**Introducing TCP/IP**

**A Brief History of TCP/IP**

The very first *Request for Comments (RFC)* was published in April 1969, which paved the way for today’s Internet and its protocols. Each of these protocols is specified in the multitude of RFCs, which are observed, maintained, sanctioned, filed, and stored by the *Internet Engineering Task Force (IETF)*.

TCP first came on the scene in 1974. In 1978, it was divided into two distinct protocols, **TCP** and **IP**, and finally documented into an RFC in 1980. Then, in 1983, TCP/IP replaced the *Network Control Protocol (NCP)* and was authorized as the official means of data transport for anything connecting to *ARPAnet*. ARPAnet was the Internet’s ancestor, created by *ARPA*, the *DoD’s Advanced Research Projects Agency*, again, way back in 1969 in reaction to the Soviets’ launching of Sputnik. ARPA was soon redubbed DARPA, and it was divided into ARPAnet and MILNET (also in 1983); both were finally dissolved in 1990. But contrary to what you might think, most of the development work on **TCP/IP** happened at *UC Berkeley* in Northern California, where a group of scientists was simultaneously working on the *Berkeley version of Unix*, which soon became known as the *BSD*, or the *Berkeley Software Distribution series of Unix versions*. Of course, because TCP/ IP worked so well, it was packaged into subsequent releases of BSD Unix and offered to other universities and institutions if they bought the distribution tape. Basically, BSD Unix bundled with TCP/IP began as shareware in the world of academia and, as a result, became the basis of the huge success and exponential growth of today’s Internet as well as smaller, private, and corporate intranets.

Q: What is TCP/IP?  
A: TCP/IP is short for **Transmission Control Protocol / Internet Protocol**. This is a set of protocol layers that is designed to make data exchange possible on different types of computer networks, also known as heterogeneous network.

- Interview Q&A

Formun Üstü

Formun Altı

Formun Üstü

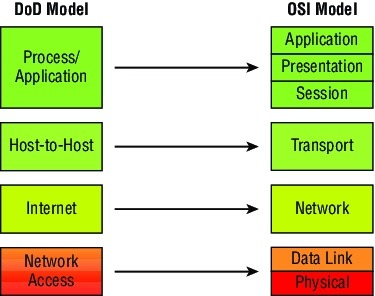
Formun Altı

**TCP/IP and the DoD Model**

The Department of Defense created TCP/IP to ensure and preserve data integrity. The DoD model is a condensed version of the OSI model and only has four layers that are:

* Process/Application layer
* Host-to-Host layer
* Internet layer
* Network Access Layer

The DoD and OSI models are identical in design and concept and have similar functions in similar layers.



**💡Tip:**

* When the different protocols in the IP stack are discussed, two layers of the OSI and DoD models are interchangeable. In other words, the Internet layer and the Network layer describe the same thing, as do the Host-to-Host layer and the Transport layer. The other two layers of the DoD model, Process/Application and Network Access are composed of multiple layers of the OSI model.

**1. Process/Application layer**

The Application layer of the DoD model is equivalent to the upper three layers of the OSI model, i.e., Session layer, Presentation layer, and Application layer. The Process/Application layer of the DoD model provides the following capabilities:

* Enable applications unicate with each other.
* Provides access to the services that operate at the lower layers of the DoD model.
* It contains a protocol that implements user-level functions such as mail delivery, file transfer, and remote login.

**2. Host-to-Host layer**

A host-to-host layer of the DoD model performs the same functions as the Transport Layer of [the OSI reference model](https://lms.clarusway.com/mod/lesson/view.php?id=1837). It handles issues such as flow control, reliable end-to-end communication, and ensuring error-free delivery of the data. Protocols that operate on the Host-to-Host layer are TCP and UDP.

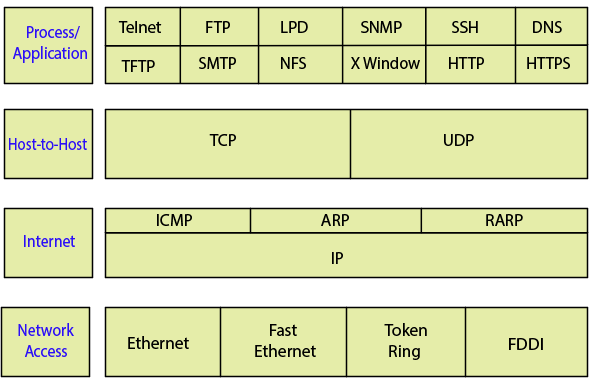
**3. Internet layer**

Internet layer of the DoD model performs the same functions as the Network layer of [the OSI reference model](https://lms.clarusway.com/mod/lesson/view.php?id=1837). It handles the packaging, addressing, and routing of packets among multiple networks. This layer also establishes a connection between two computers to exchange the data.

**4. Network Access Layer**

The Network Access layer of the DoD model is equivalent to the lower two layers (Data Link, and Physical) of the OSI model. The hardware connected to Network access layer are:

* Network medium: Cables like coaxial, twisted pair. Today, mostly, we use a wireless medium such as Bluetooth, WI-FI.
* Network Interface Card (NIC) has two types of addresses.
  + MAC Address- It is a 48 bits physical address.
  + IP Address – It is a 32 bits logical address.



**The Process/Application Layer Protocols**

**Telnet (TCP 23):** Telnet allows a user on a remote client machine, called the *Telnet Client*, to access the resources of another machine. The drawback of Telnet is that there are no encryption techniques available within the Telnet Protocols, so everything must be sent in the cleartext.

**FTP (TCP 20, 21):** File Transfer Protocol lets us transfer files, and it can accomplish this between any two machines using it. FTP allows access to both directories and files and can accomplish certain types of directory operations, such as relocating into different ones. FTP functions are limited to listing and manipulating directories, typing file contents, and copying files between hosts.

**SFTP (TCP 22):** Secure File Transfer Protocol is used when you need to transfer files over an encrypted connection. It uses an SSH session, which encrypts the connection, and SSH uses port 22, hence the port 22 is used for SFTP. Apart from the secure part, it’s used just as FTP is.

**TFTP (UDP 69):** Trivial File Transfer Protocol (TFTP) is the stripped-down, stock version of FTP. TFTP is fast and so easy to use. TFTP doesn’t offer the abundance of functions that FTP does because it has no directory-browsing abilities, meaning that it can only send and receive files. There’s no authentication as with FTP, so it’s insecure.

**SMTP (TCP 25):** Simple Mail Transfer Protocol detects the spooled mails and proceeds to deliver them to their destination. SMTP is used to send mail; POP3 is used to receive mail.

**POP (TCP 110):** Post Office Protocol gives us a storage facility for incoming mail, and the latest version is called POP3. How this protocol works is when a client device connects to a POP3 server, messages addressed to that client are released for download. It doesn’t allow messages to be downloaded selectively, but once they are, the client-server interaction ends, and you can delete and tweak your messages locally at will. A newer standard, IMAP, is being used more and more in place of POP3.

**IMAP (TCP 143):** Because Internet Message Access Protocol (IMAP) makes it so you get control over how you download your mail, with it, you also gain some much-needed security. It lets you peek at the message header or download just a part of a message. With it, you can choose to store messages on the email server hierarchically and link to documents and user groups, too. IMAP even gives you search commands to use to hunt for messages based on their subject, header, or content. As you can imagine, it has some serious authentication features—it supports the *Kerberos* authentication scheme that MIT developed. IMAP4 is the current version.

**RDP (TCP 3389):** Remote Desktop Protocol is a proprietary protocol developed by Microsoft. It allows you to connect to another computer and run programs. RDP operates somewhat like *Telnet*, except instead of getting a command-line prompt as you do with Telnet, you get the actual graphical user interface (GUI) of the remote computer. Clients exist for most versions of Windows, and Macs now come with a preinstalled RDP client. Microsoft currently calls its official RDP server software *Remote Desktop Services*. Microsoft’s official client software is currently referred to as *Remote Desktop Connection*. RDP is an excellent tool for remote clients, allowing them to connect to their work computer from home, for example, and get their email or perform work on other applications without running or installing any of the software on their home computer.

**TLS/SSL (TCP 995/465):** Both Transport Layer Security and its forerunner, Secure Sockets Layer, are cryptographic protocols that are useful for enabling secure online data-transfer activities like browsing the Web, instant messaging, Internet faxing, and so on. They’re so similar. They both use X.509 certificates and asymmetric cryptography to authenticate to the host they are communicating with and to exchange a key. This key is then used to encrypt data flowing between the hosts. This allows for data/message confidentiality, message integrity, and message authentication.

**SIP (VoIP) (TCP or UDP 5060/TCP 5061):** Session Initiation Protocol is a hugely popular signaling protocol used to construct and deconstruct multimedia communication sessions for many things like voice and video calls, videoconferencing, streaming multimedia distribution, instant messaging, presence information, and online games over the Internet.

**RTP (VoIP) (UDP 5004/TCP 5005):** Real-time Transport Protocol describes a packet-formatting standard for delivering audio and video over the Internet. It’s commonly employed for streaming media, videoconferencing, and push-to-talk systems—all things that make it a de facto standard in Voice over IP (VoIP) industries.

**MGCP (Multimedia) (TCP 2427/2727):** Media Gateway Control Protocol is a standard protocol for handling the signaling and session management needed during a multimedia conference. The protocol defines a means of communication between a media gateway, which converts data from the format required for a circuit-switched network to that required for a packet-switched network, and the media gateway controller. MGCP can be used to set up, maintain, and terminate calls between multiple endpoints.

**H.323 (Video) (TCP 1720):** H.323 is a protocol that provides a standard for video on an IP network that defines how real-time audio, video, and data information is transmitted. This standard provides signaling, multimedia, and bandwidth control mechanisms. H.323 uses the RTP standard for communication.

**SNMP (UDP 161/TCP 25):** Simple Network Management Protocol collects and manipulates valuable network information. It gathers data by polling the devices on the network from a management station at fixed or random intervals, requiring them to disclose certain information. This protocol can also stand as a *watchdog* over the network, quickly notifying managers of any sudden turn of events. The network watchdogs are called *agents*, and when aberrations occur, agents send an alert called a trap to the management station. Besides, SNMP can help simplify the process of setting up a network as well as the administration of your entire internetwork.

**SSH (TCP 22):** Secure Shell Protocol sets up a secure session that’s similar to Telnet over a standard TCP/IP connection and is employed for doing things like logging into systems, running programs on remote systems and moving file from one system to another system. And it does all this while maintaining an encrypted connection.

**HTTP (TCP 80):** Hypertext Transfer Protocol is used to manage communications between web browsers and web servers and opens the right resources when you click a link, wherever that resource may reside.

**HTTPS (TCP 443):** Hypertext Transfer Protocol Secure is also known as a secure HTTP. It uses the Secure Socket Layer (SSL). It is the secure version of the HTTP that provides some security tools for keeping transactions between web browsers and servers secure. It is what our browser needs to fill out forms, sign in, authenticate, and encrypt an HTTP message when we do things like making an online reservation, accessing online banking, or buying something online.

**NTP (UDP 123):** Network Time Protocol is used to synchronize the clocks on our computer to one standard time source. This protocol works by synchronizing devices to ensure that all the computers on a given network agree on the time.

**LDAP (TCP 389):** The Lightweight Directory Access Protocol, is a mature, flexible, and well supported standards-based mechanism for interacting with directory servers. It’s often used for authentication and storing information about users, groups, and applications, but an LDAP directory server is a fairly general-purpose data store and can be used in a wide variety of applications.

**IGMP:** Internet Group Management Protocol is the TCP/IP protocol used for managing IP multicast sessions. It accomplishes this by sending out unique IGMP messages over the network to reveal the multicast-group landscape and to find out which hosts belong to which multicast group. The host machines in an IP network also use IGMP messages to become members of a group and to quit the group, too. IGMP messages come in seriously handy for tracking group memberships as well as active multicast streams. IGMP works at the Network layer and doesn’t use port numbers.

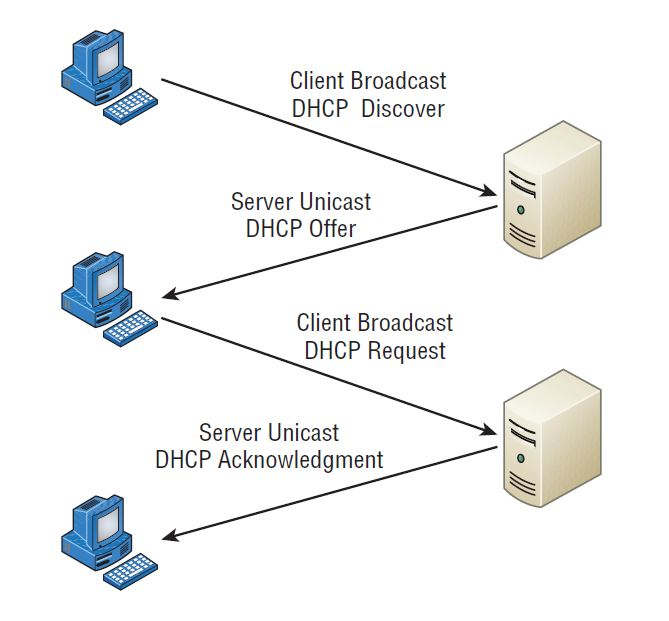
**NetBIOS (TCP and UDP 137–139):** Network Basic Input/Output System works only in the upper layers of the OSI model and allows for an interface on separate computers to communicate over a network. It was first created in the early 1980s to work on an IBM LAN and was proprietary. Microsoft and Novell both created a NetBIOS implementation to allow their hosts to communicate to their servers, but Microsoft’s version became the de facto version.

**SMB (TCP 445):** Server Message Block is used for sharing access to files and printers and other communications between hosts on a Microsoft Windows network. SMB can run on UDP port 137 and 138 and TCP port 137 and 139 using NetBIOS.

**DNS (TCP and UDP 53):** Domain Name Service resolves hostnames, internet names such as www.clarusway.com (Fully Qualified Domain Name). A *Fully Qualified Domain Name (FQDN)* is a hierarchy that can logically locate a system based on its domain identifier. An IP address identifies hosts on a network and the Internet as well, but DNS was designed to make our lives easier. Think about this: What would happen if you wanted to move your web page to a different service provider? The IP address would change and no one would know what the new one was. DNS allows you to use a domain name to specify an IP address. You can change the IP address as often as you want and no one will know the difference.

**DHCP (UDP 67/68):** Dynamic Host Configuration Protocol assigns IP Address to hosts. It allows for easier administration and works well in small to very large network environments. Many types of hardware can be used as a DHCP Server, including a Cisco Router. There is a lot of information a DHCP server can provide to a host when the host is requesting an IP address from DHCP Server, here is the list of some common types of information a DHCP server can provide:

* IP Address
* Subnet Mask
* Domain Name
* Default Gateways
* DNS Server Address
* WINS Server Address

**The DORA process**

The following is the four-step process (sometimes known as the DORA process) a client takes to receive an IP address from a DHCP server:

1. The DHCP client broadcasts a DHCP Discover message looking for a DHCP server (port 67).
2. The DHCP server that received the DHCP Discover message sends a unicast DHCP Offer message back to the host.
3. The client then broadcasts to the server a DHCP Request message asking for the offered IP address and possibly other information.
4. The server finalizes the exchange with a unicast DHCP Acknowledgment message.

**The Host-to-Host Layer Protocols - TCP**

The main purpose of the **Host-to-Host layer** is to *shield the upper-layer applications* from the complexities of the network. This layer says to the upper layer, “Just give me your data stream, with any instructions, and I’ll begin the process of getting your information ready to send.”

The following sections describe the two protocols at this layer:

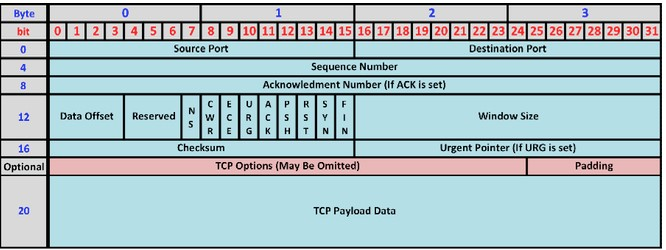
* Transmission Control Protocol (TCP)
* User Datagram Protocol (UDP)

**Transmission Control Protocol (TCP)** takes large blocks of information from an application and breaks them into segments. It numbers and sequences each segment so that the destination’s TCP process can put the segments back into the order the application intended. After these segments are sent, TCP (on the transmitting host) waits for an acknowledgment from the receiving end’s TCP process, retransmitting those segments that aren’t acknowledged.

Remember that in reliable transport operation, a device that wants to transmit sets up a connection-oriented communication with a remote device by creating a session. The transmitting device first establishes a connection-oriented session with its peer system; that session is called a **call setup** or a **three-way handshake**. Data is then transferred, and when the transfer is complete, a call termination takes place to tear down the virtual circuit.

TCP is a *full-duplex, connection-oriented, reliable*, and *accurate protocol*, but establishing all these terms and conditions, in addition to error checking, is no small task. TCP is very complicated and costly in terms of network overhead.

When the Internet layer receives the data stream, it routes the segments as packets through an internetwork. The segments are handed to the receiving host’s Host-to-Host layer protocol, which rebuilds the data stream to hand to the upper-layer protocols. The below figure shows the TCP segment format.



The TCP Segment

**The Host-to-Host Layer Protocols - UDP**

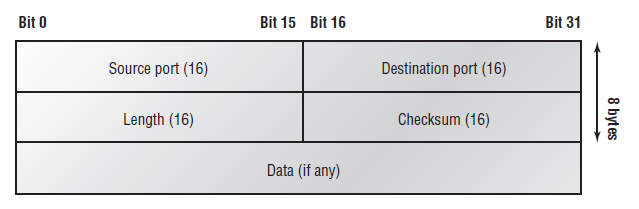
If to compare **User Datagram Protocol (UDP)** with TCP, the former is basically the scaled-down economy model that’s sometimes referred to as a thin protocol. A thin protocol doesn’t take up much bandwidth on a network. UDP doesn’t offer all the advantages of TCP either, but it does do a fabulous job of transporting information that doesn’t require reliable delivery—and it does so using far fewer network resources.

There are some situations in which it would definitely be wise for developers to opt for UDP rather than TCP. Remember the watchdog SNMP up there at the Process/Application layer. SNMP monitors the network, sending intermittent messages and a fairly steady flow of status updates and alerts, especially when running on a large network. The cost in overhead to establish, maintain, and close a TCP connection for each one of those little messages would reduce what would be an otherwise healthy, efficient network.

Another circumstance calling for UDP over TCP is when reliability is already handled at the Process/Application layer. DNS handles its own reliability issues, making the use of TCP both impractical and redundant. But ultimately, it’s up to the *application developer* to decide whether to use UDP or TCP, *not the user* who wants to transfer data faster.

UDP does not sequence the segments and doesn’t care in which order the segments arrive at the destination. But after that, UDP sends the segments off and forgets about them. It doesn’t follow through, check up on them, or even allow for an acknowledgment of safe arrival—complete abandonment. Because of this, it’s referred to as an *unreliable* protocol. This doesn’t mean that UDP is ineffective, only that it doesn’t handle issues of reliability. Because UDP assumes that the application will use its own reliability method, it doesn’t use any. This gives an application developer a choice when running the IP stack: **TCP for reliability** or **UDP for faster transfers**.

Further, UDP doesn’t create a virtual circuit, nor does it contact the destination before delivering information to it. Because of this, it’s also considered a **connectionless** protocol.

The UDP Segment

**Key features of TCP and UDP**

| **TCP** | **UDP** |
| --- | --- |
| Sequenced | Unsequenced |
| Secure | Unsecure |
| Connection-oriented | Connectionless |
| Slow | Fast |
| Guaranteed transmission | No guarantee |
| Flow control | No flow control |
| Reliable | Unreliable |
| Virtual circuit | No virtual circuit |
| High overhead | Low overhead |
| Acknowledgment | No acknowledgment |
| Windowing flow control | No windowing or flow control |
| 20 bytes header | 8 bytes header |

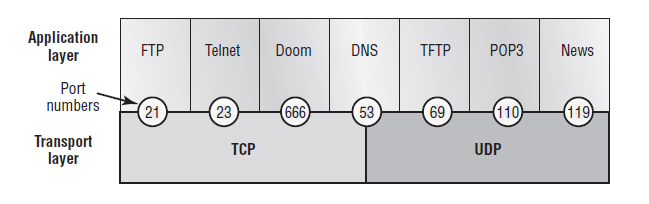
The below table highlights some of the key concepts that you should keep in mind regarding these two protocols.

A telephone analogy could really help you understand how TCP works. Most of us know that before you speak to someone on a phone, you must first establish a connection with that person—wherever they are. This is like a virtual circuit with TCP. If you were giving someone important information during your conversation, you might say, “You know?” or ask, “Did you get that?” Saying something like this is a lot like a TCP acknowledgment—it’s designed to get your verification. From time to time (especially on cell phones), people also ask, “Are you still there?” They end their conversations with a “Good-bye” of some kind, putting closure on the phone call. TCP also performs these types of functions.

Alternatively, using UDP is like sending a postcard. To do that, you don’t need to contact the other party first. You simply write your message, address the postcard, and mail it. This is analogous to UDP’s connectionless orientation. Because the message on the postcard is probably not a matter of life or death, you don’t need an acknowledgment of its receipt. Similarly, UDP doesn’t involve acknowledgments.

TCP and UDP must use **port numbers** to communicate with the upper layers because they’re what keep track of different simultaneous conversations originated by or accepted by the localhost. Originating source port numbers are dynamically assigned by the source host and will usually have a value of 1024 or higher. Virtual circuits that don’t use an application with a well-known port number are assigned port numbers randomly from a specific range instead. These port numbers identify the source and destination application or process in the TCP segment.

The below figure illustrates how both TCP and UDP use port numbers.

Port numbers for TCP and UDP

You just need to remember that numbers below 1024 are considered the well-known port numbers and are defined in RFC 3232. Numbers 1024 and above are used by the upper layers to set up sessions with other hosts and by TCP as the source and destination identifiers in the TCP segment.

The below table gives you a list of the typical applications used in the TCP/IP suite, their well-known port numbers, and the Transport layer protocols used by each application or process.

| **TCP** | **UDP** |
| --- | --- |
| Telnet 23 | SNMPv1/2 161 |
| SMTP 25 | TFTP 69 |
| HTTP 80 | DNS 53 |
| FTP 20, 21 | BOOTPS/DHCP 67,68 |
| SFTP 22 | NTP 123 |
| DNS 53 |  |
| HTTPS 443 |  |
| SSH 22 |  |
| SMB 445 |  |
| POP3 110 |  |
| IMAP4 143 |  |
| RDP 3389 |  |
| SNMPv3 161 |  |

**The Internet Layer Protocols**

In the DoD model, there are two main reasons for the Internet layer’s existence: **routing** and **providing** *a single network interface* to the upper layers.

None of the other upper- or lower-layer protocols have any functions relating to routing—that complex and important task belongs entirely to the Internet layer. The Internet layer’s second duty is to provide a single network interface to the upper-layer protocols. Without this layer, application programmers would need to write what are called hooks into every one of their applications for each different Network Access protocol. This would lead to different versions of each application—one for Ethernet, another one for Token Ring, and so on. To prevent this, IP provides one single network interface for the upper-layer protocols.

All network depends on IP. And all the other protocols at this layer, as well as all those at the upper layers, use it. Never forget that. All paths through the DoD model go through IP. The following sections describe the protocols at the Internet layer:

* Internet Protocol (IP)
* Internet Control Message Protocol (ICMP)
* Address Resolution Protocol (ARP)
* Reverse Address Resolution Protocol (RARP)

**Internet Protocol**

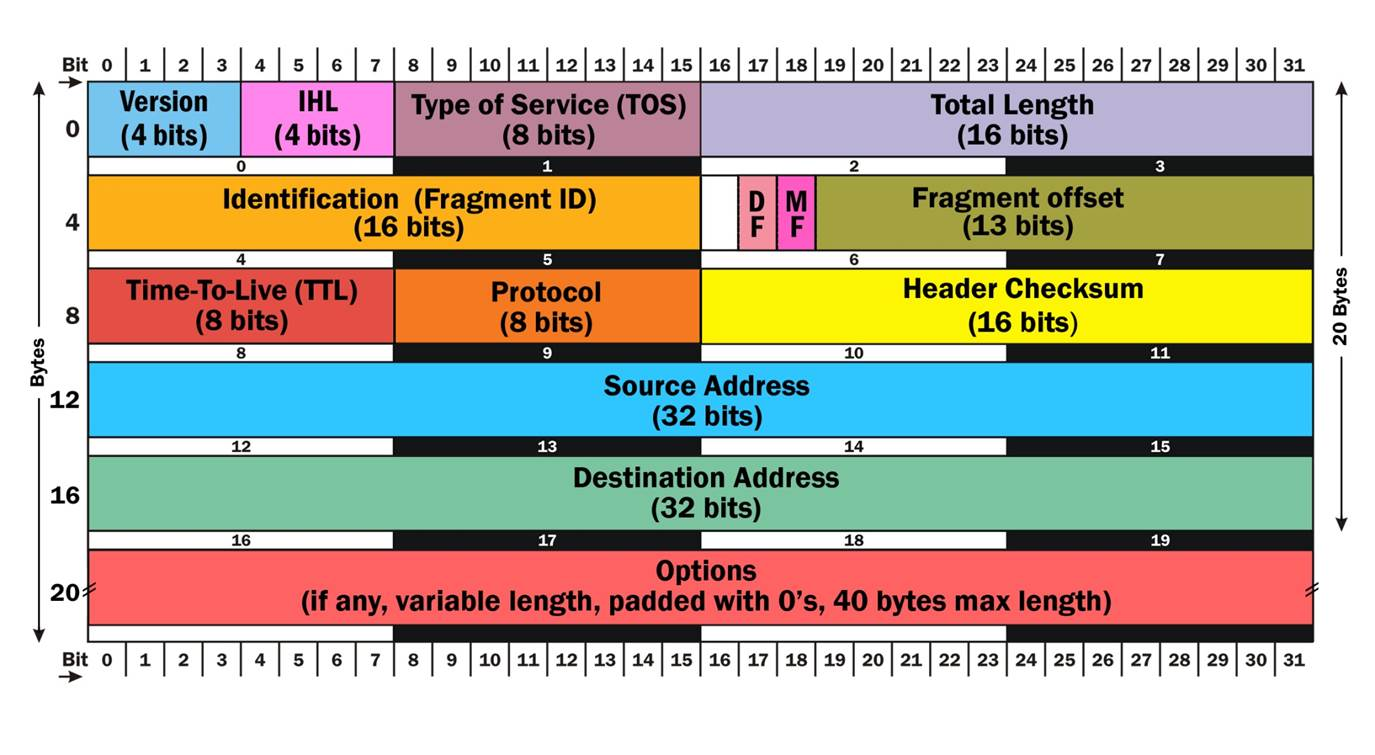
**Internet Protocol (IP)** is essentially the *Internet layer*. The other protocols at this layer simply exist to support it.

IP looks at each packet’s destination address. Then, using a routing table, it decides where a packet is to be sent next, choosing the best path. The protocols of the *Network Access layer* at the bottom of the *DoD model* don’t possess IP’s enlightened scope of the entire network; they deal only with physical links (local networks).

Identifying devices on networks requires answering these two questions: Which network is it on? And what is its ID on that network? The answer to the first question is the *software address*, or *logical address*. The answer to the second question is the hardware address. All hosts on a network have a **logical ID** called an **IP address**. This is the software -or logical- address and contains valuable encoded information, simplifying the complex task of routing.

IP receives segments from the *Host-to-Host layer* and fragments them into packets if necessary. IP then reassembles packets back into segments on the receiving side. Each packet is assigned the IP address of the sender and of the recipient. Each router (Layer 3 device) that receives a packet makes routing decisions based on the packet’s destination IP address.

The below figure is an IPv4 header.

IPv4 header

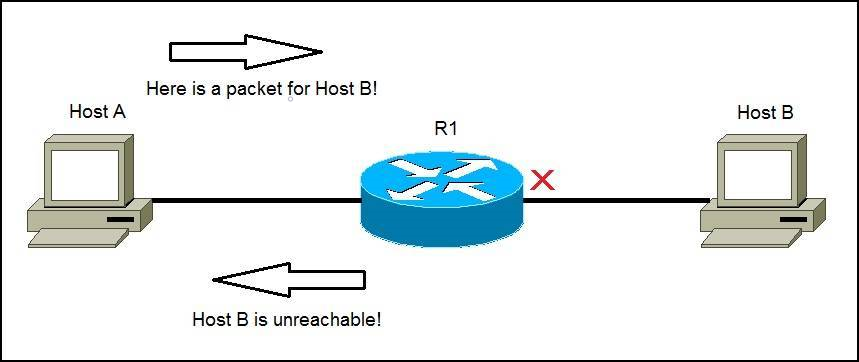
**Internet Control Message Protocol**

**Internet Control Message Protocol (ICMP)** works at the *Network layer* and is used by IP for many different services. ICMP is a management protocol and messaging service provider for IP. Its messages are carried as IP packets. ICMP packets have the following characteristics:

* They can provide hosts with information about network problems.
* They are encapsulated within IP datagrams.

The following are some common events and messages that ICMP relates to and the two most popular programs that use ICMP:

* **Destination Unreachable** - If a router can’t send an IP datagram any further, it uses ICMP to send a message back to the sender, advising it of the situation. For example, the below figure shows that the Ethernet interface of the Lab B router is down.

ICMP error message is sent to the sending host from the remote router.

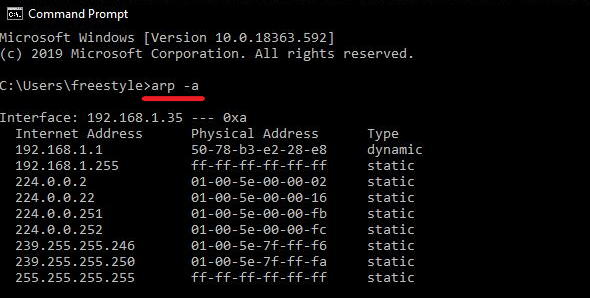
When Host A sends a packet destined for Host B, the Lab B router will send an *ICMP Destination Unreachable* message back to the sending device.

* **Buffer Full** - If a router’s memory buffer for receiving incoming datagrams is full, it will use ICMP to send out this message until the congestion abates.
* **Hops** - Each IP datagram is allotted a certain number of routers, called hops, to pass through. If a datagram reaches its limit of hops before arriving at its destination, the last router to receive it deletes it. The executioner router then uses ICMP to send an obituary (mortal) message, informing the sending machine of the demise (drop) of its datagram.
* **Ping** - Ping uses ICMP echo request and reply messages to check the physical and logical connectivity of machines on an internetwork.
* **Traceroute** - Traceroute uses IP packet Time to Live time-outs to discover the path a packet takes as it traverses an internetwork.

**Address Resolution Protocol (ARP)**

**Address Resolution Protocol (ARP)** is a procedure for mapping a dynamic **Internet Protocol address (IP address)** to a permanent physical **machine address** in a local area network (LAN). The physical machine address is also known as a **Media Access Control** or **MAC address**.

The job of the ARP is essentially to translate 32-bit addresses to 48-bit addresses and vice-versa. This is necessary because in IP Version 4 (IPv4), the most common level of Internet Protocol (IP) in use today, an IP address is 32-bits long, but MAC addresses are 48-bits long.

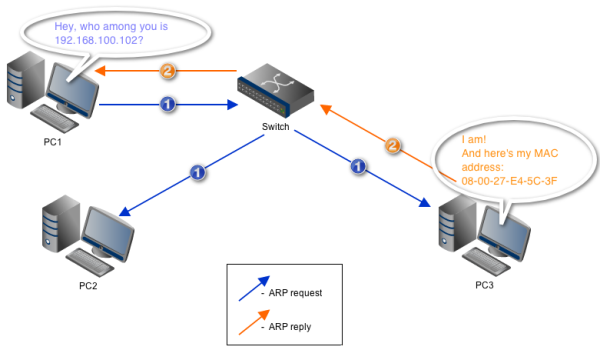


ARP table

ARP works between network layers 2 and 3 of the Open Systems Interconnection model (OSI model). The MAC address exists on layer 2 of the OSI model, the data link layer, while the IP address exists on layer 3, the network layer.

In IPv6, which uses 128-bit addresses, ARP has been replaced by the **Neighbor Discovery protocol**.

When a new computer joins a LAN, it is assigned a unique IP address to use for identification and communication. When an incoming packet destined for a host machine on a particular LAN arrives at a gateway, the gateway asks the ARP program to find a MAC address that matches the IP address. A table called the ARP cache maintains a record of each IP address and its corresponding MAC address.

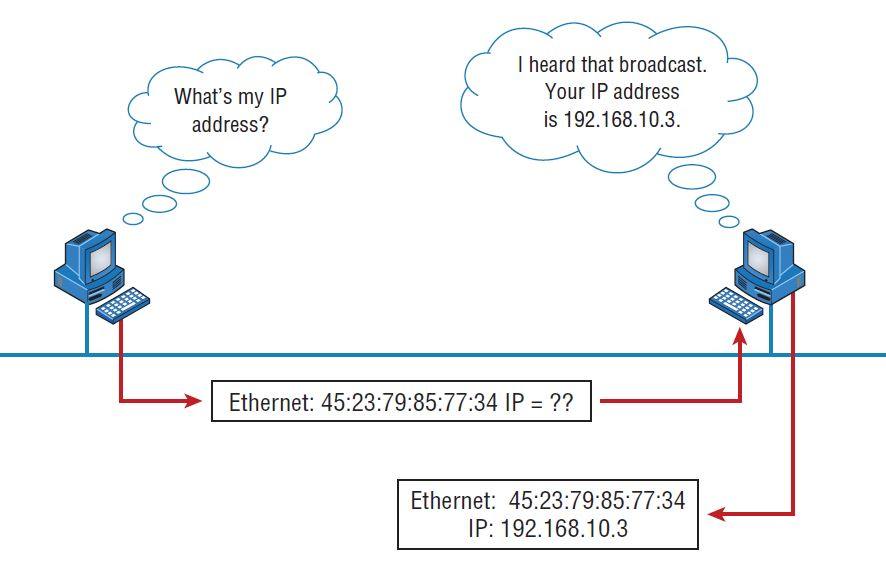


ARP query

All operating systems in an IPv4 Ethernet network keep an ARP cache. Every time a host requests a MAC address in order to send a packet to another host in the LAN, it checks its ARP cache to see if the IP to MAC address translation already exists. If it does, then a new ARP request is unnecessary. If the translation does not already exist, then the request for network addresses is sent and ARP is performed.

ARP broadcasts a request packet to all the machines on the LAN and asks if any of the machines know they are using that particular IP address. When a machine recognizes the IP address as its own, it sends a reply so ARP can update the cache for future reference and proceed with the communication.

Host machines that don't know their own IP address can use the **Reverse ARP (RARP)** protocol for discovery.

RARP broadcast example

An ARP cache size is limited and is periodically cleansed of all entries to free up space; in fact, addresses tend to stay in the cache for only a few minutes. Frequent updates allow other devices in the network to see when a physical host changes their requested IP address. In the cleaning process, unused entries are deleted as well as any unsuccessful attempts to communicate with computers that are not currently powered on.

**IP Terminology**

Throughout this section, you’ll learn several important terms vital to your understanding of the **Internet Protocol**. Here are a few to get you started:

* **Bit** - A bit is one binary digit, either a 1 or a 0.
* **Nibble** - A nibble is 4 bits.
* **Byte** - A byte is 8 bits.
* **Octet** - An octet, made up of 8 bits, is just an ordinary 8-bit binary number. In this section, the terms byte and octet are completely interchangeable.
* **Network Address** - This is the designation used in routing to send packets to a remote network—for example, 10.0.0.0, 172.16.0.0, and 192.168.10.0.
* **IP Address** - A logical address used to define a single host; however, IP addresses can be used to reference many or all hosts as well. If you see something written as just IP, it is referring to IPv4. IPv6 will always be written as IPv6.
* **Broadcast Address** - The broadcast address is used by applications and hosts to send information to all hosts on a network. Examples include 255.255.255.255, which designates all networks and all hosts; 172.16.255.255, which specifies all subnets and hosts on network 172.16.0.0; and 10.255.255.255, which broadcasts to all subnets and hosts on network 10.0.0.0.

**IPv4 Address Types**

**IPv4 Address Types**

When a DHCP client broadcasted for an IP address; a router then forwarded this as a unicast packet to the DHCP server. With IPv4, broadcasts are pretty important, but with IPv6, there aren’t any broadcasts sent at all.

Broadcast addresses were referred to in the earlier chapters. Here are the four IPv4 address types:

* **Layer 2 Broadcasts** - These are sent to all nodes on a LAN.
* **Broadcasts (Layer 3)** - These are sent to all nodes on the network.
* **Unicast** - This is an address for a single interface, and these are used to send packets to a single destination host.
* **Multicast** - These are packets sent from a single source and transmitted to many devices on different networks. Referred to as one-to-many.

**Layer 2 Broadcasts**

Layer 2 broadcasts are also known as hardware broadcasts—they only go out on a LAN, and they don’t go past the LAN boundary (router). The typical hardware address (MAC address) is 6 bytes (48 bits) and looks something like 0c.43.a4.f3.12.c2. The broadcast would be all 1s in binary, which would be all Fs in hexadecimal, as in FF.FF.FF.FF.FF.FF.

**Layer 3 Broadcasts**

Broadcast messages are meant to reach all hosts on a broadcast domain. These are the network broadcasts that have all host bits on.

Here’s an example: The network address of 172.16.0.0 would have a broadcast address of 172.16.255.255—all host bits on. Broadcasts can also be “any network and all hosts,” as indicated by 255.255.255.255.

A good example of a broadcast message is an Address Resolution Protocol (ARP) request. When a host has a packet, it knows the logical address (IP) of the destination. To get the packet to the destination, the host needs to forward the packet to a default gateway if the destination resides on a different network. If the destination is on the local network, the source will forward the packet directly to the destination. Because the source doesn’t have the MAC address to which it needs to forward the frame, it sends out a broadcast, something that every device in the local broadcast domain will listen to. This broadcast says, in essence, “*If you are the owner of IP address 192.168.2.3, please forward your MAC address to me*” with the source giving the appropriate information.

**Unicast Address**

A unicast address is assigned to a single host, and this term is used in both IPv4 and IPv6 to describe your host interface IP address.

**Multicast Address (Class D)**

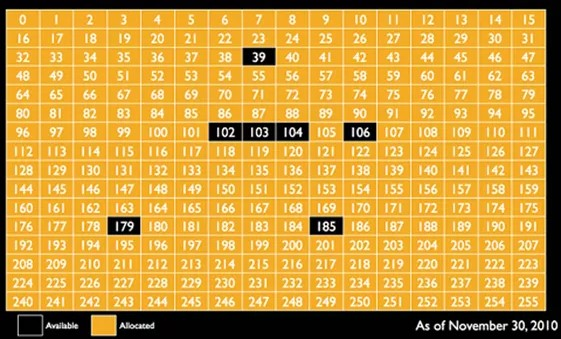
Multicast works by sending messages or data to multicast group addresses. Routers then forward copies (unlike broadcasts, which are not forwarded) of the packet out every interface that has hosts on a particular group address. This is where multicast differs from broadcast messages—with multicast communication, copies of packets, in theory, are sent only to subscribed hosts. This means that the hosts will receive, for example, a multicast packet destined for 224.0.0.10. All hosts on the broadcast LAN will pick up the frame, read the destination address, and immediately discard the frame unless they are in the multicast group. This saves PC processing, not LAN bandwidth. Multicasting can cause severe LAN congestion, in some instances, if not implemented carefully.

There are several different groups that users or applications can subscribe to. The range of multicast addresses starts with 224.0.0.0 and goes through 239.255.255.255. As you can see, this range of addresses falls within IP Class D address space based on a classful IP assignment.

Formun Üstü

Formun Altı

**Why Do We Need IPv6?**



**IPv4 Address Space Consumption**

The above picture is old already but it shows you the reason why we need IPv6. We are running out of IPv4 addresses!

So what happened to IPv4? We have 32-bits which gives us 4,294,467,295 IP addresses. Remember our Class A, B, and C ranges. When the Internet started you would get a Class A, B, or C network. Class C gives you a block of 256 IP addresses, class B is 65.535 IP addresses, and class A even 16,777,216 IP addresses. Large companies like Apple, Microsoft, IBM, and such got one or more Class A networks. Did they really need more than 16 million IP addresses? Many IP addresses were just wasted.

We started using **VLSM (Variable Length Subnet Mask)** so we could use any subnet mask we like and create smaller subnets, we had to use the class A, B or C networks for a while. We also started using **NAT** and **PAT** so we can have many private IP addresses behind a single public IP address. (Subnets, NAT, and PAT will be discussed later.)

Nevertheless, the Internet has grown in a way nobody expected 20 years ago. Despite all tricks like *VLSM* and *NAT/PAT* we really need more IP addresses and that’s why we need IPv6.

So what happened to IPv5? IP version 5 was used for an experimental project called *“Internet Stream Protocol”*.

Q: What do you mean by IPv6?  
A: IPv6 stands for Internet Protocol version 6 and is the latest version of the Internet Protocol. The IP address length is 128 bits which resolves the issue of approaching shortage of network addresses.

- Interview Q&A

**Internet Protocol Version 6 (IPv6)**

**The Benefits of and Uses for IPv6**

IPv6 has 128-bit addresses and has a much larger address space than 32-bit IPv4 which offered us a bit more than 4 billion addresses. Keep in mind every additional bit doubles the number of IP addresses…so we go from 4 billion to 8 billion, 16, 32, 64, etc. Keep doubling until you reach 128 bit. With 128 bits this is the largest value you can create:

340,282,366,920,938,463,463,374,607,431,768,211,456

This gives us enough IP addresses for networks on earth, the Moon, Mars, and the rest of the universe. To put this in perspective let’s put the entire IPv6 and IPv4 address space next to each other:

IPv6: 340282366920938463463374607431768211456

IPv4: 4294467295

Some other nice numbers: the entire IPv6 address space is 4294467295 times the size of the complete IPv4 address space. Or if you like percentages, the entire IPv4 address space is only 0.000000000000000000000000001,26% of the entire IPv6 address space.

The main reason to start using IPv6 is that we need more addresses but it also has some benefits:

* **More Efficient Routing** – IPv6 reduces the size of routing tables and makes routing more efficient and hierarchical. In IPv6 networks, fragmentation is handled by the source device, rather than a router.
* **More efficient packet processing** – Compared with the IPv4, IPv6 contains no IP-level checksum, so the checksum(sağlama toplamı) does not need to be recalculated at every router hop.
* **Directed Data Flows** – IPv6 supports multicast rather than broadcast. Multicast allows bandwidth-intensive packet flows to be sent to multiple destinations simultaneously, saving network bandwidth.
* **Simplified network configuration** – IPv6 devices can independently auto-configure themselves when connected to other IPv6 devices. Configuration tasks that can be carried out automatically include IP address assignment and device numbering.
* **Security** – IPSec security, which provides confidentiality, authentication, and data integrity, is engraved into IPv6.

**IPv6 Addressing and Expressions**

What does an IPv6 address look like? We use a different format than IPv4. We don’t use decimal numbers like for IPv4, we are using hexadecimal now. Here’s an example of an actual IPv6 address:

2041:1234:140F:1122:AB91:564F:875B:131B

As you can now see, the address is truly much larger and it has eight groups of numbers instead of four, also that those groups are separated by colons instead of periods. One other thing that should be pointed out is for when you set up your test network with IPv6, you have to type the address into the browser with brackets around the literal address. Because a colon is already being used by the browser for specifying a port number. So basically, if you don’t enclose the address in brackets, the browser will have no way to identify the information. Here’s an example of how this looks:

http://[2001:0db8:3c4d:0012:0000:0000:1234:56ab]/default.html

**Shortening IPv6 Addresses**

IPv6 addresses are hexadecimal and since they are 128-bit, they are quite long. Imagine you have to call a friend and ask him/her to ping the following address:

2041:0000:140F:0000:0000:0000:875B:131B

To make our lives a bit better, IPv6 addresses can be shortened. Let’s take a look at some examples and see how it works:

Original: 2041:0000:140F:0000:0000:0000:875B:131B

Short: 2041:0000:140F::875B:131B

If there is a string of zeros then you can remove them once. In the example above the entire 0000:0000:0000 part was removed. You can only do this once so you cannot do this:

Original: 2001:0000:0000:0012:0000:0000:1234:56ab

Wrong!: 2001::12::1234:56ab

There is more however, the address can be shortened even more:

Short: 2041:0000:140F::875B:131B

Shorter: 2041:0:140F::875B:131B

If you have a “*hextet*” with 4 zeros then you can remove those and leave a single zero. Your IPv6 device will add the remaining 3 zeros.

**💡Tip:**

* When we talk about IPv4 addresses, we use the term “octet” to define a “block” of 8 bits. In IPv6, there is no official term (yet) and there is an IETF draft that discusses the names to be used. The official term for 4 hexadecimal values is "hexadectet”, this is hard to remember/pronounce so the short form “hextet” will be used.

Leading zeros can also be removed, here’s another address to demonstrate this:

Original: 2001:0001:0002:0003:0004:0005:0006:0007

Short: 2001:1:2:3:4:5:6:7

By removing these zeros we get a nice short IPv6 address.

To summarize these rules:

* An entire string of zeros can be removed, you can only do this once.
* 4 zeros can be removed, leaving only a single zero.
* Leading zeros can be removed.

**Address Types**

IPv4 uses **unicast, broadcast**, and **multicast addresses**, which basically define how many other devices we’re communicating to. IPv6 introduces the **anycast address** type. Broadcasts, as we know them, have been eliminated in IPv6 because of their cumbersome(hantal) inefficiency.

* **Unicast Address**

Unicast addresses are same as in IPv4, packets are delivered to a single node. Unicast addresses can be divided into two categories:

1. **Global Unicast Addresses**

These are worldwide unique addresses that a network device needs in order to connect to the internet. The format prefix is usually 2000::/3 and includes all addresses that begin with 2000 to 3FFF. These types of addresses are routable and can be used to directly address a host in the local network over the internet. Global unicast addresses that are distributed to end-users begin with the hexadecimal block 2001.

1. **Link-Local Addresses**

Addresses in this category are only valid within local networks and begin with the format prefix FE80::/10. Local link addresses are used to address elements within a local network and are used, for example, for auto-configuration.

* **Multicast**

As in IPv4, packets addressed to a multicast address are delivered to all hosts identified by the multicast address. Sometimes it is called one-to-many addresses. It’s really easy to spot multicast addresses in IPv6 because they always start with FF.

* **Anycast**

Like multicast addresses, an anycast address identifies multiple hosts, but there’s a big difference: The anycast packet is delivered to only one address—actually, to the first IPv6 address it finds defined in terms of routing distance. And again, this address is special because you can apply a single address to more than one host. This is also referred to as *one-to-nearest addressing*.

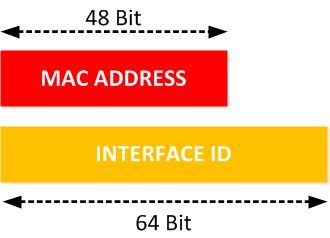
**Special Addresses**

The below table lists some of the addresses and address ranges that are all special or reserved for a specific use.

| **Address** | **Meaning** |
| --- | --- |
| 0:0:0:0:0:0:0:0 | Equals ::. This is the equivalent of IPv4’s 0.0.0.0 and is typically the source address of a host before the host receives an IP address when you’re using DHCP-driven stateful configuration. |
| 0:0:0:0:0:0:0:1 | Equals ::1. The equivalent of 127.0.0.1 in IPv4. 0::FFFF:192.168.100.1 This is how an IPv4 address would be written in a mixed IPv6/IPv4 network environment. |
| 2000::/3 | The global unicast address range allocated for Internet access. |
| FE80::/10 | The link-local unicast range. |
| FF00::/8 | The multicast range. |
| 3FFF:FFFF::/32 | Reserved for examples and documentation. |
| 2001:0DB8::/32 | Also reserved for examples and documentation. |
| 2002::/16 | Used with 6to4 tunneling, which is an IPv4-to-IPv6 transition system. |

**Stateless Autoconfiguration (EUI-64)**

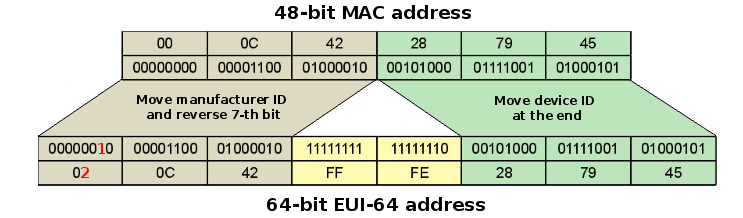
**EUI-64 (Extended Unique Identifier)** is a method we can use to automatically configure IPv6 host addresses. An IPv6 device will use the MAC address of its host to generate a unique 64-bit interface ID. However, a MAC address is 48 bit and the host ID is 64 bit. What are we going to do with the missing bits?



**Pv6 MAC address vs Interface ID**

Here’s what we will do to fill the missing bits:

1. We take the MAC address and split it into two pieces.
2. We insert “FFFE” in between the two pieces so that we have a 64-bit value.
3. We invert the 7th bit of the interface ID. So if the MAC address would be 00:0C:42:28:79:45 then this is what the interface ID will become:



**Stateless Autoconfiguration**

Here are a few examples:

* MAC address 0090:2716:fd0f
* IPv6 EUI-64 address: 2001:0db8:0:1:0290:27ff:fe16:fd0f

That one was easy! So let’s do another:

* MAC address aa12:bcbc:1234
* IPv6 EUI-64 address: 2001:0db8:0:1:a812:bcff:febc:1234

10101010 represents the first 8 bits of the MAC address (aa), which when inverting the 7th bit becomes 10101000. The answer becomes a8. This is important for you to understand.

* MAC address 0c0c:dede:1234
* IPv6 EUI-64 address: 2001:0db8:0:1:0e0c:deff:fede:1234

0c is 00001100 in the first 8 bits of the MAC address, which then becomes 00001110 when flipping the 7th bit. The answer is then 0e. Let’s practice one more:

* MAC address 0b34:ba12:1234
* IPv6 EUI-64 address: 2001:0db8:0:1:0934:baff:fe12:1234

0b in binary is 00001011, the first 8 bits of the MAC address, which then becomes 00001001. The answer is 09.

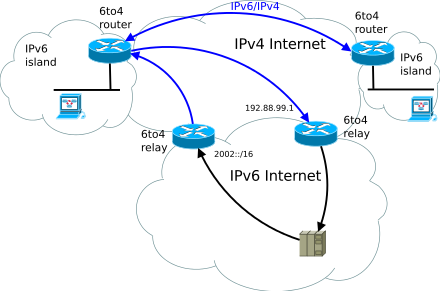
**Migrating to IPv6**

* **Dual Stacking**

This is the most common type of migration strategy because, well, it’s the easiest on us—it allows our devices to communicate using either IPv4 or IPv6. **Dual stacking** lets you upgrade your devices and applications on the network one at a time. As more and more hosts and devices on the network are upgraded, more of your communication will happen over IPv6, and after you’ve arrived—everything’s running on IPv6 and you get to remove all the old IPv4 protocol stacks you no longer need.

* **6to4 Tunneling**

**6to4 tunneling** is really useful for carrying IPv6 packets over a network that’s still running IPv4. It’s quite possible that you’ll have IPv6 subnets or other portions of your network that are all IPv6, and those networks will have to communicate with each other. The whole idea of tunneling isn’t a difficult concept, and creating tunnels really isn’t as hard as you might think. All it really comes down to is snatching(kapışma) the IPv6 packet that’s traveling across the network and sticking an IPv4 header onto the front of it.

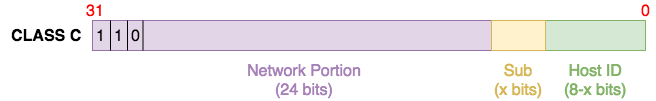


**A 6to4 tunnel**

**Subnetting Basics**

**Introduction to Subnets**

Internet designers come with a solution to the issue of IP address wastage: Subnetting. Subnetting allows us to create a smaller network (sub-networks or subnets) inside a large network by borrowing bits from the Host ID portion of the address. We can use those borrowed bits to create additional networks, resulting in smaller-sized networks.



**A Class C IP address**

Imagine that we want to build four networks that will support up to 30 devices in different segments. Without subnetting, we will need four Class C IP addresses to support this design. For example:

Network #1: 192.168.1.0

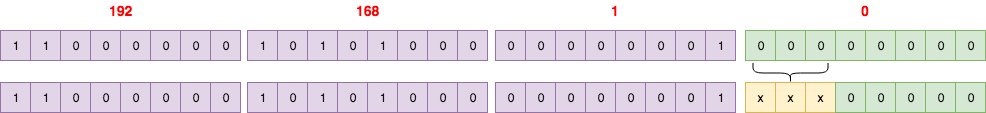
Network #2: 192.168.2.0

Network #3: 192.168.3.0

Network #4: 192.168.4.0

With four Class C IP addresses we can subscribe 254 \* 4 = 1016 hosts. But we have only 30 \* 4 = 120 hosts. This means 1016 - 120 = 896 IP addresses will be wasted!

If you look at the design requirement of 30 hosts per network, you will see that 5 bits are enough to subscribe to 30 hosts in each network. And this means we still have 3 bits unused and with subnetting, we can use those three bits to create smaller networks. For this example, let’s take the 192.168.1.0 network:

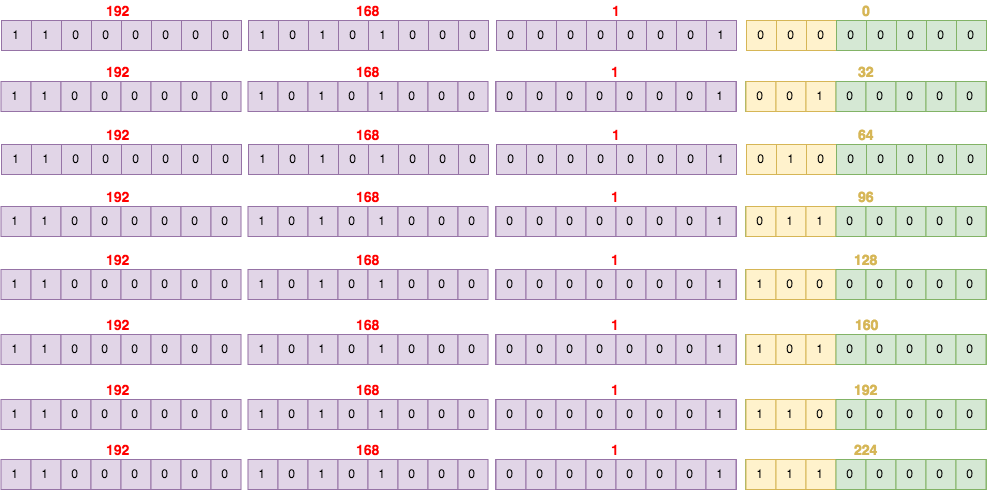


**Borrowing bits from host segment of the IP address**

By borrowing 3 bits, we can create 8 (23) subnets:

1. 192.168.1.0
2. 192.168.1.32
3. 192.168.1.64
4. 192.168.1.96
5. 192.168.1.128
6. 192.168.1.160
7. 192.168.1.192
8. 192.168.1.224

These subnet addresses look like normal IP addresses. However, looking at them in their binary form makes things clearer:



**Subnet IP addresses**

With subnetting, not only have we used only one Class C network, we have created 8 subnets from that network, each one supporting up to 30 hosts! We can use 4 of these subnets for our network and reserve the remaining 4 subnets for future expansion. This results in great waste reduction – from 896 wasted IP addresses to 120 reserved IP addresses.

Why do we need subnetting?

* **Reduce wastage:** As we have already seen, subnetting (and CIDR on a larger scale) helps us conserve IP addresses. While this is very important on the Internet (conserving public IP addresses), it is also useful on local area networks (LANs) where private IP addresses are used.
* **Improve Network Performance:** Subnetting improves the overall performance of a network. The larger a network is, the busier (more congested) it is. Consider the example of broadcasts – every host within an individual network will receive a broadcast even when it is not meant for them. This can affect performance especially during issues like broadcast storms. Therefore, the smaller the network, the more you can contain such issues within the subnet.
* **Isolation:** With smaller networks, you are able to isolate effectively as faults inside one subnet will not necessarily spread into other subnets. This is also important during security incidents so that even if one subnet is affected, the entire network is not brought down.
* **Easier administration:** Subnetting, when done properly, can make network administration more effective. For example, a multinational organization can design its network in such a way that each region is assigned an IP address block from a larger address block, and subnetting is used within regions to further divide the blocks among networks. This kind of design also improves routing as the routers in one region only need to know the “summarized” IP address block for other regions rather than all the smaller IP address blocks. This reduces the size of the routing table and ensures that fluctuations in one region do not affect the entire network

**Subnet Masks**

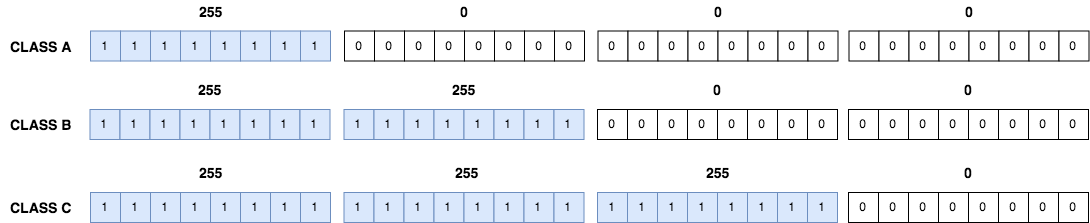
With what we have done, we have created a problem for computers and other networking devices: how are they supposed to differentiate between a subnet 192.168.1.32 and an IP address 192.168.1.32? This is where **subnet masks** (also called network masks) come in. A **subnet mask** is the representation of the network portion of an address. It is also made up of 32 bits with all the bits that represent the network portion being marked as **1s** and the other parts marked as **0s**.

For example, the default subnet masks of the IP address classes are:

Class A: 255.0.0.0

Class B: 255.255.0.0

Class C: 255.255.255.0

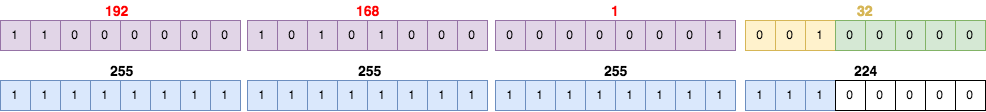


Therefore, a Class C network of 192.168.1.0 can be represented as 192.168.1.0 255.255.255.0.

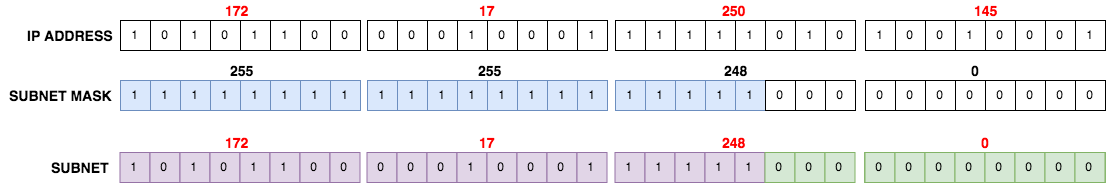
**💡Tip:**

* It can also be represented using prefix length (CIDR) notation where only the 1s that make up the network portion are counted and represented with a slash e.g. 192.168.1.0/24. We will talk about CIDR in the following lessons.

With subnetting, the borrowed bits from the host ID are counted as part of the network bits. So if we revisit our example above again, the 192.168.1.32 subnet can be represented as 192.168.1.32 255.255.255.224 (or 192.168.1.32/27).



By comparing (logical AND) the “turned on” bits (i.e. 1s) in the subnet mask to an IP address, a network device can determine what network a particular IP address belongs to. For example, the 172.17.250.145 IP address with a subnet mask of 255.255.248.0 belongs to the 172.17.248.0 255.255.248.0 subnet:



**Classless Inter-Domain Routing (CIDR)**

In order to reduce the wastage of IP addresses, a new concept of **Classless Inter-Domain Routing (CIDR)** is introduced.

When you receive a block of addresses from an ISP (Internet Service Provider), what you get will look something like this: 192.168.10.32/28. This is telling you what your subnet mask is. The slash notation **(/)** means how many bits are turned on (1s). Obviously, the maximum could only be **/32** because a byte is 8 bits and there are 4 bytes in an IP address: 4 × 8 = 32. But keep in mind that the largest subnet mask available (regardless of the class of address) can only be a **/30** because you have to keep at least 2 bits for host bits.

Take, for example, a *Class A* default subnet mask, which is 255.0.0.0. This means that the first byte of the subnet mask is all ones (1s), or 11111111. When referring to a slash notation, you need to count all the 1s bits to figure out your mask. The 255.0.0.0 is considered a **/8** because it has 8 bits that are 1s—that is, 8 bits that are turned on.

A *Class B* default mask would be 255.255.0.0, which is a **/16** because 16 bits are (1s):

11111111.11111111.00000000.00000000.

**Rules for forming CIDR Blocks:**

1. All IP addresses must be contiguous(bitişik).
2. Block size must be the power of 2 (2n). If the size of the block is the power of 2, then it will be easy to divide the Network. Finding out the Block Id is very easy if the block size is of the power of 2.
3. The first IP address of the Block must be evenly divisible by the size of the block. In simple words, the least significant part should always start with zeroes in Host Id. Since all the least significant bits of Host Id is zero, then we can use it as Block Id part.

**Example:**

Check whether 100.1.2.32 to 100.1.2.47 is a valid IP address block or not?

* All the IP addresses are contiguous.
* Total number of IP addresses in the Block = 16 = 24.
* 1st IP address: 100.1.2.00100000

Since Host Id will contain the last 4 bits and all the least significant 4 bits are zero. Hence, the first IP address is evenly divisible by the size of the block. All three rules are followed by this Block. Hence, it is a valid IP address block.

**Variable Length Subnet Masks (VLSM)**

So far, we have used subnetting to create fixed-size subnets e.g. four /26 subnets from one /24 block. However, the use of subnet masks and prefix lengths provides more flexibility – we can create subnets of varying sizes from the same address block i.e. VLSM.

Let us consider the following example. We are given a block of 172.16.1.0/24 and we need to split it such that the following requirements are met:

* A subnet that can accommodate(karşılamak) 100 hosts
* A subnet that can accommodate up to 55 hosts
* Two subnets that can accommodate up to 12 hosts each

To solve this problem, start with the biggest block and keep going down. For example, we need a minimum subnet of /25 to accommodate 100 hosts. (/25 means we have 7 bits left for the host ID.) Therefore, we can split the 172.16.1.0/24 block into two subnets:

172.16.1.0/25

172.16.1.128/25

We can use the first subnet 172.16.1.0/25 for the 100 hosts leaving us with the other subnet, 172.16.1.128/25.

The next largest subnet needs 55 hosts which can be accommodated with a /26 subnet. This means we can split the 172.16.1.128/25 subnet into two smaller subnets:

172.16.1.128/26

172.16.1.192/26

We can use the 172.16.1.128/26 subnet for the network requiring 55 hosts leaving us with the 172.16.1.192/26 subnet to further break down.

The two other networks require 12 hosts meaning we need a minimum of /28 subnets. Therefore, we can split the 172.16.1.192/26 subnet into 4 smaller subnets:

172.16.1.192/28

172.16.1.208/28

172.16.1.224/28

172.16.1.240/28

Therefore, our subnets are:

* 172.16.1.0/25 for the network with 100 hosts
* 172.16.1.128/26 for the network with 55 hosts
* 172.16.1.192/28 for the first network with 12 hosts
* 172.16.1.208/28 for the second network with 12 hosts

This means we still have two subnets (172.16.1.224/28 and 172.16.1.240/28) to use in the future.

**Subnetting Example-1**

255.255.128.0 (/17)

Let’s take a look at our first example:

172.16.0.0 = Network address

255.255.128.0 = Subnet mask

* **Subnets?**

21 = 2 (same as Class C).

* **Hosts?**

215 – 2 = 32,766 (7 bits in the third octet, and 8 in the fourth).

* **Valid subnets?**

256 – 128 = 128. 0, 128. Remember that subnetting in Class B starts in the third octet, so the subnet numbers are really 0.0 and 128.0, as shown in the next table. These are the exact numbers we used with Class C; we use them in the third octet and add a 0 in the fourth octet for the network address. To find the host addresses easily, initially write the subnet address and the broadcast address. After that, host addresses are between them and more clear to see.

* **Broadcast address for each subnet?**

The following table shows the two subnets available, the valid host range, and the broadcast address of each:

|  |  |  |
| --- | --- | --- |
| Subnet (do first) | 0.0 | 128.0 |
| First host (host addressing last) | 0.1 | 128.1 |
| Last host | 127.254 | 255.254 |
| Broadcast (do second) | 127.255 | 255.255 |

Notice that we just added the fourth octet’s lowest and highest values and came up with the answers. And again, it’s done exactly the same way as for a *Class C* subnet. We just use the same numbers in the third octet and added 0 and 255 in the fourth octet.

255.255.255.224 (/27)

This time, we’ll subnet the network address 192.168.10.0 and subnet mask 255.255.255.224.

192.168.10.0 = Network address

255.255.255.224 = Subnet mask

* **How many subnets?**

224 is 11100000, so our equation is 23 = 8.

* **How many hosts?**

25 – 2 = 30.

* **What are the valid subnets?**

256 – 224 = 32. We just start at zero and count to the subnet mask value in blocks (increments) of 32: 0, 32, 64, 96, 128, 160, 192, and 224.

* **What’s the broadcast address for each subnet (always the number right before the next subnet)?**
* **What are the valid hosts (the numbers between the subnet number and the broadcast address)?**

To answer the last two questions, first, just write out the subnets, and then write out the broadcast address—the number right before the next subnet. Last, fill in the host address. The following table gives you all the subnets for the 255.255.255.224 Class C subnet mask:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| The subnet address | 0 | 32 | 64 | 96 | 128 | 160 | 192 | 224 |
| The first valid host | 1 | 33 | 65 | 97 | 129 | 161 | 193 | 225 |
| The last valid host | 30 | 62 | 94 | 126 | 158 | 190 | 222 | 254 |
| The broadcast address | 31 | 63 | 95 | 127 | 159 | 191 | 223 | 255 |

**Network Address Translation (NAT)**

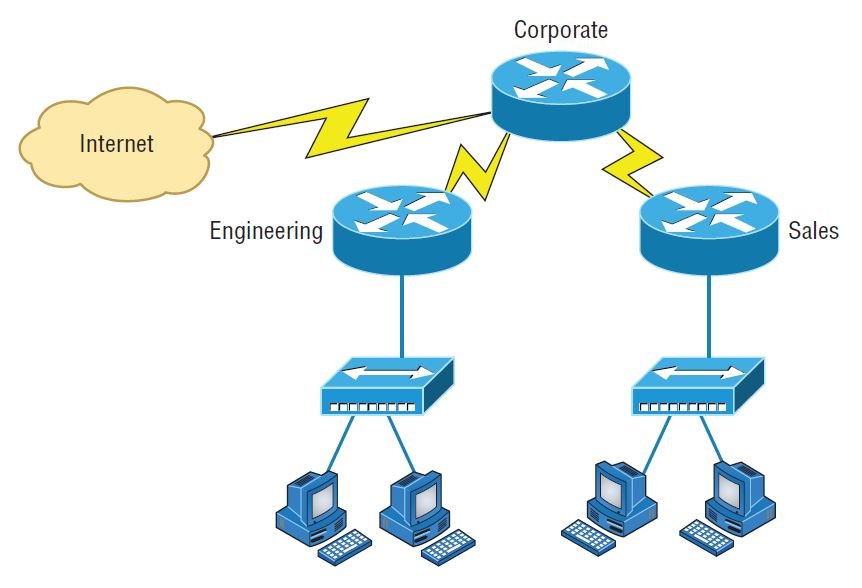
Similar to **Classless Inter-Domain Routing (CIDR)**, the original intention for **NAT** was to *slow the depletion(azaltma, tüketme) of available IP address space* by allowing many private IP addresses to be represented by some smaller number of public IP addresses. Since then, it’s been discovered that NAT is also a useful tool for network migrations and mergers, server load sharing, and creating “virtual servers.”

At times, NAT really decreases the overwhelming amount of public IP addresses required in your networking environment. And NAT comes in very handy when two companies that have duplicate internal addressing schemes merge. NAT is also great to have around when an organization changes its ISP and the networking manager doesn’t want the hassle(zorluk) of changing the internal address scheme.

Here’s a list of situations when it’s best to have NAT on your side:

* You need to connect to the Internet and your hosts don’t have globally unique IP addresses.
* You change to a new ISP that requires you to renumber your network.
* You need to merge two intranets with duplicate addresses.

You typically use NAT on a border router. See the below figure, where NAT would be configured on the Corporate router.



**Where to configure NAT**

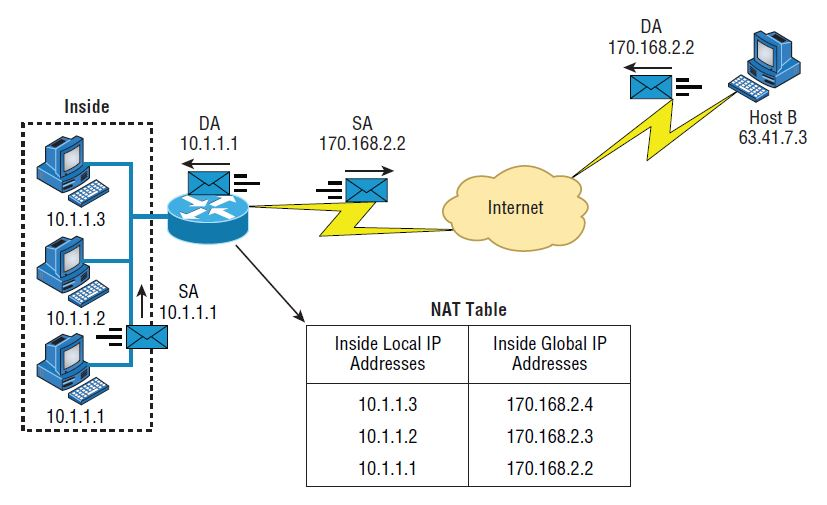
While NAT has many advantages, it also has disadvantages too.

| **Advantages** | **Disadvantages** |
| --- | --- |
| Conserves legally registered addresses. | Translation introduces switching path delays. |
| Reduces address overlap occurrences. | Loss of end-to-end IP traceability. |
| Increases flexibility when connecting to the Internet. | Certain applications will not function with NAT enabled. |
| Eliminates address renumbering as the network changes. |  |

**Types of NAT**

* **Static NAT (SNAT)** - This type of NAT is designed to allow one-to-one mapping between local and global addresses. Keep in mind that the static version requires you to have one real Internet IP address for every host on your network.
* **Dynamic NAT (DNAT)** - This version gives you the ability to map an unregistered IP address to a registered IP address from a pool of registered IP addresses. You don’t have to statically configure your router to map an inside-to-an-outside address as you would using static NAT, but you do have to have enough real, bona fide IP addresses for everyone who’s going to be sending packets to and receiving them from the Internet.
* **Overloading** - This is the most popular type of NAT configuration. Understand that overloading really is a form of dynamic NAT that maps multiple unregistered IP addresses to a single registered IP address—many-to-one—by using different ports. It’s also known as **Port Address Translation (PAT)**. And by using PAT (NAT Overload), you get to have thousands of users connect to the Internet using only one real global IP address. NAT Overload is the real reason we haven’t run out of valid IP addresses on the Internet.

**How NAT Works**

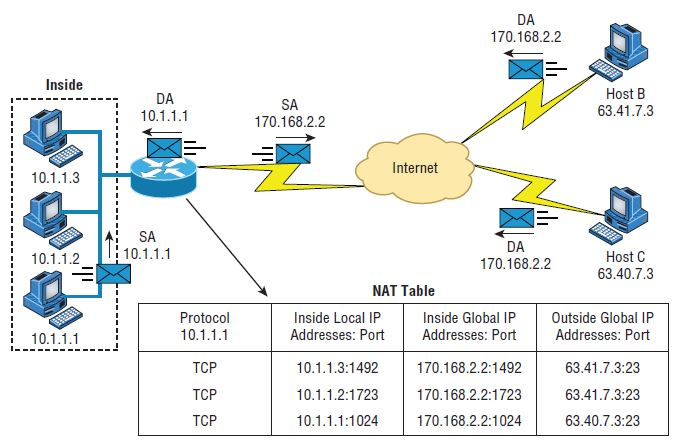


**Basic NAT translation**

In the above example, host 10.1.1.1 sends an outbound packet to the border router configured with NAT. The router identifies the IP address as an inside local IP address destined for an outside network, translates the address, and documents the translation in the NAT table.

The packet is sent to the outside interface with the new translated source address. The external host returns the packet to the destination host, and the NAT router translates the inside global IP address back to the inside local IP address using the NAT table.

Let’s take a look at a more complex configuration using overloading, or what is also referred to as PAT.



**NAT overloading example (PAT)**

With overloading, all inside hosts get translated to one single IP address, hence the term overloading. Again, the reason we have not run out of available IP addresses on the Internet is because of overloading (PAT).

Take a look at the NAT table in the above figure again. In addition to the inside local IP address and outside global IP address, we now have **port** numbers. These port numbers help the router identify which host should receive the return traffic.

**Port numbers** are used at the **Transport layer** to identify the localhost in this example. If we had to use IP addresses to identify the source hosts, that would be called static NAT, and we would run out of addresses. PAT allows us to use the **Transport layer** to identify the hosts, which in turn allows us to use (theoretically) up to 65,000 hosts with one real IP address.

Using a router or firewall, you can also perform port forwarding, which is translating the port number of a packet to a new destination. The destination may be a predetermined network port (using an IP protocol, but typically TCP or UDP ports) on a host within a private network behind a NAT router. Based on the received port number, a remote host can communicate to servers behind the NAT gateway to the local network.

**Routing Basics**

Once you create an internetwork by connecting your wide area networks (WANs) and local area networks (LANs) to a **router**, you need to configure logical network addresses, such as IP addresses, to all hosts on the internetwork so that they can communicate via routers across that internetwork.

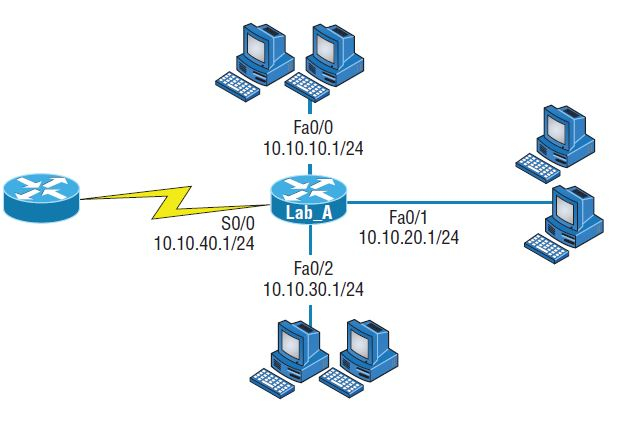
In IT, routing essentially refers to the process of taking a packet from one device and sending it through the network to another device on a different network. Routers don’t really care about hosts—they care only about networks and the best path to each network. The logical network address of the destination host is used to get packets to a network through a routed network, and then the hardware address of the host is used to deliver the packet from a router to the correct destination host.

If your network has no routers, then it should be apparent that, well, you are not routing. But if you do have them, they’re there to route traffic to all the networks in your internetwork. To be capable of routing packets, a router must know at least the following information:

* Destination network address
* Neighbor routers from which it can learn about remote networks
* Possible routes to all remote networks
* The best route to each remote network
* How to maintain and verify routing information

The router learns about remote networks from neighbor routers or from an administrator. The router then builds a **routing table** (a map of the internetwork) that describes how to find the remote networks. If a network is directly connected, then the router already knows how to get to it.

If a network isn’t directly connected to the router, the router must use one of two ways to learn how to get to it. One way is called **static routing**, which can be a ton of work because it requires someone to hand-type all network locations into the routing table. The other way is **dynamic routing**. In dynamic routing, a protocol on one router communicates with the same protocol running on neighbor routers. The routers then update each other about all the networks they know about and place this information into the routing table. If a change occurs in the network, the dynamic routing protocols automatically inform all routers about the event. If static routing is used, the *administrator* is responsible for updating all changes by hand into all routers.



Routing example

The above figure shows a simple two-router network. Lab\_A has one serial interface and three LAN interfaces. Can you figure out which interface Lab\_A will use to forward an IP datagram to a host with an IP address of 10.10.10.10? By using the Cisco IOS command show ip route, we can see the routing table (map of the internetwork) that router Lab\_A will use to make all forwarding decisions:

Router\_A#show ip route

[output cut]

Gateway of last resort is not set

C 10.10.10.0/24 is directly connected, FastEthernet0/0

C 10.10.20.0/24 is directly connected, FastEthernet0/1

C 10.10.30.0/24 is directly connected, FastEthernet0/2

C 10.10.40.0/24 is directly connected, Serial 0/0

The C in the routing table output means that the networks listed are “directly connected,” and until we add a routing protocol—something like *RIP, EIGRP*, and so on—to the routers in our internetwork, or use static routes, we’ll have only directly connected networks in our routing table.

By looking at the figure and the output of the routing table, can you tell what Lab\_A will do with a received packet that has a destination IP address of 10.10.10.10? If you answered, “The router will packet-switch the packet to interface FastEthernet 0/0, and this interface will then frame the packet and send it out on the network segment,” you’re right.

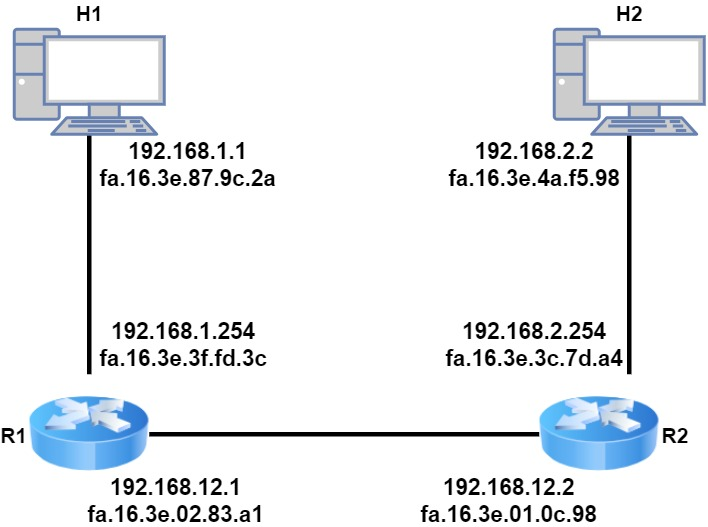
**💡Tip:**

* When the routing tables of all routers in the network are complete (because they include information about all the networks in the internetwork), they are considered converged or in a steady state.

**The IP Routing Process**

The actual forwarding of IP packets by routers is called **IP routing**. This has nothing to do with the “learning” of network routes through static or dynamic routing protocols but has everything to do with the steps that routers have to take when they forward an IP packet from one interface to another.

In this lesson, we will walk through an example and show all the steps that occur. To do this, we will use the following topology:



**IP routing example topology**

Above we have two host computers and two routers. H1 is going to send an IP packet to H2 which has to be routed by R1 and R2.

**H1**

Let’s start with **H1**. This host creates an IP packet with its own IP address (192.168.1.1) as the source and **H2** (192.168.2.2) as the destination. The first question that **H1** will ask itself is:

* *Is the destination local or remote?*

It answers this question by looking at its own IP address, its subnet mask, and the destination IP address:

C:\Users\H1>ipconfig

Windows IP Configuration

Ethernet adapter Ethernet 1:

Connection-specific DNS Suffix . : nwl.local

Link-local IPv6 Address . . . . . : fe80::88fd:962a:44d6:3a1f%4

IPv4 Address. . . . . . . . . . . : 192.168.1.1

Subnet Mask . . . . . . . . . . . : 255.255.255.0

Default Gateway . . . . . . . . . : 192.168.1.254

**H1** is in the network 192.168.1.0/24 so all IP addresses in the 192.168.1.1 – 254 range are local. Our destination (192.168.2.2) is outside of the local subnet so that means we have to use the default gateway.

**H1** will now build an *Ethernet frame*, enters its own *source MAC address* and asks itself the second question, *do I know the destination MAC address of the default gateway?*

It checks its ARP (Address Resolution Table) table to find the answer:

C:\Users\H1>arp -a

Interface: 192.168.1.1 --- 0x4

Internet Address Physical Address Type

192.168.1.254 fa-16-3e-3f-fd-3c dynamic

192.168.1.255 ff-ff-ff-ff-ff-ff static

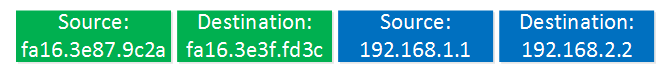
224.0.0.22 01-00-5e-00-00-16 static

224.0.0.251 01-00-5e-00-00-fb static

224.0.0.252 01-00-5e-00-00-fc static

239.255.255.250 01-00-5e-7f-ff-fa static

**H1** has an ARP entry for 192.168.1.254. If not, it would have sent an ARP request. We now have an Ethernet frame that carries an IP packet with the following addresses:

**Ethernet Frame**

The frame will be on its way to **R1**.

**R1**

This Ethernet frame makes it to **R1** which has more work to do than our host. The first thing it does is check if the FCS (Frame Check Sequence) of the Ethernet frame is correct or not:

**Ethernet Frame**

If the FCS is incorrect, the frame is dropped right away. There is no error recovery for Ethernet, this is something that is done by protocols on upper layers, like TCP on the transport layer.

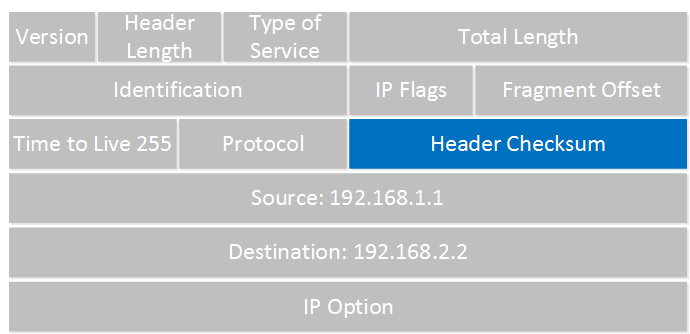
If the FCS is correct, we will process the frame if the destination MAC address is:

* the address of the interface of the router
* is a broadcast address of the subnet that the router interface is connected to
* is a multicast address that the router listens to.

In this case, the destination MAC address matches the MAC address of **R1**’s GigabitEthernet 0/1 interface so we will process it. We de-encapsulate (extract) the IP packet out of the Ethernet frame which is then discarded:

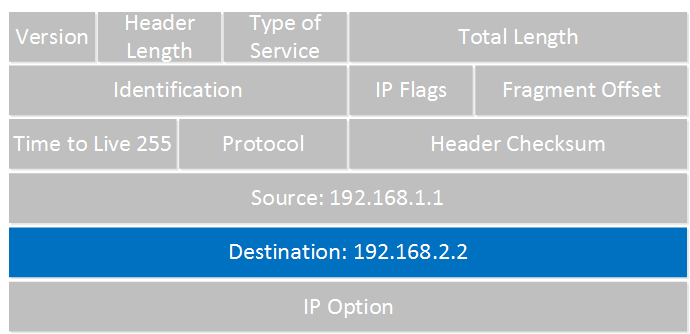
**Ethernet Frame**

The router will now look at the IP packet, and the first thing it does is check if the header checksum is OK:



**IP package**

If the header checksum is not correct, the IP packet is dropped right away. There is also no error recovery on the network layer, we rely on upper layers for this. If the header checksum is correct, we continue by looking at the destination IP address:

**IP package**

**R1** now checks its routing table to see if there is a match:

R1#show ip route

Gateway of last resort is not set

192.168.1.0/24 is variably subnetted, 2 subnets, 2 masks

C 192.168.1.0/24 is directly connected, GigabitEthernet0/1

L 192.168.1.254/32 is directly connected, GigabitEthernet0/1

S 192.168.2.0/24 [1/0] via 192.168.12.2

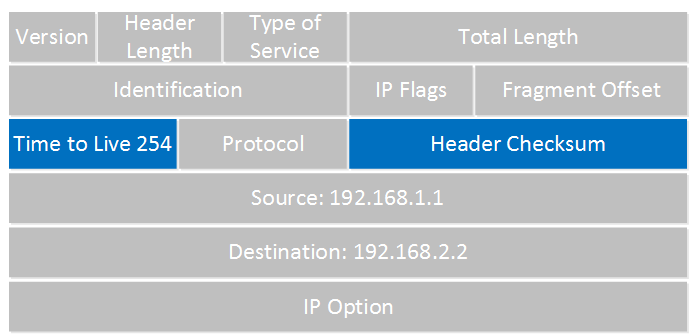
192.168.12.0/24 is variably subnetted, 2 subnets, 2 masks

C 192.168.12.0/24 is directly connected, GigabitEthernet0/2

L 192.168.12.1/32 is directly connected, GigabitEthernet0/2

Above you can see that **R1** knows how to reach the 192.168.2.0/24 network, the next-hop IP address is 192.168.12.2. It will now do a second routing table lookup to see if it knows how to reach 192.168.12.2, we call this recursive routing. As you can see, there is an entry for 192.168.12.0/24 with GigabitEthernet 0/2 as the interface to use.

There is one thing left to do with the IP packet before we can forward it. Since we are routing it, we have to decrease the TTL (Time to Live) field by one. **R1** will do this and since this changes the IP header, we have to calculate a new header checksum.



**IP package**

Once this is done, **R1** checks its ARP table to see if there is an entry for 192.168.12.2:

R1#show ip arp

Protocol Address Age (min) Hardware Addr Type Interface

Internet 192.168.1.1 58 fa16.3e87.9c2a ARPA GigabitEthernet0/1

Internet 192.168.1.254 - fa16.3e3f.fd3c ARPA GigabitEthernet0/1

Internet 192.168.12.1 - fa16.3e02.83a1 ARPA GigabitEthernet0/2

Internet 192.168.12.2 95 fa16.3e01.0c98 ARPA GigabitEthernet0/2

No problem there, we have an entry in the ARP table. If not, **R1** will send an ARP request to find the MAC address of 192.168.12.2. **R1** builds a new Ethernet frame with its own MAC address of the GigabitEthernet 0/2 interface and R2 as the destination. The IP packet is then encapsulated in this new Ethernet frame.



**Ethernet Frame**

And the frame will be on its way towards **R2**.

**R2**

This Ethernet frame makes it to **R2**. Like **R1** it will first do this:

* Check the FCS of the Ethernet frame.
* De-encapsulates the IP packet, discard the frame.
* Check the IP header checksum.
* Check the destination IP address.

In the routing table, we find this:

R2#show ip route

Gateway of last resort is not set

S 192.168.1.0/24 [1/0] via 192.168.12.1

192.168.2.0/24 is variably subnetted, 2 subnets, 2 masks

C 192.168.2.0/24 is directly connected, GigabitEthernet0/1

L 192.168.2.254/32 is directly connected, GigabitEthernet0/1

192.168.12.0/24 is variably subnetted, 2 subnets, 2 masks

C 192.168.12.0/24 is directly connected, GigabitEthernet0/2

L 192.168.12.2/32 is directly connected, GigabitEthernet0/2

Network 192.168.2.0/24 is directly connected to **R2** on its GigabitEthernet 0/1 interface. **R2** will now reduce the TTL of the IP packet from 254 to 253, recalculate the IP header checksum and checks its ARP table to see if it knows how to reach 192.168.2.2:

R2#show ip arp

Protocol Address Age (min) Hardware Addr Type Interface

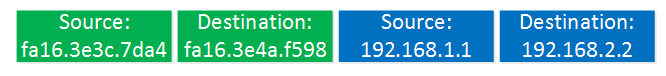
Internet 192.168.2.2 121 fa16.3e4a.f598 ARPA GigabitEthernet0/1

Internet 192.168.2.254 - fa16.3e3c.7da4 ARPA GigabitEthernet0/1

Internet 192.168.12.1 111 fa16.3e02.83a1 ARPA GigabitEthernet0/2

Internet 192.168.12.2 - fa16.3e01.0c98 ARPA GigabitEthernet0/2

There is an ARP entry there. The new Ethernet frame is created, the IP packet encapsulated and it has the following addresses:



**Ethernet Frame**

The frame is then forwarded to **H2**.

H2

**H2** receives the Ethernet frame and will:

* Check the FCS
* Find its own MAC address as the destination MAC address.
* De-encapsulates the IP packet from the frame.
* Finds its own IP address as the destination in the IP packet.

**H2** then looks for the protocol field to figure out what transport layer protocol we are dealing with, what happens next depends on the *transport layer protocol* that is used.

**Static and Dynamic Routing**

How does a router send packets to remote networks when the only way it can send them is by looking at the routing table to find out how to get to the remote networks? And what happens when a router receives a packet for a network that isn’t listed in the routing table? It doesn’t send a broadcast looking for the remote network—the router just discards the packet.

There are several ways to configure the routing tables to include all the networks so that packets will be forwarded. Understand that what’s best for one network isn’t necessarily what’s best for another. Knowing about and being able to recognize the different types of routing will really help you come up with the best solution for your specific environment and business requirements.

We can configure a router with either **static** or **dynamic routing**. If we choose **static routing**, then we have to go to each router and type in each network and the path that IP will use to send packets. However, **static routing** does not scale well in large networks, but **dynamic routing** does because network routes are automatically added to the routing table via the routing protocol.

*Dynamic routing protocols* break up into many different categories or types of protocols. The first split in the dynamic protocol branch is the division of **interior gateway protocols (IGPs)** and **exterior gateway protocols (EGPs)** (The difference between IGP and EGP is interior or exterior routing of an **autonomous system (AS)**).

An **autonomous system** is a collection of networks or subnets that are in the same administrative domain. You control and set the policy for what happens in the network or autonomous system.

The most popular protocol for an **EGP** is **Border Gateway Protocol (BGP)**, which is typically used by ISPs or really large corporations. As an administrator of a small to medium network, you’ll probably never use BGP. (BGP will be discussed in the following section)

What kinds of great things the dynamic routing protocols provide for us. Well, the thing that comes to mind first is the **amount of time** and **energy** we save configuring routers. We won’t have to go to every single router and define for it, with a static route, what and where every destination network is. We still have to know what the routing protocols are going to do and how they will do it, but the protocols will take care of most of the updating and sending information to each other.

The IGP splits into two primary categories: **distance vector (DV)** and **link-state (LS)** routing protocols. But in the distance vector category, for example, we have RIP and Interior Gateway Routing Protocol (IGRP). Under the link-state category are the **nonproprietary OSPF** and **Intermediate System-to-Intermediate System (IS-IS)** that was designed to work in larger internetworks.

**Using The traceroute Command**

**Using traceroute**

Where do all those packets really go when we send them over the Internet? And, how do all the packets actually get to their destinations? Well, we can use the **TCP/IP traceroute** (*tracert with Windows*) command-line utility to help us answer both questions because its output will show us every router interface a TCP/IP packet passes through on the way to its destination.

**Traceroute** (trace for short) displays the path a packet takes to get to a remote device by using something we call **IP packet Time to Live (TTL) time-outs** and **Internet Control Message Protocol (ICMP)** error messages. And it’s also a handy tool for troubleshooting an internetwork because we can use it to figure out which router along a path through that internetwork happens to be causing a network failure when a certain destination machine or network is, or suddenly becomes, unreachable.

To use **tracert**, at a Windows command prompt, type tracert, a space, and the Domain Name Service (DNS) name or IP address of the host machine to which you want to find the route. The **tracert utility** will respond with a list of all the DNS names and IP addresses of the routers that the packet is passing through on its way. Plus, tracert uses TTL to indicate the time it takes for each attempt.

Following is the *tracert output* from a local pc to clarusway.com server:

Microsoft Windows [Version 10.0.18363.657]

(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\clarusway>tracert www.clarusway.com

Tracing route to www.clarusway.com [54.164.151.235]

over a maximum of 30 hops:

1 1 ms 1 ms 1 ms 192.168.1.1

2 4 ms 4 ms \* 195.87.128.37

3 10 ms 10 ms \* 10.135.53.154

4 11 ms 10 ms \* 10.135.53.153

5 11 ms 11 ms \* 46.234.28.57

6 11 ms 10 ms \* ae4-17-ucr1.tuz.cw.net [195.2.23.129]

7 133 ms 134 ms 132 ms 195.2.25.86

8 130 ms 130 ms 130 ms 195.2.28.57

9 132 ms 133 ms 132 ms ae17.pcr1.fnt.cw.net [195.2.20.226]

10 \* \* 131 ms ae15-pcr1.ptl.cw.net [195.2.9.126]

11 131 ms 131 ms 133 ms et-7-1-0-xcr1.nyh.cw.net [195.2.24.241]

12 131 ms 131 ms 155 ms ae13-xcr2.nyk.cw.net [195.2.25.69]

13 132 ms 132 ms 132 ms 52.95.216.78

14 141 ms 135 ms 131 ms 52.93.4.85

15 131 ms 131 ms 132 ms 52.93.4.46

16 \* \* \* Request timed out.

17 140 ms 136 ms 137 ms 150.222.242.116

18 \* \* \* Request timed out.

19 \* \* \* Request timed out.

20 \* \* \* Request timed out.

21 \* \* \* Request timed out.

22 137 ms 158 ms 140 ms 150.222.241.173

23 \* \* \* Request timed out.

24 \* \* \* Request timed out.

25 \* \* \* Request timed out.

26 \* \* \* Request timed out.

27 \* \* \* Request timed out.

28 \* \* \* Request timed out.

29 \* \* \* Request timed out.

30 \* \* \* Request timed out.

Trace complete.

You see that the packet bounces through several routers before arriving at its destination. This utility is useful if you are having problems reaching a web server on the Internet and you want to know if a wide area network (WAN) link is down or if the server just isn’t responding. What this means to you is that, basically, wherever the trace stops is a great place to start troubleshooting. Notice in the output the *ms*. This is the latency of each hop, meaning the delay. Tracert (or traceroute) is a great troubleshooting tool you can use to find out where your network bottlenecks(dar boğazlar) are.

If you use traceroute or tracert and receive an asterisk, this indicates that the attempt to reach that router took longer than the default time-out value. This is very good to know because it can mean that either the router is extremely busy or a particular link is slow. Another reason for getting an asterisk could be that the administrator has disabled ICMP on the router that the packet is trying to hop through(atlamaya çalışıyor) because of security reasons. It happens to be a typical strategic move done on the routers that interface to the ISP to conceal their actual location so bad guys can’t hack into them and therefore into your internetwork.

In addition to traceroute and tracert, you can use pathping (for Windows), which is a lot like traceroute:

C:\Users\clarusway>pathping www.clarusway.com

Tracing route to www.clarusway.com [54.164.151.235]

over a maximum of 30 hops:

0 freestyler.home [192.168.1.22]

1 192.168.1.1

2 195.87.128.37

3 10.135.53.154

4 10.135.53.153

5 46.234.28.57

6 ae4-17-ucr1.tuz.cw.net [195.2.23.129]

7 ae2-ucr1.pra.cw.net [195.2.25.86]

8 ae16-xcr1.fix.cw.net [195.2.28.57]

9 ae17.pcr1.fnt.cw.net [195.2.20.226]

10 ae15-pcr1.ptl.cw.net [195.2.9.126]

11 et-7-1-0-xcr1.nyh.cw.net [195.2.24.241]

12 ae13-xcr2.nyk.cw.net [195.2.25.69]

13 52.95.216.78

14 52.93.4.85

15 52.93.4.46

16 \* \* \*

Computing statistics for 375 seconds...

Source to Here This Node/Link

Hop RTT Lost/Sent = Pct Lost/Sent = Pct Address

0 freestyler.home [192.168.1.22]

0/ 100 = 0% |

1 1ms 0/ 100 = 0% 0/ 100 = 0% 192.168.1.1

0/ 100 = 0% |

2 --- 100/ 100 =100% 100/ 100 =100% 195.87.128.37

0/ 100 = 0% |

3 13ms 0/ 100 = 0% 0/ 100 = 0% 10.135.53.154

0/ 100 = 0% |

4 12ms 0/ 100 = 0% 0/ 100 = 0% 10.135.53.153

0/ 100 = 0% |

5 12ms 0/ 100 = 0% 0/ 100 = 0% 46.234.28.57

0/ 100 = 0% |

6 15ms 0/ 100 = 0% 0/ 100 = 0% ae4-17-ucr1.tuz.cw.net [195.2.23.129]

0/ 100 = 0% |

7 47ms 0/ 100 = 0% 0/ 100 = 0% ae2-ucr1.pra.cw.net [195.2.25.86]

0/ 100 = 0% |

8 56ms 0/ 100 = 0% 0/ 100 = 0% ae16-xcr1.fix.cw.net [195.2.28.57]

0/ 100 = 0% |

9 59ms 0/ 100 = 0% 0/ 100 = 0% ae17.pcr1.fnt.cw.net [195.2.20.226]

0/ 100 = 0% |

10 64ms 0/ 100 = 0% 0/ 100 = 0% ae15-pcr1.ptl.cw.net [195.2.9.126]

0/ 100 = 0% |

11 134ms 0/ 100 = 0% 0/ 100 = 0% et-7-1-0-xcr1.nyh.cw.net [195.2.24.241]

0/ 100 = 0% |

12 133ms 0/ 100 = 0% 0/ 100 = 0% ae13-xcr2.nyk.cw.net [195.2.25.69]

100/ 100 =100% |

13 --- 100/ 100 =100% 0/ 100 = 0% 52.95.216.78

0/ 100 = 0% |

14 --- 100/ 100 =100% 0/ 100 = 0% 52.93.4.85

0/ 100 = 0% |

15 --- 100/ 100 =100% 0/ 100 = 0% 52.93.4.46

Trace complete.Formun Üstü

Formun Altı

Formun Üstü

Formun Altı

**Using the ipconfig Utility**

With the new Mac, Windows 10, and Windows Server 2016 operating systems, you can now see the IPv6 configuration because IPv6 is enabled by default. The output of the ipconfig command provides the basic routed protocol information on your machine. From a DOS prompt, type ipconfig, and you’ll see something like this:

C:\Users\clarusway>ipconfig

Windows IP Configuration

Ethernet adapter Ethernet:

Media State . . . . . . . . . . . : Media disconnected

Connection-specific DNS Suffix . :

Wireless LAN adapter Local Area Connection\* 3:

Media State . . . . . . . . . . . : Media disconnected

Connection-specific DNS Suffix . :

Wireless LAN adapter Wi-Fi:

Connection-specific DNS Suffix . : home

Link-local IPv6 Address . . . . . : fe80::19ac:8efb:2c6e:f512%10

IPv4 Address. . . . . . . . . . . : 192.168.1.22

Subnet Mask . . . . . . . . . . . : 255.255.255.0

Default Gateway . . . . . . . . . : 192.168.1.1

Tunnel adapter Teredo Tunneling Pseudo-Interface:

Connection-specific DNS Suffix . :

IPv6 Address. . . . . . . . . . . : 2001:0:2851:782c:148e:f3fd:6aff:55b8

Link-local IPv6 Address . . . . . : fe80::148e:f3fd:6aff:55b8%17

Default Gateway . . . . . . . . . : ::

You can see that Ethernet adapter shows up first, and it has an IP address, a mask, and a default gateway plus an IPv6 address and a DNS suffix. The next configured interface is the wireless local area network (LAN) adapter, which has an IP address, a mask, a default gateway, an IPv6 address, and the IPv6 default gateway as well.

The next adapters are disconnected because they are logical interfaces and are not being used. But just in case the ipconfig command doesn’t provide enough information for you, try the ipconfig /all command. Here’s the beginning of that output:

C:\Users\clarusway>ipconfig /all

Windows IP Configuration

Host Name . . . . . . . . . . . . : clarusway

Primary Dns Suffix . . . . . . . :

Node Type . . . . . . . . . . . . : Hybrid

IP Routing Enabled. . . . . . . . : No

WINS Proxy Enabled. . . . . . . . : No

DNS Suffix Search List. . . . . . : home

Ethernet adapter Ethernet:

Media State . . . . . . . . . . . : Media disconnected

Connection-specific DNS Suffix . :

Description . . . . . . . . . . . : Intel(R) I211 Gigabit Network Connection

Physical Address. . . . . . . . . : 9C-5C-8E-CE-D9-C9

DHCP Enabled. . . . . . . . . . . : Yes

Autoconfiguration Enabled . . . . : Yes

Ethernet adapter Ethernet 3:

Media State . . . . . . . . . . . : Media disconnected

Connection-specific DNS Suffix . :

Description . . . . . . . . . . . : Intel(R) Ethernet Connection (2) I219-V

Physical Address. . . . . . . . . : 9C-5C-8E-CE-D9-CA

DHCP Enabled. . . . . . . . . . . : Yes

Autoconfiguration Enabled . . . . : Yes

Wireless LAN adapter Local Area Connection\* 3:

Media State . . . . . . . . . . . : Media disconnected

Connection-specific DNS Suffix . :

Description . . . . . . . . . . . : Microsoft Wi-Fi Direct Virtual Adapter

Physical Address. . . . . . . . . : 76-C6-3B-00-62-86

DHCP Enabled. . . . . . . . . . . : Yes

Autoconfiguration Enabled . . . . : Yes

Wireless LAN adapter Local Area Connection\* 13:

Media State . . . . . . . . . . . : Media disconnected

Connection-specific DNS Suffix . :

Description . . . . . . . . . . . : Microsoft Wi-Fi Direct Virtual Adapter #2

Physical Address. . . . . . . . . : 76-C6-3B-00-6A-86

DHCP Enabled. . . . . . . . . . . : Yes

Autoconfiguration Enabled . . . . : Yes

Wireless LAN adapter Wi-Fi:

Connection-specific DNS Suffix . : home

Description . . . . . . . . . . . : Broadcom 802.11ac Network Adapter

Physical Address. . . . . . . . . : 74-C6-3B-00-62-86

DHCP Enabled. . . . . . . . . . . : Yes

Autoconfiguration Enabled . . . . : Yes

Link-local IPv6 Address . . . . . : fe80::19ac:8efb:2c6e:f512%10(Preferred)

IPv4 Address. . . . . . . . . . . : 192.168.1.22(Preferred)

Subnet Mask . . . . . . . . . . . : 255.255.255.0

Lease Obtained. . . . . . . . . . : March 3, 2020 01:01:19 PM

Lease Expires . . . . . . . . . . : March 4, 2020 02:54:14 PM

Default Gateway . . . . . . . . . : 192.168.1.1

DHCP Server . . . . . . . . . . . : 192.168.1.1

DHCPv6 IAID . . . . . . . . . . . : 242533947

DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-21-FA-0A-0E-9C-5C-8E-CE-D9-C9

DNS Servers . . . . . . . . . . . : 192.168.1.1

192.168.1.1

NetBIOS over Tcpip. . . . . . . . : Enabled

Tunnel adapter Teredo Tunneling Pseudo-Interface:

Connection-specific DNS Suffix . :

Description . . . . . . . . . . . : Microsoft Teredo Tunneling Adapter

Physical Address. . . . . . . . . : 00-00-00-00-00-00-00-E0

DHCP Enabled. . . . . . . . . . . : No

Autoconfiguration Enabled . . . . : Yes

IPv6 Address. . . . . . . . . . . : 2001:0:2851:782c:148e:f3fd:6aff:55b8(Preferred)

Link-local IPv6 Address . . . . . : fe80::148e:f3fd:6aff:55b8%17(Preferred)

Default Gateway . . . . . . . . . : ::

DHCPv6 IAID . . . . . . . . . . . : 167772160

DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-21-FA-0A-0E-9C-5C-8E-CE-D9-C9

NetBIOS over Tcpip. . . . . . . . : Disabled

As you can see, it’s more of the same—a whole lot more. The most important thing that you can see the hardware information about each interface, including the Media Access Control (MAC) address. Also significant is that you can see the Dynamic Host Configuration Protocol (DHCP) lease times and DNS addresses now.

There are two more valuable options you need to use with the ipconfig command. They are /release and /renew.

When you change networks, you need to get the IP address of that subnet and/or virtual LAN (VLAN). Windows 10 works most of the time without doing anything, but sometimes you have to renew the IP configuration when changing networks. For that, just type ipconfig /renew from a command prompt, and if you’re connected to a DHCP server that’s available. Now, if it still doesn’t work, you’ll need to release and renew your TCP/IP settings. To release your current DHCP TCP/IP information, you must elevate your command prompt or you’ll get this warning:

C:\Users\clarusway>ipconfig /release

The requested operation requires elevation.

In order to avoid this, choose Start > All Programs > Accessories > Command Prompt, right-click, and choose Run As Administrator.

Once your command prompt has been duly elevated(istendiği gibi yükseldikten sonra), you can use the ipconfig /release command and then the ipconfig /renew command to get new TCP/IP information for your host.

**Using the ifconfig Utility**

There is a utility in Linux/Unix/Mac that will give you information similar to what ipconfig shows. It’s called ifconfig (short for interface configuration). Although ipconfig and ifconfig show similar information, there are major differences between these two utilities.

The ipconfig utility is mainly used to view the TCP/IP configuration for a computer. You can use ifconfig to do the same thing, but ifconfig can also be used to configure a protocol or a particular network interface.

The general syntax of the ifconfig command is as follows:

ifconfig interface [address [parameters]]

The interface parameter equals the Unix name of the interface, such as eth0. If the optional address parameter is specified, the ifconfig command sets the IP address for the interface to the address you’ve specified. When the ifconfig command is used by itself with no parameters, all configured interfaces will be reported on. But if only the interface name is specified, you’ll get output that looks like this:

# ifconfig eth0

eth0 Link encap 10Mbps Ethernet HWaddr 00:00:C0:90:B3:42

inetaddr 172.16.0.2 Bcast 172.16.0.255 Mask 255.255.255.0 UP

BROADCAST RUNNING MTU 1500 Metric 0

RX packets 3136 errors 217 dropped 7 overrun 26

TX packets 1752 errors 25 dropped 0 overrun 0

Looking at this, we can see that the eth0 interface is a 10 Mbps Ethernet interface. The interface’s MAC and IP address information is displayed in this output as well. And, although not shown in the output, the ifconfig tool can show you the DNS information configured on the host.

**Using the iptables Utility**

The **iptables firewall utility** is built for the Linux operating system. It is a command-line utility that uses what are called chains to allow or disallow traffic. When traffic arrives, iptables looks for a rule that addresses that traffic type, and if none exists, it will enforce the default rule. There are three different chain types:

1. **Input:** Controls behavior for incoming connections
2. **Forward:** Used for incoming connections that aren’t being delivered locally
3. **Output:** Used for outgoing connections

You can set the default action to accept, drop, or reject, with the difference between reject and drop being that reject sends an error message back to the source.

Examples of iptables

* To block a connection from the device at 192.168.10.1, use this command:

iptables -A INPUT -s 192.168.10.1 -j DROP

* To block all connections from all devices in the 172.16.0.0/16 network, use this command:

iptables -A INPUT -s 172.16.0.0/16 -j DROP

* Here is the command to block SSH connections from 10.110.61.5:

iptables -A INPUT -p tcp --dport ssh -s 10.110.61.5 -j DROP

* Use this command to block SSH connections from any IP address:

iptables -A INPUT -p tcp --dport ssh -j DROP

* The following command is used to save the changes in Ubuntu:

sudo /sbin/iptables-save

* In Red Hat/CentOS, use either of the following commands:

/sbin/service iptables save

/etc/init.d/iptables save

**Using The ping Utility**

The **ping utility** is the most basic TCP/IP utility, and it’s included with most TCP/IP stacks for most platforms. In most cases, ping is a command-line utility, although there are many GUI implementations available. You use the ping utility for two primary purposes:

* To find out if a host is responding
* To find out if you can reach a host

Here’s the syntax (you can use either command):

ping hostname

ping IP address

If you ping any station that has an IP address, the ICMP that’s part of that particular host’s TCP/IP stack will respond to the request. The ICMP test and response looks something like this:

If you ping any station that has an IP address, the ICMP that’s part of that particular host’s TCP/IP stack will respond to the request. The ICMP test and response looks something like this:

C:\Users\clarusway>ping 3.225.75.90

Pinging 3.225.75.90 with 32 bytes of data:

Reply from 3.225.75.90: bytes=32 time=137ms TTL=233

Reply from 3.225.75.90: bytes=32 time=136ms TTL=233

Reply from 3.225.75.90: bytes=32 time=134ms TTL=233

Reply from 3.225.75.90: bytes=32 time=134ms TTL=233

Ping statistics for 3.225.75.90:

Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),

Approximate round trip times in milli-seconds:

Minimum = 134ms, Maximum = 137ms, Average = 135ms

Because we’ve received a reply from the destination station we know that we can reach the host and that it’s responding to basic IP requests. Don’t forget that you can use name resolution and ping to a name, such as ping www.clarusway.com. Most versions of ping work the same way, but there are some switches you can use to specify certain information, like the number of packets to send, how big a packet to send, and so on. And if you’re running the Windows command-line version of ping, just use the /? or –? switch to display a list of the available options like this:

C:\Users\clarusway>ping /?

Usage: ping [-t] [-a] [-n count] [-l size] [-f] [-i TTL] [-v TOS]

[-r count] [-s count] [[-j host-list] | [-k host-list]]

[-w timeout] [-R] [-S srcaddr] [-c compartment] [-p]

[-4] [-6] target\_name

Options:

-t Ping the specified host until stopped.

To see statistics and continue - type Control-Break;

To stop - type Control-C.

-a Resolve addresses to hostnames.

-n count Number of echo requests to send.

-l size Send buffer size.

-f Set Don't Fragment flag in packet (IPv4-only).

-i TTL Time To Live.

-v TOS Type Of Service (IPv4-only. This setting has been deprecated

and has no effect on the type of service field in the IP

Header).

-r count Record route for count hops (IPv4-only).

-s count Timestamp for count hops (IPv4-only).

-j host-list Loose source route along host-list (IPv4-only).

-k host-list Strict source route along host-list (IPv4-only).

-w timeout Timeout in milliseconds to wait for each reply.

-R Use routing header to test reverse route also (IPv6-only).

Per RFC 5095 the use of this routing header has been

deprecated. Some systems may drop echo requests if

this header is used.

-S srcaddr Source address to use.

-c compartment Routing compartment identifier.

-p Ping a Hyper-V Network Virtualization provider address.

-4 Force using IPv4.

-6 Force using IPv6.

As you can see, there are many options you can use with the ping command from a Windows DOS prompt.

The -a switch is handy if you have name resolution (such as a DNS server), you can see the name of the destination host even if you only know its IP address. The -n switch sets the number of echo requests to send, where four is the default, and the -w switch allows you to adjust the time-out in milliseconds. The default ping time-out is 1 second (1,000 ms).

The -6 is also nice if you want to ping an IPv6 host. And then there’s -t, which keeps the ping running.

From a MAC, you can use the ping6 command. Here are the options:

$ ping6

usage: ping6 [-DdfHmnNoqrRtvwW] [-a addrtype] [-b bufsiz] [-B boundif]

[-c count][-g gateway] [-h hoplimit] [-I interface] [-i wait] [-l preload][-p

pattern] [-S sourceaddr] [-s packetsize] [-z tclass]

[hops ...] host

**Using The Address Resolution Protocol**

The **Address Resolution Protocol (ARP)** is part of the TCP/IP protocol stack. It’s used to translate TCP/IP addresses to MAC addresses using broadcasts. When a machine running TCP/IP wants to know which machine on an Ethernet network is using a certain IP address, it will send an ARP broadcast that says, in effect, “Who is IP address xxx.xxx.xxx.xxx?” The machine that owns the specific address will respond with its own MAC address, supplying the answer. The machine that made the inquiry will respond by adding the newly gained information to its own ARP table.

The ARP table in Windows includes a list of TCP/IP addresses and their associated physical (MAC) addresses. This table is cached in memory so that Windows doesn’t have to perform ARP lookups for frequently accessed TCP/IP addresses like those of servers and default gateways. Each entry contains an IP address and a MAC address plus a value for TTL that determines how long each entry will remain in the ARP table.

Remember that the ARP table contains two kinds of entries:

* Dynamic
* Static

*Dynamic ARP table entries* are created whenever the Windows TCP/IP stack performs an ARP lookup but the MAC address isn’t found in the ARP table. When the MAC address of the requested IP address is finally found or resolved, that information is then added into the ARP table as a dynamic entry. Whenever a request to send a packet to the host is sent to the Data Link layer, the ARP cache is checked first before an ARP broadcast is sent out.