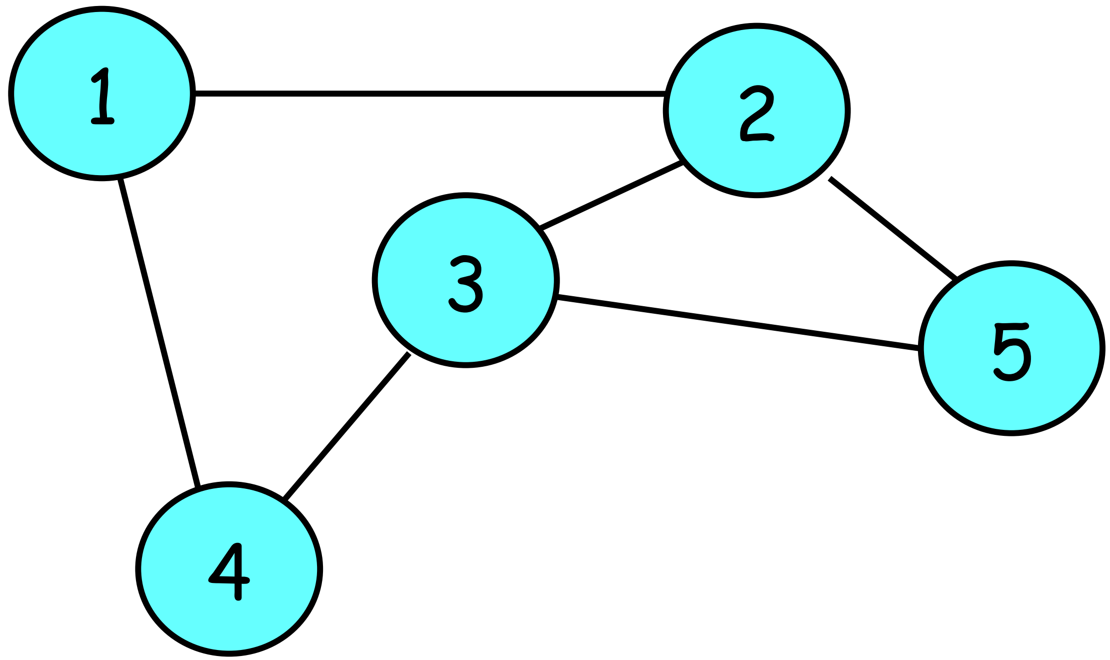
DoC 437 - 2009

Distributed Algorithms

Part 8: Routing Algorithms

The routing problem

- Process (or *host*) is not typically connected to every other process by a channel
- Subset of hosts to which it is connected are called *neighbors*



- The decision process by which one or more neighbors are selected to send (or *forward*) a message on path to destination is called *routing*

Broad classes of routing algorithms

- Flooding

  no need for addresses, just forward messages to all neighbors; sequence numbers used to avoid duplicates

- Random

  no assignment of routes; random forwarding decision

- Static

  assignment of routes established once, usually centrally, and fixed

- Adaptive

  dynamic (re)assignment of routes based on changes

Adaptive routing algorithms

- Goal: design an algorithm that generates for each host a local decision-making procedure to perform the forwarding function
- Some topological information is required at each host: *routing tables*
- Routing algorithms consist of two parts

    table computation (initially and adaptively)

    packet forwarding
- Note: confusing terminology

    "routing" tables are better called "forwarding" tables

Design criteria

- Correctness

  deliver messages to destinations

- Efficiency

  send messages along paths that incur only small delay and ensure high throughput

  an algorithm is *optimal* if it uses the best such paths

- Complexity

  minimize messages, time, and storage

Design criteria (cont.)

- Robustness

  resilient or responsive to topological changes

- Adaptiveness

  balance load on intermediate hosts and channels

- Fairness

  provide service uniformly (unless paid otherwise)

Optimality

- Routing problem treated as a graph problem

  $G = (V, E)$

  where V is set of hosts and E is set of channels/links

- Optimality of an algorithm depends on what is considered "best" path in a graph

  minimum hop: cost of a path measured in terms of the number of channels traversed

  shortest path: each channel statically assigned a weight; cost of a path measured as sum of weights

  minimum delay: each channel dynamically assigned a weight; messages influence each other's costs

Some flavors of routing algorithms

- Destination-based routing

  decision based on destination (and routing tables), independent of the original sender

  can use *spanning tree* (sometimes called a “sink” tree) rooted at the destination

- Source-based routing

  decision based on source (and routing tables)

  can use *spanning tree* (sometimes called a “delivery” tree) rooted at the source

Some flavors of routing algorithms

- Destination-based routing
- Source-based routing
- Hierarchical or "compact" routing

  network partitioned into clusters

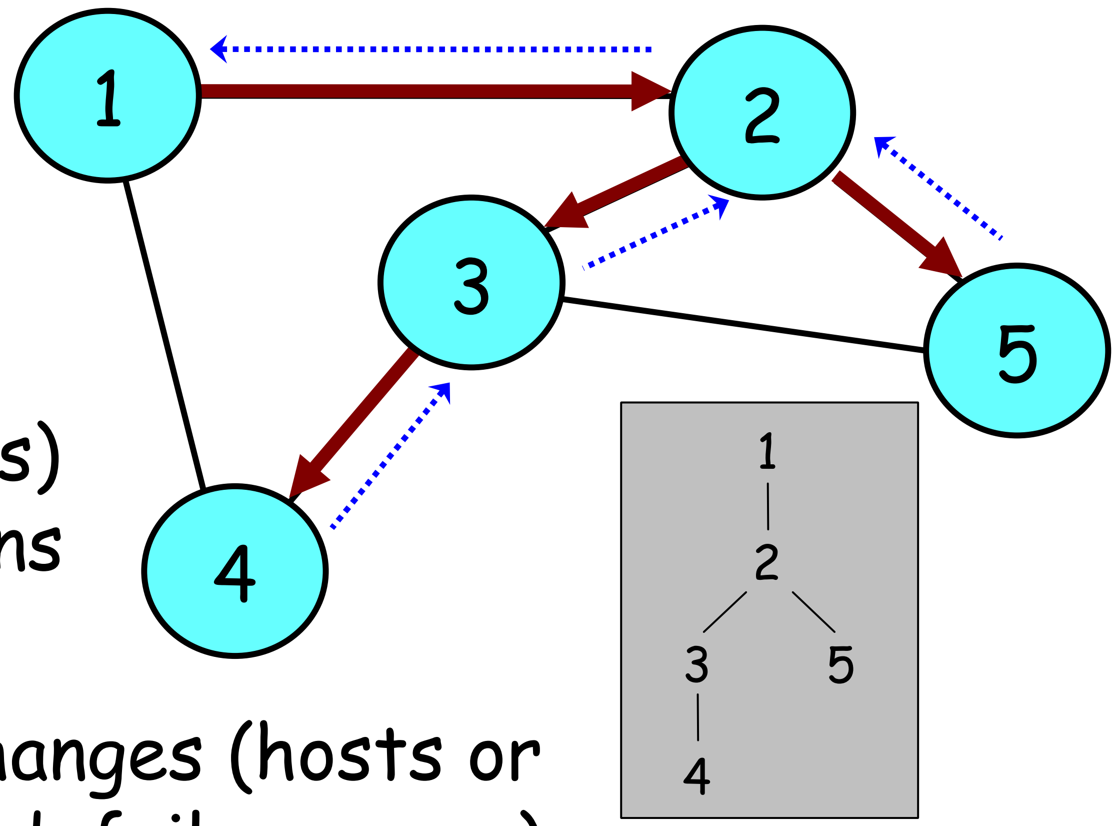
  different algorithms are used at inter-cluster and intra-cluster levels

  can reduce the space needed to store routing tables and the number of decisions needed to forward a message

Destination-based routing

- Construct spanning tree rooted at destination
- Forward along reverse paths

  table lookup → parent



- Some challenges

  combining trees (routes) for multiple destinations

  maintaining spanning trees when topology changes (hosts or channels added, deleted, fail, recover)

Destination-based routing

assumptions

- Assumed cost properties
  - cost of sending a message via a path is independent of the utilization of that path
    - cost is purely a function of the path, not its "load"
  - cost of concatenating two paths equals sum of costs of concatenated paths
    - cost of empty path is 0
  - the graph contains no cycles of negative cost
- Suitable for minimum-hop/shortest-path criteria
  - minimum-delay criterion violates first assumption
    - requires "multi-path" routing schemes

The all-pairs shortest-path problem

- Goal: compute simultaneously the routing tables for all hosts, such that they use the shortest path for each pair of hosts (u,v) and store the first channel (neighbor) of such a path at u
- Solution by Toueg is distributed version of a centralized algorithm for computing all-pairs shortest paths by Floyd and Warshall
- Note: Dijkstra's algorithm solves the "single-source" shortest-path problem

Floyd-Warshall algorithm

```
S:      set of hosts, initially {}
D[]:    array of weights
u,v,w: host

forall (u,v) do
  if u = v then D[u,v] := 0
  else if (u,v) ∈ V then D[u,v] := Weight(u,v)
                    else D[u,v] := ∞
while S not = V do
  pick w from V \ S % w is called the “pivot” host
  forall u ∈ V do
    forall v ∈ V do
      D[u,v] := min(D[u,v], D[u,w] + D[w,v])
  S := S ∪ {w}
```

What is the complexity of this algorithm?

What makes this algorithm "centralized"?

Toueg algorithm

adaptations for a distributed environment

- Assumptions

  each cycle has a positive weight

  each host initially knows identities of all hosts

  each host knows which hosts are its neighbors and the weights of its outgoing channels

Toueg algorithm

adaptations for a distributed environment

- Assumptions
- Partition operations and variables over network

  variable $D[u,v]$ allocated to u, rewritten $D_u[v]$

  assignment to $D_u[v]$ made at u

  when value of variable at host x is assigned to $D_u[v]$, it must be sent from x to u

  pivot host (w) sends information to all hosts through a broadcast

  build a "neighbor" table $Nb_u[v]$ at each host u that serves as the eventual routing table at u

Toueg algorithm (at host *u*)

```
S:           set of hosts, initially {}
Du[],Dw[]: array of weights
Nb[]:        array of hosts % Nb[v]: first hop toward v
u,v,w:       host

forall v ∈ V do
  if u = v then
    Du[v]  := 0
    Nb[v] := undefined
  else if v ∈ Neighbors(u) then
         Du[v]  := Weight(u,v)
         Nb[v] := v
       else
         Du[v]  := ∞
         Nb[v] := undefined
while S not = V do
  ...
```

Toueg algorithm (at host *u*)

```
...
while S not = V do
  pick w from V \ S  % all hosts pick w in same order
  if u = w then
    broadcast Du[]
  else
    receive Dw[] from w
  forall v ∈ V do
    if Du[w] + Dw[v] < Du[v] then
      Du[v] := Du[w] + Dw[v]
      Nb[v] := Nb[w]
  S := S ∪ {w}
```

Toueg algorithm

some observations

- How are weights shared across the network?

  the algorithm presented was a simplified version, where the sharing of weights is underspecified

  full version of algorithm tries to efficiently spread this information, but many later (and better) refinements exist

- What happens if the topology changes?

  full recomputation is required

Toueg algorithm
some more observations

- Uniform selection of next pivot host (w) means set of hosts precisely known by all hosts in advance

  requires execution of additional distributed algorithm to acquire this set in preparation for Toueg

- There are repeated applications of the triangle inequality $d(u,v) \leq d(u,w) + d(w,v)$

  $d(w,v)$ is usually remote, so not at u nor at a neighbor of u and therefore must be repeatedly broadcast

Toueg algorithm

toward an alternative

- Consider another defining equation for $d(u,v)$

$$d(u,v) = \begin{cases} 0 & \text{if } u = v \\ \min\limits_{w \in Nb_u}(\mathrm{Weight}(u,w) + d(w,v)) & \text{otherwise} \end{cases}$$

- This equation exhibits two important properties

  data locality: data are either at the host (u) or at a neighbor (w)

  destination independence: only distances to v are needed to compute distance from u to v, allowing all distances to v to be computed independently of all distances to other hosts

Chandy-Misra algorithm

```
Du,Dv:    weight, initially ∞
Nbu:      host, initially undefined
u,v,w,x: host

% processing at v = v0, the destination host
Dv := 0
forall w ∈ Neighbors(v) do send [MYDIST: v,0] to w

% processing a [MYDIST: v,d] from neighbor w at host u
receive [MYDIST: v,d] from w
if d + Weight(u,w) < Du then
  Du  := d + Weight(u,w)
  Nbu := w
  forall x ∈ Neighbors(u) do send [MYDIST: v,Du] to x
```

- A distributed *diffusion* algorithm: computation started by one process, joined as messages arrive

Tajibnapis algorithm (“Netchange”)

- Goals

  compute routing tables that are optimal according to the minimum-hop measure

  allow the tables to be updated with only a *partial recomputation* after the failure, repair, or addition of a channel

Tajibnapis algorithm ("Netchange")

overview

- Local forwarding decisions based on estimates of the distances to destination hosts

  preferred neighbor is the one estimated to have the smallest distance

- Host u estimates real distance $d(x,v)$

  $D_u[v] \approx d(u,v)$ for each destination v

  $ndis_u[w,v] \approx d(w,v)$ for each neighbor w

  $D_u[v]$ derived from $ndis_u[w,v]$

  $ndis_u[w,v]$ obtained via communication with neighbors

Tajibnapis algorithm ("Netchange")

overview

- Computation of $D_u[v]$

  if $u = v$ then $D_u[v] = d(u,v) = 0$

  if $u \neq v$ then shortest path (if path exists) consists of a channel to the neighbor whose own (estimate) of the distance to v is shortest; this may not be unique

  given n hosts, minimum-hop path is at most n-1, so no (estimated) path is represented as n

Tajibnapis algorithm ("Netchange")

overview

- Sharing distance estimates

  hosts send "mydist" messages to neighbors

  a host *u* receiving a "mydist" message stores the neighbor's estimate in $ndis_u$[] and recomputes its estimate in D_u[]

  if D_u[] changes, then host *u* shares this change with its neighbors using its own "mydist" messages

Tajibnapis algorithm ("Netchange")

overview

- Reaction to failures and repairs of *uw* channel

  u and *w* notified via "fail" and "repair" messages

  a failure causes neighbor to be removed from neighbor list, recomputation of $D_u[]$, and sharing of any change in $D_u[]$ with other neighbors

  a repair (or new channel added) causes neighbor to be added to neighbor list, but no estimates at *w* exist for distances to all destinations *v* from *u* (and vice versa), so *u* sends series of "mydist" messages to *w*

Netchange algorithm (at host *u*)

```
Du[]:      array of 0..n  % Du[v] estimates d(u,v)
Nbu[]:     array of hosts % preferred neighbors
ndisu[]:   array of 0..n  % ndis[w,v] estimates d(w,v)
u,v,w,x:   host

% initialize the data structures
forall w ∈ Neighbors(u), v ∈ V do
  ndisu[w,v] := n
forall v ∈ V do
  Du[v]  := n
  Nbu[v] := undefined
Du[u]  := 0
Nbu[u] := local
forall w ∈ Neighbors(u) do send [MYDIST: u,0] to w
...
```

Netchange algorithm (at host *u*)

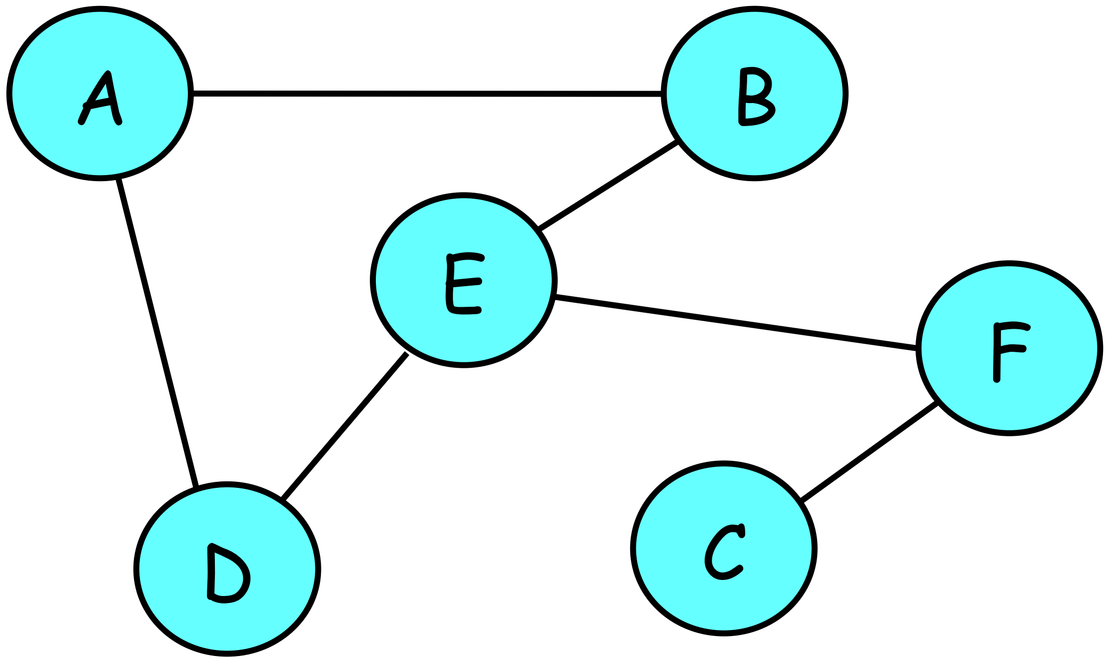
```
...
% process a [MYDIST: v,d] message from neighbor w
receive [MYDIST: v,d] from w
ndisu[w,v] := d
Recompute(v)

% upon failure of channel uw
receive [CLOSED: w]
Neighbors(u) := Neighbors(u) \ {w}
forall v ∈ V do Recompute(v)

% upon repair or new addition of channel uw
receive [OPEN: w]
Neighbors(u) := Neighbors(u) ∪ {w}
forall v ∈ V do
  ndisu[w,v] := n
  send [MYDIST: v,Du[v]] to w
...
```

Netchange algorithm (at host *u*)

```
...
% recompute distance from u to v
Recompute(v):
if v = u then
  Du[v]  := 0
  Nbu[v] := local
else
  d := 1 + min{ndisu[w,v]: w ∈ Neighbors(u)}
  if d < N then
    Du[v]  := d
    Nbu[v] := w with 1 + ndisu[w,v] = d
  else
    Du[v]  := n
    Nbu[v] := undefined
if Du[v] has changed then
  forall x ∈ Neighbors(u) send [MYDIST: v,Du[v]] to x
```

Netchange algorithm example

original topology



Netchange algorithm example

ndis[]

v	u												
	A		B		C	D		E			F		
	B	D	A	E	F	A	E	B	D	F	C	E
A													
B													
C													
D													
E													
F													

Netchange algorithm example

D[] and Nb[]

v	u											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A												
B												
C												
D												
E												
F												

Netchange algorithm example

D[] and Nb[] terminal configuration

v	u											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	4	F	1	A	2	B/D	3	E
B	1	B	0	loc	3	F	2	A/E	1	B	2	E
C	4	B/D	3	E	0	loc	3	E	2	F	1	C
D	1	D	2	A/E	3	F	0	loc	1	D	2	E
E	2	B/D	1	E	2	F	1	E	0	loc	1	E
F	3	B/D	2	E	1	F	2	E	1	F	0	loc

Netchange algorithm example

ndis[] initialized (step 1)

v	u											
	A		B		C	D		E			F	
	B	D	A	E	F	A	E	B	D	F	C	E
A	6	6	6	6	6	6	6	6	6	6	6	6
B	6	6	6	6	6	6	6	6	6	6	6	6
C	6	6	6	6	6	6	6	6	6	6	6	6
D	6	6	6	6	6	6	6	6	6	6	6	6
E	6	6	6	6	6	6	6	6	6	6	6	6
F	6	6	6	6	6	6	6	6	6	6	6	6

Netchange algorithm example

D[] and Nb[] initialized

v	*u*											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	6	?	6	?	6	?	6	?	6	?
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- Next: send [MYDIST: *u*,0] to all neighbors *w*

Netchange algorithm example

ndis[] initialized (step 2)

v	*u*											
	A		B		C	D		E			F	
	B	D	A	E	F	A	E	B	D	F	C	E
A	6	6	0	6	6	0	6	6	6	6	6	6
B	0	6	6	6	6	6	6	0	6	6	6	6
C	6	6	6	6	6	6	6	6	6	6	0	6
D	6	0	6	6	6	6	6	6	0	6	6	6
E	6	6	6	0	6	6	0	6	6	6	6	0
F	6	6	6	6	0	6	6	6	6	0	6	6

- Next: Recompute(*v*) by *u* – let's start with *A* by B

Netchange algorithm example

Recompute(A) by B

v	u											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	6	?	6	?	6	?	6	?
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- Next: send [MYDIST: A,1] to all neighbors *x*

Netchange algorithm example

at A process [MYDIST: A,1] from B

v	u											
	A		B		C	D		E			F	
	B	D	A	E	F	A	E	B	D	F	C	E
A	1	6	0	6	6	0	6	6	6	6	6	6
B	0	6	6	6	6	6	6	0	6	6	6	6
C	6	6	6	6	6	6	6	6	6	6	0	6
D	6	0	6	6	6	6	6	6	0	6	6	6
E	6	6	6	0	6	6	0	6	6	6	6	0
F	6	6	6	6	0	6	6	6	6	0	6	6

- Next: Recompute(A) by A

Netchange algorithm example

Recompute(A) by A

v	*u*											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	6	?	6	?	6	?	6	?
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- No change in D[] so continue Recompute(A) by B

Netchange algorithm example

Recompute(A) by B

v	*u*											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	6	?	6	?	6	?	6	?
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- Next: send [MYDIST: A,1] to all neighbors *x*

Netchange algorithm example

at E process [MYDIST: A,1] from B

v	u											
	A		B		C	D		E			F	
	B	D	A	E	F	A	E	B	D	F	C	E
A	1	6	0	6	6	0	6	1	6	6	6	6
B	0	6	6	6	6	6	6	0	6	6	6	6
C	6	6	6	6	6	6	6	6	6	6	0	6
D	6	0	6	6	6	6	6	6	0	6	6	6
E	6	6	6	0	6	6	0	6	6	6	6	0
F	6	6	6	6	0	6	6	6	6	0	6	6

- Next: Recompute(A) by E

Netchange algorithm example

Recompute(A) by E

v	*u*											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	6	?	6	?	2	B	6	?
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- Next: send [MYDIST: A,2] to all neighbors *x*

Netchange algorithm example

at B process [MYDIST: A,2] from E

v	u											
	A		B		C	D		E			F	
	B	D	A	E	F	A	E	B	D	F	C	E
A	1	6	0	2	6	0	6	1	6	6	6	6
B	0	6	6	6	6	6	6	0	6	6	6	6
C	6	6	6	6	6	6	6	6	6	6	0	6
D	6	0	6	6	6	6	6	6	0	6	6	6
E	6	6	6	0	6	6	0	6	6	6	6	0
F	6	6	6	6	0	6	6	6	6	0	6	6

- Next: Recompute(A) by B

Netchange algorithm example

Recompute(A) by B

v	u											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	6	?	6	?	2	B	6	?
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- No change in D[] so continue Recompute(A) by E

Netchange algorithm example

Recompute(A) by E

v	u											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	6	?	6	?	2	B	6	?
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- Next: send [MYDIST: A,2] to all neighbors *x*

Netchange algorithm example

at D process [MYDIST: A,2] from E

v	u											
	A		B		C	D		E			F	
	B	D	A	E	F	A	E	B	D	F	C	E
A	1	6	0	2	6	0	2	1	6	6	6	6
B	0	6	6	6	6	6	6	0	6	6	6	6
C	6	6	6	6	6	6	6	6	6	6	0	6
D	6	0	6	6	6	6	6	6	0	6	6	6
E	6	6	6	0	6	6	0	6	6	6	6	0
F	6	6	6	6	0	6	6	6	6	0	6	6

- Next: Recompute(A) by D

Netchange algorithm example

Recompute(A) by D

v	*u*											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	6	?	1	A	2	B	6	?
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- Next: send [MYDIST: A,1] to all neighbors *x*

Netchange algorithm example

at A process [MYDIST: A,1] from D

v	u											
	A		B		C	D		E			F	
	B	D	A	E	F	A	E	B	D	F	C	E
A	1	1	0	2	6	0	2	1	6	6	6	6
B	0	6	6	6	6	6	6	0	6	6	6	6
C	6	6	6	6	6	6	6	6	6	6	0	6
D	6	0	6	6	6	6	6	6	0	6	6	6
E	6	6	6	0	6	6	0	6	6	6	6	0
F	6	6	6	6	0	6	6	6	6	0	6	6

- Next: Recompute(A) by A

Netchange algorithm example

Recompute(A) by A

v	u											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	6	?	1	A	2	B	6	?
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- No change in D[] so continue Recompute(A) by D

Netchange algorithm example

Recompute(A) by D

v	*u*											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	6	?	1	A	2	B	6	?
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- Next: send [MYDIST: A,1] to all neighbors *x*

Netchange algorithm example

at E process [MYDIST: A,1] from D

v	u											
	A		B		C	D		E			F	
	B	D	A	E	F	A	E	B	D	F	C	E
A	1	1	0	2	6	0	2	1	1	6	6	6
B	0	6	6	6	6	6	6	0	6	6	6	6
C	6	6	6	6	6	6	6	6	6	6	0	6
D	6	0	6	6	6	6	6	6	0	6	6	6
E	6	6	6	0	6	6	0	6	6	6	6	0
F	6	6	6	6	0	6	6	6	6	0	6	6

- Next: Recompute(A) by E

Netchange algorithm example

Recompute(A) by E

v	u											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	6	?	1	A	2	B/D	6	?
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- No change in D[] so continue Recompute(A) by D

Netchange algorithm example

Recompute(A) by D

v	u											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	6	?	1	A	2	B/D	6	?
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- Next: finished so continue Recompute(A) by E

Netchange algorithm example

Recompute(A) by E

v	u											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	6	?	1	A	2	B/D	6	?
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- Next: send [MYDIST: A,2] to all neighbors *x*

Netchange algorithm example

at F process [MYDIST: A,2] from E

v	u											
	A		B		C	D		E			F	
	B	D	A	E	F	A	E	B	D	F	C	E
A	1	1	0	2	6	0	2	1	1	6	6	2
B	0	6	6	6	6	6	6	0	6	6	6	6
C	6	6	6	6	6	6	6	6	6	6	0	6
D	6	0	6	6	6	6	6	6	0	6	6	6
E	6	6	6	0	6	6	0	6	6	6	6	0
F	6	6	6	6	0	6	6	6	6	0	6	6

- Next: Recompute(A) by F

Netchange algorithm example

Recompute(A) by F

v	u											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	6	?	1	A	2	B/D	3	E
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- Next: send [MYDIST: A,3] to all neighbors *x*

Netchange algorithm example

at C process [MYDIST: A,3] from F

v	u											
	A		B		C	D		E			F	
	B	D	A	E	F	A	E	B	D	F	C	E
A	1	1	0	2	3	0	2	1	1	6	6	2
B	0	6	6	6	6	6	6	0	6	6	6	6
C	6	6	6	6	6	6	6	6	6	6	0	6
D	6	0	6	6	6	6	6	6	0	6	6	6
E	6	6	6	0	6	6	0	6	6	6	6	0
F	6	6	6	6	0	6	6	6	6	0	6	6

- Next: Recompute(A) by C

Netchange algorithm example

Recompute(A) by C

v	u											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	4	F	1	A	2	B/D	3	E
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- Next: send [MYDIST: A,4] to all neighbors *x*

Netchange algorithm example

at F process [MYDIST: A,4] from C

v	u											
	A		B		C	D		E			F	
	B	D	A	E	F	A	E	B	D	F	C	E
A	1	1	0	2	3	0	2	1	1	6	4	2
B	0	6	6	6	6	6	6	0	6	6	6	6
C	6	6	6	6	6	6	6	6	6	6	0	6
D	6	0	6	6	6	6	6	6	0	6	6	6
E	6	6	6	0	6	6	0	6	6	6	6	0
F	6	6	6	6	0	6	6	6	6	0	6	6

- Next: Recompute(A) by F

Netchange algorithm example

Recompute(A) by F

v	u											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	4	F	1	A	2	B/D	3	E
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- No change in D[] so continue Recompute(A) by C

Netchange algorithm example

Recompute(A) by C

v	u											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	4	F	1	A	2	B/D	3	E
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- Next: finished so continue Recompute(A) by F

Netchange algorithm example

Recompute(A) by F

v	u											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	4	F	1	A	2	B/D	3	E
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- Next: send [MYDIST: A,3] to all neighbors *x*

Netchange algorithm example

at E process [MYDIST: A,3] from F

v	u											
	A		B		C	D		E			F	
	B	D	A	E	F	A	E	B	D	F	C	E
A	1	1	0	2	3	0	2	1	1	3	4	2
B	0	6	6	6	6	6	6	0	6	6	6	6
C	6	6	6	6	6	6	6	6	6	6	0	6
D	6	0	6	6	6	6	6	6	0	6	6	6
E	6	6	6	0	6	6	0	6	6	6	6	0
F	6	6	6	6	0	6	6	6	6	0	6	6

- Next: Recompute(A) by E

Netchange algorithm example

Recompute(A) by E

v	u											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	4	F	1	A	2	B/D	3	E
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- No change in D[] so continue Recompute(A) by F

Netchange algorithm example

Recompute(A) by F

v	u											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	4	F	1	A	2	B/D	3	E
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- Next: finished so continue Recompute(A) by B

Netchange algorithm example

Recompute(A) by B

v	*u*											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	4	F	1	A	2	B/D	3	E
B	6	?	0	loc	6	?	6	?	6	?	6	?
C	6	?	6	?	0	loc	6	?	6	?	6	?
D	6	?	6	?	6	?	0	loc	6	?	6	?
E	6	?	6	?	6	?	6	?	0	loc	6	?
F	6	?	6	?	6	?	6	?	6	?	0	loc

- Next: finished so continue Recompute(*v*) by *u*

Netchange algorithm example

D[] and Nb[] terminal configuration

v	u											
	A		B		C		D		E		F	
	B,D		A,E		F		A,E		B,D,F		C,E	
	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu	Du	Nbu
A	0	loc	1	A	4	F	1	A	2	B/D	3	E
B	1	B	0	loc	3	F	2	A/E	1	B	2	E
C	4	B/D	3	E	0	loc	3	E	2	F	1	C
D	1	D	2	A/E	3	F	0	loc	1	D	2	E
E	2	B/D	1	E	2	F	1	E	0	loc	1	E
F	3	B/D	2	E	1	F	2	E	1	F	0	loc

- Channel failure or repair will cause new cascade

Tajibnapis algorithm ("Netchange")

questions

- Is Netchange a "stable" algorithm?

  can be shown that if the topology remains constant after a finite number of changes, then the algorithm reaches a stable configuration after a finite number of steps

- What happens to messages during a topology change?

  cycles may be introduced or erroneous information given about reachability