Digital Communication in the Modern World

Network Layer: Routing Classifications; Shortest Path Routing

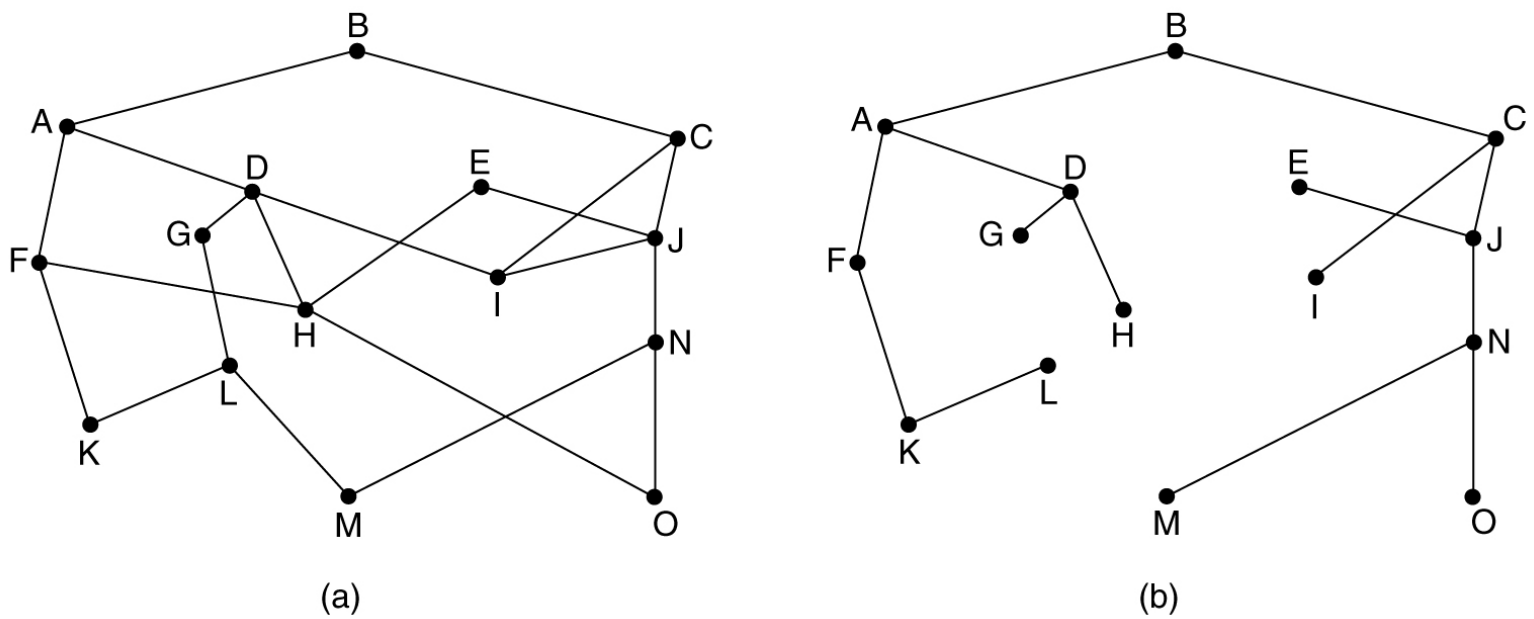
http://www.cs.huji.ac.il/~com1

com1@cs.huji.ac.il

Some of the slides have been borrowed from:
Computer Networking: A Top Down Approach Featuring the Internet,
2nd edition.
Jim Kurose, Keith Ross
Addison-Wesley, July 2002.

Computer Communication 2005-6

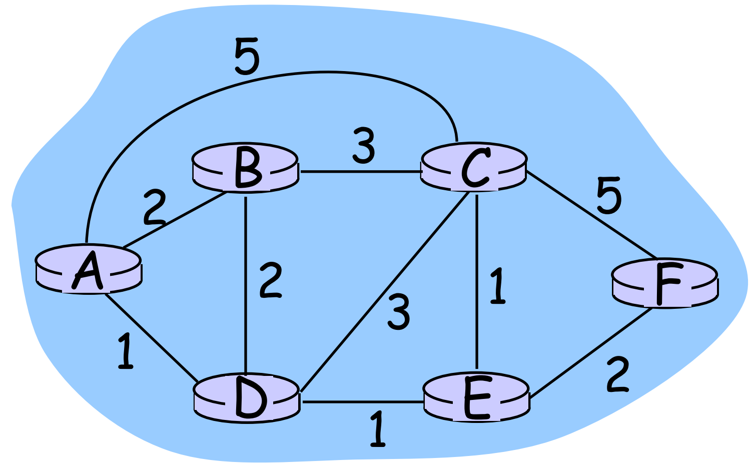
Network Layer's main problem: To get efficiently from one point to the other in a dynamic environment



Routing

Routing protocol

Goal: determine "good" path (sequence of routers) thru network from source to dest.



Graph abstraction for routing algorithms:

- graph nodes are routers
- graph edges are physical links
  - link cost: delay, $ cost, or congestion level

- "good" path:
  - typically means minimum cost path
  - other def's possible (min. num of links)

Datagram Routing Algorithm Classification

- Global (Link State) Routing
  - Shortest Path routing
    - Dijkstra routing
- Decentralized
  - Distance Vector routing
- Hierarchical Routing

A Link-State Routing Algorithm

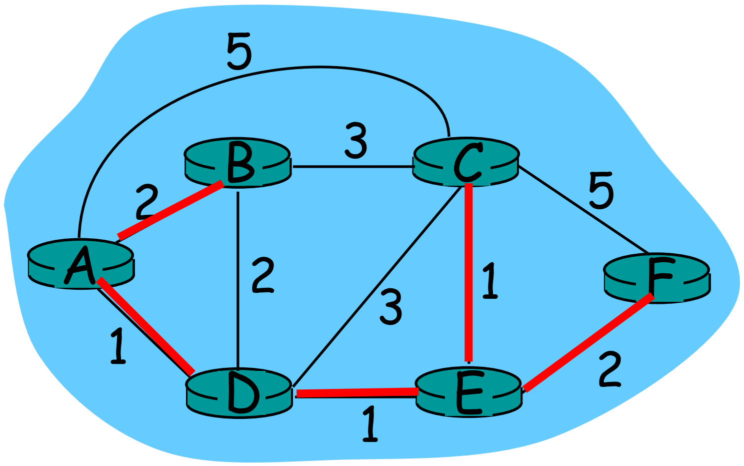
Dijkstra's algorithm

- net topology, link costs known to all nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- computes least cost paths from one node ("source") to all other nodes
  - gives a routing table for that node
- iterative: after k iterations, know the *least cost path* to k dest.'s

Notation:

- *c(i,j)*: link cost from node i to j. Cost infinite if not direct neighbors
- *D(v)*: current value of cost of path from source to dest. V
- *p(v)*: predecessor node along path from source to V, that is next v
- *N*: set of nodes whose least cost path definitively known

Dijkstra's Algorithm




```
Initialization:
  N = {A}
  for all nodes v
    if v adjacent to A
      then D(v) = c(A,v)
      else D(v) = infinity

Loop
  find w not in N such that D(w) is a minimum
  add w to N
  update D(v) for all v adjacent to w and not in N:
    D(v) = min( D(v), D(w) + c(w,v) )
  /* new cost to v is either old cost to v or known
   shortest path cost to w plus cost from w to v */
until all nodes in N
```

Dijkstra's Algorithm in C

```
#define MAX  NODES 1024                 /* maximum number of nodes */
#define INFINITY 1000000000             /* a number larger than every maximum path */
int n, dist[MAX_NODES][MAX_NODES];/* dist[i][j] is the distance from i to j */

void shortest_path(int s, int t, int path[])
{ struct state {                        /* the path being worked on */
      int predecessor;                  /* previous node */
      int length;                       /* length from source to this node */
      enum {permanent, tentative} label; /* label state */
  } state[MAX_NODES];

  int i, k, min;
  struct state *p;

  for (p = &state[0]; p < &state[n]; p++) { /* initialize state */
      p->predecessor = –1;
      p->length = INFINITY;
      p->label = tentative;
  }
  state[t].length = 0;  state[t].label = permanent;
  k = t;                                /* k is the initial working node */
```

Dijkstra's Algorithm in C

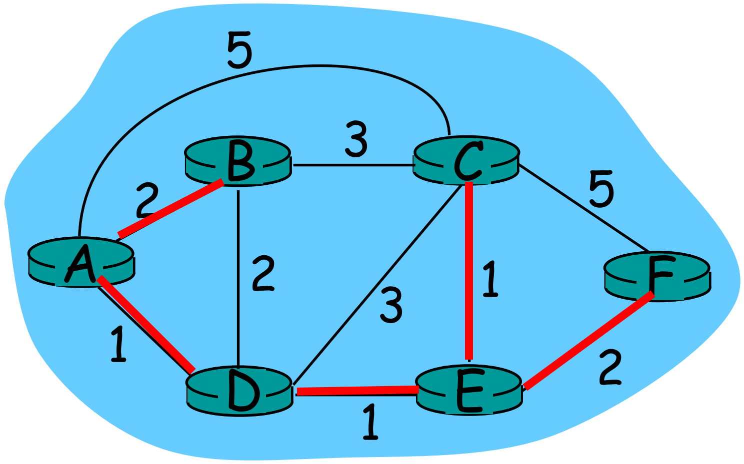
```
do {                                          /* Is there a better path from k? */
     for (i = 0; i < n; i++)                  /* this graph has n nodes */
          if (dist[k][i] != 0 && state[i].label == tentative) {
               if (state[k].length + dist[k][i] < state[i].length) {
                    state[i].predecessor = k;
                    state[i].length = state[k].length + dist[k][i];
               }
          }

     /* Find the tentatively labeled node with the smallest label. */
     k = 0; min = INFINITY;
     for (i = 0; i < n; i++)
          if (state[i].label == tentative && state[i].length < min) {
               min = state[i].length;
               k = i;
          }
     state[k].label = permanent;
} while (k != s);

/* Copy the path into the output array. */
i = 0;  k = s;
do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);
}
```

Dijkstra's algorithm: example

Step	start N	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	infinity	infinity
1	AD	2,A	4,D		2,D	infinity
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
5	ADEBCF					



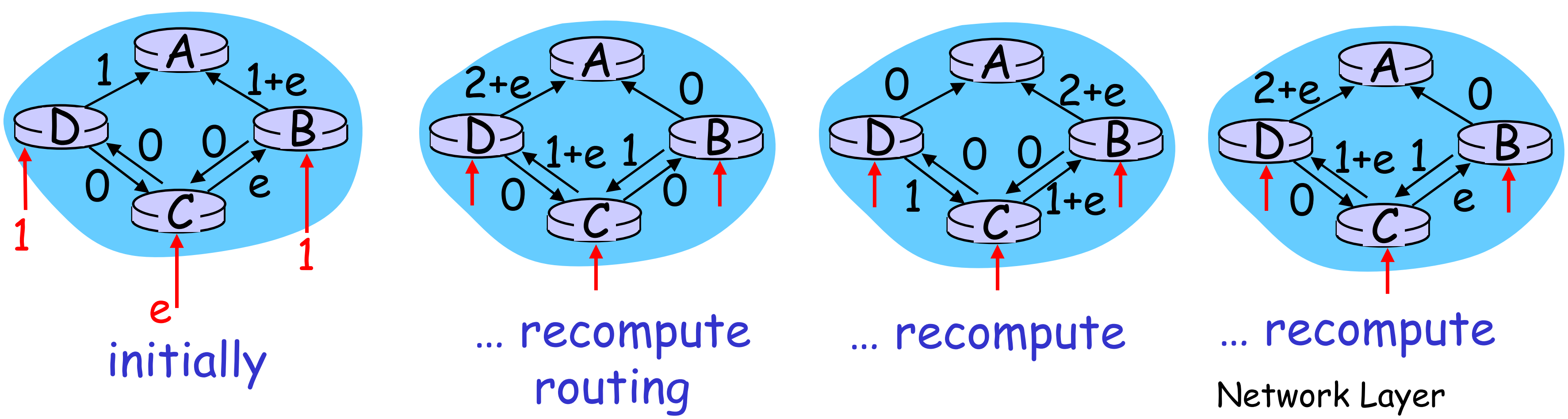
Dijkstra's algorithm, discussion

Algorithm complexity: n nodes

- each iteration: need to check all nodes, w, not in N
- $$\sum_{i=1}^{n-1} n-i = \frac{n(n+1)}{2} = O(n^2)$$
- more efficient implementations possible: O(nlogn)

Oscillations possible:

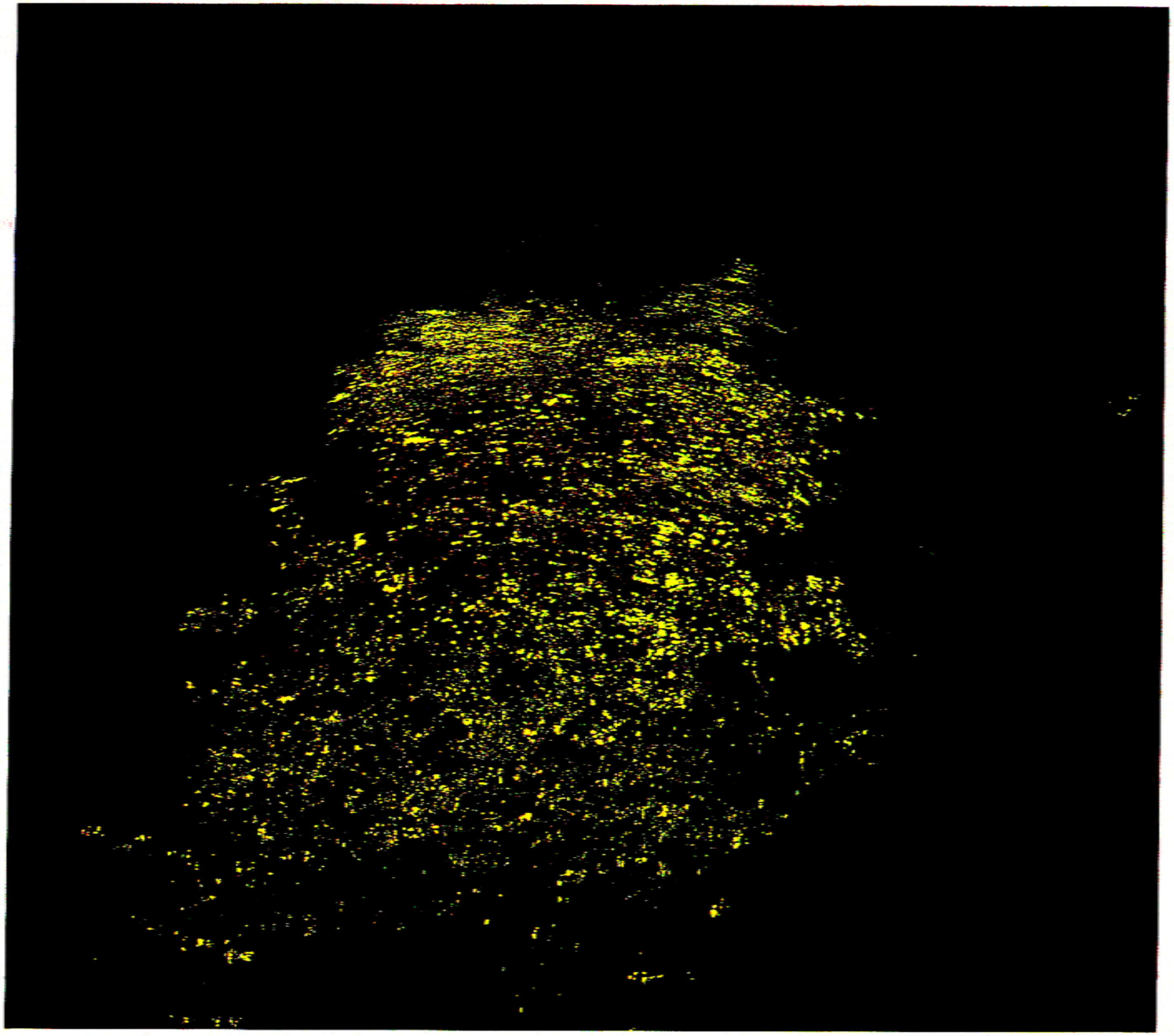
- e.g., if link cost = amount of carried traffic



Spontaneous synchronization

- To avoid oscillations make the routers recompute&send the link costs at different times?
- Turns out that if the recomputation periodicity is more or less the same on all routers then they eventually synchronize their execution times!
- The phenomenon of spontaneous synchronization occurs in physics, biology, chemistry, sociology, medicine, etc.

[redacted]



Shortest Path Routing Summary

Each router does the following:

- Discover its neighbors, learn their network address and UP state (HELLO message)
- Measure the delay or cost to each of its neighbors (ECHO message or cost function)
- Construct a packet telling what it knows (LS message)
- Send this packet to all other routers (every ROUTE REFRESH INTERVAL)
- Compute the shortest path to every other router (Dijkstra)

Shortest Path Routing Summary

Moreover:

- On every Link State change flood LS to all other routers
- Avoid oscillations through different periods
- Keep LS message counter to keep flooding in check
- Keep LS message age to keep counter in check
- Counter and age also used for fault tolerance