Using Peer-to-peer networks for mission critical data communication

Group #7
Satish Kotti
Siddhant Sonkar
Venkatesh SGS

Introduction

- Peer-to-peer based approach for data transmission in mission critical systems like disaster response systems.
- Properties:
  - Large number of recipients
  - Disseminate as fast as possible
  - Reach maximum number of nodes
  - Heterogeneous network
  - Fault Tolerance
- Applications:
  - Disaster Response Systems
  - Emergency Alert Systems

Related Work

- Various methodologies can be employed to solve the problem of flash dissemination.
- Conventional centralized client server setting
- Single point of failure
- Can be optimized for performance and improved latency

Reliable Multicast

- Application Layer Multicast
- Core of the algorithm is to build a spanning tree.
- Trade off Stretch for Stress.
- Various approaches that use this paradigm .
  - Scalable Application layer multicast
  - Farecast
  - Overcast
- Not suitable for high volume of topology changes
- Not as scalable as P2P
- Need dedicated infrastructure
- P2P is more reliable
- Not as fault tolerant as P2P
- P2P is more cost-effective

Tree-based Multicast

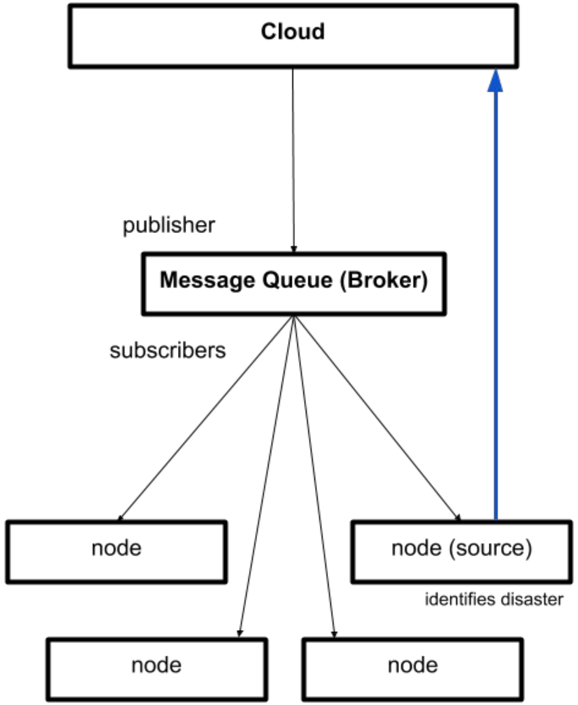
- Need information of network topology
- Constant changes to topology need to re-estimate the tree continuously
- Failure of nodes within the tree structures impacts the performance of the network

Peer to Peer approaches

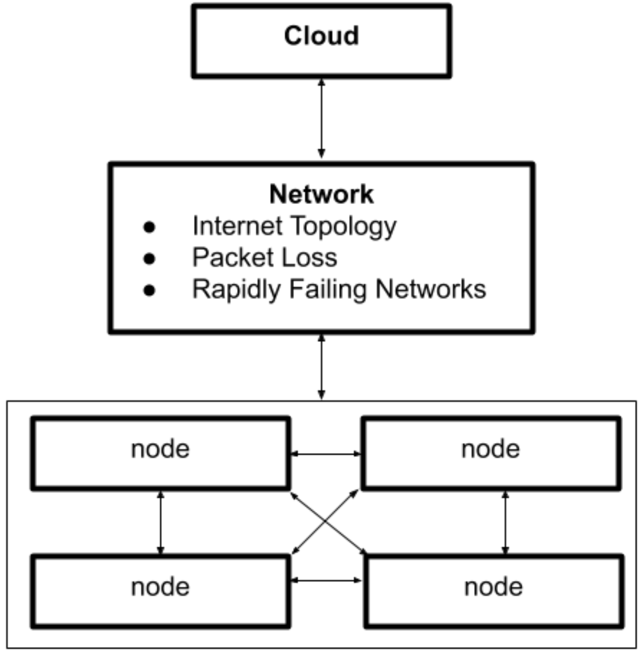
- Randomized approaches. E.g. Gossip, Random Walk etc.
- Decentralized algorithm
- Prefer redundancy and reliability over scalability.
- Under the assumption that our content is not very huge, gossip protocols are well suited for our use case.
- Flash dissemination scenarios are unpredictable and may contain heterogeneous networks, randomized approaches tackle this scenario better.
- Operate with local knowledge.
- Heuristics for obtaining global knowledge can help improve the performance and reduce dissemination time.

Project Simulation

- Centralized Architecture
- Gossip Network
- Random-walk based Gossip Network



Centralized Architecture



Gossip-based Architecture

Centralized Architecture

- Source node informs the cloud
- Cloud publishes the message to a message queue (ZeroMQ)
- All subscribers to the broadcasting topic receive information

Gossip Network

- Connection handshake with the bootstrapper node.
- Socket information exchanged. Used for subsequent message transmission.
- Connection details added to connection pool, happens asynchronously in different processes.
- All live nodes establish connection with bootstrapper node.
- A new connection at bootstrapper is informed about previous connections.
- Nodes use this data to identify peers and establish connection.
- Messages are broadcasted to all the members of connection pool.
- Every node receiving the message, broadcasts the message to the neighbours other than the source.

Random-walk Gossip Networks

- Messages are broadcasted to randomly chosen subset of connections.
- Helps with controlling flooding in the network
- Not suitable for small networks. Can cause starvation for few nodes.

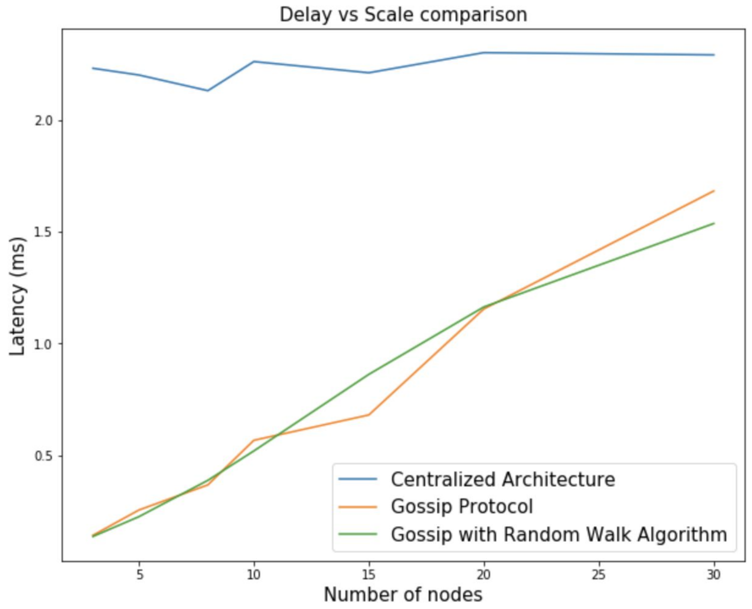
DEMO

Evaluation

- Number of nodes = { 3, 5, 8, 10, 15, 20, 30 }
- Used 4 laptops and 1 android device to simulate 5 nodes
- Repeated the experiment multiple times to generate a pool of average latency values
- Used these values for further simulation (delay time + code snippet run time)

Results

- Nearly same values for centralized architecture
- RPC consuming more time
- Sockets are quick
- Flooding in gossip started causing delay quickly
- Surprisingly, gossip and random-walk got nearly same results
- Scaling to 100 nodes can clearly signify importance of random-walk.



Future Work

- Varying Content Size
- Heterogeneous content - images, video, text
- Varying Network Bandwidth
- Modeling Network Topology
    - Rapid Packet Loss
    - Network Partitions
    - Failing nodes and links
    - High network churn rate

Supporting tools for future work

- Network Simulation
  - ModelNet
  - Planet Lab
  - NS-3
- Protocol Simulation
  - Cooja Contiki
  - CupCarbon

Challenges & learnings

- Challenges
  - Network Simulation at scale with tools
  - Realistic simulation - Google Cloud, Kubernetes
  - No good documentation for available implementations
- Learnings
  - Low level socket programming
  - Protocol implementation and simulation
  - Understanding necessity for multi-threaded implementations in such systems - maintaining mutex locks on connection pools, spawning a process for each activity, importance of asynchronous behaviour

Thank you!