F25 CS240 Project Proposal

1. Basic Information

- **Project Title:** Intergalactic Rumor Mill: Scalable Gossip with Probabilistic Liveness
- **Student Names:** Hassan Al Moalim, Lamar AlSubhi
- **Emails:** hassan.moalim@kaust.edu.sa, lamar.alsubhi@kaust.edu.sa
- **Type of Project:** Regular

2. Motivation and Project Goal

- **Explain the project goal within the scope of computing systems and concurrency.**

Implement and evaluate a gossip-based information-spreading service that distributes updates across many processes through randomized peer exchange. The system will demonstrate probabilistic liveness, where every correct node eventually receives each update, and will analyze how scalability and fault tolerance emerge from simple local rules augmented with anti-entropy reconciliation.

- **Why and what makes this project interesting?**

Gossip protocols are fundamental building blocks of large-scale distributed systems such as Cassandra, Dynamo, and Kubernetes. They enable efficient, fault-tolerant dissemination without centralized control and illustrate how local randomness can yield global reliability. Our prototype will expose the trade-offs between scalability and reliability in epidemic-style information propagation.

- **Which systems trait(s) does your project target:**

  - ☑ Reliability (e.g., fault tolerance, replication, recovery)
  - ☑ Scalability (e.g., horizontal scale, throughput, elasticity)

3. Project in Some Details

- **Describe what you plan to build or reproduce.**

A Go-based simulator of N processes that periodically exchange "rumors." Each process maintains a list of peers and a set of seen message identifiers. The base algorithm will be push–pull gossip with tunable fan-out f and rumor time-to-live T. The system includes periodic anti-entropy reconciliation to ensure eventual consistency; optional extensions may explore causal delivery via vector clocks.

- **Outline the core components, architecture, or workflow.**

    1. Each node maintains local state and a periodic gossip loop.
    2. At every interval Δ, a node selects f random peers and exchanges unseen message IDs.
    3. Missing messages are requested immediately; duplicates are ignored after T receives.
    4. Periodically, nodes perform anti-entropy rounds to compare digests and repair missing updates.
    5. The simulator records convergence time, message count, and reliability under packet loss and churn.

- **Indicate what technologies you expect to use (e.g., Go, Python, Kubernetes, gRPC, Spark, etc.).**

Go (goroutines & channels for concurrency), JSON logging for metrics, Matplotlib for plots.

- **List any benchmarks or datasets you plan to use.**

Synthetic workloads generated by random message injections across N nodes; no external datasets required.

4. Plan and Milestones (≈ 6 weeks)

Week	Milestone	Deliverable
1	Design simulator structure and message format; implement basic push gossip with reliable links	Working single-rumor prototype
2–3	Add push–pull and TTL; implement anti-entropy reconciliation; collect baseline convergence vs N and fan-out f	Preliminary plots + sanity results
4–5	Introduce loss and churn; evaluate scalability and reliability	Full system + evaluation runs
6	Final results; Report & presentation	Demonstration + Final report + Slides

5. Expected Outcomes

- **What will you measure or demonstrate?**

Convergence time to full dissemination vs number of nodes N.
Total messages sent vs fan-out f and TTL T.
Reliability (success fraction) under packet loss and node churn.

- **How will you show that your system meets its goals (e.g., latency vs throughput, scalability curve, fault-tolerance behavior, correctness under failure)?**

Scalability will be shown by convergence curves ≈ O(log N); reliability by high success rates under loss. Trade-off plots (f, T vs latency / traffic) will illustrate system tuning and fault tolerance.

6. Resources Needed

- **List any compute, software, or data resources you require.** Examples:

No special resources required; experiments will run on local machines.

7. If Reproduction Project, Paper Details

NA

8. Risk Assessment and Feasibility

- **Identify any foreseeable risks (time, complexity, missing resources) and how you plan to mitigate them.**

Potential risk: spending too much time on advanced features (e.g., causal ordering or visualization). Mitigation: complete the basic push–pull version and core metrics within two weeks, treat extensions as optional. All dependencies are standard Go packages; overall risk is low.

9. References / Topic Source

- **Cite relevant references you drew ideas from.**

Intergalactic Rumor Mill - Topic Suggestion

Demers et al., “Epidemic Algorithms for Replicated Database Maintenance,” PODC 1987

Das et al., “SWIM: Scalable Weakly-Consistent Infection-Style Membership Protocol,” DSN 2002