1 Simulation Algorithm

Following algorithm describes the discrete-event simulation loop. Each tick processes events in a fixed order to ensure deterministic execution.

Algorithm 1 Gossipsub Simulation Main Loop

Require: Network parameters (n, D, D_{lazy}, c), simulation length T, warmup T_w

Ensure: Metrics $(\bar{\delta}, \beta_{norm}, p99)$

Initialize n peers with peer tables of size $[40, 80]$
Initialize eager mesh $\mathcal{M}_p$ and gossip peers $\mathcal{L}_p$ for each peer p
$t \leftarrow 0$
while $t < T_w + T$ **do**
    RefillTokenBuckets ▷ Bandwidth accounting
    ProduceMessages(t) ▷ Poisson arrivals
    ProcessDeliveries(t) ▷ Scheduled network arrivals
    ProcessChurn(t) ▷ Bimodal leave/rejoin
    EnforceBandwidth(t) ▷ Drop excess messages
    if $t \mod 10 = 0$ **then** ▷ Heartbeat every 1 second
        EmitGossip(t) ▷ IHAVE announcements
        MaintainMesh(t) ▷ GRAFT/PRUNE
    end if
    $t \leftarrow t + 1$
    if $t = T_w$ **then**
        Reset all counters ▷ End of warmup
    end if
end while
return ComputeMetrics

Algorithm 2 Token Bucket Refill

procedure RefillTokenBuckets
    for all peer p **do**
        $p.tokens \leftarrow \min\big(B/8,\ p.tokens + B \cdot \Delta t/8\big)$
    end for
end procedure

Algorithm 3 Message Production

procedure ProduceMessages(t)
  for all online peer p **do**
    $\lambda \leftarrow v_p/384$ ▷ v_p validators, one attestation per epoch
    $p_{pub} \leftarrow 1 - e^{-\lambda \cdot \Delta t}$ ▷ Poisson probability
    if $\mathit{Uniform}(0,1) < p_{pub}$ **then**
      $m \leftarrow$ new message with $\mathit{origin} = p$, $t_m = t$
      $p.\mathit{seen} \leftarrow p.\mathit{seen} \cup \{m\}$
      $p.\mathit{mcache} \leftarrow p.\mathit{mcache} \cup \{m\}$
      $p.\mathit{delivered} \leftarrow p.\mathit{delivered} \cup \{m\}$
      for all $q \in p.\mathcal{M}$ **do** ▷ Eager forward to mesh
        if $\omega_q = \mathsf{online}$ **then**
          Schedule delivery of m from p to q at tick $t + \textsc{Delay}(p,q)$
        end if
      end for
    end if
  end for
end procedure

Algorithm 4 Process Scheduled Deliveries

procedure ProcessDeliveries(t)
  for all (m, sender) scheduled for delivery to peer p at tick t **do**
    if $\omega_p = \mathsf{offline}$ **then**
      continue ▷ Drop: peer offline
    end if
    if $m \in p.\mathit{seen}$ **then**
      continue ▷ Duplicate: already processed
    end if
    $p.\mathit{seen} \leftarrow p.\mathit{seen} \cup \{m\}$
    $p.\mathit{mcache} \leftarrow p.\mathit{mcache} \cup \{m\}$
    $p.\mathit{delivered} \leftarrow p.\mathit{delivered} \cup \{m\}$ ▷ Persistent record
    Record $t_{m,p} \leftarrow t$ ▷ For latency measurement
    for all $q \in p.\mathcal{M} \setminus \{\mathit{sender}\}$ **do** ▷ Eager forward to mesh
      if $\omega_q = \mathsf{online}$ **then**
        Schedule delivery of m from p to q at tick $t + \textsc{Delay}(p,q)$
      end if
    end for
  end for
end procedure

Algorithm 5 Network Delay Model

function DELAY(p, q)
  $r_p, r_q \leftarrow$ regions of peers p and q
  $(\mu, \sigma) \leftarrow \mathit{LatencyTable}[r_p, r_q]$ ▷ See Table **??**
  $d \leftarrow \max(0.01, \mathcal{N}(\mu, \sigma^2))$ ▷ Sample, clip to 10ms minimum
  return $\lceil d/\Delta t \rceil$ ▷ Convert seconds to ticks
end function

Algorithm 6 Bimodal Churn Model

procedure PROCESSCHURN(t)
  for all peer p **do**
    if $p.\mathit{type} = \mathsf{churny}$ **then**
      $\lambda_{leave} \leftarrow 0.01, \quad \lambda_{rejoin} \leftarrow 0.05$
    else
      $\lambda_{leave} \leftarrow 0, \quad \lambda_{rejoin} \leftarrow 0$ ▷ Stable peers never leave
    end if
    if $\omega_p = \mathsf{online}$ **and** $\mathit{Uniform}(0,1) < \lambda_{leave}$ **then**
      $\omega_p \leftarrow \mathsf{offline}$
      Clear $p.\mathcal{M}$, $p.\mathcal{L}$ ▷ Mesh connections lost
    else if $\omega_p = \mathsf{offline}$ **and** $\mathit{Uniform}(0,1) < \lambda_{rejoin}$ **then**
      $\omega_p \leftarrow \mathsf{online}$
      Clear $p.\mathit{seen}$, $p.\mathit{mcache}$ ▷ State reset on rejoin
    end if
  end for
end procedure

Algorithm 7 Bandwidth Enforcement

procedure ENFORCEBANDWIDTH(t)
  for all message m from sender s scheduled for delivery at tick $t+1$ **do**
    $\mathit{size} \leftarrow |m|$ ▷ Payload or control message size
    if $s.\mathit{tokens} < \mathit{size}$ **then**
      Remove m from schedule ▷ Drop: insufficient bandwidth
    else
      $s.\mathit{tokens} \leftarrow s.\mathit{tokens} - \mathit{size}$
      if m is payload **then**
        $s.B^{\text{Payload}} \leftarrow s.B^{\text{Payload}} + \mathit{size}$
      else
        $s.B^{\text{Ctrl}} \leftarrow s.B^{\text{Ctrl}} + \mathit{size}$
      end if
    end if
  end for
end procedure

Algorithm 8 Gossip Emission (IHAVE)

procedure EmitGossip(t)
  for all online peer p **do**
    $ids \leftarrow$ message IDs in $p.mcache$ from last 3 heartbeats
    if $ids = \emptyset$ **then**
      continue
    end if
    $k \leftarrow \min(D_{lazy}, |p.\mathcal{L}|)$
    $targets \leftarrow$ sample k peers uniformly from $p.\mathcal{L}$
    for all $q \in targets$ **do**
      if $\omega_q = \mathsf{online}$ **then**
        Schedule IHAVE(ids) from p to q at tick $t + \textsc{Delay}(p, q)$
      end if
    end for
    Advance $p.mcache$ window ▷ Expire old entries
  end for
end procedure

Algorithm 9 Process IHAVE/ IWANT

procedure ProcessIHAVE(p, ids, $sender$)
  $missing \leftarrow ids \setminus p.seen$
  if $missing \neq \emptyset$ **then**
    Schedule IWANT($missing$) from p to $sender$ at tick $t + \textsc{Delay}(p, sender)$
  end if
end procedure
procedure ProcessIWANT(p, ids, $requester$)
  for all $mid \in ids$ **do**
    if $mid \in p.mcache$ **then**
      $m \leftarrow p.mcache[mid]$
      Schedule delivery of m from p to $requester$ at tick $t + \textsc{Delay}(p, requester)$
    end if
  end for
end procedure

Algorithm 10 Mesh Maintenance (GRAFT/ PRUNE)

procedure MAINTAINMESH(t)
  for all online peer p **do**
    $p.\mathcal{M} \leftarrow \{q \in p.\mathcal{M} : \omega_q = \mathsf{online}\}$ ▷ Remove offline peers
    if $|p.\mathcal{M}| < D - 2$ **then** ▷ Below D_{lo}: graft
      $k \leftarrow D - |p.\mathcal{M}|$
      $\mathit{candidates} \leftarrow \{q \in p.\mathit{table} \setminus p.\mathcal{M} : \omega_q = \mathsf{online}\}$
      $\mathit{new} \leftarrow$ sample $\min(k, |\mathit{candidates}|)$ peers from $\mathit{candidates}$
      for all $q \in \mathit{new}$ **do**
        $p.\mathcal{M} \leftarrow p.\mathcal{M} \cup \{q\}$
        Schedule GRAFT(p) to q at tick $t + \text{DELAY}(p, q)$
      end for
    else if $|p.\mathcal{M}| > D + 2$ **then** ▷ Above D_{hi}: prune
      $k \leftarrow |p.\mathcal{M}| - D$
      $\mathit{excess} \leftarrow$ sample k peers from $p.\mathcal{M}$
      for all $q \in \mathit{excess}$ **do**
        $p.\mathcal{M} \leftarrow p.\mathcal{M} \setminus \{q\}$
        Schedule PRUNE(p) to q at tick $t + \text{DELAY}(p, q)$
      end for
    end if
    ▷ Update lazy mesh from remaining non-eager peers
    $p.\mathcal{L} \leftarrow$ sample D_{lazy} peers from $\{q \in p.\mathit{table} \setminus p.\mathcal{M} : \omega_q = \mathsf{online}\}$
  end for
end procedure

Algorithm 11 Process GRAFT/ PRUNE

procedure PROCESSGRAFT(p, $\mathit{requester}$)
  if $\mathit{requester} \in p.\mathit{table}$ **and** $\omega_{requester} = \mathsf{online}$ **then**
    $p.\mathcal{M} \leftarrow p.\mathcal{M} \cup \{\mathit{requester}\}$
    $p.\mathcal{L} \leftarrow p.\mathcal{L} \setminus \{\mathit{requester}\}$
  end if
end procedure
procedure PROCESSPRUNE(p, $\mathit{requester}$)
  $p.\mathcal{M} \leftarrow p.\mathcal{M} \setminus \{\mathit{requester}\}$
end procedure

Algorithm 12 Compute Final Metrics

function ComputeMetrics
  ▷ Delivery set for message m: peers that received it
  $R_m \leftarrow \{p \in V : m \in p.delivered\}$
  ▷ Delivery rate: fraction of all peers that received each message
  $\bar{\delta} \leftarrow \frac{1}{|M|} \sum_{m \in M} \frac{|R_m|}{|V|}$
  ▷ Total bandwidth
  $B_{total} \leftarrow \sum_{p \in V} \left(p.B^{\text{Payload}} + p.B^{\text{Ctrl}}\right)$
  ▷ Bytes per delivery
  $deliveries \leftarrow \sum_{m \in M} |R_m|$
  $\beta \leftarrow B_{total}/deliveries$
  ▷ Normalized cost
  $\beta_{norm} \leftarrow \beta/P$ ▷ $P = 1536$ bytes
  ▷ p99 latency over all successful deliveries
  $L \leftarrow \{(t_{m,p} - t_m) \cdot \Delta t : m \in M,\, p \in R_m\}$
  $p99 \leftarrow Percentile_{99}(L)$
  return $(\bar{\delta}, \beta_{norm}, p99)$
end function

2 Pareto Dominance Analysis

> **Definition 2.1 (Pareto Dominance)** *Configuration A* dominates *configuration B on delivery rate and bandwidth cost if $\delta_A \geq \delta_B$ and $(\beta/P)_A \leq (\beta/P)_B$, with at least one strict inequality.*

We exclude latency from the dominance computation because Floodsub achieves the lowest latency (0.4 s p99). Including latency would prevent any configuration from strictly dominating Floodsub, obscuring the clear delivery-cost advantage of Gossipsub. Latency is instead encoded as marker shape in Figure **??** for visual inspection.

For the dominance plot, we aggregate simulation results to worst-case metrics across churn regimes using Algorithm 13.

Algorithm 13 Worst-Case Aggregation Across Churn Regimes

Require: Raw results $\mathcal{R} = \{(D, D_{lazy}, c, \bar{\delta}, \beta_{norm}, p99)\}$ for all configurations
Ensure: Aggregated results $\mathcal{A}$

$\mathcal{A} \leftarrow \emptyset$
for all unique (D, D_{lazy}) pairs in $\mathcal{R}$ **do**
  $\mathcal{R}_{D,D_{lazy}} \leftarrow \{r \in \mathcal{R} : r.D = D \wedge r.D_{lazy} = D_{lazy}\}$
  $\delta^* \leftarrow \min_{r \in \mathcal{R}_{D,D_{lazy}}} r.\bar{\delta}$ ▷ Worst (minimum) delivery
  $\beta^*_{norm} \leftarrow \max_{r \in \mathcal{R}_{D,D_{lazy}}} r.\beta_{norm}$ ▷ Worst (maximum) cost
  $\ell^* \leftarrow \max_{r \in \mathcal{R}_{D,D_{lazy}}} r.p99$ ▷ Worst (maximum) latency
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{(D, D_{lazy}, \delta^*, \beta^*_{norm}, \ell^*)\}$
end for
return $\mathcal{A}$

Algorithm 14 computes the 2D Pareto frontier over delivery rate and cost. Latency (ℓ^*) is retained in the aggregated results for visualization but is not used in the dominance check.

Algorithm 14 Compute 2D Pareto Frontier

Require: Aggregated results $\mathcal{A} = \{(D, D_{lazy}, \delta^*, \beta^*_{norm}, \ell^*)\}$
Ensure: Pareto-optimal set $\mathcal{P} \subseteq \mathcal{A}$

function DOMINATES(A, B) ▷ Does A dominate B on delivery rate and cost?
  return $(\delta^*_A \geq \delta^*_B) \wedge (\beta^*_{norm,A} \leq \beta^*_{norm,B})$
   $\wedge \big((\delta^*_A > \delta^*_B) \vee (\beta^*_{norm,A} < \beta^*_{norm,B})\big)$
end function
$\mathcal{P} \leftarrow \mathcal{A}$
for all $A \in \mathcal{A}$ **do**
  for all $B \in \mathcal{A} \setminus \{A\}$ **do**
    if DOMINATES(B, A) **then**
      $\mathcal{P} \leftarrow \mathcal{P} \setminus \{A\}$
      break
    end if
  end for
end for
return $\mathcal{P}$