Cooperative OTP Ring Simulation

Nithesh Gurudaas Kujuluva Ganesh, Raffael Davila, Andy Then

February 17, 2026

1 Overview

This document describes a simulation of a cooperative one-time pad (OTP) ring protocol operating over an asynchronous network. The simulation models m parties arranged in a logical ring, each maintaining a pointer into a shared pad of length n, subject to a minimum gap constraint d enforced between adjacent parties. The key question studied is: how many pads are wasted (i.e. left unused) when only x out of m parties are actively sending?

2 System Components

2.1 Asynchronous Network

The `AsynchronousNetwork` class simulates a broadcast medium with variable delivery delay. Each broadcast from a sender arrives at all other nodes within a uniformly random delay of $[0, d_{\text{delay}}]$ ticks. The global clock advances by one unit per call to `tick`, which delivers all messages whose scheduled arrival time has been reached and updates the recipient parties' views accordingly.

Formally, when party i broadcasts its new index idx at time t, each recipient $j \neq i$ receives the update at time:

$$t_{\text{deliver}} = t + \delta, \quad \delta \sim \mathcal{U}\{0, d_{\text{delay}}\}.$$

2.2 Ring Party

Each `RingParty` represents a node in the ring with the following state:

- `my_index`: the party's current position in the pad array, initialised to $(\texttt{party\_id} - 1) \cdot \lfloor n/m \rfloor$.
- `view_of_others`: a dictionary mapping each party ID to its last known index, initialised to the same evenly-spaced starting positions.
- `pads_used`: a counter of pad indices this party has consumed for encryption.

The gap constraint requires that a party may only advance if the distance to its forward neighbor (modulo n) exceeds d:

$$\texttt{gap} = (\texttt{neighbor\_pos} - \texttt{my\_index}) \bmod n > d.$$

3 Simulation Protocol

3.1 Initialisation

Given parameters n, m, d, and x, the simulation proceeds as follows:

- x parties are designated *active* (chosen uniformly at random); the remaining $m - x$ are *silent*.
- A *burned* set tracks pad indices that have been consumed. It is initialised with the starting positions of all active parties.
- The theoretical maximum utilisation is $n - m \cdot d$, accounting for the mandatory gaps between all m parties.

3.2 Move Classification

At each tick, the eligibility of each party to advance is determined by `get_move_status`, which returns one of three outcomes:

`data` The gap constraint is satisfied and the next index is fresh (not burned). The party encrypts a message, burns the pad, and increments `pads_used`.

`drift` The gap constraint is satisfied but the next index is already burned. The party advances without consuming a new pad (skipping over a used index).

`None` The gap constraint is violated; the party is blocked and cannot move.

3.3 Scheduling Policy

Each tick applies a two-priority scheduling rule:

1. **Active parties have priority.** Any active party that is not blocked may move. One such party is chosen uniformly at random, advances its index, and broadcasts its new position.
2. **Silent parties move only when no active party can.** If all active parties are blocked, a random unblocked silent party performs a *yield* move: it advances without burning and broadcasts, thereby clearing space for active neighbours.

3.4 Termination

The simulation terminates when either:

- the burned set reaches the maximum utilisation $n - m \cdot d$, or
- no party can move *and* the network message queue is empty (*Clinch* state).

The function returns $n - |\texttt{burned}|$, the count of unused pad indices.

4 Algorithm

5 Experimental Setup

The simulation is run for $M \in \{3, 4\}$ parties with fixed parameters $N = 2000$ and $D = 15$. For each value of M, every scenario $S.x$ (for $x = 1, \ldots, M$) is repeated over 50 independent trials. The reported metrics are:

- **Average wasted pads**: $\mathbb{E}[n - |\texttt{burned}|]$ across trials.
- **Utilisation**: $\left(1 - \frac{\mathbb{E}[\text{wasted}]}{n}\right) \times 100\%$.

Scenario $S.x$ denotes the case where exactly x out of M parties are active senders; the remaining $M - x$ parties are silent and only perform yield moves to vacate space.

Algorithm 1 Cooperative OTP Ring Simulation

```
Input: pad length n, party count m, gap bound d, active count x
Output: number of unused pad indices
Initialise Network with delay d
Sample active set A ⊆ {1, . . . , m}, |A| = x; silent set S = {1, . . . , m} \ A
Initialise parties: pos[i] ← (i − 1)⌊n/m⌋ for all i
burned ← {pos[i] : i ∈ A}
MAX ← n − m · d
while |burned| < MAX do
    Network.Tick(parties)                        ▷ Deliver pending messages, update views
    movers ← {i ∈ A : GapOK(i)}
    if movers ≠ ∅ then
        i ← RandomChoice(movers)
        nxt ← (pos[i] + 1) mod n
        pos[i] ← nxt
        if nxt ∉ burned then
            burned ← burned ∪ {nxt}              ▷ data move
        else
                                                 ▷ drift move; pad already used
        end if
        Network.Broadcast(i, nxt)
    else
        jumpers ← {j ∈ S : GapOK(j)}
        if jumpers ≠ ∅ then
            j ← RandomChoice(jumpers)
            pos[j] ← (pos[j] + 1) mod n
            Network.Broadcast(j, pos[j])         ▷ yield move
        else
            if Network.QueueEmpty() then
                break                            ▷ Clinch state reached
            end if
        end if
    end if
end while
return n − |burned|
procedure GapOK(i)
    fwd ← (i mod m) + 1
    return (view[i][fwd] − pos[i]) mod n > d
end procedure
```

6 Security Property

The simulation includes a *loud-fail* check: if an active party attempts a **data** move onto an index already present in `burned`, a RuntimeError is raised immediately. This guarantees that pad reuse is detectable at simulation time.