Communication Protocols

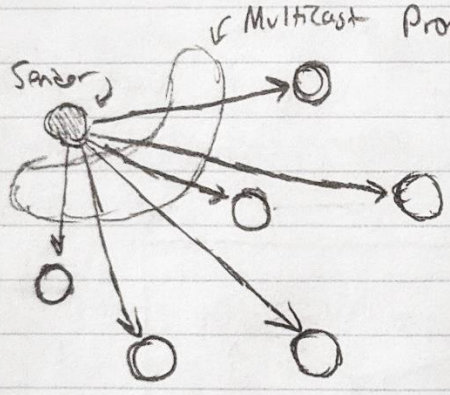
June 4th 2019

Don't assume data sent across a network is the same on arrival. To be sure, do checksums / validity checks

Keeping state across machines can be achieved by caching or replication strategies

Communication Strategies

Multicast If a single node wanted to "broadcast" a piece of data to other nodes, However, there's many ways to distribute this information, and so we call it the multicast protocol

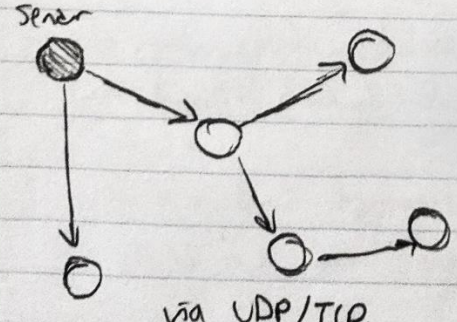


Here, nodes may crash, packets dropped, network fails, along 1000s of nodes.

So we need reliability, and speed

Centralized Multicast Protocol: Directly send data by TCP/UDP to all other nodes. However, network topology may not actually be supporting. This can only work if the topology is a mesh, but that's a bad assumption to make.

Tree-Based Multicast Protocols The goal is to build a spanning tree to distribute data.



Use acknowledgements (ACKs) or "negative" acknowledgements (NAKs) to repair multicasts not received.

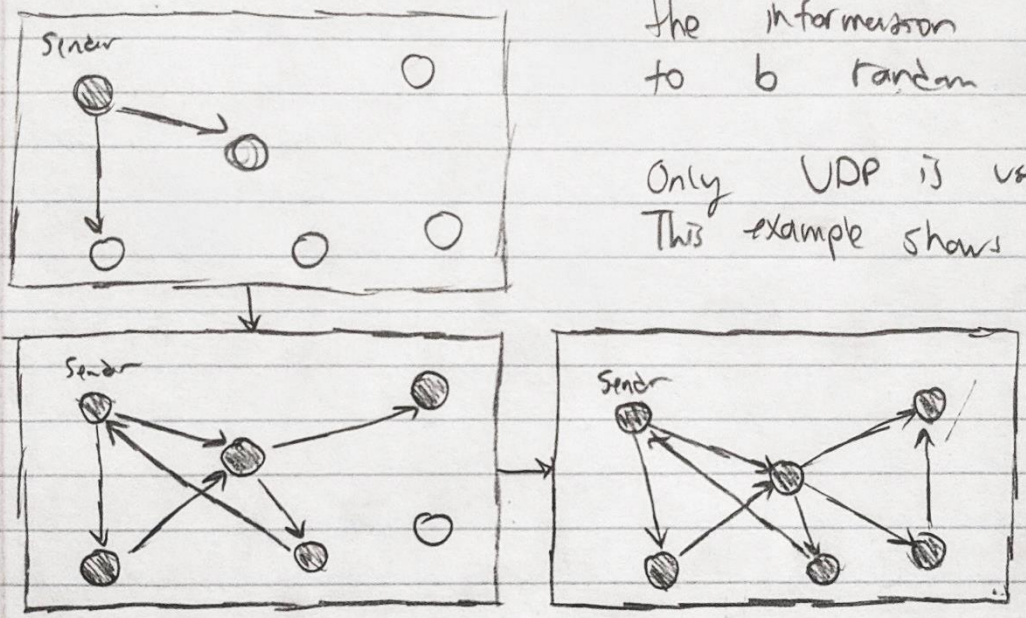
June 4th 2019

Scalable Reliable Multicasts → uses NAKs, but w/ exponential backoff to avoid NAK storms.

Reliable Multicast Transport Protocol → Uses ACKs, but only sent to designated receivers.

Both require an O(n) ACK/NAK overhead.
Some example Tree-based multicast protocols are IP multicast, SRM, RMTP, TRAM, TMTP.

Epidemic Multicast (Gossip)



The idea is that periodically, the information is transmitted to b random targets.

Only UDP is used here. This example shows "push" gossip.

There also exists pull gossip, where nodes will reach out to other nodes and update stuff. Along w/ hybrid push-pull solutions.

Some existing implementations include Cassandra for maintaining membership lists, and AWS EC2/S3 cloud

Push vs. Pull Gossip

June 25th 2019

For push, once you have the message, you gossip about it. If there's multiple messages, you can choose which to gossip about: recently received ones, higher priority ones, random subset, etc.

For pull, periodically poll a few random processes for new multicast messages you haven't received.

Hybrids also exist, both pushing & pulling.

Gossip Analysis: Anti-Entropy

Consider a system with N nodes. One of the nodes initiates a message m to the others. Suppose a node picks another at random for push/pull.

Let p_i be the probability a node f has not received m after the i^{th} round.

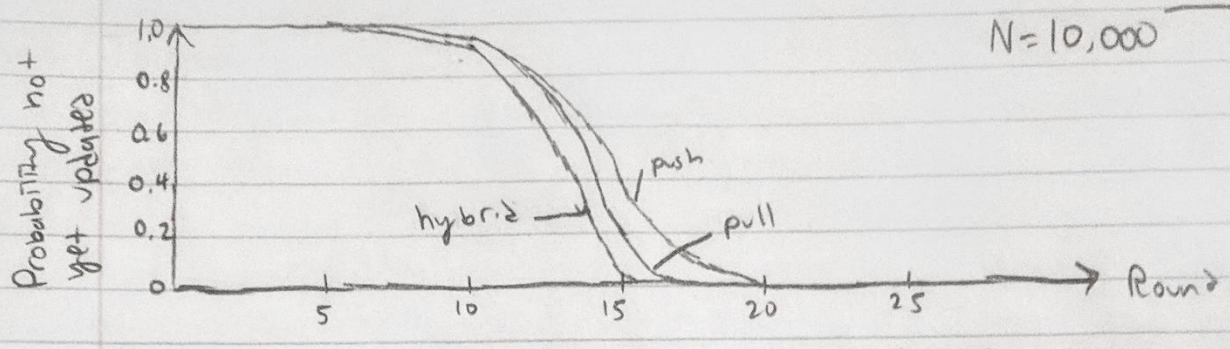
1) With a pure pull approach, $p_{i+1} = (p_i)^2$.
Not only had f not yet been updated in the previous round, but the node that f is contacting also had not yet received m.

2) With a pure push approach, $p_{i+1} = p_i\left(1 - \frac{1}{N-1}\right)^{N(1-p_i)}$
Similarly, f should not have been updated in the previous round, but also none of the nodes that have been updated with m should contact f.

The probability that a node contacts f is $\left(1 - \frac{1}{N-1}\right)$, so we can expect that there are $N(1-p_i)$ updated nodes in round i.

3) With a hybrid approach, we can just combine the 2.
f should not be contacted by one, and it shouldn't contact one.

June 25th 2014



Assuming all nodes are up & running all the time, anti-entropy is extremely effective.

Note that in a push-based approach, and many nodes are infected, then the probability of each one selecting a susceptible node is relatively small.

By contrast a pull-based approach works better when many nodes are infected, as any susceptible node that trigges an update will likely receive the update.

However, all forms will rapidly spread messages if a single node is infected, although the hybrid solution remains the best overall.

If we define a round as a period in which every node will have taken the initiative once to exchange updates with a randomly chosen other node, then the # of rounds to propogate an update takes $O(\log(n))$.

This shows anti-entropy is scalable!
This algorithm is also called the SI model, as every node is either susceptible, or infective.

It's also called antientropy by reducing entropy upon node-to-node updating.