MODULE $p2p$

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

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

Define the network to be used by the algorithm.

CONSTANT *RunningBlockchain*

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

CONSTANT *MaxGetBlocksInvResponse*

Maximum number of outbound connections a peer can have.

CONSTANT *MaxConnectionsPerPeer*

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

$PeerProcessDiffId \triangleq 1000$

--algorithm $p2p$

variables

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

```
$the\_network = RunningBlockchain$ ;
```

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

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

define

Import the operators used in the algorithm.

LOCAL $Ops \triangleq$ INSTANCE *Operators*

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

```
$ExistsSyncPath \triangleq$
    $\exists\, peer1, peer2 \in 1\,..\,Len(RunningBlockchain) :$
        $\Diamond(the\_network[peer1].chain\_tip = the\_network[peer2].chain\_tip)$
```

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

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

```
    $Liveness \triangleq$
        $\forall peer1, peer2 \in 1 \,..\, Len(RunningBlockchain) :$
            $\Diamond(the\_network[peer1].chain\_tip = the\_network[peer2].chain\_tip)$
**end define ;**
```

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

```
**procedure** $announce(local\_peer\_id, remote\_peer\_id)$
**begin**
    $SendAddrMsg:$
        $channels[local\_peer\_id][remote\_peer\_id] := [$
            $header \mapsto [command\_name \mapsto$ "addr"$]$,
            $payload \mapsto [$
                $address\_count \mapsto 1,$
                  Only a single address is supported.
                $addresses \mapsto the\_network[local\_peer\_id].peer$
            $]$
        $]$ ;
    **return ;**
**end procedure ;**
```

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

```
**procedure** $addr(local\_peer\_id, remote\_peer\_id)$
**begin**
    $SendVersionMsg:$
        $channels[local\_peer\_id][remote\_peer\_id] := [$
            $header \mapsto [command\_name \mapsto$ "version"$]$,
            $payload \mapsto [$
                $addr\_recv \mapsto the\_network[local\_peer\_id].peer,$
                $addr\_trans \mapsto the\_network[local\_peer\_id].peer\_set[remote\_peer\_id].address,$
                $start\_height \mapsto$
                    $Ops!GetPeerTip(the\_network[local\_peer\_id].peer\_set[remote\_peer\_id].address)]$
        $]$ ;
    **return ;**
**end procedure ;**
```

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

```
**procedure** $version(local\_peer\_id, remote\_peer\_id)$
**begin**
    $HandleVersionMsg:$
        $the\_network[local\_peer\_id].peer\_set[remote\_peer\_id].tip :=$
            $channels[local\_peer\_id][remote\_peer\_id].payload.start\_height$ ;
    $SendVerackMsg:$
        $channels[local\_peer\_id][remote\_peer\_id] := [$
            $header \mapsto [command\_name \mapsto$ "verack"$]$,
            $payload \mapsto defaultInitValue$
        $]$ ;
    **return** ;
**end procedure** ;

  *Given a verack message is received, establish the connection.*
**procedure** $verack(local\_peer\_id,\ remote\_peer\_id)$
**begin**
    $HandleVerackMsg$:
        $the\_network[local\_peer\_id].peer\_set[remote\_peer\_id].established := \text{TRUE}$ ;
    **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
    $Incorporate$:
        $blocks\_data := [item \in 1 \,..\, Len(channels[local\_peer\_id][remote\_peer\_id].payload.inventory) \mapsto$
            $Ops!FindBlockByHash($
                $Ops!GetPeerBlocks(the\_network[local\_peer\_id].peer\_set[remote\_peer\_id].address)$,
                $channels[local\_peer\_id][remote\_peer\_id].payload.inventory[item].hash$
            $)$
        ];
        $the\_network[local\_peer\_id].blocks := the\_network[local\_peer\_id].blocks \cup ToSet(blocks\_data)$;
    $UpdateTip$:
        $the\_network[local\_peer\_id].chain\_tip := [$
            $height \mapsto blocks\_data[Len(blocks\_data)].height$,
                $hash \mapsto blocks\_data[Len(blocks\_data)].hash$
            ];
    return;
end procedure ;
```

A set of listener process for each peer to listen to incoming messages and act accordingly.

```
process $LISTENER \in 1 \ .. \ Len(RunningBlockchain)$
variables $command$;
begin
    $Listening$:
        await $Len(the\_network) \geq 2$;
        with $remote\_peer\_index \in 1 \ .. \ Len(the\_network[self].peer\_set)$ do
            if $channels[self][remote\_peer\_index].header = defaultInitValue$ then
                goto $Listening$;
            end if ;
        end with ;
    $Requests$:
        with $remote\_peer\_index \in 1 \ .. \ Len(the\_network[self].peer\_set)$ do
            await $channels[self][remote\_peer\_index].header \neq defaultInitValue$;
            $command := channels[self][remote\_peer\_index].header.command\_name$;
            if $command$ = "addr" then
                call $addr(self, remote\_peer\_index)$;
                goto $Listening$;
            elsif $command$ = "version" then
                call $version(self, remote\_peer\_index)$;
                goto $Listening$;
            elsif $command$ = "verack" then
                call $verack(self, remote\_peer\_index)$;
            elsif $command$ = "getblocks" then
                call $getblocks(self, remote\_peer\_index)$;
                goto $Listening$;
            elsif $command$ = "inv" then
                call $inv(self, remote\_peer\_index)$;
                goto $Listening$;
            elsif $command$ = "getdata" then
                call $getdata(self, remote\_peer\_index)$;
            end if ;
        end with ;
    $ListenerLoop$:
        with $remote\_peer\_index \in 1 \ .. \ Len(the\_network[self].peer\_set)$ 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 = 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
          $\wedge$ 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 $remote\_peer\_index \in 1 \dots Len(the\_network[local\_peer\_index].peer\_set)$ 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 ;
        end if ;
end process ;

end algorithm ;
  BEGIN TRANSLATION$(chksum(pcal) = $ "9b0fbec8" $\land\, chksum(tla) = $ "ef85b7dc")
  Parameter local_peer_id of procedure announce at line 66 col 20 changed to local_peer_id_
  Parameter remote_peer_id of procedure announce at line 66 col 35 changed to remote_peer_id_
  Parameter local_peer_id of procedure addr at line 81 col 16 changed to local_peer_id_a
  Parameter remote_peer_id of procedure addr at line 81 col 31 changed to remote_peer_id_a
  Parameter local_peer_id of procedure version at line 96 col 19 changed to local_peer_id_v
  Parameter remote_peer_id of procedure version at line 96 col 34 changed to remote_peer_id_v
  Parameter local_peer_id of procedure verack at line 110 col 18 changed to local_peer_id_ve
  Parameter remote_peer_id of procedure verack at line 110 col 33 changed to remote_peer_id_ve
  Parameter local_peer_id of procedure getblocks at line 118 col 21 changed to local_peer_id_g
  Parameter remote_peer_id of procedure getblocks at line 118 col 36 changed to remote_peer_id_g
  Parameter local_peer_id of procedure request_blocks at line 160 col 34 changed to local_peer_id_r
  Parameter remote_peer_id of procedure request_blocks at line 160 col 49 changed to remote_peer_id_r
  Parameter local_peer_id of procedure inv at line 173 col 15 changed to local_peer_id_i
  Parameter remote_peer_id of procedure inv at line 173 col 30 changed to remote_peer_id_i
CONSTANT $defaultInitValue$
VARIABLES $the\_network, channels, pc, stack$

  define statement
LOCAL $Ops \triangleq$ INSTANCE $Operators$




$ExistsSyncPath \triangleq$
    $\exists\, peer1, peer2 \in 1 \dots Len(RunningBlockchain) :$
        $\Diamond(the\_network[peer1].chain\_tip = the\_network[peer2].chain\_tip)$
```

```
Liveness ≜
   ∀ peer1, peer2 ∈ 1 .. Len(RunningBlockchain) :
      ◇(the_network[peer1].chain_tip = the_network[peer2].chain_tip)

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 ≜ ⟨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⟩

ProcSet ≜ (1 .. Len(RunningBlockchain)) ∪ (PeerProcessDiffId + 1 .. PeerProcessDiffId + Len(Running

Init ≜ Global variables
       ∧ the_network = RunningBlockchain
       ∧ channels =         [i ∈ 1 .. Len(the_network) ↦
                     [j ∈ 1 .. MaxConnectionsPerPeer ↦ [
                        header ↦ defaultInitValue,
                        payload ↦ defaultInitValue
                     ]]
                  ]
        Procedure announce
       ∧ local_peer_id_ = [self ∈ ProcSet ↦ defaultInitValue]
       ∧ remote_peer_id_ = [self ∈ ProcSet ↦ defaultInitValue]
        Procedure addr
       ∧ local_peer_id_a = [self ∈ ProcSet ↦ defaultInitValue]
       ∧ remote_peer_id_a = [self ∈ ProcSet ↦ defaultInitValue]
        Procedure version
       ∧ local_peer_id_v = [self ∈ ProcSet ↦ defaultInitValue]
       ∧ remote_peer_id_v = [self ∈ ProcSet ↦ defaultInitValue]
        Procedure verack
       ∧ local_peer_id_ve = [self ∈ ProcSet ↦ defaultInitValue]
       ∧ remote_peer_id_ve = [self ∈ ProcSet ↦ defaultInitValue]
        Procedure getblocks
       ∧ local_peer_id_g = [self ∈ ProcSet ↦ defaultInitValue]
       ∧ remote_peer_id_g = [self ∈ ProcSet ↦ 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

$\land\ local\_peer\_index = [self\ \in PeerProcessDiffId + 1 \mathrel{..} PeerProcessDiffId + Len(RunningBlockchai$
$\land\ best\_tip = [self\ \ \in PeerProcessDiffId + 1 \mathrel{..} PeerProcessDiffId + Len(RunningBlockchain) \mapsto 0]$
$\land\ stack = [self\ \in ProcSet \mapsto \langle\rangle]$
$\land\ pc = [self\ \in ProcSet \mapsto \textsc{case}\ self\ \in 1 \mathrel{..} Len(RunningBlockchain) \to \text{“Listening”}$
$\qquad\qquad\qquad\qquad\qquad \Box\ \ self\ \in PeerProcessDiffId + 1 \mathrel{..} PeerProcessDiffId + Len(Running$

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

$]]$

$\land\ pc' = [pc\ \textsc{except}\ ![self] = Head(stack[self]).pc]$
$\land\ local\_peer\_id\_' = [local\_peer\_id\_\ \textsc{except}\ ![self] = Head(stack[self]).local\_peer\_$
$\land\ remote\_peer\_id\_' = [remote\_peer\_id\_\ \textsc{except}\ ![self] = Head(stack[self]).remot$
$\land\ stack' = [stack\ \textsc{except}\ ![self] = Tail(stack[self])]$
$\land\ \textsc{unchanged}\ \langle the\_network,\ local\_peer\_id\_a,$
$\qquad remote\_peer\_id\_a,\ local\_peer\_id\_v,$
$\qquad remote\_peer\_id\_v,\ local\_peer\_id\_ve,$
$\qquad remote\_peer\_id\_ve,\ local\_peer\_id\_g,$
$\qquad remote\_peer\_id\_g,\ found\_blocks,$
$\qquad hash\_count,\ block\_header\_hashes,$

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

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

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

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

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

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

```
                                        hash_count, block_header_hashes,
                                        remote_peer_blocks, start_height,
                                        end_height, hashes, local_peer_id_r,
                                        remote_peer_id_r, local_peer_id_i,
                                        remote_peer_id_i, local_peer_id,
                                        remote_peer_id, blocks_data, command,
                                        local_peer_index, best_tip⟩

SendVerackMsg(self) ≜ ∧ pc[self] = "SendVerackMsg"
                      ∧ channels′ = [channels EXCEPT ![local_peer_id_v[self]][remote_peer_id_v[self]]



                      ∧ pc′ = [pc EXCEPT ![self] = Head(stack[self]).pc]
                      ∧ local_peer_id_v′ = [local_peer_id_v EXCEPT ![self] = Head(stack[self]).local_p
                      ∧ remote_peer_id_v′ = [remote_peer_id_v EXCEPT ![self] = Head(stack[self]).re
                      ∧ stack′ = [stack EXCEPT ![self] = Tail(stack[self])]
                      ∧ UNCHANGED ⟨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⟩

version(self) ≜ HandleVersionMsg(self) ∨ SendVerackMsg(self)

HandleVerackMsg(self) ≜ ∧ pc[self] = "HandleVerackMsg"
                        ∧ the_network′ = [the_network EXCEPT ![local_peer_id_ve[self]].peer_set[rem
                        ∧ pc′ = [pc EXCEPT ![self] = Head(stack[self]).pc]
                        ∧ local_peer_id_ve′ = [local_peer_id_ve EXCEPT ![self] = Head(stack[self]).lo
                        ∧ remote_peer_id_ve′ = [remote_peer_id_ve EXCEPT ![self] = Head(stack[sel
                        ∧ 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_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⟩

verack(self) ≜ HandleVerackMsg(self)

HandleGetBlocksMsg(self) ≜ ∧ pc[self] = "HandleGetBlocksMsg"
                           ∧ hash_count′ = [hash_count EXCEPT ![self] = channels[local_peer_id_g[se
                           ∧ block_header_hashes′ = [block_header_hashes EXCEPT ![self] = channels
                           ∧ remote_peer_blocks′ = [remote_peer_blocks EXCEPT ![self] = Ops!GetPe
                           ∧ IF hash_count′[self] = 0
                                THEN ∧ start_height′ = [start_height EXCEPT ![self] = 1]
                                ELSE ∧ start_height′ = [start_height EXCEPT ![self] = Ops!FindBlo
                           ∧ end_height′ = [end_height EXCEPT ![self] = start_height′[self] + (MaxG
                           ∧ found_blocks′ = [found_blocks EXCEPT ![self] = Ops!FindBlocks(remote
                           ∧ pc′ = [pc EXCEPT ![self] = "SendInvMsg"]
                           ∧ 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, 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⟩

SendInvMsg(self) ≜ ∧ pc[self] = "SendInvMsg"
                   ∧ channels′ = [channels EXCEPT ![local_peer_id_g[self]][remote_peer_id_g[self]] =









                                                                                              ]]
                   ∧ pc′ = [pc EXCEPT ![self] = Head(stack[self]).pc]
                   ∧ found_blocks′ = [found_blocks EXCEPT ![self] = Head(stack[self]).found_blocks]
                   ∧ hash_count′ = [hash_count EXCEPT ![self] = Head(stack[self]).hash_count]
```

```
                        ∧ block_header_hashes′ = [block_header_hashes EXCEPT ![self] = Head(stack[self])
                        ∧ remote_peer_blocks′ = [remote_peer_blocks EXCEPT ![self] = Head(stack[self]).re
                        ∧ start_height′ = [start_height EXCEPT ![self] = Head(stack[self]).start_height]
                        ∧ end_height′ = [end_height EXCEPT ![self] = Head(stack[self]).end_height]
                        ∧ local_peer_id_g′ = [local_peer_id_g EXCEPT ![self] = Head(stack[self]).local_peer
                        ∧ remote_peer_id_g′ = [remote_peer_id_g EXCEPT ![self] = Head(stack[self]).remo
                        ∧ stack′ = [stack EXCEPT ![self] = Tail(stack[self])]
                        ∧ UNCHANGED ⟨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⟩

getblocks(self) ≜ HandleGetBlocksMsg(self) ∨ SendInvMsg(self)

SendGetBlocksMsg(self) ≜ ∧ pc[self] = "SendGetBlocksMsg"
                         ∧ channels′ = [channels EXCEPT ![local_peer_id_r[self]][remote_peer_id_r[se




                         ∧ pc′ = [pc EXCEPT ![self] = Head(stack[self]).pc]
                         ∧ hashes′ = [hashes EXCEPT ![self] = Head(stack[self]).hashes]
                         ∧ local_peer_id_r′ = [local_peer_id_r EXCEPT ![self] = Head(stack[self]).loca
                         ∧ remote_peer_id_r′ = [remote_peer_id_r EXCEPT ![self] = Head(stack[self]
                         ∧ stack′ = [stack EXCEPT ![self] = Tail(stack[self])]
                         ∧ UNCHANGED ⟨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⟩

request_blocks(self)     ≜ SendGetBlocksMsg(self)
```

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

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

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

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

```
                    best_tip⟩

UpdateTip(self) ≜ ∧ pc[self] = "UpdateTip"
                  ∧ the_network′ = [the_network EXCEPT ![local_peer_id[self]].chain_tip =
                                                                                          height ↦
                                                                                          hash ↦
                                                                                      ]]
                  ∧ 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 ∧ pc′ = [pc EXCEPT ![self] = "Listening"]
                          ELSE ∧ pc′ = [pc EXCEPT ![self] = "Requests"]
                  ∧ 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⟩
```

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

```
                                  $end\_height$,
                                  $local\_peer\_id\_i$,
                                  $remote\_peer\_id\_i$,
                                  $local\_peer\_id$,
                                  $remote\_peer\_id$,
                                  $blocks\_data\rangle$
ELSE  $\wedge$ IF $command'[self] = $ “verack”
           THEN $\wedge$ $\wedge$ $local\_peer\_id\_ve' = [local\_peer\_id\_ve$ EX
                  $\wedge$ $remote\_peer\_id\_ve' = [remote\_peer\_id\_v$
                  $\wedge$ $stack' = [stack$ EXCEPT $![self] = \langle[proce$
                                                              $pc$
                                                              $local\_$
                                                              $remot$
                                                              $\circ\ stac$
                $\wedge$ $pc' = [pc$ EXCEPT $![self] = $ “HandleVerackM
                $\wedge$ 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  $\wedge$ IF $command'[self] = $ “getblocks”
                      THEN $\wedge$ $\wedge$ $local\_peer\_id\_g' = [local\_pee$
                             $\wedge$ $remote\_peer\_id\_g' = [remot$
                             $\wedge$ $stack' = [stack$ EXCEPT $![se$







                           $\wedge$ $found\_blocks' = [found\_blocks$
                           $\wedge$ $hash\_count' = [hash\_count$ EX
                           $\wedge$ $block\_header\_hashes' = [block\_$
```

```
         ∧ remote_peer_blocks' = [remote_
         ∧ start_height' = [start_height EX
         ∧ end_height' = [end_height EXC
         ∧ pc' = [pc EXCEPT ![self] = "H
         ∧ 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  ∧ ∧ loc
                               ∧ 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) ≜ Listening(self) ∨ Requests(self) ∨ ListenerLoop(self)

Announce(self) ≜ ∧ pc[self] = "Announce"
                 ∧ Assert(Len(the_network) ≥ 2,
                          "Failure of assertion at line 250, column 9.")
                 ∧ Len(the_network[local_peer_index[self]].peer_set) > 0
                 ∧ ∃ remote_peer_index ∈ 1 .. Len(the_network[local_peer_index[self]].peer_set) :
                      ∧ ∧ local_peer_id_′ = [local_peer_id_ EXCEPT ![self] = local_peer_index[self]]
                        ∧ remote_peer_id_′ = [remote_peer_id_ EXCEPT ![self] = remote_peer_index
                        ∧ stack′ = [stack EXCEPT ![self] = ⟨[procedure ↦ "announce",
                                                            pc        ↦ "RequestInventory",
                                                            local_peer_id_ ↦ local_peer_id_[self],
                                                            remote_peer_id_ ↦ remote_peer_id_[se
                                                            ∘ stack[self]]
                      ∧ pc′ = [pc EXCEPT ![self] = "SendAddrMsg"]
                 ∧ UNCHANGED ⟨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⟩

RequestInventory(self) ≜ ∧ pc[self] = "RequestInventory"
                         ∧ ∃ remote_peer_index ∈ 1 . . Len(the_network[local_peer_index[self]].peer_set
                              ∧ the_network[local_peer_index[self]].peer_set[remote_peer_index].establis
                              ∧ IF the_network[local_peer_index[self]].peer_set[remote_peer_index].tip >
                                   THEN ∧ best_tip′ = [best_tip EXCEPT ![self] = the_network[local_pe
                                   ELSE ∧ TRUE
                                        ∧ UNCHANGED best_tip
                              ∧   channels[local_peer_index[self]][remote_peer_index].header = defaultIr
                                ∧ channels[local_peer_index[self]][remote_peer_index].payload = defaul
                              ∧ IF the_network[local_peer_index[self]].chain_tip.height <
                                   the_network[local_peer_index[self]].peer_set[remote_peer_index].tip
                                   THEN ∧ IF the_network[local_peer_index[self]].chain_tip.height = 0
                                             THEN ∧ ∧ hashes′ = [hashes EXCEPT ![self] = ⟨⟩]
                                                    ∧ 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 ∧ ∧ 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
                            ∧ 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.
Terminating ≜ ∧ ∀ self ∈ ProcSet : pc[self] = "Done"
              ∧ UNCHANGED vars
```

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

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

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

END TRANSLATION