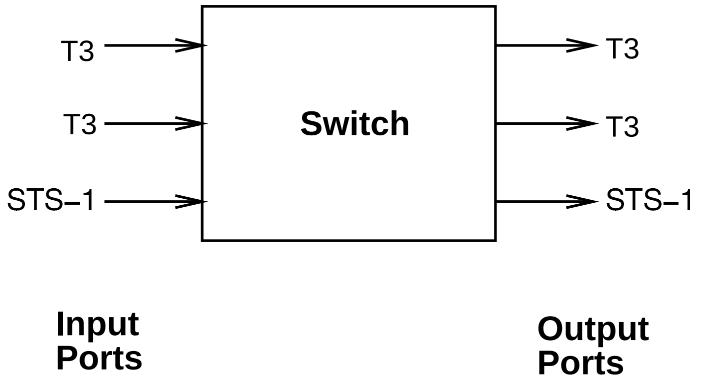
Switching and Forwarding

Scalable Networks

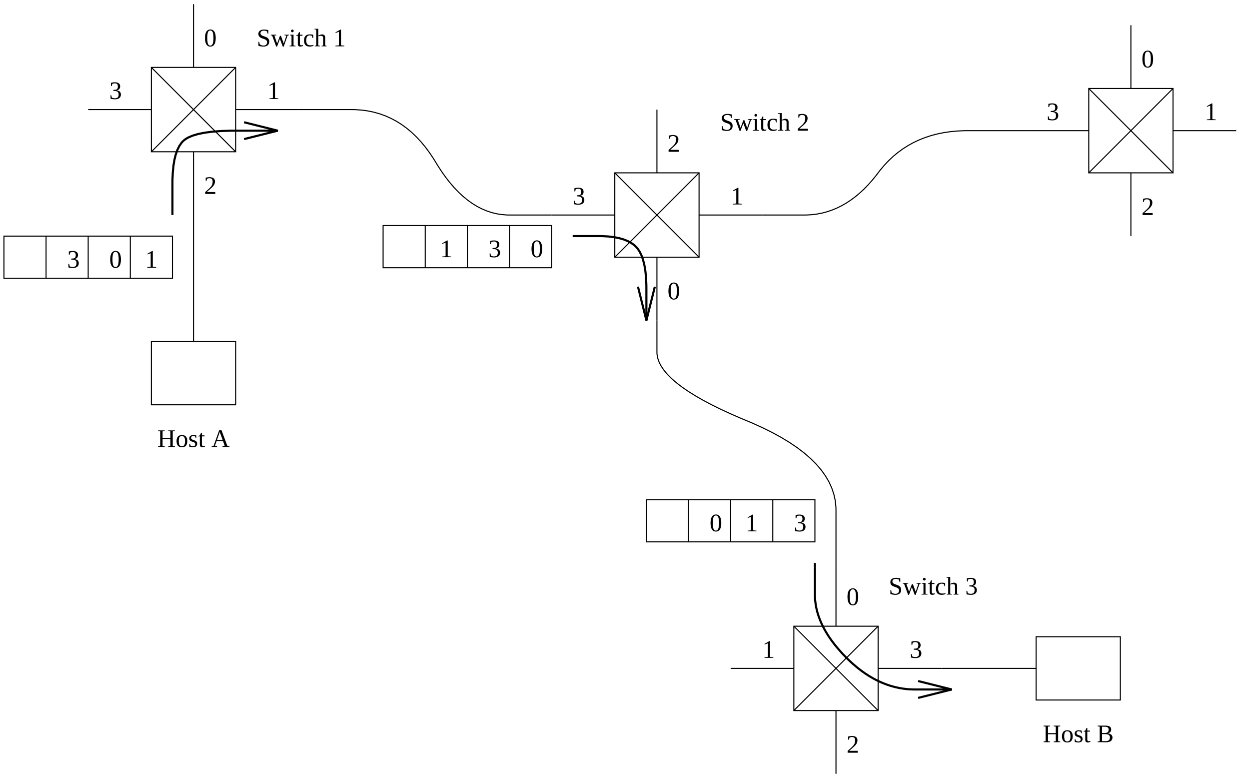
Switch: Forwards packets from input port to output port; port selected based on destination address in packet header.



- Can build networks that cover large geographic area
- Can build networks that support large numbers of hosts
- Can add new hosts without affecting performance of existing hosts

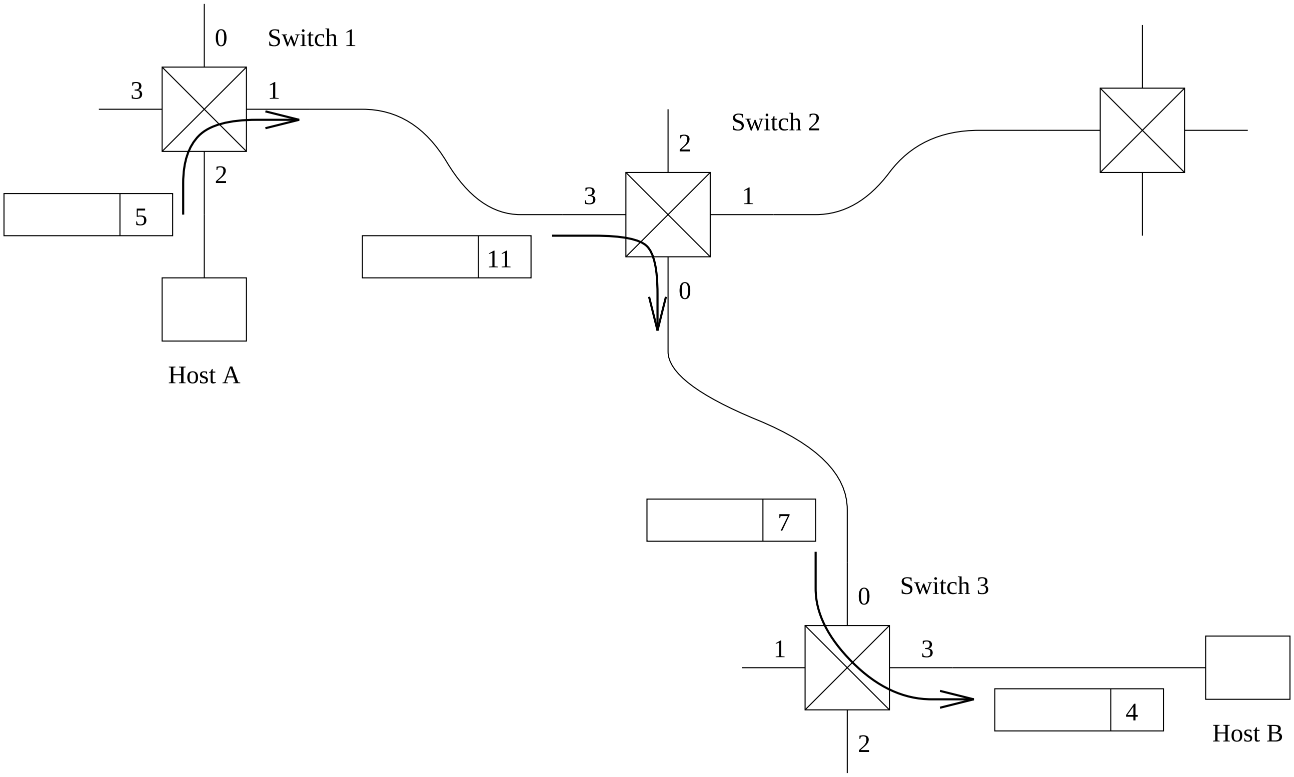
Source Routing

Address contains sequence of ports on path from source to destination.



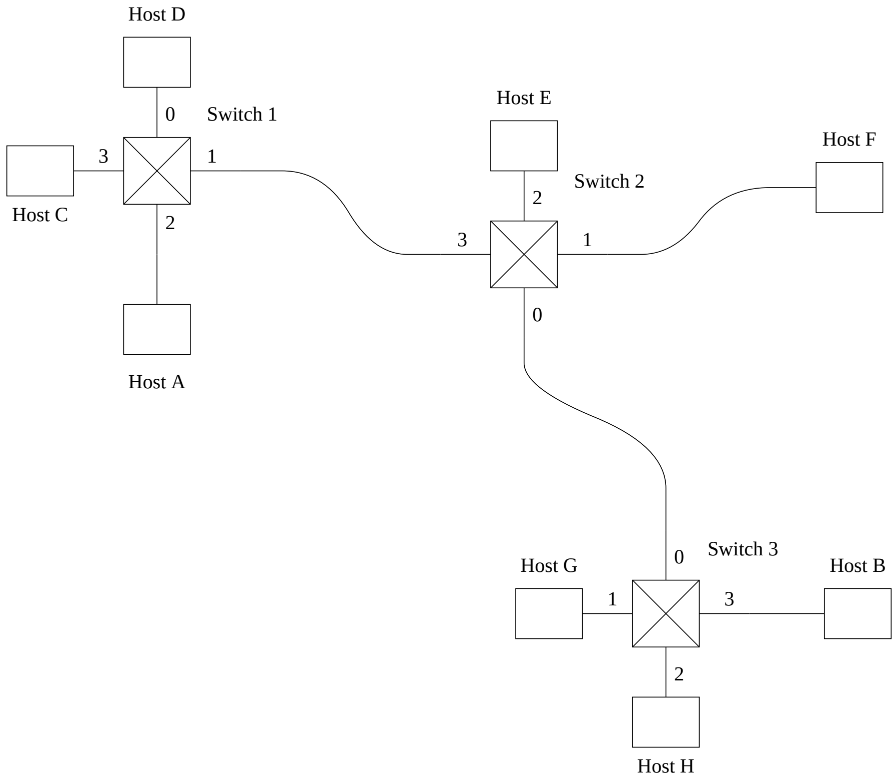
Virtual Circuit Switching

- Explicit connection setup (and tear-down) phase
- Subsequent packets follow same circuit
- Analogy: phone call
- Sometimes called *connection-oriented* model
- Each switch maintains a VC table.



Datagrams

- No connection setup phase
- Each packet forwarded independently
- Analogy: postal system
- Sometimes called *connectionless* model
- Each switch maintains a forwarding (routing) table



Virtual Circuit versus Datagram

Virtual Circuit Model:

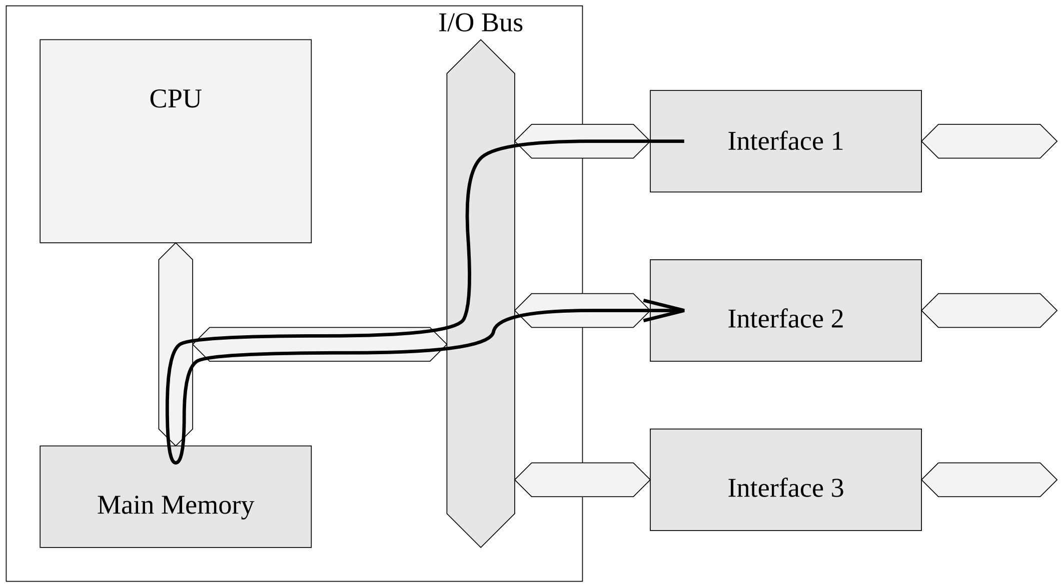
- Typically wait full RTT for connection setup before sending first data packet.
- While the connection request contains the full address for destination, each data packet contains only a small identifier, making the per-packet header overhead small.
- If a switch or a link in a connection fails, the connection is broken and a new one needs to be established.
- Connection setup provides an opportunity to reserve resources.

Datagram Model:

- There is no round trip time delay waiting for connection setup; a host can send data as soon as it is ready.
- Source host has no way of knowing if the network is capable of delivering a packet or if the destination host is even up.
- Since packets are treated independently, it is possible to route around link and node failures.
- Since every packet must carry the full address of the destination, the overhead per packet is higher than for the connection-oriented model.

Performance

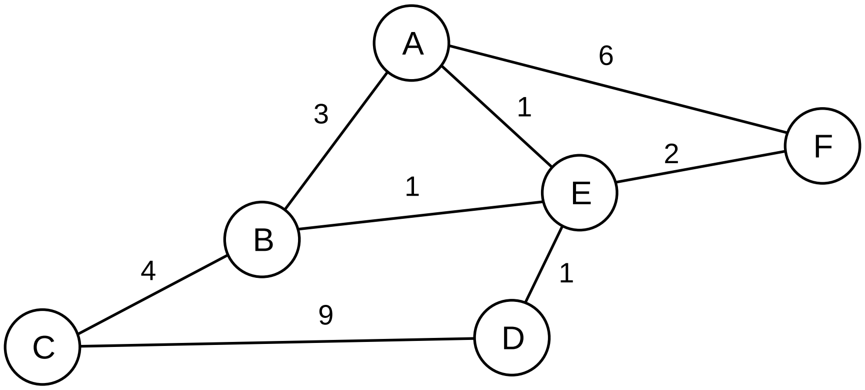
Switches can be built from a general-purpose workstations; will consider special-purpose hardware later.



- Aggregate bandwidth
  - – 1/2 of the I/O bus bandwidth
  - – capacity is shared among all hosts connected to switch
  - – example: 800Mbps bus can support 8 T3 ports
- Packets-per-second
  - – must be able to switch small packets
  - – 15,000 packets-per-second is an achievable number
  - – example: 64-byte packets implies 7.69Mbps

Routing

- Forwarding versus Routing
  - forwarding: to select an output port based on destination address and routing table
  - routing: process by which routing table is built
- Network as a Graph



- Problem: Find the lowest cost path between any two nodes
- Factors:
  - Static: topology
  - Dynamic: load

Distance Vector

- Each node maintains a set of triples:

  `(Destination, Cost, NextHop)`

- Each node sends updates to (and receives updates from) its directly connected neighbors
  - periodically (on the order of several seconds)
  - whenever its table changes (called *triggered* update)
- Each update is a list of pairs:

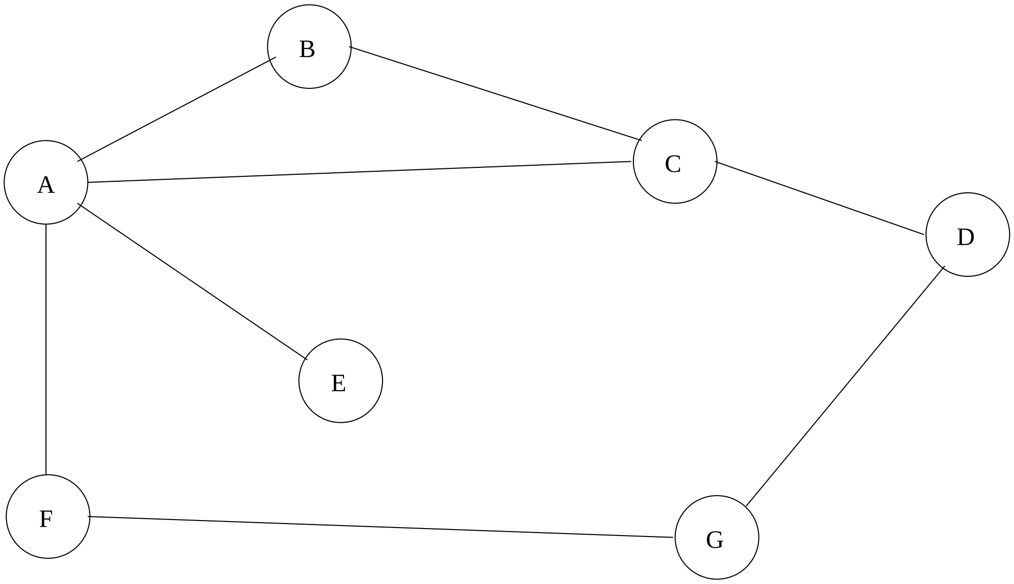
  `(Destination, Cost)`

- Update local table if receive a "better" route
  - smaller cost
  - came from next-hop
- Refresh existing routes; delete if they time out

```
void
mergeRoute (Route *new)
{
   int i;

   for (i = 0; i < numRoutes; ++i)
   {
      if (new->Dest == rt[i].Dest)
      {
         if (new->Cost + 1 < rt[i].Cost)
            break;
         else if (new->NextHop == rt[i].NextHop)
            break;
         else
            return;
      }
   rt[i] = *new;
   rt[i].TTL = MAX_TTL;
   ++rt[i].Cost;
   if (i == numRoutes)
      ++numRoutes;
}
```

Example



Routing table at node B

Destination	Cost	NextHop
A	1	A
C	1	C
D	2	C
E	2	A
F	2	A
G	3	A

Routing Loops

- Example 1
  - F detects that link to G has failed
  - F sets distance to G to infinity and sends update to A
  - A sets distance to G to infinity since it uses F to reach G
  - A receives periodic update from C with 2-hop path to G
  - A sets distance to G to 3 and sends update to F
  - F decides it can reach G in 4 hops via A
- Example 2
  - Link from A to E fails
  - A advertises distance of infinity to E
  - B and C advertise a distance of 2 to E
  - B decides it can reach E in 3 hops; advertises this to A
  - A decides it can reach E in 4 hops; advertises this to C
  - C decides that it can reach E in 5 hops......
- Heuristics to break routing loops
  - set infinity to 16
  - split horizon
  - split horizon with poison reverse

Link State

Strategy: Send to all nodes (not just neighbors) information about directly connected links (not entire routing table).

- Link State Packet (LSP)
  - id of the node that created the LSP
  - cost of link to each directly connected neighbor
  - sequence number (SEQNO)
  - time-to-live (TTL) for this packet
- Reliable Flooding
  - store most recent LSP from each node
  - forward LSP to all nodes but one that sent it
  - generate new LSP periodically; increment SEQNO
  - start SEQNO at 0 when reboot
  - decrement TTL of each stored LSP; discard when TTL=0

Route Calculation (in theory)

- Dijkstra's shortest path algorithm
- N denotes set of nodes in the graph
- $l(i, j)$ denotes non-negative cost (weight) for edge (i, j)
- $s \in N$ denotes this node
- M denotes the set of nodes incorporated so far
- $C(n)$ denotes cost of the path from s to node n

```
$M = \{s\}$
for each $n$ in $N - \{s\}$
    $C(n) = l(s, n)$
while $(N \neq M)$
    $M = M$ union $\{w\}$ such that $C(w)$
    is the minimum for all $w$ in $(N - M)$
    for each $n$ in $(N - M)$
        $C(n) = \mathtt{MIN}(C(n), C(w) + l(w, n))$
```

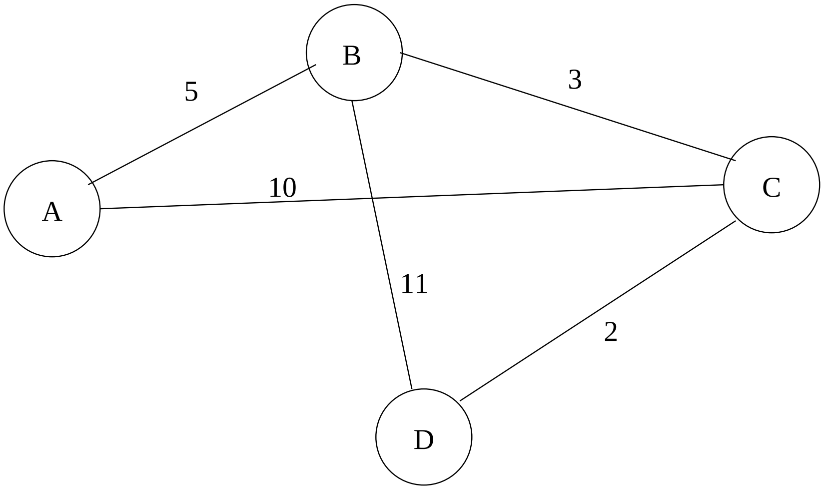
Route Calculation (in practice)

- Forward search algorithm
- Each switch maintains two lists:

  `Tentative` and **`Confirmed`**

- Each list contains a set of triples:

  `(Destination, Cost, NextHop)`



1. Initialized `Confirmed` with entry for me; cost = 0.
2. For the node just added to `Confirmed` (call it `Next`) select its LSP.
3. For each `Neighbor` of `Next`, calculate the `Cost` to reach this `Neighbor` as the sum of the cost from me to `Next` and from `Next` to `Neighbor`.
    - 3.1. If `Neighbor` is currently in neither `Confirmed` or `Tentative`, add `(Neighbor, Cost, NextHop)` to `Tentative`, where `NextHop` is the direction to reach `Next`.
    - 3.2. If `Neighbor` is currently in `Tentative` and `Cost` is less that current cost for `Neighbor`, then replace current entry with `(Neighbor, Cost, NextHop)`, where `NextHop` is the direction to reach `Next`.
4. If `Tentative` is empty, stop. Otherwise, pick entry from `Tentative` with the lowest cost, move it to `Confirmed`, and return to step 2.

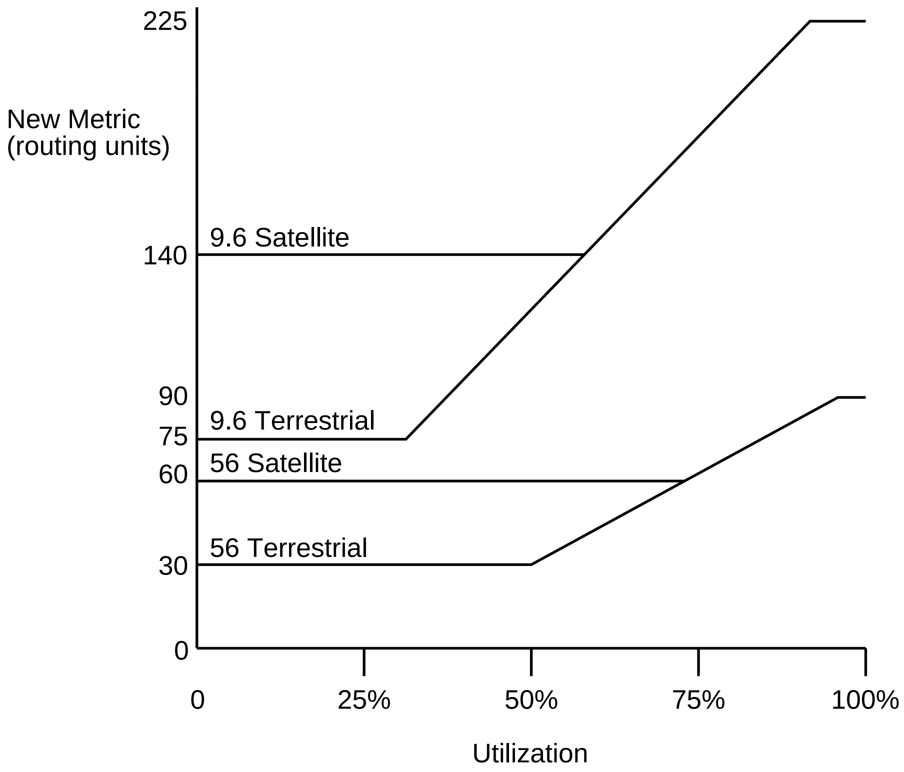
Step	**`Confirmed`**	**`Tentative`**
1.	(D,0,-)	
2.	(D,0,-)	(B,11,B) (C,2,C)
3.	(D,0,-) (C,2,C)	(B,11,B)
4.	(D,0,-) (C,2,C)	(B,5,C) (A,12,C)
5.	(D,0,-) (C,2,C) (B,5,C)	(A,12,C)
6.	(D,0,-) (C,2,C) (B,5,C)	(A,10,C)
7.	(D,0,-) (C,2,C) (B,5,C) (A,10,C)	

Metrics

- Original ARPANET metric
  - measured number of packets enqueued on each link
  - took neither latency or bandwidth into consideration
- New ARPANET metric
  - stamp each incoming packet with its arrival time (`AT`)
  - record departure time (`DT`)
  - when link-level ACK arrives, compute

    `Delay = (DT - AT) + Transmit + Latency`
  - if timeout, reset `DT` to departure time for retransmission
  - link cost = average delay over some time period

- Problems with “New” metric
  - under low load, static factors dominated cost; worked OK
  - under high load, congested links had very hight costs; packets oscillated between congested and idle links
  - range of costs too large; prefered path of 126 lightly loaded 56Kbps links to a 1-hop 9.6Kbps path
- Revised ARPANET metric
  - replaced delay measurement with link utilization
  - compressed dynamic range
    - ∗ highly loaded link never has a cost more than 3 times its idle cost
    - ∗ most expensive link only 7 times the cost of the least expensive
    - ∗ high-speed satellite link more attractive than low-speed terrestrial link
    - ∗ cost is a function of link utilization only at moderate to high loads.


225
New Metric
(routing units)
9.6 Satellite
140
90
9.6 Terrestrial
75
56 Satellite
60
56 Terrestrial
30
0
0
25%
50%
75%
100%
Utilization