An Introduction to Gossip Protocols

François Taïani
Traditional Distribution

- Typical distributed protocols:
  - do not rely on ‘chance’
  - provide deterministic guarantees

- But
  - either rely on a central core of servers
  - or not very scalable \((O(n^k), k=2,3)\)
Gossip Protocols

- Alternative: **probabilistic** approach

- Gossip (aka epidemic) Protocols
  - Introduce some ‘**chaos**’
  - Goal: system to **converge** to a desirable outcome
  - But some nodes might be left out

- Trading determinism for **scalability & robustness**
Example: Overlay Multicast

- **Deterministic** options
  - Flooding $\rightarrow$ highly inefficient
  - Spanning tree $\rightarrow$ very efficient, but complex, costly, brittle

- **Gossip** approach: for each new broadcast, each node
  - on first reception randomly pick $k$ targets in the rest of the system ($k = \text{‘fanout’}$), and forward msg to them
  - (discard duplicate reception)
Example 1: Multicast

- How to choose $k$ (fanout) to reach everyone?
Which k? Bimodal behaviour

(Kemarrec, Massoulie, Ganesh / Microsoft Research)

Flat gossip: Results in failure free execution for 10,000 node group [KMG03]

Threshold value: ~ 9.21 (= log(10000) )

Proportion of nodes reached in non atomic broadcast
Proportion of atomic broadcast
Gossip: Key Ingredients

State → Stochastic Data exchange → New State

- What **state** to maintain?
  - msg already seen (multi-cast)
  - sensor value (aggregation, averaging)
  - neighbours list (topology construction, membership)
  - user profile (social networks)

- **When** to gossip? (periodic vs. event based)

- With **which probability** and **with whom** to gossip?

- Which information to **exchange**?

- How to **compute** new state?
Ex. 2: Peer Sampling

- “Random Peer Sampling Service” [JGK04]
  ➔ Membership service: who is in the system?

- **State**: list of k neighbours (k = system param)

- Main idea: **Periodically**: each node n
  ➔ picks one neighbour i
  ➔ sends own list of neighbours, receives that of i
  ➔ i and n merge, shuffle, truncate the 2 lists
Random Peer Sampling (cont.)

Result $\rightarrow$ random graph

Highly resilient against churn, partition

(1) peer list exchange    (2) merge and truncation

Small diameter (good for multicast)
Ex. 3: Topology Construction

E.g. T-Man [JMB09]

- State: each node has a position (e.g. in $\mathbb{R}^3$)
- Metrics: Euclidian distance between node
- Goal: find $n$ closest neighbours
T-Man: Architecture

- Uses RPS service
- Periodically: each node $n$
  - merges RPS neighbours with clustering neighbours
  - sorts nodes in list according to distance
  - keeps $k$ closest neighbours
T-Man: Swap Mechanism

- To speed convergence each node \( n \)
  - picks one node \( i \) in clustering neighbours
  - sends own list of neighbours, receives that of \( i \)
  - \( i \) and \( n \) merge, sort, keep \( k \) closest neighbours

(1) peer list exchange
(2) merge and truncation
T-Man in Action

- (taken from [JMB09])

Result ➔ structured overlay

Highly resilient against churn, partition (RPS)

Fast convergence (Swap)

after 2 cycles

after 7 cycles
Not limited to Euclidian space

- E.g. with user profiles (here metric = overlap)

1 exchange of neighbors lists

2 neighborhood optimization
Other Uses of Gossip

- Self-organisation in WSN: Who should do what?
- Topology construction in fixed overlay: fastest routing
- Knowledge aggregation: e.g. average temperature
- Information dissemination (failure detection, AWS S3)

From [LGKVB06]
Application to Social Networks

- ASAP group, Rennes, Gossple ERC
  ➔ E.g. (geo)recommendations
Summing Up

- Gossip protocols
  - Conceptually simple
  - Potentially very rich
- Extremely well adapted to large-scale distribution
- Efficient, scalable, robust
- No deterministic guarantees (usually)
- Opportunity for Programmatic / Software Eng. angle
(Some) References: Protocols


(Some) References: Protocols


