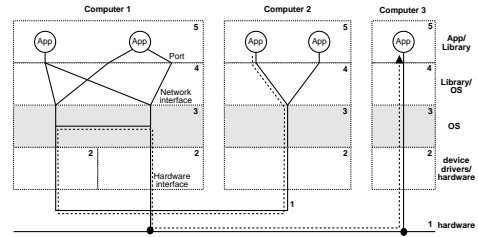


## Lecture 6 Network layer

### Review: where is network layer, functionality



Network layer (Layer 3) is the **routing layer**, allowing communication between any arbitrary nodes connected by communication channels.

Data link (in particular, switches) can do some routing, but their routing is brute-force, and can only deal with 1 technology.

#### References:

- CN Sections 5.1–5.2.

Network(0234B)

Network(0234B)-6.1

### Why network layer

- Computers are connected by various point-to-point or broadcast **physical media**, operated using **data link** and **MAC protocols**.
- Suppose we have a set of point-to-point channels connecting computers A–B, B–C, and C–D.
- For user of computer A to send a message to computer D, he can send a data link frame to computer B saying “*please tell computer C to send a this frame to computer D, telling that it’s my words*”.
- But this requires that computer A **knows all the intermediate links exactly**, which is very error-prone and inconvenient.
- Network layer is a layer of network software that **hide details of network topology**. User of computer A should just need to ask its network layer to “*send a network packet to computer D*”, and the network layer do the rest.

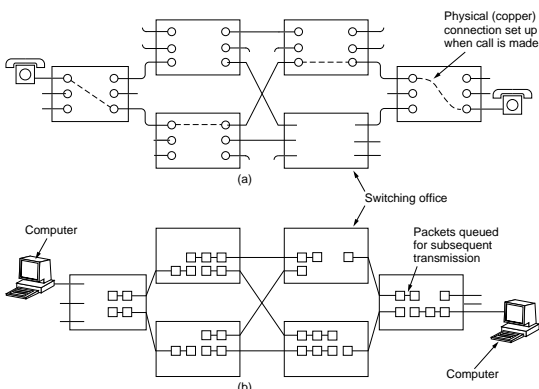
Network(0234B)-6.2

### Store and Forward

- **Each computer** of a network runs the network layer software.
- In old phone networks, a **physical** electrical path is made when one dial a number. This is done by configuring **switches** in the network, and the method is called **circuit switching**.
- This **wastes resources** when the path is not actually used, which is **always** the case for bursty computer conversations.
- Alternative: for the intermediate nodes (“**routers**”), to **store** the network data (“**network packet**”) when a frame containing it arrives.
- Once a packet arrives completely, and the router decides what to do with it, it puts it in a queue, waiting to copy, or “**forward**”, it to the next node.
- No physical connection is involved, so different packet can follow different paths: **packet switching**.

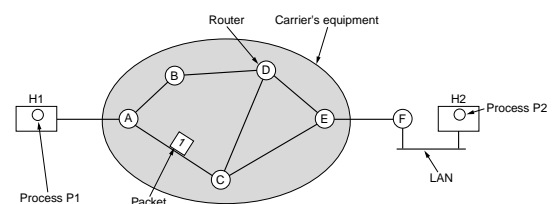
Network(0234B)-6.3

### Circuit switching and packet switching: cont



Network(0234B)-6.4

### Typical environment



- The carrier has **routers** that connect all the smaller LANs and hosts.  
Carrier = company providing this network service.
- Processes P1 and P2 talks the **network protocol** with each other, by talking the **data link protocol** to the routers.  
Or to the computer (F) connected to it in case of the LAN.
- Much of the network protocol works in the **carrier’s equipment**.

Network(0234B)-6.5

## Addressing

If H1 wants to talk to H2, a packet for H2 is sent from H1 to A...

**Problem 1:** how to name H2?

- We can't use data link address, as **some data links has no address**. Hardware address exists for ethernet, but not modems or lease lines.
- Some **universal address** must be given to each host and router. We call this **network address**, and is stored in the packet.

**Problem 2:** How A knows that to reach H2, the packet must arrive E?

- Keep a **list of host to router mappings** in all routers? Unattractive... Okay only if there are only a few hosts.  
And what to do when a new host is added to a LAN?
- H1 also name E when addressing H2? How H1 knows E is for H2?
- Internet: Make use of the network address (next chapter).

Network(0234B)-6.6

## What the router needs

Apart from the multiple data link connections to the right places...

- Enough **memory** to store the packet queued on links. When a frame arrives, the router (a computer) will receive an **interrupt**, and at that time the computer must give allocate a **new buffer** for it. If no buffer is available, the packet has to be removed.
- Enough **outgoing capacity** in data link. If output rate is smaller than input rate and the situation subtain for some time, **all queue buffer** will be used up no matter how much memory is available.
- **CPU cycles** to process the copying, to **decide** what to do with each packet. If this has to be delayed, capacity of output links will be lost.

Insufficient of any of these resources will lead to packets being discarded. It is always in the form of insufficient buffers, but the solution may or may not be adding more memory.

Network(0234B)-6.7

## Datagram vs. Virtual Circuit

Given the need for packet switching, there are still **two ways** to do it...

**Datagram** (Internet)

Each packet **stores the destination address** (as well as the source address). Routers consider **routing of each packet independently**.

Routers just keep tables about **how to get to each destination**.

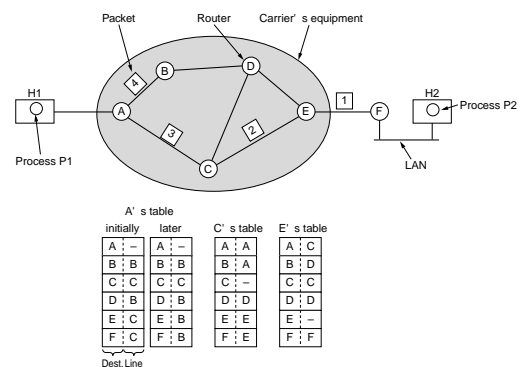
**Virtual Circuit (VC)** (ATM—used by phone companies)

A **setup phase** asks the source router to find a path to a destination. After that, each router is given a label, and **all future packets carries that label** to identify that "circuit". **No routing decision** is made from that point onwards.

In addition to the table like the datagram case, routers keep tables about **what circuits are created in it**.

Network(0234B)-6.8

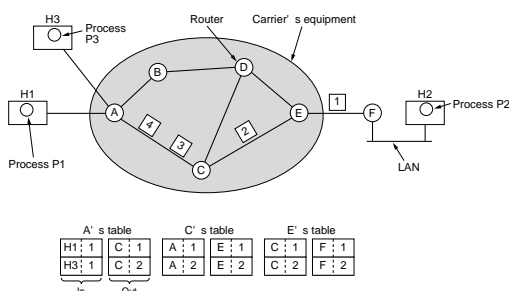
## Routing in datagram network



The routing tables list where to go for packets toward each destination.

Network(0234B)-6.9

## Routing in VC network



The circuit table lists what to do for packets of each incoming VC. (Accept it, or forward it to some VC of some other router.)

The routing table is still needed for the creation of circuit tables.

Network(0234B)-6.10

## Benifits and drawbacks

- **Datagram better:**
  - No need for circuit setup.
  - Simpler router software (no need to store circuits), require less RAM.
  - Tolerate crash of routers (no state is stored there anyway).
  - Adjust to congestions immediately.
- **VC better:**
  - Faster switching (routing done once for each VC, not packet).
  - Shorter frames (No long destination address).
  - Possibility for service guarantee (routers can reserve buffers).
  - Easier for charging (Carriers charge by VC connection time).

Network(0234B)-6.11

### Routing and optimality

- The core of this chapter is **how routing tables are created**.
- We want routing algorithms that are simple, robust, stable, fair and provide high throughput, but doing them all is not possible.
- Most routing algorithm tries to minimize **number of intermediate routers, or hops**, visited by a packet.
  - In case a link is slower, one can charge multiple hops for that connection. In Internet it is called "metric" of the connection.
- Routing is **optimal** if all packets visit the fewest hops to reach their destination.
- Question:** Routing table doesn't tell the whole route, it just tell the next hop. Can it get optimal routing?

Network(0234B)6.12

### Optimality Principle

Answer: **yes!**

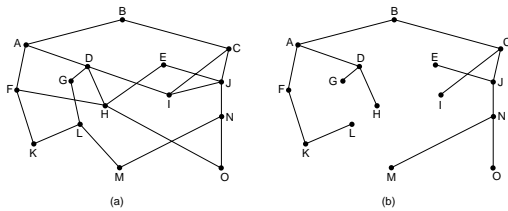
- Suppose router A have a packet towards E, and assume we also know the optimal path from A to E has the next hop C.
- Then A-C-???-E is optimal if C-???-E is optimal! Otherwise, if A-C-!!!-E is optimal, then C-???-E must not be optimal.
- This behaviour is called the "optimality principle".
  - If optimal path from A to E contains C, then it the concatenation of the optimal paths from A to C and from C to E is an optimal path from A to E.
- So if **every router uses the best next hop**, the routing is optimal.
  - In other words, it makes sense to rely on the next hop to find the remainder of the optimal path.
- What happens if some of the routers are mis-configured? A **routing loop** may occur, and the packet might never get to the destination.

Network(0234B)6.13

### Sink tree

If we **focus on paths towards the same destination**, and highlight the links used, we end up in a rooted tree. We call this the **sink tree**.

Due to the routing table: the routing table entry is used as the parent link. It is obvious that there can't be loops, so we end up into a tree.

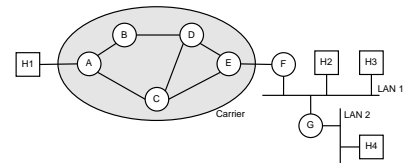


(a) the network, (b) a possible sink tree for B.

Network(0234B)6.14

### Hard-coding a routing table

One way to find the routing table is to **hard-code** it. This is usually done to more complicated local networks.



- F sends packets in LAN1 locally, sends packets towards LAN2 to G, and send everything else to E ("**default route**").
  - If G is instead a bridge rather than a router, LAN1 and LAN2 will look like a single LAN, so a frame can be sent directly to any host.
- This is optimal if F is the only way to go the the carrier's network. Because there is simply no other way.

Network(0234B)6.15

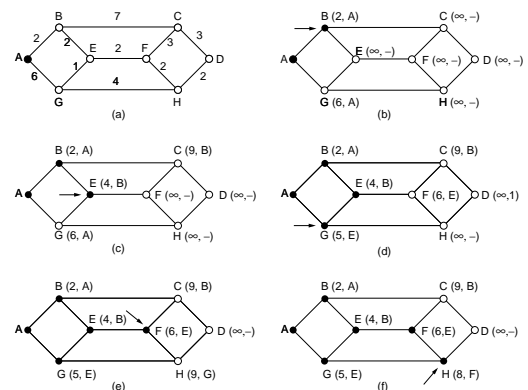
### Centralized algorithm

One way to derive the routing table is to **calculate the sink tree** for each destination. This can be done using the **Dijkstra** algorithm:

- Suppose we want to know paths **from all routers to a router A**.
- Keep a **known subtree S** of the sink tree. Initially, S contains only A.
- Also remember the **cost** for packets of each router in S to arrive A. A's cost is 0.
- In each step, find a neighbour N of S (and its link L to S) which, if added to S, results in the **least cost**.
  - How to find the cost from N to a router B in S? Just add up the remembered cost from B to A and the cost of L.
- N and L must be in the sink tree. So add them to S. Repeat.
  - N must somehow get to S, and if L is not the way to go, it must go through another way, which costs more.

Network(0234B)6.16

### Illustration



Network(0234B)6.17

### Need of a distributed algorithm

- There is a big obstacle to applying Dijkstra: **how a router know the costs of all links?**
- Another problem: **which router to compute it**, and what if the chosen router is down?
- Another problem is about distribution. One would require sending the whole routing table, of size  $N^2$ , to all the hosts.
- What we want: routing table computation to be **distributed**.
  - Each router should **periodically send information about the network** to its neighbour.
  - When such information is received, each host should **independently update its network information**, and recompute the routing table.
- Two methods: Distance Vector, and Link-State Routing.

Network(0234B)6.18

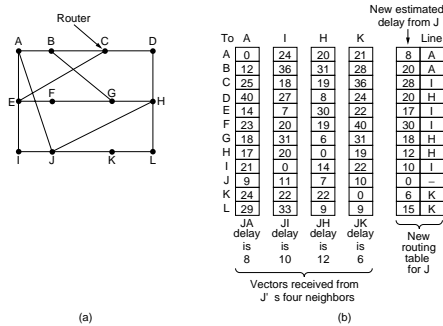
### Distance Vector

- Each router keeps, for each other destination D, the **cost** and **next hop** for packets towards D.
- When a router boots, it **initializes** the table to contain itself.
- Periodically (and when boot), the **list of costs are sent to neighbours**. The neighbours **records** them and **recompute** the routing table. The record is removed when the link is broken (e.g., after a few frames are lost).
- The routing table computation:
  - For a list from neighbour N, add the link cost to N to all its entries.
  - The cost for each destination is set to the **minimum cost among the resulting lists**. (Next hop is the neighbour containing the minimum.)
  - Cost towards oneself is set to 0. (No next hop.)

Network(0234B)6.19

### Example

This shows the computation done by J.



Network(0234B)6.20

### Behaviour

- The algorithm **always** converge to an optimal routing table.
- Some other good properties: Only a **small amount of information** are sent to each neighbour, and it is **quite easy to implement**.
- Used as the primary ARPANET routing protocol until 1979. ARPANET: the predecessor of the Internet, officially switch to TCP/IP in 1983.
- **Downside:** it **takes a long time** to learn **bad** news about a link is down, and it never know a host is down.
- **Reason:** Suppose a link A–B is down because B is down.
- A found the link is down, so remove its list. But its neighbour, say C, will tell A that it has a route, say with hop count=2 (C–A–B).
- Not knowing itself is involved in this path, A thinks that its hop count should be updated from 1 to 3, and later to 5, 7, and so on.
- We call this **"Count to infinity"** problem of Distance Vector.

Network(0234B)6.21

### Link state routing: the strategy

A completely different strategy: **don't propagate processed results**. Instead, just propagate information "as-is".

- Periodically, each router discovers its neighbours, and **measure the cost** (delay) to each of them.
- The resulting information is **broadcasted to every router in the network**, by "flooding".
- Upon receiving such information, each router **apply Dijkstra algorithm independently** to update the routing table.

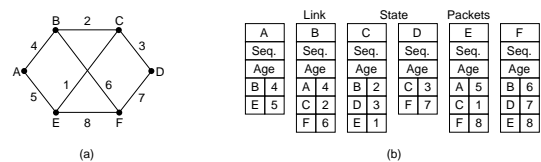
In other words, the algorithm is used only to **distribute** the network information. The **computation is left centralized**.

Network(0234B)6.22

### Link state packets

Every router periodically sends "HELLO" packets to discover neighbours, and "ECHO" packets to its neighbours to measure how long it takes.

This can also be done when some event occurs, e.g., when a new neighbour is up.



**Each router generates a packet**, to describe its neighbours and costs. They will be **flooded over the network**, so every router will have a copy.

Network(0234B)6.23

### How to flood?

Send it out to all network links. The recipient does the same.

**Problem:** This generate infinitely many packets!

1. Have a “hops-to-live”, e.g., 60. When a packet pass a hop, the count is decremented. When it reaches 0, the packet is discarded.  
If average fanout is 3, then we have at most  $3^{60}$  packets... what a bargain...
2. **Don't send back** to the originating link.  
Now it is  $2^{60}$ ... still huge.
3. Each packet has a source router identifier, and a sequence number. **Routers remember packets received**, and old packets (coming through different path) is dropped without forwarded.  
If router reboots and forgets its sequence number, its new packets seem old!
4. Routers **forget** packets after a while, say 1 minute. Each router broadcast more frequently than that (e.g., every 10 seconds).

Network(0234B)6.24

### Data structure held by routers

For each router (source) other than itself (B in the figure below):

- The **sequence number** of last packet received, and its **age**.
- The link state **packet data** itself.
- For each neighbour, whether we need to **forward** the packet to it, and whether it has **acknowledged** the packet.

Source	Seq.	Age	Send flags			ACK flags			Data
			A	C	F	A	C	F	
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	

Network(0234B)6.25

### Hierarchical Routing

- Suppose we have a router with  $b$  neighbours, in a network containing  $e$  links and  $n$  routers. How much data is **transferred**, and how much **time** and **memory** is needed, for finding routing tables?
- **Distance vector:**  $O(nb)$  bytes of memory and data transferred,  $O(nb)$  time is used in every round.
- **Link state:**  $O(eb)$  bytes of data transferred,  $O(nb + e)$  bytes of memory is used, and  $O((n + e)\log n)$  time is used in each round.  
CPU time due to Dijkstra once rather than  $n$  times.
- Both can support reasonably large network topology. When the **network size grows too large**, we cannot rely on them completely.
- Instead, a **two tier** routing scheme can be used. E.g., separate all routers to regions, the root network route packets to the right region, while a regional network route them to the right router.  
Perhaps not optimal, but loss in efficiency is hopefully small.

Network(0234B)6.26

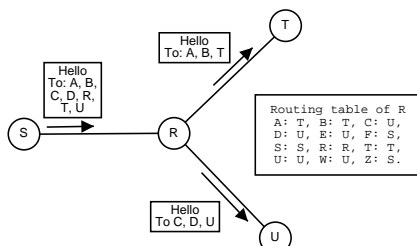
### Broadcasting

- Some applications requires the ability for a host to **send the same packet to all destinations:** broadcasting.
- On **broadcast data links**, there are usually **hardware addresses** reserved for this purpose (hardware broadcast). **What for networks containing point-to-point links?**
- Method 1: send normal (“unicast”) packets to **each host** individually: **wasteful** (the same packet appear in the same link many times).
- Method 2: **flooding** like link state routing: **use a lot of memory**.  
It is okay for each router to store a sequence number for each other router, but it is probably too much to ask each host to store one for each other host.
- Method 3: **multidestination routing:** packets include a list of destinations, so that routers forward the message with its own routing table.  
The list is **modified** before forwarding.

Network(0234B)6.27

### Example: multidestination routing

Now **each router only receives one copy** of each broadcast packet. Packets won't arrive via two different route, unless the routing table is bad.



This requires the **fewest packets**. Problem is that **routing is slow** (to generate the new destinations for each message), and **packets are huge** (for holding all destinations).

Network(0234B)6.28

### Emulate flooding

- Method 4: **flood using the sink tree**. When a router R receives a packet from A, forward it to routers and hosts whose next hop for A is R. Sends the fewest packets, but **difficult to implement**.  
How R knows whether its neighbour, say N, has R as the next hop for A? Possible if we use link state routing, impossible if we use distance vector.
- Method 5 (Reverse Path Forwarding): Forward it to everyone except the incoming link, **conditionally**: Suppose R receives a broadcast packet from A through a link from N. R will forward it if next hop for A is N.  
Resolves the problem in method 4 by letting the receiver to do the selection.

Can be implemented easily, and number of messages should be the same as a normal flooding (i.e.,  $O(e)$ , the number of links).

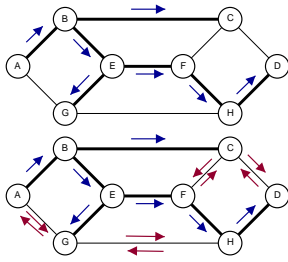
Note that all these **assumes a correct routing table**, so link state routing cannot use them.

Packets can be lost or duplicated if routing table is wrong, hopefully the table is fix ed in a timely manner.

Network(0234B)6.29

**Example: flooding emulations**

**Sink tree** for A (thick lines), and the **packets sent** (arrows) during a broadcast by A using (a) sink tree directly and (b) reverse path routing. Red arrows are rejected packets.

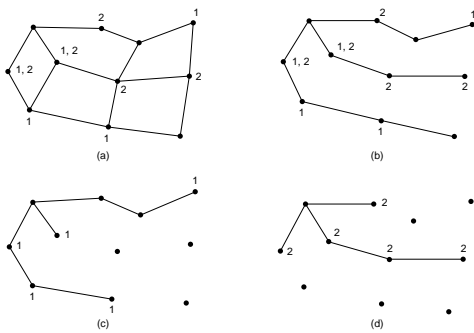


**Multicasting**

- Some applications require hosts **sending the same packet to a group of hosts** rather than a single host (or all hosts): **multicast**.  
E.g., video on-demand.
- This requires **group management**: it must be possible to **create and destroy groups**, and also for hosts to **join and leave** groups.  
Routers remembers whether it has a host belonging to each group.
- This also requires **multicast routing**. It can be done using unicast or broadcast, but in general they are not very efficient.
- **Strategy 1**: each sender uses its **sink tree** for sending packets.
- For each group G and each source router (i.e., tree), **each router records which down-links in the tree has routers interested in G**.
- When all hosts in a router and its down-links left a group, a **prune** message is sent to the parent.

**Pruning**

Packets of a group is sent only to down-links with interested members.



(a) Network, (b) Sink tree for top-left router, (c) pruned for group 1, and (d) pruned for group 2.

**Inefficiency and Core based routing**

- **Not scalable**: Each router needs to keep a **lot of trees!**  
If there are  $n$  groups, each with  $m$  members, then we need to keep information of  $mn$  trees.
- Strategy 2: **“Core-based routing”**. Use a **different spanning tree** for different group.
- The **root, or core**, of each tree is selected to be **close to the “center” of the members**. **All messages are sent to the root** for routing.
- Method to find pruned tree can remain unchanged.
- In this way, there is only 1 (pruned) tree **for each group**.  
An improvement of  $m$  folds.
- The routing **won't be optimal**. This is the cost to pay for the savings in memory space.