CSE535 Asynchronous Systems
Elementary Graph Algorithms

YoungMin Kwon

Shortest Path Algorithm

- Shortest path algorithms
  - Given a weighted graph representing the network topology, a shortest path algorithm finds the shortest path from a node to all other nodes.

Synchronous Bellman-Ford: A Single Source Shortest Path Algorithm

- Network topology
  - Weighted graph
  - Potentially unidirectional link

- Algorithm
  - Given a node and a network topology, find the shortest path from the node to all other nodes
  - Each node knows its neighbors and the weights to them

Synchronous Bellman-Ford: A Single Source Shortest Path Algorithm

(local variables)
int $length \longleftarrow \infty$
int $parent \longleftarrow \perp$
set of int $Neighbors \longleftarrow$ set of neighbors
set of int $\{weight_{i,j}, weight_{j,i} \mid j \in Neighbors\} \longleftarrow$ the known values of the weights of incident links
(message types)
UPDATE

(1) **if** $i = i_0$ **then** $length \longleftarrow 0$;
(2) **for** $round = 1$ **to** $n - 1$ **do**
(3) **send** UPDATE(i, $length$) to all neighbors;
(4) **await** UPDATE(j, $length_j$) from each $j \in Neighbors$;
(5) **for** each $j \in Neighbors$ **do**
(6) **if** $(length > (length_j + weight_{j,i})$ **then**
(7) $length \longleftarrow length_j + weight_{j,i}$; $parent \longleftarrow j$.

Distance Vector Routing (DVR)

- Based on the Synchronous Bellman-Ford algorithm with the following modification
  - The outer for loop runs indefinitely
  - length is replaced by Length[1..n] array
    - Length[k]: the length when node k is the source
    - Length vector is included in the message
  - parent is replaced by Parent[1..n] array
    - Parent[k]: next hop destination four a packet to node k
- Processes exchange their distance vectors periodically

Asynchronous Bellman-Ford: A Single Source Shortest Path Algorithm

(local variables)
int $length \longleftarrow \infty$
set of int $Neighbors \longleftarrow$ set of neighbors
set of int $\{weight_{i,j}, weight_{j,i} \mid j \in Neighbors\} \longleftarrow$ the known values of the weights of incident links

(message types)
UPDATE

(1) **if** $i = i_0$ **then**
(1a) $length \longleftarrow 0$;
(1b) **send** UPDATE$(i_0, 0)$ to all neighbors; **terminate**.

(2) When UPDATE$(i_0, length_j)$ arrives from j:
(2a) **if** $(length > (length_j + weight_{j,i}))$ **then**
(2b) $length \longleftarrow length_j + weight_{j,i}$; $parent \longleftarrow j$;
(2c) **send** UPDATE$(i_0, length)$ to all neighbors;

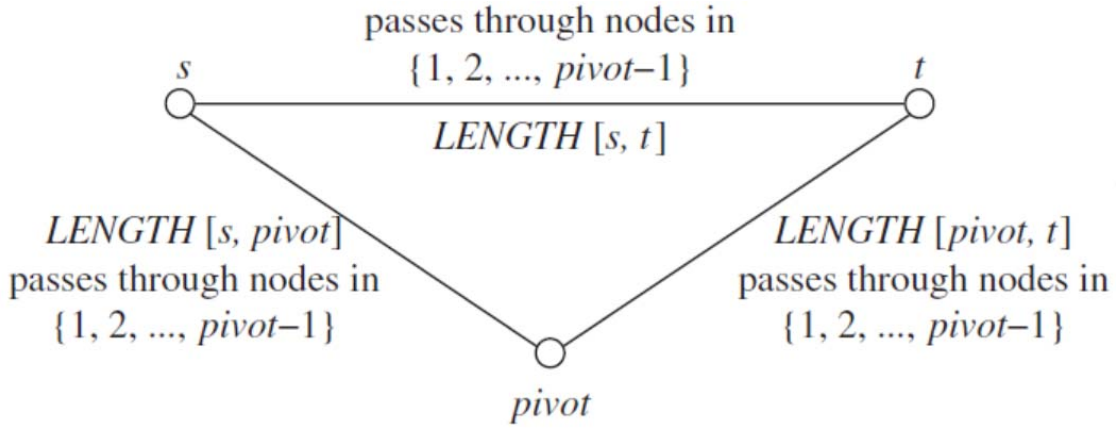
Centralized All Source Shortest Paths: Floyd-Warshall

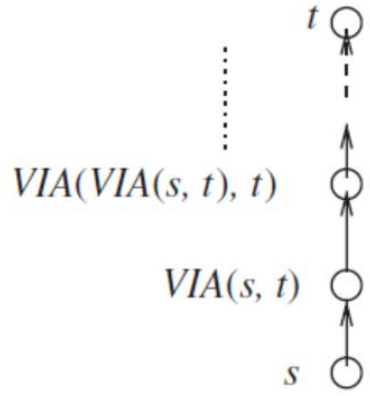
- Let shortestPath(i, j, k) be a function that returns a shortest path from i to j using nodes only from the set {1, 2, …, k}.

- Suppose that we want to find a shortest path from each i to each j using only nodes {1, 2, …, k+1}
  - A path may still use nodes in the set {1, …, k} or
  - A path goes from i to k+1 and from k+1 to j

  - The second case means, a shortest path from i to k+1 using nodes in {1, …, k} and a shortest path from k+1 to j using nodes in {1, …, k}.
  - Hence, shortestPath(i, j, k+1)
    = min(shortestPath(i, j, k),
    shortestPath(i, k+1, k) + shortestPath(k+1, j, k))

Centralized All Source Shortest Paths: Floyd-Warshall





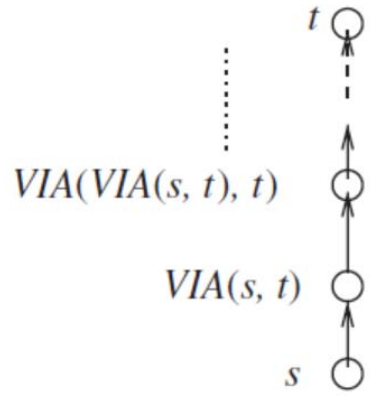

```
(1)  for pivot = 1 to n do
(2)      for s = 1 to n do
(3)          for t = 1 to n do
(4)              if LENGTH[s, pivot] + LENGTH[pivot, t]
                    < LENGTH[s, t] then
(5)                  LENGTH[s, t] ⟵ LENGTH[s, pivot]
                        +LENGTH[pivot, t];
(6)                  VIA[s, t] ⟵ VIA[s, pivot].
```

Toueg's *Asynchronous* All Source Shortest Paths

- Two Issues:
  - How to access remote LENGTH[pivot,t] info.
  - How to synchronize the execution with other nodes.

Toueg's *Asynchronous* All Source Shortest Paths

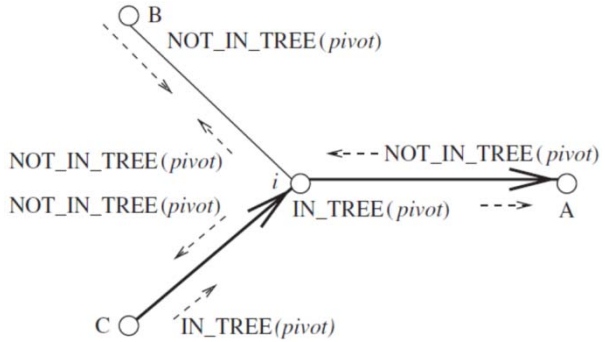
(local variables)
int $LEN[1..n]$ // $LEN[j]$ is the length of the shortest known
// path from i to node j.
// $LEN[j] = weight_{ij}$ for neighbor j, 0 for
// $j = i$, ∞ otherwise
int $PARENT[1..n]$ // $PARENT[j]$ is the parent of node i (myself)
// on the sink tree rooted at j.
// $PARENT[j] = j$ for neighbor j, $\perp$ otherwise



set of int $Neighbors \longleftarrow$ set of neighbors
int $pivot, nbh \longleftarrow 0$

(message types)
IN_TREE($pivot$), NOT_IN_TREE($pivot$),
PIV_LEN($pivot$, $PIVOT\_ROW[1..n]$)
// $PIVOT\_ROW[k]$ is $LEN[k]$ of node $pivot$, which is $LEN[pivot, k]$ in
// the central algorithm.
// the PIV_LEN message is used to convey $PIVOT\_ROW$.

Toueg's *Asynchronous* All Source Shortest Paths




```
(1)   for pivot = 1 to n do
(2)       for each neighbor nbh ∈ Neighbors do
(3)           if PARENT[pivot] = nbh then
(4)               send IN_TREE(pivot) to nbh;
(5)           else send NOT_IN_TREE(pivot) to nbh;
(6)       await IN_TREE or NOT_IN_TREE message from each neighbor;
(7)       if LEN[pivot] ≠ ∞ then
(8)           if pivot ≠ i then
(9)               receive PIV_LEN(pivot, PIVOT_ROW[1..n]) from
                      PARENT[pivot];
(10)          for each neighbor nbh ∈ Neighbors do
(11)              if IN_TREE message was received from nbh then
(12)                  if pivot = i then
(13)                      send PIV_LEN(pivot, LEN[1..n]) to nbh;
(14)                  else send PIV_LEN(pivot, PIVOT_ROW[1..n])
                              to nbh;
(15)          for t = 1 to n do
(16)              if LEN[pivot] + PIVOT_ROW[t] < LEN[t] then
(17)                  LEN[t] ⟵ LEN[pivot] + PIVOT_ROW[t];
(18)                  PARENT[t] ⟵ PARENT[pivot].
```

Flooding Algorithm w/o Spanning Tree

- Flooding Algorithm
  - An algorithm to broadcast messages to all nodes in the system.
  - Spanning trees don’t need to be built in advance.

Asynchronous Flooding Algorithm

(local variables)
int $SEQNO[1..n] \longleftarrow \overline{0}$
set of int $Neighbors \longleftarrow$ set of neighbors
(message types)
UPDATE

(1) To send a message M:
(1a) **if** $i = root$ **then**
(1b) $SEQNO[i] \longleftarrow SEQNO[i] + 1$;
(1c) **send** UPDATE(M, i, $SEQNO[i]$) to each $j \in Neighbors$.

(2) When UPDATE(M, j, $seqno_j$) arrives from k:
(2a) **if** $SEQNO[j] < seqno_j$ **then**
(2b) Process the message M;
(2c) $SEQNO[j] \longleftarrow seqno_j$;
(2d) **send** UPDATE(M, j, $seqno_j$) to $Neighbors/\{k\}$;
(2e) **else** discard the message.

Synchronous Flooding Algorithm

```
(local variables)
int STATEVEC[1..n] ⟵ 0̄
set of int Neighbors ⟵ set of neighbors
(message types)
UPDATE
(1)  STATEVEC[i] ⟵ local value;
(2)  for round = 1 to diameter d do
(3)        send UPDATE(STATEVEC[1..n]) to each j ∈ Neighbors;
(4)        for count = 1 to |Neighbors| do
(5)              await UPDATE(SV[1..n]) from some j ∈ Neighbors;
(6)              STATEVEC[1..n] ⟵ max(STATEVEC[1..n], SV[1..n])
```

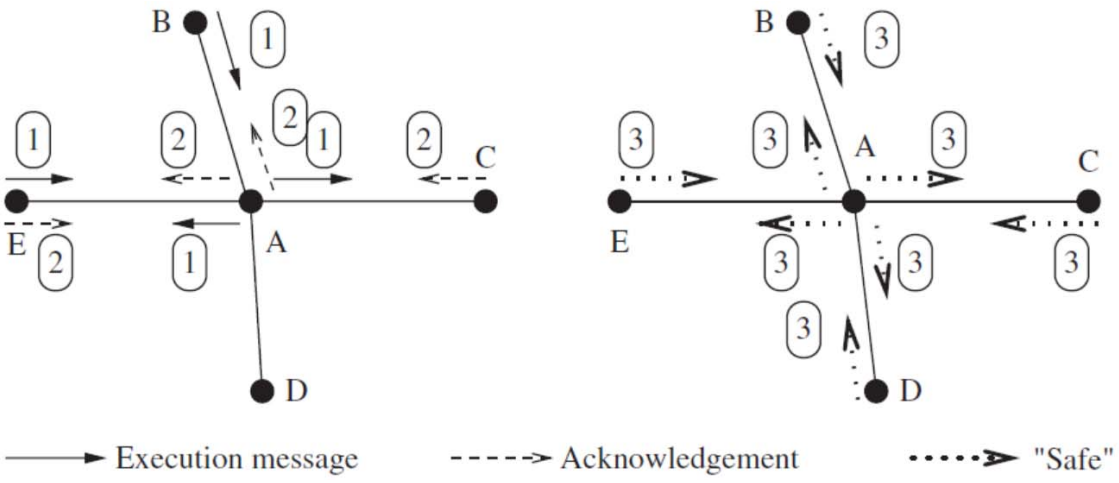
- After d rounds, the local value of each process is propagated through out the network

Synchronizers

- Designing an asynchronous algorithm is more difficult than a synchronous one
  - Spanning-tree algorithm, Shortest-path algorithm, flooding…
- Can an algorithm designed for a synchronous system run on an asynchronous system?
- Syncrhonizers: the generic class of transformation algorithms to run synchronous algorithms on asynchronous systems

The α Synchronizer

- At any process p_i, the α synchronizer in round r moves the process to the next round r+1 if all the neighboring processes are safe for round r
- A process can learn the safety of its neighbor if any message sent by this process is required to be acknowledged.
- Once a neighbor p_j has received ack for all the messages it sent, it sends a message informing p_i that it is safe.

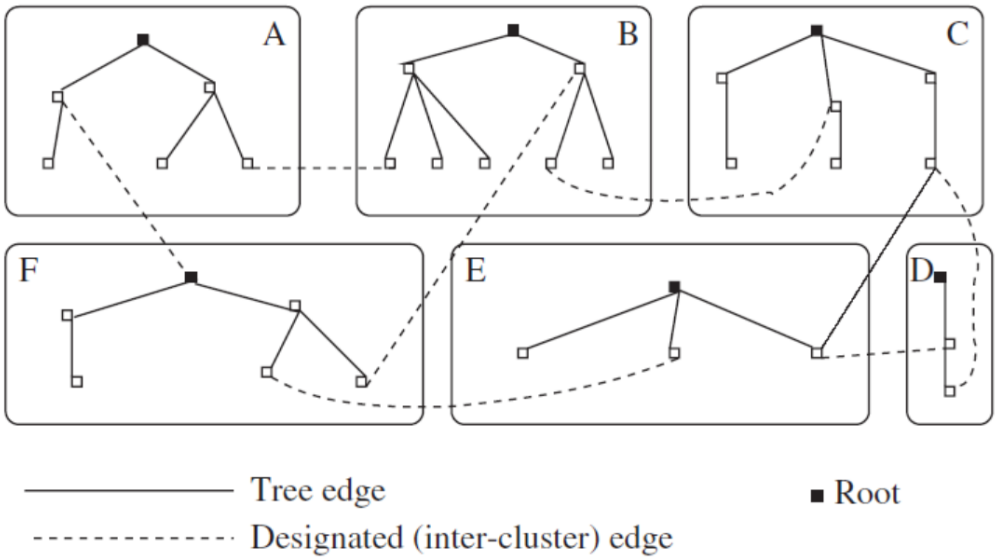


The β Synchronizer

- Assumes a spanning tree.
  - Safe leaf nodes initiate a convergecast
  - An intermediate node propagates the convergecast to its parent when all the nodes in its subtree, including itself, are safe.
  - When the root node becomes safe and receives the convergecast from all its children, it broadcasts all the tree nodes to move to the next phase.

The γ Synchronizer

- The network is organized into a set of clusters.
  - Within a cluster, a spanning tree hierarchy exists.
  - Two clusters are neighbors if there is at least one edge between one node in each of the clusters.
  - Within a cluster, the β synchronizer is executed; once a cluster is stabilized, the α synchronizer is executed among the clusters.



Leader Election

- Leader Election: all the processes agree on a common distinguished process, also termed as the leader.
  - Assumes a ring topology
- Leaders are used in
  - Minimum spanning tree, broadcast and convergecast, …
  - Prevents all processes from initiating a distributed algorithm

Leader Election Algorithm

(variables)
boolean *participate* ← *false* // becomes true when P_i is participates in
// leader election
(message types)
PROBE **integer** // contains a node identifier
SELECTED **integer** // announcing the result

(1) When a process wakes up to participate in leader election:
(1a) **send** PROBE(i) to right neighbor;
(1b) *participate* ⟵ *true*.

(2) When a PROBE(k) message arrives from the left neighbor P_j:
(2a) **if** *participate* = *false* **then** execute step (1) first.
(2b) **if** $i > k$ **then**
(2c) discard the probe;
(2d) **else if** $i < k$ **then**
(2e) **forward** PROBE(k) to right neighbor;
(2f) **else if** $i = k$ **then**
(2g) declare i is the leader;
(2h) circulate SELECTED(i) to right neighbor;

(3) When a SELECTED(x) message arrives from left neighbor:
(3a) **if** $x \neq i$ **then**
(3b) note x as the leader and forward message to right neighbor;
(3c) **else** do not forward the SELECTED message.