Contents

[IP classification: 1](#_Toc49655861)

[Subnets : 2](#_Toc49655862)

[How to Subnet: 4](#_Toc49655863)

[Reverse Proxy Server 6](#_Toc49655864)

[Load Balancer: 6](#_Toc49655865)

[DNS Round Robin (rarely used) 7](#_Toc49655866)

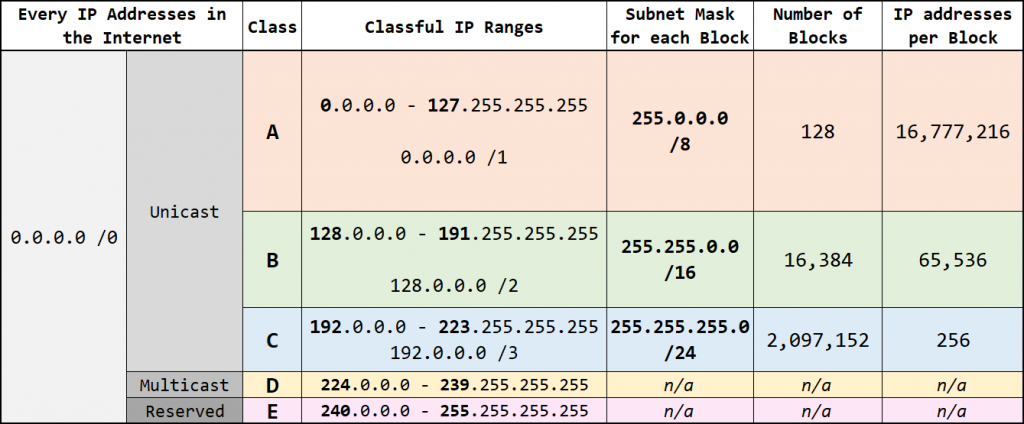
[L3/L4 Load Balancer 7](#_Toc49655867)

[L7 Load Balancer 7](#_Toc49655868)

# IP classification:

IP address are assigned using two strategies

1. Classful: Classful addressing is how the early Internet was formed. IP assignments were given on the Octet boundaries:



However, this led to a lot of wasted IP addresses. If, for instance, you only needed 300 IP addresses, a Class C would not suffice, so you would end up with a Class B and nearly 60,000 IP addresses would be wasted.

Three popular private IP ranges are:

|  |  |  |
| --- | --- | --- |
| Class | IP Range | Purpose |
| A | 10.0.0.0/8 | Used for local communications within a private network |
| B | 172.16.0.0/12 | Used for local communications within a private network |
| C | 192.168.0.0/16 | Used for local communications within a private network |

1. Classless (CIDR)

With **Classless Inter-Domain Routing** (**CIDR**), IP assignments are not limited to the three classes. The whole unicast range (any IP address with a first octet of 0 – 223) can be allocated in any size block. In effect, the whole concept of IP address classes is done away with entirely.

## Subnets :

( <https://www.youtube.com/watch?v=XQ3T14SIlV4> )

**Subnetting allows you to create smaller network (sub networks; 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.

Imagine I want to build a network that will support up to 30 devices in different segments. Without subnetting, I will need four (4) Class C networks to support this design. For example:

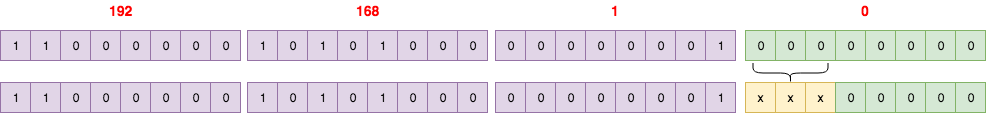
* Network #1: 192.168.1.0
* Network #2: 192.168.2.0
* Network #3: 192.168.3.0
* Network #4: 192.168.4.0

Each of these networks will support 254 IP addresses leading to a wastage of (254 \* 4) – (30 \* 4) IP addresses i.e. 896 IP addresses!

If you look at the design requirement of 30 hosts per network, you will discover that I only need 5 bits in the host ID portion of a Class C network to satisfy my requirement.

Usable Ip addresses

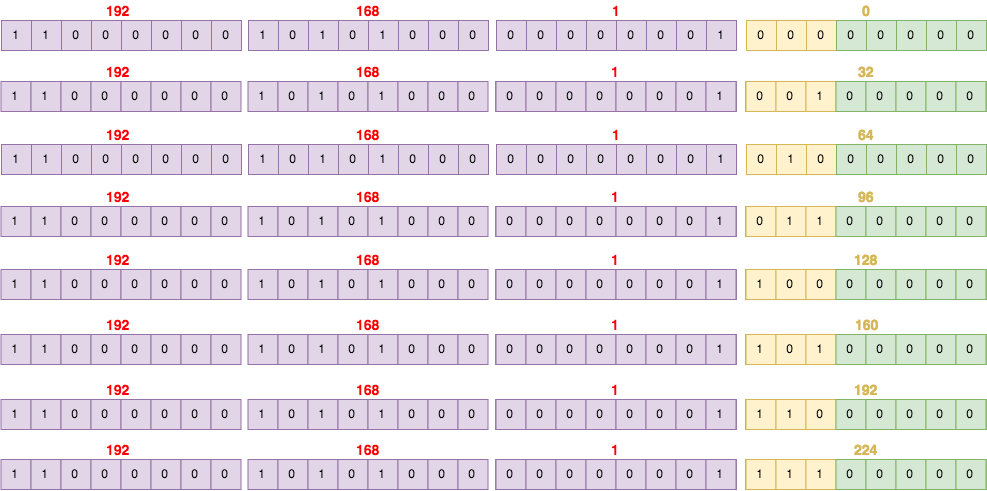
This means I still have 3 bits unused and with subnetting, I can use those three bits to create smaller networks. For this example, let’s take the 192.168.1.0 network:



By borrowing 3 bits, I can create 8 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 probably look weird to you – they look like normal IP addresses. However, looking at them in their binary form makes things clearer:



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.

**A subnet mask is the representation of the network portion of an address** in a Subnet.

**A switch or layer 2 devices looks for subnet mask to route traffic within the subnet or outside it.**

<https://www.youtube.com/watch?v=yLeuGOOrUvo>

<https://www.youtube.com/watch?v=Xb1JA5CIssI>

<https://www.ittsystems.com/introduction-to-subnetting/>

## How to Subnet:

Subnetting problem comes in two parts i.e either you will have to create subnets that can accommodate certain number of devices or directly create X number of subnet.

**Example 1: Directly create X number of subnet**

( <https://www.youtube.com/watch?v=ecCuyq-Wprc> )

Here is the network ID **192.168.4.0/24**, please create **3** separate networks or subnets for a coffee shop: Sunny Cafe.

List each of new **network ID**, **subnet mask**, **host ID range**, **# of usable hosts**, and **broadcast ID**.

Step 1: Identify number of **non-reserved bit** i.e 32-24 = 8

Step 2: Since we need to block that many IP bits so for 3 subnets we would require two bits i.e 2^2 = 4 . While one subnet will never be used but this is the best possible combination.

So **subnet mask: /26**

Now left number of bits would be 6 and hence **no. of usable host will be: 2^6-2 = 62**

Note: Subtracting two from each subnet mask as each subnet has Network ID (The first IP of subnet) and Broadcast ID ( Last IP of subnet) . **Cloud solution normally reserve 5 IPs instead of two. Hence usable will be what is left from reserved IPs in a subnet mask.**

**For multiplication help, remember this table:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 124 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

Step 3: Now for finding host range and network ID and broadcast ID

Since the IP was /24 CIDR, last octet of IP will start from 0 as :

11111111.11111111.11111111.**00000000**

Also each subnet has 64 hosts, Hence network ID becomes 192.168.4.0 and Broadcast ID becomes **192.168.4.63**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Network ID | Broadcast ID | Host range | Subnet Mask | #Usable Hosts |
| 192.168.4.0 | **192.168.4.63** | **192.168.4.1-192.168.4.62** | /26 | 62 |
| 192.168.4.64 | **192.168.4.127** | **192.168.4.65-192.168.4.126** | /26 | 62 |
| 192.168.4.128 | **192.168.4.191** | **192.168.4.129-192.168.4.190** | /26 | 62 |
| 192.168.4.192 | **192.168.4.255** | **192.168.4.193-192.168.4.254** | /26 | 62 |

**Example 2: Suppose I have a class B ID: 172.16.0.0/16 and now we want 4 subnets**

**(** [**https://www.youtube.com/watch?v=wuIdYxaV46Y**](https://www.youtube.com/watch?v=wuIdYxaV46Y) **)**

Step 1 : Identify non reserved bits i.e 32-16 = 16 bit.

Step 2 : Number of bits to be reserved for subnet : 2^2= 4 i.e 2 bits. Hence our **subnet mask will be /18**

Number of usable host per subnet will be 2^14-2 = 16384-2 = **16382** hosts ( Two being reserved for Network and Broadcast ID )

Step 3: Now for finding host range, network ID and broadcast ID

Since the IP cidr is /16 hence the first IP of subnet will start from last two octets or non-reserved bits.

11111111.11111111.00000000.00000000 i.e 172.16.0.0

Also since each subnet has 2^14 hosts that would translate as

|  |  |  |  |
| --- | --- | --- | --- |
| Network Mask | Network ID | Network Mask | Broadcast ID |
| XXXXXXXX.XXXXXXXX.**00**000000.000000 | 172.16.0.0 | XXXXXXXX.XXXXXXXX.00111111.111111 | 172.16.63.255 |
| XXXXXXXX.XXXXXXXX.**01**000000.000000 | 172.16.64.0 (2^6) | XXXXXXXX.XXXXXXXX.01111111.111111 | 172.16.127.255 |
| XXXXXXXX.XXXXXXXX.**10**000000.000000 | 172.16.128.0 (2^7) | XXXXXXXX.XXXXXXXX.10111111.111111 | 172.16.191.255 |
| XXXXXXXX.XXXXXXXX.**11**000000.000000 | 172.16.192.0  (2^7+2^6) | XXXXXXXX.XXXXXXXX.11111111.111111 | 172.16.255.255 |

To summarize:

If one is give /24 type networks use following table for quick translation of Class C subnets :

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Number of Subnets** | **1** | **2** | **4** | **8** | **16** | **32** | **64** | **128** | **256** |
| **Hosts** | **256** | **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |
| **Subnet Mask** | **/24** | **/25** | **26** | **/27** | **/28** | **/29** | **/30** | **/31** | **/32** |

Similarly, if one is give /16 type of network then one can use the following Class B subnets:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Number of Subnets** | **1** | **2** | **4** | **8** | **16** | **32** | **64** | **128** | **256** | **512** | **1024** | **2048** | **4096** | **8192** | **163184** | **32768** | **65536** |
| **Hosts** | **65536** | **32768** | **163184** | **8192** | **4096** | **2028** | **1024** | **512** | **256** | **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |
| **Subnet Mask** | **/16** | **/17** | **/18** | **/19** | **/20** | **/21** | **/22** | **/23** | **/24** | **25** | **/26** | **/27** | **/28** | **/29** | **/30** | **/31** | **/32** |

Note : Azure and AWS reserve 5 IPs per subnet so in that case subtract 5 from number of hosts available per subnet for calculating usable hosts.

Azure reserves 5 IP addresses within each subnet. These are x.x.x.0-x.x.x.3 and the last address of the subnet. x.x.x.1-x.x.x.3 is reserved in each subnet for Azure services.

* x.x.x.0: Network address
* x.x.x.1: Reserved by Azure for the default gateway
* x.x.x.2, x.x.x.3: Reserved by Azure to map the Azure DNS IPs to the VNet space
* x.x.x.255: Network broadcast address

# Reverse Proxy Server

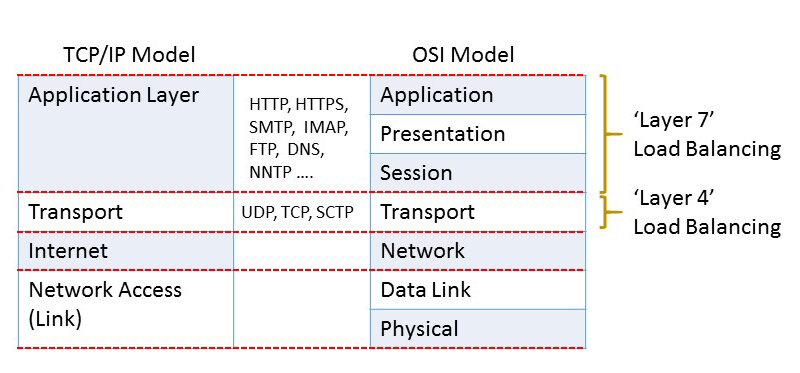
A proxy server is a go‑between or intermediary server that forwards requests for content from multiple clients to different servers across the Internet. A **reverse proxy server** is a type of proxy server that typically sits behind the firewall in a private network and directs client requests to the appropriate backend server. A reverse proxy provides an additional level of abstraction and control to ensure the smooth flow of network traffic between clients and servers.

Common uses for a [reverse proxy server](https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/) include:

* **Load balancing** – A reverse proxy server can act as a “traffic cop,” sitting in front of your backend servers and distributing client requests across a group of servers in a manner that maximizes speed and capacity utilization while ensuring no one server is overloaded, which can degrade performance. If a server goes down, the [load balancer](https://www.nginx.com/solutions/adc/) redirects traffic to the remaining online servers.
* **Web acceleration** – Reverse proxies can compress inbound and outbound data, as well as cache commonly requested content, both of which speed up the flow of traffic between clients and servers. They can also perform additional tasks such as SSL encryption to take load off of your web servers, thereby [boosting their performance](https://www.nginx.com/resources/glossary/web-acceleration/).
* **Security and anonymity** – By intercepting requests headed for your backend servers, a reverse proxy server protects their identities and acts as an additional defence against security attacks. It also ensures that multiple servers can be accessed from a single record locator or URL regardless of the structure of your local area network.

<https://www.nginx.com/resources/glossary/reverse-proxy-server/>

# Load Balancer:



Types of Load Balancer:

## DNS Round Robin (rarely used)

clients get a randomly-ordered list of IP addresses.  
pros: easy to implement and free  
cons: hard to control and not responsive, since DNS cache needs time to expire

## L3/L4 Load Balancer

traffic is routed by IP address and port. L3 is network layer (IP). L4 is session layer (TCP).  
pros: better granularity, simple, responsive.

Today the term “Layer 4 load balancing” most commonly refers to a deployment where the load balancer’s IP address is the one advertised to clients for a web site or service (via DNS, for example). As a result, clients record the load balancer’s address as the destination IP address in their requests.

## L7 Load Balancer

traffic is routed by what is inside the HTTP protocol. L7 is application layer (HTTP).

Layer 7 load balancers base their routing decisions on various characteristics of the HTTP header and on the actual contents of the message, such as the URL, the type of data (text, video, graphics), or information in a cookie.

Taking into consideration so many more aspects of the information being transferred can make [Layer 7 load balancing](https://www.nginx.com/resources/glossary/layer-7-load-balancing) more expensive than Layer 4 in terms of time and required computing power, but it can nevertheless lead to greater overall efficiency. For instance, because a Layer 7 load balancer can determine what type of data (video, text, and so on) a client is requesting, you don’t have to duplicate the same data on all of the load-balanced servers. It uses buffering to offload slow connections from the upstream servers, which improves performance.

**A device that performs Layer 7 load balancing is often referred to as a** [***reverse‑proxy server***](https://www.nginx.com/resources/glossary/reverse-proxy-server/)**.**

<https://www.nginx.com/resources/glossary/layer-7-load-balancing>