CS5320 - Distributed Computing: Autumn 2018
Course Project

Ganesh Vernekar CS15BTECH11018
Nidhi Dhamnani CS15BTECH11028

Problem Statement

Implement push and pull based **Gossip protocols** and compare their message complexity. Also test each gossip protocol for robustness.

Introduction

A gossip protocol is process of computer based communication which is similar to how the epidemics are spread. It is used for communications. Gossip based solutions are used in distributed systems having extremely large network. In gossip algorithm, each node exchanges the information with random peer in it's neighbourhood.

Properties

- **Highly decentralised** - The processes/nodes can be located anywhere in the network.
- **Periodic Interactions** - Each process exchanges messages at fixed regular intervals.
- **Eventual Consistency** - Each process gets the messages from it's neighbour, hence the state of all the processes need not be always same but ultimately all the processes receive all the messages.
- **Fault tolerant** - Even if few processes fail, the algorithm works correctly because the the non-faulty nodes get information from other neighbours. Hence, the algorithm is quite robust.
    - In normal broadcast, if a single message transfer fails, then the broadcast fails.
- **No reliable interaction needed** - The communication channel is assumed to be non-reliable.
- **Redundancy** - A process may receive same message from multiple neighbours.

Applications

- Failure Detection.
- Monitoring.
- Messaging.
- Analysis of community structure.
- Analysis of spread of fake news.
- Used as a replacement for normal broadcast.
- Widely in distributed databases, bitcoins, etc.

Algorithm

Push based Gossip:

```
S: set of all messages received.

Receiving_Gossip(g):
# Here g contains set of messages sent by neighbour.
 1.   Add all messages from g to my set S.

Sending_Gossip():
 1.   At every 1 second interval ->
       a.   Pick random K neighbours n1..nk
       b.   Send S to n1..nk

Add_New_Gossip_Message(msg):
 1.   Add msg to S.
```

Note: For testing, we chose an interval of 1 second for sending gossip, but it is higher than 1 second in general.

Pull based Gossip:

```
S: set of all messages received.

Receiving_Gossip(g):
[] (g.type==ENQUIRE) ->
       1.   Send S to g.sender
[] (g.type==GOSSIP) ->
       1.   Add all messages from g to my set S.

Enquiring_Gossip():
 1.   At every 1 second interval ->
       a.   Pick random K neighbours n1..nk
       b.   Send ENQUIRE to n1..nk

Add_New_Gossip_Message(msg):
 1.   Add msg to S.
```

Note: For testing, we chose an interval of 1 second for enquiring, but it is higher than 1 second in general.

Experiments

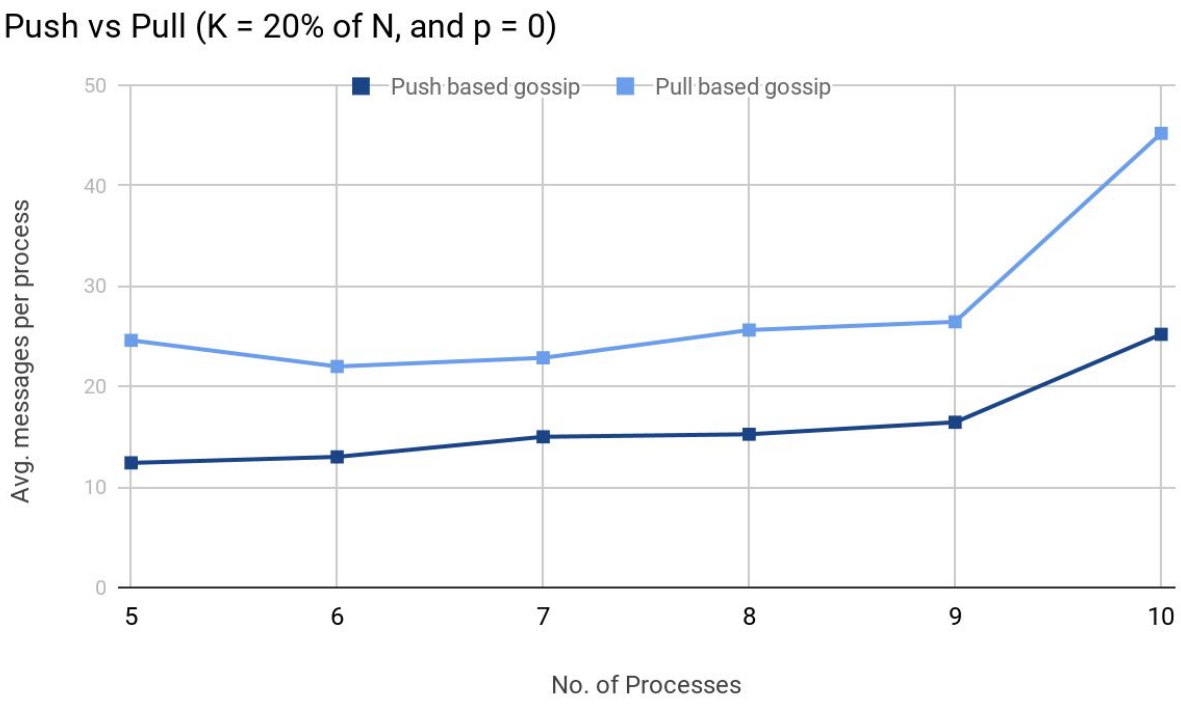
We tested the algorithms using the following parameters:

No. of Processes (**N**)	5 - 10
Random **K%** processes to push or pull from	20% and 50% of N
Packet loss probability (**p**)	0, 0.1, 0.5
No. of messages generated per process	5

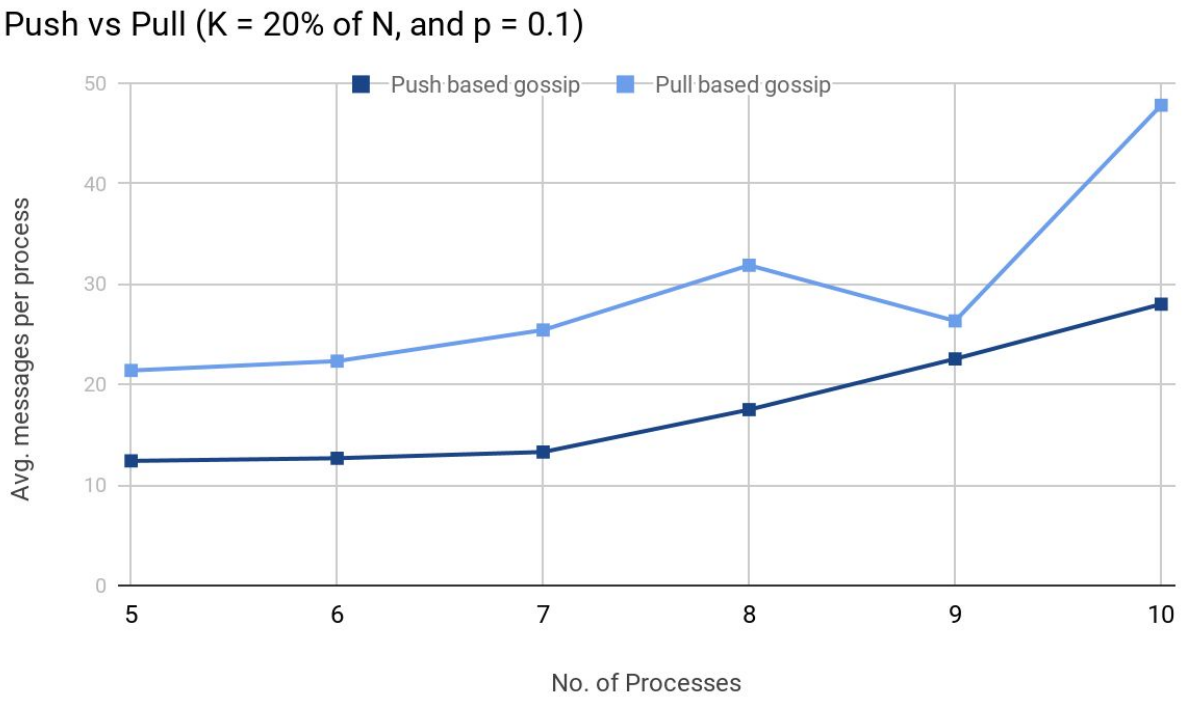
Graph for communication channels was randomly generated with each node having **N/2** neighbours.

Results

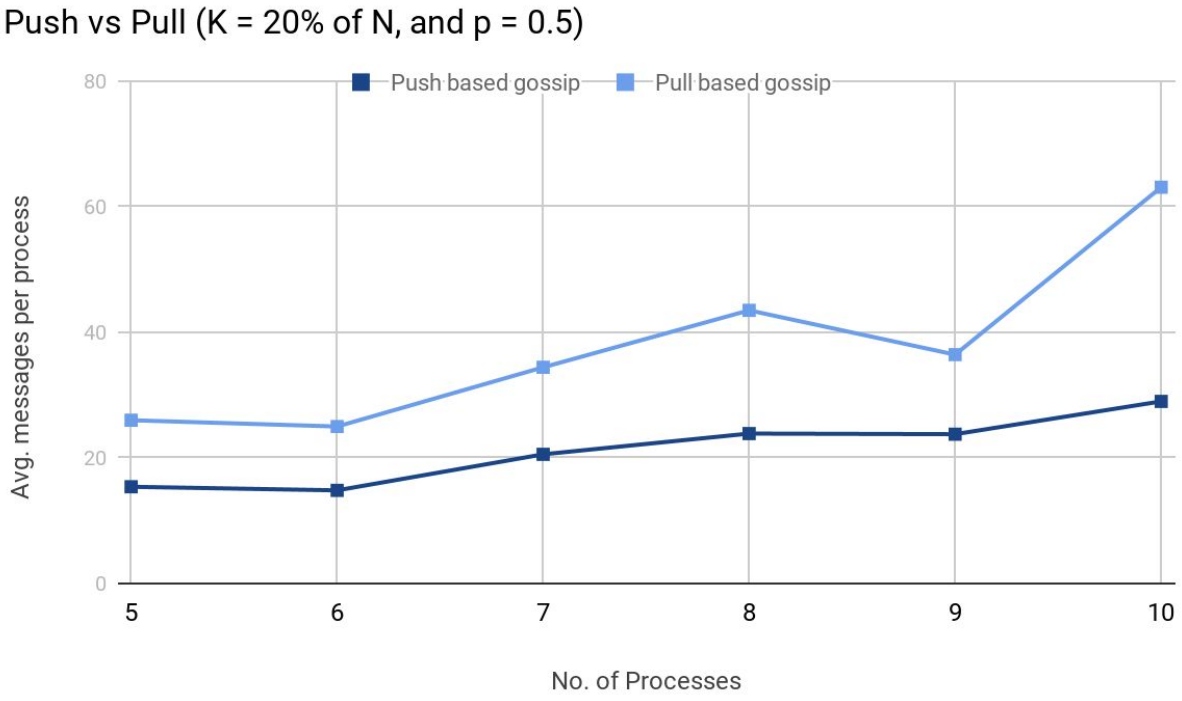
Graph 1



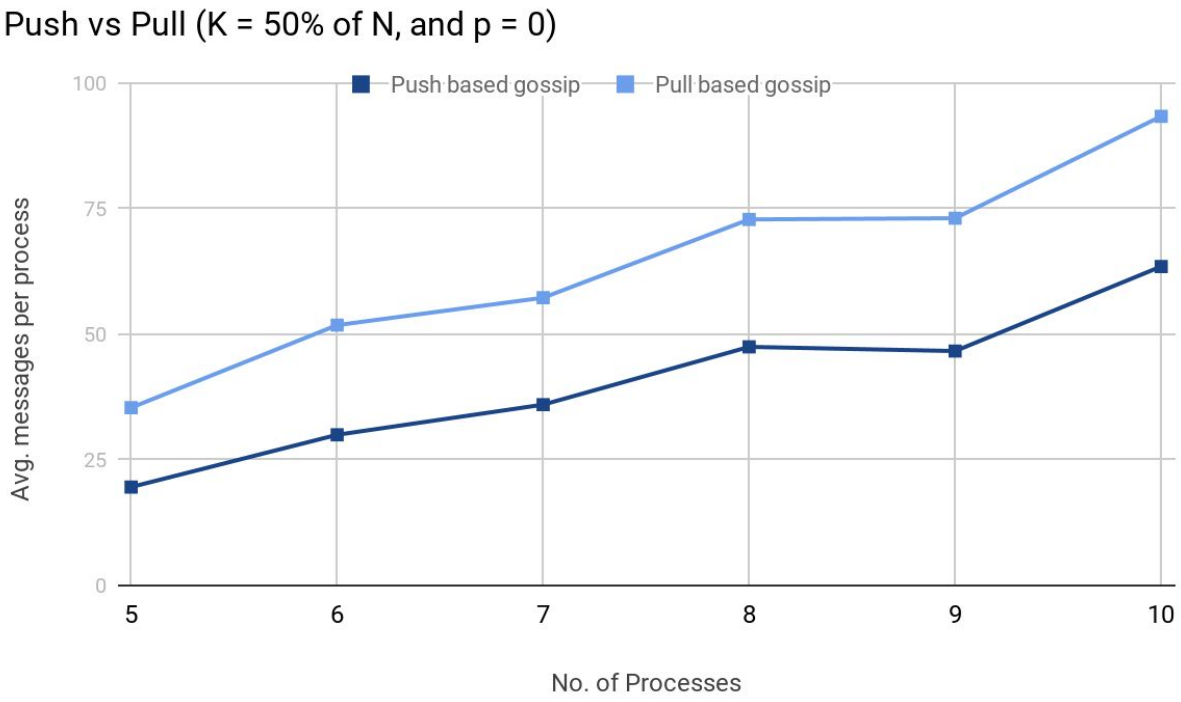
Graph 2



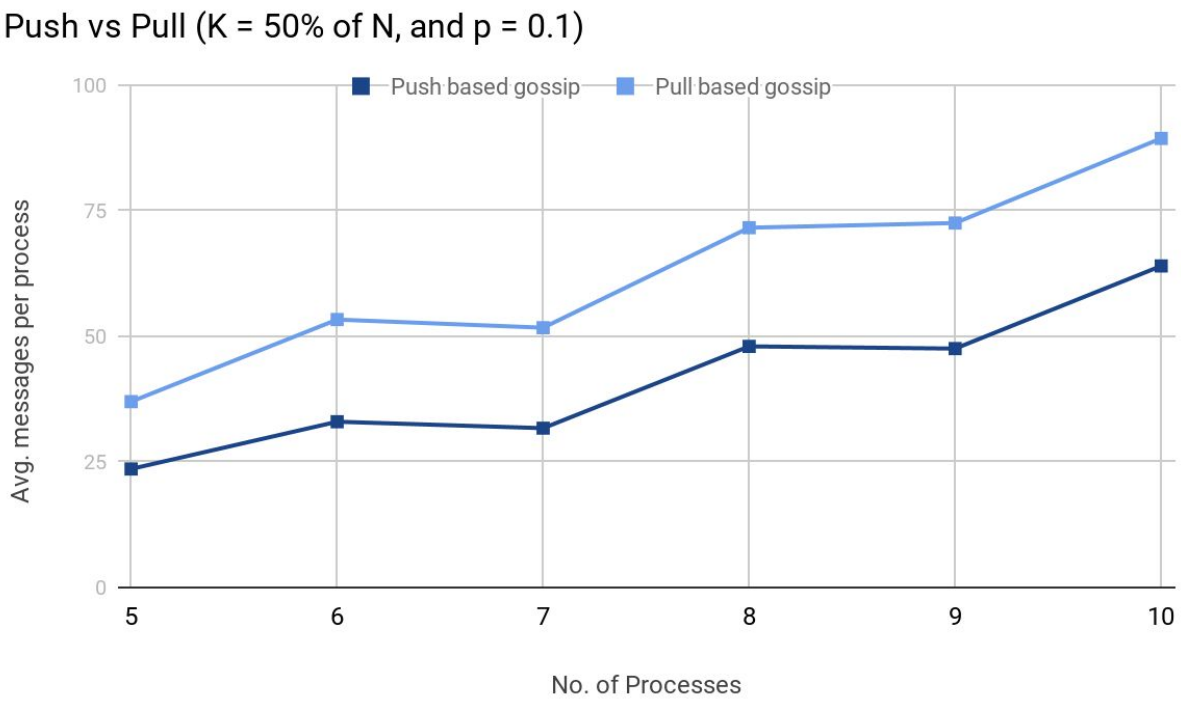
Graph 3



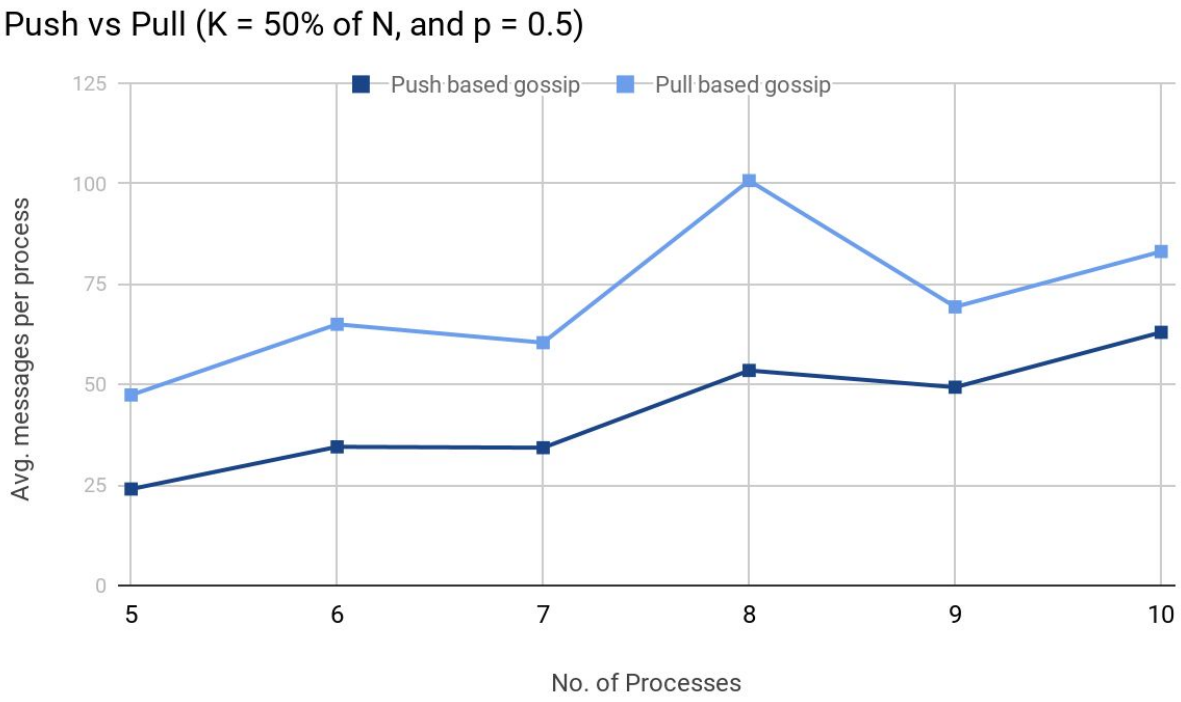
Graph 4



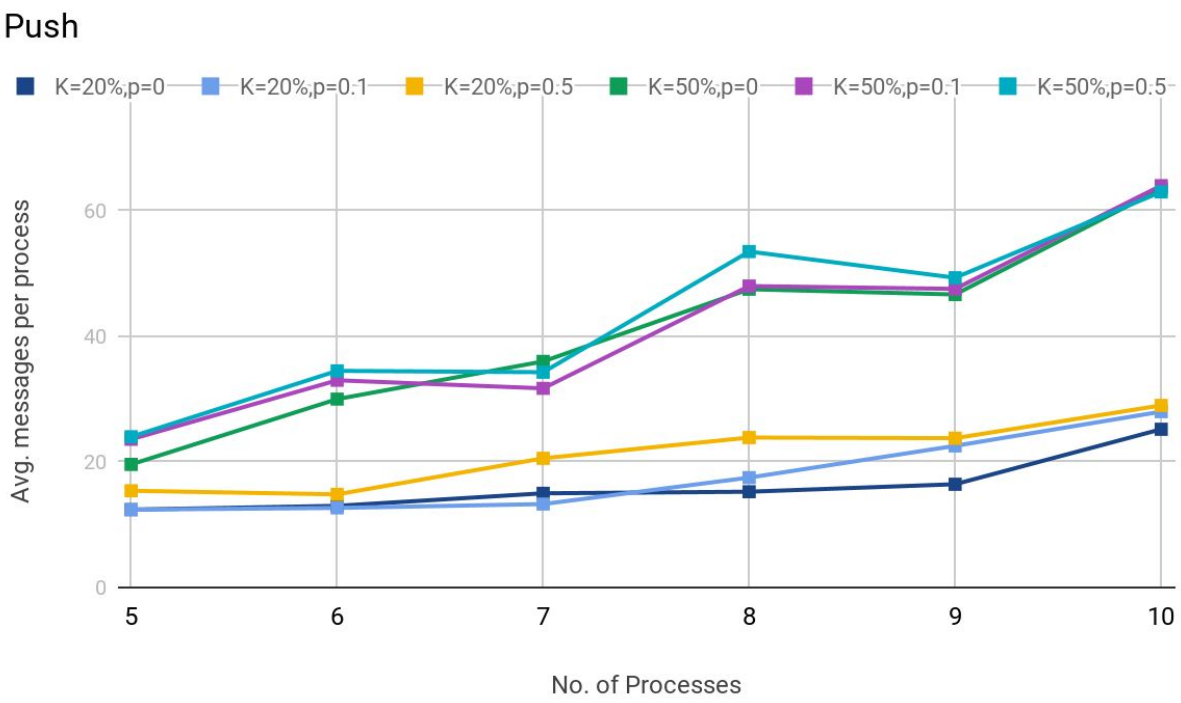
Graph 5



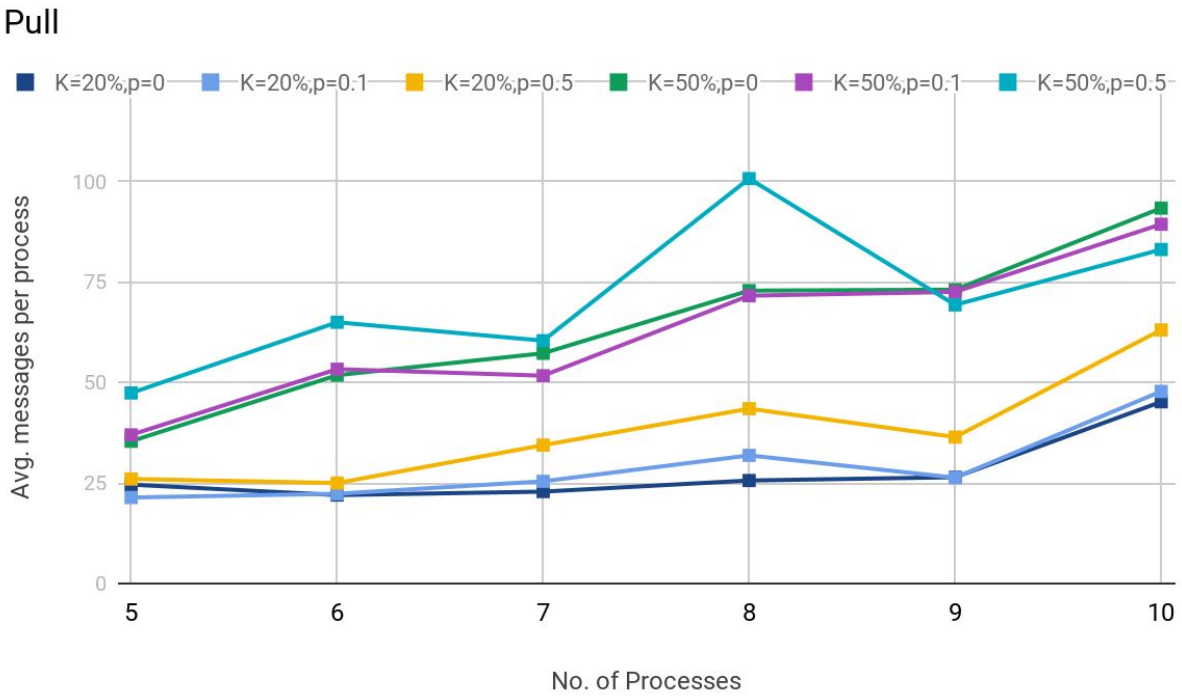
Graph 6



Graph 7



Graph 8

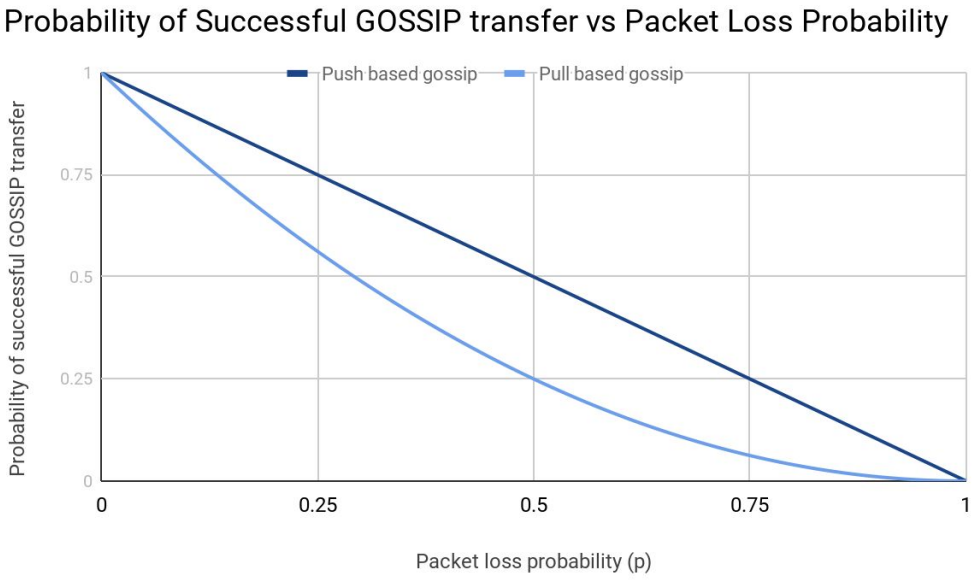


Observations

Push vs Pull (Graph 1-6)

- From Graph 1 to 6 we can see that pull based gossip always took more number of messages on average.
    - To exchange a single gossip message, push based gossip requires only 1 GOSSIP message to be sent. Whereas in pull, the process has to send an ENQUIRE message before it gets the reply as GOSSIP message. Hence 2 messages exchanged to get 1 gossip message.

- When the packet loss probability (**p**) was increased, the messages per process in pull based gossip increased more significantly compared to pull based. Here is our reasoning behind it:
    - In Push based gossip, as only 1 message is involved to transfer a GOSSIP message, the probability of successful gossip transfer is **(1-p)**.
    - But as in Pull based there are 2 messages involved to transfer a GOSSIP message, the probability of successful transfer of both becomes $\mathbf{(1-p)^2}$



- In practice >10% of packet losses is considered bad connection. In that range we can see that Pull based gossip has lower probability of GOSSIP message transfer compared to Push.

Push and Pull (Graph 7-8)

- When **K** was higher (i.e. pushing or pulling from more number of neighbours), the message complexity was significantly higher in both Push and Pull, although the time taken was similar, which means there was redundancy.
  - From this we can infer that, we don't necessarily need to push or pull from more neighbours to broadcast a message.

- For a fixed **K**, when packet loss probability (**p**) was more, it seemed to take more messages on average to gossip all message in both Push and Pull. This is expected as lost messages needs to be retransferred.

- With increase in **p** (for same **K**), the average no. of messages did not grow very quickly for both Push and Pull, which is an indication that gossip protocols are robust against message losses.

- The average no. of messages sent increases with increase in number of processes, this might be because it takes more time to get eventual consistency when there are more processes, and hence more messages are pushed/pulled in that duration.

- The ups and down in the graph line can be attributed to the graph topology and not specifically **N**.

Pros and Cons

Push	Pull
Pros ● Less message complexity compared to Pull.	Pros ● A process can ask neighbour for gossip only when it wants to. There is no unnecessary flooding of messages.
Cons ● A process keeps receiving the gossip messages even if it doesn't want to.	Cons ● More prone to message loss as 2 messages are involved for a single gossip message. ● Higher message complexity then Push.

References

[1] https://en.wikipedia.org/wiki/Gossip_protocol
[2] https://www.inf.ed.ac.uk/teaching/courses/ds/slides1718/GossipAlgo.pdf
[3] https://www.coursera.org/lecture/cloud-computing/1-2-the-gossip-protocol-5AOex