Network Layer

Main service: Provide facilities for getting data from a source to a destination ⇒ **routing**.

Again, distinguish between connectionless and connection-oriented services. There are two *implementation* techniques:

- **Virtual circuits** are complete routes that are set up in advance.
- **Datagrams** comprise individual packets of which the route is determined on the fly: they hop from router to router.

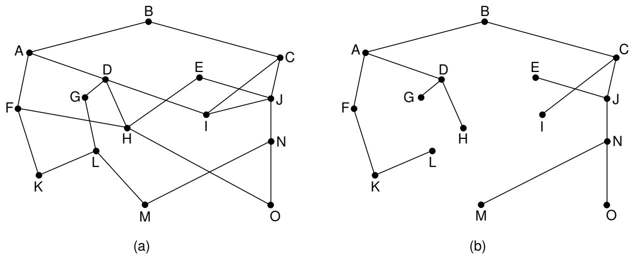
Note: The services are independent of their implementation:

	Datagram	**Virtual circuit**
CL	UDP over IP	UDP over IP over ATM
CO	TCP over IP	ATM AAL1 over ATM

Routing

Main issue: Routers that constitute the network layer of a network, should cooperate to find the **best routes** between all pairs of stations.

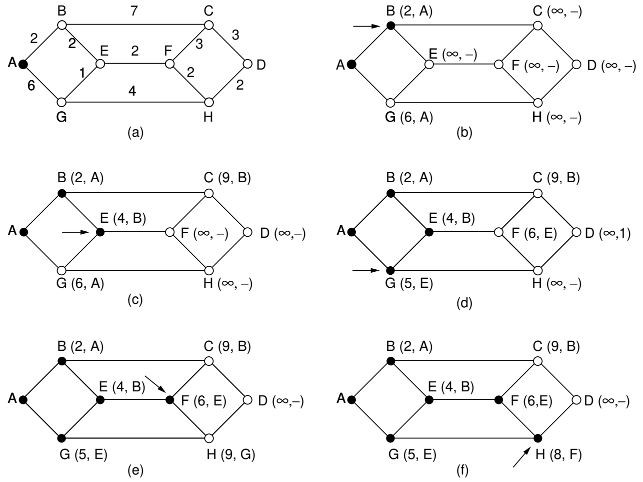
Observation: All optimal routes from station A to other stations in the network, jointly constitute a **sink tree**:



This means: Routers have to collaborate to build the sink tree (or something that comes near to that) for each source station.

Shortest Path Routing

Basic idea: During each step, select a newly reachable node at the lowest cost, and add the edge to that node, to the tree built so far.



Flooding

Basic idea: Forward an incoming packet across every outgoing line, except the one it came through.

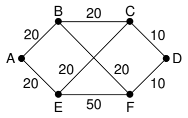
Basic problem: how to avoid "drowning by packets"?

- Use a hop counter: after a packet has been forwarded across *N* routers, it is discarded. Gotta find the right hop count, though.
- Be sure to forward a packet only once (i.e. avoid directed cycles). Requires sequence numbers per source router. Each router keeps track of the last sequence number per source router.
- Flood selectively: only in the direction that makes sense.

In general: flooding makes only sense when robustness is needed.

Flow-Based Routing

Basic idea: Rather than just looking at the network topology, consider the traffic load as well. Evaluate several routing schemes.



(a)

Destination

Source	A	B	C	D	E	F
A		9 AB	4 ABC	1 ABFD	7 AE	4 AEF
B	9 BA		8 BC	3 BFD	2 BFE	4 BF
C	4 CBA	8 CB		3 CD	3 CE	2 CEF
D	1 DFBA	3 DFB	3 DC		3 DCE	4 DF
E	7 EA	2 EFB	3 EC	3 ECD		5 EF
F	4 FEA	4 FB	2 FEC	4 FD	5 FE	

(b)

1. Calculate the total line traffic. $\lambda_{AB} = 9+4+1$, $\lambda_{BF} = 1+3+2+4$, etc.
2. Calculate the queuing and service time per line: $T_i = (\mu C_i - \lambda_i)^{-1}$, where μC_i denotes the capacity.
3. Calculate delay time for the network: $\sum \alpha_i \cdot T_i$, with $\alpha_i = \lambda_i / \sum \lambda_i$.

Distance Vector Routing

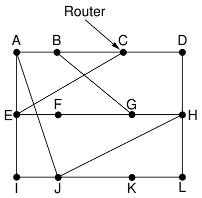
Basic idea: Take a look at the costs that your direct neighbors are advertising to get a packet to the destination. Select the neighbor whose advertised cost, added with the cost to get to that neighbor, is the lowest. Advertise that new cost to the other neighbors.

Example:

Neighbor:	R_1	R_2	R_3
Link cost:	12	8	5
Advertised:	28	25	39
Total:	40	33	44

Question: What will be the effect of this routing data?

DVR – Example



(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	New routing table for J	

Vectors received from J's four neighbors

(b)

Note: DVR is pretty expensive: it turns out to be $O(n^3)$ which is too high for a fast convergence. There are also a number of peculiar problems...

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)

Link State Routing

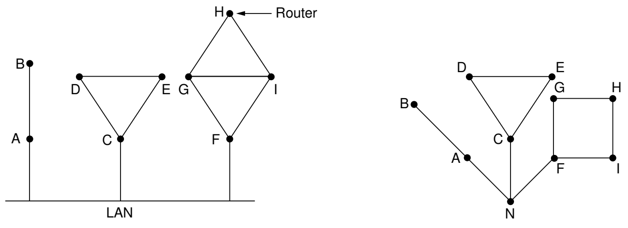
Better idea: One way or the other, broadcast info on the entire network topology to all routers, and let each of them calculate a sink tree to the other routers.

What a router needs to do (next to just routing):

- Find out who its neighbors are and get their network addresses.
- Calculate the cost for getting a packet to a neighbor.
- Construct a **link state packet** telling all it has just learned.
- Send that packet to all other routers.
- Do a Dijkstra.

LSP: Who are my Neighbors?

Simple: you know your network interface(s), so just send an HELLO packet through each of them. May be just a bit confusing when you're on a broadcast LAN:



(a) (b)

LSP: What's the Link Cost?

Simple: Just send an ECHO packet through each interface, and measure the round-trip delay. That'll give you a reasonable estimate of the actual delay.

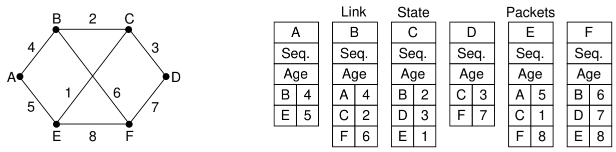
Problem: do we take the local load into account, or not?

YES: You can choose for momentarily better routes. Example: take a train to Amsterdam at 8 'o clock, rather than going by car using the A4.

NO: You may redirect traffic in such a way that the alternative route is overloaded. Example: when people are advised to take the train when the Schiphol tunnel has only one lane. Trains will be delayed (more than usual).

LSP: Building Packets

Again simple: just put in a sequence number and aging information. The hard part is when to build them. Practice shows that once an hour is often enough.

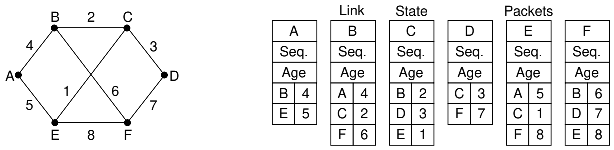


(a) (b)

LSP: Distribution

Basic idea: use a flooding algorithm, and dam the flood through sequence numbers: all routers maintain a list of *(source,seq.number)*-pairs.

To safeguard against old data, down links, etc. an age is added to an LSP. The age is decremented once a second, and every time it is forwarded by a router. When the age hits zero, the LSP is discarded.



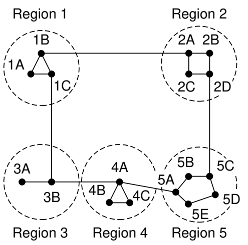
(a) (b)

			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	

Hierarchical Routing

Problem: No routing algorithm discussed so far can scale: all of them require each router to know about all others ⇒ too demanding with respect to memory capacity and processing power.

Solution: Go for *suboptimal* routes by introducing **regions**, and separate algorithms for **intra-region** and **inter-region** routing. Two or three levels will generally do.



(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)

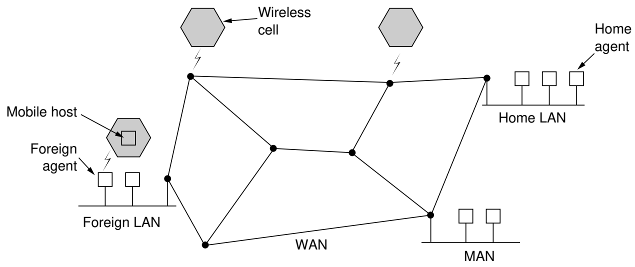
Hierarchical table for 1A

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

(c)

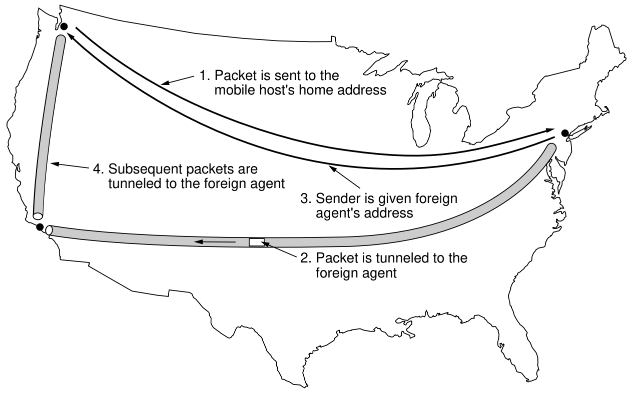
Routing for Mobile Hosts

This is fun: How can we forward messages to things that are constantly on the move, preferably in a wide-area context?



Note: The problem is not moving hosts from LAN to LAN, but finding the mobile computers. It becomes a bit easier when we assume that there is always a **home location**.

Routing for Mobile Hosts II



- Tunneling: sending an IP packet in an IP packet. That's the way to keep the routers ignorant of the fact that they're routing something else.
- Adapt routers: when you send the new address back to the source, intermediate routers can adapt their tables.

Broadcast Routing

Problem: We want to send a message to (almost) every host on the network. In practice, this means we're talking about (interconnected) LANs, or (relatively small) WANs.

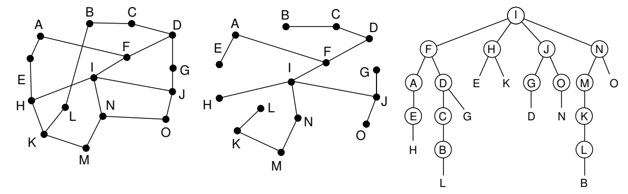
- Just send the message to each host individually. Not really good.
- Use flooding. Acceptable provided that we can dam the flood.
- Use multidestination routing, by means of a bitmap or list that is sent along with the packet. A router checks the destinations, and splits the list when forwarding it across different output lines.
- Build a sink tree at the source and use that as your multicast route. The sink tree has to be a spanning tree (as in Dijkstra). The routers need to know the trees.

Broadcasting: Reverse Path Forwarding

Problem: Suppose we don't know every spanning tree. How can we construct one at low cost?

Solution: Make the assumption that when a router *R* forwards a packet for node *N*, it chooses an outgoing link that lies on a minimal spanning tree rooted at *N*.

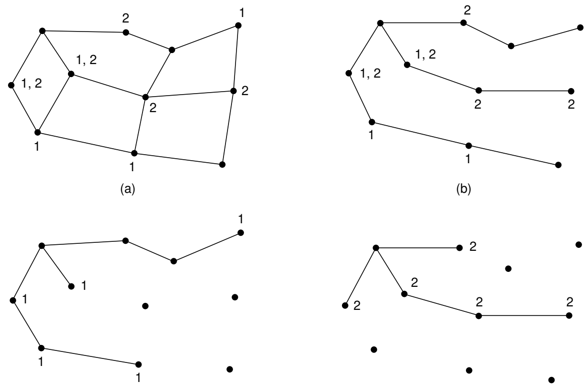
Question: When is this a realistic assumption?


(a) (b) (c)

Multicast Routing

Problem: But suppose that we want to send a message to only a subset of all the nodes in a network. In that case, we need to know when a host enters or leaves the **multicast group**.

Solution: Construct a spanning tree (at each router) for the entire network. Use the group-id to prune paths to nodes that do not contain members for that group.


(a) (b) (c) (d)