ pc[self] = "SendVersionMsg"
                       ∧ channels′ = [channels EXCEPT ![local_peer_id_a[self]][remote_peer_id_a[self





                       ∧ pc′ = [pc EXCEPT ![self] = Head(stack[self]).pc]
                       ∧ local_peer_id_a′ = [local_peer_id_a EXCEPT ![self] = Head(stack[self]).local_
                       ∧ remote_peer_id_a′ = [remote_peer_id_a EXCEPT ![self] = Head(stack[self]).r
                       ∧ stack′ = [stack EXCEPT ![self] = Tail(stack[self])]
                       ∧ UNCHANGED ⟨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,
```

$$\begin{aligned}
&\mathit{remote\_peer\_id\_r},\ \mathit{local\_peer\_id\_i},\\
&\mathit{remote\_peer\_id\_i},\ \mathit{local\_peer\_id},\\
&\mathit{remote\_peer\_id},\ \mathit{blocks\_data},\ \mathit{command},\\
&\mathit{local\_peer\_index},\ \mathit{best\_tip}\rangle
\end{aligned}$$

$$\mathit{addr}(\mathit{self}) \triangleq \mathit{SendVersionMsg}(\mathit{self})$$

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

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

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

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

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

$HandleGetBlocksMsg(self) \triangleq \land\ pc[self] = \text{“HandleGetBlocksMsg”}$
$\quad \land\ hash\_count' = [hash\_count \text{ EXCEPT } ![self] = channels[local\_peer\_id\_g[se$
$\quad \land\ block\_header\_hashes' = [block\_header\_hashes \text{ EXCEPT } ![self] = channels$
$\quad \land\ remote\_peer\_blocks' = [remote\_peer\_blocks \text{ EXCEPT } ![self] = Ops!GetPe$
$\quad \land\ \text{IF } hash\_count'[self] = 0$
$\qquad \text{THEN } \land\ start\_height' = [start\_height \text{ EXCEPT } ![self] = 1]$
$\qquad \text{ELSE } \land\ start\_height' = [start\_height \text{ EXCEPT } ![self] = Ops!FindBlo$
$\quad \land\ end\_height' = [end\_height \text{ EXCEPT } ![self] = start\_height'[self] + (MaxG$
$\quad \land\ found\_blocks' = [found\_blocks \text{ EXCEPT } ![self] = Ops!FindBlocks(remote$
$\quad \land\ pc' = [pc \text{ EXCEPT } ![self] = \text{“SendInvMsg”}]$
$\quad \land\ \text{UNCHANGED } \langle the\_network, channels, stack,$
$\qquad local\_peer\_id\_, remote\_peer\_id\_,$
$\qquad local\_peer\_id\_a, remote\_peer\_id\_a,$
$\qquad local\_peer\_id\_v, remote\_peer\_id\_v,$
$\qquad local\_peer\_id\_ve,$
$\qquad remote\_peer\_id\_ve, local\_peer\_id\_g,$
$\qquad remote\_peer\_id\_g, hashes,$
$\qquad local\_peer\_id\_r, remote\_peer\_id\_r,$
$\qquad local\_peer\_id\_i, remote\_peer\_id\_i,$
$\qquad local\_peer\_id, remote\_peer\_id,$
$\qquad blocks\_data, command,$
$\qquad local\_peer\_index, best\_tip\rangle$

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

$\quad \land channels' = [channels \text{ EXCEPT } ![local\_peer\_id\_g[self]][remote\_peer\_id\_g[self]] =$

$]]$

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

$\quad \land found\_blocks' = [found\_blocks \text{ EXCEPT } ![self] = Head(stack[self]).found\_blocks]$

$\quad \land hash\_count' = [hash\_count \text{ EXCEPT } ![self] = Head(stack[self]).hash\_count]$

$\quad \land block\_header\_hashes' = [block\_header\_hashes \text{ EXCEPT } ![self] = Head(stack[self])$

$\quad \land remote\_peer\_blocks' = [remote\_peer\_blocks \text{ EXCEPT } ![self] = Head(stack[self]).re$

$\quad \land start\_height' = [start\_height \text{ EXCEPT } ![self] = Head(stack[self]).start\_height]$

$\quad \land end\_height' = [end\_height \text{ EXCEPT } ![self] = Head(stack[self]).end\_height]$

$\quad \land local\_peer\_id\_g' = [local\_peer\_id\_g \text{ EXCEPT } ![self] = Head(stack[self]).local\_peer$

$\quad \land remote\_peer\_id\_g' = [remote\_peer\_id\_g \text{ EXCEPT } ![self] = Head(stack[self]).remo$

$\quad \land stack' = [stack \text{ EXCEPT } ![self] = Tail(stack[self])]$

$\quad \land \text{UNCHANGED } \langle the\_network, local\_peer\_id\_,$
$\quad\quad remote\_peer\_id\_, local\_peer\_id\_a,$
$\quad\quad remote\_peer\_id\_a, local\_peer\_id\_v,$
$\quad\quad remote\_peer\_id\_v, local\_peer\_id\_ve,$
$\quad\quad remote\_peer\_id\_ve, hashes, local\_peer\_id\_r,$
$\quad\quad remote\_peer\_id\_r, local\_peer\_id\_i,$
$\quad\quad remote\_peer\_id\_i, local\_peer\_id,$
$\quad\quad remote\_peer\_id, blocks\_data, command,$
$\quad\quad local\_peer\_index, best\_tip\rangle$

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

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

$\quad \land channels' = [channels \text{ EXCEPT } ![local\_peer\_id\_r[self]][remote\_peer\_id\_r[se$

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

$\quad \land hashes' = [hashes \text{ EXCEPT } ![self] = Head(stack[self]).hashes]$

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

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

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

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



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

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

```
$Incorporate(self) \triangleq \land pc[self] = \text{"Incorporate"}$
    $\land blocks\_data' = [blocks\_data \text{ EXCEPT } ![self] =$          $[item \in 1 \mathrel{..} Len(char$
                                               $Ops!FindBlockByHash($
                                                   $Ops!GetPeerBlocks(the\_net$
                                                   $channels[local\_peer\_id[self]]$
```

```
                                                        )
                                                     ]]
                     ∧ the_network′ = [the_network EXCEPT ![local_peer_id[self]].blocks = the_network[l
                     ∧ pc′ = [pc EXCEPT ![self] = "UpdateTip"]
                     ∧ UNCHANGED ⟨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⟩

UpdateTip(self) ≜ ∧ pc[self] = "UpdateTip"
                  ∧ the_network′ = [the_network EXCEPT ![local_peer_id[self]].chain_tip =
                                                                              height ↦
                                                                              hash ↦ l
                                                                           ]]
                  ∧ pc′ = [pc EXCEPT ![self] = Head(stack[self]).pc]
                  ∧ blocks_data′ = [blocks_data EXCEPT ![self] = Head(stack[self]).blocks_data]
                  ∧ local_peer_id′ = [local_peer_id EXCEPT ![self] = Head(stack[self]).local_peer_id]
                  ∧ remote_peer_id′ = [remote_peer_id EXCEPT ![self] = Head(stack[self]).remote_pee
                  ∧ stack′ = [stack EXCEPT ![self] = Tail(stack[self])]
                  ∧ UNCHANGED ⟨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⟩

getdata(self) ≜ Incorporate(self) ∨ UpdateTip(self)

Listening(self) ≜ ∧ pc[self] = "Listening"
                  ∧ Len(the_network) ≥ 2
                  ∧ ∃ remote_peer_index ∈ 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 $![self] =$
```

```
            ∧ remote_peer_id_v' = [remote_peer_id_v EXCEPT ![se
            ∧ stack' = [stack EXCEPT ![self] = ⟨[procedure ↦  "ver
                                                 pc        ↦  "Lis
                                                 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⟩
ELSE    ∧ IF command'[self] = "verack"
          THEN  ∧ ∧ local_peer_id_ve' = [local_peer_id_ve EX
                  ∧ remote_peer_id_ve' = [remote_peer_id_v
                  ∧ stack' = [stack EXCEPT ![self] = ⟨[proce
                                                     pc
                                                     local_
                                                     remot
                                                     ∘ stac
                ∧ pc' = [pc EXCEPT ![self] = "HandleVerackM
                ∧ UNCHANGED ⟨local_peer_id_g,
                             remote_peer_id_g,
                             found_blocks,
                             hash_count,
                             block_header_hashes,
                             remote_peer_blocks,
                             start_height,
                             end_height,
                             local_peer_id_i,
                             remote_peer_id_i,
                             local_peer_id,
                             remote_peer_id,
                             blocks_data⟩
          ELSE  ∧ IF command'[self] = "getblocks"
```

```
THEN  ∧  ∧ local_peer_id_g′ = [local_pee
         ∧ remote_peer_id_g′ = [remote
         ∧ stack′ = [stack EXCEPT ![se







       ∧ found_blocks′ = [found_blocks
       ∧ hash_count′ = [hash_count EXC
       ∧ block_header_hashes′ = [block_
       ∧ remote_peer_blocks′ = [remote
       ∧ start_height′ = [start_height E
       ∧ end_height′ = [end_height EXC
       ∧ pc′ = [pc EXCEPT ![self] = "Ha
       ∧ UNCHANGED ⟨local_peer_id_i,
                     remote_peer_id_
                     local_peer_id,
                     remote_peer_id,
                     blocks_data⟩
ELSE   ∧ IF command′[self] = "inv"
         THEN  ∧  ∧ local_peer_id_i′
                  ∧ remote_peer_id
                  ∧ stack′ = [stack



               ∧ pc′ = [pc EXCEPT
               ∧ UNCHANGED ⟨loca
                             rem
                             bloc
         ELSE  ∧ IF command′[self]
                  THEN  ∧  ∧ loca
                           ∧ rem
                           ∧ sta
```

```
                                                                  ∧ blocks
                                                                  ∧ pc′ =
                                                            ELSE  ∧ pc′ =
                                                                  ∧ UNCH


                                                          ∧ UNCHANGED ⟨loca
                                                                       rem
                                                    ∧ UNCHANGED ⟨local_peer_id_g,
                                                                 remote_peer_id_
                                                                 found_blocks,
                                                                 hash_count,
                                                                 block_header_ha
                                                                 remote_peer_blo
                                                                 start_height,
                                                                 end_height⟩
                                              ∧ UNCHANGED ⟨local_peer_id_ve,
                                                           remote_peer_id_ve⟩
                                        ∧ UNCHANGED ⟨local_peer_id_v,
                                                     remote_peer_id_v⟩
                                  ∧ UNCHANGED ⟨local_peer_id_a,
                                               remote_peer_id_a⟩
                          ∧ UNCHANGED ⟨the_network, channels, local_peer_id_,
                                       remote_peer_id_, hashes, local_peer_id_r,
                                       remote_peer_id_r, local_peer_index, best_tip⟩

ListenerLoop(self) ≜  ∧ pc[self] = "ListenerLoop"
                      ∧ ∃ remote_peer_index ∈ 1 .. Len(the_network[self].peer_set) :
                           ∧ channels′ = [channels EXCEPT ![self][remote_peer_index] = [header ↦ defa
                           ∧ pc′ = [pc EXCEPT ![self] = "Listening"]
                      ∧ UNCHANGED ⟨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⟩
```

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

$Announce(self) \triangleq \land pc[self] = $ "Announce"
    $\land Assert(Len(the\_network) \geq 2,$
            "Failure of assertion at line 224, column 9.")
    $\land Len(the\_network[local\_peer\_index[self]].peer\_set) > 0$
    $\land \exists remote\_peer\_index \in 1 \,..\, Len(the\_network[local\_peer\_index[self]].peer\_set) :$
        $\land \land local\_peer\_id\_' = [local\_peer\_id\_ \text{ EXCEPT } ![self] = local\_peer\_index[self]]$
          $\land remote\_peer\_id\_' = [remote\_peer\_id\_ \text{ EXCEPT } ![self] = remote\_peer\_index$
          $\land stack' = [stack \text{ EXCEPT } ![self] = \langle[procedure \mapsto$ "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] = $ "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] = $ "RequestInventory"
    $\land \exists remote\_peer\_index \in 1 \,..\, Len(the\_network[local\_peer\_index[self]].peer\_set$
        $\land the\_network[local\_peer\_index[self]].peer\_set[remote\_peer\_index].establis$
        $\land \text{IF } the\_network[local\_peer\_index[self]].peer\_set[remote\_peer\_index].tip >$
              $\text{THEN } \land best\_tip' = [best\_tip \text{ EXCEPT } ![self] = the\_network[local\_pe$
              $\text{ELSE } \land \text{TRUE}$
                     $\land \text{UNCHANGED } best\_tip$
        $\land \quad channels[local\_peer\_index[self]][remote\_peer\_index].header = defaultIn$
          $\land channels[local\_peer\_index[self]][remote\_peer\_index].payload = defaul$
        $\land \text{IF } the\_network[local\_peer\_index[self]].chain\_tip.height <$
              $the\_network[local\_peer\_index[self]].peer\_set[remote\_peer\_index].tip$
              $\text{THEN } \land \text{IF } the\_network[local\_peer\_index[self]].chain\_tip.height = 0$
                         $\text{THEN } \land \land hashes' = [hashes \text{ EXCEPT } ![self] = \langle\rangle]$
                                  $\land local\_peer\_id\_r' = [local\_peer\_id\_r \text{ EXCEPT}$
                                  $\land remote\_peer\_id\_r' = [remote\_peer\_id\_r \text{ EXC}$
                                  $\land stack' = [stack \text{ EXCEPT } ![self] = \langle[procedure$
                                                                   $pc$
                                                                   $hashes$
```

```
                                                                                local_peer
                                                                                remote_p
                                                                                ∘ stack[se
                                                     ∧ pc′ = [pc EXCEPT ![self] = "SendGetBlocksMsg
                                                ELSE ∧ ∧ hashes′ = [hashes EXCEPT ![self] = ⟨the_ne
                                                       ∧ local_peer_id_r′ = [local_peer_id_r EXCEPT
                                                       ∧ remote_peer_id_r′ = [remote_peer_id_r EXC
                                                       ∧ stack′ = [stack EXCEPT ![self] = ⟨[procedure
                                                                                pc
                                                                                hashes
                                                                                local_peer
                                                                                remote_p
                                                                                ∘ stack[se
                                                     ∧ pc′ = [pc EXCEPT ![self] = "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, $hashes$,
$local\_peer\_id\_r$, $remote\_peer\_id\_r$,
$local\_peer\_id\_i$, $remote\_peer\_id\_i$,
$local\_peer\_id$, $remote\_peer\_id$, $blocks\_data$,
$command$, $local\_peer\_index$, $best\_tip\rangle$

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

Allow infinite stuttering to prevent deadlock on termination.

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

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

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

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

END TRANSLATION