Gossip Protocols

CSE 486/586: Distributed Systems

Ethan Blanton

Department of Computer Science and Engineering
University at Buffalo
Gossip

The multicast protocols we have looked at have common properties:

- Processes must know all other processes
- Message count of $O(|G|)$ for unreliable or for $O(|G|^2)$ reliable transmission
- Messages are either unreliable or always received

Gossip protocols can provide:

- Processes must know a small fraction of other processes
- Typically $O(|G| \log |G|)$ messages per multicast
- Messages are probabilistically received by all correct processes
Origins

Gossip protocols have their origins in epidemiology.\(^1\)

An epidemiology book [1] was noticed by computer scientists [2]. It describes epidemics as proceeding in rounds of infection.

In gossip protocols, as in epidemiology, a process is either:

- Susceptible to infection by a new message
- Infected by a new message and capable of retransmitting it
- Removed from the set of infected processes (and now “immune” to the message)

\(^1\)Seems apropos in 2021, doesn’t it?
Simple Multicast

$|G|$ processes, $|G|$ messages.

If a message is lost or the sender fails, messages are lost.
Reliable Multicast

\(|G|\) processes, \(|G|^2\) messages.

If any correct process receives the message, all correct processes receive the message.
Simple Gossip

Gossip proceeds in **rounds**.

A process decides that it wants to multicast a message \( m \).
Simple Gossip

It multicasts it to $k$ randomly selected processes.
Simple Gossip

If a process *hears* $m$ for the first time, it re-multicasts.

Each such process chooses $k$ randomly selected processes.
Simple Gossip

This repeats until no new process hears the message.

Some nodes may never hear the message!

The probability of this is exponentially decreasing in $k$ [2].
Benefits of Gossip

Far fewer than $O(|G|^2)$ messages even with $k \gg 1$. (Bounded above by $k \cdot |G|$.)

Only one process must hear the message to start an epidemic.

Every process receives every message with high probability.

Message loss and process failure are tolerated by raising $k$. 
Disadvantages of Gossip

Some processes may not receive a message even without failure.

Small groups require $k \approx |G|$ anyway.

Delay between first transmission and final infection can be large.
Lightweight Probabilistic Broadcast

Lightweight Probabilistic Broadcast [3] (lpbcast) uses gossip for:
- Message distribution
- Group membership

This allows:
- Large groups
- Dynamic membership
- Configurable reliability
- Low message traffic
LPBCast Actions

LPBCast uses publish-subscribe terminology.

In lpbcast, processes can:

- **Subscribe** to a topic (join a group)
- **Unsubscribe** from a topic (leave a group)
- **Send notifications** (messages) to a topic (group)

All of these actions are communicated via **one message type**.

Unlike simple gossip, messages are sent **on a heartbeat**.
Notifications

A notification in lpbcast is a message to be sent.

Every notification has an associated unique ID.

Processes keep track of:

- Recently-seen notifications in the variable events
- The identifiers of recently-seen notifications in eventIds

The rules for keeping track of these are different.
Subscriptions

Processes subscribed to the *lpbcast* topic are group members.

Processes keep track of:
- Recently subscribed processes in *subs*
- Recently unsubscribed processes in *unSubs*
- Exactly *l* processes believed to be subscribed in *view*
Messages in *lpbcast*

Each *lpbcast* process sends a message to *F* processes every *T* ms.

Every *lpbcast* message contains:
- A list of all new notifications since the last message.
- A list of some recent notifications
- A list of some recent subscriptions
- A list of some recent unsubscriptions

The total number of messages sent per *T* ms is exactly *F* ⋅ |*G*|.

Note that *F* is like the *k* from our previous gossip example!
Receiving Messages

Upon receiving a message, a lpbcast process will:

1. Update its view and unSubs from the recent unsubscriptions
2. Update its view and subs from the recent subscriptions
3. Remove random elements from subs and unSubs until they reach a configurable size
4. Remove random elements from view until $|\text{view}| \leq l$
5. Deliver any new notifications
6. Update its events and eventIds with the new notifications
7. Remember event IDs for unknown events from the message
8. Remove random elements from events and eventIds until they reach a configurable size
Probability and Reliability

Items are removed uniformly at random from each set: events, eventIds, subs, unSubs, view

The set sizes are configured taking into account:

- The expected number of subscribers
- The probability of process failures
- The probability of message loss

Note that:

- notifications are sent only once
- eventIds is pruned randomly
Subscriptions

To subscribe to the topic, a process must send a request to any subscribed process.

If it does not start receiving notifications, it tries again.

A subscribed process periodically gossips its subscription.

To unsubscribe from a topic, it gossips its unsubscription.

Failed processes are eventually forgotten.
Partitions

The group may become partitioned.

This is a condition where:

- There exist sets $G'$, $G'' \in G$ such that:
  - $\forall g \in G'$, $G \not\in G''$ and $\forall g \in G''$, $G \not\in G'$

Once this happens, $G'$ and $G''$ will remain disjoint.

$l$ is selected such that the probability of this is extremely low.

Some privileged processes can be kept by all processes to prevent partition.
Benefits of *lpbcast*

*LPBCast* adds **membership management** to simple gossip.

It also adds **reliability** through **events** and **eventIds**.

It uses a **relatively constant bandwidth** due to $T$.

Each process only has to know $l$ hosts regardless of $|G|$.

Reliability ($l$, other sizes), latency ($T$), and cost ($F$) are configurable.
Uses of Gossip

The first use of gossip was in distributed database updates.

It was later used for maintaining group membership.

Then, for general multicast as in lpbdcast.

It can be used for failure detection.

It has been used in sensor networks (“IoT”).
Choosing Gossip

Gossip is appropriate when:

- The occasional lost message can be tolerated
- Simple multicast is not reliable enough
- Reliable multicast is too expensive
- Group membership is unstable

Tuning gossip for the application is critical!

What is $|G|$? What should $k$ (l for $lpbcast$) be?
Gossip for Failure Detection

How might we use gossip for failure detection?

- Is it complete?
- Is it accurate?

What parameters are configurable?
Gossip protocols provide probabilistic delivery
Cost is usually about $c \cdot |G| \log |G|$ per message
Lightweight Probabilistic Broadcast solves:
  - Changing group membership
  - Process membership knowledge overhead for very large $|G|$
References I

Required Readings


Optional Readings

References II

Copyright 2021 Ethan Blanton, All Rights Reserved.

Reproduction of this material without written consent of the author is prohibited.

To retrieve a copy of this material, or related materials, see https://www.cse.buffalo.edu/~eblanton/.