Application Delivery Controller (18CS7G2)

Instructor: **Dr. Vishalakshi Prabhu H**, Dept of CSE, RVCE, Bangalore-59

Author of notes: **Dr. Vishalakshi Prabhu H,** Date :24/10/2021

Contents of Unit - 1

Load balancers

Concepts of L4 load balancing, Managing application delivery using load balancers, L7 Load balancing, persistence methods, health monitoring

ADC – Introduction

Why ADC is needed and a brief introduction, How ADC is different from a legacy load balancer? Overview of broadened ADC use cases

Load Balancer

Load balancing refers to efficiently distributing incoming network traffic across a group of backend servers, also known as a *server farm* or *server pool*. Modern high-traffic websites must serve hundreds of thousands, if not millions, of concurrent requests from users or clients and return the correct text, images, video, or application data, all in a fast and reliable manner.

A load balancer sits between the client and the server farm accepting incoming network and application traffic and distributing the traffic across multiple backend servers using various methods. This is shown in Figure 1 below. By balancing application requests across multiple servers, a load balancer reduces individual server load and prevents any one application server from becoming a single point of failure, thus improving overall application availability and responsiveness. If a single server goes down, the load balancer redirects traffic to the remaining online servers.

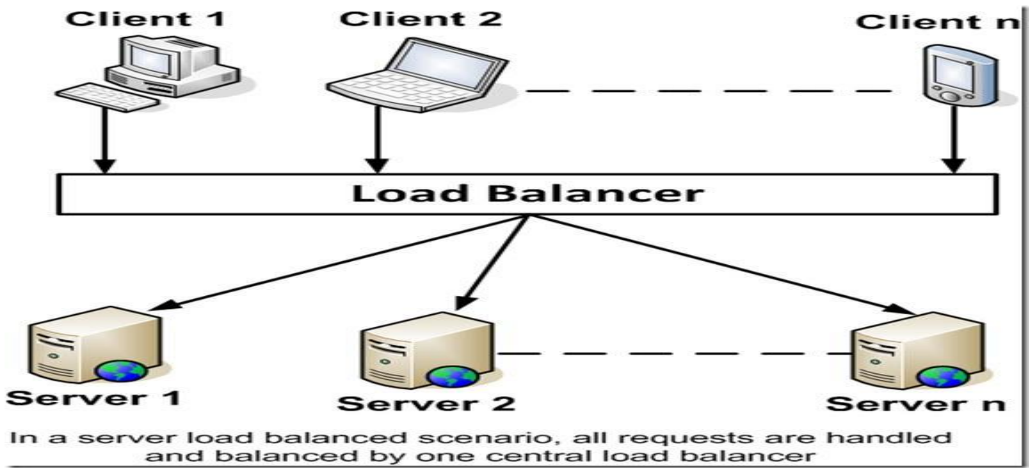

Client 1
Client 2
Client n
Load Balancer
Server 1
Server 2
Server n
In a server load balanced scenario, all requests are handled and balanced by one central load balancer

Figure 1: Placement of Load balancer

Load balancing is the most straightforward method of scaling out an application server infrastructure. As application demand increases, new servers can be easily added to the resource pool, and the load balancer will immediately begin sending traffic to the new server. Core load balancing capabilities include:

- Layer 4 (L4) load balancing - The ability to direct traffic based on data from network and transport layer protocols, such as IP address and TCP port
- Layer 7 (L7) load balancing and content switching – The ability to make routing decisions based on application layer data and attributes, such as HTTP header, uniform resource identifier, SSL session ID and HTML form data
- Global server load balancing (GSLB) - Extends the core L4 and L7 capabilities so that they are applicable across geographically distributed server farms

Load Balancing Algorithms and methods

Different load balancing algorithms provide different benefits; the choice of load balancing method depends on your needs. Requests are received by both L4 and L7 type of load balancers and they are distributed to a particular server based on a configured algorithm. Some industry standard algorithms are:

- The Least Connection Method

  The default method, when a virtual server is configured to use the least connection, it selects the service with the fewest active connections.
- The Round Robin Method

  This method continuously rotates a list of services that are attached to it. When the server receives a request, it assigns the connection to the first service in the list, and then moves that service to the bottom of the list.
- Weighted Round Robin -- as Round Robin, but some servers get a larger share of the overall traffic.
- Least connections, weighted least connections. The load balancer monitors the number of openconnections for each server and sends to the least busy server.
- The Least Response Time Method

  This method selects the service with the fewest active connections and the lowest average response time.
- The Least Bandwidth Method

  This method selects the service that is currently serving the least amount of traffic, measured in megabits per second (Mbps).
- The Least Packets Method

  This method selects the service that has received the fewest packets over a specified period.
- The Custom Load Method

  When using this method, the load balancing appliance chooses a service that is not handling any active transactions. If all the services in the load balancing setup are handling active transactions, the appliance selects the service with the smallest load.

Internet based services - Background

In the Beginning, There Was DNS

Before there were any commercially available, purpose-built load balancing devices, there were many attempts to utilize existing technology to achieve the goals of scalability and high availability

(HA). The most prevalent, and still used, technology was DNS round-robin. Domain name system (DNS) is the service that translates human-readable names (www.example.com) into machine recognized IP addresses. DNS also provided a way in which each request for name resolution could be answered with multiple IP addresses in different order, mostly in round robin. This is depicted by Figure 2 below.

From a scalability standpoint, this solution worked remarkable well; probably the reason why derivatives of this method are still in use today particularly regarding global load balancing or the distribution of load to different service points around the world. As the service needed to grow, all the business owner needed to do was add a new server, include its IP address in the DNS records, and get increased capacity.

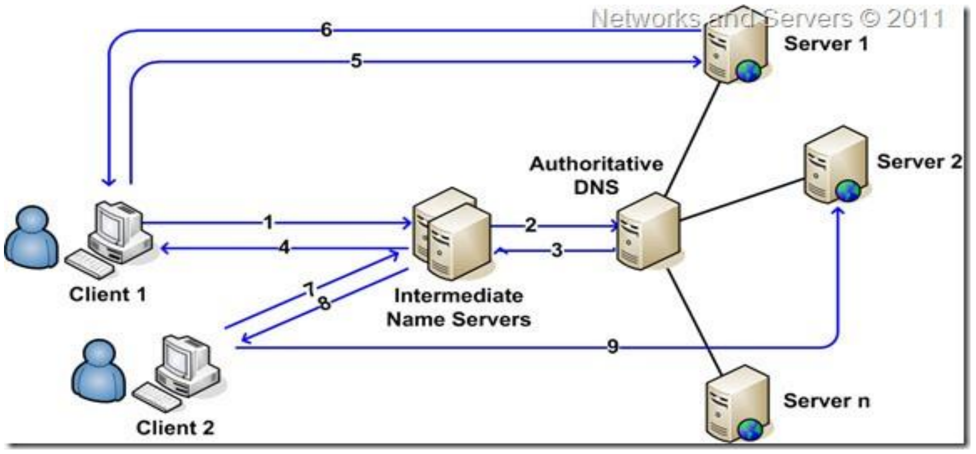


Figure 2: Round-robin DNS

In its simplest implementation, Round-robin DNS works by responding to DNS requests not only with a single potential IP address, but with one out of a list of potential IP addresses corresponding to several servers that host identical services. The order in which IP addresses from the list are returned is the basis for the term round robin. With each DNS response, the IP address sequence in the list is permuted. Usually, basic IP clients attempt connections with the first address returned from a DNS query, so that on different connection attempts, clients would receive service from different providers, thus distributing the overall load among servers.

Network-Based Load balancing Hardware

The second iteration of purpose-built load balancing came about as network-based appliances. These are the true founding fathers of today's Application Delivery Controllers. Because these boxes were application-neutral and resided outside of the application servers themselves, they could achieve their load balancing using much more straight-forward network techniques. In essence, these devices would present a virtual server address to the outside world and when users attempted to connect, it would forward the connection on the most appropriate real server doing bi-directional network address translation (NAT). This is depicted in Figure 3.

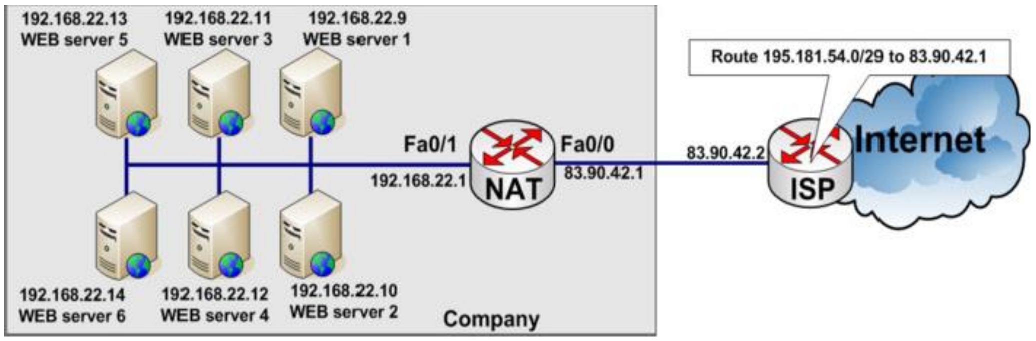


Figure 3: LB performing NAT

The load balancer could control exactly which server received which connection and employed "health monitors" of increasing complexity to ensure that the application server (a real, physical server) was responding as needed; if not, it would automatically stop sending traffic to that server until it produced the desired response (indicating that the server was functioning properly). Although the health monitors were rarely as comprehensive as the ones built by the application developers themselves, the network-based hardware approach could provide at least basic load balancing services to nearly every application in a uniform, consistent manner—finally creating a truly virtualized service entry point unique to the application servers serving it.

Scalability with this solution was only limited by the throughput of the load balancing equipment and the networks attached to it. It was not uncommon for organization replacing software-based load balancing with a hardware-based solution to see a dramatic drop in the utilization of their servers. HA was also dramatically reinforced with a hardware-based solution. Predictability was a core component added by the network-based load balancing hardware since it was much easier to predict where a new connection would be directed and much easier to manipulate. This is the basis from which Application Delivery Controllers (ADCs) originated.

Widely used ways of load balancing

Client-side random load balancing

Another approach to load balancing is to deliver a list of server IPs to the client, and then to have client randomly select the IP from the list on each connection. This essentially relies on all clients generating similar loads, and the Law of Large Numbers to achieve a reasonably flat load distribution across servers. It has been claimed that client-side random load balancing tends to provide better load distribution than round-robin DNS. This has been attributed to caching issues with round-robin DNS, that in case of large DNS caching servers, tend to skew the distribution for round-robin DNS, while client-side random selection remains unaffected regardless of DNS caching. With this approach, the method of delivery of list of IPs to the client can vary and may be implemented as a DNS list (delivered to all the clients without any round-robin) or via hardcoding it to the list. If a "smart client" is used, detecting that randomly selected server is down and connecting randomly again, it also provides fault tolerance.

Server-side load balancers

For Internet services, server-side load balancer is usually a software program that is listening on the port where external clients connect to access services. The load balancer forwards requests to one of the "backend" servers, which usually replies to the load balancer. This allows the load balancer to reply to the client without the client ever knowing about the internal separation of functions. It also prevents clients from contacting back-end servers directly, which may have security benefits by hiding the structure of the internal network and preventing attacks on the kernel's network stack or unrelated services running on other ports.

Some load balancers provide a mechanism for doing something special in the event that all backend servers are unavailable. This might include forwarding to a backup load balancer or displaying a message regarding the outage. It is also important that the load balancer itself does not become a single point of failure. Usually load balancers are implemented in high-availability pairs which may also replicate session persistence data if required by the specific application.

Hashing

Load balancing methods based on hashes of certain connection information or header information helps in persistence. Hashes are shorter and easier to use than the information that they are based on. Still hash retains enough information to ensure that no two different pieces of information generate the same hash and hence collision resistant.

One can use the hashing load balancing methods in an environment where a cache serves a wide range of content from the Internet or specified origin servers. Caching the requests reduces latency due to request-response and ensures better resource (CPU) utilization. Hence caching is popular on heavily used Web sites and application servers. Since these sites also benefit from load balancing, hashing load balancing methods are widely useful.

These hashing algorithms ensure minimal disruption when services are added to or deleted from any load balancing setup. Most of them calculate two hash values:

- A hash of the service’s IP address and port.
- A hash of the incoming URL, the domain name, the source IP address, the destination IP address, or the source and destination IP addresses, depending on the configured hash method.

URL Hashing

URL Hashing is typically used in situations where load-balanced servers serve content that is mostly (but not necessarily) unique per server. It is used, for example, in a deployment where a pool of cache servers responds to requests for content. When the NetScaler is configured to use the URL hash method, it selects a service based on the hashed value of an incoming HTTP URL. The NetScaler caches the hashed value of the URL, and subsequent requests that use the same URL make a cache hit and are forwarded to the same service. By default, the NetScaler calculates the hash value based on the first 80 bytes of the URL. You must specify the Hash Length parameter to calculate a different URL value. If the NetScaler cannot accurately parse the incoming request, it uses the round robin method for load balancing.

The Destination IP Hash Method

A load balancing virtual server configured to use the destination IP hash method uses the hashed value of the destination IP address to select a server. You can mask the destination IP address to specify which part of it to use in the hash value calculation, so that requests that are from different networks but destined for the same subnet are all directed to the same server. This method supports IPv4 and IPv6-based destination servers. This load balancing method is appropriate for usewith the cache redirection feature.

The Source IP Hash Method

A load balancing virtual server configured to use the source IP hash method uses the hashed value of the client IPv4 or IPv6 address to select a service. To direct all requests from source IP addresses that belong to a particular network to a specific destination server, you must mask the source IP address.

The Source IP Destination IP Hash Method

A load balancing virtual server configured to use the source IP destination IP hash method uses the hashed value of the source and destination IP addresses (either IPv4 or IPv6) to select a service. Hashing is symmetric; the hash-value is the same regardless of the order of the source and destination IPs. This ensures that all packets flowing from a particular client to the same destination are directed to the same server.

To direct all requests that belong to a particular network to a specific destination server, you must mask the source IP address. For IPv4 addresses, use the netMask parameter. For IPv6 addresses, use the v6NetMaskLength parameter.

Persistence

An important issue when operating a load-balanced service is how to handle information that must be kept across the multiple requests in a user's session. If this information is stored locally on one backend server, then subsequent requests going to different backend servers would not be able to find it. This might be cached information that can be recomputed, in which case load-balancing a request to a different backend server just introduces a performance issue.

Ideally the cluster of servers behind the load balancer should be session-aware, so that if a client connects to any backend server at any time the user experience is unaffected. This is usually achieved with a shared database or an in-memory session database, for example **Memcached**.

One basic solution to the session data issue is to send all requests in a user session consistently to the same backend server. This is known as persistence or stickiness. A significant downside to this technique is its lack of automatic failover: if a backend server goes down, its per-session information becomes inaccessible, and any sessions depending on it are lost. The same problem is usually relevant to central database servers; even if web servers are "stateless" and not "sticky", the central database is.

Assignment to a particular server might be based on a username, client IP address, or be random. Because of changes of the client's perceived address resulting from DHCP, network address translation, and web proxies this method may be unreliable. Random assignments must be remembered by the load balancer, which creates a burden on storage. If the load balancer is replaced or fails, this information may be lost, and assignments may need to be deleted after a timeout period or during periods of high load to avoid exceeding the space available for the assignment table. The random assignment method also requires that clients maintain some state, which can be a problem, for example when a web browser has disabled storage of cookies. Sophisticated load balancers use multiple persistence techniques to avoid some of the shortcomings of any one method.

Health monitoring

Classic Load Balancer periodically sends requests to its registered instances to test their status. These tests are called *health checks*. The status of the instances that are healthy at the time of the health check is `InService`. The status of any instances that are unhealthy at the time of the health check is `OutOfService`. The load balancer performs health checks on all registered instances, whether the instance is in a healthy state or an unhealthy state.

The load balancer routes requests only to the healthy instances. When the load balancer determines that an instance is unhealthy, it stops routing requests to that instance. The load balancer resumes routing requests to the instance when it has been restored to a healthy state. The load balancer checks the health of the registered instances using either the default health check configuration provided by Elastic Load Balancing or a health check configuration that you configure.

The load balancer sends a health check request to each registered instance every `Interval` seconds, using the specified port, protocol, and path. Each health check request is independent and lasts the entire interval. The time it takes for the instance to respond does not affect the interval for the next health check. If the health checks exceed `UnhealthyThresholdCount` consecutive failures, the load balancer takes the instance out of service. When the health checks exceed `HealthyThresholdCount` consecutive successes, the load balancer puts the instance back in service. Table 1 gives more details about fields used in typical health checking commands. An HTTP/HTTPS health check succeeds if the instance returns a 200-response code within the health check interval. A TCP health check succeeds if the TCP connection succeeds. An SSL health check succeeds if the SSL handshake succeeds.

Table 1: Fields in health checking commands.

Field	Description
Protocol	The protocol to use to connect with the instance. Valid values: `TCP`, `HTTP`, `HTTPS`, and `SSL` Console default: `HTTP` CLI/API default: `TCP`
Port	The port to use to connect with the instance, as a `protocol: port` pair. If the load balancer fails to connect with the instance at the specified port within the configured response timeout period, the instance is considered unhealthy. Protocols: `TCP`, `HTTP`, `HTTPS`, and `SSL` Port range: 1 to 65535 Console default: `HTTP:80` CLI/API default: `TCP:80`

Field	Description
Path	The destination for the HTTP or HTTPS request. An HTTP or HTTPS GET request is issued to the instance on the port and the path. If the load balancer receives any response other than "200 OK" within the response timeout period, the instance is considered unhealthy. If the response includes a body, your application must either set the Content-Length header to a value greater than or equal to zero or specify Transfer-Encoding with a value set to 'chunked'. Default: `/index.html`
Response Timeout	The amount of time to wait when receiving a response from the health check, in seconds. Valid values: 2 to 60 Default: 5
Health Check Interval	The amount of time between health checks of an individual instance, in seconds. Valid values: 5 to 300 Default: 30
Unhealthy Threshold	The number of consecutive failed health checks that must occur before declaring an EC2 instance unhealthy. Valid values: 2 to 10 Default: 2
Healthy Threshold	The number of consecutive successful health checks that must occur before declaring an EC2 instance healthy. Valid values: 2 to 10 Default: 10

Example command:

```
aws elb configure-health-check --load-balancer-name my-load-balancer --health-check Target=HTTP:80/path, Interval=30, UnhealthyThreshold=2, HealthyThreshold=2, Timeout=3
```

Load balancer features

Hardware and software load balancers may have a variety of special features. The fundamental feature of a load balancer is to be able to distribute incoming requests over several backend servers in the cluster according to a scheduling algorithm. Most of the following features are vendor specific:

1. **Asymmetric load**: A ratio can be manually assigned to cause some backend servers to get a greater share of the workload than others. This is sometimes used as a crude way to account for some servers having more capacity than others and may not always work as desired.
2. **Priority activation**: When the number of available servers drops below a certain number, orload gets too high, standby servers can be brought online.

3. **SSL Offload and Acceleration**: Depending on the workload, processing the encryption and authentication requirements of an SSL request can become a major part of the demand on the Web Server's CPU; as the demand increases, users will see slower response times, as the SSL overhead is distributed among Web servers. To remove this demand on Web servers, a balancer can terminate SSL connections, passing HTTPS requests as HTTP requests to the Web servers. If the balancer itself is not overloaded, this does not noticeably degrade the performance perceived by end users. The downside of this approach is that all the SSL processing is concentrated on a single device (the balancer) which can become a new bottleneck. Some load balancer appliances include specialized hardware to process SSL. Instead of upgrading the load balancer, which is quite expensive dedicated hardware, it may be cheaper to forgo SSL offload and add a few Web servers. Also, some server vendors such as Oracle/Sun now incorporate cryptographic acceleration hardware into their CPUs such as the T2000. F5 Networks incorporates a dedicated SSL acceleration hardware card in their local traffic manager (LTM) which is used for encrypting and decrypting SSL traffic. One clear benefit to SSL offloading in the balancer is that it enables it to do balancing or content switching based on data in the HTTPS request.
4. **Distributed Denial of Service (DDoS) attack protection**: Load balancers can provide features such as SYN cookies and delayed-binding (the back-end servers don't see the client until it finishes its TCP handshake) to mitigate SYN flood attacks and generally offload work from the servers to a more efficient platform.
5. **HTTP compression**: Reduces amount of data to be transferred for HTTP objects by utilizing gzip compression available in all modern web browsers. The larger the response and the further away the client is, the more this feature can improve response times. The trade-off is that this feature puts additional CPU demand on the load balancer and could be done by web servers instead.
6. **TCP offload**: Different vendors use different terms for this, but the idea is that normally each HTTP request from each client is a different TCP connection. This feature utilizes HTTP/1.1 to consolidate multiple HTTP requests from multiple clients into a single TCP socket to the back-end servers.
7. **TCP buffering**: The load balancer can buffer responses from the server and spoon-feed the data out to slow clients, allowing the web server to free a thread for other tasks faster than it would if it had to send the entire request to the client directly.
8. **Health checking**: The balancer polls servers for application layer health and removes failed servers from the pool.
9. **HTTP caching**: The balancer stores static content so that some requests can be handled without contacting the servers.
10. **Content filtering**: Some balancers can arbitrarily modify traffic on the way through.
11. **HTTP security**: Some balancers can hide HTTP error pages, remove server identification headers from HTTP responses, and encrypt cookies so that end users cannot manipulate them.
12. **Priority queuing**: Also known as rate shaping, the ability to give different priority to different traffic.
13. **Content-aware switching**: Most load balancers can send requests to different servers based on the URL being requested, assuming the request is not encrypted (HTTP) or if it is encrypted (via HTTPS) that the HTTPS request is terminated (decrypted) at the load balancer.

14. **Client authentication**: authenticate users against a variety of authentication sources beforeallowing them access to a website.
15. **Programmatic traffic manipulation**: at least one balancer allows the use of a scripting language to allow custom balancing methods, arbitrary traffic manipulations, and more.

Hardware versus Software Load Balancing

Load balancers typically come in two flavours: hardware-based and software-based. Vendors of hardware-based solutions load proprietary software onto the machine they provide, which often uses specialized processors. To cope with increasing traffic at your website, you must buy more or bigger machines from the vendor. Software solutions generally run-on commodity hardware, making them less expensive and more flexible. You can install the software on the hardware of your choice or in cloud environments like AWS EC2.

Virtual Server and NAT

A virtual server is a proxy of the actual server (physical, virtual, or container). Combined with a virtual IP address, this is the application endpoint that is presented to the outside world as shown in Figure 4. With an understanding of these terms, we've got the basics of load balancing. The load balancing ADC presents virtual servers to the outside world. Each virtual server points to a cluster of services thatreside on one or more physical hosts.

Linux Virtual Server (LVS) via NAT is does not require modification on the real servers. The real servers can be any operating system that supports TCP IP stack. LVS via NAT is implemented using the IP masquerading technique in Linux. The request from the user first arrives at the virtual IP assigned to the front-end Load Balancer. The Load balancer then does an investigation on the packet and modifies it with a destination address of the real server from the pool and forwards it to the real server. Another important thing to note here is that the virtual IP (The IP accessed by the client, which is generally added as an A record for the domain name providing the service) is only assigned to the Load Balancer. The Real servers can have any Private addressing scheme.

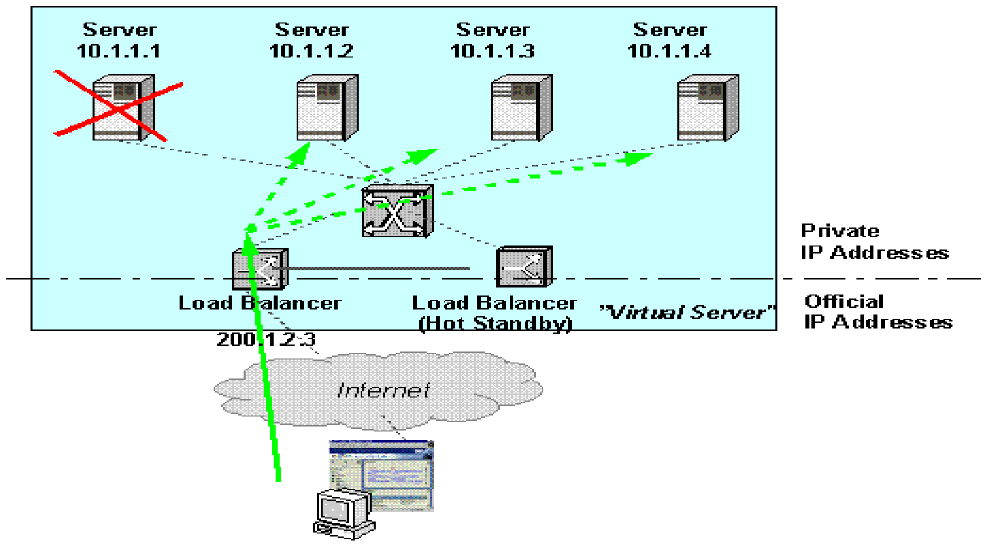


Figure 4: LVS via NAT

L7 load balancing

Layer 7 load balancing operates at the higher-level namely *application* layer, which deals with the actual content of each message. HTTP is the predominant Layer 7 protocol for website traffic on the Internet. Layer 7 load balancers route network traffic in a much more sophisticated way than Layer 4 load balancers, particularly applicable to TCP-based traffic such as HTTP. A Layer 7 load balancer terminates the network traffic and reads the message within. It can make a load-balancing decision based on the content of the message (the URL or cookie, for example). It then makes a new TCP connection to the selected upstream server (or reuses an existing one, by means of HTTP keepalives) and writes the request to the server.

Layer 7 load balancing is also known as "request switching," "application load balancing," "content-based routing," "content-based switching," and "content-based balancing."

Benefits of Layer 7 Load Balancing

Layer 7 load balancing is more CPU-intensive than packet-based Layer 4 load balancing, but rarely causes degraded performance on a modern server. Layer 7 load balancing enables the load balancer to make smarter load-balancing decisions, and to apply optimizations and changes to the content (such as compression and encryption). 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.

A layer 7 load balancer consists of a listener that accepts requests on behalf of several back-end pools and distributes those requests based on policies that use application data to determine which pools should service any given request. This allows for the application infrastructure to be specifically tuned/optimized to serve specific types of content. For example, one group of back-end servers (pool) can be tuned to serve only images, another for execution of server-side scripting languages like PHP and ASP, and another for static content such as HTML, CSS, and JavaScript.

****

Application Delivery Controller (ADC)

Application delivery controllers are purpose-built networking appliances whose function is to improve the performance, security and resiliency of applications delivered over the web. ADC is part of Application delivery networks (ADN). Application Delivery Network can be broken down into two components: Application Delivery Controller and WAN Optimization Controller.
Application delivery networks comprise a set of technologies that aim to maximize application availability, acceleration, security, and visibility. They are a holistic and broad-based approach to content delivery, and performance optimization for remote applications. ADCs are often placed in the DMZ, between the outer firewall or router and a web farm.

WAN Optimization Controllers (WOCs) are typically located at both the data center facilities and the client device.

Hardware-based load balancers with network level traffic management were the forerunners of modern application delivery controllers. A common misconception is that an ADC is an advanced load-balancer. This is not an adequate description. In fact, an ADC includes many OSI layer 3-7 services including load-balancing. Other features commonly found in most ADCs include SSL offload, Web Application Firewall, NAT64, DNS64 and proxy/reverse proxy, ….

An advanced ADC will route users to destination servers based on a variety of criteria that the data center manager implements using policies and advanced application-layer knowledge to support business requirements. And much like our example traffic officer, an advanced ADC will ensure that the users get to the applications based on their specific needs while protecting the network and applications from security threats. Among the advanced acceleration functions present in modern ADCs are SSL offloading technology, data compression, TCP and HTTP protocol optimization and virtualization awareness. Typical scenario is shown in Figure 5.

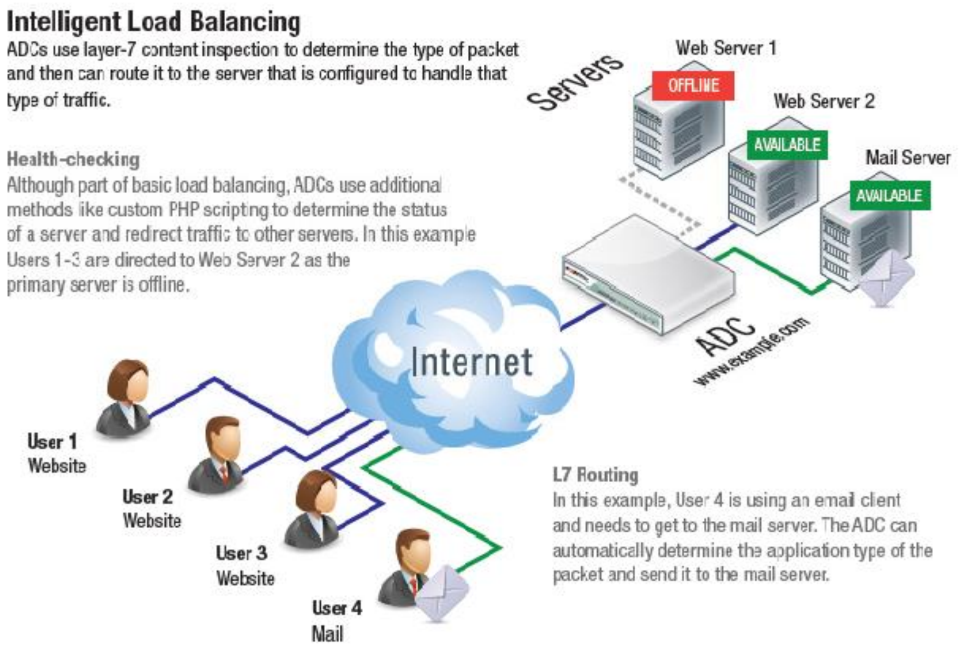

Intelligent Load Balancing
ADCs use layer-7 content inspection to determine the type of packet and then can route it to the server that is configured to handle that type of traffic.
Health-checking
Although part of basic load balancing, ADCs use additional methods like custom PHP scripting to determine the status of a server and redirect traffic to other servers. In this example Users 1-3 are directed to Web Server 2 as the primary server is offline.
Servers
Web Server 1
OFFLINE
Web Server 2
AVAILABLE
Mail Server
AVAILABLE
Internet
ADC
www.example.com
User 1
Website
User 2
Website
User 3
Website
User 4
Mail
L7 Routing
In this example, User 4 is using an email client and needs to get to the mail server. The ADC can automatically determine the application type of the packet and send it to the mail server.

Figure 5: ADC facing server farm

ADCs offload servers by reducing the bandwidth utilization required to deliver application data from the data center to the desktop. ADCs offer compression to remove non-essential data from traversing network links. This helps to deliver maximum bandwidth utilization to support more traffic and avoids the need for network upgrades. By offloading and accelerating SSL encryption, decryption and certificate management from servers, ADCs enable web and application servers to use their CPU and memory resources exclusively to deliver application content and thus respond more quickly to user requests.

Web-based applications consist of a variety of different data objects which can be delivered by different types of servers. ADCs provide application-based routing using file types to direct users to the server (or group of servers) that is set up to handle their specific information requests, such as ASP or PHP applications. User requests can be routed to different servers by sending requests for static file types (jpg, html, etc.) to one server group, and sending user requests for dynamic data to other servers optimized for that purpose. Like the ultimate traffic cop, the ADC knows the optimal path for each destination.

Transaction-based applications require connections to the same server to operate correctly. The best-known example of this is the "shopping cart" problem when you establish a session with one server to add an item to your cart and then are load balanced to a different server to checkout. If you don't have a persistent connection to the original server, you'll find your cart is empty. ADCs use session state with HTTP headers and cookies to ensure that users and servers remain "persistent". The ADC uses the cookie within the HTTP header to ensure that users continue to be directed to the specific server where the session state information resides. Without this capability, if the user went to a different server, the previous transaction history would be lost, and the user would need to start the transaction over. Once again, the ultimate traffic cop saves the day by understanding the application, network conditions and your priorities.

Finally, today's ADCs need to operate in and manage virtual environments. Advanced ADCs offer deep resource management of virtual environments and not just basic health-checking for server availability. With this tight virtual integration, the ADC can make load balancing decisions based on the status of the virtual machines and the servers they run on.

Application and user security

Delivery over the web has introduced new threats and vulnerabilities that traditional LAN- bound applications never had to contend with. As workers become more mobile and require remote access to applications and data, IT must devise more-stringent safeguards against external attacks and data leakage.

Application delivery controllers serve as the natural entry point or gateway to the network. They authenticate each user attempting to access an application. If the application is SaaS based, the ADC can validate a user's identity using an on-premises Active Directory data store that eliminates the need to store credentials in the cloud. Not only is this process more secure, but it also enhances the user experience by providing single sign-on capabilities across multiple applications.

SAML, the XML-based protocol, is now widely used to simplify the application login process. The ADC can act as a SAML agent, authorizing users via any data stores where their identity can be confirmed. Some applications allow the use of credentials from sites such as Facebook or Google+ to validate identity before granting access. ADCs can act as a SAML identity or service provider in this respect.

Distributed Denial-of-Service (DDoS) attacks have become rampant. Enterprise web properties, specifically, are being targeted with the intent of overwhelming their servers and disrupting their ability to conduct business. The ADC can implement rate-limiting measures to protect internal server resources from being targeted by these specially designed attacks. When an unusually massive surge of inbound requests occurs, the ADC can throttle these requests and minimize the amount of available bandwidth they consume or reject the request entirely.

ADC's have converged load balancing and advanced Layer 7 protection, which traditionally were only available as standalone solutions. Application firewalls can inspect data packet headers for suspicious content or malicious scripts that may not be detected by network firewall (See Figure 6).

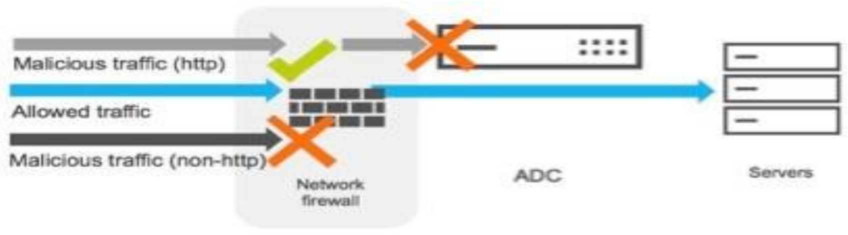


Figure 6: Firewall bypassed packets blocked by ADC

An application delivery controller can support both positive and negative security models. When an ADC is placed in "learning" mode, it can analyze traffic to determine usage patterns that signify normal behavior. If a malicious inbound request is sent, for example, using SQL injection or cross-site scripting, it will automatically flag that request and block it. It can also employ signature-based protection via integration with third- party security providers such as Qualys. Combining these protection methods allows the ADC to use a comprehensive hybrid security model for applications and users.

Basic Load-Balancer and ADC Placement Scenarios

Once you know the basic principles, every load-balancer or ADC from any vendor is easy to configure by knowing the three basic scenarios that are mostly deployed in real life.

1. Two-Arm (sometimes called In-Line)
2. One-Arm (dual NAT mode)
3. Direct Server Response

Two-Arm Load-Balancer (in-line mode)

Two-Arm is basic scenario where you have a server farm in one side of the network (Back End) and the ADC is essentially the default gateway router for the physical servers in the Back End network. In general, Two-Arm can also be used in something called "bridge mode" or "transparent mode".

The scenario diagram would be the same with the exception that the Load-Balancer would be essentially a switch load-balancing by modifying L2 frames. But the focus here is on the routed mode as it is the most common and easier to understand. For the "transparent mode", the same traffic paths would apply, but the load-balancer must intercept traffic at L2 to load-balance by changing L2 frames to get them to the physical routers in server farm.

But let's get back to the routed mode scenario as it is much simpler, cleaner, and easier to troubleshoot and therefore deployed much more often. Basic scenario of routed two-arm solution can be seen on next figure. Figure 7 is generalized for overview purposes, and you can note the Load-Balancer separating users and server physically, therefore all the traffic must go via the Load-Balancer.

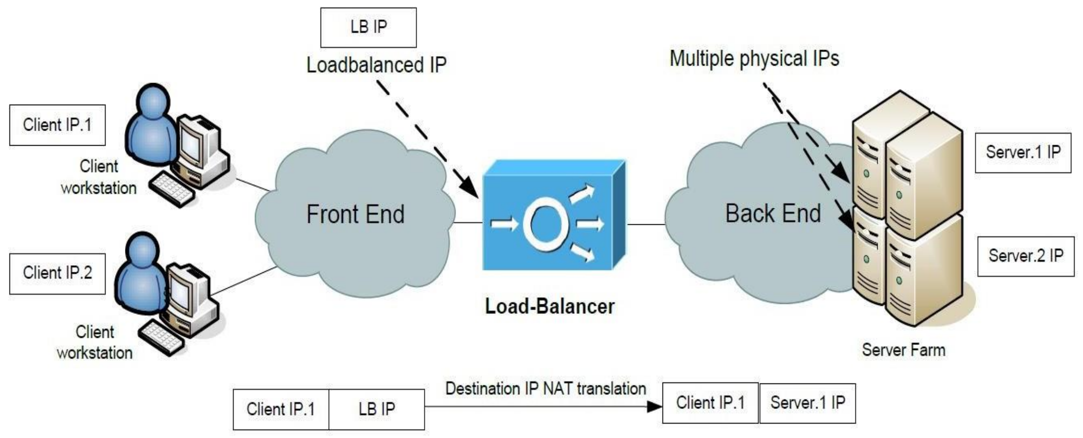


Figure 7: Two arm (In-line) mode

Here, the Load-Balancer is also a router between the "Front End" and "Back End" networks. As such, it can simply do destination IP NAT in client request coming to the load-balanced virtual IP and forward the packet to one of the servers in server farm. During this process, the destination physical server is chosen by the load-balancing algorithm. Return traffic is going back via the Load-Balancer and the source IP is again changed to the virtual load-balanced IP in the response to the Client.

One-Arm Load-Balancer

One-Arm means that the Load-Balancer is not physically "in-line" of the traffic. But it must get into the way of traffic somehow, to have control over the entire Client to Server connections going in both ways. So, let's have a look at the topology with One- Arm solution shown in Figure 8 to understand what the Load-Balancer must do.

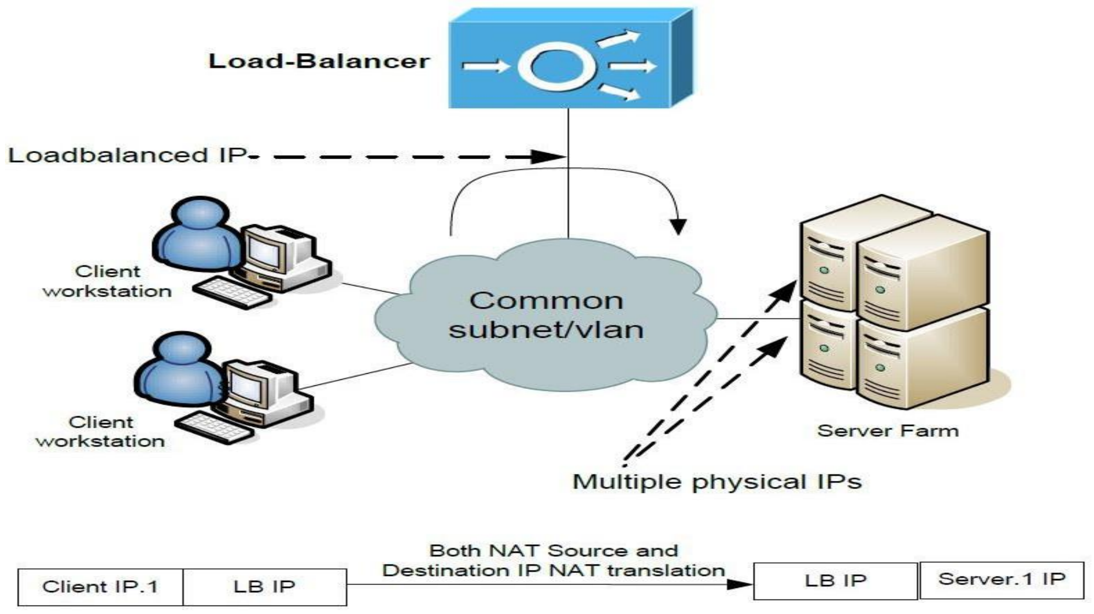


Figure 8: One-Arm (dual NAT mode)

It is not important how far away the Client workstations are, they can be behind internet or in the same LAN and the load-balancing would be the same. However, the Load-Balancer is using only one interface and this interface is on the same L2 network with all the servers.
The traffic that the client initializes will get to the Load-Balancer that has the virtual load-balanced IP. The load-sharing algorithm will pick a physical server to which the Load-Balancer will forward the traffic with destination IP NAT'ed to the physical IP of the server and forward it out the same interface towards the physical server.

But the Load-balancer also needs to do source IP NAT so that the server reply will go back from the server to the Load-Balancer and not directly back to the Client. Clients are not expecting a reply directly from physical server IP. From the physical server's perspective, all the traffic iscoming from Load-Balancer.

The theoretical limit of NAT (or PAT) hiding entire networks behind a single IP is 65355 sessions, but in real life the pragmatically limit is somewhere around 3000 connections (this value is recommended by Cisco)

Direct Server Response (sometimes called Direct Server Return)

To understand this Load-Balancer scenario in Direct Server Response, we need to bring a switch into the topology. As we know, switches learn about MAC addresses as they see frames coming on ports with source MACs. Also imagine that we have a router that must know the MAC address of the Load-Balanced IP on the last L3 hop. With the Figure 9 below, you can already spot the "trick" this scenario tries to present here once you notice the disabled ARP on physical servers.

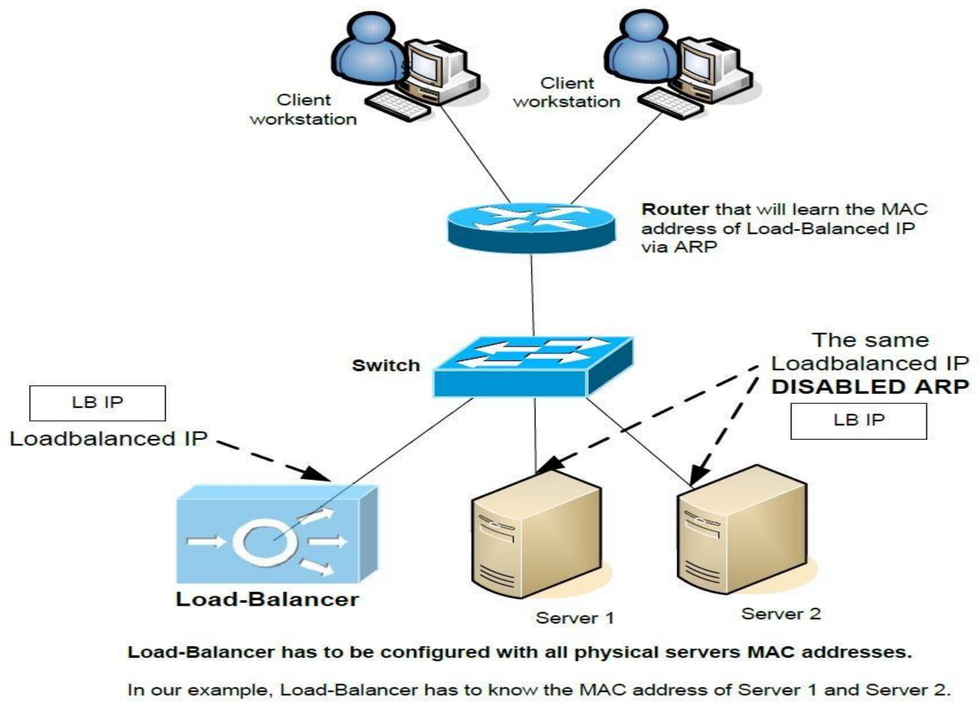

Client workstation
Client workstation
Router that will learn the MAC address of Load-Balanced IP via ARP
Switch
The same Loadbalanced IP DISABLED ARP
LB IP
LB IP
Loadbalanced IP
Load-Balancer
Server 1
Server 2
Load-Balancer has to be configured with all physical servers MAC addresses.
In our example, Load-Balancer has to know the MAC address of Server 1 and Server 2.

Figure 9: Direct Server Response mode

In this scenario, Load-balancer only sees the incoming part of client-server traffic and all the returning traffic from physical servers is coming directly back to the client IP. The biggest advantage of this solution is that there is no NAT, and the Load-Balancer throughput is only used in one way, so less performance impact for the Load-Balancer system. Disabling ARP on a physical server is not a difficult task.
Disadvantages however are that you must manually configure the Load-Balancer with all the server MAC addresses and might be more difficult to troubleshoot with only one way traffic seen by the Load-Balancer on the whole L2 segment.

Summary of Load Balancing Modes

1. One-Arm mode (Dual NAT)
2. Two-arm mode (Server NAT)
3. Direct Routed Mode- (Transparent Mode)

Table 2 mentions the changes made to client requests so that the request is routed to the correct server.

Table 2

Type	Change source IP address?	Change source TCP port?
Dual NAT	Yes	Yes
Server NAT	No	Yes
Transparent	No	No

Performance optimization on mobile networks

ADCs can also provide performance benefits across mobile networks. Web pages designed for high-speed Internet links often fail to deliver the same user experience on a mobile device connecting over a bandwidth-constrained network.

Several creative mechanisms enable an ADC to optimize web content delivery over mobile networks. Domain sharding is one example. Connection-layer optimization is applied to a single domain. Content on each page is broken down into a sequence of subdomains that allow a larger number of channels to be opened simultaneously, which decreases page load time and improves performance.

ADCs have visibility into the content that is being delivered and can further optimize delivery of web pages containing large images by converting GIF files into more-efficient PNG formats. The other large components of a web page include extensive scripts and cascading style sheet (CSS) files, which ADCs can compress by removing unnecessary characters and white space. When compressed, files traverse the network at a much faster rate, so download times are significantly reduced.

NetScaler (Citrix ADC)

Citrix NetScaler is an ADC system from Citrix that provides Level 4 load balancing to deliver better performance for apps and services. It optimizes, secures, and controls the delivery of applications, providing the required flexibility for businesses to improve performance and continuity.

Citrix NetScaler is available as a **virtual appliance** (VPX), **logical appliance** (SDX), or a **physical appliance** (MPX). Any appliance can be selected in Standard, Enterprise, or Platinum. The best option for each business depends on the specifics of an organization. However, for a company that delivers virtual applications to about 2,000 employees and is looking to improve the availability and performance of the application, Citrix NetScaler VPX Standard — 200 Mbps is very popular.

The **cost** of Citrix NetScaler VPX Standard — 200 Mbps is about $15000 (nearly Rs. 11,25,000).

NetScaler MPX

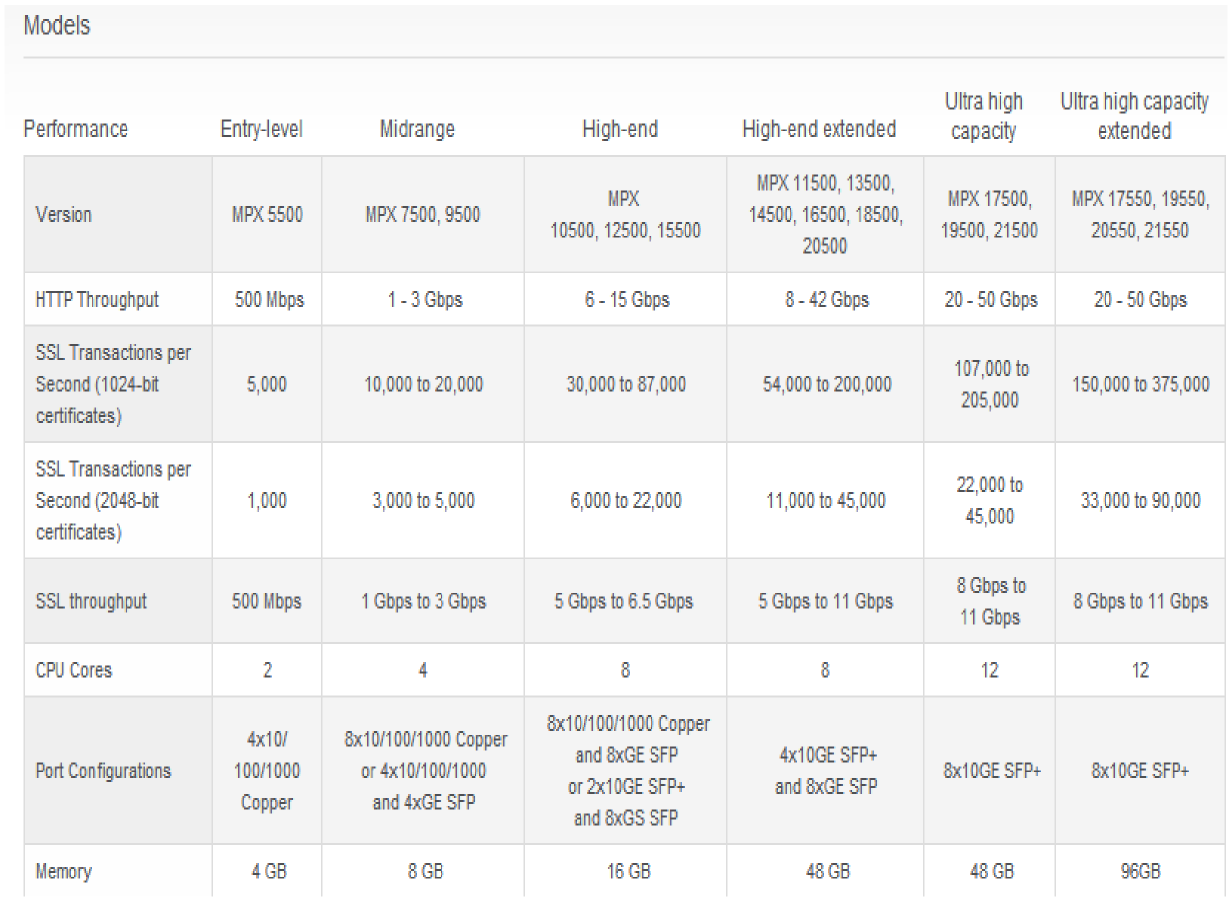
- MPX is NetScaler hardware platform
- Runs on Core software utilizing multiple cores
- Scales from 0.5 Gbps to 50Gbps of unmatched performance

NetScaler VPX

- VPX is software-based NetScaler
- VPX can run over XenServer, VMware and HyperV
- Current version of VPX supports up to 3Gbps throughput
- Every NS feature is supported over VPX

NetScaler MPX Models

Various configurations of MPX available in market is summarized in Table 3.



Models

Performance	Entry-level	Midrange	High-end	High-end extended	Ultra high capacity	Ultra high capacity extended
Version	MPX 5500	MPX 7500, 9500	MPX 10500, 12500, 15500	MPX 11500, 13500, 14500, 16500, 18500, 20500	MPX 17500, 19500, 21500	MPX 17550, 19550, 20550, 21550
HTTP Throughput	500 Mbps	1 - 3 Gbps	6 - 15 Gbps	8 - 42 Gbps	20 - 50 Gbps	20 - 50 Gbps
SSL Transactions per Second (1024-bit certificates)	5,000	10,000 to 20,000	30,000 to 87,000	54,000 to 200,000	107,000 to 205,000	150,000 to 375,000
SSL Transactions per Second (2048-bit certificates)	1,000	3,000 to 5,000	6,000 to 22,000	11,000 to 45,000	22,000 to 45,000	33,000 to 90,000
SSL throughput	500 Mbps	1 Gbps to 3 Gbps	5 Gbps to 6.5 Gbps	5 Gbps to 11 Gbps	8 Gbps to 11 Gbps	8 Gbps to 11 Gbps
CPU Cores	2	4	8	8	12	12
Port Configurations	4x10/ 100/1000 Copper	8x10/100/1000 Copper or 4x10/100/1000 and 4xGE SFP	8x10/100/1000 Copper and 8xGE SFP or 2x10GE SFP+ and 8xGS SFP	4x10GE SFP+ and 8xGE SFP	8x10GE SFP+	8x10GE SFP+
Memory	4 GB	8 GB	16 GB	48 GB	48 GB	96GB

How ADC is different from a legacy load balancer?

The following list of NetScaler features is extensive, and the capability is the basis of the continued success of the product, as recognized by Gartner for the last 10 years in the leading Magic Quadrant (ADCs).

The core features of NetScaler include the following.

- ICA Proxy (Secure remote access XenApp and XenDesktop)
- Full SSL VPN
- Smart Access and Smart Control
- Clientless Access
- Micro VPN (including XenMobile Connector)
- Cache Redirection
- Content Switching (Layer 7 load balancing)
- DataStream
- Domain Name System
- Firewall Load Balancing
- AppFirewall
- Global Server Load Balancing (GSLB)
- Link Load Balancing
- Layer 4-7 Load Balancing
- SSL Offload and Acceleration
- Single Sign-On
- Federated Authentication
- AAA (Authentication, Authorization and Audit)
- nFactor Authentication (SmartCard, RADIUS, SMS etc.)
- User Experience Monitoring (HDX Insights)
- TCP Optimization
- Front End Optimization
- Traffic Domains
- RDP Proxy
- IP Reputation

Few examples of load balancers:

Software based

- HAProxy – A TCP load balancer.
- NGINX – An http load balancer with SSL termination support. (Install Nginx on Linux)
- mod_athena – Apache based http load balancer.
- Varnish – A reverse proxy-based load balancer.
- Balance – Open-source TCP load balancer.
- LVS – Linux virtual server offering layer 4 load balancing

Hardware based

- F5 BIG-IP load balancer (Setup HTTPS load balance on F5)
- CISCO system catalyst
- Barracuda load balancer
- Coytepoint load balancer

References

https://docs.citrix.com/en-us/legacy-archive/netscaler.html

http://www.linuxvirtualserver.org/how.html

https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/elb-healthchecks.html

https://cloud.google.com/load-balancing/docs/health-check-concepts

https://f5.com/resources/white-papers/load-balancing-101-nuts-and-bolts