Chapter 5(Module E): Internetworking and Routing

In the previous chapter, we have covered the fundamental operations of multiplexing and multiple access techniques. In this chapter, we first consider interconnected networks, and investigate related problems that need to be resolved so that a host can send information to an arbitrary host in the interconnected networks.

Then, we will turn our focus to Internet Protocol, arguably the most popular protocol to support internetworking. Starting with IPv4, we will look at IP addressing scheme and how different functionalities of IPv4 are implemented. Next, IPv6, the next generation of Internet Protocol will be described to show its operations and enhancements over IPv4.

The final topic covered in this chapter is internetwork routing. At first, we will examine the issue of providing a good routing algorithm and different ways to quantify the goodness of a communication path. After understanding these fundamental concepts, we will dive into the world of routing by investigating the two commonly used routing algorithms in the Internet, namely Dijkstra and Bellman-Ford algorithms. Furthermore, to resolve the issue of routing scalability at the scope of the Internet, hierarchical routing will be also discussed.

5.1 Introduction to Internetworking

To realize a worldwide data network, various communication segments have to be connected together and, new issues related to internetworking arise. In this section, we first identify the various issues to be considered in an internetworking environment. We will next discuss the potential functions to be implemented for internetworking. Finally, we will describe the structural design of the network devices that enable the interconnection of networks.

5.1.1 Issues related to internetworking

Figure 1 illustrates an interconnection of multiple networks. In order to construct this internetworking environment, *router* is introduced as a new network component to connect these networks together. When a user device wants to communicate with a content server or vice versa, routers in this internetwork must forward data packets one hop at a time towards the destination until they reach the target device. With multiple networks of different types, sizes, capability, and/or protocols, transferring data between them is a challenging task.

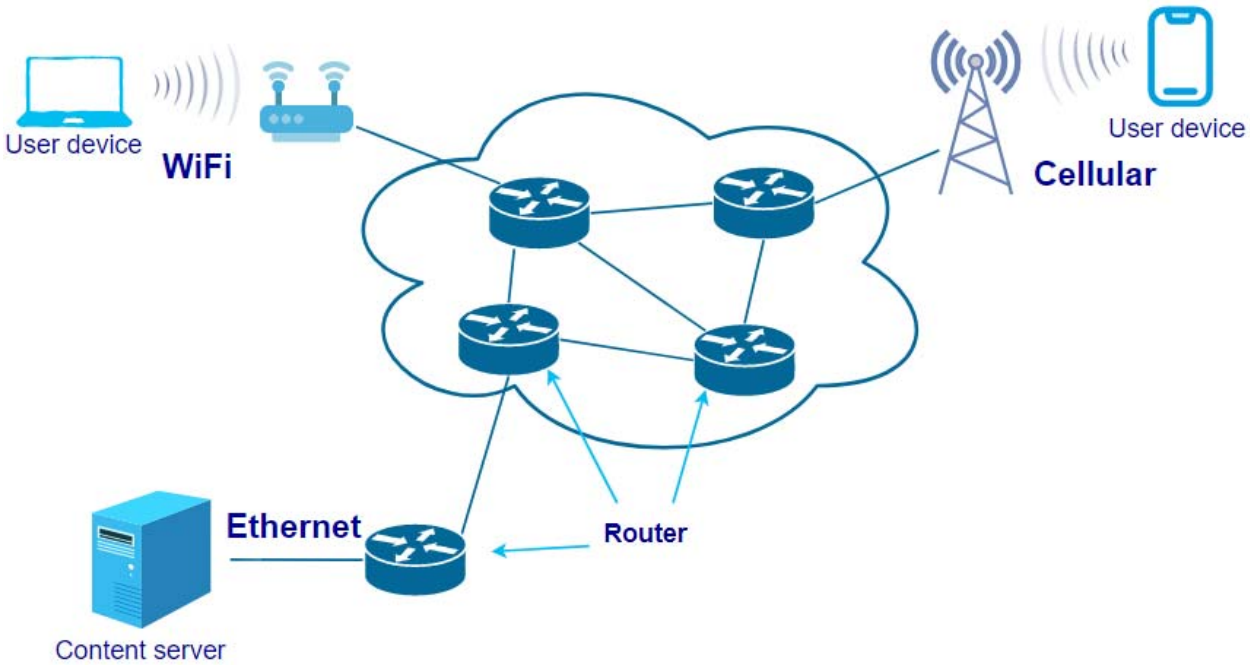


Figure 1: Interconnection of networks.

First, there is a *global addressing* issue. On a vast internetwork like the Internet, in order to make data delivery possible, the source node must be able to uniquely identify the destination node from the billions of participated nodes. While the protocols operate on a directly connected link can also identify the communicating nodes on this link, e.g., in the case of HDLC, this addressing scheme is local and not compatible with other networks using different protocols. Even among all networks using the same link-level protocol, it is not possible to tell if a node belongs to one network or another.

Second, there is a *path determination* issue from one node to another node in this multiple-network environment. With an unknown number of networks of different shapes

and sizes separates the communicating nodes, there is a need to determine a path through these many networks so that a packet can follow towards its destination. In addition, in a large set of interconnected networks such as the Internet, changes in topology and connectivity can often happen. At anytime, there can be a network or a link that is added, removed or augmented, making the originally established path no longer valid nor optimal.

Third, there is a *Quality of Service* (QoS) issue in transferring the packets through the interconnected networks. The interconnected networks support not only one data stream for one source-destination pair but also many other pairs and traffics from the participating nodes can be very bursty. As a result, traffic flowing over various links can largely fluctuate, introducing possible congestions and bottlenecks in the interconnected networks. Consequently, those congestions and bottlenecks may cause packet losses, and/or increase in packet delivery delay, which can degrade the Quality of Service of related applications in this common internetworking environment.

Fourth, there is an *interoperation* issue among different types of networks. Since those networks in the Internet can employ different protocols, and operate on different types of media, they require different types of framing structures, signalling schemes, and channel access procedures. As a result, it is challenging to adapt a data frame that is sent on one type of network to be transmitted or relayed on a totally different type of network.

5.1.2 Functions for internetworking

The internetworking functions reside in the network layer in the TCP/IP protocol stack. The ultimate goal of this layer is to support end-to-end packet delivery between two arbitrary nodes in the network. While the *applications* are only required at the two communications end nodes, the *internetworking* functions must be implemented at the communications end nodes as well as all the routers in the network. As a result, the design of *internetworking* functions must consider the network utilization, complexity and cost of networking devices.

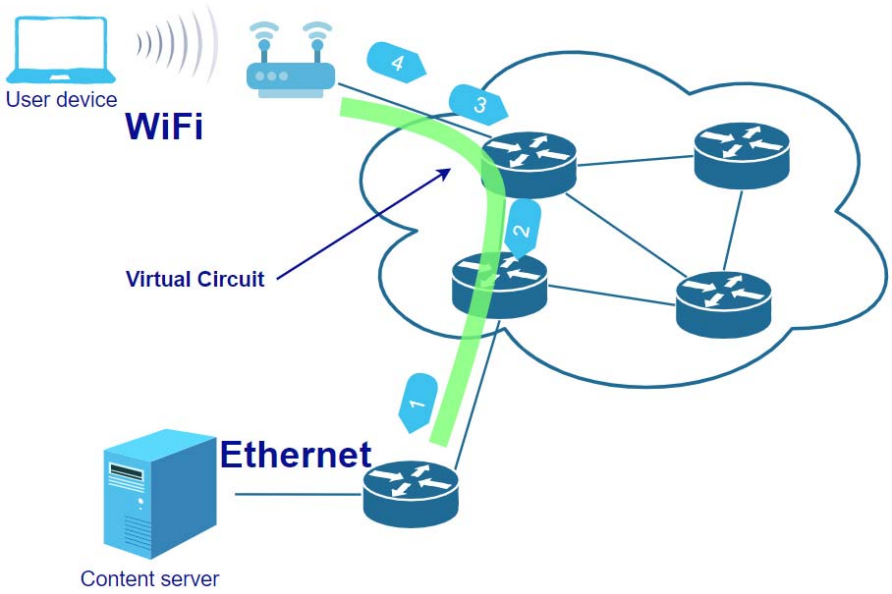


Figure 2: Packet forwarding in connection-oriented approach.

First, to carry user traffic over this common internetworking environment, there are two fundamental service models: *connection-oriented* and *connectionless*.

In the *connection-oriented* approach, a connection, called *Virtual Circuit* (VC), is established for traffic transfer as illustrated in Figure 2. With the VC established in advance, the routers along the VC can reserve communications resources for this traffic flow in advance and hence, can guarantee a certain QoS level for the user traffic. User packets using this VC, need to carry a short identification for the routers to determine the appropriate VC. However, full source and destination addresses may not be needed. Since all user packets traverse through a same path established by the VC, they arrive in order at the destination. Connection-oriented scheme requires the setup, maintenance and release of a VC for each transmission, which may incur additional delay in data transmission as well as additional overhead for the routers along the VC. Moreover, if a link or a router along the path of the VC fails during the transmission, the VC must be re-established, which may corrupt the ongoing data in the network.

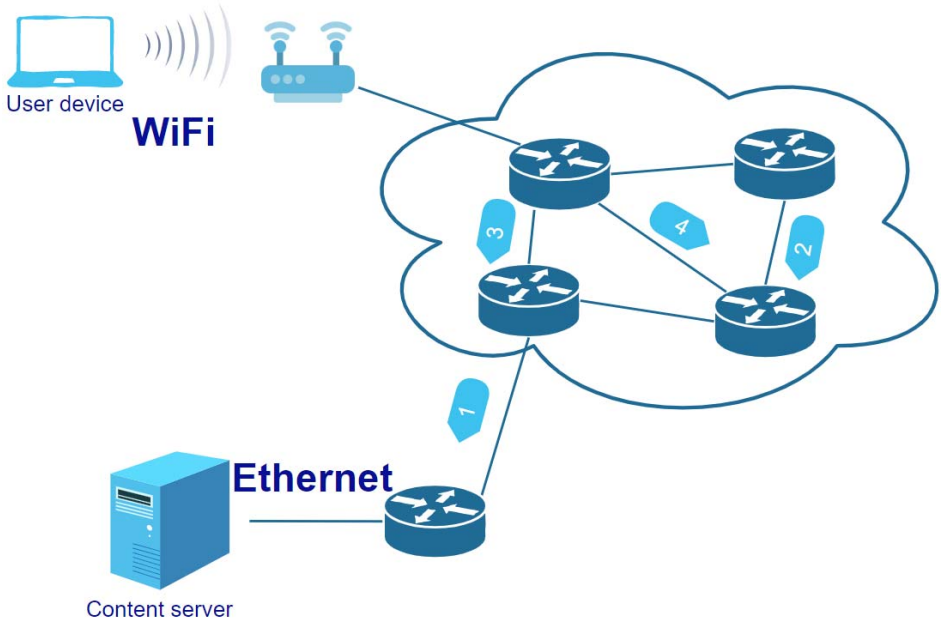


Figure 3: Packet forwarding in connectionless approach.

In contrast, the *connectionless* approach does not require a connection to be setup before each transmission. The transmitter simply starts transmitting packets into the network and depends on the capability of the network to forward its packets to the destination according to the destination address embedded in the packets. Since no connection is established in advance, user packets are routed independently from each other and may take different paths. Connectionless scheme is simple to implement and does not incur overhead in the network devices for connection setup, maintenance and release. Moreover, if one router fails, user packets can be re-routed in an adhoc manner without the need for connection re-setup. However, since connectionless approach depends on the network capability for carrying user packets, it does not offer any guarantee on packet delivery, QoS level and it does not guarantee that the received packets follow their transmission sequence as illustrated in Figure 3. As such, connectionless service model is sometimes referred to as the *best-effort*

service model, which implies that the network will try its best to transfer user packets to their destination but does not guarantee the packet delivery nor QoS.

Second, for internetworking to be possible, the network layer must provide an *abstraction* of the operation details at the link level. In other words, it must allow data to be transferred seamlessly between multiple networks with different employed protocols as if they are transmitting on a common virtual medium, unaware and regardless of the number and types of links and routers between the two communicating nodes. This function is more complicated than it sounds. When moving the data from one link to the next, the involved router must adapt the framing structure and the signalling procedures to that of the next link. This needs the encapsulation and de-capsulation process of the layering structure. By wrapping the link-dependent frame around the network layer packet, routers can easily adapt the requirements of one protocol to another as illustrated in Figure 4. However, there are cases where this abstraction can not be easily achieved without intercepting the network packet integrity. For instance, when the maximum frame sizes of the two protocols operate on the two links are different. In this case, if the incoming frame size is larger than the maximum frame size supported in the outgoing link, the data packet must be partitioned into multiple smaller packets to be able to fit into the next link. As a result, there must be a way for the destination node to indicate that the received packet is only a fraction of the transmitted packet and a procedure to re-assemble these fragments must also be specified.

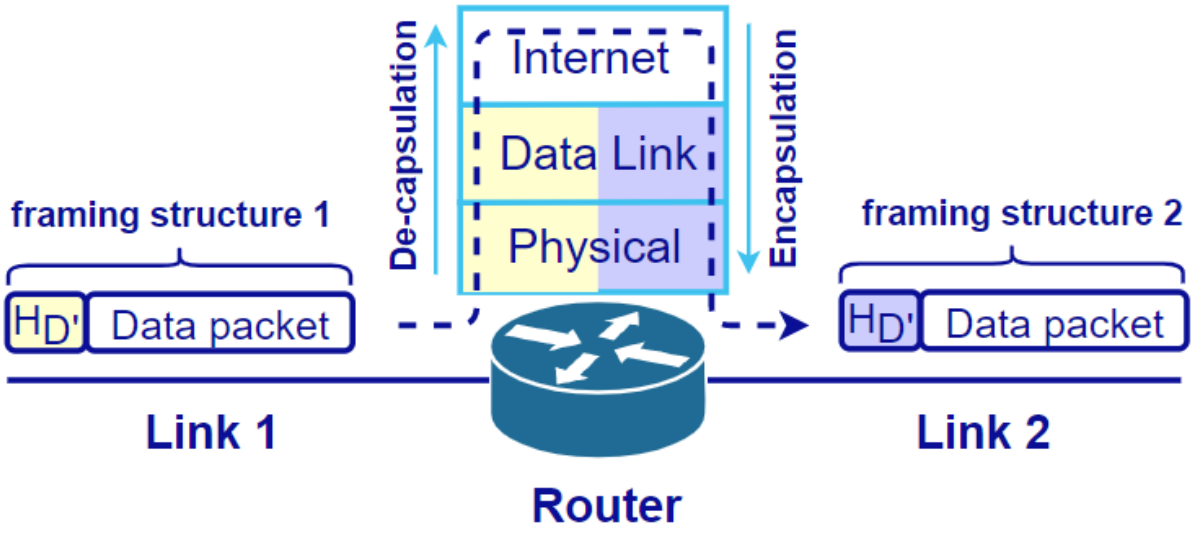


Figure 4: Interoperations between different link-layer protocols.

Third, a *globally significant addressing scheme* to uniquely identify each device on the internetwork must be defined. Different from the link-layer address that is *locally* significant only on that link, a unique identification of a device in an internetworking environment must have a *global* significance and be flexible. Such a logical addressing scheme must be *hierarchical*[1], containing information about the network to which the node is currently attached, and a unique identification of the node within that network. For this, an internetworking address must contain at least two parts, the *network* part and *host* part. For

[1] This hierarchical scheme is similar to the mailing address structure containing the street number, street name, district, city, province, and country.

routing convenience, the network part is normally placed before the host part. As a network-level address contains also the network identity, a logical address structure must be flexible to cope with the mobility of a device from one network to another.

Another function required for internetworking is *routing*. As illustrated in Figure 1, when a host needs to send data to another host in a different network, it encapsulates the data into packets and sends them to its closest router. This router will forward these packets to the next router on the path to the destination and this forwarding process continues until the packets reach their destination or being dropped due to network events such as congestion. In this forwarding process, it is assumed that the router knows in advance which direction to forward the packet. However, this assumption is not trivial in an internetworking environment with a large number of links, networks and routers. As a result, one of the most critical functionalities required for internetworking is to provide a router with the capability to determine the suitable route to forward each packet it receives. Ideally, this path finding procedure must aware and adapt to the changes in the internetworking environment. Routing is desired to be simple to be implemented in every routers but still capable of providing accurate and robust routes for packet forwarding.

Moreover, as a router forwards a packet from one link to another, it may find that the destination link is congested with many packets queuing for transmission. In this case, the router can be implemented with *scheduling* mechanisms to prioritize a certain type of packet while dropping some other packets to enforce QoS or ensure fairness among the data flows.

The network layer should also support *error reporting* capability to inform end devices of the errors related to their transmission sessions. For example, if the destination node, or network is not available or the destination node is corrupted or goes offline during the transmission session, the network must be able to report this event to the source node.

Furthermore, there is a set of functions that can be optionally provided by the network layer to enhance the network usability. One example of such function is the ability to do resource allocation to minimize the interference among the data flows and guarantee the minimum available resource for communication for the user applications to function properly. By doing this, internetworking can guarantee packet delivery without loss, with a certain maximum transmission delay or minimum bandwidth.

5.1.3 A generic router structure

As discussed above, the two most important functions of a router are *routing* and *forwarding* of packets it receives. With these two functions, routers become the nuts and bolts that hold the Internet together. As such, it is important for us to spend some time to understand the basic structure of a router. Figure 5 illustrates the simplified internal structure of a generic router, including four main components: a route processor, many input and output interfaces[2] and a switching fabric.

Route processor

The route processor is in charge of routing function. It exchanges routing information with the other routers in the network, processes them according to some routing protocols and the policies defined by the network administrator. The output of this processing is *the forwarding table*, which gives instructions of the output interface that a packet should be

[2] In real implementation, input and output interfaces are generally grouped into line cards.

forwarded to in order to reach the next hop on its path to the destination. Since the network address may be very long, processing directly based on the network address can be very memory-intensive and also introduce a long delay due to searching operations through such gigantic table.

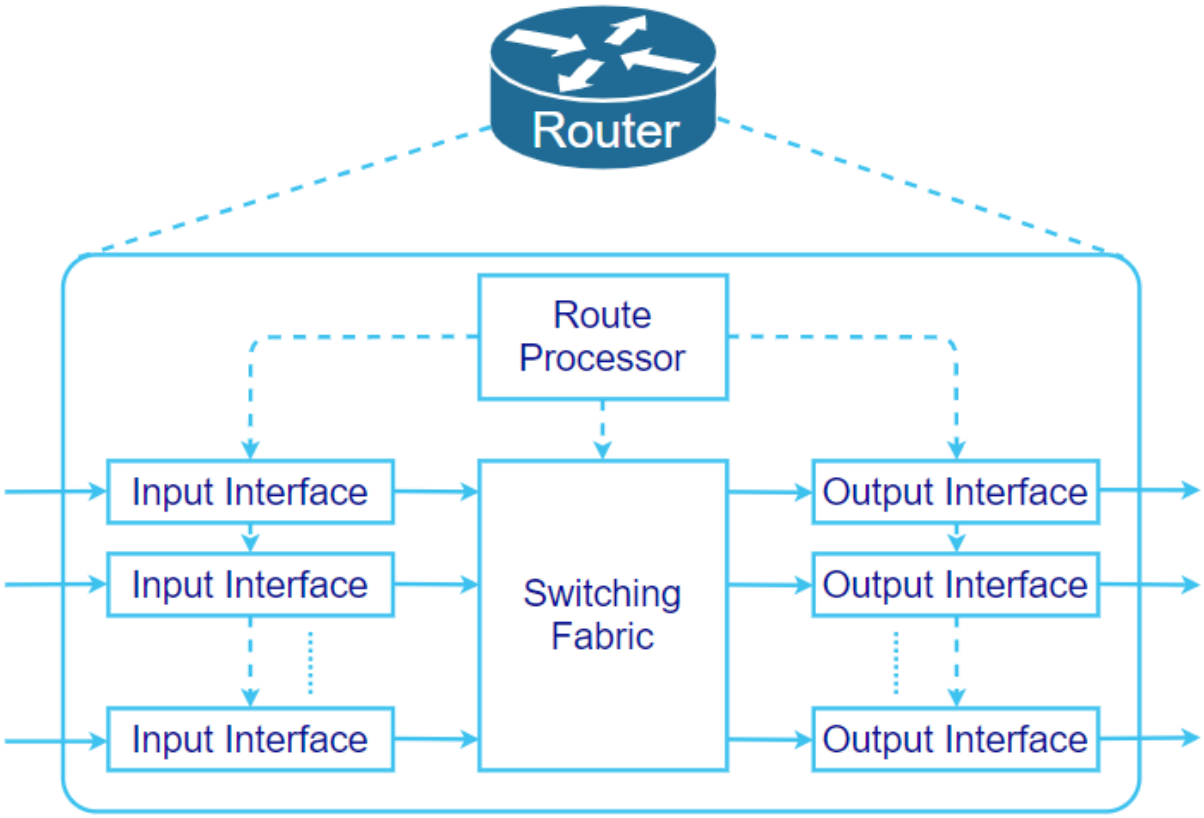


Figure 5: Internal structure of a generic router.

Instead, it is sufficient and much more efficient to use only the network part of the address to build the forwarding table. Each of such network part is called a prefix as it is used to match the first highest significant bits (hence prefix) in a destination address. Each entry in a forwarding table consists of a prefix and the corresponding output interface that the packet should be forwarded to, as illustrated in Table 5.1. It is noted that multiple different prefixes can be mapped to the same output interface in a forwarding table because the link connected to this output interface can be shared among multiple destinations.

Table 5.1: An example of a forwarding table

Forwarding Table	
Prefix	**Output Interface**
0011	1
0101	1
1011	2
10111	3

A forwarding decision can be made by comparing the destination address of the packet with the prefixes in the forwarding table. If the destination address matches multiple prefixes, the longest prefix will be selected for the forwarding decision. The rational behind this is that the longest prefix contains the most specific and accurate information about this address and hence, should be used.

For example, consider an address space of 8 bits and a forwarding table as shown in Table 5.1. A packet with a destination address of **00111010** will match to the first entry in this forwarding table (starts with **0011**) and hence, should be forwarded to output interface 1. On the other hand, another packet with a destination address of **10111010** will match both the third and the fourth entries of the above forwarding table; however, the fourth entry has the longest prefix that matches this destination address. Hence, the packet will be forwarded to the output interface 3. How about a destination address of **10110111**? In this case, because the fourth entry no longer matches this address; hence, it should be forwarded according to entry 3, i.e. to output interface 2.

In all of the above examples, we have not covered the case when the destination address does not match any entry in the forwarding table. In this case, the router does not know where to forward the packet to; hence, the packet will be dropped and depending on the implementation, an error reporting message will be returned to the sender of the packet to notify this error.

It is noted that except for high-end routers in the core of ISPs, small routers such as those we use in our home does not have enough capacity to store all the prefixes in the vast Internet. To solve this problem of dropping packets to an unknown destination, the forwarding tables of these routers usually contain a **default route** pointing to an interface that connects to an upstream router. For example, in those small routers in our homes, it is likely that their forwarding tables contain only a default route pointing to the upstream ISP router and not the specific routes on the Internet.

In addition to the path determination tasks that are discussed above, the route processor also performs management and coordination tasks to ensure the smooth operations of the overall router.

Input Interfaces

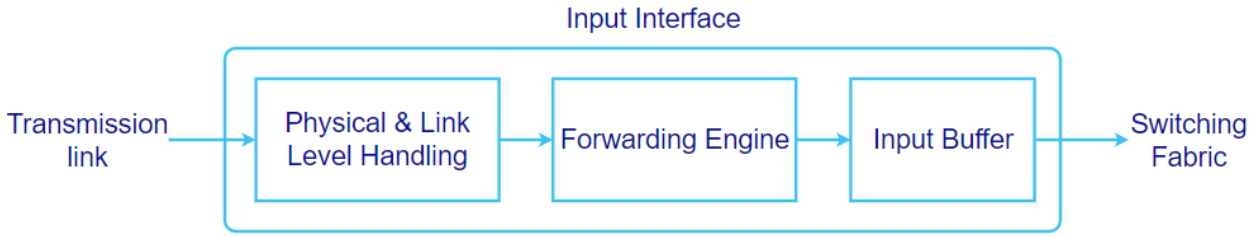


Figure 6: Structure of an input interface.

In the first function block as shown in Figure 6, an input interface performs all tasks required to physically attach to the link and interpret the signalling scheme that is used on the link. In addition, it also conducts all tasks according to the link-level protocol that is used on the link, including the channel access method, framing structure and related procedures. The de-capsulation process as depicted in Figure 4 is also executed at the input interface to

strip the link-specific details and keep only the network-level packet that can be adapted to any other specific requirements of the output link.

After receiving the packets from the link, an input interface will consult the resulting forwarding table of the route processor to find the output interface for the corresponding packet. As the number of interfaces in a router increases, polling to consult the route processor for every received packet becomes inefficient and will eventually make the communications with the route processor a bottleneck of the router. In order to speed up this lookup operations, in high-performance routers, a dedicated forwarding engine is built into each line card as shown in the second function block in Figure 6. This forwarding engine will receive the forwarding table from the route processor and store it locally in the line card using a dedicated high-speed memory. As a result, the lookup can be executed more efficiently, greatly enhancing the forwarding performance of the router.

Another important feature of the input interface is the presence of an input buffer to temporarily store the received packet if the switching fabric is not ready for transferring the packet. This can happen when the corresponding output interface is currently busy to receive a packet from another input interface and hence, blocks the packet from entering the switching fabric.

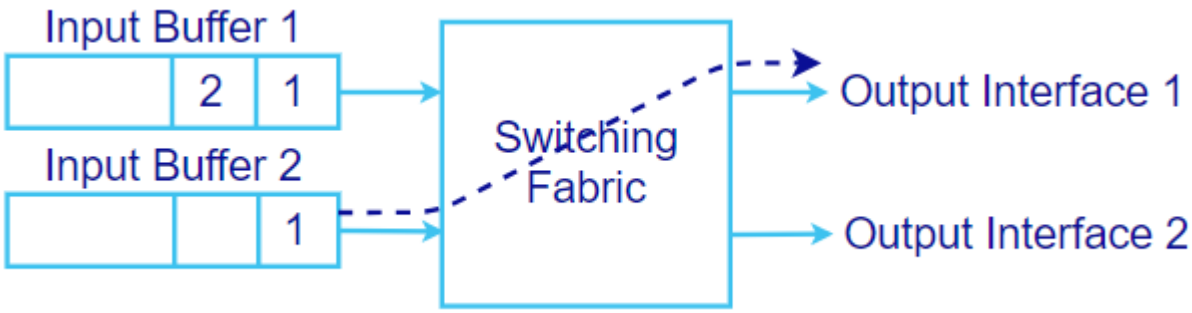


Figure 7: Head of the Line blocking.

The input buffer structure also plays a critical part in the performance of the router. Figure 7 illustrates an example of the so-called *Head of the Line (HOL) blocking* phenomenon resulting from using simple *First-in-First-out (FIFO)* buffering structure. In this figure, the numbers presented in each of the input buffer packet specify the output interface that the packet should be transferred to according to the forwarding table. This figure shows that the switching fabric is currently serving the packet from input buffer 2 to output Interface 1 and hence, the first packet of the input buffer 1 to be forwarded to the same output interface 2, is blocked. This packet in turn blocks all the packets behind it from entering the switching fabric regardless of their destinations. For example, the next packet in Input Buffer 1 is blocked although the output interface 2 is currently free. HOL blocking severely degrades the throughput of a router, e.g., in some cases, it can limit the throughput of the router to only 59% of the total link bandwidth [1].

One approach to mitigate this HOL blocking phenomenon is to employ a more complex queuing structure at the input port. For example, a number of queues equals to the number of output interfaces can be utilized instead of one simple FIFO queue. Another approach to eliminate the input queuing is increasing the speed of the switching fabric.

Switching Fabric

The role of the switching fabric is to move the packets from an input interface to the corresponding output interface in the shortest duration of time. As illustrated in Figure 5, as the number of input and output interfaces grows, more traffic will be passed on the switching fabric, making it the bottleneck in the router structure. As such, many switching structures were proposed for the implementation of switching fabric. In fact, the evolvement of switching fabric structures is quite involved and deserves a section on its own. As such, the descriptions and discussions on this topic will be elaborated more in Chapter 9.

Output Interfaces

The basic function of the output interfaces is to receive the packets from the switching fabric and send it to the connected transmission link as illustrated in Figure 8. In order to do this, the output interface must adapt the packet to the framing structure, protocol and the physical signalling schemes of the output transmission link. The encapsulation process as depicted in Figure 4 is also executed at the output interface.

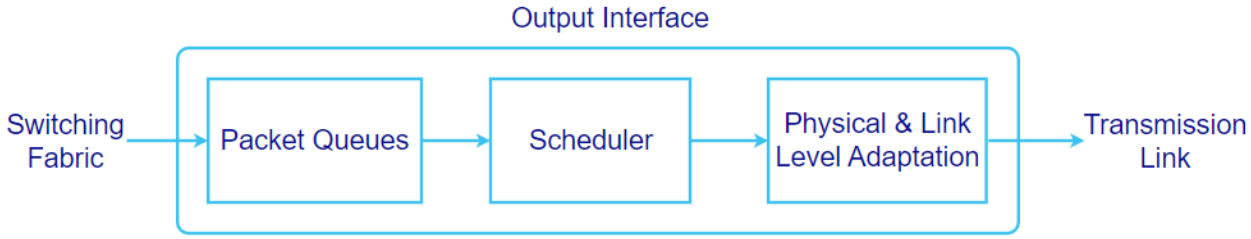


Figure 8: Structure of an output interface.

However, when there are multiple traffic flows from many input interfaces to the same output interface, the transmission speed of the output interface may not withstand the packet arrival rate. In this case, the excess packets will be cached in a temporary buffer in the output interface namely, queues, while waiting for its turn to be served. The queuing delay caused by this waiting period is the main factor that contributes to the packet delay through a router. Because this delay varies depending on the traffic demand imposed on the routers, it is also the main factor that contributes to the **end-to-end delay variations**. Moreover, when the traffic demand for the output interface is larger than its transmission rate, these queues will eventually overrun. In this case, excess packets will be dropped, causing **packet loss**.

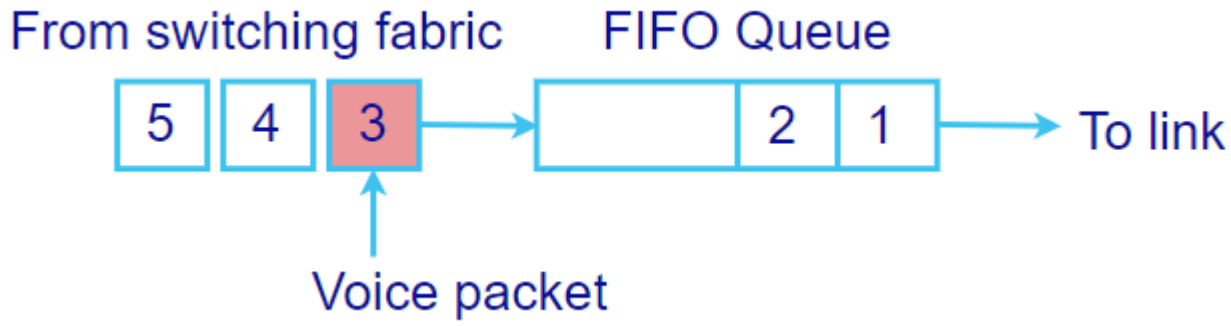


Figure 9: A FIFO queue.

In a simplest case, the output interface employs a single **FIFO queue** for all of its arrival traffic as illustrated in Figure 9. In this scenario, the queue sends out packets in the same

order as they arrived. As a result, a packet in the queue must wait for all of the packets that arrives ahead of it before it can be served. For instance, in Figure 9, packet 3 must wait for the serving of packet 1 and 2 before it can be served. This first-come-first-serve approach is simple and can be used if the packets are from a time non-sensitive application. However, as the number and types of applications grows, this assumption is no longer valid. Assuming that packet 1, 2, 4, 5 are from a time non-sensitive file downloading application while packet 3 is from a time sensitive application such as a voice conferencing application. Having to wait for the serving of two packets ahead of it may violate the QoS requirements of packet 3, degrading the voice service or even make the transmission of packet 3 meaningless at the time due to its failure in meeting the delay deadline. Moreover, a simple FIFO queue also gives rise to unfairness. Imagine a scenario where an aggressive user sends traffic all the time at high speed and fill up the FIFO queue with only its own packets. This traffic flow will steal most of the bandwidth of the link as there is no space in the queue for the packets from other users. To improve the QoS and fairness of different services traverse through the router, different queuing methods including priority, fair and weighted fair queuing.

Priority queuing

Different from FIFO queuing, priority queuing divides the available buffer into multiple queues, each for a type of service or application. Figure 10 illustrates a priority queuing system with three levels of priorities (High, Medium and Low). The packets from the switching fabric are classified by a traffic classifier before putting them into queues with appropriate priorities. The traffic classifier normally classifies the type of traffic based on the information in the packet header such as port number or class of services. In each transmission period, the scheduler always selects a packet from the highest priority queue to transmit. If there is no packet awaits in the highest priority queue, the next highest priority queue will be selected for transmission and so on. In this way, if delay sensitive packets are assigned into a high priority queue, they can be served immediately without waiting for other packets with lower priorities, hence, decreasing their delay when traversing the router.

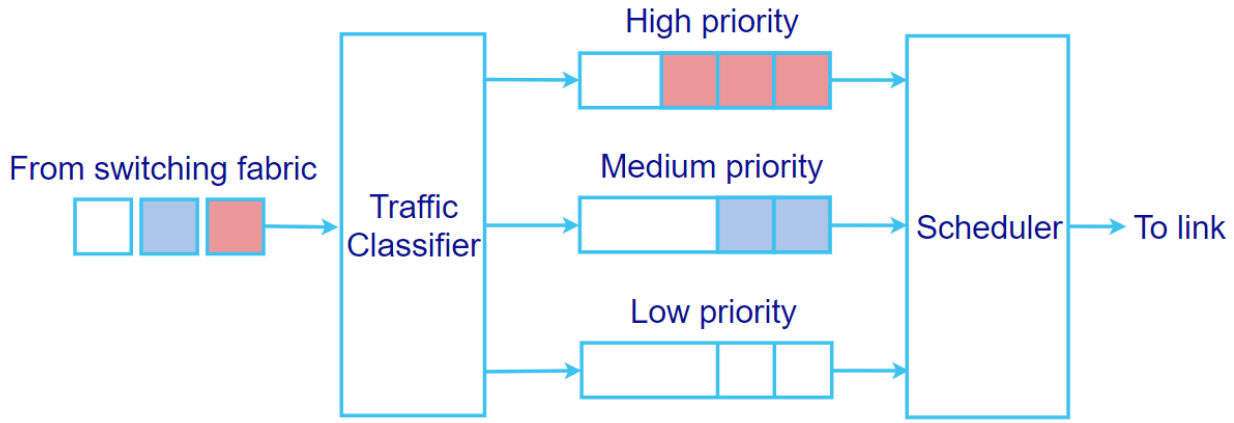


Figure 10: Priority queuing.

As illustrated above, priority queuing can support different levels of services; however, it does not give any incentive about the transmission of low priority packets. For instance, if the high priority queue is constantly filled with packets, it will hog all the available bandwidth and leaves no chance for lower priority packets to be transmitted.

Fair queuing

Trying to strike a middle ground between FIFO and priority queuing, fair queuing aims at supporting equal bandwidths for equal traffic classes. As illustrated in Figure 11, the arriving packets are also classified into multiple classes, each is associated with a queue. The queues are served in a round-robin manner. For example, a packet from traffic class 1 queue will be served, then, it will be the turn of a packet from traffic class 2 queue, and so on.

If all of the packets in the system have the same size, fair queuing allocates exactly a bit rate of R/n bps to each of the traffic class, hence, achieving its fairness goal, where R is the bit rate of the output interface and n is the number of traffic classes. It is interesting to see that when one traffic class does not have any packet to send, fair queuing automatically scales the bandwidth share accordingly to $R/(n-1)$ bps. This means that fair queuing is able to **guarantee a minimum supported bandwidth** for each of the traffic class while allowing maximal bandwidth utilization by automatically adjusting the offered bandwidth with the number of traffic classes.

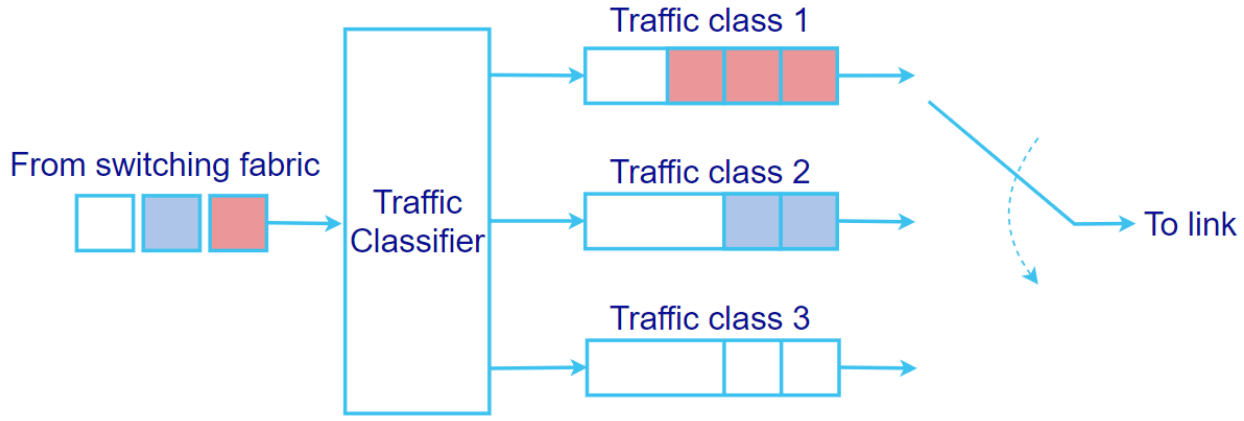


Figure 11: Fair queuing.

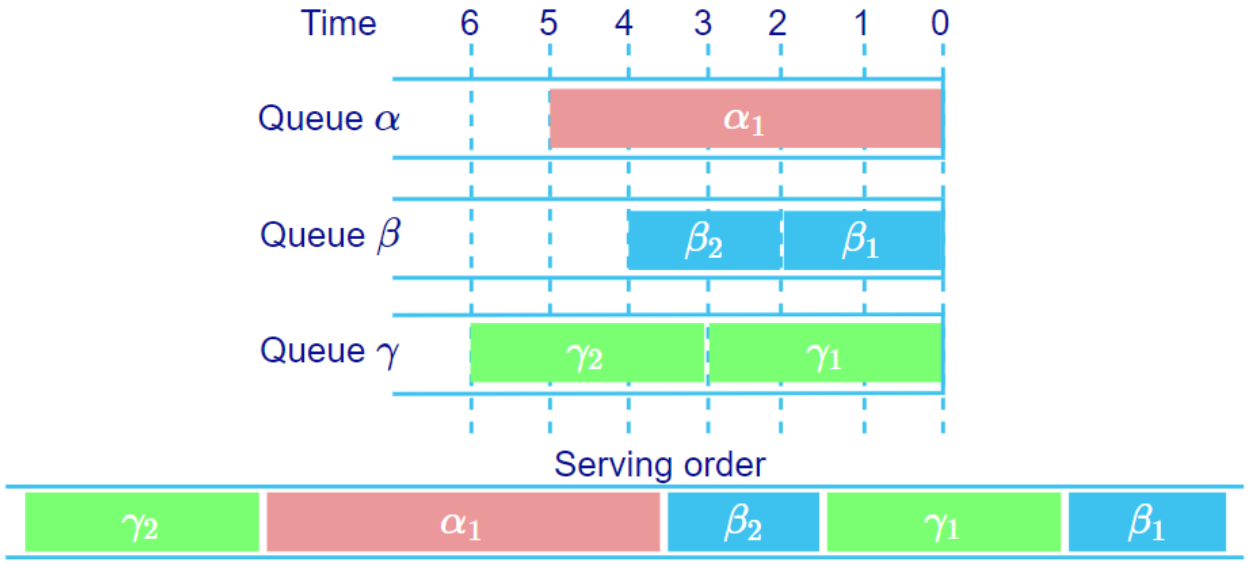


Figure 12: An example of bit-by-bit Fair Queuing.

However, this fairness goal is no longer true when the packet sizes are not equal. In this case, by rotating turns on a per packet basis, the mechanism tends to favour traffic classes with larger packet size by giving it more transmission time every turn and hence, more bit rate. To resolve this unfair issue when the packet sizes are different, a **bit-by-bit fair queuing** algorithm is used instead. In a bit-by-bit round robin algorithm, each packet in the system will be calculated for their expected finished time and will be served in the order of this expected time.

Suppose that there is only one queue in the system. Let A_i be the arrival time of packet i, and P_i be the serving time of the packet, the finished time when packet i is fully served can be expressed as in equation (5.1). In this equation, the terms in the max() operation exist because if there is no other packet in the queue, packet i should be served right after it arrives, otherwise, it must wait for the packet in front of it to be served first.

$$F_i = \max(F_{i-1}, A_i) + P_i \quad (5.1)$$

Now, let's move to the case of multiple queue in the system. In this case, the expected finished time will be calculated for each packet in the system according to equation (5.1). However, it is noted that F_i no longer tells the exact finished time that the packet will be fully served as the fair queuing system may alternate between queues. Instead, F_i is used as a priority for the system to choose the next packet for service, in the order from lowest to highest.

Table 5.2: Serving order calculation for the example in Figure 12

Packet	A_i	P_i	F_i	Serving order
$\boldsymbol{\alpha_1}$	0	5	5	4
$\boldsymbol{\beta_1}$	0	2	2	1
$\boldsymbol{\beta_2}$	2	2	4	3
$\boldsymbol{\gamma_1}$	0	3	3	2
$\boldsymbol{\gamma_2}$	3	3	6	5

Figure 12 illustrates an example of bit-by-bit fair queuing with three queues. The time axis indicates the packet arrival time as well as the packet length in terms of serving time. By applying equation (5.1) for packets in each queue, the serving order of each packet can be derived accordingly as in Table 5.2.

Weighted fair queuing

As discussed above, fair queueing helps to guarantee a minimum bandwidth to be distributed evenly to all the traffic classes; however, this evenly distributed bandwidth behaviour may not be desirable in many cases. For instance, we may want to allocate more bandwidth for video traffic to meet its desired QoS level. Fortunately, fair queuing can be tweaked to integrate a weighting factor for configurable bandwidth sharing among the many classes of traffic. This modification of fair queuing is called **weighted fair queuing**.

As illustrated in Figure 13, the structure of weighted fair queuing is similar to that of fair queuing. However, there is a weight factor w_i assigned to each of the traffic class to materialize the appropriate priority. As a result, equation (5.1) can be modified to incorporate this weight as in (5.2).

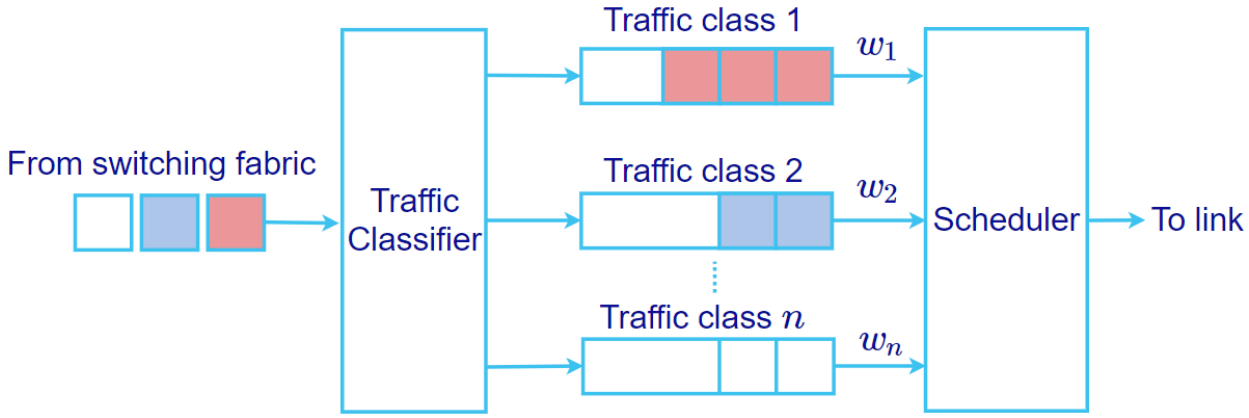


Figure 13: Weighted Fair Queuing

$$F_i = \max(F_{i-1}, A_i) + \frac{P_i}{w_i} \tag{5.2}$$

It can be interpreted that by scaling P_i by w_i, the modification in equation (5.2) controls the appropriate traffic class to transfer more or less bits each time it is served, hence, supervising the link bandwidth portion that the traffic class will get. For instance, if traffic class i has $w_i = 2$ while the other classes have $w_j = 1$ $(j \neq i)$, this configuration allows F_i to carry twice as many bits and hence, effectively allocates the traffic class i double the link bandwidth in comparison to the other traffic classes. As a result, the guaranteed bandwidth share for traffic class i, R_i, can be expressed as in equation (5.3), where R is the total link bandwidth. The denominator in equation (5.3) is the sum of all weights assigned to all traffic classes configured in the system.

$$R_i = \frac{w_i}{\sum_{j=1}^{n} w_j} R \tag{5.3}$$

5.2 The Internet Protocol

In the previous section, we have studied the basic issues, and the set of functions that can be implemented at network layer to facilitate internetworking. In this section, we will turn our focus to the **Internet Protocol (IP)**, which is the widely used network layer protocol today.

IP chose the connectionless service model to support packet transfer across the Internet. Despites many of its drawbacks as discussed in the previous section, a connectionless service model helps to keep the implementation simple enough to provide lightweight, low cost internetworking while maintaining an adequate performance. In addition, IP also defines encapsulation and de-capsulation structure to ensure a seamless operation in a heterogeneous internetworking environment. Moreover, IP also defines a flexible addressing scheme that is convenience for both identifying communication devices and routing purpose. Since there are currently two versions of IP that is running simultaneously on the Internet, namely IPv4 and IPv6, we will start first with the design and operations of IPv4, then, we will move on to discuss IPv6.

5.2.1 IPv4 packet structure

For a network protocol to materialize its services and functionalities, a header must be added to every packet embedding information on how the routers along the traversed path or the destination should do to properly handle the packet. Consequently, the power, effectiveness and versatility of a protocol are designed and integrated in these few bits in its header. As a result, seemingly a boring and daunting burden at first, learning and understanding the fields in a protocol header is the first important step in grasping the capabilities of a protocol and how the support for different functionalities of that protocol is materialized.

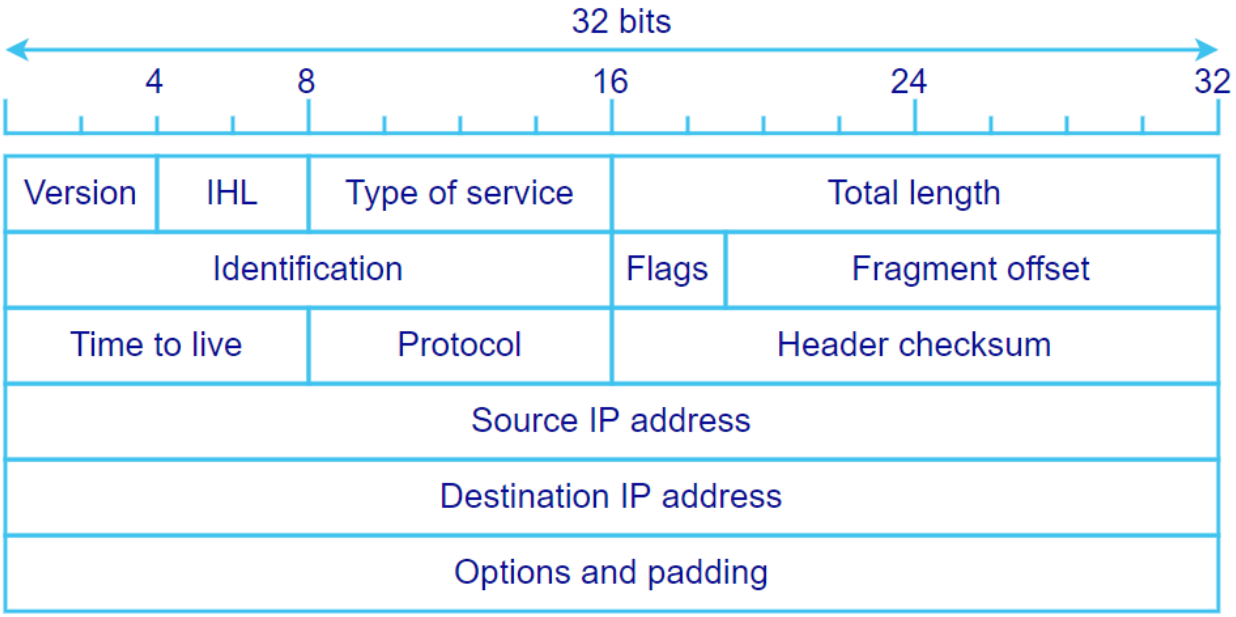


Figure 14: IPv4 header structure.

Figure 14 illustrates the structure of the IPv4 header structure with a minimum size of 20 bytes when no option is included and a maximum size of 40 bytes with options. The key fields in an IPv4 header are described as the following:

- **Version (4 bits)**: this field indicates the version of the current IP packet. For an IPv4 packet, this field contains a binary representation of number 4 (0100). It is designed to instruct the receiving router how to interpret the fields that follows afterwards. For example, an IPv6 packet would have a binary representation of number 6 (0110) in this field, indicating a very different header structure for IPv6, which will be described latter in this chapter.

- **Internet header length (IHL - 4 bits)**: since the header of an IP packet can have a variable length, the IHL field is used to indicate the header part of the packet. The value in this field specifies the length of the current IP header **in blocks of 32-bit words (4 bytes)**. For example, an IHL of 5 is used to specify the minimum header length of an IPv4 packet with $5 \times 4 = 20$ bytes.

- **Type of service (ToS - 8 bits)**: this field was designed as a mean to differentiate the various traffic classes that a packet belongs to. For instance, it is desirable to allocate more resources to real-time traffic such as video or IP telephony to satisfy their QoS requirements

while the transfer of a file transfer session can tolerate more in terms of delay and offered bandwidth. As a result, by assigning these ToS bits to the appropriate packets, the routers along the traversed path can distinguish and assign these packets to the right traffic classes, with suitable weights for their weighted fair queues, for example. However, whether these ToS bits are used at all or the use of a particular resource allocation schemes according to these bits may vary from router to router, depending on their administrations, capabilities and configurations. A standardized definition of these bits is specified in RFC 3260 [2] by Differentiated Service Working Group.

- **Total length (16 bits)**: this field specifies the length of the entire packet in bytes, including its header. It is noted that different from the IHL, no further calculation is needed to interpret this field. With 16 bits available, the theoretical maximum size of an IP packet is 65,535 bytes. However, in practice, packet sizes are normally much smaller than this maximum number due to the restriction in frame size of link-layer protocols. When a packet size is larger than that supported at the link-layer, the packet must be partitioned into smaller fragments. This fragmentation process will be described in the next sub-section 5.2.2.

- **Identification (16 bits)**: this field is used along with the source, destination address, and the protocol field to uniquely identify a packet. It is used in the fragmentation process to recognize the fragmented packets that originally belong to the same IP packet.

- **Flags (3 bits)**: this field consists of 3 bits but only the last two bits are defined to be used in the fragmentation process. The first bit is not yet defined and hence must be zero. The second bit is specified as **Don't Fragment (DF)** flag. Once it is set to 1, it signifies the routers on its path that this packet must not be fragmented. If a link cannot support the size of the packet, it must be dropped. In this case, an error notification may be issued to the source node to alert about this event. The third bit is defined as **More Fragment (MF)** flag to specify if the packet is the last fragment of an original packet or not. If the fragmentation process is used, this flag will be set on all the fragments except for the last one.

- **Fragment Offset (13 bits)**: this field is used in a fragmentation process to specify the relative order of a fragment in the original packet with the first fragment having an offset of 0. This field is used at the destination to reassemble the fragments together. With only 13 bits in fragment offset field, the minimum size of each fragment can be calculated as $2^{16}/2^{13} = 8$ bytes. In other words, each value increment in this field specifies an 8-byte offset in the original IP packet.

- **Time-to-Live (TTL – 8 bits)**: this field specifies the lifetime of a packet on the Internet, e.g., When a packet is sent to the network, the source node initiates TTL to some value depending on the operating system. When a router receives a packet, it decreases this TTL by one. If the resulting TTL is still larger than 0, the new TTL is written in the packet header and the packet will be forwarded to the next link towards the destination. Otherwise, if the resulting TTL is 0, the packet will be dropped and an error notification will be sent back to the source. Consequently, as the packet travels across the Internet, its TTL is gradually reduced with each passing router and hence, preventing a packet from looping indefinitely on the Internet. As will be discussed latter, this field is also used as a debug tool to find the path from one source node to a destination node.

- **Protocol (8 bits)**: this field indicates which upper-layer protocol the packet should be sent to at the destination node. This is the bridge that connects between the network layer and the transport layer of the TCP/IP architecture. A full list of this protocol number is defined by the Internet Assigned Numbers Authority (IANA) [3]. Table 5.3 shows some common protocol numbers such as TCP (6) and UDP (17).

Table 5.3: Common protocol numbers.

Protocol Number	Protocol Name
1	Internet Control Message Protocol (ICMP)
6	Transmission Control Protocol (TCP)
17	User Datagram Protocol (UDP)
41	IPv6 Encapsulation
89	Open Shortest Path First (OSPF)

- **Header checksum (16 bits)**: this field is used for checking the integrity of an IP packet's header. This field is calculated using only the IP header of the packet according to the Internet checksum algorithm previously described in Chapter 3. At the receiver and each router along the traversed path, this checksum is verified. If errors are detected, the packet will be discarded. From the previous discussions, at least the TTL field of each packet will be altered at each router and hence, the checksum must be re-calculated for each passing packet at each router. This process causes unnecessary burden on routers, which is one of the reasons that header checksum is eliminated in IPv6 as we shall see. It is noted that the payload of the IP packet is not involved in this checksum process and often depends on the error control mechanisms of upper layers.

- **Source and destination addresses (32 bits each)**: these fields contain the identifications of the source and destination nodes of the packet. These addresses are used for the routing of the packet towards the destination.

- **Options (variable length)**: this field is not mandatory in an IP header and is rarely used. Examples of its use are to specify the list of routers that the packet must pass through, or for debugging purpose.

- **Padding (variable length)**: if option fields are used, padding may be needed to ensure that the IP header terminates at a multiple of 32-bit words.

5.2.2 IPv4 fragmentation

As abstraction is one of the mandatory features of a network layer, IP is used as a unified layer that buries the differences at data link and physical layers. However, when being transferred from one link to another, if the maximum supportable payload size in a frame of the outgoing link, called the **maximum transmission unit (MTU)**, is smaller than that of the incoming link, the transit router must perform fragmentation in order to fit the incoming frame to the outgoing link. This process partitions the received packet into smaller fragments, each fits the MTU of the outgoing link. However, this fragmentation process must be done in such a way that destination node can identify these segments and put them in a correct order.

In IPv4, this operation is executed using the *Identification & Fragment offset*, and *MF flag* fields in the IPv4 packet. For any packet that was not fragmented, the Fragment offset must

be 0 and the MF flag must be 0. This combination literally implies that the packet is the first and also the last fragment. If a packet needs to be fragmented, all of its fragments must have the **same Identification number** of the original packet to support the re-assembly of the fragments at the destination node.

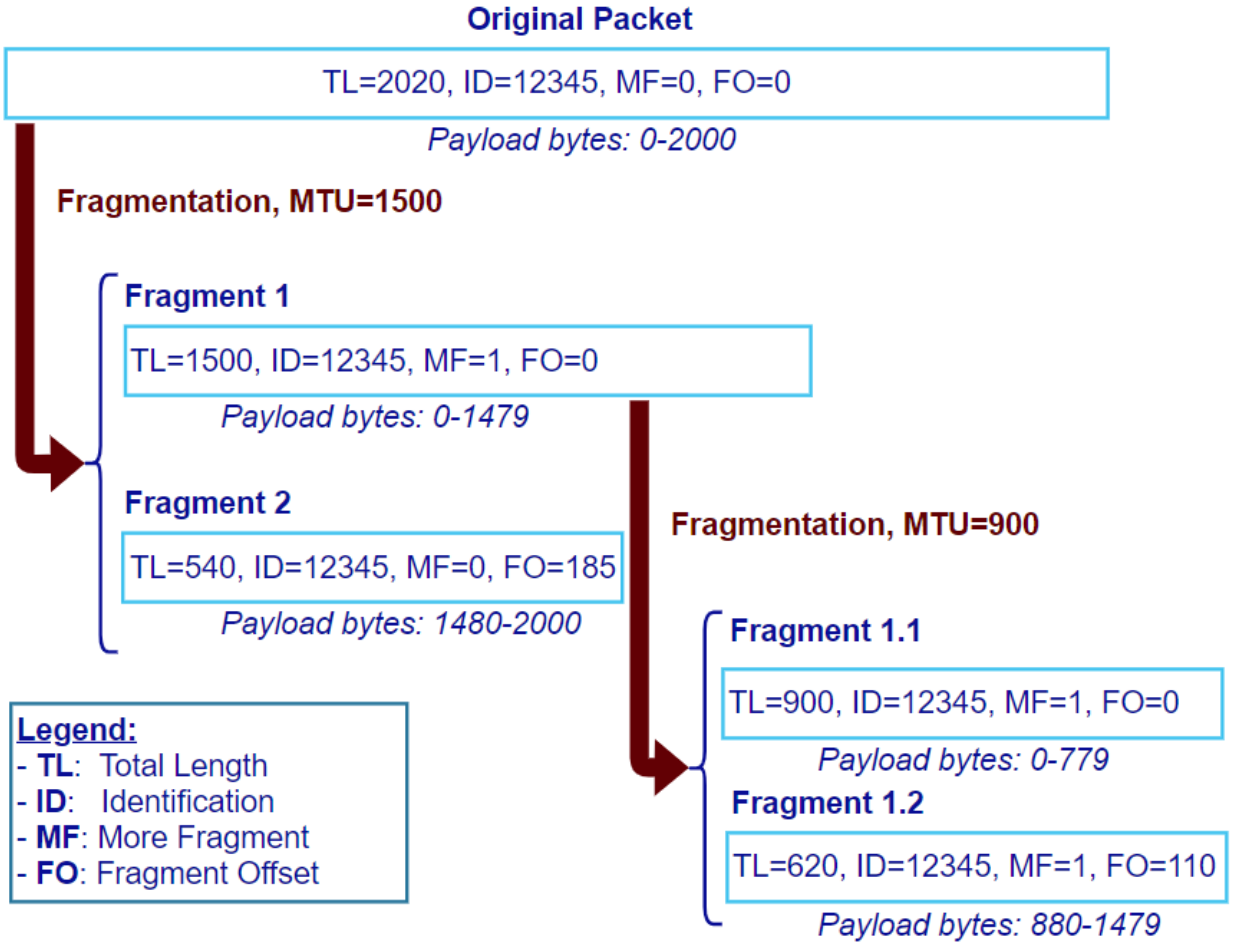


Figure 15: Example of Fragmentation.

At this stage, the destination node still needs to arrange these fragments into the right order so that the original packet can be reconstructed correctly. To make this operation possible, the combination of Fragment offset and MF flag is used. In simple terms, the **Fragment offset** indicates the byte order of the fragment in the original packet. The first fragment packet will have the Fragment offset of 0. As mentioned above, this byte order is interpreted by multiplying the value of this field with 8. For instance, a Fragment offset of 185 indicates that the payload of the fragmented packet starts at the $185 \times 8 = 1480^{th}$ byte in the original packet. Moreover, as the Fragment offset does not tell how big the original packet is, we need an indicator that specifies this. This is where the MF flag comes into play. In this process, the **MF flag** helps the destination node to identify the last segment of the original packet. All fragments except for the last fragment will have MF=1, indicating that they are not the last fragment. The last fragment will have MF=0. In other words, the destination node can stop looking for more fragments of a packet when it receives a packet

with MF=0 and all other fragments that can build up the payload of the original packet up to this last fragment.

Figure 15 illustrates an example of the fragmentation process in IPv4. The original packet has the Total length of 2020 bytes (20 bytes header and 2000 bytes payload). Due to an MTU mismatch on the way, this packet must be partitioned into two fragments of maximum 1500 bytes each. Both fragments must possess the same Identification field as indicated in the original packet. The first segment spans the maximum 1500 bytes and carries the first 1480 bytes of payload (the 20-byte difference is due to the header overhead of this packet). Since there is another fragment follows fragment 1, its MF flag is 1 and its Fragment offset is 0 (it is the first segment of the original packet). The second fragment carries the remaining bytes from the 1480th byte to the last byte (2000). Its Total length can then be calculated as 520+20(header)=540. Since this is the last fragment, its MF flag is 0, indicating that there is no further fragment. The Fragment offset of this second fragment can be calculated as 1480/8=185.

It is important to note that this fragmentation process may take place more than once along the traverse path and this fragmentation process allows packet fragments to be further fragmented while still support the consistency in reconstruction of the original packet at the destination. Figure 15 depicts this process. In this figure, the first fragment must be further fragmented to fit into a link with an MTU of 900 bytes. In this case, the same procedure can be applied for this fragment to yield its sub-fragments. The detailed calculations are shown in Figure 15 and it is left as a small assignment for interested readers to verify the values in the related fields of these sub-fragments.

5.2.3 IPv4 addressing

As previously discussed, being able to uniquely identify a communication device in an internetwork environment is crucial for the operation of the Internet. A host connects to a network using a network adapter, or an interface. Since a device can possess multiple interfaces at the same time, it is important to understand that *a network-layer address is assigned to each active interface of the device and is the identity of this device on the according network*. A prime example is a router with many interfaces, each may connect to a separated transmission link. Each of these interfaces will be assigned with a network-layer address to make it ready for communications. As a result, a router may possess multiple network-layer addresses; each is assigned to one of its network interfaces and is the identity of the router when communicating on that network. Similarly, even an average laptop today may have multiple networking interfaces such as one for wired Ethernet and another for WiFi. This is analogous to a big house at a corner of an intersection with two addresses, each is associated with a street it has a door open to. However, to simplify the discussion in the following, a host is assumed to have only one interface so that the IP address of the host will be the IP address of this connected interface.

An IPv4 address consists of 32 bits in length and is normally presented in a human readable format of 4 decimal numbers separated from each other by a dot (“.”), naming the **dotted-decimal notation**. Each of these numbers is the decimal representation of 8 bits in the IPv4 address. For example, a valid IPv4 address is 192.168.1.10. Here, the last decimal “10” indicates that the binary representation of the last 8 bits of the above IP address is “00001010”.

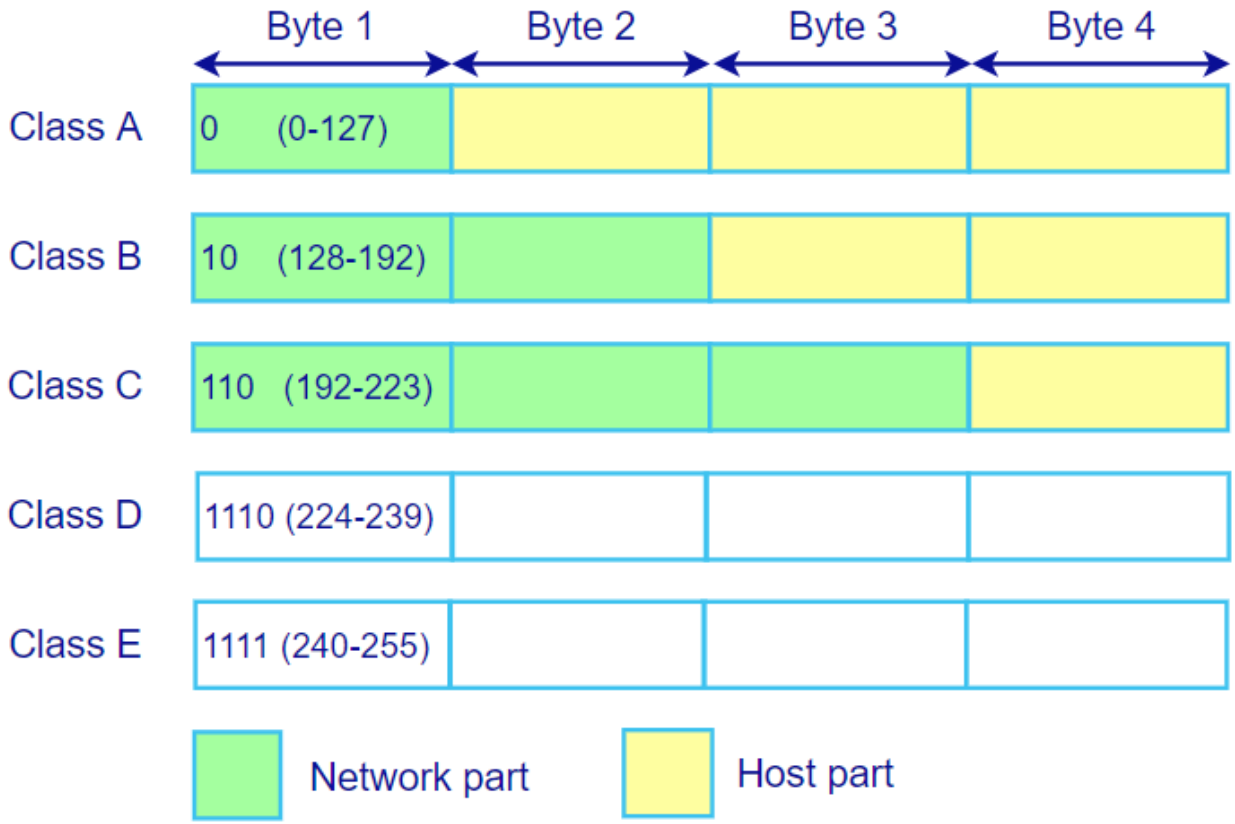


Figure 16: Addressing structures of the five classes of IPv4.

As mentioned in the first section, in order for an address to be routable, or to be usable on an internetwork environment, it must consist of at least a network and a host identification. The host identification is used to recognize the individual hosts that connects to the same network while the network identification is used to distinguish among multiple networks in the Internet. Accordingly, IPv4 addresses also uses this hierarchical addressing scheme with two parts: a network part and a host part of the addresses. The easiest way to separate these two parts is using a fixed number of bits for each part. However, this simple approach is not desirable in practice since this universal scheme does not always fit well in complex scenarios with different demands of host in each network. In the cases where there are networks with many connected hosts, the number of hosts in each network may not be enough while in other networks with only a few connected hosts, this number may be too abundant and wasteful.

To strike a balance between these cases, the initial IPv4 addressing space is divided into five classes, namely class A, B, C, D and E. Among these five, only class A, B and C networks are used to assign to hosts and these three classes are different from each other in the way the network and host parts are separated.

The three classes are distinguished from each other by the value in the first byte of the address. A class A address has the first bit in the first byte of its address as 0. As a result, the first byte of a class A address is in the range 0-127 (in decimal). Similarly, the first byte in a class B address is in the decimal range of 128-191 (starts with 10 binary). And the first byte in a class C address is in the decimal range of 192-223 (starts with 110 binary). For a class A

address, the first byte is used for the network part and all the last three bytes compose the host part of the address. As such, class A addresses are intended to be used in networks with a huge number of hosts. Similarly, in a class B address, the byte composition of the network and host part is 2-2 and is intended to be used in networks with a medium number of hosts. Finally, the byte composition in a class C address is 3-1, intending to be used in networks with a small number of hosts. This network-host separation scheme is normally referred as classful network addressing and is illustrated in Figure 16.

Class D and E addresses are not separated into network and host parts due to their special uses. Class D addresses (having first bytes in the decimal range of 224-239) are used for multicast purposes and class E addresses (having first bytes in the decimal range of 240-255) are reserved for experiments and future applications.

It is also noted that not all addresses in classes A, B and C are assignable to be used in the network. Many of them are reserved to be used for special purposes. For example, the range IPv4 address with the first byte as 127 is reserved for loopback testing.

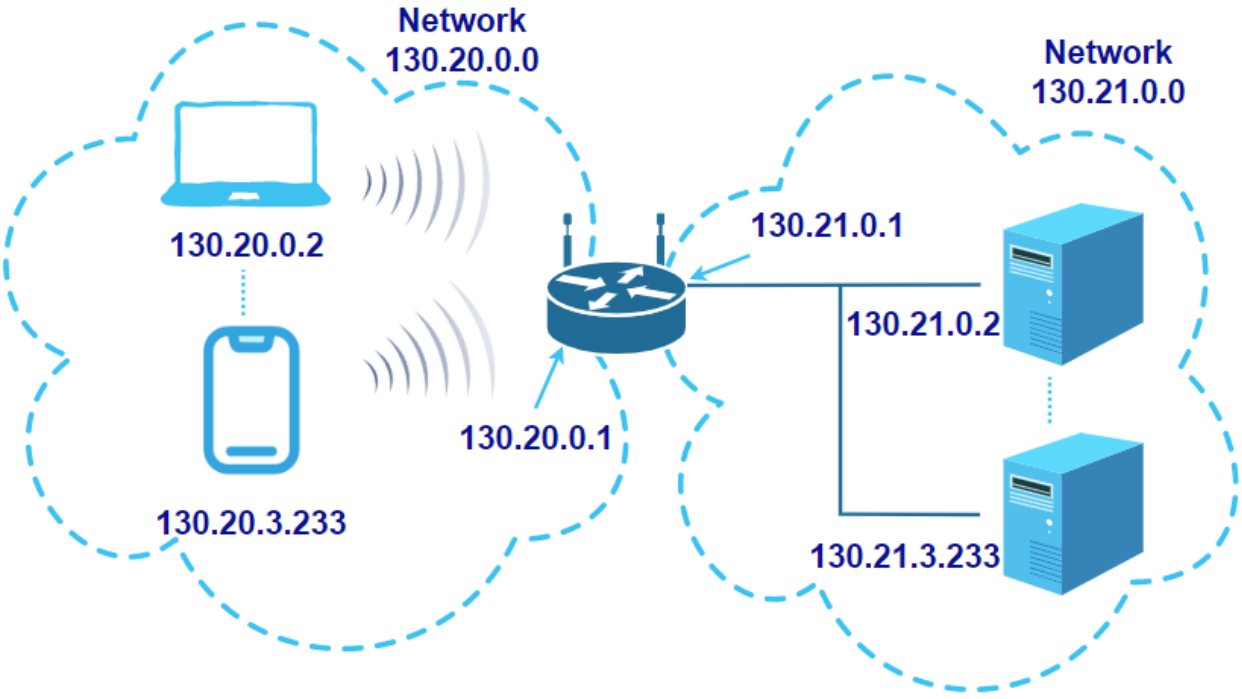


Figure 17: IPv4 address assignment in an internetwork environment.

Also, within a network of a class A, B or C address space, not all possible addresses are assignable to hosts. The address with the host part composes of all "0" bits is reserved for **network identifier** to distinguish between the networks. In addition, the address with the host part composes of all "1" bits is reserved or **broadcast address**, which allows a packet to be delivered to all of the hosts in the network that share the same network part instead of one host at a time. Table 5.4 illustrates an example of a class B network where its network identifier, broadcast address and its assignable host range are presented.

Figure 17 illustrates the IP assignments to end devices and router interfaces in a simple internetwork environment of two networks. The network on the left consists of 1,000 hosts which are assigned with the IPv4 addresses from a class B network 130.20.0.0 (as described

in Table 5.4) while the 1,000 servers on the right is assigned with an IPv4 address from another class B network 130.21.0.0. It is noted here that in order to attach the router to each network, its attached interfaces must be assigned with addresses which is in the same network as it connected to (normally the router interface is assigned with the first usable IP address in the network). In this figure, it is an exercise for the reader to verify that there are actually 1,000 addresses in the address range specified by the first and the last communication nodes in each network within this figure.

Table 5.4: An example of a Class B network addressing.

Class B network with network part of 130.20	
Network identifier	130.20.0.0
Broadcast address	130.20.255.255
Assignable host range	130.20.0.1-130.20.255.254

Subnet addressing

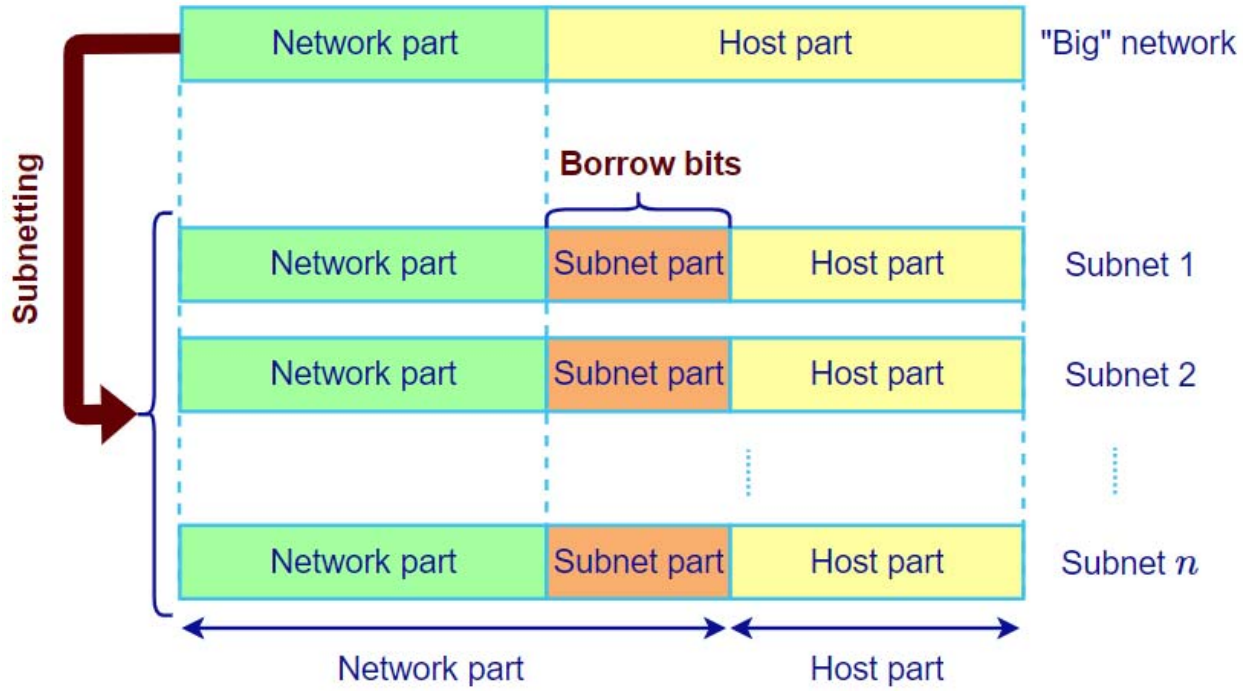


Figure 18: Principle of subnet addressing.

In spite of its separation into classes with different sizes, the IPv4 classful addressing scheme still does not provide enough flexibility in allocating address spaces to networks of all sizes. For example, Figure 17 illustrates a scenario when two networks of 1,000 hosts each must be allocated with IP addresses. A class B network supports $2^{16} - 2 = 65{,}534$ addresses, hence, assigning one class B network for 1000 hosts as in Figure 17 would lead to an address utilization of only $1000/65{,}534 \approx 1.5\%$ of the usable addresses, wasting about 64,000 addresses. In addition, while it is true that for many enterprise networks, the number of connected hosts is quite large, it is normally undesirable to arrange all hosts to the same network due to management and security reasons. In this case, the best practice is to group

connected hosts into multiple groups depending on their functionalities and requirements for example, and then arrange each of this group into a separate network.

In either case, it is inefficient to use a large address space as is. This motivates the development of **subnet addressing**. The fundamental principle behind subnetting is to divide a big classful network into smaller partitions, namely **subnets**, to meet the addressing demand by borrowing a number of bits from the host part of a network address as illustrated in Figure 18. The borrow bits is called the **subnet part** and it **is appended with the original network part to make up the network part of the subnet address**. To make the network part contiguous, the subnet part must be **borrowed from the left most bits of the host part**. As depicted in Figure 18, the borrowed bits effectively divide the original network into many smaller subnets, each with a smaller number of supportable hosts than the original one.

Subnet mask and prefix length

At this point, it is clear that with the presence of this subnet part, the traditional way of separation network part and host part in classful addressing scheme is no longer valid. The network part of the resulting subnet addresses may not end well with the byte boundaries. Recall that routers forward packets based on their forwarding tables with prefixes based on these network parts and without a clear separation between network and host parts of an address, routers cannot forward a packet correctly to its destination. To solve this issue of distinguishing between the network and the host parts, subnet mask was introduced to accompany an IP address. A **subnet mask** consists of 32 bits as in an IP address; however, the bits in the network part of the subnet (including the subnet part) will be presented in the subnet mask as all bits with binary values of "1" while the bits in the host part of the subnet mask will be presented with binary values of "0". In this way, by using bit-wise AND operation between an address and its subnet mask, the network address of the subnet can be easily revealed. Another equivalent representation of the subnet mask, and perhaps shorter, is prefix length. A **prefix length** accompanies an IP address to specify the number of bits in the address is used for its network part. It is normally presented after the address in the form of "/X", where X is the number of network bits in the address.

For example, a Class B network address 130.20.0.2 has its first 16 bits as network part, hence will have its prefix length as /16, and its subnet mask as 255.255.0.0. This address can be presented with its prefix as 130.20.0.2/16. Table 5.5 illustrates an example of a class B IPv4 address with its prefix length, subnet mask and the AND operation between this IPv4 address and its subnet mask to yield the network address of the network.

Table 5.5: Example of subnet mask and prefix length.

Class B IPv4 address 130.20.0.2/16					Dotted Decimal
IP address	10000010	00010100	00000000	00000010	130.20.0.2
Subnet mask	11111111	11111111	00000000	00000000	255.255.0.0
AND operation					
Network address	10000010	00010100	00000000	00000000	130.20.0.0

Subnetting example

To illustrate the subnetting procedure, it is best to work through an example. Consider the network demand as in Figure 17 with two subnets of 1,000 hosts each. In this case,

instead of two we illustrate that by using only one class B address of 130.20.0.0/16 and appropriate subnet addressing, we can easily support the given demand.

Table 5.6: Network address calculation for the subnets in the example above.

Subnet index	Binary representation of the last two bytes from the host portion		Network address of the subnet
0	**000000** 00	00000000	130.20.0.0/22
1	**000001** 00	00000000	130.20.4.0/22
		...	
63	**111111** 00	00000000	130.20.252.0/22

To meet a demand of 1,000 hosts for each subnet, it requires a 10 bit space for the host part ($2^{10} = 1024 - 2 = 1022 \geq 1000$, the "-2" results from the unusable network and broadcast addresses for each subnet). Consequently, from the 16 bits for the host part, we are left with $16 - 10 = 6$ bits for subnet part. Consequently, the resulting addresses for the subnets would have the prefix length of /22 (16+6=22), or a subnet mask of 255.255.252.0. By changing the subnet part while keeping the host part as "0", the network addresses of all the subnets can be easily determined as in Table 5.6. It is noted in this table that the original class B address space already has its network part occupies the first two bytes, hence, only the last two bytes are presented. Also, within these two bytes, the subnet part is presented in bold blue font for the ease of interpretation.

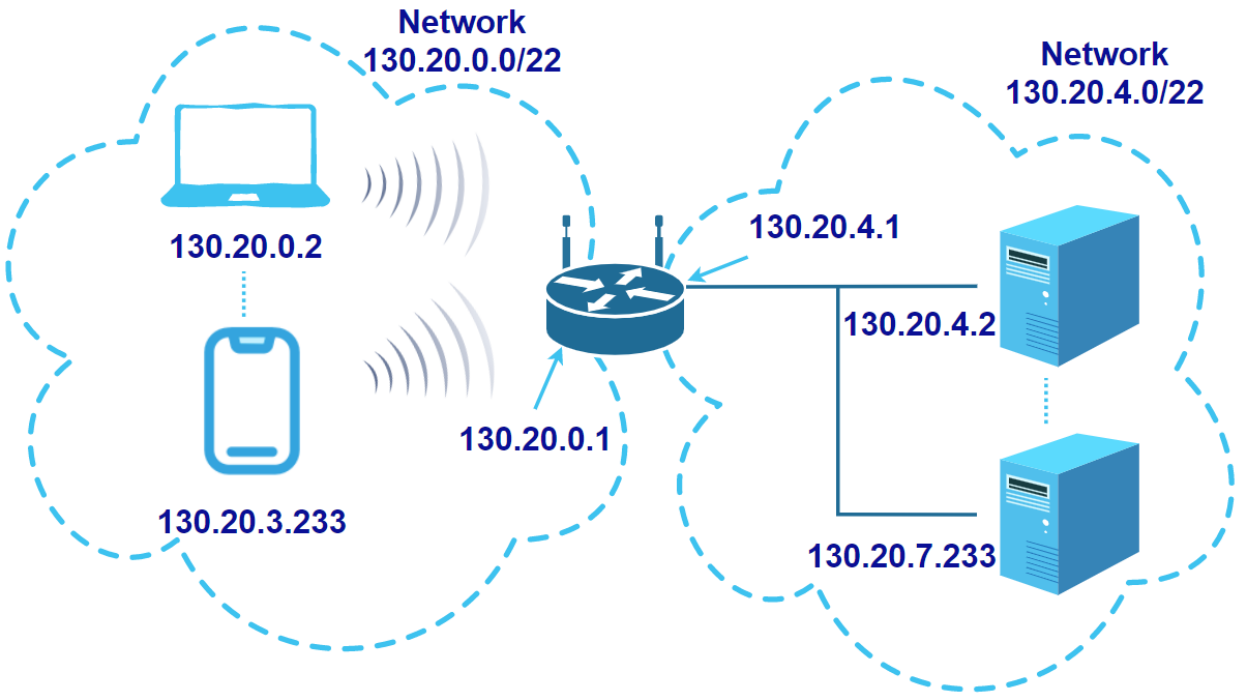


Figure 19: Network addressing with subnets.

Then, by changing only the host part within each subnet, the assignable host range can be determined as in Table 5.7. It is noted that the last address of the host range, which contains all bits "1" in its host part is reserved as the broadcast address for that subnet and hence, not usable.

Using the subnet addressing in Table 5.7, we can replace the wasteful classful addresses assignment in Figure 17 with the more efficient subnet addressing as illustrated in Figure 19. It can be seen that by using the first two subnets, we can easily meet the stated demand with very little wastage of addresses.

It is noted that this subnetting procedure can be done many times in a recursive manner, i.e. one unused subnet from Table 5.7 can be further divided if in the future there is a demand for another network but with a less number of hosts. These recursive divisions lead to a scenario when multiple subnets coexist in the same subnet with different prefix lengths and subnet masks. This ability to use multiple subnet masks and prefix lengths from the same original addressing space in the same network is introduced in the **Variable Length Subnet Mask (VLSM)** standard.

Table 5.7: Subnetting calculation of the network address, assignable host range and broadcast address of each subnets.

Subnet index	Network address of the subnet	Assignable host range	Broadcast address of the subnet
0	130.20.0.0/22	130.20.0.1- 130.20.3.254/22	130.20.3.255/22
1	130.20.4.0/22	130.20.4.1- 130.20.7.254/22	130.20.7.255/22
		…	
63	130.20.252.0/22	130.20.252.1- 130.20.255.254/22	130.20.255.255/22

Classless Inter-Domain Routing (CIDR)

Table 5.8: An example of route aggregation in IPv4

	Binary Presentation	Dotted Decimal presentation
Subnet 1	**10000010 00010100 00000**000 00000000	130.20.0.0/22
Subnet 2	**10000010 00010100 00001**00 00000000	130.20.4.0/22
Aggregated Prefix	**10000010 00010100 00000**000 00000000	130.20.0.0/21

As discussed above, the use of classful addressing is very inefficient and while subnetting and VLSM solves part of the problem, it introduces another issue. With the use of recursive subnets, the number of prefixes a router must carry in its forwarding table keeps increasing. This issue does not only increase the lookup time to find the outgoing interface for a packet but also puts a lot of stresses in the memory of backbone routers.

To resolve this issue, **Classless Inter-Domain Routing (CIDR)** was introduced. On one hand, CIDR incorporates the mechanisms of VLSM. On the other hand, CIDR proposes two more important amendments. First, it lifts up the classful network-host boundary restriction and allows assignments of arbitrary prefix lengths (classless). For example, an addressing scheme based on a CIDR block of 201.21.16.0/20 is possible (in a classful scheme, "201" must be a class C network and have a /24 prefix). Of course, once allocated with a CIDR block of addresses, the according administrator can utilize subnetting to divide this IP address range to fit his particular needs.

Second, to resolve the increases number of prefixes in routers' forwarding table, CIDR allows the aggregation of multiple network prefixes into one single prefix (**supernet**) in a router's forwarding table. For instance, eight contiguous prefixes of /24 can be aggregated into a single entry in router's forwarding table with a prefix of /21. To find the aggregated prefix, one must keep all the common bits in the original prefixes and zero out all the different bits. For example, the two /22 prefixes used in Figure 19 can be represented by a single prefix of /21 in an upstream router as in Table 5.8.

Figure 20 illustrates the effect of CIDR route aggregation on a router forwarding table. Assuming that the router R1 in Figure 19 is connected to an upstream router R2 (for simplification purpose, we omit the addressing scheme on the link between R1 and R2). In the forwarding table of R1, there are three entries, two for the two subnets previously mentioned and one for a default route to the Internet (using the same aggregation technique above, it can be verified that a prefix of 0.0.0.0/0 represents the entire address space in the Internet). On R2, the upstream router of R1, it is not needed to retain the detail information of both subnets connected to R1 since the only way to approach either of these two is by forwarding the packet to R1. Instead, it is beneficial to aggregate these two prefixes into a single prefix and point it to the interface connected with R1.

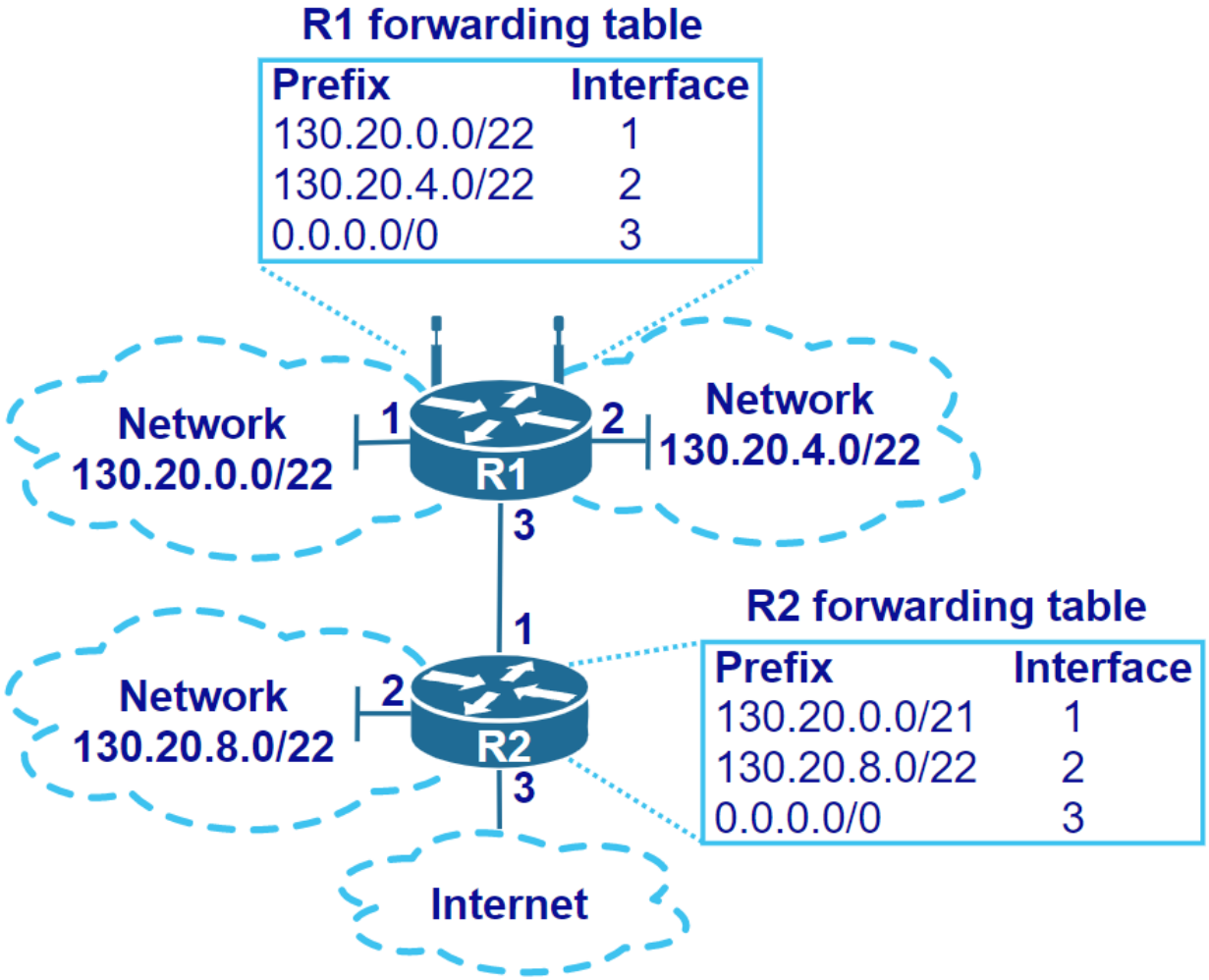


Figure 20: Router forwarding table with route aggregation.

Although already mentioned in section 5.1.3, it is worth mentioning here again that in case of multiple prefixes match the destination address of the packet, the appropriate action is to forward the packet according to the prefix with the **longest prefix match**. As illustrated in Figure 20, in the forwarding table of R2, there are two entries to the prefix starts with "130.20". If a R2 receives a packet destined for an address of 130.20.8.100, this destination address matches both prefixes. However, the second prefix matches the first 22 bits of this address while the first prefix matches only the first 20 bits, hence, the packet should be forwarded to interface 2 of R2 according to entry 2 in R2's forwarding table. On the other hand, if there is a packet destined for a destination address of 130.20.4.100 arrives at R2, this address matches the first 21 bits of the first entry and first 20 bits in the second entry of R2's forwarding table. As a result, this packet should be forwarded according to the first entry in R2's forwarding table following the longest prefix match. This action transfers the packet to R1, which is indeed the next router in the path to the packet's destination.

Through these examples, it is demonstrated that the combination of classless addressing, subnetting and route aggregation enable an efficient allocation of IP addresses to meet a wide range of demands while still maintain a compact routing table at the routers in the network. It is also apparent that routers will maintain a more detailed view of the networks in its vicinity while having a more abstract view of the networks that are far away.

Network Address Translation (NAT)

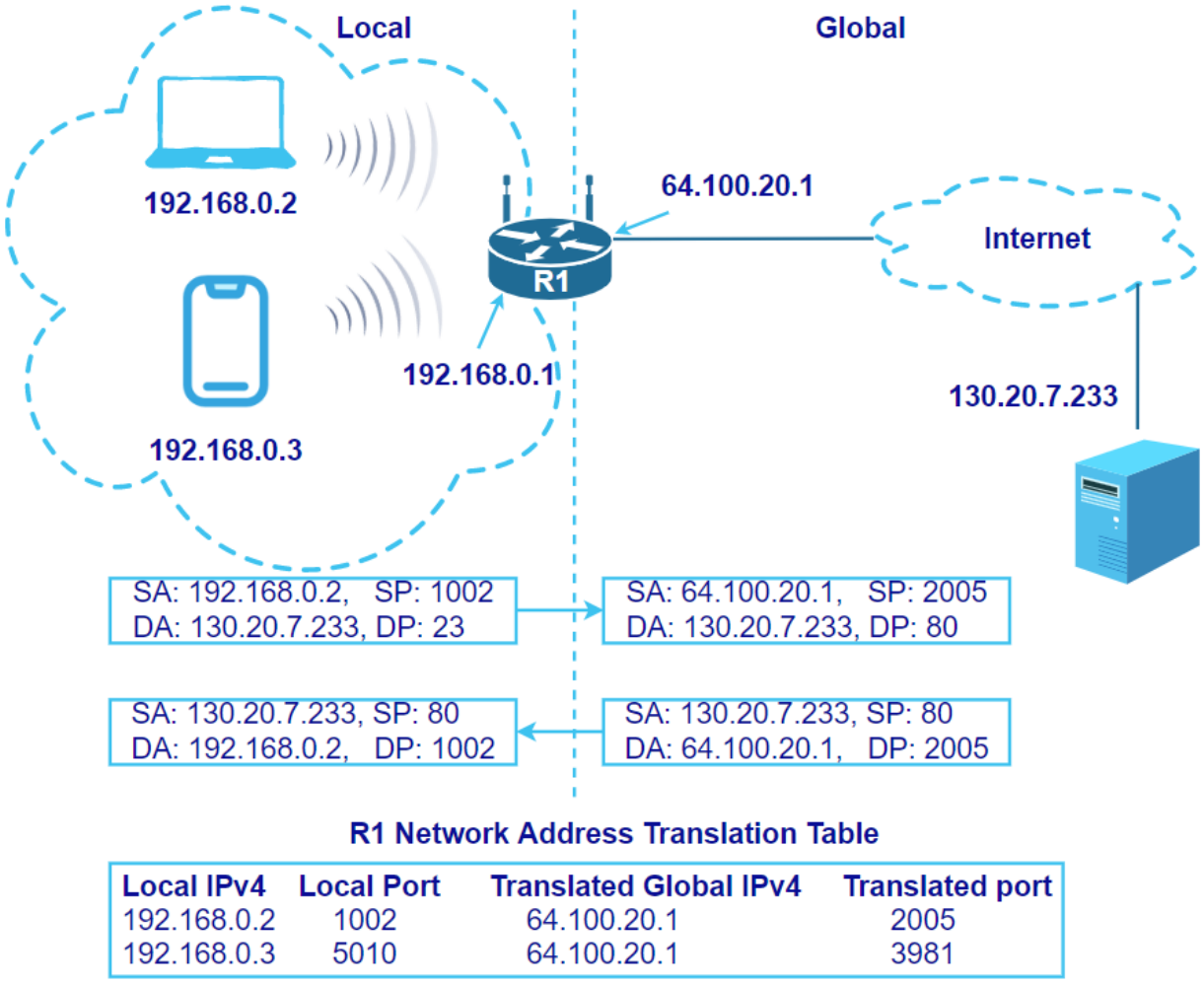
With the available addressing space of 32 bits, IPv4 can support a maximum of $2^{32} \approx 4$ billions of addresses. However, this number is hardly enough for the increasing demands for connections. As an example, as for April 2019, the world population is approximately 7.7 billion, nearly double the maximum supportable addresses of IPv4. In addition, one person may possess more than one connected device, such as a few laptops, PCs, smartphones and tablets at a time. Moreover, with the ongoing deployment of the Internet-of-Things, the addressing demand will only increase in the future. As such, it is clear that the available IPv4 addresses are inadequate for the requirement of "globally uniquely identifier" as they were designed. Actually, this address depletion issue is not new and was in fact acknowledged as soon as the 1980s. Despites the introduction of many techniques such as CIDR, this exhaustion issue of IPv4 addressing space requires more effective solutions.

To this end, two approaches were proposed, and they are both being used today. The first approach is to introduce a new scheme with a larger address space. This is the approach of IPv6. With 128 bits in its address space, IPv6 promises to be a long-term solution which is able to withstand the addressing need for a good number of years to come. However, IPv6 is not only an expansion in IPv4 addressing scheme but it brings more thorough changes and will be discussed latter. The second approach, namely **Network Address Translation (NAT)**, is a short-term solution of still using IPv4 but dividing the IPv4 address space into two sub-spaces: global address and private address spaces. The global address space has a global significant, which usage must be registered and paid for, and can be used for routing of packets across the Internet. The private address space has only a local significant and can be assigned freely by network administrators, but they are not valid for routing in the Internet. Then, a translation mechanism is defined to allow the representation of a large number of local addresses by one or a few global addresses.

In NAT, there are three local address spaces, which is illustrated in Table 5.9. These IP address ranges are free to use and can be assigned to a local network at will. If you are using a device that connects to your home network, there is a high chance that your device is using a private IPv4 address (you can verify the assigned address by using a terminal command **ipconfig** in windows or **ifconfig** in Linux).

Table 5.9: IPv4 private address space.

CIDR block	IP address range
10.0.0.0/8	10.0.0.0-10.255.255.255
172.16.0.0/12	172.16.0.0-172.31.255.255
192.168.0.0/16	192.168.0.0-192.168.255.255



R1 Network Address Translation Table

Local IPv4	Local Port	Translated Global IPv4	Translated port
192.168.0.2	1002	64.100.20.1	2005
192.168.0.3	5010	64.100.20.1	3981

Figure 21: Operations of NAT.

NAT operates on routers by temporary mapping the private IP addresses of users to some global address pool (NAT pool). In this process, the source IP address and possibly the source transport-level port number of the passing packets will be changed by the NAT router to an address in this NAT pool before forwarding to the Internet. The details of this IP and port changes will be recorded in some temporary memory space within this router. The

record of this mapping is useful when this router receives a packet destined for an address in its NAT pool from the Internet. In this case, it will find and use the appropriate mapping by means of the source, destination IP address and port and re-writes the destination IP address and port with the proper private IP address and port number. In the simplest case, such as in a home network, the NAT pool consist of only one address, usually the interface address of the router that connects to the ISP. In this case, all the devices inside the house are assigned with private IP addresses and are all presented by the same global IP address in the NAT pool.

Figure 21 illustrates the operation of NAT. The network topology depicts a traditional home network where many devices in the house connects to a router, possibly via WiFi, to gain access to the Internet. The network topology can be divided into two parts, the local part includes the router interface that connects with the home devices, and the global part includes the interface of the router that connects to the ISP and the rest of the Internet.

Upon receiving a packet from the local network destined for the Internet, the home router creates an entry in its NAT table that maps the source private IP (192.168.0.2) and the source port (1002) to the public IP it connects to the ISP (64.100.20.1) and an arbitrary port (2005) that is not yet used in the NAT table. The source address and port number of the packet will be re-written with these new values before forwarding to the Internet. When the router receives a packet from the Internet destined for its NAT address, it looks into the NAT table and search for the entry with the matching translated global IP address and port. If found, the router will replace the destination address and port with the appropriate local IP address and port in the table before forwarding the packet to the local network. In this example, other hosts in the same local network can also send and receive packets to and from the Internet simultaneously. In this case, the router creates multiple entries in the NAT table, each for a session going through it. However, in order to make it possible to translate the incoming packets from the Internet to the correct local address, it is required that the translated port for each session must be different from each other. As we shall see in the next chapter, the transport layer port numbering is defined using 16 bits, hence, by using NAT, each global address can effectively represent about 65,000 simultaneous sessions at the same time. While being not a tremendously large number, this figure should be sufficient for the demand of a family of a few users such as a home network in Figure 21.

In the example above, from the perspective of a server on the Internet, it receives the packets from the NAT router or the addresses in the NAT pool and not the source hosts. In other words, the translated global NAT addresses represents the source hosts in the Internet. However, being able to use a limited number of translated global NAT addresses to represent a much larger number of private addresses makes it possible to support the hosts with private addresses to communicate over the Internet while greatly saving the number of global addresses.

It is noted that the address translation and replacing operations are done only at the border router between the local and global domain while the end nodes are not aware of it. In other words, NAT is transparent to the end nodes, making it possible for local hosts to be plugged and play without requiring any further configurations. Furthermore, NAT is considered by some as a mean to protect the local network from attacks from the Internet. Because the global Internet does not know the local addresses of the local network, and a

connection from a global Internet to a local host is not possible without first establish one from the local network, it is harder for an intruder to attack hosts that sit behind NAT.

However, NAT is not without its disadvantages. NAT requires further configurations on the router connected between the two domains, making it more complex for network deployment. NAT also increases the overhead that imposes on the border router. Moreover, the use of NAT breaks the end-to-end IP traceability as the original IP addresses and ports are not preserved. It makes it hard for some protocols that requires also this information for their operations. Besides, the fact that NAT does not allow incoming traffic from the global Internet without first an established connection from the local network prohibits the usability of some applications that require connection initiation from the outside. Last but not least, the use of NAT, in a way, breaks the consistency of the TCP/IP layering structure. In NAT, both the information at network and transport layer need to be manipulated at the NAT router, being a network layer device.

5.2.4 Dynamic Host Configuration Protocol (DHCP)

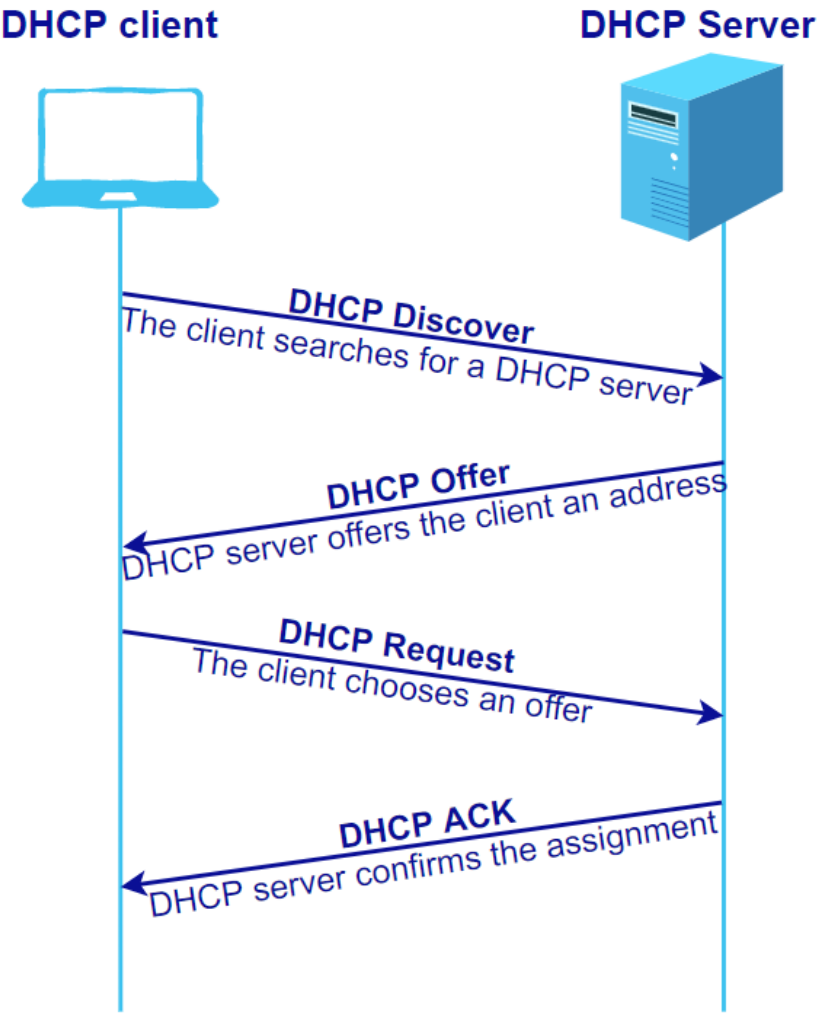


Figure 22: Assignment of an IP address using DHCP.

Once a host participates in a subnet, it should be assigned with an address that is compatible with the addressing scheme on that subnet. To achieve this goal, each

participated host can be manually configured with an address within an assigned subnet by the network administrator or by the user. However, this is a very tedious process if the user device needs to move between multiple networks from time to time. This manual process is also very error-prone as one typo could lead to address conflictions in the subnet, and interfere with the operations of other hosts in the network. As a result, **Dynamic Host Configuration Protocol (DHCP)** were developed as a method to ease the IP assignment process to newly joined hosts by automatic configurations. DHCP operates transparently to the user and hence, realizing a plug-and-play approach to IP addressing configurations. As such, DHCP is widely adopted in almost all types of networks from a home network, an internet café, a campus network to enterprise and ISP networks.

DHCP operates in a client-server manner where there is at least one always on DHCP server on the network (in a small network such as a home network, this task is usually integrated with the connected router). This DHCP server will be in charge of IP assignments in the appropriate subnet. The DHCP server is assigned with an IP address pool that can be used for address assignment on the subnet, which it manages. Normally, each IP assignment is associated with a lease time. When this lease time expires, the host must renew its IP address, and this released IP address will be returned to the pool for reuse. In addition to assigning IP addresses, DHCP server can also provide its serving hosts with other information such as the subnet mask, the gateway address, the domain name servers, time server, boot files (if the host boots from network), etc.

Figure 22 illustrates a DHCP session where a newly arrived host requests and is assigned an IP address through DHCP. This process includes four steps. First, the newly arrived host broadcast a **DHCP Discover** message to the network to find a DHCP server on the network. The destination address of this packet is a broadcast address (255.255.255.255) so that the message can be delivered to all of the connected devices on the network. If there is a DHCP server on the network, upon receiving this message, it will allocate an available IP address for the client and reply it with a **DHCP Offer** message. This message contains the offered IP address, subnet mask, the lease time (valid duration) of the offered IP address and the IP address of the DHCP server. Since there could be multiple DHCP servers on the same network, the DHCP client has to choose one among the offers it received by replying to it with a **DHCP Request** message. This message repeats the assigned configurations of the DHCP offer message along with the server IP address. When the other servers received this message, by using the server IP address, they know that their offers were not accepted, hence, they will release all the offers they sent to the client, including the offered IP addresses. When the DHCP server with the selected offer receives this DHCP request, it acknowledges its allocation by sending a **DHCP ACK** message. Upon receiving this message, the DHCP client can start using its newly allocated IP address for communication. It is noted that similar to DHCP Discover, DHCP Offer, Request and ACK messages are sent using a broadcast destination address so that the un-configured client can receive them.

In case there is no DHCP server on a network, DHCP address assignment can be also possible by using a relay approach. In this case, a DHCP relay agent (usually the network router) can receive the DHCP messages from the client and relay it to a DHCP server located on a different network. The responses from the DHCP server are also relayed in the reversed direction using the relay agent.

5.2.5 Internet Control Message Protocol (ICMP)

As discussed in the first section, error reporting is one of the important functions for internetworking. In IP, the protocol that materializes this function is the **Internet Control Message Protocol (ICMP)**. Although ICMP operates at network layer, ICMP messages are encapsulated inside IP packets before sending to the network. As its name indicates, ICMP is not used for transporting user data, but instead is used for error reporting, debugging and control purposes.

ICMP messages are specified based on RFC 792 [4] and includes several message types for several purposes. The ICMP message types are defined based on the first two fields in its header, namely type and code fields. Since it is out of the scope of this book to present a complete description of ICMP messages, in this sub-section, only the most useful and commonly used message types are presented.

Echo request and **echo reply** messages are perhaps the most commonly used by networkers for network troubleshooting. **Ping** program makes uses of these two message types to verify the availability and reachability of a device on a network. A ping program sends a sequence of echo requests to the destination device. Upon receiving these messages, the destination device will issue echo replies for each of the echo requests back to the source device. When the source device receives these replies, it can infer about the availability, reachability, packet loss rate and the round-trip time between the two nodes.

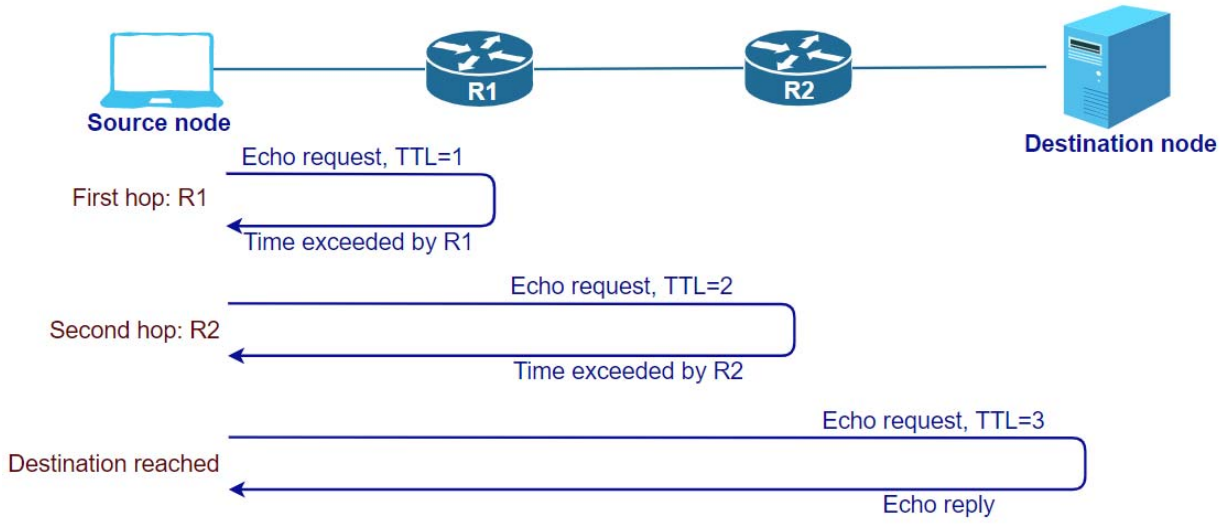


Figure 23: Tracert operations.

Tracert program incorporates the use of echo request, echo reply messages with **time exceeded** message to help finding the path the packets take from the source to the destination node. ICMP time exceeded message is basically an error reporting message issues by routers to the sending node when the TTL of the packet reaches 0. By issuing echo request messages with increasing TTL, tracert program forces the routers along the travelling path to reply with time exceeded messages. These messages are issued with the IP address of the sending router; hence, helping to find the hop-by-hop route towards the destination. Figure 23 illustrates an example of this operation. The source node issues an echo request to the destination node with TTL of 1, when reaching router R1, R1 reduces the TTL of this packet by 1, to 0. Unable to forward the packet to the next link, R1 issue a time

exceeded report with its address. Upon receiving this message, the source node knows the first hop on the path to the destination is R1. It then issues an echo request with TTL of 2 to find the next hop after R1. The process continues until the destination node is reached and an echo reply is issued to the source node. By combining the pieces of information from the time exceeded messages, the source node can reconstruct the full path to the destination node.

Destination unreachable messages are another important type of ICMP messages. A **destination network unreachable** message is issued by a router if it does not have enough routing information to route the packet onto the next link. A **destination host unreachable** message is issued when the router that connected to the destination subnet cannot forward the packet to the destination host (may be due to the unavailability of the host). A **destination port unreachable** message is issued by the destination host if it does not support any service on the request port. For instance, a variation of tracert on Linux/Unix operating system, namely **traceroute**, operates a bit different than tracert. In traceroute, the source device sends UDP packets with a random port, instead of an echo request. In this case, upon receiving these packets, the destination node will return a destination port unreachable, instead of an echo reply.

5.2.6 Address Resolution Protocol (ARP)

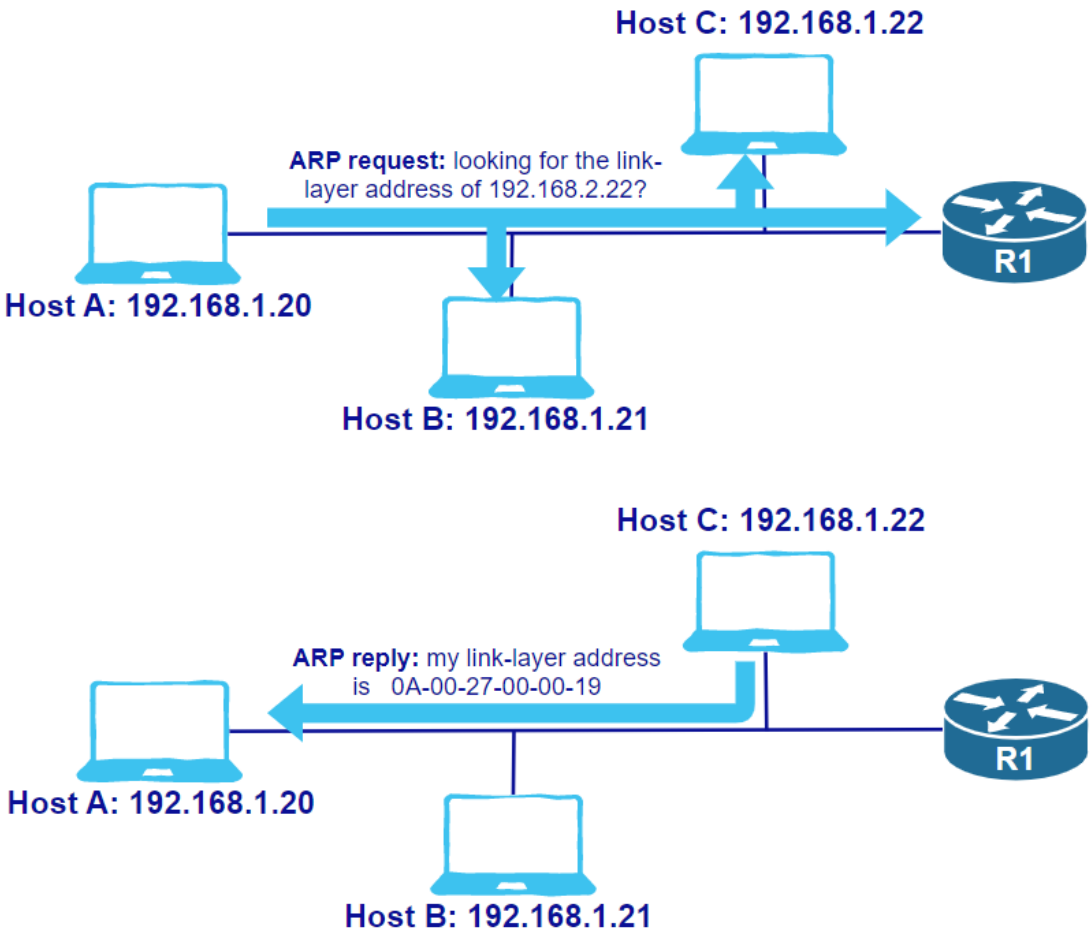


Figure 24: Operations of ARP.

In the discussions so far, we assumed that the involved communication devices somehow know the link-layer address of its destination node on its link. Different from the IP address of a destination node which may be found using mechanisms such as Domain Name System (DNS), link-layer addresses are usually unknown to the senders. Without this important component, the data link layer is unable to encapsulate the frame and the frame cannot be sent at all.

The **Address Resolution Protocol (ARP)** was proposed to resolve this issue, i.e. to find a link-layer address that associates with a given IP address. Figure 24 illustrates the operations of ARP. First, the source node broadcast an ARP request message to all the nodes in the network asking for the link layer address of the IP address indicated in the request. Since the ARP request is broadcasted, all nodes in the network should receive the message; however, only the node that matches the IP address indicated in the request answers to the request with its link-layer address. Knowing the link-layer address of the destination, the source node can now communicate with the destination node. To avoid doing ARP repeatedly every time, the information obtained from the ARP reply is cached temporary at the source node, in an ARP table, for latter use.

The above example works if the two communicating nodes reside within the same subnet. In the case a node wants to communicate with another node in another subnet, it must send the packet to the connected router, or the gateway of the subnet. In this case, using ARP, the source node will ask for the link-layer address of the connected router/gateway.

5.2.7 IPv6

In the previous sub-sections, it was demonstrated that the addressing space supported by IPv4 is not adequate for future growth of the Internet. The short-term solution of using of IPv4 with NAT has several disadvantages that we already studied and discussed. In this sub-section, let's turn to the long-term solution to use a bigger address space, which motivates the development of IPv6. To this end, one may ask what happens with IPv5? Well, IPv5 existed and was developed for the transportation of streaming traffic such as voice and video. However, IPv5 adopts the use of the 32-bit addressing space of IPv4 and hence faces the same issue of address exhaustion of IPv4. In the end, with the development of IPv6, IPv5 was abandoned before it was even standardized. As of 2017, IPv6 was standardized as the latest Internet Standard [5].

The use of a bigger addressing space requires at least an extra space in the source and destination address fields; hence, making IPv6 not compatible with IPv4. Making use of this fact, instead of just expanding the addressing scheme, IPv6 was built from ground up, retaining the goodness of IPv4 while eliminating the inefficient components of IPv4 and providing the necessary enhancement for future use of IPv6. The main features of IPv6 can be summarized as the following:

- **Expanded address space:** the address space in IPv6 is expanded from 32 bits in IPv4 to 128 bits. With this longer address, IPv6 supports approximately 3.4×10^{38} unique addresses, which is unlikely that it will exhaust like IPv4 in any near future. To give an illustration of how large this address space is, it is calculated that this address space is large enough to provide about 10^{23} addresses for each square meter on Earth. Hence, even if somehow, we fill up the Earth surface with connected devices, we must pile up 10^{23} devices

on each square meter to be able to exhaust IPv6 address space, which sounds outrageous at the moment.

In addition, beside multicast and unicast, another type of address was defined, namely anycast, enabling packet delivery from a node to any member in a group of nodes.

- **Simplified header format:** IPv6 uses a fixed length header in which many fields in IPv4 were eliminated to improve the packet handling on routers and decrease the processing overhead and delay of packets transiting a router.

- **Improved and flexible support for options:** instead of a dedicated option field which may not be used very often and may not need to be processed by routers in IPv4, IPv6 facilitates options in **extension headers**. These extension headers are separated from the main header and are linked together by a next header field. This approach allows IPv6 functionalities to be easily and flexibly expanded in the future. In addition, upon reading the next header field, the intermediate routers along the path know immediately whether the next extension headers are needed to be processed or not without looking into the details of these options.

- **Flow identification capability:** in IPv6, a new field, namely **Flow Label** was added to identify packets that belongs to the same flow between any pair of source and destination. This capability promises to provide a mean for a source node to request special treatments for its particular traffic flow.

- **Security capabilities:** another enhancement was built into IPv6 was the capability to support authentication, data integrity and confidentiality. These security functions are added by the use of extension headers and is optional.

IPv6 header structure

Figure 25 illustrates the structure of an IPv6 packet. An IPv6 header is 40 bytes long and consist of only 8 fields as follows:

- **Version (4 bits):** this field was retained from IPv4 and is used to specify the IP version of the packet. For IPv6, it contains a decimal number of 6 (binary 0110), which stands for version 6.

- **Traffic Class (8 bits):** similar to the Type of Traffic field in IPv4, this field is used to differentiate the various traffic classes in the network and can be used as an indication to prioritize certain packets to meet their QoS requirements.

- **Flow Label (20 bits):** this field is newly introduced in IPv6 to help a source node to label a sequence of packets that needs a special handling at the intermediate routers. From the perspective of the intermediate routers, packets belong to a same flow share the same traffic characteristics such as maximum delay or packet loss rate and must be handled in harmony among themselves but differently from other packets. The source node can inform intermediate routers regarding its use of a new flow label by means of signalling. The routers on the way can reserve its communication resources to meet the QoS level required by the source node in handling the flow. Hosts that do not support flow label must set this field as 0. However, to this day, the deployment and use of flow labels on the Internet are still in experimental phase.

- **Payload Length (16 bits):** this field was modified from the Total Length field of IPv4 but since the header of IPv6 packets are fixed now, it indicates the length in bytes of the

payload only. With 16 bits, it supports a maximum packet length of about 64kB. It is noted that IPv6 also provides support for bigger packets (of up to 4TB) as will be discussed latter on.

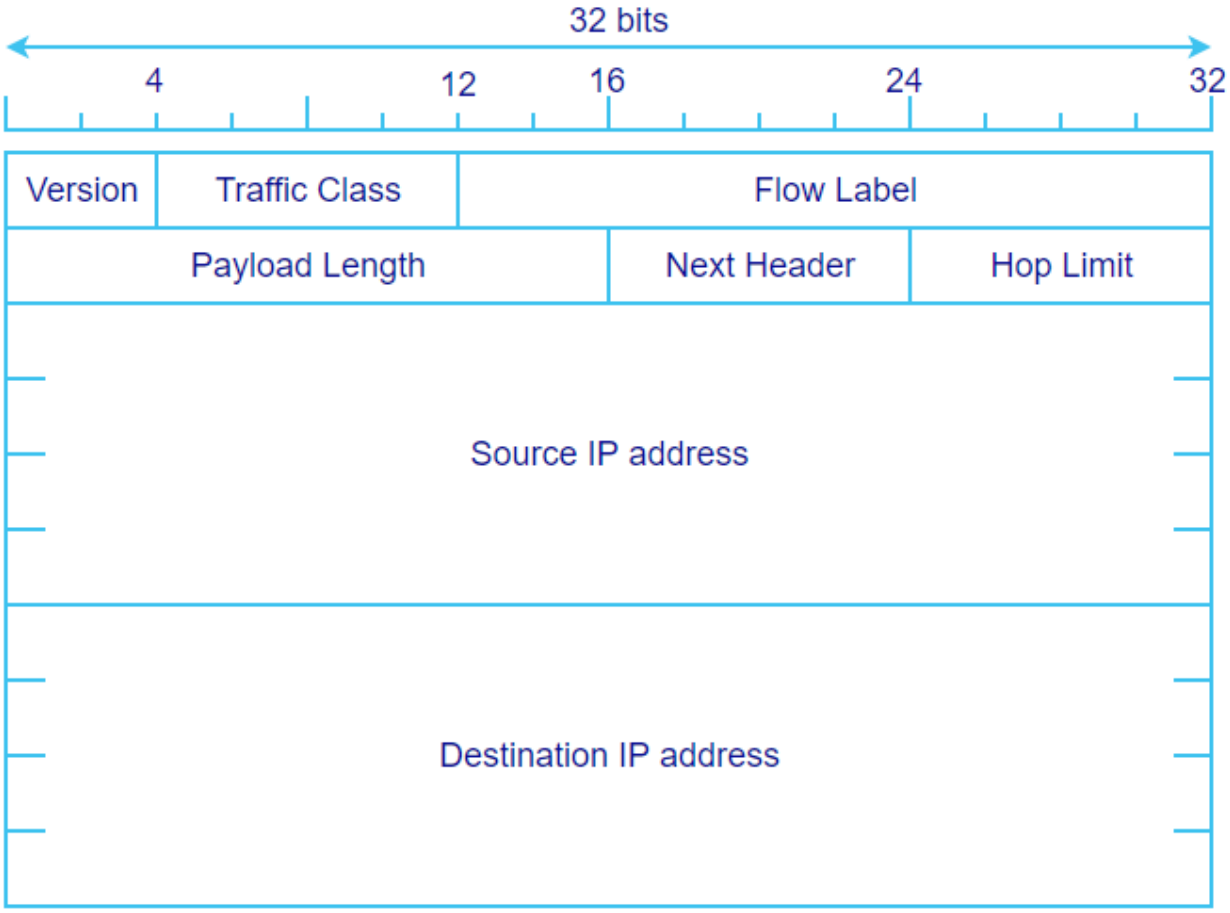


Figure 25: IPv6 header structure.

- **Next header (8 bits):** this field specifies the type of the next extension header that comes after this header of the packet. IPv6 extension headers will be discussed latter in this section. If there is no extension header, this field specifies the transport-layer protocol, such as TCP and UDP that the payload of the packet should be handle over. In the latter case, this field has the same function as the Protocol field in an IPv4 packet.
- **Hop limit (8 bits):** this field has the same functionality as the TTL field in IPv4, i.e. to prevent a packet from circulating forever on the Internet. The value in this field is decreased by 1 by each router on the path to the packet destination. If the Hop Limit of a packet reaches 0 before the packet arrives at its destination, it must be discarded.
- **Source and Destination Address (128 bits each):** these two fields identify the source and destination nodes of the packet and have the same meanings and names as in IPv4. However, different from those of an IPv4 packet, each of these fields contains 128 bits to accommodate an IPv6 address.

In comparison with IPv4, an IPv6 header is about double the size of an IPv4 header, which increases the packet overhead; however, structural wise, an IPv6 packet is simplified greatly by removing several fields from the IPv4 header as the followings:

- **Internet header length:** since the header length of an IPv6 packet is constant at 40 bytes, there is no need to specify the header length in IPv6.
- **Fields related to fragmentation (Identification, Flags and Fragment Offset):** in IPv4, the task of fragmentation at the intermediate routers consumes a lot of processing resources from the router and degrades the packet forwarding performance within the router significantly. Acknowledging this issue, in IPv6, the fragmentation task is removed from the router for good and must be performed in advance by the source host by fragment extension header. In IPv6, if a router receives a larger packet than the MTU of the next link on the path towards the destination, it discards the packet and sends an ICMP error report back to the source node.
- **Header Checksum:** beside fragmentation, header checksum was also a burden to the routers. In fact, since TTL is modified every time a router forwards a packet, the header checksum must be calculated by the intermediate routers on a per-packet basis. In addition, the error detection tasks are already presented in transport and data link layer, making an additional error detection at the network layer unnecessary. Besides, with the introduction of modern transmission media with very low bit error rates such as optical fiber, the probability of bit errors becomes very small. With all of the above observations, it is reasonable to remove and trade-off this error detection functionality at network layer for the packet forwarding performance on the routers.
- **Options:** as discussed in the beginning of this section, IPv6 re-implements the options in IPv4 in extension headers. The elimination of this field makes it possible for IPv6 to have a constant length packet size, which helps to simplify and hence, speed up the packet forwarding in the routers. In addition, the implementation of options in extension headers is a more flexible way to expand the functionalities of IPv6 in the future.

IPv6 packet structure and extension headers

Table 5.10: Types of IPv6 extension headers.

Header Type	Description
Hop-by-Hop Options	Parameters to be examined and processed by all hops along the path.
Routing	Indicates the route the packet should take towards its destination
Fragment	Information of packet fragments and how to reassemble these fragments
Authentication	Uses for checking the authenticity and integrity of the packet
Encapsulating Security Payload	Uses to support data confidentiality, integrity and authentication of the packets.
Destination Options	Information to be examined and processed by the destination of the packet

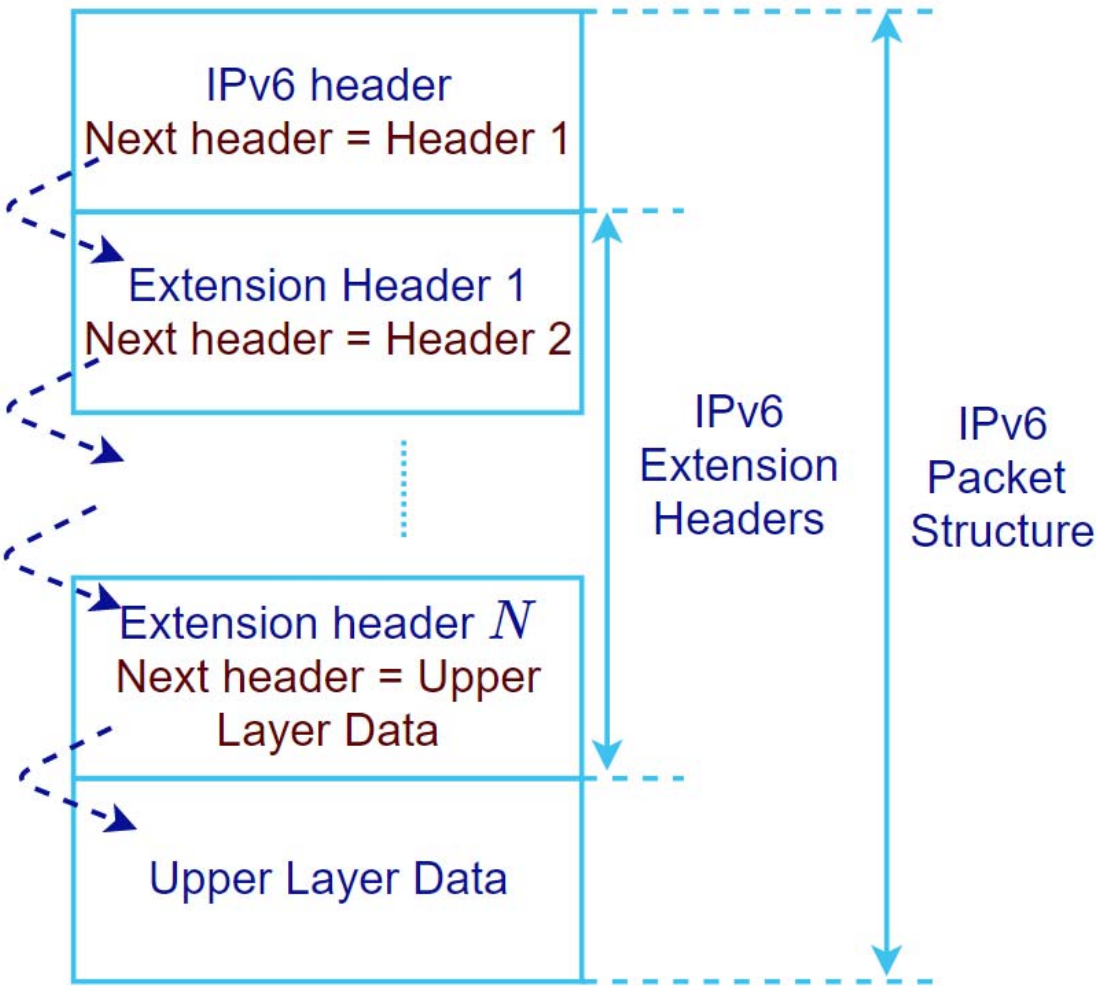


Figure 26: IPv6 packet structure.

Figure 26 illustrates the packet structure of an IPv6 packet, including a basic header as described above, follows by 0 or more extension headers and then the data from the upper layer protocol. The extension headers are used to materialize additional functionalities that are beyond the capability of the basic IPv6 header. For the receiver node or the intermediate routers to comprehend the presence and meaning of the next option in a packet so that they can cooperate and handle the packet accordingly, a next header field is used to link these options together. In particular, a next header field is presented in each of these headers to indicate what will come next. The extension header in the basic IPv6 header reveals what is the next header that follows right after it. This extension header in turn indicates what is the next header that comes next and so on until the upper layer data. In the simplest case where extension headers are not used, the next header in the basic IPv6 header will specify the protocol of the upper layer data.

In IPv6, six of extension headers are defined as summarized in Table 5.10. Since some of the extension headers require the routers on the way to handle the packet accordingly to the information inside the header, the order of these extension headers was carefully designed so as to minimize the processing on these routers. In particular, if multiple headers are present at the same time, they should appear in the order as in Table 5.10 except for the case

of Destination Options which can also appear before the routing header. This order was designed so that those headers that appear earlier should be processed by more devices on the path while those appear closer to the upper layer data should be processed by only the destination node.

Hop-by-Hop Options Header: this header contains information that should be examined and processed by every router along the path of the packet. The most commonly use of this extension header is to allow **Jumbo packets** that can carry up to about 4TB of data. These ultra large packets are used with supercomputers. In that domain, due to the powerful processing capability, transferring data blocks must be large to be able to keep up with the computing power and the maximum of 64KB packet size in the basic header is apparently too small for this application. This extension header uses a 32-bit payload size, which allows packets of up to $2^{32} \approx 4$TB.

Routing Header: this header is used by the source node to list one or more intermediate routers that should be directed to on the path to the packet's destination.

Fragment Header: since fragmentation is not allowed at the intermediate routers in IPv6, if a packet is larger than the MTU of a link on its path, the fragmentation must be done at the source node. In this case, the fragment header is used to provide the necessary information regarding the fragments so that the destination node can resemble them correctly.

Authentication and Encapsulating Security Payload Headers: these extension headers are used to support security in IPv6. Authentication Header is used to ensure the integrity of the packet and guarantee the packet origin. Encapsulating Security Payload Header is used to provide both the authentication and encryption services for the IP packets. These are obtained from the Internet Protocol Security (IPsec) standard which is now an integrated part of IPv6. It is noted that due to their usefulness, these IPsec mechanisms were also adopted as enhancements in IPv4.

Destination Options Header: this header carries auxiliary information about how to handle packet. However, this information needs to be examined and processed by only the destination nodes. One of the commonly use of this type of extension header is to support IPv6 mobility.

IPv6 Addressing

An IPv6 address is 128 bits long, which was discussed above, technically should be enough for connectivity usage in the future. Besides, with its abundant of addressing space, IPv6 enjoys the luxury of being able to trade-off efficient addressing allocation for convenient packet handling and flexible usage. In particular, IPv6 addresses are classified into three types:

- **Unicast address:** uses to identify a single interface in the network. When there is a packet destined for a unicast address, it will be delivered to the interface with this identifier. Unicast addresses are further divided into two main types: link local addresses and global unicast addresses. **Link-local addresses** are used for communication between hosts in the same subnet only and are normally automatically configured by the operating system. This address is used so that connected hosts can still communicate with each other in an ad-hoc manner even if there is no router in the network. Link-local addresses cannot be routed on the Internet, similar to the private addresses in IPv4, and must be filtered by routers. **Global**

Unicast addresses are used for uniquely identifying a host on the Internet, equivalent to public IPv4 addresses.

- **Multicast address:** uses to identify a set of interfaces, typically belonging to different nodes and may resides in different subnets at different places in the network. A packet destined for a multicast address will be delivered to all the interfaces in this set.
- **Anycast address:** In IPv6, this type of address is newly introduced. This type of address is used to identify a set of interfaces, typically belonging to different nodes. A packet destined for an anycast address will be delivered to one of the interfaces in this set, normally the nearest interface in the set. For instance, when a user wants to access to a service of a provider with global-wide server deployment. Using anycast address, the user request will be directed to the nearest server from the user for the most responsive and context-aware service.

With a 128-bit long address, it is hard for IPv6 to have a compact and human readable form as with IPv4 addresses. In IPv6, hexadecimal digits are used for address presentation instead of decimal digits. Recall that each hexadecimal digit represents 4 bits, consequently, an IPv6 address is represented by 32 hexadecimal digits. These hexadecimal digits are arranged in eight blocks, 16 bits or 4 hexadecimal digits each, separating from each other by colons (":"). An example of an IPv6 address is:

2001:0DB8: 0000:2A5F: 0000:0000:0000:A75B

As shown from the above example, even after using hexadecimal digits, an IPv6 address is still very long. To have a more compact representation, several shortening rules were defined. First, whenever there are leading zeros in a block of four hexadecimal digits, these leading zeros can be dropped. Second, if there are one or more blocks of consecutive 0s, these blocks can be shortened by replacing with a double colons ("::"). It is important to note that the second rule can only be applied once as it is impossible to recover the original IPv6 address otherwise. Following these rules, the above IPv6 address can be rewritten as follows, where the application of shortening rules are highlighted in blue (first rule) and red (second rule):

2001:**DB8**:**0**:2A5F**::**A75B

Similar to IPv4, a prefix in IPv6 is also represented in the form of **<IPv6 prefix>/<prefix length>**, where prefix length is a decimal value used to indicate the number of leftmost bits makes up the network part of the prefix. For example, the above address can be presented in a prefix form as 2001:DB8:0:2A5F::A75B/64. Based on IPv6 prefixes, some of the notable address types defined in IPv6 are presented as in table

Table 5.11: Some notable address types defined in IPv6.

IPv6 Prefix	Descriptions
::0/0	Default route for routing purpose.
::1/128	Loopback address, similar to that in IPv4.
FE80::/10	Link local address
2001:DB8::/32	Global Unicast address
FF00::/8	Multicast address

Transition to IPv6

As IPv6 was not designed for backward compatibility with IPv4, and with the already mass deployment of IPv4, there would be a transition time before IPv6 becomes the universal and the only Internet Protocol on the Internet. In fact, the adoption of IPv6 is rather slow since its ratification in the 90s. As for the middle of 2019, according to Google statistics[3], the usage of IPv6 constitutes only about 25% the traffic in the Internet. However, with the exhaustion of IPv4 addresses, IPv6 is increasingly adopted in recent years. The deployment of IPv6 would be first implemented in separate networks and as its usage becomes more common, some of these separate networks will merge together, making a larger IPv6 domain. These IPv6 domains will gradually converge into one universal IPv6 world. In the meantime, it is expected that IPv4 and IPv6 will coexist and some transition mechanisms must be defined for IPv6 and IPv4 nodes to be able to communicate with each other.

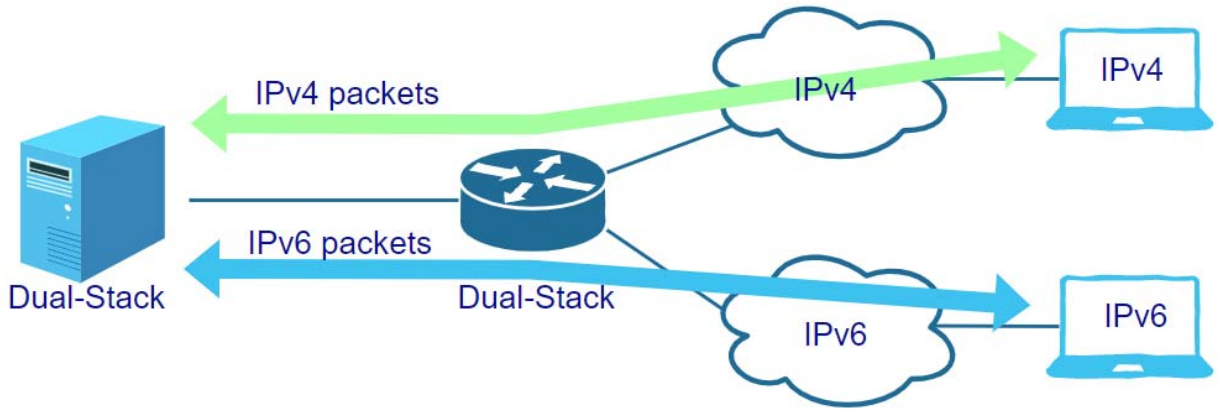


Figure 27: Dual-stack transition.

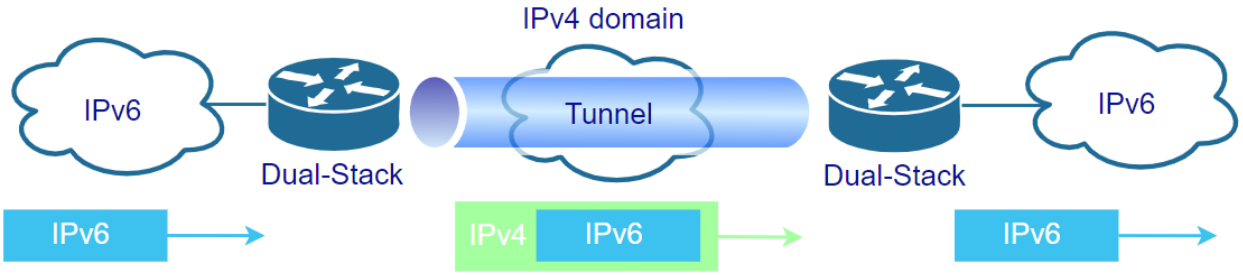


Figure 28: Tunneling transition.

The most obvious way to do this transition is to use a **dual-stack** approach, in which devices are updated to support both IPv4 and IPv6 protocol stacks. These devices will use their IPv4 stack to communicate with IPv4-only devices and IPv6 stack with IPv6-only devices as illustrated in Figure 27. In fact, for end devices, most operating systems nowadays provide supports for both IPv4 and IPv6 on the same interface. However, for this approach to be applicable, all the routers on the network must also be able to support dual-stack, which is a far more challenging problem due to historical reasons.

[3] Google IPv6 statistics: https://www.google.com/intl/en/ipv6/statistics.html

The second approach is **Tunneling** where the IPv6 packet is re-encapsulated in an IPv4 packet so that it can travel through IPv4-only networks as illustrated in Figure 28. In this case, two dual-stack routers are separated by an IPv4-only domain and are configured to cooperate with each other to form a tunnel between them for transporting IPv6 packets through this IPv4-only domain. When an IPv6 packet reaches one end of the tunnel, the router encapsulates the IPv6 packet inside an IPv4 packet but with the destination of the router on the other end of the tunnel. Being an IPv4 packet, this packet can travel through the IPv4-only domain to the other end of the tunnel with ease. At the other tunnel end, the egress router will remove the outside IPv4 header to retain the original IPv6 packet and forward this packet normally in the IPv6 domain.

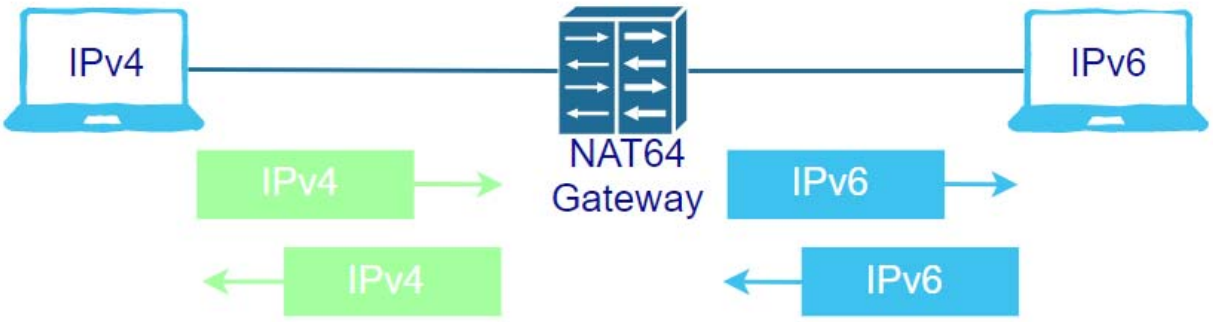


Figure 29: NAT64 transition.

For an IPv6 device to communicate with an IPv4 device and vice versa, **NAT64** is used. NAT64 is basically a protocol translation to translate between IPv4 and IPv6 packets. In this approach, a NAT64 gateway is used to connect between the IPv4 and the IPv6 domain. To do the protocol translation, the gateway must strip off the header of the ingress packet and replace it with a new header that is compatible with the egress network with an appropriate source address as illustrated in Figure 29. This process is more complicated than NAT as not only the addresses but also the entire packet header must be translated.

5.3 Internetworking Routing

One of the most fundamental and most important functionalities of the network layer is to support end-to-end delivery of packets through a complex and dynamic internetwork environment that connects together by routers. As such, the end-to-end packet delivery reduces to the capability of the routers to find a good path to forward the packet to its destination. In particular, when a packet arrives at a router, the router needs to decide which interface it should forward the packet to, for the packet to reach its desired destination. In the previous sections, we assumed that the routers on the way already knew where they should forward ingress packets by referring to their forwarding tables. In this section, we will take a further step to look at path finding or routing algorithms that enable routers to determine their forwarding tables.

As the path between the source and destination host could include different links of different capability and the connected topology is under constant change, the performance indicator for routing algorithms would include the following:

- **Accuracy:** a good routing algorithm must be able of finding a correct path to the destination, if available. This requirement is straightforward to understand but is the most important characteristic of a routing protocol.
- **Fast convergence:** is the capability of a routing algorithm to reliably find good paths in a network quickly. In a complex environment, there may exist multiple paths between the source and the destination hosts. A good routing algorithm must be able to obtain the global knowledge about the network topology and process this information to select a good path in a short time. In addition, when there are changes happen in the network, it is especially important for the routers to quickly find alternated paths to route their packets.
- **Adaptability:** is the capability of a routing algorithm to adjust its routing decisions accordingly to different network conditions. In an enormous network, such as the Internet, the network dynamicity may vary greatly from one instant to another. For example, network failures and router failures may occur without any priori knowledge. In addition, depending on the time of the day or the occurrence of certain event, the traffic loads on the network may fluctuate drastically. A good routing algorithm should be able to be aware of these changes and responds accordingly by modifying the paths of traffic flows in the network.
- **Low processing and networking overhead:** in order to yield a good forwarding table, a routing algorithm requires information about the network status and processing power to digest this information. It is ideal that a routing algorithm should be able to calculate optimal paths in the network while consuming minimal processing and networking overhead.

5.3.1 Classifications of Routing Algorithms

Depending on the different perspectives, routing algorithms can be categorized in many ways. Based on the adaptability to network changes, routing algorithms can be classified into static and dynamic routing. In **static routing algorithms**, the forwarding table are pre-calculated and manually loaded into each router. This approach does not require expensive processing and networking overhead and is well-suited in small and simple networks. One prime example of static routing is its use in home network where there is only one connection to the Internet Service Provider. However, static routing becomes inconvenient when the network size and complexity grows. In this case, the pre-calculated and fixed routing may not well-behave nor even work at all when there are changes in the network. In contrast, **dynamic routing algorithm** requires the regular or event-based exchange of routing information among routers in the network to determine the current status of the network. Then based on this information, each router in the network calculate independently to yield its own forwarding table. In this way, dynamic routing is more adaptable to network changes and is suitable to operate in large and complex network environment in an autonomous manner. However, dynamic routing demands network overhead for exchanging routing information and processing power from router for path finding process, which normally scale up with the network size and complexity.

Based on their approaches for processing routing information, dynamic routing algorithms can be categorized into centralized and distributed routing algorithms. In **centralized routing algorithms**, all routers possess the same global routing information of the current network state. Based on this common knowledge, the shortest paths to all of the destinations in the network can be calculated on each of the router in the network. A prime

example of centralized routing algorithm is the Dijkstra's or the link state algorithm, which will be described in the next section. On the other hand, in **distributed routing algorithms**, each router knows only its directly connected links and neighbors. By exchanging this knowledge with its neighbors, a router gets to know about the networks that its neighbors connect to, and hence, gradually discovers the network and finds the best path to the destinations in the network. Different than the previous approach, in distributed routing algorithms, each router sees the network through the view of its neighbors and no router possesses the global routing status of the network. An example of distributed routing algorithm is the Bellman-Ford or the distance vector algorithm, which will be studied in the next section.

Another way to classify routing algorithms is based on their partitioning of the network. To this end, the dynamic routing algorithms can be classified into flat and hierarchical routing. In **flat routing**, the entire network is treated as a contiguous domain and all routers can exchange routing information with its neighbor arbitrarily. This approach is simple to implement but it hinders network scalability. As the network grows bigger, so do the network and processing overhead of the routers. For example, a link failure would require all the routers in the network to recalculate their routing table, even if the link failure is very far from the current router and hence may not change its forwarding table at all. In this process, the entire network may be flooded with routing information exchanges, which are very expensive in terms of network overhead. In addition, in a very large network, changes may happen frequently, stressing the routers to do routing calculation all the time and exhausting the network bandwidth with routing information exchanges. To resolve this problem, in **hierarchical routing**, the network is partitioned into smaller domains. The routing information exchange and calculation is restricted in each domain only. The routing information exchange between different domains is executed through border routers and happens at a much lower frequency and can be controlled with routing policies. It can be seen that hierarchical routing is suitable for large networks and for connecting networks under different administrations. This approach also supports route aggregation between domains easier. However, this hierarchical routing requires a careful design to network partitioning and intricate configurations at the border routers.

5.3.2 Routing Costs

Before diving into the details of each routing algorithm, it is important to elaborate ways to differentiate how good a path is, or routing costs. A network topology can be abstracted by connected routers and the links that connect between them as illustrated in Figure 30. However, in this figure, it can be seen that there could exist multiple paths from one source to a destination. For example, from router A, to Router C, a packet can take the path A-B-C but can also take the path A-E-D-C or even A-E-B-D-C. Which one of the paths above is the best? To answer this question, first, we have to come up with a scheme to quantify the "goodness" of a link or a **link cost**. The higher a link cost, the less desirable it is to send a packet onto this link. In this way, the "goodness" of a path, or **path cost**, can be calculated as the sum of the link costs that are included in the path. For instance, the path cost of the path A-B-C can be calculated as in equation (5.4), where C_{ABC} is the cost of the path A-B-C and c_{AB}, c_{BC} are the link cost of the link A-B and B-C accordingly. In this way, the higher cost of a path, the less favourable it is to route the packet along this path.

$$C_{ABC} = c_{AB} + c_{BC} \quad (5.4)$$

The simplest way to quantify a link cost is to use the same cost for all the links. In this case, the path cost is the count of the number of hops, or routers that a packet must travel to reach its destination. However, one may argue that this is a naive approach because it would be more reasonable to give a lower cost to a 100Gbps link than a 128kbps satellite link. As such, link costs can be calculated based on the link characteristics such as bandwidth, delay or packet loss. However, one may further argue that a good link should satisfy not only one factor but also other criteria. For example, a good link should have a high bandwidth and low delay at the same time. As a result, link cost can be calculated as a combination of multiple characteristics of a link, or namely **composite cost**. It is important to note that care must be taken in defining link cost calculation to make sure that a lower link cost corresponds to a more desirable link. For example, to use bandwidth to calculate the link cost, the link cost should be inversely proportional to the link bandwidth while delay and packet loss can be calculated as linearly proportions to these factors. Another issue must be taken care when using composite cost is to normalize the different factors before adding them together since they are normally measured in different scales. In addition, since the characteristic of a link may be different depending on its transmission direction, its link cost may be different accordingly. For example, the link bandwidth may be higher in one direction than the other. Figure 30 illustrates a simplified case of symmetric cost where the link costs are the same in both directions.

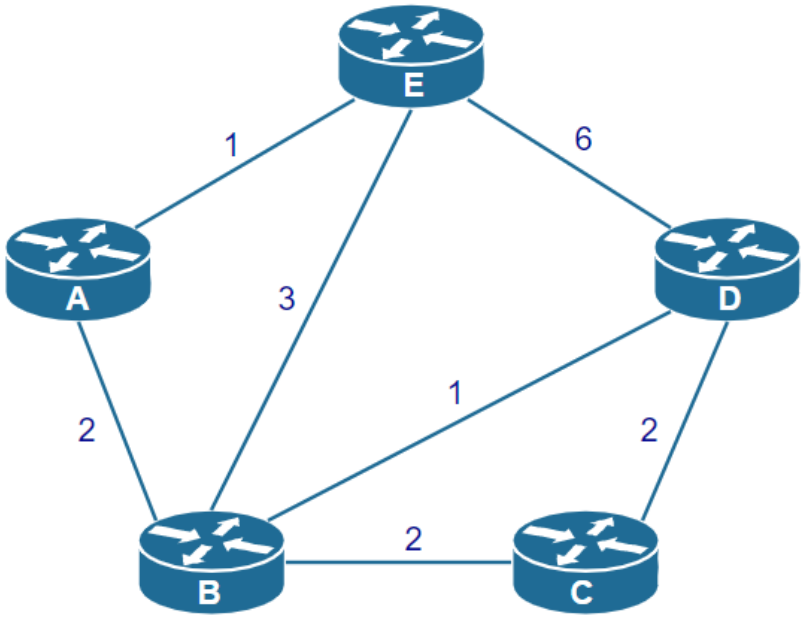


Figure 30: Example of a network topology and link metrics.

However, link cost must also be defined with care. For example, one may argue that the traffic load on the links would theoretically be a very good metric for measuring the "goodness" of a link. In practice, while such metric can be defined easily, its effects on routers and the overall network is devastating. Assume that such a cost is used in Figure 30 (instead of the denoted costs in the figure). A stream of traffic from A to D taking the path A-B-D will increase the load on this link, making its path cost increase. Hence, the path A-E-D with no load becomes more desirable and then, traffic from A to D will be switched to the latter path.

As the load is switched to the A-E-D path, the path A-B-D becomes empty and, hence its cost is lower than the path A-E-D. As a result, the traffic will be switched back and forth between these two paths causing **routing oscillations**. This phenomenon is undesirable as it makes inefficient use of the network capacity and putting the routers at the stress of calculating and updating the routing information all the time. In addition, this oscillation increases the need for network overhead in transporting these routing updates, exhausting the network bandwidth.

5.3.3 Dijkstra's Algorithm

Table 5.12: The Dijkstra's algorithm.

Initialization:

$$N = \{s\}$$

$C_{sd} = c_{sd}$ $for\ \forall d \neq s, c_{sd} = \infty$ if s and d are not directly connected

$$p_d = s\ if\ C_{sd} \neq \infty$$

Repeat

// add the next closest node to N

Add $d \notin N$, such that $C_{sd} = \min_{i \notin N} C_{si}$, to N

// Update shortest paths and costs if the path via the new added node is shorter

for $\forall k \notin N$

if $C_{sk} \leq C_{sd} + c_{dk}$

$C_{sk} = C_{sd} + c_{dk}$

$p_k = d$

end if

end for

Until all nodes in N

Dijkstra's algorithm is a centralized algorithm to find the best paths, in terms of the least path cost, from one source to all other nodes in the network. In order for Dijkstra's algorithm to work, each router (node) in the network must possess the global routing information about the link costs of all of the links in the network. This information can be obtained by having each node in the network send the information about its directly connected link to all other nodes, this process is referred as **link-state broadcast**. As a result, all nodes in the network share a common global understanding about the network status. Using this global routing information, each node place itself at its shortest path tree and calculate the shortest paths to all other nodes in an iterative way. The fundamental principle of this algorithm is for the source node to find and add a closest node to its shortest path tree in each iteration. For a network of n nodes, the algorithm takes at most n steps to finish.

Let N be the set of nodes that already has lowest cost to the source node s; p_d be the previous node to the destination node d on the path from the source node to the destination node; c_{xy} be the link cost between node x and y; and C_{sd} be the path cost from the source node s to the destination node d. The Dijkstra's algorithm can be calculated iteratively as in Table 5.12.

As an illustrative example, let's use the network depicted in Figure 30 and find the shortest path tree for node A. The iterations of Dijkstra's algorithm are shown as in Table 5.13. In the initialization iteration, the source node (A) is added to the set N and the path costs to all other nodes are calculated. If the destination node is not directly connected to the source node, the cost is calculated as ∞. The previous node for those paths with non-infinity costs is this source node. After the initial iteration, it can be seen that the path to node E has the lowest cost, hence, E is chosen to be added to the set N. In the next iteration, the paths and their costs via E are determined. In this case, the new path to D via E will be discovered with the path cost of $1 + 6 = 7$. The path to B via E has the cost of $1 + 3 = 4 > 2$, the cost of the previously discovered path directly connected B with A and hence, this new path does not replace the old one. It is noted that since E does not have a direct connection to C, D_C is still ∞. In iteration 2, since the path cost to B is the lowest, B is added to N. In this iteration, the path to C via B is discovered with the path cost of 2+2=4. In this iteration, the path to D via B is also discovered with the path cost of $2 + 1 = 3 < 7$, the path cost to D via E. Consequently, this new path replaces the previous one with B be the previous node on the way to the destination. With the above detail explanation, it would be easy for the readers to trace the algorithm operations in iterations 3, and 4. After iteration 4, all the nodes in the network are already added in the set N, hence, the algorithm terminates.

Table 5.13: Dijkstra's algorithm in action with the topology in Figure 30.

Iteration	N	D_B, p_B	D_C, p_C	D_D, p_D	D_E, p_E
Initialization	A	$2, A$	∞	∞	$1, A$
1	A, E	$2, A$	∞	$7, E$	
2	A, E, B		$4, B$	$3, B$	
3	A, E, B, D		$4, B$		
4	A, E, B, D, C				

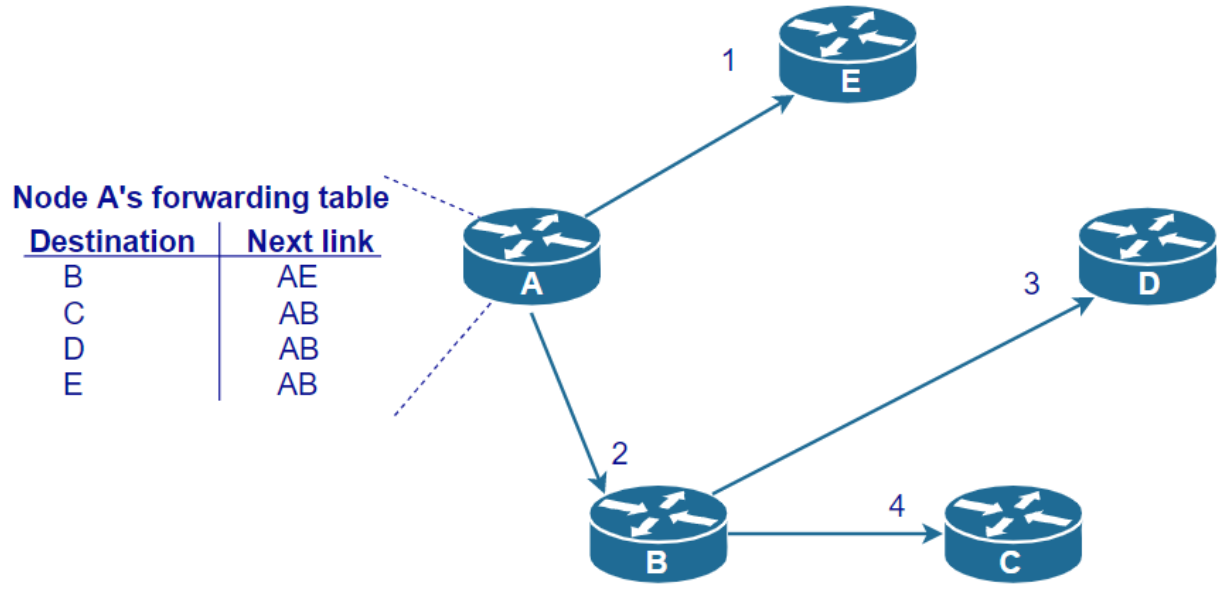


Figure 31: Dijkstra's shortest path tree with A as the source node

After Dijkstra's algorithm terminates, we are left with the information about the least path cost to all the nodes and the predecessor for each of the destination node. This information allow tracing back the shortest travel path to the destination to build the shortest path tree with the source node at the root of the tree. For example, we can trace back the path from A to D according to Table 5.13 as follows. The predecessor of D is B, the predecessor of B is A. Consequently, the shortest path from A to D is A-B-D. Figure 31 illustrates the resulting shortest path first tree from A to all other nodes and the forwarding table of A accordingly to this shortest path tree.

5.3.4 Bellman-Ford Algorithm

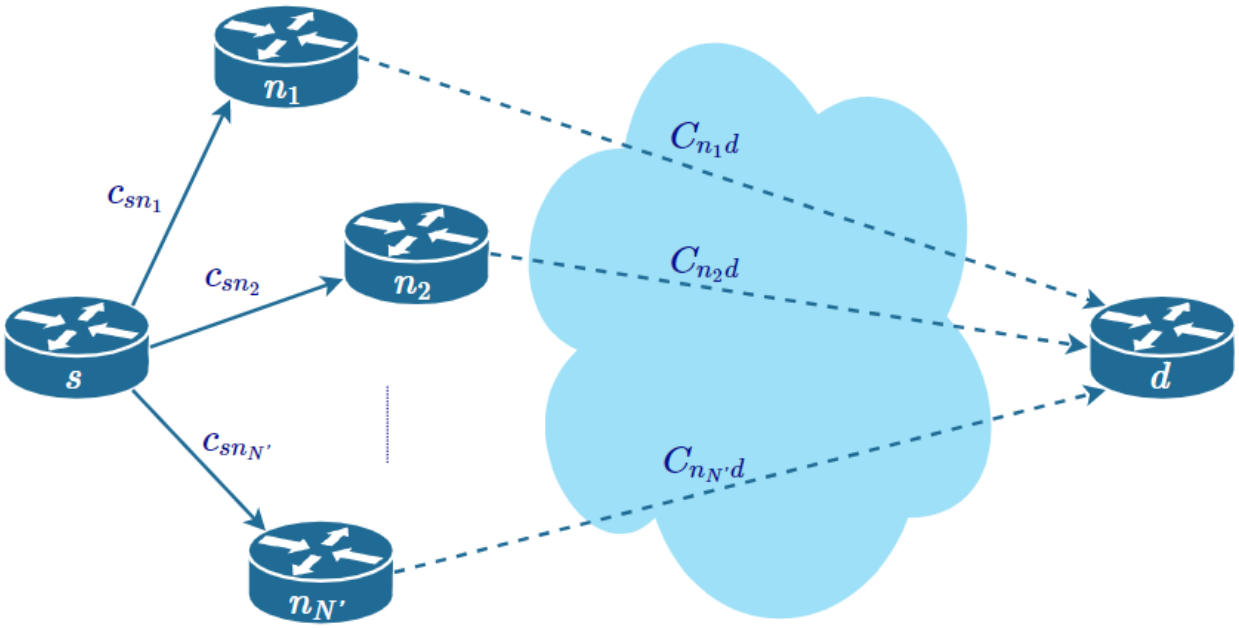


Figure 32: Illustration of Bellman-Ford equation.

The second class of dynamic routing protocol that we will study in this section is the Bellman-Ford algorithm, or normally referred to as distance vector routing algorithm. Different from Dijkstra's algorithm, the distance vector algorithm operates based on the partial view of its neighbors about the network to calculate the best path to each destination. Initially, each node is only aware of its directly connected links and directly connected neighbors. Each router will then share its knowledge about the network with its directly connected neighbors. By receiving these updates, a router could re-calculate its best paths in the network. Since all routers see the network only through the perspective of their directly connected neighbors, this algorithm is sometimes referred to as "routing by rumors".

The determination of best path in Bellman-Ford is based on the intuition that if a node B is on the best path (lowest cost) from node A to node C, the best path from A to B and from B to C must be on this same best path from A to C. It is easy to justify this intuition as if there exist another path with lower cost from B to C, for instance, then one can just use this path to replace the appropriate section in the A-C path, making the new A-C path having lower cost and contradicting the previous assumption that the old A-C path is optimal. As a result, the lowest path cost from a source node s to a destination node d, C_{sd} can be determined through its directly connected neighbors using equation (5.5) where $N' = \{n_1, n_2, \dots n_N\}$ is

the set of all directly connected neighbors of s, c_{sn} is the cost of the directly connected link from s to its directly connected neighbor n_i, and $C_{n_i d}$ is the path cost from the neighbor n_i of s to the destination node d.

$$C_{sd} = \min_{n_i \in N'} \{c_{sn_i} + C_{n_i d}\} \tag{5.5}$$

Equation (5.5) is also referred to as Bellman-Ford equation and is illustrated in Figure 32. As illustrated in this figure, if a source node s needs to find a best path to a destination node d, this best path must traverse through one of its directly connected neighbors. As a result, the source node does not have to know about the global information beyond its neighbor and can still find the best path to the destination node by choosing the path via one of its neighbors which has the minimum total cost $\{c_{sn_i} + C_{n_i d}\}$. Applying this equation to the topology in Figure 30, we can calculate the minimum cost path from A to D as in equation (5.6). This equation shows that the lowest cost path from A to D is via A's neighbor B and having the path cost of 3, which agrees with our calculation in Dijkstra's algorithm.

$$\begin{aligned} C_{AD} &= \min\{c_{AB} + C_{BD}, c_{AE} + C_{ED}\} \\ &= \min\{2 + 1, 1 + 4\} = 3 \end{aligned} \tag{5.6}$$

Table 5.14: The Bellman-Ford algorithm for each node in the network.

Initialization:

Node s knows its directly connected neighbor set N' and the cost c_{sn_i} to its directly connected neighbor n_i

Node s calculates its estimate of path costs to all directly connected neighbor d

$$C_{sd} = c_{sd}$$

Node s advertises its set of C_{sd}'s to all neighbors

Repeat

// Update shortest paths

for each other node d in the network

$$C_{sd} = \min_{n_i \in N'} \{c_{sn_i} + C_{n_i d}\}$$

end for

// Advertise updated shortest paths

if at least one of the C_{sd} was updated

advertise the set of C_{sd}'s to all neighbors

end if

Until no changes occur in the previous iteration

However, to compute equation (5.5) and (5.6) there is a issue of knowing $C_{n_i d}$, the best path cost from a neighbor of the source node to the destination node. In a simple topology like that in Figure 30, it is easy to trace the topology to find this result but in a big network, this value is generally not straightforward to find. Luckily, if we allow equation (5.5) to be executed iteratively and in a distributed manner on each router in the network, the routing information will be propagated hop-by-hop and be optimized gradually as the routing

information is propagated. The distributed Bellman-Ford algorithm for each node in the network is illustrated in Table 5.14.

As an illustrative example, let's use a simplified network depicted in Figure 33 for finding the forwarding table for all the routers using Bellman-Ford equation. It is noted that since Bellman-Ford algorithm is executed in a distributed manner, we have to reduce the number of routers from 5 in the previous example to 3 to reduce the space without compromising the algorithm illustration. In this figure, the estimated shortest path costs for all the routers are presented. In the initialization step, each router is only aware of its directed connected neighbor and the corresponding link costs. By exchanging this set of estimated minimum path costs, each node updates its shortest path cost to each of the destination in the network. For instance, the shortest path cost update for node A is illustrated in equation (5.7). The path cost that are changed are highlighted in Figure 33. It is noted that in this figure, we use the notation (X, Y) for the path cost where X denotes the path cost and Y denotes the next hop neighbor on this path of best cost.

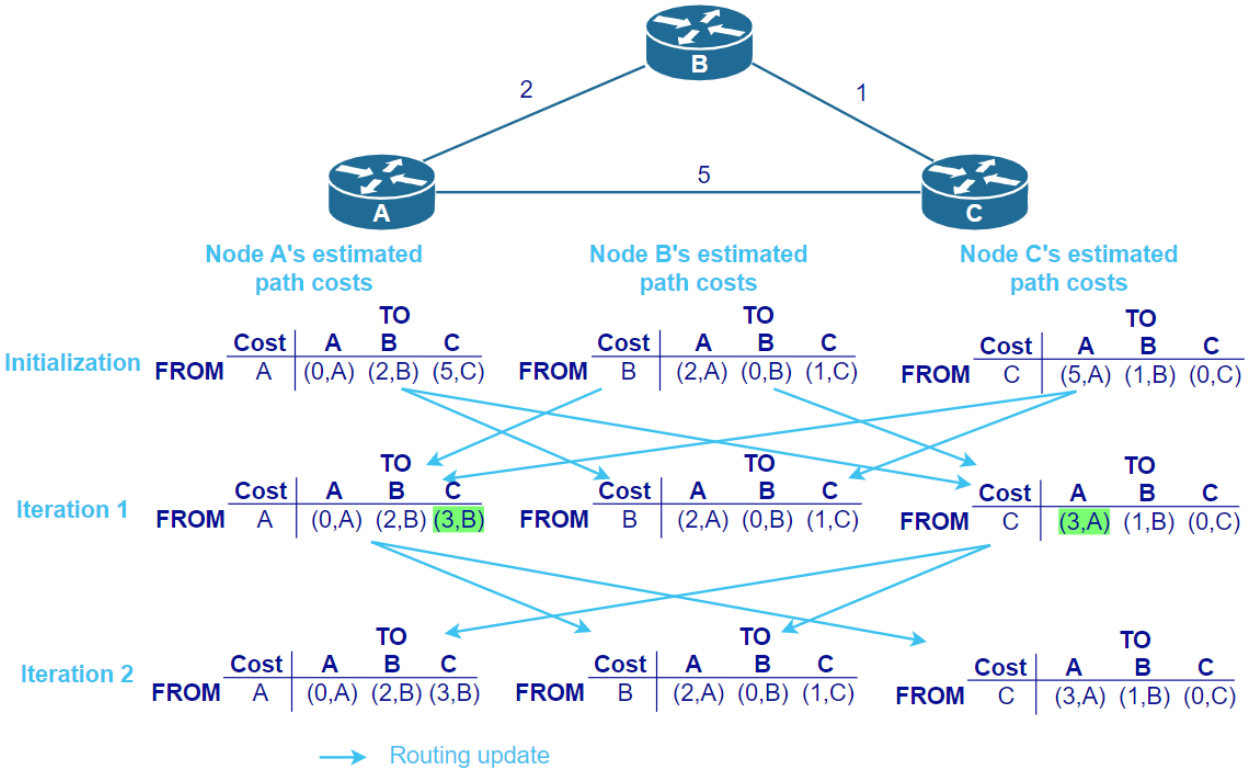


Figure 33: Bellman-Ford algorithm with routing updates in a simple topology.

$$C_{AB} = \min\{ c_{AB} + C_{BB}, c_{AC} + C_{CB}\} = \min\{2 + 0, 5 + 1\} = 2$$
$$C_{AC} = \min\{ c_{AB} + C_{BC}, c_{AC} + C_{CC}\} = \min\{2 + 1, 5 + 1\} = 3 \quad (5.7)$$

Similarly, the shortest path costs of node C are updated accordingly. After iteration 1, since node B does not have its estimated path cost change, it does not send update to its neighbors. In iteration 2, since the routing advertisements from neighboring nodes no longer change the estimated path costs in node either A, B and C, no node continues to advertise its estimated path cost and the algorithm terminates.

Count-to-infinity issue

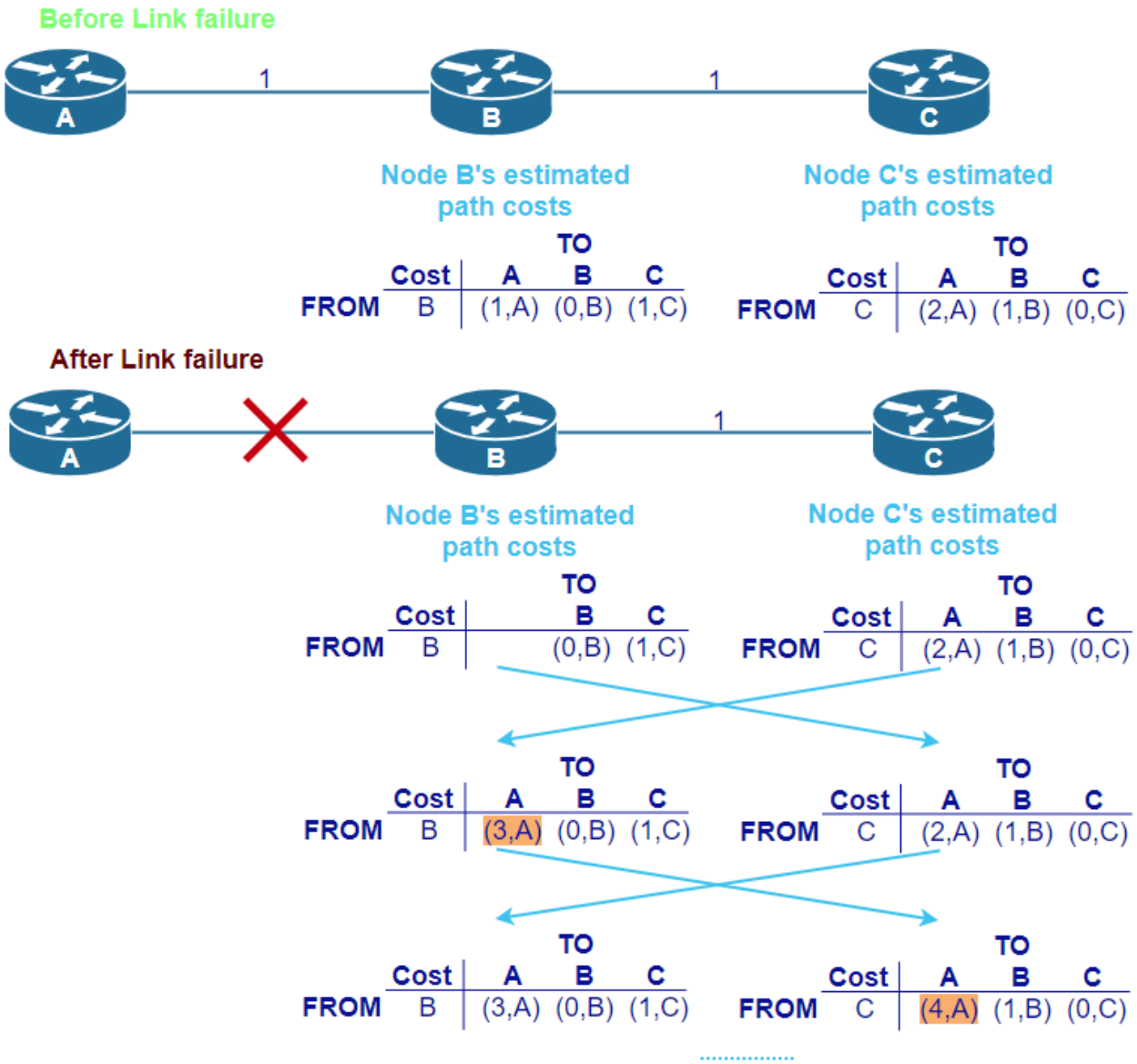


Figure 34: Count to infinity issue.

From equation (5.5), the min operator indicates that Bellman-Ford algorithm favours a better routing information but not the worse one. This characteristic helps the algorithm to select the best path quickly but does not help when the routing information degrades due to link failure for instance. Figure 34 illustrates a simple topology with three nodes: A, B, and C. For simplicity, the link cost for all links are unified at 1. It can be verified that the estimated best path costs to A from B and C are 1 and 2 respectively. Assume that the link between A and B fails. B removes this entry from its estimated path cost. However, at the next update interval, B and C exchanges their estimated path costs. In its estimated path cost, C indicates that it can reach A with the path cost of 2; hence, B falsely learns that it can reach A through C with the path cost of $2 + 1 = 3$ and insert this entry into its estimated path cost table. In the next iteration that B and C exchange their path costs, C learns that B has just changed its

path cost to A from 1 to 3, so C updates its estimated path cost table and change the cost of his link to 3+1=4. In this way, as the nodes exchange routing information, the estimated path cost of the route to A increases gradually to infinity (or up to a large software-defined number), when the software implementation declares that the path to A is no longer available. This increase to infinity behaviour is called the **count to infinity** issue.

The count to infinity issue causes the slow convergence speed in Bellman-Ford algorithm and causes routing loop. For instance, if there is a packet destined for A reach C, during this count to infinity process, this packet will be sent back and forth between B and C until its TTL expires. It can be seen that the root cause of this issue is due to the fact that each router in the network does not have the global information and just depends blindly on the information it receives from its neighbors. If the received information is wrong, the node that receives this information may miscalculate routing paths and hence, causing rooting loops.

One of the solutions for this count to infinity problem is to use split horizon with poison reverse technique. In this case, each router will tailor its estimated path cost table before sending to its neighbors to avoid spreading outdated and wrong information about the network. In particular, if a neighbor is on the best path to a destination, the source router will modify its estimated path cost table for this router so that the cost to this destination is set to infinity. As the cost was set to infinity, the received neighbor will not use reversed path for the mentioned destination and the confusions as above can be avoided.

5.3.5 Hierarchical Routing

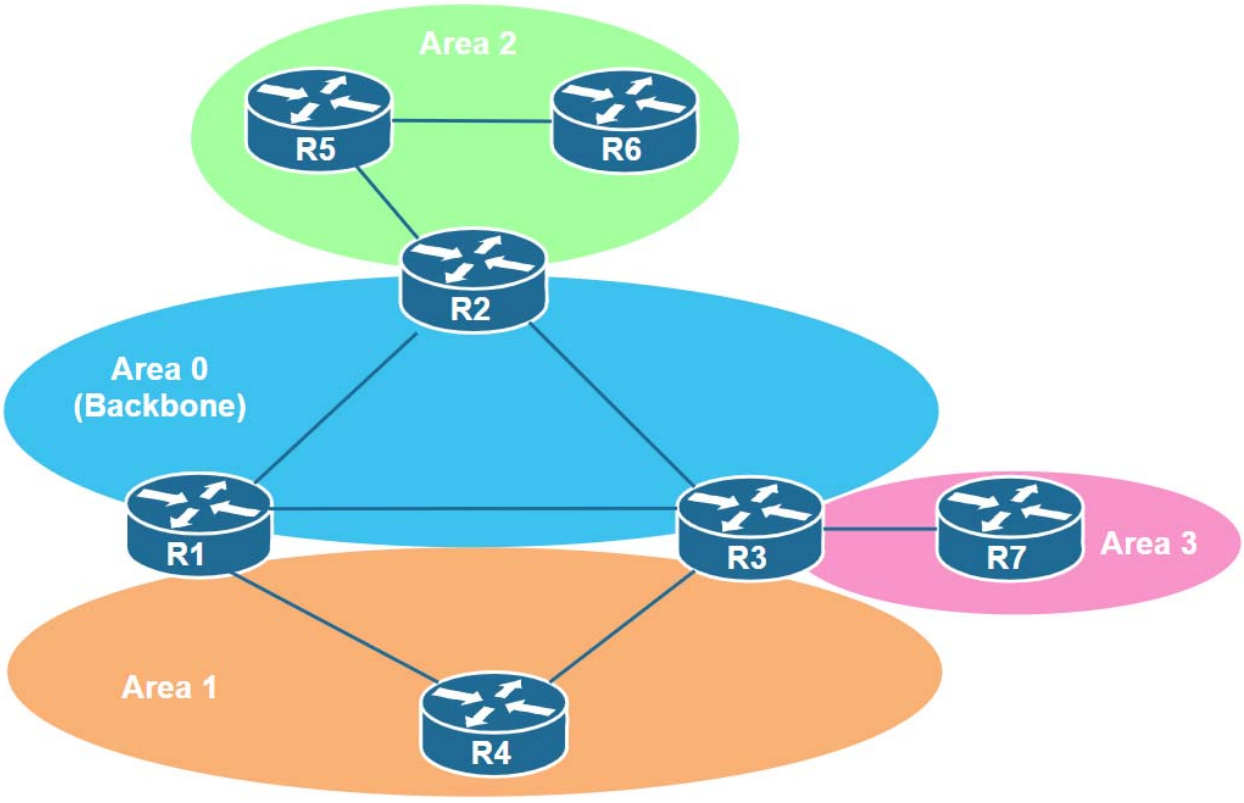


Figure 35: Areas in OSPF.

Despites the differences in their mechanisms, the use of routing algorithms such as Dijkstra and Bellman-Ford can support routing of packets between two arbitrary nodes in the network. However, as the network size increases, the convergence time and overhead of these approaches also increase, making them unsuitable for a large-scale network. One approach to resolve this problem is to divide the large network into smaller partitions where each partition can be efficiently supported by one of the above routing algorithms.

Area-based hierarchical routing

As a first step towards this objective for scaling, consider a scenario of enlarging a network under a single administrator. In this case, using the same partitioning idea, **Open Shortest Path First (OSPF)** protocol divides the routing domain into multiple areas. Despites using Dijkstra's algorithm, the link-state broadcast is constrained within each area only. In this way, OSPF could reduce significantly the burden in networking and processing overhead in comparison with the case when the entire routing domain uses Dijkstra's algorithm without partitioning.

Structure wise, as illustrated in Figure 35, there are two main types of areas in OSPF: normal area and backbone area (area 0). The philosophy behind area-based hierarchical routing of OSPF is that the backbone area serves as the network backbone where the normal areas are attached to and the transfer of packets between normal areas must happen through this area 0. In OSPF, it is required that the routers that make up area 0 must be physically contiguous. OSPF areas are defined per interface of each router in the network. As such, one router can reside in multiple areas. Those routers that have one of their interfaces connects to area 0 are called **Area Border Router (ABR)** and are the gateways for inter-area traffic. For example, in Figure 35, R1, R2, R3 are ABRs.

Routing wise, within each area, the routing exchange and best path determination happen according to Dijkstra's algorithm. As such, intra-area routing happens exactly the same as described above. Inter-area routing, however, is a bit more complicated. At the ABRs, since they participate in multiple areas, they must maintain routing exchanges and shortest path trees separately for each area to which they connect to. An ABR distributes routing information regarding one area to another area as if the networks and nodes in the source area are directly connected to it with the related accumulated path cost. This exchange of routing information makes it feasible for the routers in another area to learn about the reachability of networks in other areas while significantly reduce the routing overhead.

For instance, as illustrated in Figure 36, from the perspective of the routers in area 2, the rest of the networks is attached to R2. Consequently, in order to transfer a packet to a destination that does not reside in the same area, R5 will send it to R2 and depend on R2 for transferring it to the appropriate destination. In its turn, R2 sees the network in the perspective of area 0. If the destination network resides in another area at the other ABR, R2 sees it as if it is directly connected to that ABR. In this way, R2 just forwards the packet to the appropriate ABR. In the case that there are two ABRs that connects to an area, such as area 1 in Figure 35, it is possible that for some destinations, the path through one ABR could be better than the other and the internal area router of area 1 must select one ABR to send the packet to. Luckily, when distributing the routing information from one area to another, the ARB keeps the accumulated cost from itself to the appropriate destination. As such, the internal router can easily choose which ABR to forward the packet to. For example, in Figure

35, when R4 wants to send a packet to R7, it will find that the path via R3 is a better choice due to lower path cost and choose this path instead for sending the packet to R1.

It can be seen that using the area-based approach with a backbone area allows the network to grow in size while maintaining its efficiency. However, it requires a careful network design since the administrator must define where the area boundary should be. Moreover, because the inter-area traffic must pass through the backbone area, reasonable redundancy level must be incorporated in this backbone area to avoid network disruption. In addition, due to the two-tier hierarchical structure, sub-optimal routing may occur when traffic between areas must travel through the backbone area even if a shorter path is available.

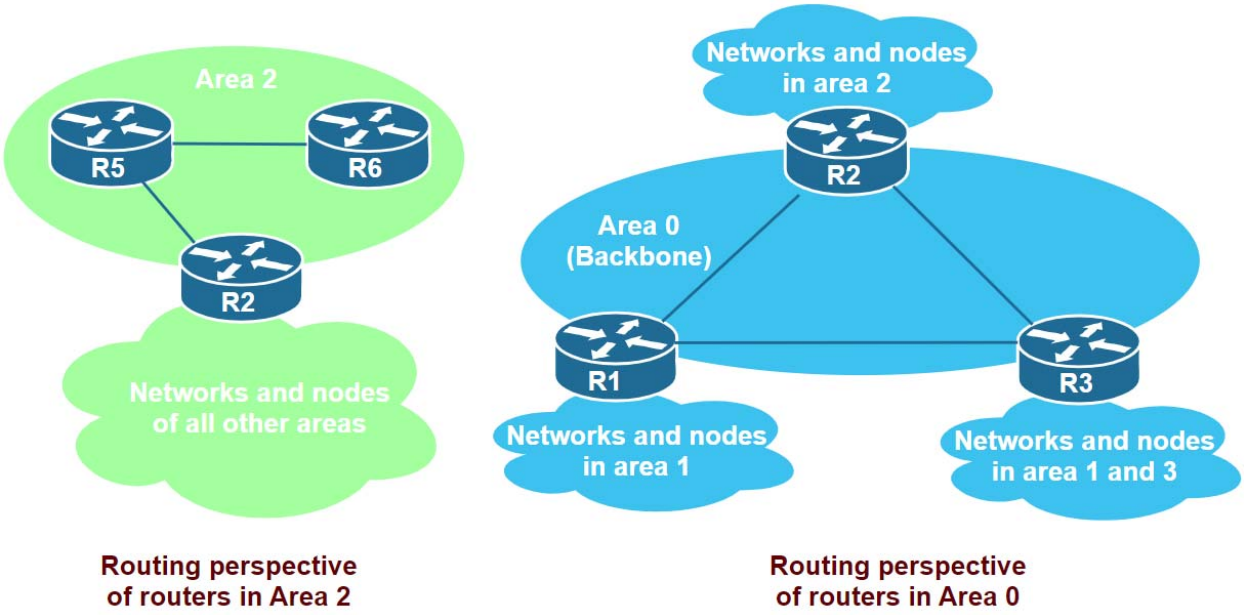


Figure 36: The view of routers internal to an area about the rest of the network.

It is noted that OSPF is a remarkably complex and feature-rich protocol. In this chapter, it is only used as an example of how area-based routing might be implemented. For example, one notable feature of OSPF is the ability to divide normal area into multiple types of areas, such as stub area, totally stub area, etc. to further optimize the routing information distribution between areas in particular network setups. Another important feature of OSPF is the use of virtual link to allow a router that is not physically connected to the backbone to participate in this area. Besides, OSPF also provides support for load sharing among multiple equal-cost paths. In addition, OSPF facilitates authentication between routers to prevent a malicious router from participate into the network, eavesdropping and disrupting routing information by injecting falsified routes. Interested readers can seek for further information regarding these and the other details of OSPF in [6] and [7].

Autonomous Systems (AS) and policy routing

The above area-based hierarchical routing would be enough for the case of an enterprise network where areas can be designed based on its needs and its capabilities. However, when the network is called to the scope of the Internet where multiple networks under different administrations connect together, this design is no longer a suitable solution. First, in the

Internet environment, the number of routing prefixes is enormously high, in terms of hundreds of thousands of entries. Being able to process and adapt to route changes with this degree of scalability would require a large amount of computing, storage and communication resources. As such, not all routers can afford to handle Internet routing. Moreover, the Internet as we see today consists of networks under various administrations. Each network administrator may have a different idea about how routing should be best implemented. For instance, what routing protocol is best suited for their scenario, and how to calculate link costs in their network. Moreover, due to security reasons, network administrators may not want to share the routing information of their networks inside out. In addition, when utilizing uplink provider bandwidth, they may want to have multiple providers at the same time for redundancy but may prefer to use the service of one provider to the other for a particular destination. In this case, the optimality of the route may be traded off for scalability and the path selection between these independently administrative domains may depend on some flexible policies rather than the best path cost.

Due to these reasons, another level of independence must be defined to support Internet routing. An **Autonomous System (AS)** is defined as a set of devices that is under the same administration. ASes are connected together via routers at the edge of each AS, namely AS gateway or **AS border router (ASBR)**. Based on this concept of AS, two types of routing protocols are required for Internet routing:

- **Interior Gateway Protocol (IGP):** this type of routing protocol is used for maintaining routing information within an AS. Normally routing within an AS is achieved using a common routing protocol, such as OSPF, with a common definition of path cost. The ultimate objective of IGPs is to find the optimal path based on the objective path cost.
- **Exterior Gateway Protocol (EGP):** this type of routing protocol is used for exchanging routing information between Autonomous Systems, or inter-AS routing. In order to materialize EGP, the **Border Gateway Protocol** (BGP) is the only official protocol on the Internet today. At the EGP level, route optimality is normally not as important as routing policies

To differentiate one AS from another, each AS is specified by an **AS Number (ASN)**. ASN is managed and assigned by Internet Assigned Number Authority (IANA)[4] and must be globally unique to be routable on the Internet. Figure 37 illustrates the connections between the ASes. Depending on the connectivity and the traffic forwarding policy, ASes are classified into three main types:

- **Sub AS:** this kind of AS has only one connection to one other AS. In this case, routing policy many not be relevant as the AS can only exchange traffic with one other AS only. The AS2 in Figure 37 is an example of this concept.
- **Multihomed AS:** this kind of AS has multiple connections to more than one other ASes. A multihomed AS allows only traffic that come from or destine to the addresses within this AS and does not allow traffic from one AS to transit through it to another AS. The AS1 in Figure 37 is an example of such concept. In this case Customer 1 enjoys the redundancy of connecting to two ASes simultaneously but since it pays the providers for carrying its traffic rather than providing services, it does not allow transit traffic from Provider 1 to pass

[4] Internet Assigned Number Authority: https://www.iana.org/

through it to Provider 2. In this way, 100% of the bandwidth of its connected links are dedicated to its traffic only.

- **Transit AS:** this kind of AS has multiple connections to other ASes. A transit AS is configured to allow traffic from other ASes to use its networking resources on the way to another AS. Service providers are normally configured to be transit ASes. AS 100 and 200 in Figure 37 are examples of transit ASes.

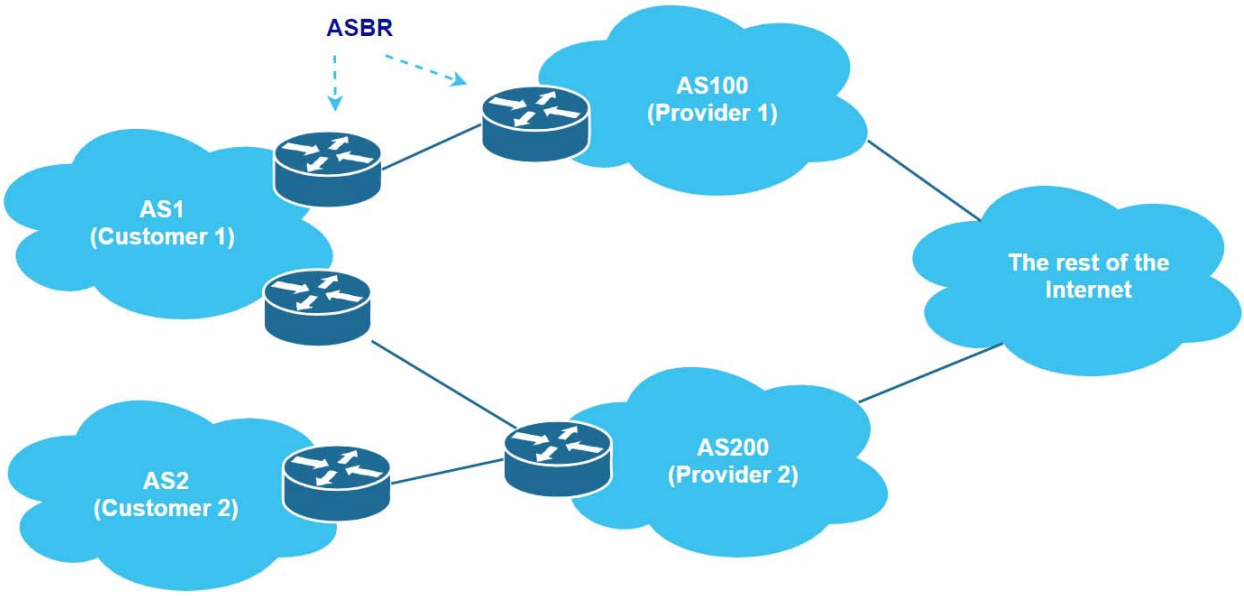


Figure 37: Connections between ASes

To appreciate the use of policy routing, let's look at the policies that the administrator of AS 1 may need to consider for regulating the traffic flow in his network. First, he has to make sure that the links to his providers carry only the traffic to and from his AS and do not carry additional traffic that may want to transit through his AS. Second, he may prefer the link to Provider 2 for Microsoft Azure services since he subscribes this link for Microsoft services only and the link to Provider 1 will be used for all other services. In addition, He may also want to use the two links to the two providers in a primary-backup or load sharing manner. In case of load sharing, he may want to share only 20% the traffic of his network through Provider 1 since the bandwidth of the link to Provider 2 is four times higher, or the link to Provider 2 is cheaper and more reliable. He may also want to route his traffic temporary in a few hours to Provider 1 in order to upgrade the software or hardware on the router that connects to Provider 2. These are some of the use cases that show the usefulness and effectiveness of policy routing when connecting ASes together. It is also demonstrated that the routing policies could be very subjective, temporary and may comes from business logics rather than technical, objective metrics.

Border Gateway Protocol (BGP)

For the purpose of exchanging routing information between ASes, BGP is the only official protocol that is used today with its current version is BGPv4. It is acknowledged that before BGP, Exterior Gateway Protocol (EGP) was used but it imposes limitations on the structure of AS connections and was then replaced by BGP.

BGP enabled routers must be explicitly configured to establish a neighboring relationship and becomes **peers**. The communication between two BGP peers is supported over TCP, a reliable transport protocol, which will be discussed in the next chapter. Over this TCP session, BGP peers exchange routing information and keep track of the availability of their neighbors. Since a routing update may consist of thousands of prefixes, the use of a reliable protocol such as TCP ensures the delivery of these changes to the appropriate BGP peers. Depending on the AS's relationship between the BGP peers, BGP connections are classified into **Internal BGP (iBGP)** and **External BGP (eBGP)** connections. If a BGP connection is established between two peers that reside inside the same AS, it is referred as iBGP, otherwise, if the BGP connection is established between two peers of different ASes, it is referred to as eBGP.

Routing information exchange between different ASes in BGP

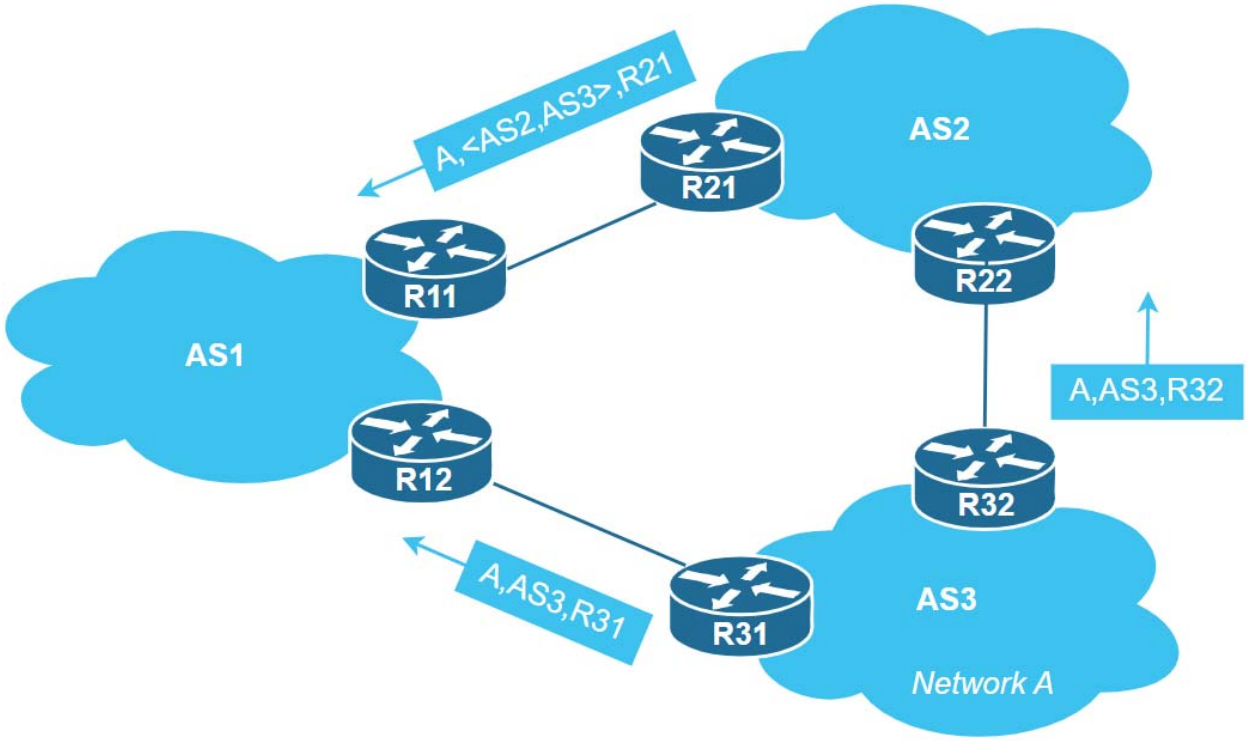


Figure 38: Exchange of routing information between different ASes in BGP.

In order to exchange routing information, BGP routers advertise route prefixes with a complete AS path for each prefix that should be taken to arrive at this destination network. As such, BGP is also referred as a path-vector protocol to differentiate with the distance vector and link-state approaches studied earlier. An illustrative example is shown in Figure 38 with 2 ASes and the advertisement of network A in AS3 to the other ASes. An advertised prefix in BGP is normally of a form "prefix, <list of ASes>, next hop", which literally means to reach the mentioned prefix, the path through the AS list should be taken and the address of the first hop router to the first AS on this list is specified by the "next hop". In addition, when a prefix is advertised outside of an AS, the ASN of this AS is prepended to the AS list of this prefix. In Figure 38, when R32 advertises network A to R22 in AS2, the advertisement is of the form "A, AS3, R32" to indicate that AS2 can reach network A through AS3 using its router R32. This prefix is further forwarded to R11 of AS1 by R21. The components of this update

is slightly altered when AS2 advertise this network to AS1. This advertisement is of the form "A, <AS2, AS3>, R21", where AS2 is prepended to the AS list to inform AS1 that to reach network A, AS1 can take a path through AS2 and AS3.

The AS list is also used in BGP to prevent routing loop. When receiving a BGP routing update from another AS, the routers in the receiving AS will examine the AS list and eliminate prefixes that contain the ASN of itself as it would cause routing loop. For instance, in Figure 38, R12 may advertise R31 about network A reachability through AS2. This advertisement would be of the form "A, <AS1, AS2, AS3>, R12". Upon receiving this update R31 sees its ASN (AS3) in the list and concludes that this routing information contains routing loops. R31 would then eliminate this prefix.

Propagating external routing information to AS internal

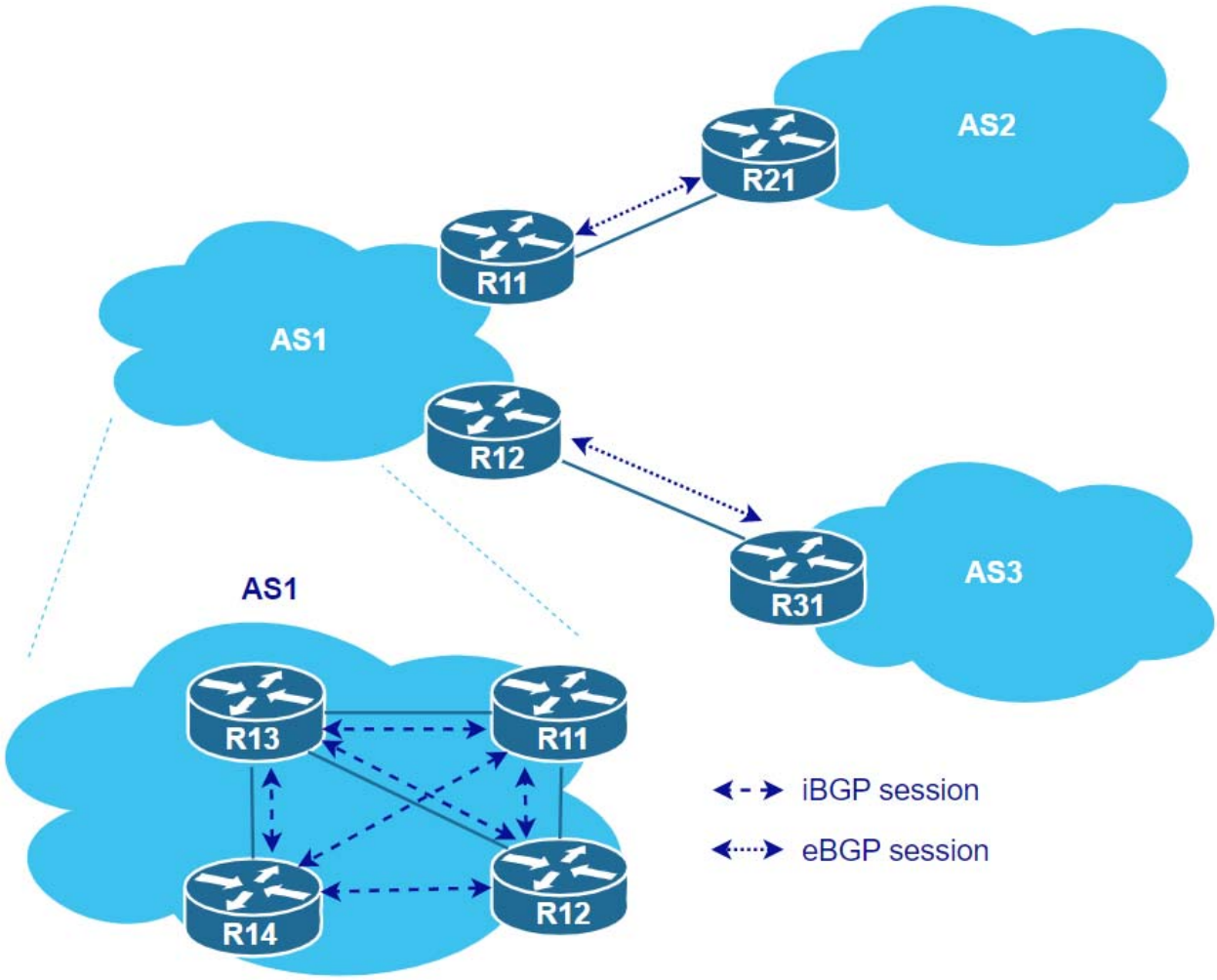


Figure 39: iBGP and eBGP sessions in AS1.

At this point, the routing information can be exchanged between ASBRs. However, there is a question that how internal routers of an AS can be aware of these prefixes to route their traffic to the appropriate direction. To this end, several approaches can be implemented. First, if the AS is a stub AS with only one link to the outside world or having multiple links but via only one ASBR, the ASBR can distribute a default route to the AS internal routers.

Hence, the AS internal routers can use their IGP routing information to reach AS internal networks while sending all other traffic to the ASBR.

Another approach is for the ASBR to redistribute the AS external networks into the IGP. This approach simplifies the configuration of AS internal routers which run over only an IGP. Besides, this method allows the AS administrator to selectively fine-tune the path or the ASBR to take for some particular prefixes. However, IGPs are not designed to process a large number of prefixes and a redistribution of hundreds of thousands of prefixes into an IGP could impose significant overheads on the processing and network resources of the routers in the AS.

Last but not least, the administrator can configure the AS internal routers with iBGP for them to be aware of the external prefixes and be able to find the best route to reach these. Figure 39 illustrates iBGP and eBGP sessions in AS1. To ensure routing consistency within the AS, all iBGP routers must be configured in a full mesh manner (other advanced methods are also available to mitigate the issue of too many iBGP connections in a big network but is not mentioned here). It is noted that the full mesh iBGP connections are logical connections and can be different from the physical connections between the routers in the AS as depicted in Figure 39. At this point, it is important to understand that the iBGP is used to bring the knowledge of external networks into the AS internal routers, but it does not replace the use of the IGP inside the AS. In particular, the lookup for an external network from an AS internal router involves both the routing information from the IGP and the iBGP. With the knowledge from iBGP, an AS internal router is aware of the reachability of the external network prefix and the best ASBR to forward the packet to, but it does not know the best route to reach this ASBR. As a result, it must consult its IGP for the best way to route packets to this ASBR. This two-step lookup is referred to as the **recursive lookup**, which illustrates the integration and cooperation between IGP and iBGP for packet forwarding.

Best route selection in BGP

As discussed above, there may be multiple paths to reach a network prefix in BGP. In IGP, determining the best route is quite straightforward with the use of an objective path cost, but in BGP, different AS may use totally different routing metrics, not mentioning totally different IGP, making the best path determination no longer an easy task. Instead, for best path selection in BGP, a set of attributes is associated with each prefix when being sent or received by a BGP router. For instance, **AS-PATH** is a BGP attribute which is essentially the AS list mentioned above. Beside AS-PATH, many attributes are defined in BGP, which can be tweak by administrators to impose their policy to the routing of packets to and from their AS. However, the discussion of policy routing would be lengthy and is out of the scope of this chapter. If policy routing is not configured or equal, the best route selection in BGP can be summarized as follows. First, the route with the shortest AS-PATH (fewest number of ASes in the AS list) is preferred. In the context of ASes, it may be not true that the path via a few small ASes is worse than that via one but a gigantic AS, but it is generally a good indicator of a good path. Then, if there are multiple paths with the same AS-PATH length, the route with the shortest path cost to the appropriate ASBR is selected (more precisely to the appropriate next hop in the routing advertisement). If all are the same, the router ID can be used as the tie-breaker.

It is important to note that BGP is a highly complex routing protocol that enables customizations of routes in various ways and we just scratch its surface in this section. Interested readers can refer to more in-depth materials for details regarding BGP operations and its support for policy routing in [8] and [9].

5.4 Conclusions

In this chapter, we studied the issues and approaches to end-to-end packet delivery through an internetworking environment. This is arguably one of the core functional blocks of the Internet. At first, an introduction to internetworking is given to raise the important issues arise when communication via an internetworking environment. With this understanding, the essential functions that should be presented for internetworking are presented. Furthermore, as routers are the glue that bond different networks together, a discussion is provided to examine the generic structure of a router and the functionalities and variabilities of their basic components, including the input and output interfaces, route processor, and witching fabric.

Next, we studied the Internet Protocol to understand how these functions mentioned above are materialized. Starting our journey with IPv4, we examined its packet structure to have a first impression of how different functions are supported. We examine how fragmentation can be facilitated with the use of the fields in an IPv4 packet. IPv4 addressing scheme is the next topic on the line with its basic descriptions, its subnetting procedure, Classless Inter-Domain Routing and Network Address Translation to confront the growth in number of devices today. Auxiliary protocols for supporting IPv4 are also studied such as DHCP, ICMP and ARP. Furthermore, an introduction to IPv6 was provided to discuss its improvement over IPv4, its packet structure and extension headers, addressing scheme and the transition from IPv4 to IPv6.

The chapter is ended with discussions on internetwork routing. First, the important characteristics of a routing protocol, its classifications and routing cost are studied to provide a big picture before going into the detail operations of each routing algorithm. After this preamble, we dived deeper into the routing world and explored the two most important routing algorithms that are widely use today, namely, the Dijkstra and the Bellman-Ford algorithms. In each case, their operations are illustrated through practical examples. Then, we wrapped up this chapter with hierarchical routing to resolve the issue of scalability in the Internet. Beginning with a routing domain under the same administration, we learnt the approach of area-based hierarchical routing, which is used in OSPF. By dividing the routing domain into multiple areas connected by a common backbone, it can be shown that the routing and processing overheads can be reduced significantly. Then, we enlarged our scope to routing between routing domains under different administrations and introduced the concept of autonomous system (AS) and policy routing. The feasibility of inter-AS routing was demonstrated through the different operations of BGP.

References

[1] M. Karo, M. Hluchyj, S. Morgan, “Input Versus Output Queuing on a Space-Division Packet Switch”, *IEEE Transaction on Communications*, Vol. 35 (12), 1987.

[2] RFC 3260; D. Grossman, “New Terminology and Clarifications for Diffserv”, 2002.

[3] IANA – Assigned Internet Protocol Numbers - http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml
[4] RFC 792; J. Postel, "Internet Control Message Protocol", 1981.
[5] RFC 8200; S. Deering, R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", 2017.
[6] RFC 2328; J. Moy, "OSPF Version 2", 1998.
[7] RFC 5340; R. Coltun, D. Ferguson, J. Moy, A. Lindem, "OSPF for IPv6", 2008.
[8] RFC 4271; Y.Rekhter, T. Li, S. Hares, "A Border Gateway Protocol 4 (BGP-4)", 2016.
[9] S. D. Fordham, "BGP for Cisco Networks: A CCIE v5 guide to the Border Gateway Protocol", CreateSpace Independent Publishing Platform, 2014.

Review questions

1. Why do we need internetworking? Briefly discuss the internetworking issues
2. Provide three distinct characteristics that network-layer addresses are different from data-link-layer addresses.
3. Briefly define connection-oriented and connectionless services. What is Virtual-Circuit? Why connectionless service can be referred to as best-effort service?
4. What might be a drawback of weighted fair queuing?
5. Suppose that all routers and hosts in the network are working properly, and the network is not under congestion. Illustrate a case in which a packet could be delivered to an incorrect address?
6. In comparison to the IPv6 header, which fields are removed from the IPv4 header? Why it is possible to remove these fields?
7. What is the motivation of Network Address Translation (NAT) and how does NAT work? Provide three disadvantages of NAT.
8. A setup uses two (or more) routers connecting a company to the Internet in order to provide some redundancy in case one of them goes down. Is this setup still possible with NAT? Justify your answer.
9. Briefly describe how a DHCP server work.
10. What are the tasks of ICMP and ARP in Internet Protocol (IP)?
11. Provide two methods that allow coexistence of IPv4 and IPv6. Explain how these methods collaborate to transmit a packet from the IPv6-based network to the IPv4-based network.
12. Suppose that, in Dijkstra's algorithm, we have the metrics (weights) in the nodes, not the links. Does the short-paths problem still make sense in terms of routing feasibility for this kind of network? Justify your answer.
13. Does Dijkstra's algorithm work for negative link metrics? Justify your answer.
14. Provide three disadvantages of Bellman-Ford algorithm?
15. What is the cause of count-to-infinity problem in Bellman-Ford algorithm? What damage would the count-to-infinity problem cause to the Bellman-Ford algorithm?

16. Provide the solution to resolve the count-to-infinity problem in Bellman-Ford algorithm.
17. Provide 3 benefits when dividing a network into multiple areas.
18. Illustrate an application where policy routing is beneficial.
19. How can a BGP router detect routing loop in the routing information it receives?

Problems:

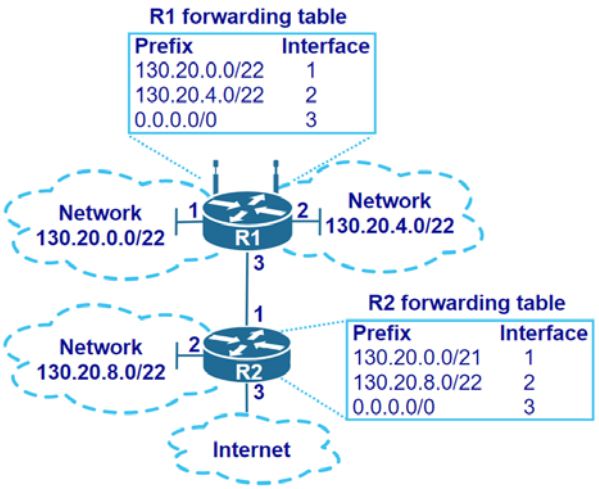
1. Suppose that a captured IPv4 header is represented in hexadecimal as the following:

   0x45 00 01 63 00 03 00 00 31 06 7D A5 0A 8F 1A F2 C0 1E 1F 04

   What is the length of the data portion of this packet? What is the source IP address of this packet.
2. Consider an IPv4 packet of total length of 2100 bytes to be sent through an interface with the MTU of 1500 bytes. Assume that the IP header is 20 bytes long.
   a) Show the total length, MF bit and the fragment offset of each fragment after sending through this interface.
   b) If these fragments need further fragmentation on a link with MTU=820 bytes, show the total length, MF bit and the fragment offset of each fragment of this packet after sending through this interface.
3. Given the following CIDR block as in the following table, find their subnet IP addresses and broadcast addresses.

CIDR block	Subnet IP address	Broadcast address
10.0.3.76/8		
123.128.16.221/16		
123.5.0.0/16		
192.129.255.0/24		

4. Suppose a network has a subnet mask of 255.255.248.0, what is the maximum number of hosts it can serve?
5. A library has 6 workstations, and each workstation consists of 15 computers. What mask could be applied to the library network to equally divide the network?
6. Routing prefixes 57.6.96.0/21, 57.6.104.0/21, 57.6.112.0/21, and 57.6.120.0/21 are transmitted to a router. If all of them use the same outgoing line, can they be aggregated?
7. Consider the router forwarding tables with route aggregation shown in the figure. What does the router R2 do if a packet with the following address arrive?
   a) 130.20.11.12.
   b) 130.20.7.12.
   c) 130.20.16.12.



8. A router has the following (CIDR) entries in its routing table:

Address/mask	Next hop
135.46.56.0.22	Interface 0
135.46.60.0/22	Interface 1
192.53.40.0/23	Router 1
Default	Router 2

For each packet of the following IP addresses, which interface or router will the packet be forwarded to?

   a) 135.46.63.10
   b) 135.46.57.14
   c) 135.46.52.2
   d) 192.53.40.7
   e) 192.53.56.7

9. Shorten the following IPv6 addresses:
   a) 0000:0000:6F53:6F82:AB21:67DB:BB37:7112
   b) 0000:0000:0000:0000:0000:005D: AB21:ABCD
   c) 0000:0000:0000: AB21: 7112: 11AA:0000: 0221

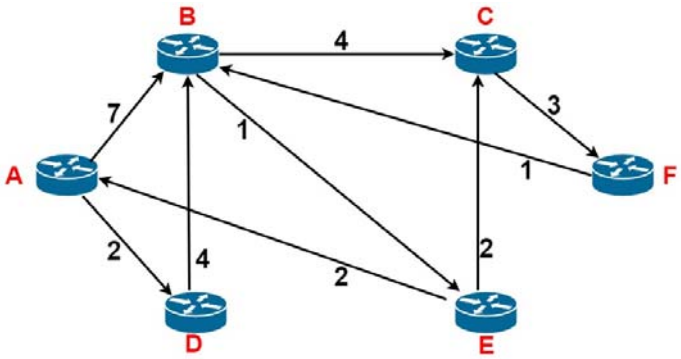
10. Consider three queues A, B, and C, each has two packets. The packet arrival and serving times are as follows.

Queue	Packet	Arrival time	Serving time
A	A_1	0	5
	A_2	1	3
B	B_1	0	4
	B_2	3	7
C	C_1	2	8
	C_2	5	7

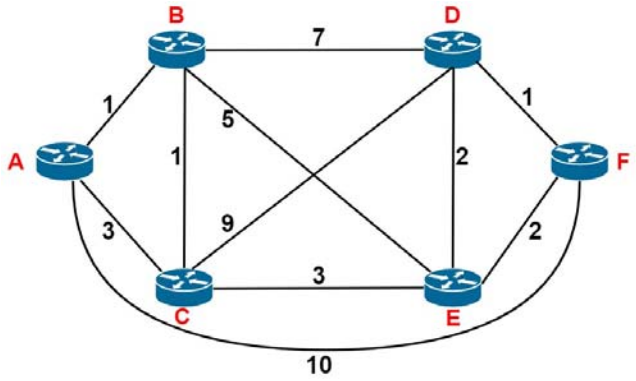
   a) Using Fair Queuing, calculate the serving order of each packet.
   b) Assume that queues A, B, and C are assigned with the weights $w_A = 1, w_B = 2$, and $w_C = 3$, respectively. Using the Weight Fair Queuing, calculate the serving order of each packet.

11. Two subnets are used in the same local network. Assume that when a subnet send a broadcast packet, the hosts of the other subnet will also see this packet.
   a) Suppose that each subnet uses one DHCP server. Explain a problem that might occur in this case. Propose a solution to resolve this problem.
   b) How will this scenario affect the ARP?

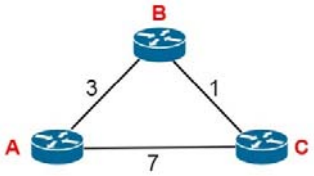
12. Given the following network having *directed links, meaning that the data can only traverse in one dirrection (i.e., arrow direction)*, use the Dijkstra's algorithm to find the shortest path from node A to other nodes. Provide the shortest path from A to C and calculate the corresponding cost?



13. Given the following network diagram, use the Dijkstra's algorithm to find the shortest path from node A to other nodes. Provide the shortest path from A to F.



14. Using the Bellman Ford algorithm to calculate the shortest path in the following network. Show the routing update including path costs of each nodes in each iteration.



15. Suppose that in the following network, the link from node B to node C is failed, show that applying the Bellman Ford algorithm to this network would result the count-to-infinity issue.

16. Consider a network of 3 routers. The routing table for each router is given as follows.

```
R0#sh ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
Gateway of last resort is not set

     10.0.0.0/8 is variably subnetted, 8 subnets, 4 masks
C       10.0.0.0/30 is directly connected, GigabitEthernet0/1
L       10.0.0.1/32 is directly connected, GigabitEthernet0/1
C       10.0.0.4/30 is directly connected, GigabitEthernet0/0
L       10.0.0.5/32 is directly connected, GigabitEthernet0/0
C       10.0.16.0/20 is directly connected, GigabitEthernet0/2
L       10.0.16.1/32 is directly connected, GigabitEthernet0/2
S       10.0.48.0/24 [1/0] via 10.0.0.2
S       10.0.64.0/20 [1/0] via 10.0.0.2
```

```
R1#sh ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
Gateway of last resort is not set

     10.0.0.0/8 is variably subnetted, 8 subnets, 3 masks
C       10.0.0.0/30 is directly connected, GigabitEthernet0/0
L       10.0.0.2/32 is directly connected, GigabitEthernet0/0
C       10.0.0.8/30 is directly connected, GigabitEthernet0/1
L       10.0.0.9/32 is directly connected, GigabitEthernet0/1
S       10.0.16.0/20 [1/0] via 10.0.0.1
C       10.0.32.0/20 is directly connected, GigabitEthernet0/2
L       10.0.32.1/32 is directly connected, GigabitEthernet0/2
S       10.0.48.0/20 [1/0] via 10.0.0.10
```

```
R2#sh ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
Gateway of last resort is not set

     10.0.0.0/8 is variably subnetted, 10 subnets, 3 masks
C       10.0.0.4/30 is directly connected, GigabitEthernet0/0
L       10.0.0.6/32 is directly connected, GigabitEthernet0/0
C       10.0.0.8/30 is directly connected, GigabitEthernet0/1
L       10.0.0.10/32 is directly connected, GigabitEthernet0/1
S       10.0.16.0/20 [1/0] via 10.0.0.5
S       10.0.32.0/20 [1/0] via 10.0.0.9
C       10.0.48.0/20 is directly connected, GigabitEthernet0/2
L       10.0.48.1/32 is directly connected, GigabitEthernet0/2
C       10.0.64.0/20 is directly connected, GigabitEthernet0/3/0
L       10.0.64.1/32 is directly connected, GigabitEthernet0/3/0
```

a) Based on the routing table, reconstruct the diagram of that network, including the interconnection between the routers, the subnet address and prefix length of each subnet, and all assigned IP addresses on each router according to each connection.

b) Based on the routing table, will a device in the network 10.0.48.0/20 with an IP address of 10.0.48.20 be able to ping any device on the network 10.0.16.0/20? If yes, show the series of forwarding routers of the ping request and ping response messages. What happen if the IP address of the source device is 10.0.51.100 instead?

c) Based on the routing table, will a device in the network 10.0.64.0/20 with an IP address of 10.0.64.20 be able to ping any device on the network 10.0.16.0/20? If yes, show the series of forwarding router of the ping request and ping response messages.