Consensus Algorithms in Wireless Blockchain System

1 Consensus Algorithm in Each Round

Algorithm 1 The SWIB Protocol

```
Input: List of consensus nodes $\{1, 2, \cdots, N\}$ with active time $\{T_1, T_2, \cdots, T_N\}$; Transactions $Txs = \{tx_1, tx_2, \cdots, \}$; Target contention success probability $\varsigma$; Target transmission success probability $\xi$; channel compete probability $p$ Transmission parameters $\{P_t, \alpha, \beta\}$
Output: Blockchain $BC$
//** Achieving consensus on block round-by-round,$r$ round
▷ Initialization:Slots 1:
Get $N$ nodes sorted based on public key value;
Compute elected probability of all nodes according to stability;
Compute required contention time slots $x_{comp}$;
Compute required transmission time slots $x_{trans}$
▷ Block Proposer Election:Slots 2:
$Rdm^r = GenerateRandomValue(r, B_H^{r-1}, sig_{full}^{r-1})$
$ID_{BP}$ = Block Proposer Election$(NodeList, Rdm^r)$
▷ Block Generation:
for $x_{comp} + T_B \cdot x_{trans}$ Slots do
    if $BP_{ID} == Node_{ID}$ then
        $B^r$ = Generate Block$(B_H^{r-1}, Txs)$
        Broadcast the block $B^r$ to other nodes with transmit power $P_t$
    else
        Listen on the channel to receive the block
    end if
end for
▷ Block Verification
for $N \cdot x_{comp} + T_{sig} \cdot x_{trans}$ Slots do
    for Any node in $\{1, \cdots, N\}$ do
        if $Node_{ID}! = BP_{ID}$ then
            if $isLegal(BPNode_{ID})$ and $isValid(B^r)$ then
                $sig^r$ = Generate Signature$(B^r, sk)$
                Broadcast $sig^r$ to other nodes
            else
                Discard the new block and generate an empty block $B_{empty}^r$
                $sig^r$ = Generate Signature$(B_{empty}^r, sk)$
                Broadcast $sig^r$ to other nodes
            end if
        end if
        if received enough partial signatures $Count(Sigs^r) > threshold$ then
            $sig_{full}^r$ = Recover Full Signature$(Sigs^r)$
            Broadcast $sig^r$ to other nodes
        end if
    end for
    //**Broadcast signatures
    Listen on the channel
    if Node $v$ decides to send a signature with transmit probability $p_v$ then
        $Broadcast(sig_v^r)$ with transmit power $P_t$
    else
        if channel is idle then
            $e_v = e_v + 1$ //** Count idle slots within $T_v$
        else
            Receive a message from others
        end if
    end if
    //**maintain the estimate of adversary time window
    for Any nodes $v \in \{1, \cdots, N\}$ do
        $count_v = count_v + 1$
        if $count_v > T_v$ then
            $count_v = 1$
            if $e_v == 0$ then //** No idle round in the past $T_v$ slots
                $\hat{p}_v = p_v/(1 + \frac{1}{T_v})$
                $\hat{T}_v = \hat{T}_v + 2$
            else if $e_v >= 1$ then
                $\hat{p}_v = \hat{p}_v * (1 + \frac{e_v}{T_v})$
                $\hat{T}_v = \hat{T}_v - 1$
            end if
        end if
    end for
end for
▷ Block FinalizationSlots 2:
for Any node $\{1, \cdots, N\}$ do
    if r theneceived or generated $sig^r_{full}$
        $AddSig(B^r, sig^r_{full})$
        $Append(BC, B^r)$
    end if
end for
```

Algorithm 2 The SWIB Protocol

```
Input: List of consensus nodes $\{1, 2, \cdots, N\}$ with active time $\{T_1, T_2, \cdots, T_N\}$; Transactions $Txs$; Target contention success probability $\varsigma$; Target transmission success probability $\xi$
Output: Blockchain $BC$
//** Achieving consensus on block round-by-round,$r$ round
▷ Initialization:
Compute elected probability of all nodes according to stability;
Get sorted node list based on public key value;
$Finalized = false$
▷ Block Proposer Election:Slots 2:
$Rdm^r = GenerateRandomValue(r, B_H^{r-1}, sig_{full}^{r-1})$
$ID_{BP} =$ Block Proposer Election$(NodeList, Rdm^r)$
▷ Block Generation:
for $x_{comp} + T_B \cdot x_{trans}$ Slots do
    if $BP_{ID} == Node_{ID}$ then
        $B^r =$ Generate Block$(B_H^{r-1}, Txs)$
        Broadcast the block $B^r$ to other nodes with transmit power $P_t$
    else
        Listen on the channel to receive the block
    end if
end for
▷ Block Verification
for $N \cdot x_{comp} + T_{sig} \cdot x_{trans}$ Slots do
    for Any node in $\{1, \cdots, N\}$ do
        if $Node_{ID}! = BP_{ID}$ then
            if $isLegal(BPNode_{ID})$ and $isValid(B^r)$ then
                $sig^r =$ Generate Signature$(B^r, sk)$
                Broadcast $sig^r$ to other nodes with transmit probability and transmit power $P_t$
            else
                Discard the new block and generate an empty block $B_{empty}^r$
                $sig^r =$ Generate Signature$(B_{empty}^r, sk)$
                run broadcsatSignature
            end if
        end if
        if collect enough partial signatures $Count(Sigs^r) > threshold$ then
            $sig_{full}^r =$ Recover Full Signature$(Sigs^r)$
            run broadcsatSignature
        end if
    end for
end for
▷ Block Finalization:
for Any node $\{1, \cdots, N\}$ do
    if received or generated the $sig_{full}^r$ then
        $AddSig(B^r, sig_{full}^r)$
        $Append(BC, B^r)$
        $Update(NodeList)$
        $Finalized = True$
    end if
end for
```

Algorithm 3 Synchronization Mechanism

```
Input: Latest blockchain length $Length$ Neighbor list $Neighbors$
Output: blockchain
for node $v$ in Neighbors do
    if $Len(BC_v) == Length$ and $Stability_v > threshold$ then
        add the node to RequestNeighbors
    end if
end for
for node $i$ in the Requested Neighbors do
    Request $m$ blocks from node
    if all blocks are valid then
        Add missing block to local blockchain
    else
        request these blocks from other nodes in RequestNeighbors
    end if
end for
```

Algorithm 2 broadcsatSignature Subroutine

```
for Any node $v$ in $\{1, \cdots, N\}$ do
    if Node $v$ decides to send a signature with transmit probability $p_v$ then
        $Broadcast(sig_v^r)$ with transmit power $P_t$
    else
        if channel is idle then
            $e_v = e_v + 1$ //** Count idle slots within $T_v$
        else
            Receive a message from others
        end if
    end if
    //**maintain the estimate of adversary time window
    $count_v = count_v + 1$
    if $count_v > \hat{T}_v$ then
        $count_v = 1$
        if $e_v == 0$ then //** No idle round in the past $\hat{T}_v$ slots
            $\hat{p}_v = p_v/(1 + \frac{1}{T_v})$
            $\hat{T}_v = \hat{T}_v + 2$
        else if $e_v >= 1$ then
            $\hat{p}_v = \hat{p}_v * (1 + \frac{e_v}{T_v})$
            $\hat{T}_v = \hat{T}_v - 1$
        end if
    end if
end for
```

Algorithm 3 The SWIB Blockchain Consensus Protocol for each node v

```
while true do
    //** Iteration for round $r$
    ▷ Initialization:
    for $j < K$ slots do
        $BroadcastMSG()$
        $j = j + 1$
    end for
    $Rds^r = GenerateRandomValue(r, B_H^{r-1}, sig_{full}^{r-1})$
    ▷ Consensus Process:
    Block Proposer Election();
    Block Verification();
    Block Finalization();
    $r = r + 1$
end while
```

Algorithm 4 Block Verification for each node v

```
$B^r, proof = RcvMSG()$
//**Check the validation of new block
$result_v$ = Verify Block Proposer($pk_{BP}, proof, Rdm^r$)
if $result_v == True$ then
    if $H^r_{pre} == B^{r-1}_H$ then
        if isvalid(Txs) then
            $sig^r_v$ = Generate Signature($B^r_H, sk_v$)
        end if
    end if
end if
```

Algorithm 5 Block Finalization for each node v

```
while $!finalized$ do
    $BroadcastMSG()$
    $sig^r_u, sig^r_{full} = RcvMSG()$
    //**Check the Finalization of new block
    if $isValid(sig^r_{full})$ then
        $AddSig(B^r, sig^r_{full})$
        $Append(BC, B^r)$
        $finalized = True$
    else if $Count(Sigs^r) \geq \lceil \frac{N+1}{2} \rceil$ then
        $sig^r_{full}$ = Recover Full Signature($Sigs^r$)
        $broadcast(sig^r_{full})$ with probability $p_{max}$ and power $P_{max}$
        $AddSig(B^r, sig^r_{full})$
        $Append(BC, B^r)$
        $finalized = True$
    else if $sig^r_u \notin Signs^r$ then
        Append Signature($Sigs^r, sig^r_u$)
    end if
end while
```

Algorithm 6 Stable Wireless Blockchain Protocol

```
▷ Initialization:
$Sortition(PKs^r, S^r)$
$Rds^r = GenerateRandomness(r, B^{r-1}_{hash}, sig^r_{final})$
▷ Leader Election and Block Proposal:
$result = BlockProposerSelection(sk, Rds^r)$
if $result == True$ then                                    ▷ As Block Proposer
    $B^r = GenerateBlock(B^{r-1}, Txs)$
    $sig^r_{partial} = Sign(B^r_{hash})$
    $broadcast(B^r, sig^r_{partial})$ with probability $p$
else                                                        ▷ As Ordinary Nodes
    Waiting to receive new Block
end if
▷ Block Verification and Finalization:
while $!finalized$ do                                       ▷ All Consensus Nodes
    $(B^r, Signs^r, sig^r_{full}, Tx) = RcvMSG()$
    //**Check the validation of new block
    if $isValid(B^r)$ and $VerifyBlockProposer(pk_{BP}, Rds^r)$ then
        $sig^r_v = GenerateSignature(B^r_{hash}, sk_v)$
    end if
    if $isValid(sig^r_{full})$ then
        $\sigma^r_F = sig^r_{full}$
        $broadcast(\sigma^r_F)$ with probability $p$
        $Append(B^r, \sigma^r_F)$
        $finalized = True$
    else if $Count(Signs^r) >= \lceil \frac{N}{2} \rceil$ then
        $\sigma^r_F = RecoverFullSignature(Signs^r)$
        $broadcast(\sigma^r_F)$ with probability $p$
        $Append(B^r, \sigma^r_F)$
        $finalized = True$
    else if $sig^r_u \notin Signs^r$ then
        $Signs^r = AppendSignature(sig^r_u)$
    else if $v$ did not broadcast its partial signature then
        $broadcast(sig^r_v)$ with probability $p$
    else
        $broadcast(Tx)$ with probability $p$
    end if
    $count = count + 1$
    if $count > T$ then
        $count = 1$
        if Received $T$ consecutive transactions in the past $T$ rounds then
            $p = p * (1 + \delta)^{-1}$
            $T = T + 2$
        end if
    end if
end while
```

```
function RecNewBlock($m_B, \sigma_v$)
    if $\sigma_v \notin sigShares$ then
        $sigShares = AppendSignature(\sigma_v)$
    end if
    if $Count(sigShares) > K$ then
        $FinalSig = RecoverFinalSig(sigShares)$
    else
        $FinalSig = null$
    end if
    return $sigShares, FinalSig, B_v^{new}$
end function
function AppendSignature($\sigma_v$)
    if $\sigma_v \notin sigShares$ then
        $sigShares \leftarrow sigShares + \sigma_v)$
    end if
    return $sigShares$
end function
```