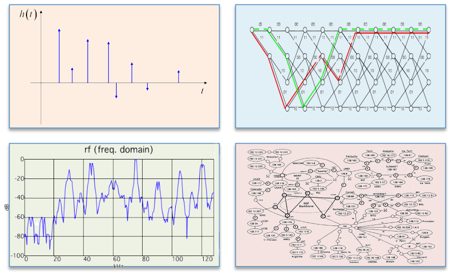


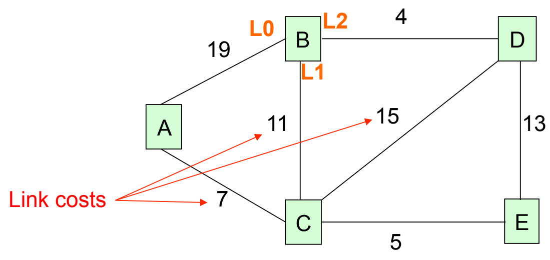
INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Spring 2011
Lecture #19

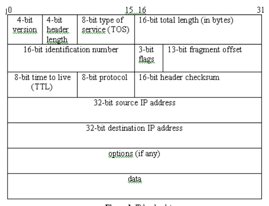
- addressing, forwarding, routing
- liveness, advertisements, integration
- distance-vector routing
- routing loops, counting to infinity

The Problem: Finding Paths



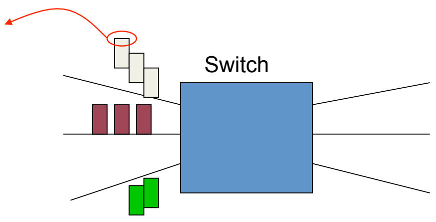
- Addressing (how to name nodes?)
  - Unique identifier for global addressing
  - Link name for neighbors
- Forwarding (how does a switch process a packet?)
- Routing (building and updating data structures to ensure that forwarding works)
- Functions of the *network layer*

Forwarding



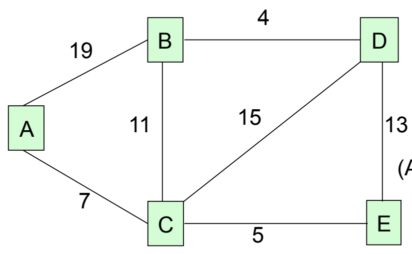
4-bit version	4-bit header length	8-bit type of service (TOS)	16-bit total length (in bytes)	
16-bit identification number			3-bit flags	13-bit fragment offset
8-bit time to live (TTL)		8-bit protocol	16-bit header checksum	
32-bit source IP address				
32-bit destination IP address				
options (if any)				
data				

Figure 1: IP header datagram



- Core function is conceptually simple
  - `lookup(dst_addr)` in routing table returns *route* (i.e., *outgoing link*) for packet
  - `enqueue(packet, link_queue)`
  - `send(packet)` along outgoing link
- And do some bookkeeping before enqueue
  - Decrement hop limit (TTL); if 0, discard packet
  - Recalculate checksum (in IP, header checksum)

Shortest Path Routing



(Assume all costs ≥ 0)

- Each node wants to find the path with *minimum total cost* to other nodes
  - We use the term "shortest path" even though we're interested in min cost (and not min #hops)
- Several possible distributed approaches
  - Vector protocols, esp. *distance vector* (DV)
  - *Link-state* protocols (LS)

Routing Table Structure

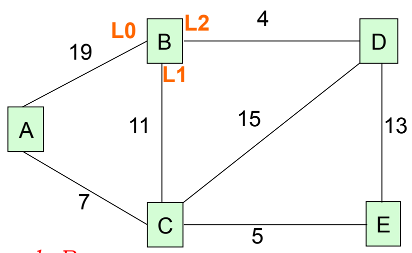


Table @ node B

Destination	*Link (next-hop)*	*Cost*
A	ROUTE L1	18
B	'Self'	0
C	L1	11
D	L2	4
E	L1	16

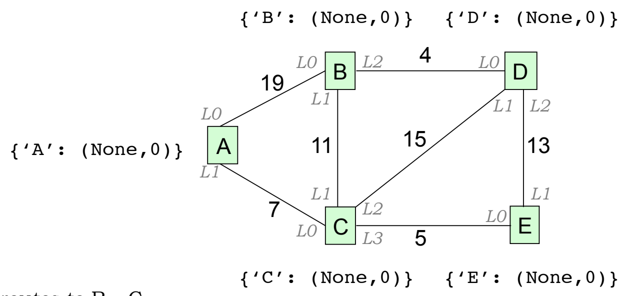
Distributed Routing: A Common Plan

- Determining live neighbors
  - Common to both DV and LS protocols
  - HELLO protocol (periodic)
    - Send HELLO packet to each neighbor to let them know who's at the end of their outgoing links
    - Use received HELLO packets to build a list of neighbors containing an information tuple for each link: (timestamp, neighbor addr, link)
    - Repeat periodically. Don't hear anything for a while → link is down, so remove from neighbor list.
- Advertisement step (periodic)
  - Send some information to all neighbors
  - Used to determine connectivity & costs to reachable nodes
- Integration step
  - Compute routing table using info from advertisements
  - Dealing with stale data

Distance-Vector Routing

- DV advertisement
  - Send info from routing table entries: (dest, cost)
  - Initially just (self,0)
- DV integration step [Bellman-Ford]
  - For each (dest,cost) entry in neighbor's advertisement
    - Account for cost to reach neighbor: (dest,my_cost)
    - my_cost = cost_in_advertisement + link_cost
  - Are we currently sending packets for dest to this neighbor?
    - See if link matches what we have in routing table
    - If so, update cost in routing table to be my_cost
  - Otherwise, is my_cost smaller than existing route?
    - If so, neighbor is offering a better deal! Use it…
    - update routing table so that packets for dest are sent to this neighbor

DV Example: round 1



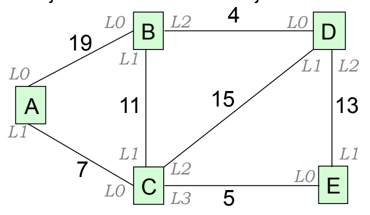
Node A: update routes to B_B, C_C
Node B: update routes to A_A, C_C, D_D
Node C: update routes to A_A, B_B, D_D, E_E
Node D: update routes to B_B, C_C, E_E
Node E: update routes to C_C, D_D

Subscript indicates node that gave better route

DV Example: round 2

```
{'A': (L0,19),     {'B': (L0,4),
 'B': (None,0),     'C': (L1,15),
 'C': (L1,11),      'D': (None,0),
 'D': (L2,4)        'E': (L2,13)
}                  }
```

```
{'A': (None,0),
 'B': (L0,19),
 'C': (L1,7)
}
```



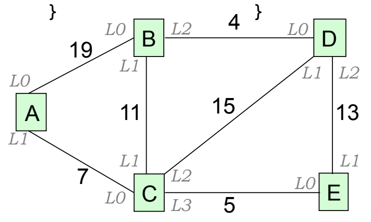
```
{'A': (L0,7),      {'C': (L0,5),
 'B': (L1,11),      'D': (L1,13),
 'C': (None,0),     'E': (None,0)
 'D': (L2,15),     }
 'E': (L3,5)
}
```

Node A: update routes to B_C, D_C, E_C
Node B: update routes to A_C, E_C
Node C: no updates
Node D: update routes to A_C
Node E: update routes to A_C, B_C

DV Example: round 3

```
{'A': (L1,18),     {'A': (L1,22),
 'B': (None,0),     'B': (L0,4),
 'C': (L1,11),      'C': (L1,15),
 'D': (L2,4),       'D': (None,0),
 'E': (L1,16)       'E': (L2,13)
}                  }
```

```
{'A': (None,0),
 'B': (L1,18),
 'C': (L1,7),
 'D': (L1,22),
 'E': (L1,12)
}
```



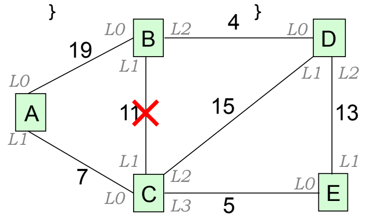
Node A: no updates
Node B: no updates
Node C: no updates
Node D: no updates
Node E: no updates

```
{'A': (L0,7),      {'A': (L0,12),
 'B': (L1,11),      'B': (L0,16),
 'C': (None,0),     'C': (L0,5),
 'D': (L2,15),      'D': (L1,13),
 'E': (L3,5)        'E': (None,0)
}                  }
```

DV Example: Break a Link

```
{~~'A': (L1,18),~~  {'A': (L1,22),
 'B': (None,0),     'B': (L0,4),
 ~~'C': (L1,11),~~   'C': (L1,15),
 'D': (L2,4),       'D': (None,0),
 ~~'E': (L1,16)~~    'E': (L2,13)
}                  }
```

```
{'A': (None,0),
 'B': (L1,18),
 'C': (L1,7),
 'D': (L1,22),
 'E': (L1,12)
}
```



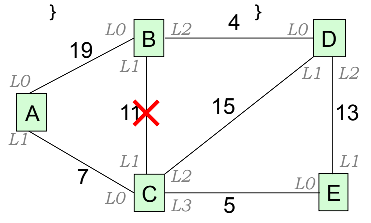
When link breaks: eliminate routes that use that link.

```
{'A': (L0,7),      {'A': (L0,12),
 ~~'B': (L1,11),~~   'B': (L0,16),
 'C': (None,0),     'C': (L0,5),
 'D': (L2,15),      'D': (L1,13),
 'E': (L3,5)        'E': (None,0)
}                  }
```

DV Example: round 4

```
{'A': (None,∞),    {'A': (L1,22),
 'B': (None,0),     'B': (L0,4),
 'C': (None,∞),     'C': (L1,15),
 'D': (L2,4),       'D': (None,0),
 'E': (None,∞)      'E': (L2,13)
}                  }
```

```
{'A': (None,0),
 'B': (L1,18),
 'C': (L1,7),
 'D': (L1,22),
 'E': (L1,12)
}
```



Node A: update cost to B_C
Node B: update routes to A_A, C_D, E_D
Node C: update routes to B_D
Node D: no updates
Node E: update routes to B_D

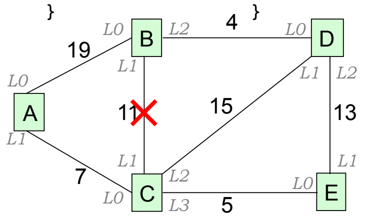
```
{'A': (L0,7),      {'A': (L0,12),
 'B': (None,∞),     'B': (L0,16),
 'C': (None,0),     'C': (L0,5),
 'D': (L2,15),      'D': (L1,13),
 'E': (L3,5)        'E': (None,0)
}                  }
```

DV Example: round 5

```
{'A': (L0,19),     {'A': (L1,22),
 'B': (None,0),     'B': (L0,4),
 'C': (L2,19),      'C': (L1,15),
 'D': (L2,4),       'D': (None,0),
 'E': (L2,17)       'E': (L2,13)
}                  }
```

Update cost

```
{'A': (None,0),
 'B': (L1, ∞),
 'C': (L1,7),
 'D': (L1,22),
 'E': (L1,12)
}
```



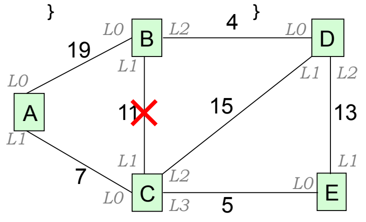
```
{'A': (L0,7),      {'A': (L0,12),
 'B': (L2,19),      'B': (L1,17),
 'C': (None,0),     'C': (L0,5),
 'D': (L2,15),      'D': (L1,13),
 'E': (L3,5)        'E': (None,0)
}                  }
```

Node A: update route to B_B
Node B: no updates
Node C: no updates
Node D: no updates
Node E: no updates

DV Example: final state

```
{'A': (L0,19),     {'A': (L1,22),
 'B': (None,0),     'B': (L0,4),
 'C': (L2,19),      'C': (L1,15),
 'D': (L2,4),       'D': (None,0),
 'E': (L2,17)       'E': (L2,13)
}                  }
```

```
{'A': (None,0),
 'B': (L0,19),
 'C': (L1,7),
 'D': (L1,22),
 'E': (L1,12)
}
```



```
{'A': (L0,7),      {'A': (L0,12),
 'B': (L2,19),      'B': (L1,17),
 'C': (None,0),     'C': (L0,5),
 'D': (L2,15),      'D': (L1,13),
 'E': (L3,5)        'E': (None,0)
}                  }
```

Node A: no updates
Node B: no updates
Node C: no updates
Node D: no updates
Node E: no updates

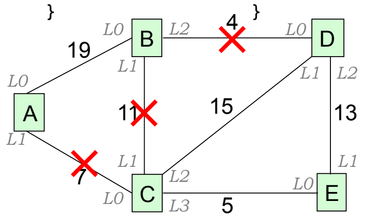
Correctness & Performance

- Optimal substructure property fundamental to correctness of both Bellman-Ford and Dijkstra's shortest path algorithms
  - ***Suppose shortest path from X to Y goes through Z. Then, the sub-path from X to Z must be a shortest path.***
- Proof of Bellman-Ford via induction on number of walks on shortest (min-cost) paths
  - Easy when all costs > 0 and *synchronous model* (see notes)
  - Harder with distributed async model (not in 6.02)
- How long does it take for distance-vector routing protocol to *converge*?
  - Time proportional to largest number of hops considering all the min-cost paths

Partitioning the Network

```
{'A': (L0,19),     {'A': (L1,22),
 'B': (None,0),     'B': (None,∞),
 'C': (None,∞),     'C': (L1,15),
 'D': (None,∞),     'D': (None,0),
 'E': (None,∞)      'E': (L2,13)
}                  }
```

```
{'A': (None,0),
 'B': (L0,19),
 'C': (None,∞),
 'D': (None,∞),
 'E': (None,∞)
}
```



```
{'A': (None,∞),    {'A': (L0,12),
 'B': (L2,19),      'B': (L1,17),
 'C': (None,0),     'C': (L0,5),
 'D': (L2,15),      'D': (L1,13),
 'E': (L3,5)        'E': (None,0)
}                  }
```

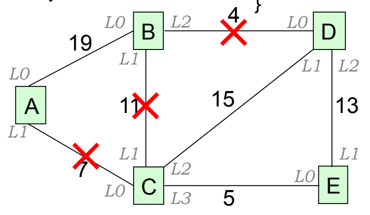
Node A: delete routes to C, D, E
Node B: delete routes to C, D, E
Node C: update routes to A_E, B_E
Node D: update routes to A_E, B_E
Node E: update route to A_D, cost to B_D

DV Example: round 6

```
{'A': (L2,25),
 'B': (L2,30),
 'C': (L1,15),
 'D': (None,0),
 'E': (L2,13)
}
```

```
{'A': (L0,19),
 'B': (None,0)
}
```

```
{'A': (None,0),
 'B': (L0,19)
}
```



Node A: no updates
Node B: no updates
Node C: update costs to A_E, B_E
Node D: update route to A_C, cost to B_E
Node E: update routes to A_C, B_C

```
{'A': (L3,17),
 'B': (L3,22),
 'C': (None,0),
 'D': (L2,15),
 'E': (L3,5)
}
```

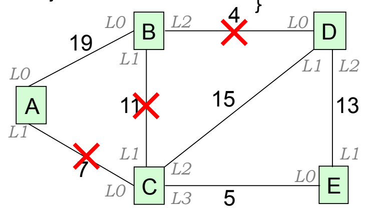
```
{'A': (L1,35),
 'B': (None,∞),
 'C': (L0,5),
 'D': (L1,13),
 'E': (None,0)
}
```

Counting to Infinity

```
{'A': (L2,38),
 'B': (None,∞),
 'C': (L1,15),
 'D': (None,0),
 'E': (L2,13)
}
```

```
{'A': (L0,19),
 'B': (None,0)
}
```

```
{'A': (None,0),
 'B': (L0,19)
}
```



Nodes C, D, and E each update their costs in response to earlier updates by neighbors. Costs spiral upwards towards ∞!

remove route when cost reaches self.INFINITY

```
{'A': (L3,40),
 'B': (None,∞),
 'C': (None,0),
 'D': (L2,15),
 'E': (L3,5)
}
```

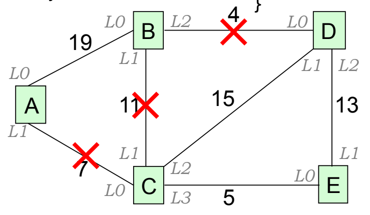
```
{'A': (L0,22),
 'B': (L0,27),
 'C': (L0,5),
 'D': (L1,13),
 'E': (None,0)
}
```

Routing Loop!

```
{'A': (L2,38),
 'B': (None,∞),
 'C': (L1,15),
 'D': (None,0),
 'E': (L2,13)
}
```

```
{'A': (L0,19),
 'B': (None,0)
}
```

```
{'A': (None,0),
 'B': (L0,19)
}
```



Suppose E sends a packet to A:

- E forwards to C
- C forwards to E
- … repeat …
- Drop packet when TTL is decremented to 0

```
{'A': (L3,40),
 'B': (None,∞),
 'C': (None,0),
 'D': (L2,15),
 'E': (L3,5)
}
```

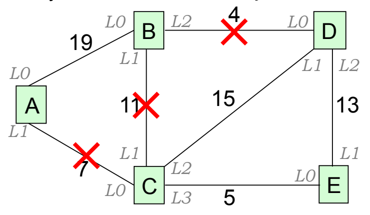
```
{'A': (L0,22),
 'B': (L0,27),
 'C': (L0,5),
 'D': (L1,13),
 'E': (None,0)
}
```

Eventual Final State

```
{'C': (L1,15),
 'D': (None,0),
 'E': (L2,13)
}
```

```
{'A': (L0,19),
 'B': (None,0)
}
```

```
{'A': (None,0),
 'B': (L0,19)
}
```



Eventually all the unreachable nodes are removed from routing table and all routing loops are resolved.

```
{'C': (None,0),
 'D': (L2,15),
 'E': (L3,5)
}
```

```
{'C': (L0,5),
 'D': (L1,13),
 'E': (None,0)
}
```