Algorithm 1: Programmable consensus workflow at node ni.

// Implement modules

Dictionary Supported\_PC\_FILTER\_FUNC = {“f1”: Filter1, “f2”: Filter2, “f3”: Filter3, …}

Dictionary Supported\_PC\_SELECT\_FUNC = {“s1”: Select1, “s2”: Select2, “s3”: Select3, …}

Dictionary Supported\_PC\_ORDER\_FUNC = {“o1”: Order1, “o2”: Order2, “o3”: Order3, …}

Dictionary Supported\_PC\_MIDPROC\_FUNC = {“m1”: MidProc1, “m2”: MidProc2, “m3”: MidProc3, …}

Function Filter1(Tx): // Input Tx, Output if\_pass(true/false)

if Tx.Signature = sig(Tx.TxHash, Tx.Sender) then

return true

else

return false

Function Select1(Txs): // Input Txs, Output Txs\_selected

foreach Tx in Txs do

if Tx.CreatedTime > Now – Timeout then

append Tx to Txs\_selected

remove Tx from Txs

return Txs\_selected

Function Order1(Txs): // do not order which is also one strategy

return Txs

Function MidProc1(Txs, sandbox, localWS): // Input Txs, Output Txs\_processed

foreach Tx in Txs do

{ Tx.RS, Tx.WS} <- execute Tx based on sandbox which can read from localWS when the key-value pair getted is not contained in sandbox, but sandbox cann’t write to localWS

foreach w in Tx.WS do // assume Tx can executed successful

sandbox[w.key] = w.value

Tx.Executed = true

return Txs

// Initialize modules on ni

Input: The map of supported Tx types and pool name it allocated TPM, pool names PNs, pool filter names PFNs, pool selector names PSNs, pool orderer names PONs, pool mid-processer names PMNs and pool consensus protocol names PCNs

Ouput: Pool

ni.Sorter = TPM

ni.PCers = CreatePCers()

Function CreatePCers():

foreach PN in PNs do

Idx <- get Index of PN in PNs

PCer.Pool.Name = PN

PCer.Filter = Supported\_PC\_FILTER\_FUNC[PFNs[Idx]]

PCer.Selector = Supported\_PC\_SELECT\_FUNC[PSNs[Idx]]

PCer.Orderer = Supported\_PC\_ORDER\_FUNC[PONs[Idx]]

PCer.MidProcesser = Supported\_PC\_MIDPROC\_FUNC[PMNs[Idx]]

PCers[PN] = PCer

return PCers

// Phase 2 and 3: Order and Execution

Input: Txs\_arrive

Output: Block

foreach Tx in Txs\_arrive parallel do

PCer = ni.PCers[ni.Sorter[Tx.Type]]

if PCer.Filter(Tx) then

Append Tx to PCer.Pool.PendingTxs

// Tx appending and block generating are also parallel

foreach PCer in ni.PCers parallel do

**Upon** PCer’s block generation condition is triggered **do** // START strategy, may be waiting timeout or pool is overflow

// programmable consensus modules are inserted in consensus protocol flow

Txs\_selected = PCer.Selector(PCer.Pool.PendingTxs)

Txs\_ordered = PCer.Orderer(Txs\_selected)

Txs\_processed = PCer.MidProcesser(Txs\_ordered)

Blockj.Txs = Txs\_processed

Calculate Blockj’s other variables but not set Blockj.PrevBlockHash

Blockj.BlockID = Hash(Blockj)

Append Blockj to ni.BlockPool

// Phase 4: Validation

// Input: current blockchain ni.BC, world state ni.**WS** based on current blockchain and a empty sandbox ni.SB which just can read from ni.WS

Upon ni.BlockPool is not empty

if isLeader(ni) == true then

B <- select a block from ni.BlockPool

B.PrevBlockHash = Hash(ni.BC.NewestBlock)

send <B.BlockID, B.PrevBlockHash, sig\_leader> to other nodes

Upon reception of <B.BlockID, B.PrevBlockHash, sig\_leader > do

B <- fetch block from ni.BlockPool which block’s BlockID is B.BlockID

if B.PrevBlockHash == Hash(ni.BC.NewestBlock) then

validate signatures which created in phase2 and phase3 by consensus nodes of B

foreach Tx in B.Txs do

if Tx.Executed == true then

B.TxsRes[IdxOfTx] = DetectConflict(Tx)

else

B.TxsRes[IdxOfTx] = true // Post-Process Tx wouldn’t conflict

Send <B.TxsResi, sig\_ni> to leader node

Upon reception of <B.TxsRes, sig\_nx> do

If isLeader(ni) == true then

Append sig\_nx to resCache[Hash(B.TxsRes)]

if len(resCache[Hash(B.TxsRes)]) meets condition then

send <B.TxsRes, resCache[Hash(B.TxsRes)], sig\_leader> to other nodes

Upon reception of <B.TxsRes, resCache[Hash(B.TxsRes)], sig\_leader> do

If resCache’s signatures true and len of them meets number condition then

Foreach Tx in B.Txs do

If B.TxsRes[IdxOfTx] == true && Tx.Executed == true then

CommitWriteSetToWorldState(Tx.WS)

Foreach Tx in B.Txs do

If B.TxsRes[IdxOfTx] == true && Tx.Executed == false then

PostProcess(Tx)

Function DetectConflict(Tx):

foreach r in Tx.RS do

if ni.SB[r.key] is empty then

ni.SB[r.key] = ni.WS[r.key]

if ni.SB[r.key] == r.value then

Return false

foreach w in Tx.WS do

ni.SB[w.key] = w.value

return true

Function PostProcess(Tx):

{ Tx.RS, Tx.WS} <- execute Tx based on sandbox which can just read from ni.WS

CommitWriteSetToWorldState(Tx.WS)

Function CommitWriteSetToWorldState(writeSet):

Foreach w in writeSet then

Ni.WS[w.key] = w.value