Scalable Distributed Topologies

Graphs

A graph G(V, E) can be defined by a set of vertices, and a set of edges that connect pairs of vertices.

- Graphs can be **directed** or **undirected** (bi-directional edges)
- **Simple Graph**: undirected graph with no loops and no more than one edge between any two different vertices
- **Adjacent Vertices**: have an edge connecting them (neighbours)
- **Weighted Graph**: each edge has a weight
- **Path**: sequence of vertices with edges connecting them
  - **Walk**: edges and vertices can be repeated
  - **Trail**: only vertices can be repeated
  - **Path**: no repeated vertices or edges
- **Complete Graph**: each pair of vertices has an edge connecting them
- **Connected Graph**: there is a path between any 2 nodes
- **Star**: a "central" vertice and many leaf nodes connected to center
- **Tree**: a connected graph with no cycles
- **Planar Graph**: vertices and edges can be drawn in a plane and no 2 edges intersect (Rings and Trees)
- **Connected Component**: maximal connected subgraph of G
- **Degree of vi**: number of adjacent vertices to vi
- **Distance d(vi, vj)**: length of the shortest path connecting those nodes
- **Eccentricity of vi**: ecc(vi) = max({d(vi, vj)|vj ∈ V})
- **Diameter**: D = max({ecc(vi)|vi ∈ V})
- **Radius**: R = min({ecc(vi)|vi ∈ V})
- **Center**: {vi |ecc(vi) == R}
- **Periphery**: {vi |ecc(vi) == D}
- **Random Geometric**: Vertices are dropped randomly uniformly into a unit square and adding edges to connect any two points within a given euclidean distance.
- **Random Erdos-Renyi**: G(n, p) model, n nodes are connected randomly. Each edge is included with independent probability p.
- **Watts-Strogatz model**: Seen later on Small World
- **Barabasi-Albert model**: Preferential attachment → the more connected a node is, the more likely it is to receive new links. Degree Distribution follows a power law.

Spanning Trees

Synchronous SyncBFS Algorithm

- **Strongly Connected**: for every pair of vertices u and v there is a path from u to v and a path from v to u
- **Distance from u to v**: length of the shortest path from u to v
- **Breadth First (root node i)**: each node at distance d from i in the graph appears at depth d in the tree
- Every strongly connected graph has a breadth-first directed spanning tree.

Processes communicate over directed edges. Unique UIDs are available, but network diameter and size is unknown

Initial state in SyncBFS

- `parent = nil`
- `marked = False` (True in root node i0)

SyncBFS Algorithm

- Process i0 sends a search message in round 1
- Unmarked processes receiving a search message from x do `marked = True` and set `parent = x`, in the next round search message are sent from these processes

Complexity

- **Time**: at most *diam* rounds (depending on i0 eccentricity)
- **Message**: $|E|$. Messages are sent across all edges E

Child Pointers

For parents to know offsprings:

- processes must reply to search messages with either `parent` or `nonparent`

Termination: Making i0 know that the tree is constructed

All processes respond with `parent` or `nonparent`.

- Parent terminates when all children terminate. Responses are collected from leaves to tree root.

Applications of Breath First Spanning Trees

Aggregation (Global Computation)

Input values in each process can be aggregated towards a sync node. Each value only contributes once, many functions can be used: `Sums`, `Averages`, `Max`, `Voting`.

Leader Election

Largest UID wins. All process become root of their own tree and aggregate a `Max(UID)`. Each decide by comparing their own UID with `Max(UID)`.

Broadcast

Message payload m can:

- be attached to SyncBFS construction (m |E| message load)
- broadcasted once tree is formed (m |V| message load).

Computing Diameter

1. Each process constructs a SyncBFS.
2. Determines maxdist, longest tree path.
3. All processes use their trees to aggregate max(maxdist) from all roots/nodes.

- **Complexity**: Time O(diam) and messages O(diam × |E|)

System Model

The lack of helping tools is compensated by a "generous" system model:

- **Faults**: No faults
- **Channels**: Reliable FIFO send/receive channels

Reliable FIFO send/receive channels

- Messages don't come out of the blue
- Messages are not lost
- Messages are not duplicated
- Order is preserved

AsynchSpanningTree vs SynchBFS

- While AsynchSpanningTree looks like an asynchronous translation of SynchBFS, the former **does not necessarily produce a breadth first spanning tree**.
- **Faster longer paths will win over** a slower direct path when setting up parent.
- One can however show that a spanning tree is constructed.

AsynchSpanningTree

The AsynchSpanningTree algorithm constructs a spanning tree in the undirected graph G.
Altough a tree with height h, such that h > diam, can occur it only occurs if it does not take more time than a tree with h = diam. → **Faster long paths must be faster!**

Complexity

- **Message**: O(|E|)
- **Time**: O(diam(l + d))

Child pointers and Broadcast

If nodes report `parent` or `nonparent` one can build a tree that broadcasts.

- **Time Complexity**: O(h(l + d)), at most O(n(l + d)), where n = |V|

Broadcast with Acks

- Collects acknowledgements as the tree is constructed
- Upon incoming broadcast messages each node Acks if they already know the broadcast and Acks to parent once when all neighbours Ack to them

Leader Election with AsynchBcastAck

- Termination and if all nodes initiate it and report their UIDs it can be used for Leader Election with unknown diameter and number of nodes

Epidemic Broadcast Trees

Plumtree protocol

	Pros	Cons
Gossip Broadcast	Highly scalable and resilient	Excessive message overhead
Tree-based Broadcast	Small message complexity	Fragile in the presente of failures

Gossip Strategies

- **Eager Push**: Nodes immediately forward new messages
- **Pull**: Nodes periodically query for new messages
- **Lazy Push**: Nodes push new message ids and accept pulls (there is a separation among payload and metadata)

Gossiping into tree

1. Nodes **sample random peers** into eagerPush set
2. Neighbours should be **stable** and TCP can be used
3. Links are kept **reciprocal** (towards undirected graph)
4. First message reception puts **origin in eagerPush**

5. Further duplicate receptions **moves source to lazyPush**
6. **Eager** push of **payload** and **lazy** push of **metadata**

Tree breacks and graph stays connected → nodes get metadata but not payloads

- detected by timer expiration
- metadata source in lazyPush is moved to eagerPush

Small Worlds

Milgram experiment "Six degrees of separation"

- Path lengths were calculated on sequences of letter forwardings.
- In theory, letters would be send from random senders and progressively forwarded to random recipients.
- At each point the letter was forwarded to an address and recipient more likely to know the final destination recipient.
- An average path length close to 6 hops was found in the results.

Random graphs and clustering

- Consider a graph where a given number of edges is created uniformely at random among the graph vertices.
- The resulting random graph is known to depict a **low diameter** and thus could support **small paths**. O(log n).
- Are random graphs a good model for people acquaintances?
    - Not so, since people's graphs have **more clustering**. If A is friend to B and C, then it is likely that B and C are also friends.

Watts and Strogatz proposed a model that mixes short range and long range contacts.

- Nodes establish k local contacts using some distance metric among vertices (say in a ring or lattice) and then a few long range contacts uniformly at random.
- Resulting in **low diameter** and **high clustering**.

Routing in Small Worlds

- Flood a Watts and Strogatz graph → short route between to arbitrary points. A global observer could also pinpoint a O(log N) path.

- But, can we pick a path with local knowledge and a distance metric?

    - Not so easy. Going to the next nearest point does not home in the target, we can keep jumping and only achieve O(√ N) paths. These paths lack locallity.

- **Kleinberg Solution**

- choosing a probability function that can restore locality to long links. (Check in R: n=10; s = exp(log(n)* (runif(1000) -1)); hist(s,100).)
- Long range contacts can be tuned to become more clustered in the vicinity. The target is to have uniformity across all distance scales, a property found in DHT designs like Chord, and locally find O(log2 N) routes.