Distributed Algorithms - Continued
Leader Election – Ring algorithm

Initiates the election

Election: 3

Goal: Elect highest id process as leader
Initiates the election

Election: 32

Goal: Elect highest id process as leader
Initiates the election

Goal: Elect highest id process as leader

Election: 32
Initiates the election

Election: 80

Goal: Elect highest id process as leader
Initiates the election

Goal: Elect highest id process as leader
Goal: Elect highest id process as leader

Initiates the election

Election: 80
Initiates the election

Goal: Elect highest id process as leader
Initiates the election

Elected: 80

Goal: Elect highest id process as leader
Initiates the election

Elected: 80

elected = 80

Goal: Elect highest id process as leader
Initiates the election

Goal: Elect highest id process as leader
Final decision

Goal: Elect highest id process as leader

Initiates the election

N12

elected = 80

N3

elected = 80

N6

elected = 80

N32

elected = 80

N80

elected = 80

N5

elected = 80
Complexity Measures and Metrics

- Space complexity per node
- System-wide space complexity
- Time complexity per node
- System-wide time complexity. Do nodes execute fully concurrently?
- Message complexity
  - Number of messages (affects space complexity of message overhead)
  - Size of messages (affects space complexity of message overhead + time component via increased transmission time)
  - Message time complexity: depends on number of messages, size of messages, concurrency in sending and receiving messages
- Other metrics: # send and # receive events; # multicasts, and how implemented?
- (Shared memory systems): size of shared memory; # synchronization operations
Analysis

- Assume no failures occur during the election protocol itself

- How many messages will be sent?
  - Single initiator
    - Note: Worst case occurs when the initiator is the ring successor of the would-be leader
  - Multiple initiators
What about failures?

- There are several types of failures:
  - Process vs Link failures
  - Malicious vs Benign failures

- Link failures
  - Crash failure: Properly functioning link stops carrying messages
  - Omission failure: Link carries only some of the messages sent on it, not others
  - Byzantine failure (malicious): Link exhibits arbitrary behavior, including creating fake messages and altering messages sent on it

- Timing failures (sync systems): messages delivered faster/slower than specified behavior
Process failures

- Fail-stop: Properly functioning process stops execution. Other processes learn about the failed process (thru some mechanism).
- Crash: Properly functioning process stops execution. Other processes do not learn about the failed process.
- Receive omission: Properly functioning process fails by receiving only some of the messages that have been sent to it, or by crashing.
- Send omission: Properly functioning process fails by sending only some of the messages it is supposed to send, or by crashing. Incomparable with receive omission model.
- General omission: Send omission + receive omission
- Byzantine (or malicious) failure, with authentication: Process may (mis)behave anyhow, including sending fake messages.
- Byzantine (or malicious) failure, no authentication
Can you design your own distributed algorithm?
Maximal Independent Set: Definition

- For a graph \((N; L)\), an independent set of nodes \(N`\), where \(N` \subseteq N\), is such that for each \(i\) and \(j\) in \(N`\), \((i, j) \notin L\).
- An independent set \(N`\) is a maximal independent set if no strict superset of \(N`\) is an independent set.
- A graph may have multiple MIS; perhaps of varying sizes.

- The largest sized independent set is the maximum independent set.

- Application: wireless broadcast - allocation of frequency bands (mutex)
- NP-complete 😊
If we have time

...
Single Source Shortest Path: Sync Bellman-Ford

- Weighted graph, no cycles with negative weight
- No node has global view; only local topology

- Assumption: node knows n; needed for termination
- Termination: n-1 rounds

- After k rounds: length at any node has length of shortest path having k hops
- After k rounds: length of all nodes up to k hops away in final MST has stabilized
Sync Distributed Bellman-Ford: Code

(local variables)
int \( length \leftarrow \infty \)
int \( parent \leftarrow \bot \)
set of int \( \text{Neighbors} \leftarrow \) set of neighbors
set of int \( \{ \text{weight}_{i,j}, \text{weight}_{j,i} \mid j \in \text{Neighbors} \} \leftarrow \) the known values of the weights of incident links

(message types)
UPDATE

(1) if \( i = i_0 \) then \( length \leftarrow 0 \);
(2) for \( \text{round} = 1 \) to \( n - 1 \) do
(3) \hspace{1em} send \( \text{UPDATE}(i, length) \) to all neighbors;
(4) \hspace{1em} await \( \text{UPDATE}(j, length_j) \) from each \( j \in \text{Neighbors} \);
(5) \hspace{1em} for each \( j \in \text{Neighbors} \) do
(6) \hspace{2em} if \( length > (length_j + \text{weight}_{j,i}) \) then
(7) \hspace{3em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} length \leftarrow length_j + \text{weight}_{j,i}; \hspace{1em} parent \leftarrow j.\)
Distance Vector Routing

- Used in Internet routing (popular up to mid-1980s), having dynamically changing graph, where link weights model delay/load
- Variant of sync Bellman-Ford; outer *for* loop is infinite
- Track shortest path to every destination
- length replaced by LENGTH[1...n]; parent replaced by PARENT[1...n]
- kth component denotes best-known length to LENGTH[k]

In each iteration
- apply triangle inequality for each destination independently
- Triangle inequality: \((LENGTH[k] > (LENGTH_j[k] + weight_{j,i})\)
- Node \(i\) estimates \(weight_{i,j}\) using RTT or queuing delay to neighbor \(j\)