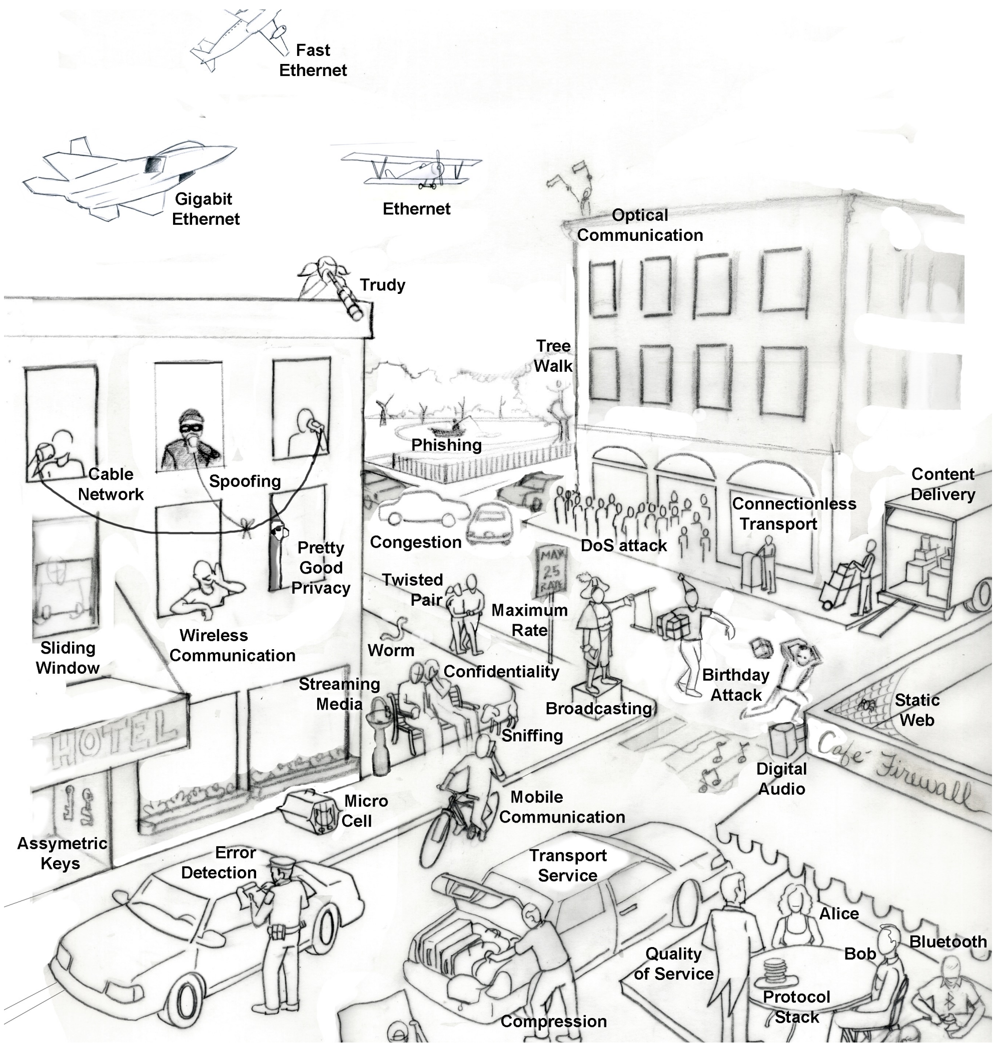
# Computer Networking

Sixth edition




## Chapter 5

### The Network Layer

Routing

(should be forwarding)

# A Word of Warning

- The Next 4 Lectures are Probably the Most Important
- This time we will be covering:
  - Routing
  - Traditional Network Layer
- And Next Time:
  - Naming and Addressing (by far the most important of the 4)
  - (Transport and Congestion will be after the mid-term.)
- Forward References are Unavoidable.
- So Hang on, here we go!

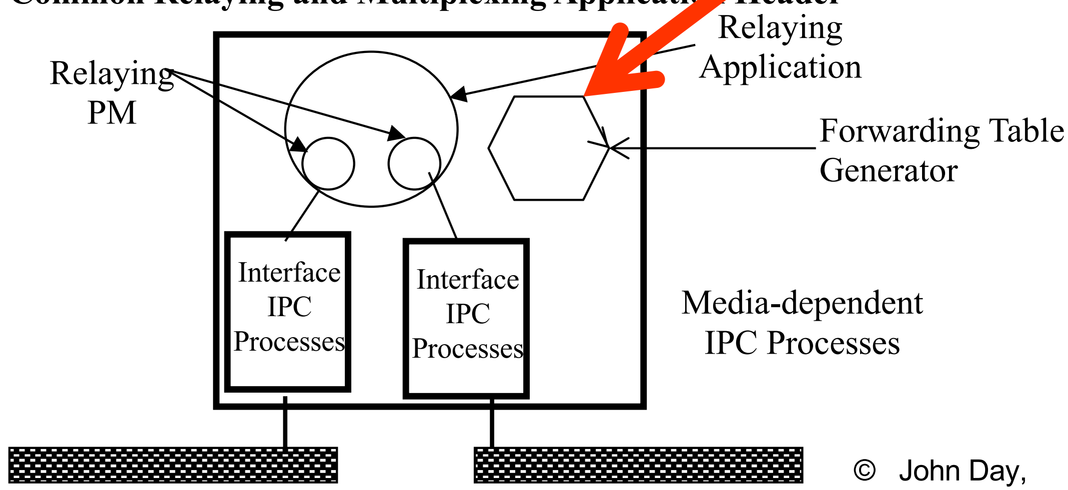
# Communications on the Cheap

- We will need systems dedicated to relaying and multiplexing.
- That requires some new elements:
  - A wire from a host no longer goes to just one place, therefore
  - Names for all IPC Processes in the Network.
  - Will need to add more information to the PDUs for the names of where the PDU is going.
  - A process that uses that information to determine, is this PDU's destination here or does it have to be forwarded and if so, where?
  - Need some way to figure out the forwarding table, traditionally this has been routing, which will need to exchange information on connectivity.

First, the Forwarding Table Generator

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Dest Addr | Src Addr | Dest-port | Src-port | Op | Seq # | CRC | Data |

**Common Relaying and Multiplexing Application Header**



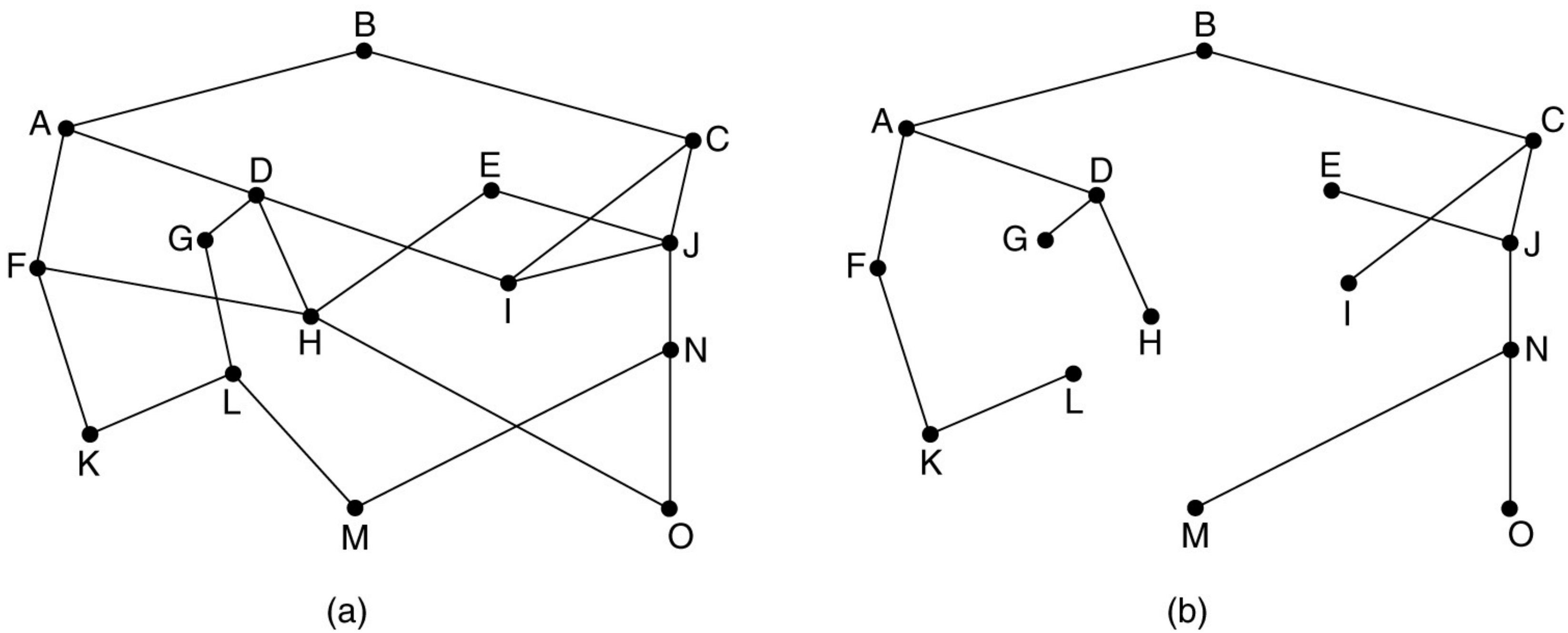
# Routing Algorithms

- The Optimality Principle
- Shortest Path Routing
- Flooding
- Distance Vector Routing
- Link State Routing
- Hierarchical Routing
- Broadcast Routing
- Multicast Routing
- [Routing for Mobile Hosts
- Routing in Ad Hoc Networks] . . . Later

# Why Do We Do Routing?

- Not to Determine the Route a PDU Follows
- But to build a Forwarding Table of Next Hops
- Here is what we do:
  - Each router exchanges routing information with others,
  - They all execute the *same algorithm*, with what we *hope* is the same or nearly the same data and
  - We *hope* they all get the consistent results for an optimal route through the network, for some value of ‘optimal.’
  - Then the routers build a forwarding table based on where they are in that solution.
- If all goes well, the PDU *does* follow an optimal route.
- If not, (very) weird things can happen.
- Why does this work?

# The Optimality Principle



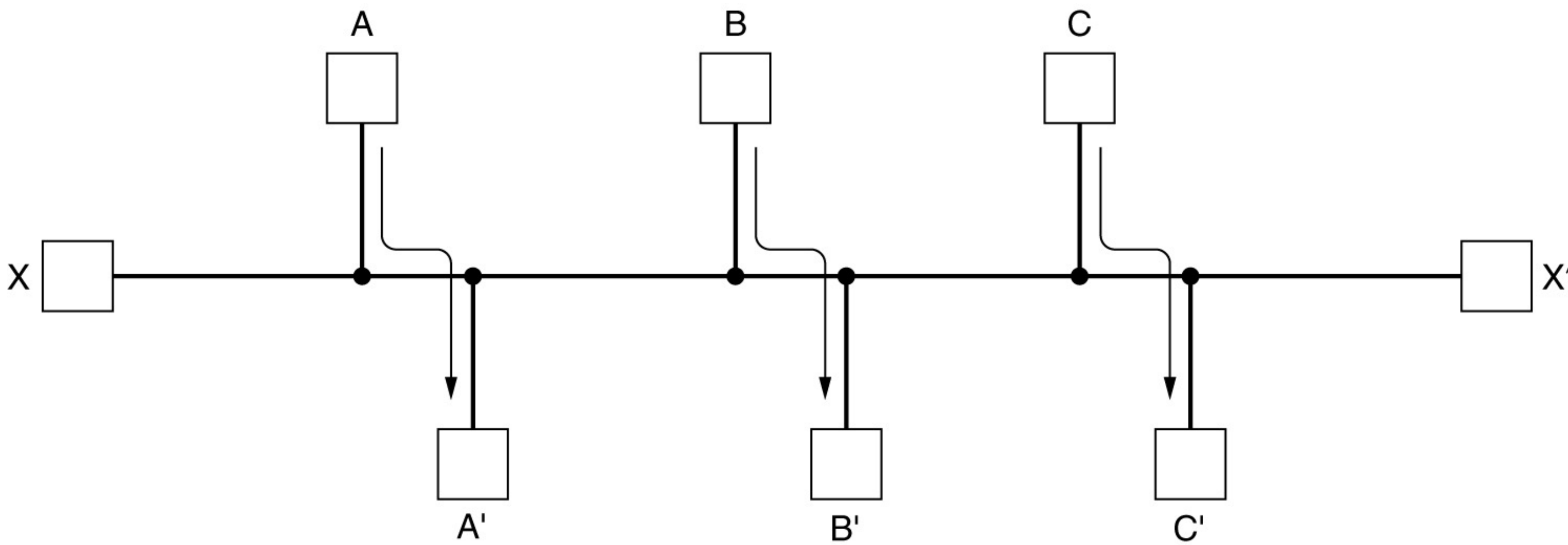

(a) A subnet. (b) A sink tree for router B.

Any sub-path of an optimal path is optimal.

# Taking Routing Apart

- All Routing Protocols Confuse, err *Combine*, 4 Separate Problems
  - Updating Multiple Copies of a Database
    - With Multiple sources for the same information
  - Determining Connectivity
    - Finding the routes
  - Choosing Routes based on Metrics
    - Weighting the routes according to various metrics
  - Create Forwarding Table
- Fundamentally 3 types of Routing (so far)
  - Link State - which requires complete knowledge of the graph
  - Distance Vector - which has partial knowledge
  - Hierarchical Routing - which tries to organize things
  - There are some new things under study.

# Routing Algorithms (2)



# Dijkstra's Algorithm (1)
## Definitions

- $n$ = set of vertices in network
- $s$ = source vertex (starting point)
- $T$ = set of vertices so far incorporated
- Tree = spanning tree for vertices in $T$ including edges on least-cost path from $s$ to each vertex in $T$
- $w(i,j)$ = link cost from vertex $i$ to vertex $j$
  - $w(i,i) = 0$
  - $w(i,j) = \infty$ if $i$, $j$ not directly connected by a single edge
  - $w(i,j) \geq 0$ if $i,j$ directly connected by single edge
- $L(n)$ = cost of least cost path from $s$ to $n$ currently known
  - At termination, this is least cost path from $s$ to $n$

Another source:
https://www.youtube.com/watch?v=pVfj6mxhdMw

From Stallings, **High Speed Networks and Internets**, 2002

# Dijkstra's Algorithm (2)

## Steps

- Initialization
  - T = Tree = {s} - only source is incorporated so far
  - $L(n) = w(s,n)$ for $n \neq s$ - initial path cost to neighbors are link costs
- Get next vertex
  - Find $x \notin T \ni L(x) = \min L(j),\ j \notin T$
  - Add x to T and Tree
  - Add edge to T incident on x and has least cost
    - Last hop in path
- Update least cost paths
  - $L(n) = \min[L(n), L(x) + w(x,n)]\ \forall\ n \notin T$
    - If latter term is minimum, path from s to n is now path from s to x concatenated with edge from x to n

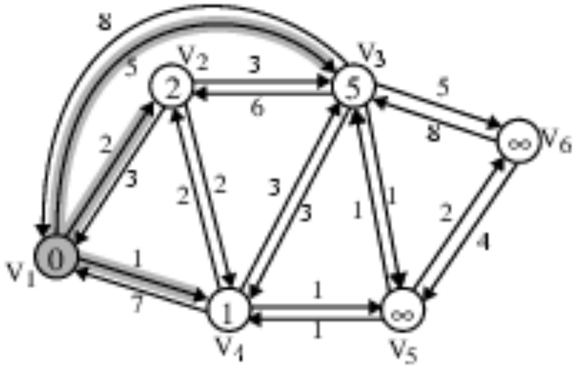
From Stallings, **High Speed Networks and Internets**, 2002

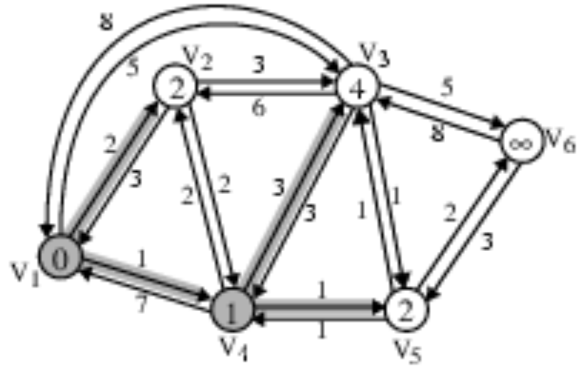
# Dijkstra's Algorithm (3)
## Notes

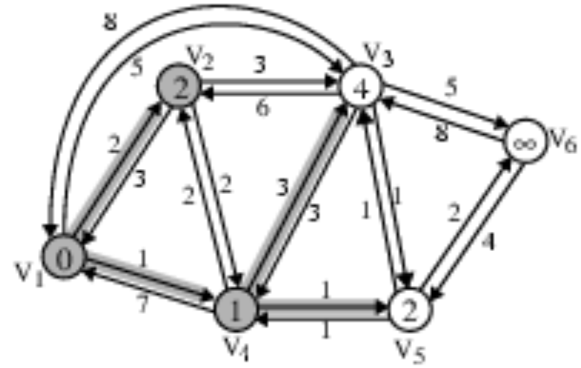
- Terminate when all vertices added to T
- Requires |V| iterations
- At termination
  - L(x) associated with each vertex is cost of least cost path from s to x
  - Tree is a spanning tree
    - Defines least cost path from s to each other vertex
- One step adds one vertex to T and defines least cost path from s to that vertex
- Running time $\Theta( |V|^2 )$
  - With Fibonacci Heaps $\Theta(|E| + |V| \log |V|)$

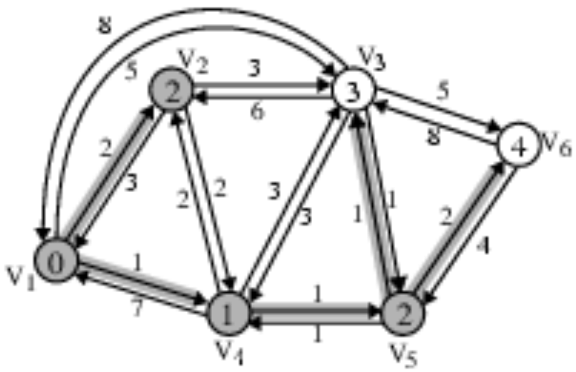
From Stallings, **High Speed Networks and Internets**, 2002

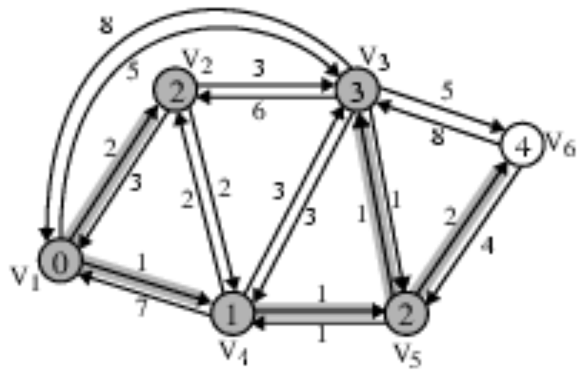
# Dijkstra’s Algorithm
## Example Graph

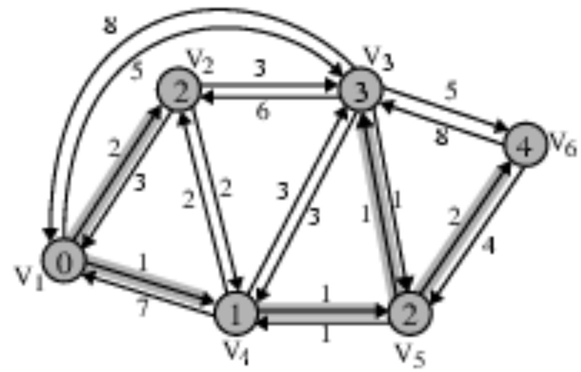

T = {1}


T = {1, 4}


T = {1, 2, 4}

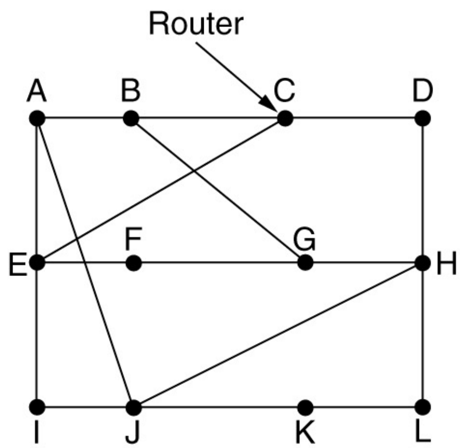

T = {1, 2, 4, 5}


T = {1, 2, 3, 4, 5}


T = {1, 2, 3, 4, 5, 6}

From Stallings, **High Speed Networks and Internets**, 2002

# Distance Vector Routing




(a)

| To | A | I | H | K | New estimated delay from J | Line |
|---|---|---|---|---|---|---|
| A | 0 | 24 | 20 | 21 | 8 | A |
| B | 12 | 36 | 31 | 28 | 20 | A |
| C | 25 | 18 | 19 | 36 | 28 | I |
| D | 40 | 27 | 8 | 24 | 20 | H |
| E | 14 | 7 | 30 | 22 | 17 | I |
| F | 23 | 20 | 19 | 40 | 30 | I |
| G | 18 | 31 | 6 | 31 | 18 | H |
| H | 17 | 20 | 0 | 19 | 12 | H |
| I | 21 | 0 | 14 | 22 | 10 | I |
| J | 9 | 11 | 7 | 10 | 0 | – |
| K | 24 | 22 | 22 | 0 | 6 | K |
| L | 29 | 33 | 9 | 9 | 15 | K |
| | JA delay is 8 | JI delay is 10 | JH delay is 12 | JK delay is 6 | | |

$\underbrace{\text{A, I, H, K}}_{\text{Vectors received from J's four neighbors}}$ $\underbrace{\text{delay, Line}}_{\text{New routing table for J}}$

(b)

(a) A subnet. (b) Input from A, I, H, K, and the new routing table for J.

# Bellman-Ford Algorithm

```
function bellmanFord(G, S)
  for each vertex V in G  do
    distance[V] <- infinite
    previous[V] <- NULL
  rof;
   distance[S] <- 0
 for each vertex V in G do
     for each edge (U,V) in G  do
          tempDistance <- distance[U] + edge_weight(U, V)
          if tempDistance < distance[V]  then
               distance[V] <- tempDistance
               previous[V] <- U
          fi
     rof
 rof
for each edge (U,V) in G
    if distance[U] + edge_weight(U, V) < distance[V}
            Error: Negative Cycle Exists
    fi
return distance[], previous[]
```

https://www.programiz.com/dsa/bellman-ford-algorithm

Running time $\Theta(|V| \times |E|)$
generally E > V

The algorithm is also used in financial transactions, where the edges can have negative weights, which require extra steps.

# Distance Vector Routing (2)

| A | B | C | D | E | |
|---|---|---|---|---|---|
| | • | • | • | • | Initially |
| | 1 | • | • | • | After 1 exchange |
| | 1 | 2 | • | • | After 2 exchanges |
| | 1 | 2 | 3 | • | After 3 exchanges |
| | 1 | 2 | 3 | 4 | After 4 exchanges |

(a)

| A | B | C | D | E | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | Initially |
| | 3 | 2 | 3 | 4 | After 1 exchange |
| | 3 | 4 | 3 | 4 | After 2 exchanges |
| | 5 | 4 | 5 | 4 | After 3 exchanges |
| | 5 | 6 | 5 | 6 | After 4 exchanges |
| | 7 | 6 | 7 | 6 | After 5 exchanges |
| | 7 | 8 | 7 | 8 | After 6 exchanges |
| | | ⋮ | | | |
| | • | • | • | • | |

(b)

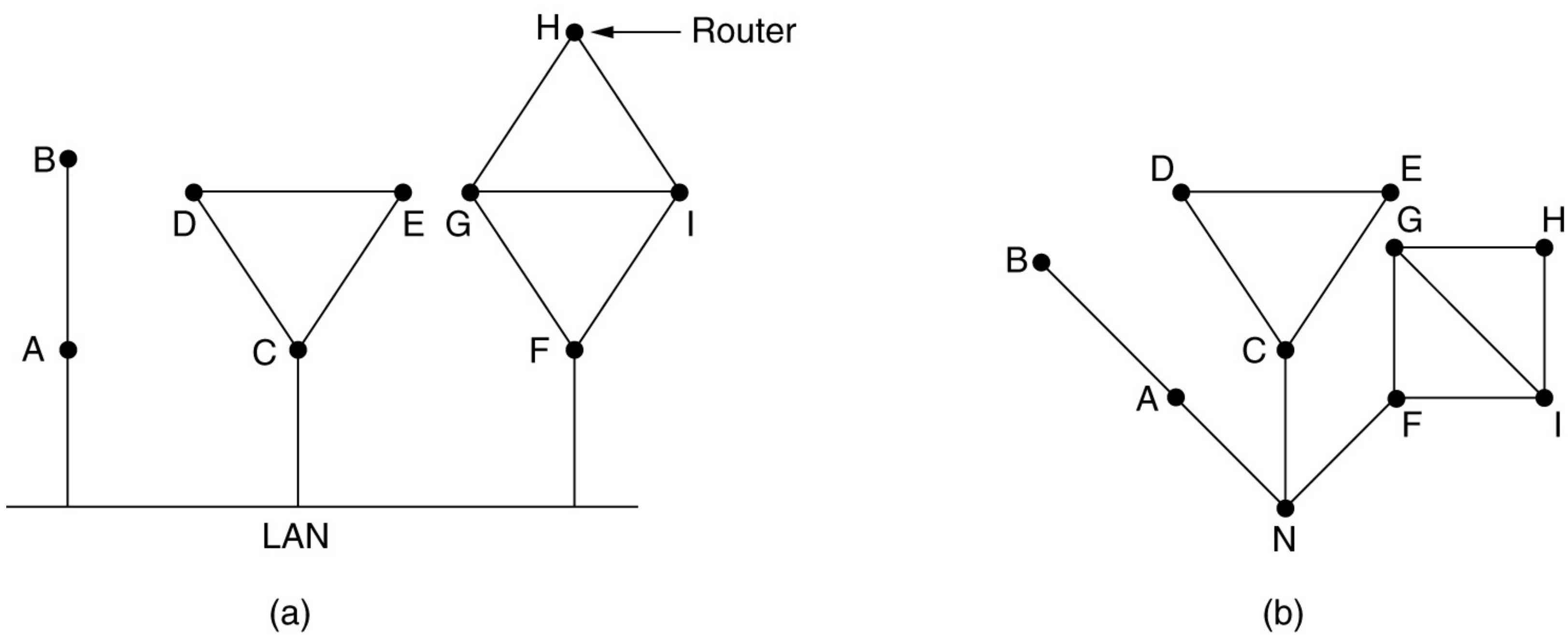
The count-to-infinity problem.

# Link State Routing

Each router must do the following:

1. Discover its neighbors, learn their network address.
2. Measure the delay or cost to each of its neighbors.
3. Construct a packet telling all it has just learned.
4. Send this packet to all other routers.
5. Compute the shortest path to every other router.

# Learning about the Neighbors

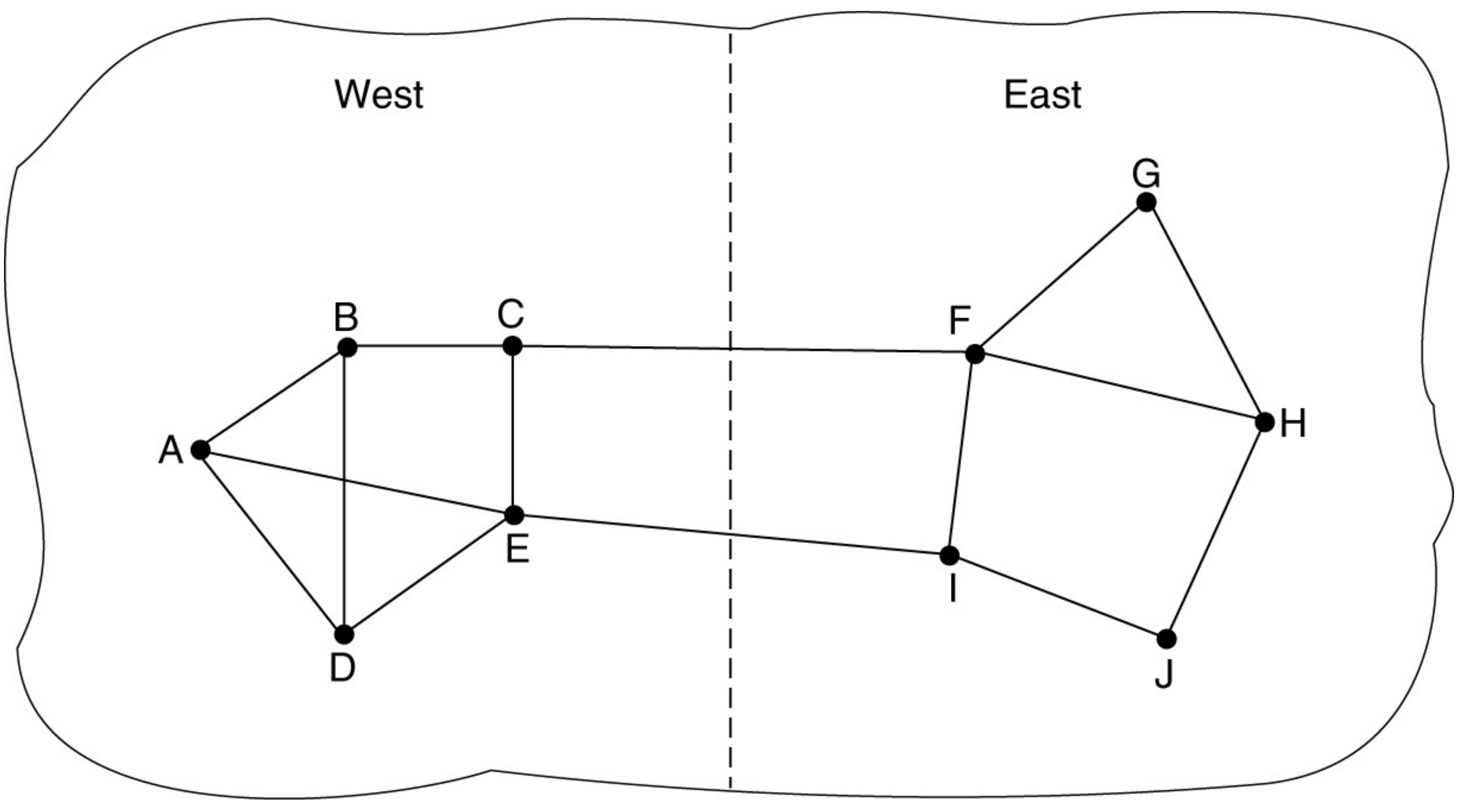



(a) Nine routers and a LAN. (b) A graph model of (a).

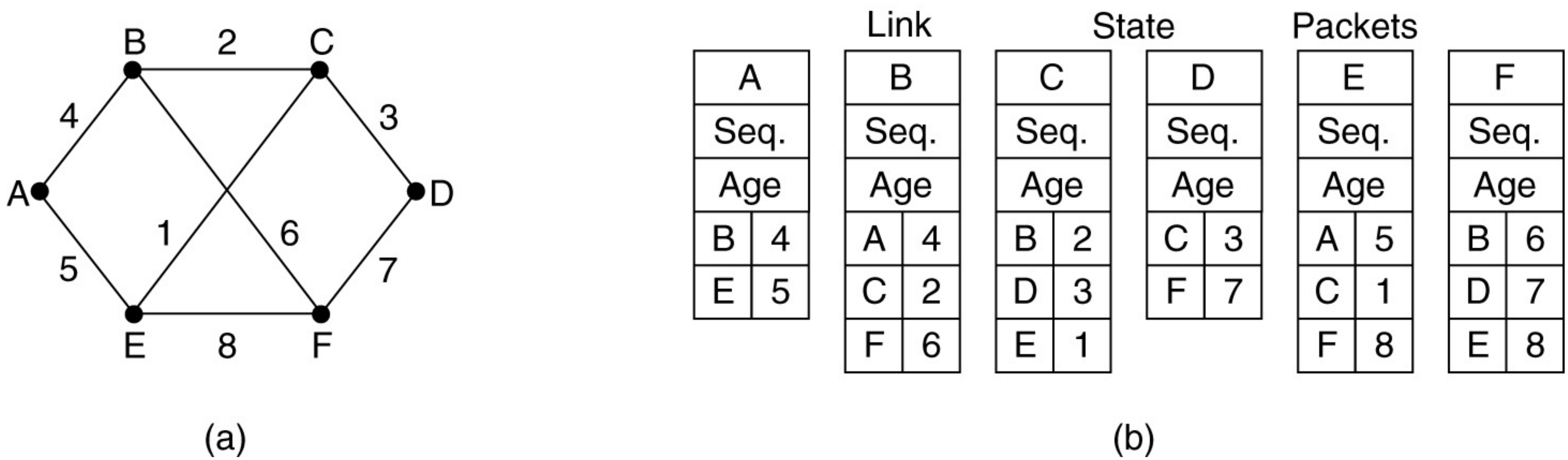
# Measuring Line Cost

A subnet in which the East and West parts are connected by two lines.

Picking the best tends to make it the worst.



# Building Link State Packets



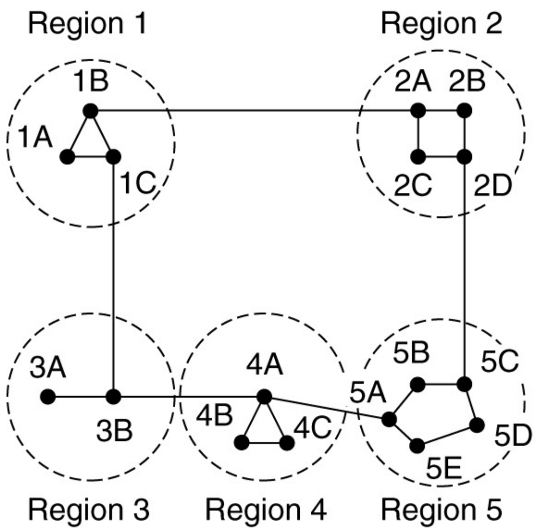

(a) A subnet. (b) The link state packets for this subnet.

# Distributing the Link State Packets

| | | | Send flags | | | ACK flags | | | |
|---|---|---|---|---|---|---|---|---|---|
| Source | Seq. | Age | A | C | F | A | C | F | Data |
| A | 21 | 60 | 0 | 1 | 1 | 1 | 0 | 0 | |
| F | 21 | 60 | 1 | 1 | 0 | 0 | 0 | 1 | |
| E | 21 | 59 | 0 | 1 | 0 | 1 | 0 | 1 | |
| C | 20 | 60 | 1 | 0 | 1 | 0 | 1 | 0 | |
| D | 21 | 59 | 1 | 0 | 0 | 0 | 1 | 1 | |

- Link State converges faster than Distance Vector
  - And to the same answer
  - And avoids the count to infinity problem and is less likely to develop loops.
  - Notice that neither of these algorithms use “addresses”
    - Only used as labels to keep track of nodes touched by the algorithm.
- However, Link-State doesn’t scale.
  - We can’t have every router in the Internet exchanging Link State PDUs with every other router!
- There is a two-tiered routing scheme:
  - Link State is used within in autonomous systems (subnets), called intra-domain.
  - A variation of Distance Vector is used among networks, called inter-domain.

# Hierarchical Routing



(a)

Full table for 1A

| Dest. | Line | Hops |
|---|---|---|
| 1A | – | – |
| 1B | 1B | 1 |
| 1C | 1C | 1 |
| 2A | 1B | 2 |
| 2B | 1B | 3 |
| 2C | 1B | 3 |
| 2D | 1B | 4 |
| 3A | 1C | 3 |
| 3B | 1C | 2 |
| 4A | 1C | 3 |
| 4B | 1C | 4 |
| 4C | 1C | 4 |
| 5A | 1C | 4 |
| 5B | 1C | 5 |
| 5C | 1B | 5 |
| 5D | 1C | 6 |
| 5E | 1C | 5 |

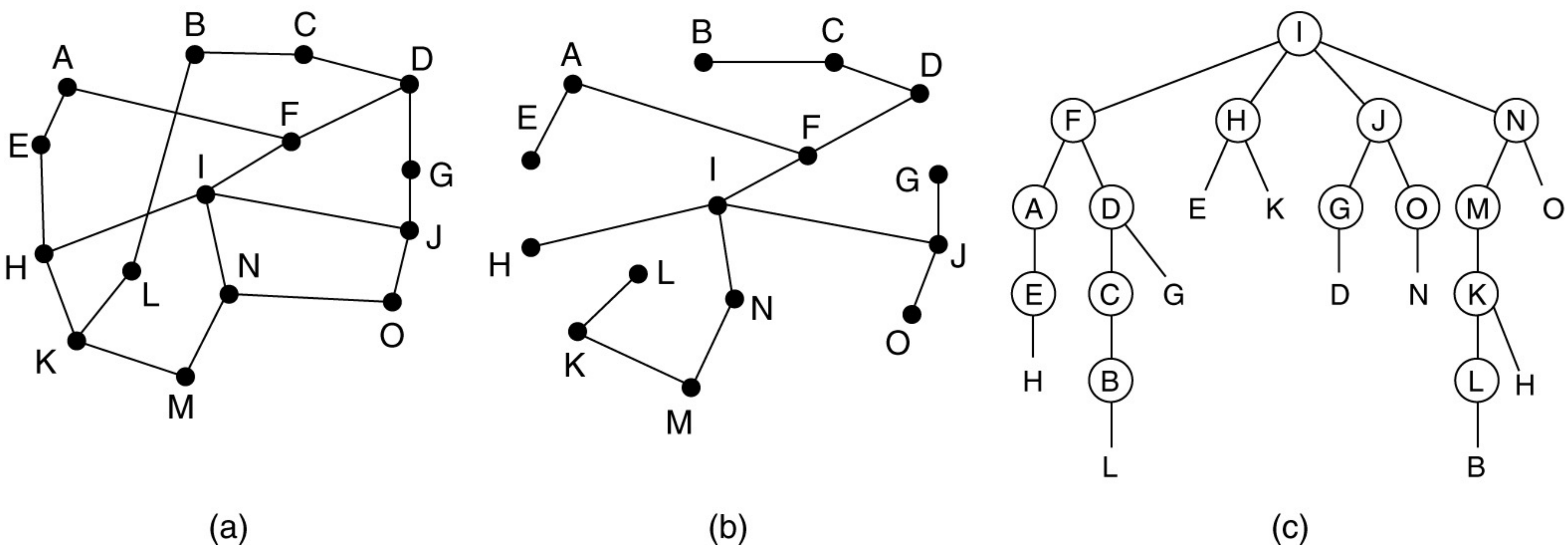
(b)

Hierarchical table for 1A

| Dest. | Line | Hops |
|---|---|---|
| 1A | – | – |
| 1B | 1B | 1 |
| 1C | 1C | 1 |
| 2 | 1B | 2 |
| 3 | 1C | 2 |
| 4 | 1C | 3 |
| 5 | 1C | 4 |

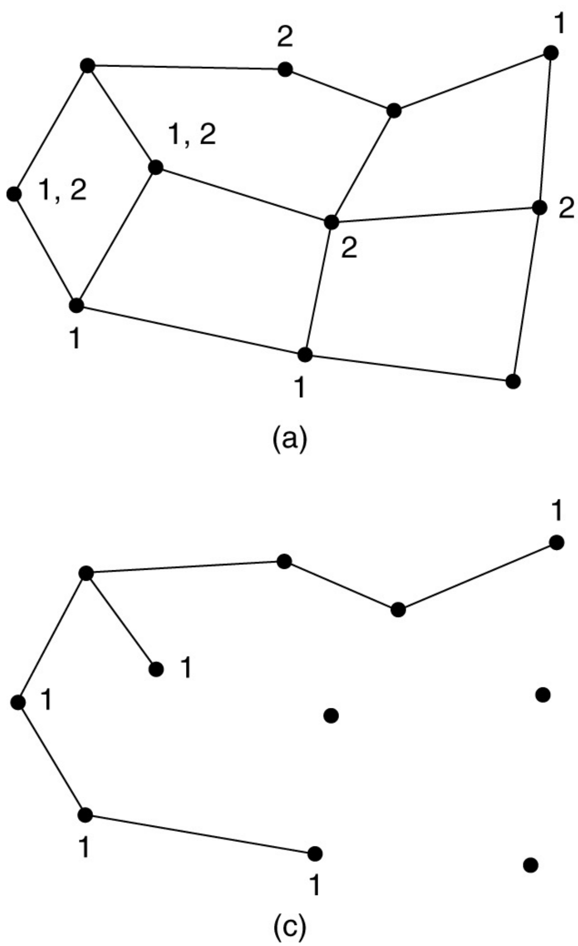
(c)

# Broadcast Routing



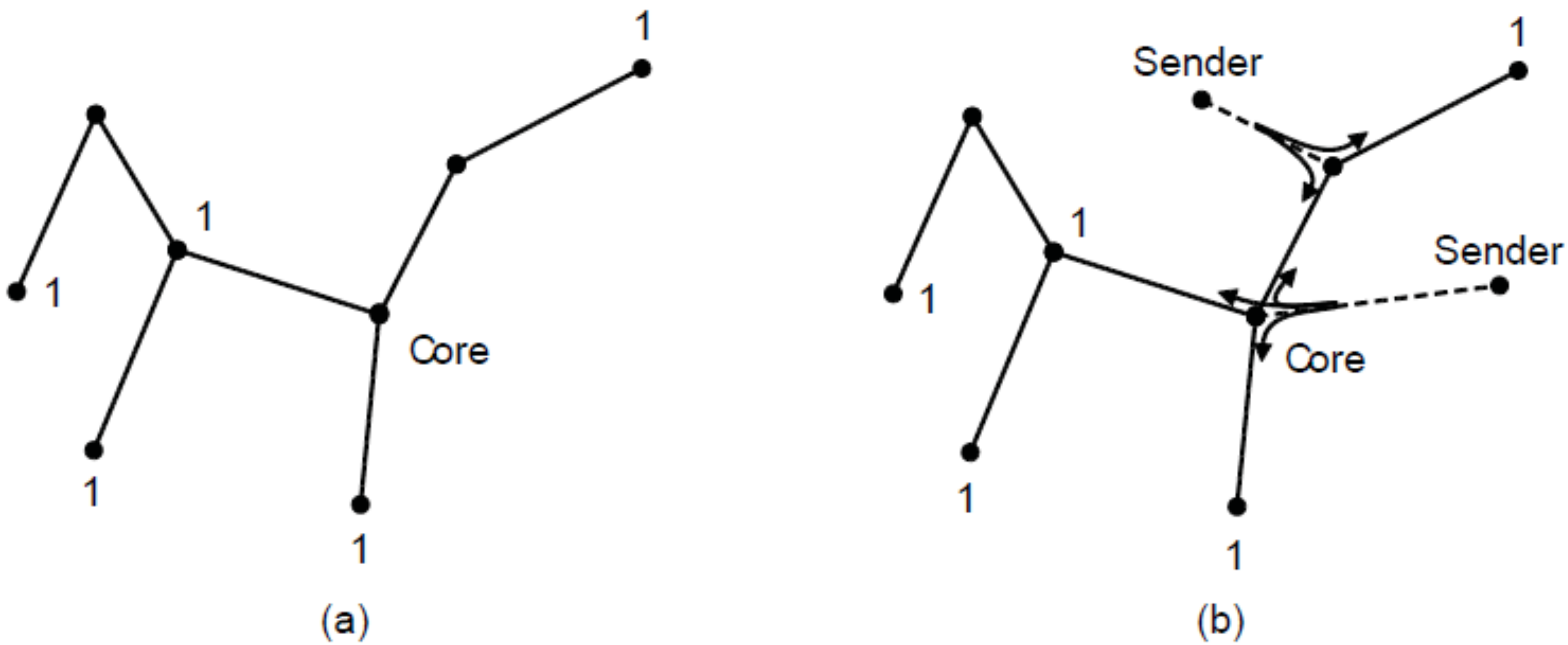

Reverse path forwarding. (a) A subnet. (b) a Sink tree. (c) The tree built by reverse path forwarding.

# Multicast Routing




(a) A network. (b) A spanning tree for the leftmost router.
(c) A multicast tree for group 1. (d) A multicast tree for group 2.

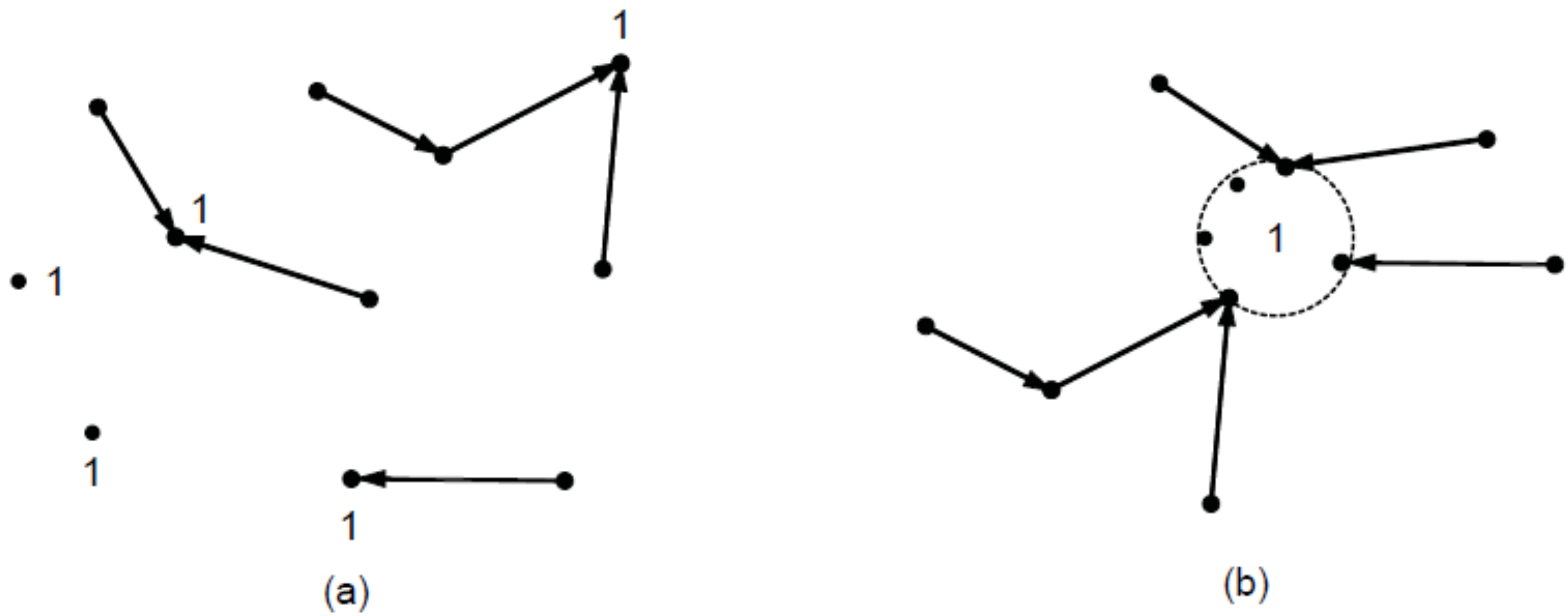
# Multicast Routing (2)




(a) Core-based tree for group 1.

(b) Sending to group 1.

# Anycast Routing




(a) Anycast routes to group 1.

(b) Graph seen by the routing protocol.

# The Truth about Multicast

- Multicast has little or no benefit to users.
  - Primary benefit is to the *provider*. Saves bandwidth
    - Does the provider choose better margins or pass the savings to user?
  - But is it worth the trouble? Requires a parallel routing scheme.
    - Not as long as there is a fiber glut.
- The user need never be aware of multicast.
  - The Interface interaction should be the same.
  - The important thing is to distinguish creating a multicast group (enrollment) from joining a multicast group (allocation).
  - If it is done right, it is easy. No current approaches do it right.
- Large number of variations
  - Is sender a member of the group, population static or dynamic, centralized/distributed, known/unknown population, etc.
  - And the bad news: Almost all combinations are realistic.

# More Truth about Multicast

- Multicast addresses aren't really addresses
    - Addresses should be location-dependent and route independent.
  - They can't be location-dependent in any meaningful way.
    - Not in general, but there is an interesting exception
  - In the Internet they have been ambiguous names. This practically requires a flooding or a parallel system. However, this is changing.
  - A multicast "address" is the name of a set such that referencing the set yields all members of the set.
- Anycast is the flip side of Multicast.
  - Multicast is $\forall$, while Anycast is $\exists$ (Sentential operators)
  - Anycast is defined by a set, a name for the set and a rule, such that when the name is referenced, the rule is applied to select one element of the set.

# The Real Truth

- Both are examples of a general case:  Whatevercast!
- Whatevercast name is the name of a set of addresses and a rule. Reference to the name causes the rule to be evaluated and returns one or more elements of the set.
  - When a forwarding table is generated, the rule is evaluated relative to this router and the resulting list is the entry in the forwarding table for this name.
  - Multicast - the rule returns all members
  - Anycast - the rule returns one member
  - But could also return something in between.
- There is a wide range of variations:

# What About Reliable Multicast?

- Hundreds of papers written about it. But still many problems.
- The Biggest Issue is "Ack Implosion."
  - As we have seen, Reliability requires feedback mechanisms:
    - Retransmission and Flow Control
  - Feedback by its nature is pairwise, which contradicts multicast.
    - A 1:N multicast is going to result in N:1 Acks and Credits.
  - For large N this is a huge problem,
  - and it generates potential inconsistency
    - What if some destinations get behind with Acks or Credits?
    - All Proposals answer this relative to specific applications
    - Some propose aggregating Acks and Credits at nodes of the spanning tree.
      - How long to wait for Acks or Credits to aggregate?
    - What about Watson's bounds? What do they imply?
      - All of the papers were written without knowledge of Watson.