The **OSI** (**Open Systems Interconnection**) **Model** was created to standardize the language used to describe networking protocols. It defines the manner in which systems communicate with one another using abstraction layers.

Each layer communicates with the layer directly above and below.

Table 2.1: The OSI Model

Layer	Name	Description
7	Application	Most networking stacks have a concept of the application layer.
6	Presentation	Most networking stacks combine the presentation layer in layers four five or seven.
5	Session	Sometimes used by different stacks, often combined into other layers (seven or six).
4	Transport	Most networking stacks have a concept of the transport layer.
3	Network	Most networking stacks have a concept of the network layer.
2	Data Link	Most networking stacks have a concept of the data link layer.
1	Physical	Most networking stacks have a concept of the physical layer

There are other models which are used to talk about networking. The most popular networking stack on the Internet today is the **Internet Protocol Suite**.

The **Internet Protocol Suite** can be described using a subset of the **OSI Model**.

The **Application Layer** is the most well-known. This layer is at the top of the stack and deals with the protocols which make a global communications network function.

Some of the common protocols which exist in the **Application Layer** are:

- **HTTP**: **H**ypertext **T**ransfer **P**rotocol.
- **SMTP**: **S**imple **M**ail **T**ransfer **P**rotocol.
- **DNS**: **D**omain **N**ame **S**ystem.
- **FTP**: **F**ile **T**ransfer **P**rotocol.
- **DHCP**: **D**ynamic **H**ost **C**onfiguration **P**rotocol.

Protocols at this level are the most familiar to users. They are defined by **RFC1123**. To learn more, go to https://tools.ietf.org/html/rfc1123.

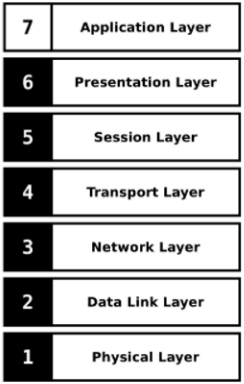

Figure 2.1: The OSI Model - The Application Layer (7)

The **Presentation Layer** is commonly rolled up into a different layer. This layer deals with the formatting of data (e.g. conversion of **EBCDIC** to **ASCII**).

For example, the HTTP protocol (an **Application Layer** protocol) has methods for converting character encoding. In other words, this **Presentation Layer** step happens at the **Application Layer**.

Many networking stacks and protocols make no distinction between layers 6 and 7.

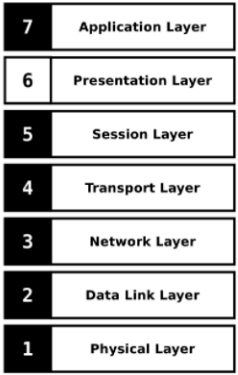

Figure 2.2: The OSI Model - The Presentation Layer (6)

The **Session Layer** deals with managing of session data. It creates a semi-permanent connection, which is then used for communications.

Many of the **RPC**-type protocols depend on this layer:

- **NetBIOS**: **N**etwork **B**asic **I**nput **O**utput **S**ystem.
- **RPC**: **R**emote **P**rocedure **C**all.
- **PPTP**: **P**oint to **P**oint **T**unneling **P**rotocol.

This layer is used by protocols which need reliable sessions, such as videoconferencing and **SOCKS** proxy.

If an established connection is lost or disrupted, this layer may try to recover the connection.

If a connection is not used for a long time, the session layer may close and then reopen it.

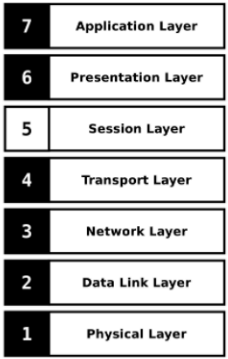

Figure 2.3: The OSI Model - The Session Layer (5)

The **Transport Layer** is responsible for the end-to-end communication protocols. Data is properly multiplexed by defining the source and destination port numbers. This layer also deals with reliability by adding check sums, doing request repeats, and avoiding congestion.

Some of the common protocols in the **Transport Layer** are:

- **TCP**: **T**ransmission **C**ontrol **P**rotocol:
  It is the main component of the TCP/IP (**Internet Protocol Suite**) stack.
- **UDP**: **U**ser **D**atagram **P**rotocol:
  This is another popular component of the **Internet Protocol Suite** stack.
- **SCTP**: **S**tream **C**ontrol **T**ransmission **P**rotocol.
- It uses port numbers to allow for connection multiplexing.

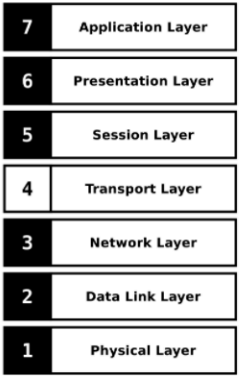

Figure 2.4: The OSI Model - The Transport Layer (4)

Transport layer protocols use ports to distinguish between different types of traffic or to do multiplexing. The ports are classed three different ways:

- **Well-Known Ports** (0-1023) - assigned by the **IANA** (Internet **A**ssigned **N**umbers **A**uthority).
  They usually require super-user privilege to be bound. Some of the well-known ports are: **22 TCP: SSH**; **25 TCP: SMTP**; **80 TCP: HTTP**; **443 TCP: HTTPS**.
- **Registered Ports** (1024-49151) - assigned by the **IANA**.
  They can be bound on most systems by non-super-user privilege. Some of the registered ports are: **1194 TCP/UDP: OpenVPN**; **1293 TCP/UDP: IPSec**; **1433 TCP: MSSQL Server**.
- **Dynamic** or **Ephemeral** Ports (49152-65535).
  The **Ephemeral** ports are used as source ports for the client-side of a TCP or UDP connection. You can also use the **Ephemeral** ports for a temporary or non-root service.

TCP is useful when data integrity, ordered delivery, and reliability are important. It is the backbone to many of the most popular protocols.

UDP is useful when transmission speed is important and the integrity of the data isn't as important, or is managed by an above layer.

Table 2.2: TCP vs UDP Details

	TCP	UDP
Connection-Oriented	YES	NO
Reliable	YES	NO
Ordered Delivery	YES	NO
Checksums	YES	Optional
Flow Control	YES	NO
Congestion Avoidance	YES	NO
NAT Friendly	YES	YES
ECC	YES	YES
Header Size	20-60 bytes	8 bytes

The **Network Layer** is all about routing packets. This layer is responsible for getting the packets to the next point in the path to the destination. So, this layer deals with routing and packet forwarding, as well as with managing the quality of service.

In many cases, the final destination is not adjacent to this machine, so the packets are routed based on the local routing table information.

It is connectionless; connection tracking can happen at the layers above.

Many routing and control protocols live at this layer, such as:

- **IP**: Internet **P**rotocol.
- **OSPF**: **O**pen **S**hortest **P**ath **F**irst.
- **IGRP**: Interior **G**ateway **R**outing **P**rotocol.
- **ICMP**: Internet **C**ontrol **M**essage **P**rotocol.

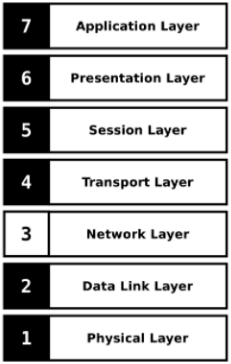

Figure 2.5: The OSI Model - The Network Layer (3)

Originally the datagram service for TCP, the **Internet Protocol** now transfers many different higher level protocols.

The Internet Protocol has two main functions:

- Addressing.
- Fragmentation.

The addressing function examines the address on the incoming packet and decides if the datagram (packet) is for the local system or for another system. If the address indicates the datagram is for the local system, the headers are removed and the datagram is passed up to the next layer in the protocol stack. If the address indicates the datagram is for another machine, then it is passed to the next system in the direction of the final destination.

The fragmentation component will split and re-assemble the packets if the path to the next system uses a smaller transmission unit size.

There are two major versions of **IP**:

- IPv4.
- IPv6.

For additional information, please see https://www.ietf.org/rfc/rfc791.txt.

IPv4 was the first major **Internet Protocol** version:

- It is the most used protocol on the **Internet**.
- The 32bit address size allows for **4,294,967,296** possible addresses.
- The address space was exhausted on January 31, 2011.
- The last allocation of blocks was given out on February 3, 2011.

The **IPv4** address space exhaustion has been a concern for some time. There are different solutions for mitigating the problem:

- The move from **Classed** networks to **CIDR**.
- The invention of **NAT**.
- The move to **IPv6**.

2.13. IPv6

IPv6 is the successor to **IPv4**.

example.com - `2001:500:88:200::10`

IPv6 has a 128bit address size that allows for **3.4 x 10^{38}** possible addresses. **IPv6** was designed to deal with the exhaustion of **IPv4** addresses and other **IPv4** shortcomings:

- Expanded addresses capabilities.
- Header format simplification.
- Improved support for extensions and options.
- Flow labeling capabilities.

To learn more, please visit https://ietf.org/rfc/rfc2460.txt.

IP addresses have two parts:

- The Network Part.
- The Host Part.

They are distinguished by using a **Netmask**, a bitmask defining which part of an IP address is the network part.

Table 2.3: IP Address Parts

IP Addr Dec	10	20	0	100
Netmask Dec	255	255	0	0
IP Addr Bin	00001010	00010100	00000000	01100100
Netmask Bin	11111111	11111111	00000000	00000000
Network Part Bin	00001010	00010100		
Host Part Bin			00000000	00110010

IP networks can be broken into smaller pieces by using a subnet. The example below is breaking a network with a netmask of `255.255.0.0` into a smaller subnet with a netmask of `255.255.255.0`.

Table 2.4: IP Subnetting

IP Addr Dec	10	20	0	100
Netmask Dec	255	255	0	0
IP Addr Bin	00001010	00010100	00000000	01100100
Netmask Bin	11111111	11111111	00000000	00000000
Subnet Bin			11111111	
Network Part Bin	00001010	00010100		
Subnet Part Bin			00000000	
Host Part Bin				00110010

2.16. IP Network Classes

Originally, the **IPv4** addresses were broken into classes:

- **Class A -** `0.0.0.0/255.0.0.0`
- **Class B -** `128.0.0.0/255.255.0.0`
- **Class C -** `192.0.0.0/255.255.255.0`

These original classes of networks and subnets did not scale well. Networks which did not fit in a class B were often given a class A. Wasted IP addresses led to the creation of **CIDR** (**C**lassless **I**nter-**D**omain **R**outing). **CIDR** uses a numbered bitmask instead of the class bitmask.

There are two types of netmasks:

- **Classed Network Netmasks.**

Table 2.5: Classed Network Netmasks

Class A Dec	255	0	0	0
Class A Bin	11111111	00000000	00000000	00000000
Class B Dec	255	255	0	0
Class B Bin	11111111	11111111	00000000	00000000
Class C Dec	255	255	255	0
Class C Bin	11111111	11111111	11111111	00000000

- **CIDR Network Netmasks** are more flexible, and they do not have to end on "nibble" boundaries.

Table 2.6: CIDR Network Netmasks

CIDR /18 Dec	255	255	192	0
CIDR /18 Bin	11111111	11111111	11000000	00000000

IP packets can only reach from end to end if they are routed properly.

Network routers inspect packet headers, and make routing decisions based on the destination address.

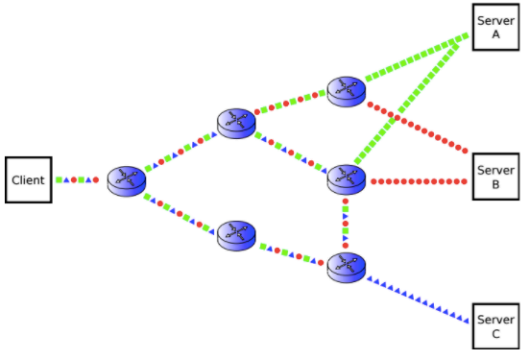

Figure 2.6: IP Routing

IP management tools include **ifconfig** and **ip**.

ifconfig is part of the **net-tools** package. This historical tool is not very flexible.

ip is part of the **iproute** package. It is the new default tool for many distributions and manages Layer 2 and 3 settings.

Setting an address with **ifconfig**:

```
# ifconfig eth0 10.20.0.100 netmask 255.255.0.0
```

Setting an address with **ip**:

```
# ip addr add 10.20.0.100/16 dev eth0
```

The **Data Link Layer** deals with transferring data between network nodes:

- Adjacent nodes in a **W**ide **A**rea **N**etwork (**WAN**).
- Nodes on the same **L**ocal **A**rea **N**etwork (**LAN**) segment.

Some of the common **Data Link Layer** protocols are:

- **Ethernet.**
- **ARP**: **A**ddress **R**esolution **P**rotocol.
- **PPP**: **P**oint to **P**oint **P**rotocol.
- **STP**: **S**panning **T**ree **P**rotocol.

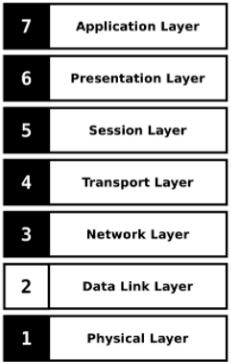

Figure 2.7: The OSI Model - The Data Link Layer (2)

The **Physical Layer** is the lowest possible layer and deals with the actual physical transfer of information. This layer deals with transferring bits over a physical medium:

- Electric pulses over copper cables.
- Laser pulses over fiber optic cables.
- Frequency modulations over radio waves.
- Scraps of paper over carrier pigeons (to learn more, go to http://tools.ietf.org/html/rfc1149).

There are various different protocols, hardware types, and standards defined for different types of physical networks (commonly referred to as **PHY**s):

- **IEEE 802.3**: Copper or fiber connections.
- **IEEE 802.11**: Wireless (Wi-Fi) connections.
- **Bluetooth**: Wireless connections.
- **USB**: Copper connections.
- **RS232**: Copper serial connections.

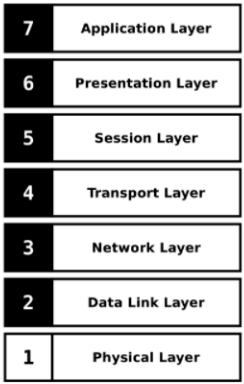

Figure 2.8: The OSI Model - The Physical Layer (1)

Smaller network segments are sometimes referred to as **collision domains**, since there can be only one transmitter on the physical medium at a time. This is common in hub-based networks. Modern switch technology removes the restriction of one at a time, but the concept of **broadcast domains** still exists.

Local **A**rea **N**etworks (**LAN**s) are smaller, locally-connected networks, connected at Layer 2 by the same series of switches or hubs. The node-to-node communication happens at Layer 3, using the same network. Layer 2 to Layer 3 associations may use Layer 2 broadcasts and the ARP and RARP protocols to associate MAC addresses with IP addresses.

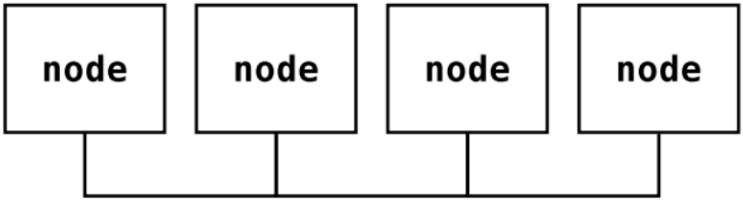

Figure 2.9: Local Area Network (LAN)

A network bridge or repeater accepts packets on one side of the bridge and passes them through to the other side of the bridge. The effect is bi-directional:

- It is a combination of two or more networks at Layer 2.
- Bridged networks communicate as a single network.

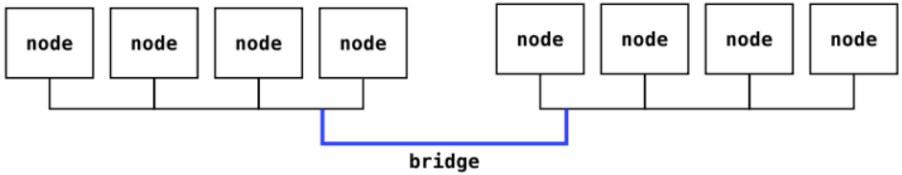

Figure 2.10: Bridged Network

A **W**ide **A**rea **N**etwork (**WAN**) is a large and geographically diverse network. It is connected at Layer 3 from node to node.

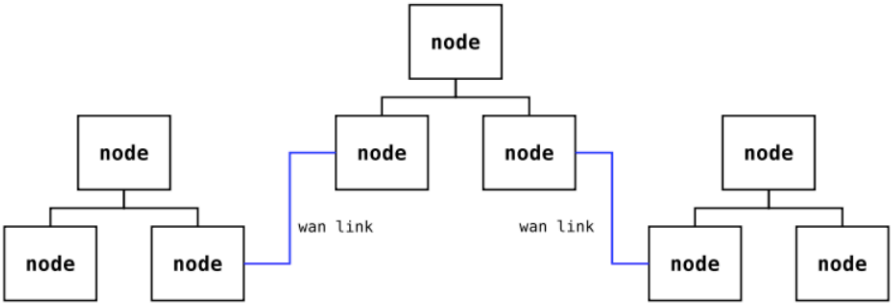

Figure 2.11: Wide Area Network (WAN)

A **V**irtual **L**ocal **A**rea **N**etwork (**VLAN**) is a method for combining two or more separated **LAN**s to appear as the same **LAN**. It is also a method for securing two or more **LAN**s from each other on the same set of switches.

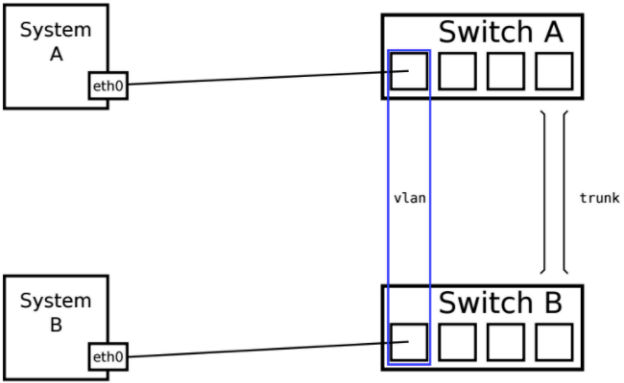

Figure 2.12: Virtual Local Area Network (VLAN)

The **D**omain **N**ame **S**ystem (**DNS**) is a distributed, hierarchical database for converting **DNS** names into **IP** addresses. The key-value store can be used for more than just **IP** address information. The **DNS** protocol runs in two different modes:

- Recursive with caching mode.
- Authoritative mode.

When a network node makes a **DNS** query, it most often makes that query against a recursive, caching server. That recursive, caching server will then make a recursive query through the **DNS** database, until it comes to an authoritative server. The authoritative server will then send the answer for the query.

2.27. DNS Database

The DNS database consists of a tree-like, key-value store. The database is broken into tree nodes called **domains**. These **domains** are managed as part of a **zone**. **Zones** are the area of the namespace managed by authoritative server(s).

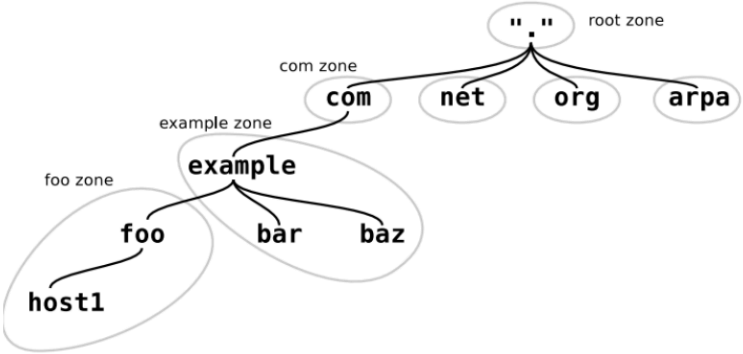

Figure 2.13: The DNS Tree

2.28.a. Recursive DNS Query

A theoretical recursive **DNS** query for **`host1.foo.example.com.`** would take the following steps:

1. The Client makes a recursive request to the caching nameserver it has configured.
2. The caching nameserver makes a query for **`host1.foo.example.com.`** to the root (".") zone nameserver.
3. The root (".") zone nameserver points to the nameserver for the **`com.`** zone.
4. The caching nameserver makes a query for **`host1.foo.example.com.`** to the nameserver for the **`com.`** zone.
5. The **`com.`** zone nameserver sends a response that points to the nameserver for the **`example.com.`** zone.
6. The caching nameserver makes a query for **`host1.foo.example.com.`** to the nameserver for the **`example.com.`** zone.
7. The **`example.com.`** nameserver sends a response that points to the nameserver for the **`foo.example.com.`** zone.
8. The caching nameserver makes a query for **`host1.foo.example.com.`** to the nameserver for the **`foo.example.com.`** zone.
9. The **`foo.example.com.`** nameserver responds authoritatively for the address **`host1.foo.example.com.`** to the caching nameserver.

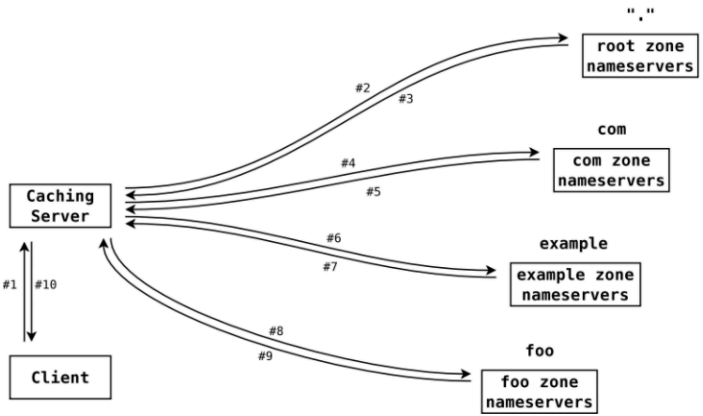

Figure 2.14: A DNS Query

DNS tools include servers and clients.

Among the servers included is:

- **BIND** (**B**erkeley **I**nternet **N**ame **D**aemon).

Among the clients included are:

- **dig**: lots of info for debugging.
- **host**: simple interface for **DNS** queries.
- **nslookup**: deprecated tool.

If you are familiar with **nslookup** ,you should learn one of the other tools. The **I**nternet **S**ystems **C**onsortium (**ISC**), who maintained **nslookup**, has deprecated the tool in favor of **dig** and **host**.

Many system services are provided by daemons which run at the **Application Layer**. Different initialization systems have been built to ensure that system daemons are running. Some of these systems are:

- **init** (commonly referred to as **SYSV init**).
- **Upstart.**
- **Systemd.**

2.31. System V Init Scripts

System V init scripts are shell scripts used to start services. Using serial execution, dependency management is done serially. The execution order and dependencies for **SYSV init** scripts are managed by renaming scripts to run in proper order. Long boot times are due to serial execution.

2.32. BSD Init Scripts

BSD init scripts are shell scripts used to start services. They use serial execution and the dependency management is done automatically.

The execution order and dependencies for **BSD init** scripts are managed with dependency tags in the scripts themselves. The command **rcorder** reads these tags and prints out the proper order.

As with **SYSV init**, the boot system is still done in series, leading to long boot times.

To counteract the problems with historical **init** systems, different systems have been developed.

- **Systemd:**
  - **Fedora** (Since **Fedora** 15).
  - **Mandriva** (Since 2011).
  - **OpenSUSE** (Since 12.1).
  - **RHEL** (Since 7.0).
  - **Debian** (Since 8.0).
  - **Ubuntu** (Since 15-04).
- **Upstart:**
  - **Ubuntu** (since 9.10).
  - **WebOS.**
  - **RHEL** (**RHEL** 6.X only).
  - **Fedora** (**Fedora** 9 to **Fedora** 14).

To start system services, you must manually run the daemon.

To manually start a daemon, use:

```
# /usr/sbin/httpd -f /etc/httpd/httpd.conf
```

To start a daemon using a **SYSV init** script, use:

```
# /etc/init.d/httpd start
```

To start a daemon using the **service** command, use:

```
# service httpd start
```

(The service name can differ on different distributions.)

To start a daemon using the **OpenSUSE** `rc<COMMAND>` command, use:

```
# rcapache2 start
```

To start a daemon using **upstart**, use:

```
# start apache
```

To manually stop a daemon, use:

```
# kill <PID of daemon>
```

To stop a daemon using a **SYSV init** script, use:

```
# /etc/init.d/httpd stop
```

To stop a daemon using the **service** command, use:

```
# service httpd stop
```

(The service name can differ on different distributions.)

To start a daemon using the **OpenSUSE** `rc<COMMAND>` command, use:

```
# rcapache2 stop
```

To stop a daemon using **upstart**, use:

```
# stop apache
```

To stop a daemon using **systemd**, use:

```
# systemctl stop httpd.service
```

System services must be enabled so they automatically start at boot time.

To manually enable and disable a system by linking its **SYSV init** script, use:

```
# ln -s /etc/init.d/httpd /etc/rc3.d/S85httpd
# rm /etc/rc3.d/S85httpd
# ln -s /etc/init.d/httpd /etc/rc3.d/K25httpd
```

To use the **chkconfig** helper script on **CentOS6** or **OpenSUSE** to enable and disable a service, do:

```
# chkconfig httpd on
# chkconfig httpd off
```

To use the **Ubuntu update-rc.d** helper script to enable a service, do:

```
# update-rc.d apache2 enable 2345
# update-rc.d apache2 disable 2345
```

To enable or disable a service using **systemd**, use:

Some services are not used enough to keep a daemon running full time. The **xinet** daemon was created to manage these transient daemons.

Xinetd is started by using the system-specific **init** system.

xinet listens on the proper port and, when a connection is made, it starts the managing daemon and passes off the connection.

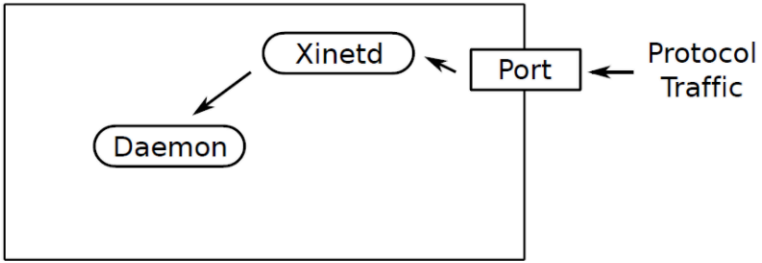

Figure 2.15: Transient System Services

systemd processes the configuration files is predictable and extensible. The common sequence is:

- The vendor-supplied unit file in `/usr/lib/systemd/<servcice>.service` or `/lib/systemd/system/<service>.service` (Ubuntu 16.04).
- Optional or dynamically-created unit files in the `/run/systemd/system` directory.
- Optional user-unit override files in the `/etc/systemd/system` directory.
- Optional user drop-in files in `/etc/systemd/system/<service>.d`.
- There are drop-in directories available in all of the configuration directories, but the most popular drop-in location is in `/etc/systemd/system/<service>.d/`.

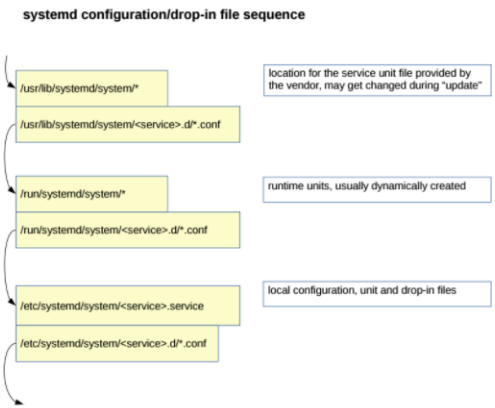

Figure 2.16:Systemd Service Configuration