# Fundamental Network Characteristics

There are many different kinds of networks, and network technologies used to create them. The proliferation of networking methods has generally occurred for a very good reason: different needs require different solutions. The drawback of this is that there are so many different types of protocols and technologies for the networking student to understand! Before you can really compare these approaches, you need to understand some of the basic characteristics that make networks what they are. While network types may be quite dissimilar, they are often described and even contrasted on the basis of a number of common attributes.

In this section, I introduce and discuss a number of key networking concepts that describe and differentiate different types of networks and networking technologies. I also introduce and define a number of terms and “buzzwords” that you cannot avoid if you are going to learn about networks. The topics here include explanations of protocols, switching methods, types of network messages, message formatting, and ways of addressing messages. I also discuss the differences between client-server and peer-to-peer networking.

**Note:** If you have considerable experience in networking, you may not need to read everything in this section. I'd suggest scanning the headings of the various topics here; if you understand the terminology mentioned in a topic’s title, you can probably feel pretty safe in skipping it.

## Networking Layers, Models and Architectures

One of the reasons why many people find networking difficult to learn is that it can be a very complicated subject. One of the chief reasons for this complexity is that networks consist of so many hardware and software elements. While a network user may only perceive that he or she is using one computer program (like a Web browser) and one piece of hardware (like a PC), these are only parts of a much larger puzzle. In order for even the simplest task to be accomplished on a network, dozens of different components must cooperate, passing control information and data to accomplish the overall goal of network communication.

The best way to understand any complex system is to break it down into pieces and then analyze what they do and how they interact. The most logical approach for this is to do is divide the overall set of functions into modular components, each of which is responsible for a particular function. At the same time, we also need to define interfaces between these components, which describe how they fit together. This enables us to simplify the complexity of networking by approaching it in digestible chunks.

### *Networking Layers*

Networking technologies are most often compartmentalized in this manner by dividing their functions into *layers*, each of which contains hardware and/or software elements. Each layer is responsible for performing a particular type of task, as well as interacting with the layers above it and below it. Layers are conceptually arranged into a vertical stack. Lower layers are charged with more concrete tasks such as hardware signaling and low-level communication; they provide services to the higher layers. The higher layers in turn use these services to implement more abstract functions such as implementing user applications.

Dividing networks into layers this way is somewhat like the division of labor in a manufacturing facility, and yields similar benefits. Each hardware device or software program can be specialized to perform the function needed by that layer, like a well-trained specialist on an assembly line. The different modules can be combined in different ways as needed. Understanding how a network functions overall is also made much easier this way.

### *Networking Models*

One other important benefit of layering is that makes it possible for technologies defined by different groups to interoperate. For this to be possible, it is necessary for everyone to agree on how layers will be defined and used. The most common tool for this purpose is a *networking model*. The model describes what the different layers are in the network, what each is responsible for doing, and how they interact. A univerally-accepted model ensures that everyone is on the same page when creating hardware and software.

The most common general model in use today is the *Open Systems Interconnection (OSI) Reference Model*, which concepts of seven stacked layers. These range from the Physical Layer (layer one) at the bottom, which is responsible for low-level signaling, to the Application Layer (layer seven) at the top, where application software is implemented. Understanding the OSI model is essential to understanding networking as a whole. I explain models and layers in more detail, as well as providing a complete description of the OSI Reference Model, in its own dedicated section.

### *Networking Architectures*

Closely related to the concept of a model is that of an *architecture*. An architecture is essentially a set of rules that describes the function of some portion of the hardware and software that constitute a stack of layers. Such a ruleset usually takes the form of a specification or standard that describes how equipment and programs using the technology must behave. A networking architecture is designed to implement the functions associated with a particular contiguous set of layers of the OSI Reference Model, either formally or informally.

In this Guide we are, of course, interested in the TCP/IP protocol suite, which runs the Internet, and a complex set of technologies that spans many layers of the OSI model. It is by examining the various components of TCP/IP and how they implement different OSI model layers that we will really learn how TCP/IP works. For starters, the name of the suite, TCP/IP, comes from the Transmission Control Protocol (TCP), which operates at layer four

 

of the OSI model, and the Internet Protocol (IP) that runs at OSI model layer three. IP provides services to layer four and uses services of layer two below it. TCP uses IP's functions and provides functions to the layers above it. The complete examination of TCP/IP starts by looking at its architecture and a second, special model that was developed specifically to make sense of TCP/IP.

## Protocols: What Are They, Anyway?

If there's one word you will get used to seeing a lot as you go through this Guide, it is this one: *protocol*. You will see reference to networking protocols, internetworking protocols, high-level protocols, low-level protocols, protocol stacks, protocol suites, sub-protocols, and so on. Clearly protocols are important, yet many reference works and standards use the term over and over again without ever explaining it. One reason for this may be because the term is somewhat vague and can have many meanings, which can make it difficult to grasp.

### *The Meaning of the Word "Protocol"*

In some cases, understanding a technical term is easier if we go back to look at how the term is used in plain English. In the real world, a protocol often refers to a code of conduct, or a form of etiquette observed by diplomats. These people must follow certain rules of ceremony and form to ensure that they communicate effectively, and without coming into conflict. They also must understand what is expected of them when they interact with representatives from other nations, to make sure that, for example, they do not offend due to unfamiliarity with local customs. Even we "normal people" follow protocols of various sorts, which are sort of the "unwritten rules of society".

This may seem to have little to do with networking, but in fact, this is a pretty good high-level description of what networking protocols are about. They define a language and a set of rules and procedures that enable devices and systems to communicate. Obviously, computers do not have "local customs", and they hardly have to worry about committing a "faux pas" that might cause another computer to take offense. What networking protocols concern themselves with is ensuring that all the devices on a network or internetwork are in agreement about how various actions must be performed in the total communication process.

So, a protocol is basically a way of ensuring that devices are able to talk to each other effectively. In most cases, an individual protocol describes how communication is accomplished between one particular software or hardware element in two or more devices. In the context of the OSI Reference Model, a protocol is formally defined as a set of rules governing communication between entities at the same Reference Model layer. For example, the Transmission Control Protocol (TCP) is responsible for a specific set of functions on TCP/IP networks. Each host on a TCP/IP network has a TCP implementation, and they all communicate with each other logically at layer four of the OSI model.

While OSI Reference Model definitions are sometimes overly theoretical in nature, this particular one is rather accurate in assessing protocols in real-world networking. If something doesn't specify a means of communication, it arguably isn't a protocol.

**Key Concept:** A *networking protocol* defines a set of rules, algorithms, messages and other mechanisms that enable software and hardware in networked devices to communicate effectively. A protocol usually describes a means for communication between corresponding entities at the same OSI Reference Model layer in two or more devices.

**Related Information:** The formalized OSI Reference Model meaning of the word "protocol" is covered in the OSI model topic on horizontal layer communication.

### *Different Uses of the Word "Protocol"*

Despite the strict OSI definition, the term "protocol" is often used colloquially to refer to many different concepts in networking. Some of the more common "alternative" uses of the word include the following:

- **Protocol Suites:** It is very common to hear the word "protocol" used to refer to sets of protocols that are more properly called *protocol suites* (or *stacks*, in reference to a stack of layers). For example, TCP/IP is often called just a "protocol" when it is really a (large) set of protocols.

  Sometimes, the name of the technology itself leads to this confusion. The Point-to-Point Protocol (PPP), for example, is not one protocol; it contains many individual protocols that serve different functions and even have distinct message formats. Thus, PPP is really a protocol suite, or alternately, can be considered a protocol with "sub-protocols".

- **Microsoft Windows Protocols:** One important example of the issue of referring to protocol suites as single protocols is the networking software in Microsoft Windows. It usually calls a full networking stack like TCP/IP or IPX/SPX just a "protocol". When you install one of these "protocols", however, you actually get a software module that supports a full protocol suite.

- **Other Technologies:** Sometimes technologies that are not protocols at all are called protocols, either out of convention or perhaps because people think it sounds good. For example, TCP/IP Remote Network Monitoring (RMON) is often called a protocol when it is really just an enhancement to the Simple Network Management Protocol (SNMP)—which is a protocol!

So, does it really matter whether a protocol is a “true” protocol or not? Well, the networking hardware devices and software programs sure don’t care. ☺ But hopefully having read about the term and what it means, you will be able to better understand the word when you encounter it in your studies—especially in the places where it may not always be used in a way entirely consistent with its formal definition.

## Circuit Switching and Packet Switching Networks

In my “grand overview” of networking, I describe networks as devices that are connected together using special hardware and software, to allow them to exchange information. The most important word in that sentence is the final one: *information*. As you will see in your exploration of this Guide, there are many methods for exchanging information between networked devices. There are also a number of ways of categorizing and describing these methods and the types of networks that use them.

One fundamental way of differentiating networking technologies is on the basis of the method they use to determine the path between devices over which information will flow. In highly simplified terms, there are two approaches: either a path can be set up between the devices in advance, or the data can be sent as individual data elements over a variable path.

### *Circuit Switching*

In this networking method, a connection called a *circuit* is set up between two devices, which is used for the whole communication. Information about the nature of the circuit is maintained by the network. The circuit may either be a fixed one that is always present, or it may be a circuit that is created on an as-needed basis. Even if many potential paths through intermediate devices may exist between the two devices communicating, only one will be used for any given dialog. This is illustrated in Figure 1.

The classic example of a circuit-switched network is the telephone system. When you call someone and they answer, you establish a circuit connection and can pass data between you, in a steady stream if desired. That circuit functions the same way regardless of how many intermediate devices are used to carry your voice. You use it for as long as you need it, and then terminate the circuit. The next time you call, you get a new circuit, which may (probably will) use different hardware than the first circuit did, depending on what's available at that time in the network.

### *Packet Switching*

In this network type, no specific path is used for data transfer. Instead, the data is chopped up into small pieces called *packets* and sent over the network. The packets can be routed, combined or fragmented, as required to get them to their eventual destination. On the receiving end, the process is reversed—the data is read from the packets and re-

 

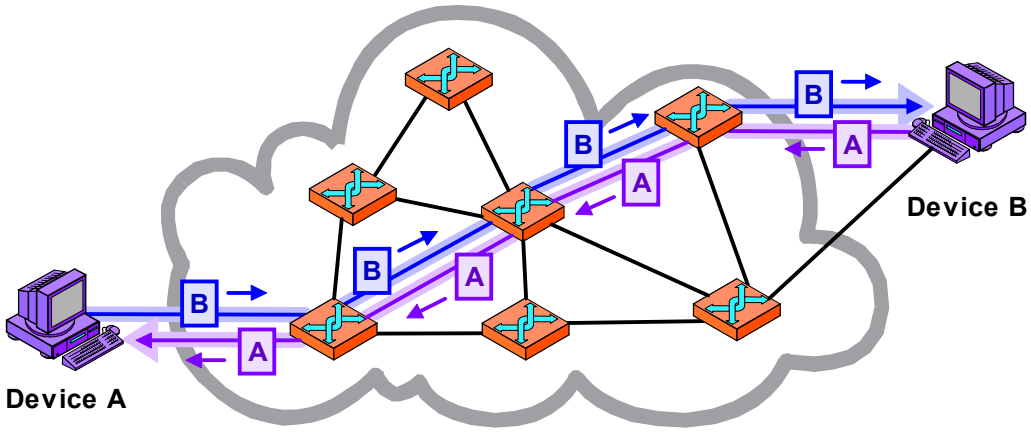


**Figure 1: Circuit Switching**

In a circuit-switched network, before communication can occur between two devices, a circuit is established between them. This is shown as a thick blue line for the conduit of data from Device *A* to Device *B*, and a matching purple line from *B* back to *A*. Once set up, all communication between these devices takes place over this circuit, even though there are other possible ways that data could conceivably be passed over the network of devices between them. Contrast this diagram to Figure 2.

assembled into the form of the original data. A packet-switched network is more analogous to the postal system than it is to the telephone system (though the comparison isn't perfect.) An example is shown in Figure 2.

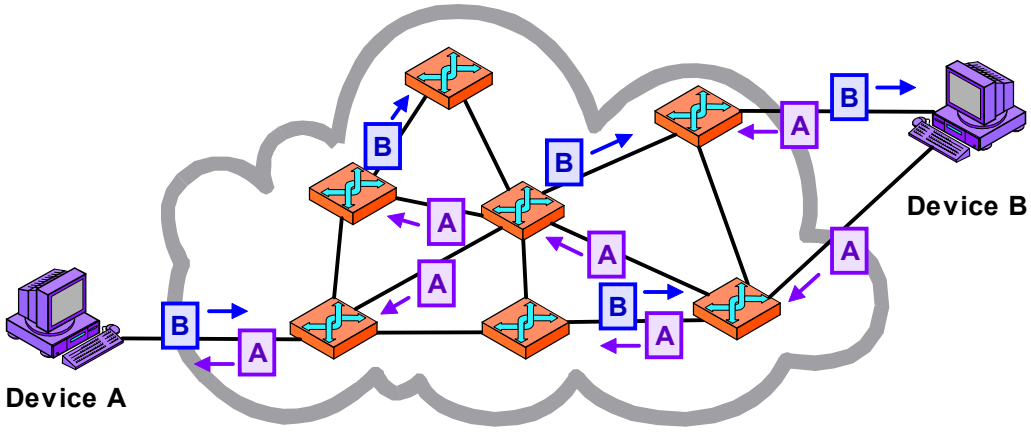


**Figure 2: Packet Switching**

In a packet-switched network, no circuit is set up prior to sending data between devices. Blocks of data, even from the same file or communication, may take any number of paths as it journeys from one device to another. Compare this to Figure 1

**Key Concept:** One way that networking technologies are categorized is based on the path used to carry data between devices. In *circuit switching*, a circuit is first established and then used to carry all data between devices. In *packet switching* no fixed path is created between devices that communicate; it is broken into packets, each of which may take a separate path from sender to recipient.

### *Comparing Circuit Switching and Packet Switching*

A common temptation when considering alternatives such as these is to ask which is "better"—and as usually is the case, the answer is "neither". There are places where one is more suited than the other, but if one were clearly superior, both methods wouldn't be used.

One important issue in selecting a switching method is whether the network medium is *shared* or *dedicated*. Your phone line can be used for establishing a circuit because you are the only one who can use it—assuming you can keep that pesky wife/husband/child/sister/brother/father/mother off the phone.

However, this doesn't work well in LANs, which typically use a single shared medium and baseband signaling. If two devices were to establish a connection, they would "lock out" all the other devices for a long period of time. It makes more sense to chop the data into small pieces and send them one at a time. Then, if two other devices want to communicate, *their* packets can be interspersed and everyone can share the network.

The ability to have many devices communicate simultaneously without dedicated data paths is one reason why packet switching is becoming predominant today. However, there are some disadvantages of packet switching compared to circuit switching. One is that since all data does not take the same, predictable path between devices, it is possible that some pieces of data may get lost in transit, or show up in the incorrect order. In some situations this does not matter, while in others it is very important indeed.

While the theoretical difference between circuit and packet switching is pretty clear-cut, understanding how they are used is a bit more complicated. One of the major issues is that in modern networks, they are often combined. For example, suppose you connect to the Internet using a dial-up modem. You will be using IP datagrams (packets) to carry higher-layer data, but it will be over the circuit-switched telephone network. Yet the data may be sent over the telephone system in digital packetized form. So in some ways, both circuit switching and packet switching are being used concurrently.

Another issue is the relationship between circuit and packet switching, and whether a technology is connection-oriented or connectionless. The two concepts are related but not the same; the next topic discusses this in much more detail.

 

**Note:** Note that the word “packet” is only one of several terms that are used to refer to messages that are sent over a network. Other terms you will encounter include frame, datagram, cell and segment.

## Connection-Oriented and Connectionless Protocols

In the previous topic I described and contrasted networking technologies based on whether or not they use a dedicated path, or *circuit*, over which to send data. Another way in which technologies and protocols are differentiated has to do with whether or not they use *connections* between devices. This issue is closely related to the matter of packet versus circuit switching.

### *Division of Protocols into Connection-Related Categories*

Protocols are divided into two categories based on their use of connections:

- **Connection-Oriented Protocols:** These protocols require that a logical connection be established between two devices before transferring data. This is generally accomplished by following a specific set of rules that specify how a connection should be initiated, negotiated, managed and eventually terminated. Usually one device begins by sending a request to open a connection, and the other responds. They pass control information to determine if and how the connection should be set up. If this is successful, data is sent between the devices. When they are finished, the connection is broken.
- **Connectionless Protocols:** These protocols do not establish a connection between devices. As soon as a device has data to send to another, it just sends it.

**Key Concept:** A *connection-oriented* protocol is one where a logical connection is first established between devices prior to data being sent. In a *connectionless* protocol, data is just sent without a connection being created.

### *The Relationship Between Connection Orientation and Circuits*

You can probably immediately see the relationship between the concepts of circuits and connections. Obviously, in order to establish a circuit between two devices, they must also be connected. For this reason, circuit-switched networks are inherently based on connections. This has led to the terms “circuit-switched” and “connection-oriented” being used interchangeably.

However, this is an oversimplification that results due to a common logical fallacy—people make the mistake of thinking that if A implies B, then B implies A, which is like saying that since all apples are fruit, then all fruit are apples. A connection is needed for a circuit, but a circuit is ***not*** a prerequisite for a connection. There are, therefore, protocols that are connection-oriented, while not being predicated on the use of circuit-based networks at all.

These connection-oriented protocols are important because they enable the implementation of applications that require connections, over packet-switched networks that have no inherent sense of a connection. For example, to use the TCP/IP File Transfer Protocol, you want to be able to connect to a server, enter a login and password, and then execute commands to change directories, send or retrieve files, and so on. This requires the establishment of a connection over which commands, replies and data can be passed. Similarly, the Telnet Protocol obviously involves establishing a connection—it lets you remotely use another machine. Yet, both of these work (indirectly) over the IP protocol, which is based on the use of packets, through the principle of layering.

To comprehend the way this works, one must have a basic understanding of the layered nature of modern networking architecture (as I discuss in some detail in the chapter that talks about the OSI Reference Model). Even though packets may be used at lower layers for the mechanics of sending data, a higher-layer protocol can create logical connections through the use of messages sent in those packets.

**Key Concept:** Circuit-switched networking technologies are inherently connection-oriented, but not all connection-oriented technologies use circuit switching. Logical connection-oriented protocols can in fact be implemented on top of packet switching networks to provide higher-layer services to applications that require connections.

### *Connection-Oriented and Connectionless Protocols in TCP/IP*

Looking again at TCP/IP, it has two main protocols that operate at the transport layer of the OSI Reference Model. One is the Transmission Control Protocol (TCP), which is connection-oriented; the other, the User Datagram Protocol (UDP), is connectionless. TCP is used for applications that require the establishment of connections (as well as TCP's other service features), such as FTP; it works using a set of rules, as described earlier, by which a logical connection is negotiated prior to sending data. UDP is used by other applications that don't need connections or other features, but do need the faster performance that UDP can offer by not needing to make such connections before sending data.

Some people consider this to be like a "simulation" of circuit-switching at higher network layers; this is perhaps a bit of a dubious analogy. Even though a TCP connection can be used to send data back and forth between devices, all that data is indeed still being sent as packets; there is no real circuit between the devices. This means that TCP must deal with all the potential pitfalls of packet-switched communication, such as the potential for data

 

loss or receipt of data pieces in the incorrect order. Certainly, the existence of connection-oriented protocols like TCP doesn't obviate the need for circuit switching technologies, though you will get some arguments about that one too. ☺

The principle of layering also means that there are other ways that connection-oriented and connectionless protocols can be combined at different levels of an internetwork. Just as a connection-oriented protocol can be implemented over an inherently connectionless protocol, the reverse is also true: a connectionless protocol can be implemented over a connection-oriented protocol at a lower level. In a preceding example, I talked about Telnet (which requires a connection) running over IP (which is connectionless). In turn, IP can run over a connection-oriented protocol like ATM.

## Messages: Packets, Frames, Datagrams and Cells

Many networking technologies are based on packet switching, which involves the creation of small chunks of data to be sent over a network. Even though the word “packet” appears in the name of this method, the data items sent between networked devices are most generically called *messages*. “Packet” is one of a variety of similar words that are used in different contexts to refer to messages sent from one device to another.

In some cases these different terms can be very useful; simply the type of name used for the message can tell you something about what the message contains. In particular, different message names are usually associated with protocols and technologies operating at specific layers of the OSI Reference Model. Thus, the use of these different names can help clarify discussions that involve multiple protocols operating at different layers.

Unfortunately, these terms can also cause confusion, because they are not always applied in a universal or even consistent manner. Some people are strict about applying particular message designations only to the appropriate technologies where they are normally used, while others use the different terms completely interchangeably. This means that you should be familiar with the different message types and how they are normally used, but be prepared for the unexpected.

### *Common Names For Messages*

The most common terms that are used for messages are the following:

- **Packet:** This term is considered by many to most correctly refer to a message sent by protocols operating at the network layer of the OSI Reference Model. So, you will commonly see people refer to “IP packets”. However, this term is commonly also used to refer generically to any type of message, as I mentioned at the start of this topic.
- **Datagram:** This term is basically synonymous with “packet” and is also used to refer to network layer technologies. It is also often used to refer to a message that is sent at a higher level of the OSI Reference Model (more often than “packet” is).

- **Frame:** This term is most commonly associated with messages that travel at low levels of the OSI Reference Model. In particular, it is most commonly seen used in reference to data link layer messages. It is occasionally also used to refer to physical layer messages, when message formatting is performed by a layer one technology. A frame gets its name from the fact that it is created by taking higher-level packets or datagrams and "framing" them with additional header information needed at the lower level.
- **Cell:** Frames and packets, in general, can be of variable length, depending on their contents; in contrast, a *cell* is most often a message that is fixed in size. For example, the fixed-length, 53-byte messages sent in Asynchronous Transfer Mode (ATM) are called cells. Like frames, cells usually are used by technologies operating at the lower layers of the OSI model.
- **Protocol Data Unit (PDU) and Service Data Unit (SDU):** These are the formal terms used in the OSI Reference to describe protocol messages. A PDU at layer N is a message sent between protocols at layer N. It consists of layer N header information and an encapsulated message from layer N+1, which is called both the *layer N SDU* and the *layer N+1 PDU*. After you stop scratching your head, see the topic on OSI model data encapsulation for a discussion of this that may actually make sense. ☺

I should also point out that there are certain protocols that use unusual names to refer to their messages, which aren't used elsewhere in the world of networking. One prominent example is the Transmission Control Protocol (TCP), which calls its messages *segments*.

**Key Concept:** Communication between devices on packet-switched networks is based on in items most generically called *messages*. These pieces of information also go by other names such as *packets*, *datagrams*, *frames* and *cells*, which often correspond to protocols at particular layers of the OSI Reference Model. The formal OSI terms for messages are *protocol data unit (PDU)* and *service data unit (SDU)*.

### *Message Terminology in this Guide*

As for this Guide and its use of these terms, I have made a specific effort not to imply anything about the nature of a message solely based on the name it uses, but I do to follow the most common name used for a particular technology. For example, messages sent over Ethernet are almost always called Ethernet frames—they are not generally called Ethernet datagrams, for example. However, I do not structure discussions so that the type of name used for a message is the only way to determine what sort of message it is.

# Message Formatting: Headers, Payloads and Footers

*Messages* are the structures used to send information over networks. They vary greatly from one protocol or technology to the next in how they are used, and as I described in the previous topic, they are also called by many different names. Shakespeare had the right idea about names, however. The most important way that messages differ is not in what they are called but in terms of their *content*.

Every protocol uses a special *formatting method* that determines the structure of the messages it employs. Obviously, a message that is intended to connect a Web server and a Web browser is going to be quite different from one that connects two Ethernet cards at a low level. This is why I separately describe the formats of dozens of different protocol messages in various parts of this Guide.

### *Fundamental Message Elements*

While the format of a particular message type depends entirely on the nature of the technology that uses it, messages on the whole tend to follow a fairly uniform overall structure. In generic terms, each message contains the following three basic elements (see Figure 3):

- **Header:** Information that is placed before the actual data. The header normally contains a small number of bytes of control information, which is used to communicate important facts about the data that the message contains and how it is to be interpreted and used. It serves as the communication and control link between protocol elements on different devices.
- **Data:** The actual data to be transmitted, often called the *payload* of the message (metaphorically borrowing a term from the space industry!) Most messages contain some data of one form or another, but some actually contain none: they are used only for control and communication purposes. For example, these may be used to set up or terminate a logical connection before data is sent.
- **Footer:** Information that is placed after the data. There is no real difference between the header and the footer, as both generally contain control fields. The term *trailer* is also sometimes used.

Since the header and footer can both contain control and information fields, you might rightly wonder what the point is of having a separate footer anyway. One reason is that some types of control information are calculated using the values of the data itself. In some cases, it is more efficient to perform this computation as the data payload is being sent, and then transmit the result after the payload in a footer. A good example of a field often found in a footer is redundancy data, such as a CRC code, that can be used for error detection by the receiving device. Footers are most often associated with lower-layer protocols, especially at the data link layer of the OSI Reference Model.

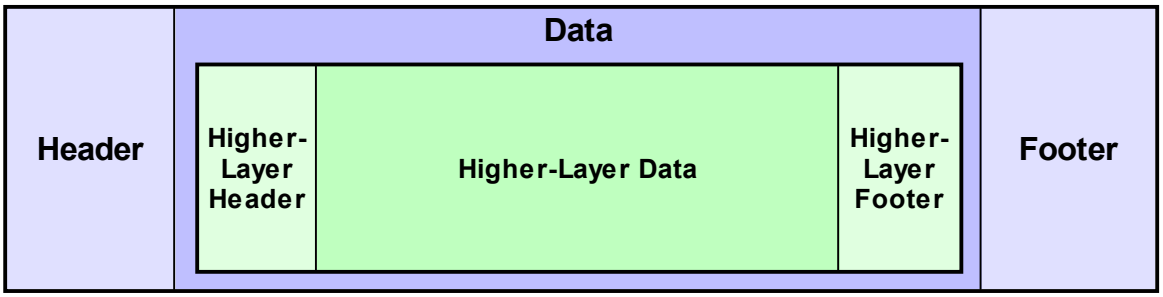


**Figure 3: Network Message Formatting**

In the most general of terms, a message consists of a *data payload* to be communicated, bracketed by a set of *header* and *footer* fields. The data of any particular message sent in a networking protocol will itself contain an encapsulated higher-layer message containing a header, data, and footer. This “nesting” can occur many times as data is passed down a protocol stack. The header is found in most protocol messages; the footer only in some.

**Key Concept:** The general format of a networking message consists of a *header*, followed by the *data* or *payload* of the message, followed optionally by a *footer*. Header and footer information is functionally the same except for position in the message; footer fields are only sometimes used, especially in cases where the data in the field is calculated based on the values of the data being transmitted.

### *Interpretation of Message Elements*

Generally speaking, any particular protocol is only concerned with its *own* header (and footer, if present). It doesn't care much about what is in the data portion of the message, just as a delivery person only worries about driving the truck and not so much on what it contains. At the beginning of that data will normally be the headers of other protocols that were used higher up in the protocol stack; this too is shown in Figure 3. In the OSI Reference Model, a message handled by a particular protocol is said to be its *protocol data unit* or *PDU*; the data it carries in its payload is its *service data unit* or *SDU*. The SDU of a lower-layer protocol is usually a PDU of a higher-layer protocol. The discussion of data encapsulation contains a full explanation of this important concept.

## Message Addressing and Transmission Methods: Unicast, Broadcast and Multicast Messages

In a networking technology that uses messages to send data, there are a number of tasks that must be undertaken in order to successfully transmit the data from one place to another. One is simply the *addressing* of the message—putting an address on it so that the system knows where it is supposed to go. Another is *transmitting* the message, which is of course sending it to its intended recipient.

There are several different ways of addressing and transmitting a message over a network, One way in which messages are differentiated is in how they are addressed, and to how many recipients. Which method is used depends on what the function of the message is, and also on whether or not the sender knows specifically whom they are trying to contact, or only generally.

### *Message Transmission Methods*

To help explain these different methods, I will use a real-world analogy. Consider a social function with 300 people that is being held in a large hall. These people are mingling and are having different conversations. There are different kinds of messages that may need to be sent in this setting, much as is the case with networks.

Bearing this analogy in mind, consider these three kinds of message transmissions, which are illustrated in Figure 4:

- **Unicast Messages:** These are messages that are sent from one device to another device; they are not intended for others. If you have a friend at this social event, this is the equivalent of pulling him or her aside for a private conversation. Of course, there is still the possibility of someone else at the event overhearing your conversation—or even eavesdropping on it. The same is true in networking as well—addressing a message to a particular computer doesn't guarantee that others won't also read it, just that they normally will not do so.
- **Broadcast Messages:** As the name suggests, these messages are sent to every device on a network. They are used when a piece of information actually needs communicating to everyone on the network, or used when the sending station needs to send to just one recipient, but doesn't know its address.

  For example, suppose a new arrival at the social gathering saw a blue sedan with New Hampshire plates in the parking lot that had its lights left on. He of course does not know whose car this is. The best way to communicate this information is to broadcast it by having the host make an announcement that will be heard by all, including the vehicle's owner. In networks, broadcast messages are used for a variety of purposes, including finding the locations of particular stations or the devices that manage different services.
- **Multicast Messages:** These are a compromise between the previous two types: they are sent to a group of stations that meet a particular set of criteria. These stations are usually related to each other in some way, such as serving a common function, or being set up into a particular *multicast group*. (Note that one can also consider broadcast messages to be a special case of multicast, where the group is "everyone".

  Back to our analogy: this would be somewhat like a group of friends who go to this large social hall and then stay together in a small discussion group—or perhaps use radios to talk to each other from a distance. Multicasting requires special techniques that make clear who is in the intended group of recipients.

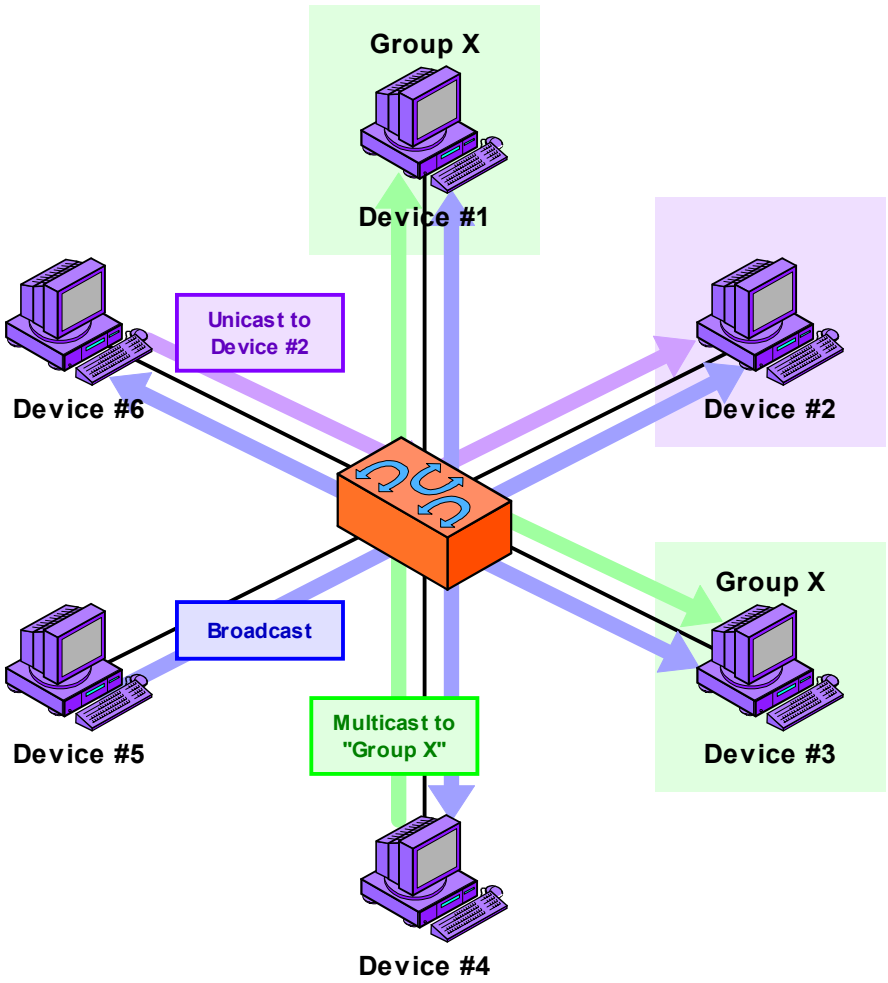


**Figure 4: Unicast, Multicast and Broadcast Message Addressing and Transmission**
The three basic type of addressing and message delivery in networking are illustrated in this simplified local area network. Device #6 is sending a *unicast* message to #2 shown in purple. Device #4 is sending a *multicast* message to multicast group "X". In this case, that group includes devices #1 and #3, shown in green. Finally, Device #5 is sending a *broadcast* message, which goes to all other devices on the LAN.

### *Message Addressing Methods*

Since the transmission methods above differ based on how many and which devices receive the transmission, they are tied directly to the methods used for addressing:

- **Unicast Addressing:** Unicast delivery requires that a message be addressed to a specific recipient. This is the most common type of messaging, so this addressing capability is present in almost all protocols.
- **Broadcast Addressing:** Broadcasts are normally implemented via a special address that is reserved for that function. Whenever devices see a message sent to that address, they all interprets it as meaning "this message goes to everyone".

- **Multicast Addressing:** Multicasts are the most complex type of message because they require a means of identifying a set of specific devices to receive a message. It is often necessary to create several such groups, which may or may not partially overlap in their membership. Some mechanism is needed to manage which devices are in which groups.

**Key Concept:** Three basic methods are used to address and transmit data between networked devices. A *unicast* transmission goes from one device to exactly one other; this is the "normal" method used for most message transactions. A *broadcast* transmission is sent from one device to all connected devices on a network. A *multicast* transmission is addressed and sent to a select group of devices.

**Note:** A new type of message addressing method was defined as part of IP version 6: the *anycast* message. This term identifies a message that should be sent to the closest member of a group of devices. The topic on IPv6 multicast and anycast addressing describes this type of addressing and transmission.

Finally, one special case in the field of addressing is worth mentioning. In some networks or links, only two devices are connected together, forming what is often called a *point-to-point network*. In this situation, everything sent by one device is implicitly intended for the other, and vice-versa. Thus, no addressing of messages on a point-to-point link is strictly necessary.

## Network Structural Models and Client/Server and Peer-to-Peer Networking

I mentioned in my discussion of the advantages of networking that networks are normally set up for two primary purposes: *connectivity* and *sharing*. If you have a network with a number of different machines on it, each computer can interact with the hardware and software of the others, to enable a variety of tasks to be performed. How precisely this is done depends to a large degree on the overall design of the network.

### *Resource Sharing Roles and Structural Models*

One very important issue in network design is how to configure the network for the sharing of resources. Specifically, the network designer must decide whether or not to dedicate resource management functions to the devices that constitute it. In some networks, all devices are treated equal in this regard, while in others, each computer is responsible for a particular job in the overall function of providing services. In this latter arrangement, the devices are sometimes said to have *roles*, somewhat like actors in a play.

Two common terms are used to describe these different approaches to setting up a network, sometimes called choosing a *structural model*.

### Peer-to-Peer Networking

In a strict peer-to-peer networking setup, every computer is an equal, a *peer* in the network. Each machine can have resources that are shared with any other machine. There is no assigned role for any particular device, and each of the devices usually runs similar software. Any device can and will send requests to any other, as illustrated in Figure 5.

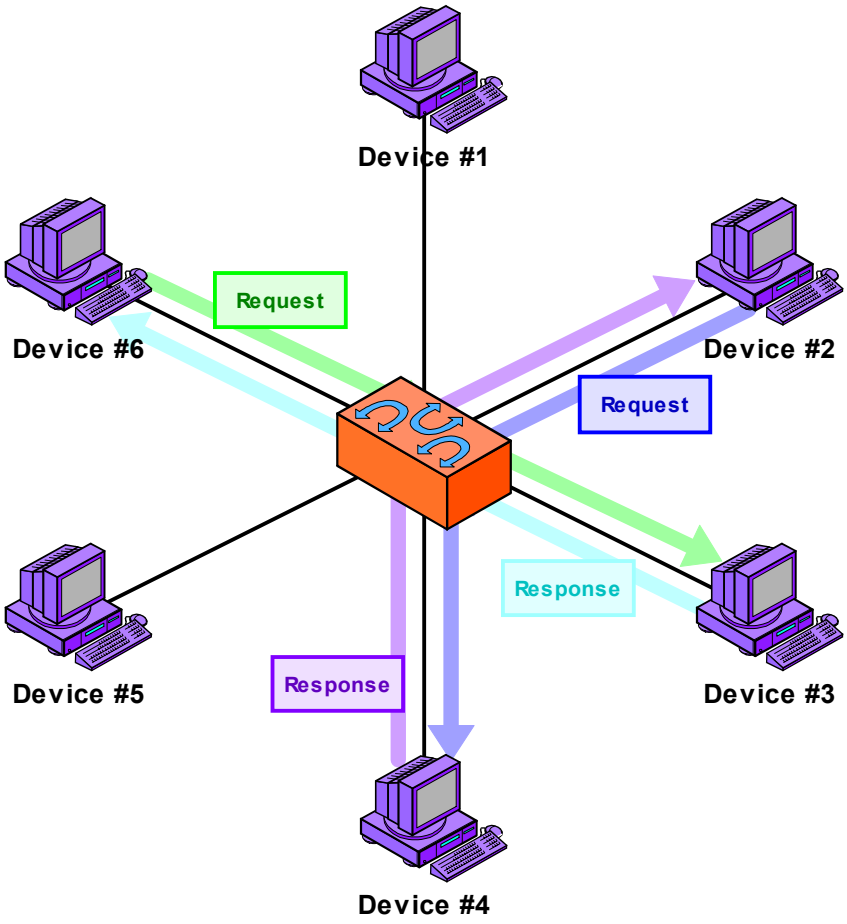


**Figure 5: Peer-to-Peer Networking**

In this model, each device on the network is treated as a peer, or equal. Each device can send requests and responses, and none are specifically designated as performing a particular role. This model is more often used in very small networks. Contrast to Figure 6.

### Client/Server Networking

In this design, a small number of computers are designated as centralized *servers* and given the task of providing services to a larger number of user machines called *clients*. The servers are usually powerful computers with a lot of memory and storage space, and fast network connections. The clients are typically smaller, “regular” computers like PCs, optimized for human use.

The term “client/server” also frequently refers to protocols and software, which are designed with matching, complementary components. Usually, server software runs on server hardware, and client software is used on client computers that connect to those servers. Most of the interaction on the network is between client and server, and not between clients, as shown in Figure 6. Server software is designed to efficiently respond to requests, while client software provides the interface to the human users of the network.

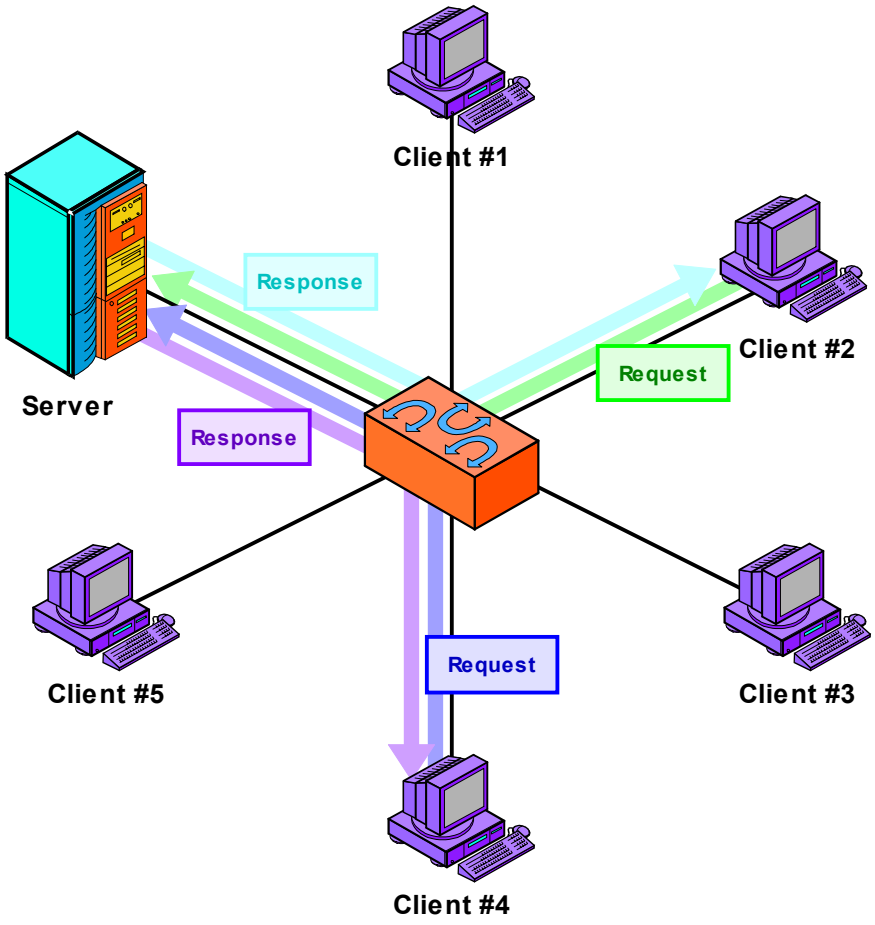


**Figure 6: Client/Server Networking**

In the client/server model, a small number of devices are designated as servers and equipped with special hardware and software that allows them to more efficiently interact simultaneously with multiple client machines. While the clients can still interact with each other, most of the time they send requests of various sorts to the server, and the server sends back responses to them. Contrast this to the peer-to-peer networking example in Figure 5.

**Key Concept:** Networks are usually configured to share resources using one of two basic *structural models*. In a *peer-to-peer network*, each device is an equal and none are assigned particular jobs. In a *client/server network*, however, devices are assigned particular roles—a small number of powerful computers are set up as *servers* and respond to requests from the other devices, which are *clients*. Client/server computing also refers to the interaction between complementary protocol elements and software programs, and is rising in popularity due to its prevalence in TCP/IP and Internet applications.

### *Comparing Client/Server and Peer-to-Peer Networking*

The choice of client/server or peer-to-peer is another where there is no “right answer” in this regard. Which should be used depends entirely on the needs of the particular network.

Peer-to-peer networking has primary advantages of simplicity and low cost, which means it has traditionally been used on small networks. Client/server networking provides advantages in the areas of performance, scalability, security and reliability, but is more complicated and expensive to set it up. This makes it better-suited to larger networks. Over time, however, there has been a steady evolution towards client/server networking, even on smaller networks. Many years ago it was common to see even networks with 20 to 50 machines using the peer-to-peer model; today, even networks with only a half-dozen machines sometimes are set up in a client/server mode because of the advantages of centralized resource serving.

The rise in popularity of client/server networking is ironic because in some ways, it is actually a throwback to the days of large mainframes decades ago. A mainframe with attached terminals can be thought of as a client/server network with the mainframe itself being the server and the terminals being clients. This analogy is not perfect, of course, because modern client computers do a lot more work than dumb terminals do on mainframes.

One of the reasons why the client/server structural model is becoming dominant is that it is the primary model used by the world’s largest network: the Internet. Client/server architecture is the basis for most TCP/IP protocols and services. For example, the term “Web browser” is really another name for a “Web client”, and a “Web site” is really a “Web server”.

**Related Information:** For more information on client/server computing, I recommend you read the topic devoted to TCP/IP client/server operation. That topic also contains a very relevant exposition on the different meanings of the terms “client” and “server” in hardware, software and transactional contexts.