**Routing Principles , Routing and Overview, DVR and LSR**

**The network layer**

The transport layer enables the applications to efficiently and reliably exchange data. Transport layer entities expect to be able to send segment to any destination without having to understand anything about the underlying subnetwork technologies. Many subnetwork technologies exist. Most of them differ in subtle details (frame size, addressing, ...). The network layer is the glue between these subnetworks and the transport layer. It hides to the transport layer all the complexity of the underlying subnetworks and ensures that information can be exchanged between hosts connected to different types of subnetworks.

In this chapter, we first explain the principles of the network layer. These principles include the datagram and virtual circuit modes, the separation between the data plane and the control plane and the algorithms used by routing protocols. Then, we explain, in more detail, the network layer in the Internet, starting with IPv4 and IPv6 and then moving to the routing protocols (RIP, OSPF and BGP).

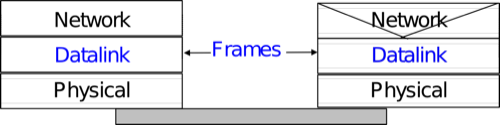
Principles

The main objective of the network layer is to allow endsystems, connected to different networks, to exchange information through intermediate systems called[***router***](http://cnp3book.info.ucl.ac.be/1st/html/glossary.html#term-router). The unit of information in the network layer is called a [***packet***](http://cnp3book.info.ucl.ac.be/1st/html/glossary.html#term-packet).

**[](http://cnp3book.info.ucl.ac.be/1st/html/_images/osi-network.png)**

The network layer in the reference model

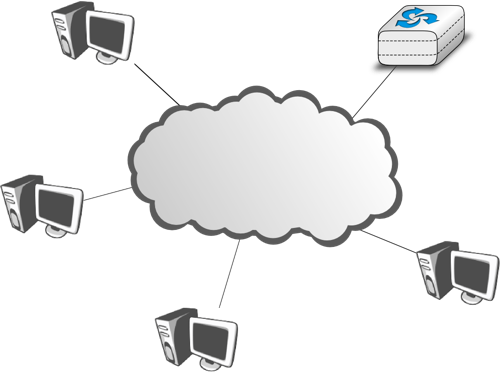
Before explaining the network layer in detail, it is useful to begin by analysing the service provided by the *datalink* layer. There are many variants of the datalink layer. Some provide a connection-oriented service while others provide a connectionless service. In this section, we focus on connectionless datalink layer services as they are the most widely used. Using a connection-oriented datalink layer causes some problems that are beyond the scope of this chapter. See [**RFC 3819**](http://tools.ietf.org/html/rfc3819.html) for a discussion on this topic.

**[](http://cnp3book.info.ucl.ac.be/1st/html/_images/osi-datalink.png)**

The point-to-point datalink layer

There are three main types of datalink layers. The simplest datalink layer is when there are only two communicating systems that are directly connected through the physical layer. Such a datalink layer is used when there is a point-to-point link between the two communicating systems. The two systems can be endsystems or routers. PPP, defined in [**RFC 1661**](http://tools.ietf.org/html/rfc1661.html), is an example of such a point-to-point datalink layer. Datalink layers exchange *frames* and a datalink [***frame***](http://cnp3book.info.ucl.ac.be/1st/html/glossary.html#term-frame) sent by a datalink layer entity on the left is transmitted through the physical layer, so that it can reach the datalink layer entity on the right. Point-to-point datalink layers can either provide an unreliable service (frames can be corrupted or lost) or a reliable service (in this case, the datalink layer includes retransmission mechanisms similar to the ones used in the transport layer). The unreliable service is frequently used above physical layers (e.g. optical fiber, twisted pairs) having a low bit error ratio while reliability mechanisms are often used in wireless networks to recover locally from transmission errors.

The second type of datalink layer is the one used in Local Area Networks (LAN). Conceptually, a LAN is a set of communicating devices such that any two devices can directly exchange frames through the datalink layer. Both endsystems and routers can be connected to a LAN. Some LANs only connect a few devices, but there are LANs that can connect hundreds or even thousands of devices.

**[](http://cnp3book.info.ucl.ac.be/1st/html/_images/simple-lan.png)**

A local area network

In the next chapter, we describe the organisation and the operation of Local Area Networks. An important difference between the point-to-point datalink layers and the datalink layers used in LANs is that in a LAN, each communicating device is identified by a unique *datalink layer address*. This address is usually embedded in the hardware of the device and different types of LANs use different types of datalink layer addresses. A communicating device attached to a LAN can send a datalink frame to any other communicating device that is attached to the same LAN. Most LANs also support special broadcast and multicast datalink layer addresses. A frame sent to the broadcast address of the LAN is delivered to all communicating devices that are attached to the LAN. The multicast addresses are used to identify groups of communicating devices. When a frame is sent towards a multicast datalink layer address, it is delivered by the LAN to all communicating devices that belong to the corresponding group.

The third type of datalink layers are used in Non-Broadcast Multi-Access (NBMA) networks. These networks are used to interconnect devices like a LAN. All devices attached to an NBMA network are identified by a unique datalink layer address. However, and this is the main difference between an NBMA network and a traditional LAN, the NBMA service only supports unicast. The datalink layer service provided by an NBMA network supports neither broadcast nor multicast.

Unfortunately no datalink layer is able to send frames of unlimited side. Each datalink layer is characterised by a maximum frame size. There are more than a dozen different datalink layers and unfortunately most of them use a different maximum frame size. The network layer must cope with the heterogeneity of the datalink layer.

The network layer itself relies on the following principles :

1. Each network layer entity is identified by a *network layer address*. This address is independent of the datalink layer addresses that it may use.
2. The service provided by the network layer does not depend on the service or the internal organisation of the underlying datalink layers.
3. The network layer is conceptually divided into two planes : the *data plane* and the *control plane*. The *data plane* contains the protocols and mechanisms that allow hosts and routers to exchange packets carrying user data. The *control plane* contains the protocols and mechanisms that enable routers to efficiently learn how to forward packets towards their final destination.

The independence of the network layer from the underlying datalink layer is a key principle of the network layer. It ensures that the network layer can be used to allow hosts attached to different types of datalink layers to exchange packets through intermediate routers. Furthermore, this allows the datalink layers and the network layer to evolve independently from each other. This enables the network layer to be easily adapted to a new datalink layer every time a new datalink layer is invented.

There are two types of service that can be provided by the network layer :

* an *unreliable connectionless* service
* a *connection-oriented*, reliable or unreliable, service

Connection-oriented services have been popular with technologies such as [***X.25***](http://cnp3book.info.ucl.ac.be/1st/html/glossary.html#term-x-25) and [***ATM***](http://cnp3book.info.ucl.ac.be/1st/html/glossary.html#term-atm) or [***frame-relay***](http://cnp3book.info.ucl.ac.be/1st/html/glossary.html#term-frame-relay), but nowadays most networks use an *unreliable connectionless* service.

### Organisation of the network layer

There are two possible internal organisations of the network layer :

* datagram
* virtual circuits

The internal organisation of the network is orthogonal to the service that it provides, but most of the time a datagram organisation is used to provide a connectionless service while a virtual circuit organisation is used in networks that provide a connection-oriented service.

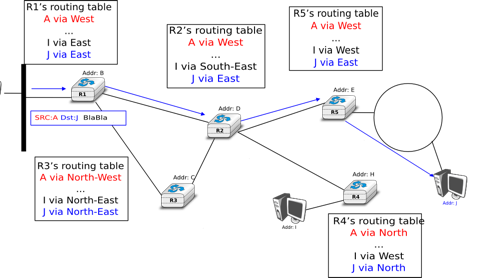
#### Datagram organisation

The first and most popular organisation of the network layer is the datagram organisation. This organisation is inspired by the organisation of the postal service. Each host is identified by a network layer address. To send information to a remote host, a host creates a packet that contains :

* the network layer address of the destination host
* its own network layer address
* the information to be sent

The network layer limits the maximum packet size. Thus, the information must have been divided in packets by the transport layer before being passed to the network layer.

To understand the datagram organisation, let us consider the figure below. A network layer address, represented by a letter, has been assigned to each host and router. To send some information to host J, host A creates a packet containing its own address, the destination address and the information to be exchanged.

**[](http://cnp3book.info.ucl.ac.be/1st/html/_images/simple-internetwork.png)**

A simple internetwork

With the datagram organisation, routers use hop-by-hop forwarding. This means that when a router receives a packet that is not destined to itself, it looks up the destination address of the packet in its routing table. A routing table is a data structure that maps each destination address (or set of destination addresses) to the outgoing interface over which a packet destined to this address must be forwarded to reach its final destination.

The main constraint imposed on the routing tables is that they must allow any host in the network to reach any other host. This implies that each router must know a route towards each destination, but also that the paths composed from the information stored in the routing tables must not contain loops. Otherwise, some destinations would be unreachable.

In the example above, host A sends its packet to router R1. R1 consults its routing table and forwards the packet towards R2. Based on its own routing table, R2decides to forward the packet to R5 that can deliver it to its destination.

To allow hosts to exchange packets, a network relies on two different types of protocols and mechanisms. First, there must be a precise definition of the format of the packets that are sent by hosts and processed by routers. Second, the algorithm used by the routers to forward these packets must be defined. This protocol and this algorithm are part of the data plane of the network layer. The data plane contains all the protocols and algorithms that are used by hosts and routers to create and process the packets that contain user data.

The data plane, and in particular the forwarding algorithm used by the routers, depends on the routing tables that are maintained on reach router. These routing tables can be maintained by using various techniques (manual configuration, distributed protocols, centralised computation, etc). These techniques are part of thecontrol plane of the network layer. The control plane contains all the protocols and mechanisms that are used to compute and install routing tables on the routers.

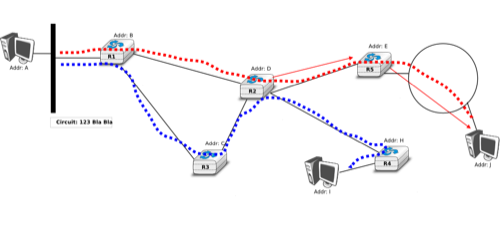
The datagram organisation has been very popular in computer networks. Datagram based network layers include IPv4 and IPv6 in the global Internet, CLNP defined by the ISO, IPX defined by Novell or XNS defined by Xerox [**[Perlman2000]**](http://cnp3book.info.ucl.ac.be/1st/html/bibliography.html#perlman2000).

#### Virtual circuit organisation

The main advantage of the datagram organisation is its simplicity. The principles of this organisation can easily be understood. Furthermore, it allows a host to easily send a packet towards any destination at any time. However, as each packet is forwarded independently by intermediate routers, packets sent by a host may not follow the same path to reach a given destination. This may cause packet reordering, which may be annoying for transport protocols. Furthermore, as a router usinghop-by-hop forwarding always forwards packets sent towards the same destination over the same outgoing interface, this may cause congestion over some links.

The second organisation of the network layer, called virtual circuits, has been inspired by the organisation of telephone networks. Telephone networks have been designed to carry phone calls that usually last a few minutes. Each phone is identified by a telephone number and is attached to a telephone switch. To initiate a phone call, a telephone first needs to send the destination’s phone number to its local switch. The switch cooperates with the other switches in the network to create a bi-directional channel between the two telephones through the network. This channel will be used by the two telephones during the lifetime of the call and will be released at the end of the call. Until the 1960s, most of these channels were created manually, by telephone operators, upon request of the caller. Today’s telephone networks use automated switches and allow several channels to be carried over the same physical link, but the principles remain roughly the same.

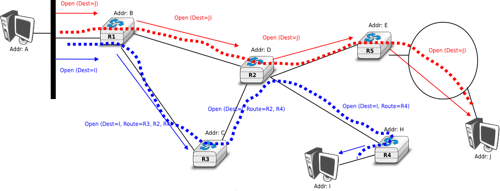
In a network using virtual circuits, all hosts are identified with a network layer address. However, a host must explicitly request the establishment of a virtual circuitbefore being able to send packets to a destination host. The request to establish a virtual circuit is processed by the control plane, which installs state to create the virtual circuit between the source and the destination through intermediate routers. All the packets that are sent on the virtual circuit contain a virtual circuit identifier that allows the routers to determine to which virtual circuit each packet belongs. This is illustrated in the figure below with one virtual circuit between host A and hostI and another one between host A and host J.

**[](http://cnp3book.info.ucl.ac.be/1st/html/_images/simple-internetwork-vc.png)**

A simple internetwork using virtual-circuits

The establishment of a virtual circuit is performed using a signalling protocol in the control plane. Usually, the source host sends a signalling message to indicate to its router the address of the destination and possibly some performance characteristics of the virtual circuit to be established. The first router can process the signalling message in two different ways.

A first solution is for the router to consult its routing table, remember the characteristics of the requested virtual circuit and forward it over its outgoing interface towards the destination. The signalling message is thus forwarded hop-by-hop until it reaches the destination and the virtual circuit is opened along the path followed by the signalling message. This is illustrated with the red virtual circuit in the figure below.

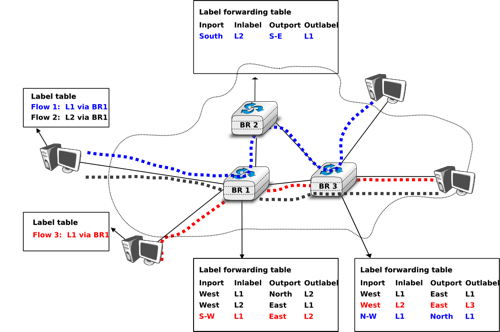
**[](http://cnp3book.info.ucl.ac.be/1st/html/_images/simple-internetwork-vc-estab.png)**

Virtual circuit establishment

A second solution can be used if the routers know the entire topology of the network. In this case, the first router can use a technique called source routing. Upon reception of the signalling message, the first router chooses the path of the virtual circuit in the network. This path is encoded as the list of the addresses of all intermediate routers to reach the destination. It is included in the signalling message and intermediate routers can remove their address from the signalling message before forwarding it. This technique enables routers to spread the virtual circuits throughout the network better. If the routers know the load of remote links, they can also select the least loaded path when establishing a virtual circuit. This solution is illustrated with the blue circuit in the figure above.

The last point to be discussed about the virtual circuit organisation is its data plane. The data plane mainly defines the format of the data packets and the algorithm used by routers to forward packets. The data packets contain a virtual circuit identifier, encoded as a fixed number of bits. These virtual circuit identifiers are usually called labels.

Each host maintains a flow table that associates a label with each virtual circuit that is has established. When a router receives a packet containing a label, it extracts the label and consults its label forwarding table. This table is a data structure that maps each couple (incoming interface, label) to the outgoing interface to be used to forward the packet as well as the label that must be placed in the outgoing packets. In practice, the label forwarding table can be implemented as a vector and the couple (incoming interface, label) is the index of the entry in the vector that contains the outgoing interface and the outgoing label. Thus a single memory access is sufficient to consult the label forwarding table. The utilisation of the label forwarding table is illustrated in the figure below.

**[](http://cnp3book.info.ucl.ac.be/1st/html/_images/label-forwarding.png)**

Label forwarding tables in a network using virtual circuits

The virtual circuit organisation has been mainly used in public networks, starting from X.25 and then Frame Relay and Asynchronous Transfer Mode (ATM) network.

Both the datagram and virtual circuit organisations have advantages and drawbacks. The main advantage of the datagram organisation is that hosts can easily send packets to any number of destinations while the virtual circuit organisation requires the establishment of a virtual circuit before the transmission of a data packet. This solution can be costly for hosts that exchange small amounts of data. On the other hand, the main advantage of the virtual circuit organisation is that the forwarding algorithm used by routers is simpler than when using the datagram organisation. Furthermore, the utilisation of virtual circuits may allow the load to be better spread through the network thanks to the utilisation of multiple virtual circuits. The MultiProtocol Label Switching (MPLS) technique that we will discuss in another revision of this book can be considered as a good compromise between datagram and virtual circuits. MPLS uses virtual circuits between routers, but does not extend them to the endhosts. Additional information about MPLS may be found in [**[ML2011]**](http://cnp3book.info.ucl.ac.be/1st/html/bibliography.html#ml2011).

### The control plane

One of the objectives of the control plane in the network layer is to maintain the routing tables that are used on all routers. As indicated earlier, a routing table is a data structure that contains, for each destination address (or block of addresses) known by the router, the outgoing interface over which the router must forward a packet destined to this address. The routing table may also contain additional information such as the address of the next router on the path towards the destination or an estimation of the cost of this path.

In this section, we discuss the three main techniques that can be used to maintain the routing tables in a network.

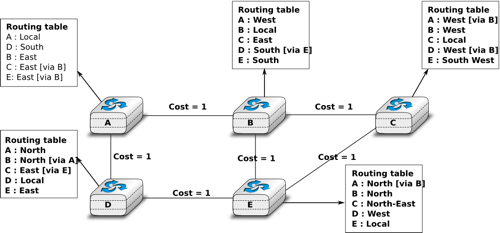
#### Static routing

The simplest solution is to pre-compute all the routing tables of all routers and to install them on each router. Several algorithms can be used to compute these tables.

A simple solution is to use shortest path routing and to minimise the number of intermediate routers to reach each destination. More complex algorithms can take into account the expected load on the links to ensure that congestion does not occur for a given traffic demand. These algorithms must all ensure that :

* all routers are configured with a route to reach each destination
* none of the paths composed with the entries found in the routing tables contain a cycle. Such a cycle would lead to a forwarding loop.

The figure below shows sample routing tables in a five routers network.

**[](http://cnp3book.info.ucl.ac.be/1st/html/_images/routing-tables.png)**

Routing tables in a simple network

The main drawback of static routing is that it does not adapt to the evolution of the network. When a new router or link is added, all routing tables must be recomputed. Furthermore, when a link or router fails, the routing tables must be updated as well.

#### Distance vector routing

Distance vector routing is a simple distributed routing protocol. Distance vector routing allows routers to automatically discover the destinations reachable inside the network as well as the shortest path to reach each of these destinations. The shortest path is computed based on metrics or costs that are associated to each link. We use l.cost to represent the metric that has been configured for link l on a router.

Each router maintains a routing table. The routing table R can be modelled as a data structure that stores, for each known destination address d, the following attributes :

* R[d].link is the outgoing link that the router uses to forward packets towards destination d
* R[d].cost is the sum of the metrics of the links that compose the shortest path to reach destination d
* R[d].time is the timestamp of the last distance vector containing destination d

A router that uses distance vector routing regularly sends its distance vector over all its interfaces. The distance vector is a summary of the router’s routing table that indicates the distance towards each known destination. This distance vector can be computed from the routing table by using the pseudo-code below.

Every N seconds:

v=Vector()

**for** d **in** R[]:

*# add destination d to vector*

v.add(Pair(d,R[d].cost))

**for** i **in** interfaces

*# send vector v on this interface*

send(v,interface)

When a router boots, it does not know any destination in the network and its routing table only contains itself. It thus sends to all its neighbours a distance vector that contains only its address at a distance of 0. When a router receives a distance vector on link l, it processes it as follows.

*# V : received Vector*

*# l : link over which vector is received*

**def** received(V,l):

*# received vector from link l*

**for** d **in** V[]

**if** **not** (d **in** R[]) :

*# new route*

R[d].cost=V[d].cost+l.cost

R[d].link=l

R[d].time=now

**else** :

*# existing route, is the new better ?*

**if** ( ((V[d].cost+l.cost) < R[d].cost) **or** ( R[d].link == l) ) :

*# Better route or change to current route*

R[d].cost=V[d].cost+l.cost

R[d].link=l

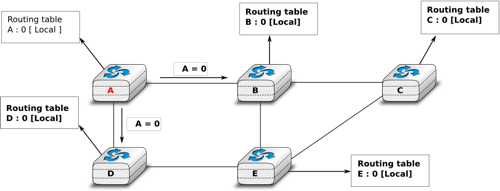
R[d].time=now

The router iterates over all addresses included in the distance vector. If the distance vector contains an address that the router does not know, it inserts the destination inside its routing table via link l and at a distance which is the sum between the distance indicated in the distance vector and the cost associated to link l. If the destination was already known by the router, it only updates the corresponding entry in its routing table if either :

* the cost of the new route is smaller than the cost of the already known route ( (V[d].cost+l.cost) < R[d].cost)
* the new route was learned over the same link as the current best route towards this destination ( R[d].link == l)

The first condition ensures that the router discovers the shortest path towards each destination. The second condition is used to take into account the changes of routes that may occur after a link failure or a change of the metric associated to a link.

To understand the operation of a distance vector protocol, let us consider the network of five routers shown below.

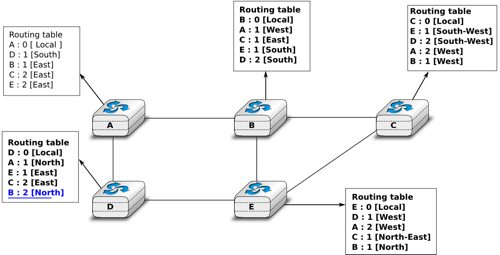
**[](http://cnp3book.info.ucl.ac.be/1st/html/_images/dv-1.png)**

Operation of distance vector routing in a simple network

Assume that A is the first to send its distance vector [A=0].

* B and D process the received distance vector and update their routing table with a route towards A.
* D sends its distance vector [D=0,A=1] to A and E. E can now reach A and D.
* C sends its distance vector [C=0] to B and E
* E sends its distance vector [E=0,D=1,A=2,C=2] to D, B and C. B can now reach A, C, D and E
* B sends its distance vector [B=0,A=1,C=1,D=2,E=1] to A, C and E. A, B, C and E can now reach all destinations.
* A sends its distance vector [A=0,B=1,C=2,D=1,E=2] to B and D.

At this point, all routers can reach all other routers in the network thanks to the routing tables shown in the figure below.

**[](http://cnp3book.info.ucl.ac.be/1st/html/_images/dv-full.png)**

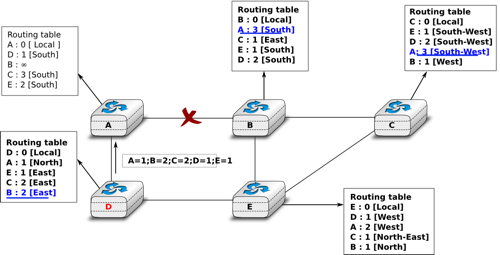
Routing tables computed by distance vector in a simple network

To deal with link and router failures, routers use the timestamp stored in their routing table. As all routers send their distance vector every N seconds, the timestamp of each route should be regularly refreshed. Thus no route should have a timestamp older than N seconds, unless the route is not reachable anymore. In practice, to cope with the possible loss of a distance vector due to transmission errors, routers check the timestamp of the routes stored in their routing table every N seconds and remove the routes that are older than 3 \times N seconds. When a router notices that a route towards a destination has expired, it must first associate an \infty cost to this route and send its distance vector to its neighbours to inform them. The route can then be removed from the routing table after some time (e.g. 3 \times Nseconds), to ensure that the neighbouring routers have received the bad news, even if some distance vectors do not reach them due to transmission errors.

Consider the example above and assume that the link between routers A and B fails. Before the failure, A used B to reach destinations B, C and E while B only used the A-B link to reach A. The affected entries timeout on routers A and B and they both send their distance vector.

* A sends its distance vector [A=0,D=\infty,C=\infty,D=1,E=\infty]. D knows that it cannot reach B anymore via A
* D sends its distance vector [D=0,B=\infty,A=1,C=2,E=1] to A and E. A recovers routes towards C and E via D.
* B sends its distance vector [B=0,A=\infty,C=1,D=2,E=1] to E and C. D learns that there is no route anymore to reach A via B.
* E sends its distance vector [E=0,A=2,C=1,D=1,B=1] to D, B and C. D learns a route towards B. C and B learn a route towardsA.

At this point, all routers have a routing table allowing them to reach all another routers, except router A, which cannot yet reach router B. A recovers the route towards B once router D sends its updated distance vector [A=1, B=2,C=2,D=1,E=1]. This last step is illustrated in figure [**Routing tables computed by distance vector after a failure**](http://cnp3book.info.ucl.ac.be/1st/html/network/network.html#fig-afterfailure), which shows the routing tables on all routers.

**[](http://cnp3book.info.ucl.ac.be/1st/html/_images/dv-failure-2.png)**

Routing tables computed by distance vector after a failure

Consider now that the link between D and E fails. The network is now partitioned into two disjoint parts : (A , D) and (B, E, C). The routes towards B, C and E expire first on router D. At this time, router D updates its routing table.

If D sends [D=0, A=1, B=\infty, C=\infty, E=\infty], A learns that B, C and E are unreachable and updates its routing table.

Unfortunately, if the distance vector sent to A is lost or if A sends its own distance vector ( [A=0,D=1,B=3,C=3,E=2] ) at the same time as D sends its distance vector, D updates its routing table to use the shorter routes advertised by A towards B, C and E. After some time D sends a new distance vector : [D=0,A=1,E=3,C=4,B=4]. A updates its routing table and after some time sends its own distance vector [A=0,D=1,B=5,C=5,E=4], etc. This problem is known as the count to infinity problem in networking literature. Routers A and D exchange distance vectors with increasing costs until these costs reach \infty. This problem may occur in other scenarios than the one depicted in the above figure. In fact, distance vector routing may suffer from count to infinity problems as soon as there is a cycle in the network. Cycles are necessary to have enough redundancy to deal with link and router failures. To mitigate the impact of counting to infinity, some distance vector protocols consider that 16=\infty. Unfortunately, this limits the metrics that network operators can use and the diameter of the networks using distance vectors.

This count to infinity problem occurs because router A advertises to router D a route that it has learned via router D. A possible solution to avoid this problem could be to change how a router creates its distance vector. Instead of computing one distance vector and sending it to all its neighbors, a router could create a distance vector that is specific to each neighbour and only contains the routes that have not been learned via this neighbour. This could be implemented by the following pseudocode.

Every N seconds:

*# one vector for each interface*

**for** l **in** interfaces:

v=Vector()

**for** d **in** R[]:

**if** (R[d].link != i) :

v=v+Pair(d,R[d.cost])

send(v)

*# end for d in R[]*

*#end for l in interfaces*

This technique is called split-horizon. With this technique, the count to infinity problem would not have happened in the above scenario, as router A would have advertised [A=0], since it learned all its other routes via router D. Another variant called split-horizon with poison reverse is also possible. Routers using this variant advertise a cost of \infty for the destinations that they reach via the router to which they send the distance vector. This can be implemented by using the pseudo-code below.

Every N seconds:

**for** l **in** interfaces:

*# one vector for each interface*

v=Vector()

**for** d **in** R[]:

**if** (R[d].link != i) :

v=v+Pair(d,R[d.cost])

**else**:

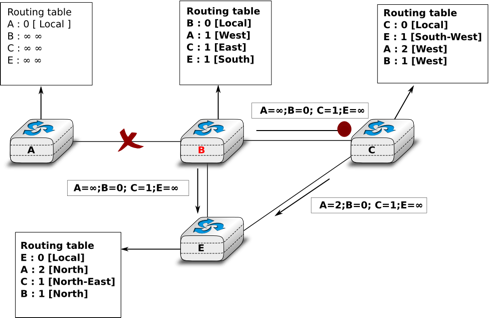
v=v+Pair(d,infinity);

send(v)

*# end for d in R[]*

*#end for l in interfaces*

Unfortunately, split-horizon, is not sufficient to avoid all count to infinity problems with distance vector routing. Consider the failure of link A-B in the network of four routers below.

**[](http://cnp3book.info.ucl.ac.be/1st/html/_images/dv-infinity.png)**

Count to infinity problem

After having detected the failure, router A sends its distance vectors :

* [A=\infty,B=0,C=\infty,E=1] to router C
* [A=\infty,B=0,C=1,E=\infty] to router E

If, unfortunately, the distance vector sent to router C is lost due to a transmission error or because router C is overloaded, a new count to infinity problem can occur. If router C sends its distance vector [A=2, B=1,C=0, E= **∞** ]to router E, this router installs a route of distance 3 to reach A via C. Router E sends its distance vectors [A=3, B=**∞**,C=1, E= 1 ] to router B and   [A=**∞**, B=1, C=**∞**, E= 0 ] to router C. This distance vector allows B to recover a route of distance 4 to reach A.

#### Link state routing

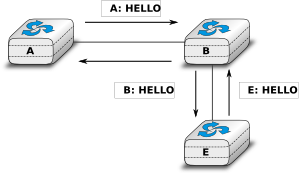
Link state routing is the second family of routing protocols. While distance vector routers use a distributed algorithm to compute their routing tables, link-state routers exchange messages to allow each router to learn the entire network topology. Based on this learned topology, each router is then able to compute its routing table by using a shortest path computation [**[Dijkstra1959]**](http://cnp3book.info.ucl.ac.be/1st/html/bibliography.html#dijkstra1959).

For link-state routing, a network is modelled as a directed weighted graph. Each router is a node, and the links between routers are the edges in the graph. A positive weight is associated to each directed edge and routers use the shortest path to reach each destination. In practice, different types of weight can be associated to each directed edge :

* unit weight. If all links have a unit weight, shortest path routing prefers the paths with the least number of intermediate routers.
* weight proportional to the propagation delay on the link. If all link weights are configured this way, shortest path routing uses the paths with the smallest propagation delay.
* weight=\frac{C}{bandwidth} where C is a constant larger than the highest link bandwidth in the network. If all link weights are configured this way, shortest path routing prefers higher bandwidth paths over lower bandwidth paths

Usually, the same weight is associated to the two directed edges that correspond to a physical link (i.e. R1 \rightarrow R2 and R2 \rightarrow R1). However, nothing in the link state protocols requires this. For example, if the weight is set in function of the link bandwidth, then an asymmetric ADSL link could have a different weight for the upstream and downstream directions. Other variants are possible. Some networks use optimisation algorithms to find the best set of weights to minimize congestion inside the network for a given traffic demand [**[FRT2002]**](http://cnp3book.info.ucl.ac.be/1st/html/bibliography.html#frt2002).

When a link-state router boots, it first needs to discover to which routers it is directly connected. For this, each router sends a HELLO message every N seconds on all of its interfaces. This message contains the router’s address. Each router has a unique address. As its neighbouring routers also send HELLO messages, the router automatically discovers to which neighbours it is connected. These HELLO messages are only sent to neighbours who are directly connected to a router, and a router never forwards the HELLO messages that they receive. HELLO messages are also used to detect link and router failures. A link is considered to have failed if no HELLO message has been received from the neighbouring router for a period of k \times N seconds.

**[](http://cnp3book.info.ucl.ac.be/1st/html/_images/ls-hello.png)**

The exchange of HELLO messages

Once a router has discovered its neighbours, it must reliably distribute its local links to all routers in the network to allow them to compute their local view of the network topology. For this, each router builds a link-state packet (LSP) containing the following information :

* LSP.Router : identification (address) of the sender of the LSP
* LSP.age : age or remaining lifetime of the LSP
* LSP.seq : sequence number of the LSP
* LSP.Links[] : links advertised in the LSP. Each directed link is represented with the following information : - LSP.Links[i].Id : identification of the neighbour - LSP.Links[i].cost : cost of the link

These LSPs must be reliably distributed inside the network without using the router’s routing table since these tables can only be computed once the LSPs have been received. The Flooding algorithm is used to efficiently distribute the LSPs of all routers. Each router that implements flooding maintains a link state database(LSDB) containing the most recent LSP sent by each router. When a router receives an LSP, it first verifies whether this LSP is already stored inside its LSDB. If so, the router has already distributed the LSP earlier and it does not need to forward it. Otherwise, the router forwards the LSP on all links except the link over which the LSP was received. Reliable flooding can be implemented by using the following pseudo-code.

*# links is the set of all links on the router*

*# Router R's LSP arrival on link l*

**if** newer(LSP, LSDB(LSP.Router)) :

LSDB.add(LSP)

**for** i **in** links :

**if** i!=l :

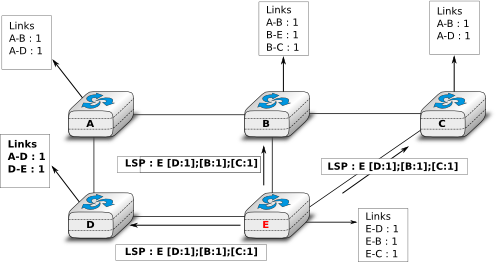
send(LSP,i)

**else**:

*# LSP has already been flooded*

In this pseudo-code, LSDB(r) returns the most recent LSP originating from router r that is stored in the LSDB. newer(lsp1,lsp2) returns true if lsp1 is more recent than lsp2.

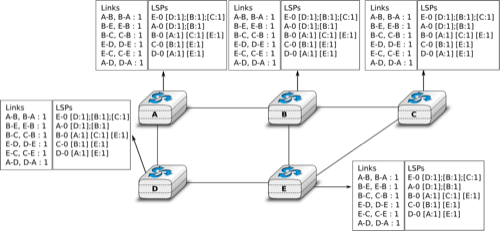
Flooding is illustrated in the figure below. By exchanging HELLO messages, each router learns its direct neighbours. For example, router *E* learns that it is directly connected to routers *D*, *B* and *C*. Its first LSP has sequence number *0* and contains the directed links *E->D*, *E->B* and *E->C*. Router *E* sends its LSP on all its links and routers *D*, *B* and *C* insert the LSP in their LSDB and forward it over their other links.

**[](http://cnp3book.info.ucl.ac.be/1st/html/_images/ls-flooding.png)**

Flooding : example

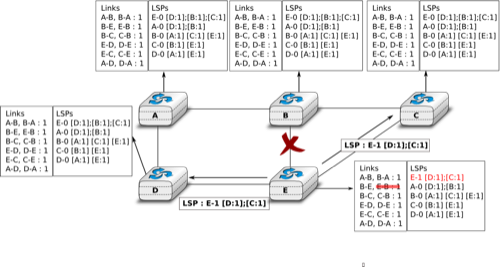
Flooding allows LSPs to be distributed to all routers inside the network without relying on routing tables. In the example above, the LSP sent by router *E* is likely to be sent twice on some links in the network. For example, routers *B* and *C* receive *E*‘s LSP at almost the same time and forward it over the *B-C* link. To avoid sending the same LSP twice on each link, a possible solution is to slightly change the pseudo-code above so that a router waits for some random time before forwarding a LSP on each link. The drawback of this solution is that the delay to flood an LSP to all routers in the network increases. In practice, routers immediately flood the LSPs that contain new information (e.g. addition or removal of a link) and delay the flooding of refresh LSPs (i.e. LSPs that contain exactly the same information as the previous LSP originating from this router) [**[FFEB2005]**](http://cnp3book.info.ucl.ac.be/1st/html/bibliography.html#ffeb2005).

To ensure that all routers receive all LSPs, even when there are transmissions errors, link state routing protocols use *reliable flooding*. With *reliable flooding*, routers use acknowledgements and if necessary retransmissions to ensure that all link state packets are successfully transferred to all neighbouring routers. Thanks to reliable flooding, all routers store in their LSDB the most recent LSP sent by each router in the network. By combining the received LSPs with its own LSP, each router can compute the entire network topology.

**[](http://cnp3book.info.ucl.ac.be/1st/html/_images/ls-lsdb.png)**

Link state databases received by all routers

When a link fails, the two routers attached to the link detect the failure by the lack of HELLO messages received in the last k \times N seconds. Once a router has detected a local link failure, it generates and floods a new LSP that no longer contains the failed link and the new LSP replaces the previous LSP in the network. As the two routers attached to a link do not detect this failure exactly at the same time, some links may be announced in only one direction. This is illustrated in the figure below. Router *E* has detected the failures of link *E-B* and flooded a new LSP, but router *B* has not yet detected the failure.

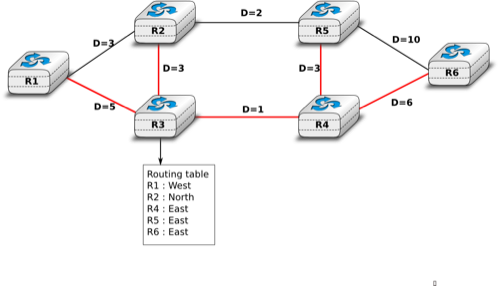
**[](http://cnp3book.info.ucl.ac.be/1st/html/_images/ls-twoway.png)**

The two-way connectivity check

When a link is reported in the LSP of only one of the attached routers, routers consider the link as having failed and they remove it from the directed graph that they compute from their LSDB. This is called the *two-way connectivity check*. This check allows link failures to be flooded quickly as a single LSP is sufficient to announce such bad news. However, when a link comes up, it can only be used once the two attached routers have sent their LSPs. The *two-way connectivity check* also allows for dealing with router failures. When a router fails, all its links fail by definition. Unfortunately, it does not, of course, send a new LSP to announce its failure. The *two-way connectivity check* ensures that the failed router is removed from the graph.

When a router has failed, its LSP must be removed from the LSDB of all routers [**[1]**](http://cnp3book.info.ucl.ac.be/1st/html/network/network.html#foverload). This can be done by using the *age* field that is included in each LSP. The *age*field is used to bound the maximum lifetime of a link state packet in the network. When a router generates a LSP, it sets its lifetime (usually measured in seconds) in the *age* field. All routers regularly decrement the *age* of the LSPs in their LSDB and a LSP is discarded once its *age* reaches *0*. Thanks to the *age* field, the LSP from a failed router does not remain in the LSDBs forever.

To compute its routing table, each router computes the spanning tree rooted at itself by using Dijkstra’s shortest path algorithm [**[Dijkstra1959]**](http://cnp3book.info.ucl.ac.be/1st/html/bibliography.html#dijkstra1959). The routing table can be derived automatically from the spanning as shown in the figure below.

**[](http://cnp3book.info.ucl.ac.be/1st/html/_images/ls-computation.png)**

Computation of the routing table

[http://cnp3book.info.ucl.ac.be/1st/html/network/network.html]