Application Delivery Controller and Virtualization (18CS7G2)
Instructor: Dr. Vishalakshi Prabhu H, Dept of CSE, RVCE, Bangalore-59
Author of notes: Dr. Vishalakshi Prabhu H, Date :**01-Nov-2021**
Contents of Unit - 3
Traffic Management: Core principles of traffic management, Multiprotocol Label Switching, DNS and global server load balancing, Content switching, AppQoE, TCP and SSL profiles, Introduction to Optimization and Security.

Core principles Network Traffic Management

Rapid growth and increasing requirements for service quality, reliability, and efficiency have made traffic engineering an essential consideration in the design and operation of large public Internet backbone networks. Internet traffic Engineering (TE) addresses the issue of performance optimization of operational networks. A paramount objective of Internet traffic engineering is to facilitate the transport of IP traffic through a given network in the most efficient, reliable, and expeditious manner possible.
Network Traffic is the density of data present in the network. Communication devices access resources and get requests to carry out some work. So, a lot of request, response, and control data increases Load on the network. Some devices may get delayed in their requirements.

Traffic Management is controlling network traffic requires limiting bandwidth to certain applications, guaranteeing minimum bandwidth to others, and marking traffic with high or low priorities. This exercise is called traffic management. Two major issues that have received attention in TE approaches are quality of service (QoS) and resilience.

NetScaler is in use in many large public organizations all over the world, doing traffic management for all their public web services. In large organizations, we might have over 100 different services masked behind one URL with different policies and features being used. This chapter tries to address some of the content related to Traffic Management as applied in WAN and from Citrix NetScaler perspective.

Organizational Requirements for managing Network Traffic?

1. To guarantee maximum bandwidth to mission critical applications.
2. To block music or video downloads.
3. To block music file sharing and avoid copyright infringement liability.

4. To delay investments in additional network capacity
5. Service monitoring - making sure things keep working.
6. Cost recovery - session times and traffic volumes can provide billing data.
7. Research - an improved understanding of what's happening should allow us to
8. Improve network performance.

Basic performance metrics:

Packet loss, Delay, Throughput, Error rate and Availability

Where should we manage traffic?

Usually deployed at the WAN edge of an enterprise site. The LAN-WAN juncture is also where both Internet and Intranet traffic enters and exits the enterprise.

Processes for Traffic Management

Once a problem of traffic management has been identified, the following tasks should be worked through in detail:

- □ Decide what is to be measured.
- □ Decide where to put traffic meters.
- □ Decide how data will be read from the meters and consolidated into a central database.
- □ Decide what reports will be generated, and how they will be published (on demand, via web page.).

Network Traffic Measurement Tools

Local Systems: NETSTAT, TCPDUMP, WIRESHARK

Remote End Systems: MIBS, IF-MIB, SNMP, RMON

Routers: NETFLOW (CISCO), LFAP (ENTERASYS)

Experiments have shown that these tools can currently handle up to several gigabits.

Traffic management approaches

Look from control theory point of view

- Open loop: - These act at source and destination to avoid problems from occurring
- Closed loop: - They act once the problem is on. Based on explicit & implicit feedback

Overview of congestion in network

Congestion: A state of a network resource in which the traffic incident on the resource exceeds its output capacity over an interval of time.

Congestion control: An approach to congestion management that attempts to remedy congestion problems that have already occurred.

Some of the techniques to control congestion are

- Traffic-aware routing
- Admission control
- Traffic throttling
- Load shedding

Quality of Service?

Quality of Service (QoS) is a set of technologies that work on a network to guarantee its ability to dependably run high-priority applications and traffic under limited network capacity.
Using QoS, businesses can prevent disruption in the form of IP packet loss, delays and jitter for VoIP (voice over IP), AoIP (audio over IP) and other real-time communications applications.
More often, QoS will be incorporated within the Service Level Agreement (SLA) given by their service provider. This guarantees a specific level of service. However, tools and techniques can be used independently to achieve QoS.

Approaches to Quality of Service

- Application requirements
- Traffic shaping
- Packet scheduling
- Admission control
- Integrated services
- Differentiated services

Some of the QoS tools in market are

- SolarWinds NetFlow Traffic Analyzer
- ManageEngine NetFlow Analyzer
- Paessler PRTG Network Monitor
- Ntopng

Traffic monitoring: Link–level versus Flow–level

Link–level traffic monitoring: Relies on SNMP statistics maintained by each router for each link. Management station polls each router frequently.

Advantages

- Simple to use and to deploy
- Tools can automate data collection/presentation
- Rough information about network load

Drawbacks

- No addressing information
- Not always easy to know the cause of congestion

Flow−level traffic monitoring: Routers identify flow boundaries. It does not cause problems on cache−based routers but more difficult on hardware−based routers. Higher−end routers rely on sampling. Routers forward the following information inside special packets to monitoring workstation.

Layer−3 flows

- IP packets with same source and destination prefix
- IP packets with same source and destination Autonomous System (AS)
- IP packets with same Border Gateway Protocol (BGP) next hop

Layer−4 flows

- one TCP connection corresponds to one flow
- UDP flows

Advantages are that it provides detailed information on the traffic carried out on some links.

Drawbacks

- Flow information exported to monitoring station
- CPU load is high on routers
- Disk and processing requirements on workstation that collects monitoring data
- Router−based flow aggregation can sometimes help

ROUTING OPTIMIZATION CONCEPTS IN IP NETWORKS

A. Destination-Based vs. Source/Flow-Based Routing

Two fundamentally different routing concepts exist, which strongly influence the optimization procedure and the achievable results: destination-based routing and source- or flow-based routing. Conventional routing protocols such as Open Shortest Path First (OSPF), Enhanced Interior Gateway Routing Protocol (EIGRP), Intermediate System to Intermediate System (IS-IS) follow the next-hop destination-based routing paradigm. Within each router the forwarding decision for an IP packet is based solely on the destination address specified in the packet header. No information about the origin or any other context of the packet is considered. Therefore, this routing procedure is simple and quite efficient. However, it imposes limitations on routing optimization, as illustrated in Figure 1.

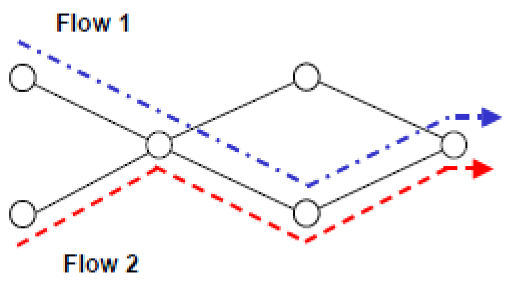

Figure 1

Whenever two traffic flows with the same destination cross each other's way they are merged and sent out over the same interface. This might cause traffic overload on some links, while other links are still only lightly utilized.

To overcome these limitations, new flow-based routing technologies such as Multiprotocol Label Switching (MPLS) have been developed. MPLS makes it possible to establish an overlay routing structure within the IP network – independently of the used routing protocol – and to set up explicit paths for individual traffic flows. Each MPLS-routed IP packet is marked with a special label, which the routers along the way consider for forwarding decisions. Thus, the routing pattern does not depend on the underlying routing protocol, but rather on the label and forwarding information that is stored in the MPLS routers. Instead of determining the path of a packet based on the destination address in a hop-by-hop manner, the path is now fixed by the router, which marks the packet with the appropriate label. This introduces a high degree of "routing freedom" as any desired routing pattern can be obtained.

B. Single-Metric vs. Dual-Metric Routing

In the case of destination-based routing protocols a router determines an outgoing interface based on metric values, which quantitatively describe the distance to a destination node. Most commonly, single additive metrics are assigned to every link and a shortest-path algorithm is used to determine the preferred path from each node to every other node in the network ("single-metric routing"). Link metrics often have physically relevant meanings such as "delay" or "cost", they can also be used in a generic way purely for the sake of routing optimization. By setting appropriate link metric values, one can indirectly influence and, thus, optimize the routing scheme.

Routing schemes exist, which allow more than one metric considered when computing the length of a path towards a destination node ("multiple-metric routing"). One example is Cisco's routing protocol EIGRP, which incorporates four metric types. However, only two of them are used by default: one additive metric ("delay") and one concave metric ("bandwidth"). In the following, we will focus on these two metric types and show how they can be used for routing optimization. The distance to a destination node is now computed by the normalized metric formula

$$M = \frac{1}{\min_i (bw_i)} + \sum_i d_i = \max_i (icm_i) + \sum_i d_i \;..$$

Parameter bw_i denotes the bandwidth of link i, while d_i refers to its static delay value. Thus, a router takes the sum of all delay values towards the destination node and adds a bandwidth component, which is the inverse of the smallest bandwidth along the path (“bottleneck”). From all possible path options, it selects the one with smallest path metric M. For further considerations we will refer to the bandwidth component as “inverse capacity metric” icm and take the maximum along the path instead of the reciprocal value of the bandwidth minimum.

Figure 2 illustrates the concept of bandwidth-delay sensitive routing. The distance metric M associated with path A-B-C-E is 7 (delay sum of 3 plus bandwidth component of 4), while path A-D-E has only an overall metric of 5. Therefore, router A selects router D as its next- hop neighbor towards E, and flow 1 is routed over A-D-E.

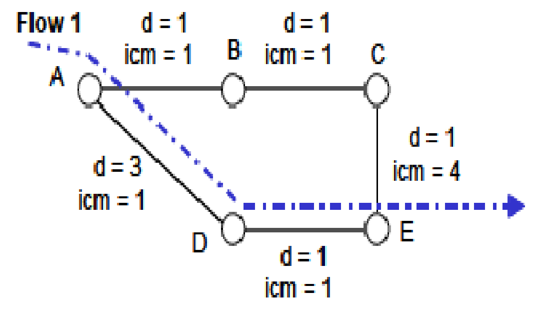

Figure 2

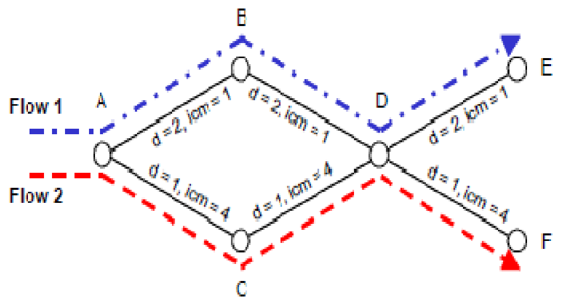

Figure 3

Depending on the values for d and icm, emphasis is either put on small overall delay (i.e., “short” paths), on high throughput, or on a mixture of both. In case metrics d are substantially larger than metrics icm, the overall path metric is mainly determined by delay values. Only when there are several alternatives with equal smallest delay sum, the bandwidth component really matters. The router then selects the outgoing interface with the largest possible throughput (“widest-shortest path”). In the opposite case (icm >> d), high throughput paths are preferred, and delay is mainly used to break ties (“shortest-widest path”). Whenever the two-link metrics are of the same order, no clear preference is given to either one of them. Link metrics are then used in their most generic form as means of routing optimization without any physically relevant significance.

Routing optimization based on dual-metric protocols has the potential to achieve better results than its single-metric counterpart. Since the second metric introduces some more flexibility, a greater variety of routing patterns can be realized. Note that it is always possible for dual-metric protocols to produce single-metric routing patterns. Simply setting all icm values to the same constant c would result in a single-metric shortest-path computation with metric function (M-c).

The benefit of dual-metric protocols can be demonstrated by means of the network scenario in Figure 3. Assume we have two traffic flows with different destinations, whose paths have several nodes in common. Let A be the first node where the two flows come together, and D be the last common node on their way. While single-metric shortest-path routing would merge the flows at node A and send both either via B or via C, dual-metric routing protocols can achieve the flow pattern given in the figure. For flow 1, the chosen path has a total metric of 7, while the link metrics along the route via C would sum up to 8. For flow 2 the situation is different and the total metrics of the upper and the lower path are 9 and 7, respectively. The trick is to use the inverse capacity metric to make one path option appear more costly for one traffic flow, while for the other flow a larger icm value has no extra effect (since it already experiences high icm values on further links, which the two flows do not share). However, the actual gain in QoS, which EIGRP can achieve over OSPF, depends greatly on the network topology and the traffic demands.

C. Equal-Cost Multi-Path ECMP

Another possible feature of routing protocols, which influences the optimization process, is load sharing. In destination-based routing protocols this capability is often implemented in form of the "equal-cost multi-path" concept. Whenever a router can reach a destination node via several paths with equal metric sums, it splits up the traffic evenly across all corresponding outgoing interfaces. Note, if dual-metric routing protocols are used, "equal cost" does not necessarily mean that all metric components are the same on all load-sharing paths. While metric combinations of several paths might be equal, their delay and bandwidth portions could be quite different.

Shortcoming of conventional IP systems

One shortcoming of conventional IP systems is the inadequacy of measurementfunctions. For example, a traffic matrix, which is a basic data set needed for traffic engineering, is difficult to estimate from interface statistics on IP routers.
The limitations of intradomain routing control functions are another issue with conventional IP systems. Interior Gateway Protocols (IGPs), such as Intermediate System–Intermediate System (IS-IS) and Open Shortest Path First (OSPF), commonly used to route traffic within autonomous systems (AS) on the Internet, are topology-driven and employ per-packet progressive connection control. Each router makes independent routing decisions using a local instantiation of a synchronized routing area link state database. Route selection is based on shortest path

computations using simple additive link metrics. This approach is highly distributed and scalable but flawed.
The flaw is that these protocols do not consider the characteristics of offered traffic and network capacity constraints when making routing decisions. This results in subsets of network resources becoming congested, while other resources along alternate paths remain underutilized.

Multi-Protocol Label Switching (MPLS)

MPLS traffic engineering provides an integrated approach to traffic engineering. Recent developments in multiprotocol label switching open new possibilities to address some of the limitations of IP systems concerning traffic engineering. Although MPLS is a relatively simple technology (based on the classical label swapping paradigm), it enables the introduction of sophisticated control capabilities that advance the traffic engineering function in IP networks. A particularly interesting aspect of MPLS is that it efficiently supports origination connection control through explicit label-switched paths (LSP).

An explicit LSP is one whose route is determined at the origination node. Origination connection control permits explicit routes to be established which are independent of the destination-based IP shortest path routing model. Once an explicit route is determined, a signaling protocol is then used to install the LSP. Through explicit LSPs, a quasi-circuit switching capability is superimposed on the IP routing model. When deployed in IP over SONET or IP over DWDM configurations, the traditional L3 and L2 functions are virtualized in one network element called the label switching router (LSR). Fewer network elements are required than with overlay alternatives, reliability is increased, and operating costs and queuing delays are reduced.

Additionally, MPLS simplifies network architecture and network design relative to the Overlay model. When MPLS is combined with differentiated services (DiffServ) and constraint-based routing, they become powerful and complementary abstractions for quality of service (QoS) provisioning in IP networks.

MPLS is best summarized as a “Layer 2.5 networking protocol”.
In the traditional OSI model:

- Layer 2 covers protocols like Ethernet and SONET, which can carry IP packets, but only over simple LANs or point-to-point WANs.
- Layer 3 covers Internet-wide addressing and routing using IP protocols.
- MPLS sits between these traditional layers, providing additional features for the transport of data across the network.

In a traditional IP network:

• Each router performs an IP lookup ("routing"), determines a next-hop based on its routing table, and forwards the packet to that next-hop.

• Rinse and repeat for every router, each making its own independent routing decisions, until the destination is reached.

MPLS does "label switching" instead:

• The first device does a routing lookup, just like before:
  - o But instead of finding a next hop, it finds the destination router.
  - o And it finds a pre-determined path from "here" to that final router.

• The router applies a "label" based on this information.

• Future routers use the label to route the traffic without needing to perform any additional IP lookups.

• At the destination router, the label is removed, and the packet is delivered via normal IP routing.

Originally, it was intended to reduce IP routing lookups. Label switching (or "tag switching") lookups use exact matching. The idea was to have only the first router do an IP lookup, then all future routes in the network could do exact match "switching" based on a label. This would reduce load on the core routers, where high-performance was the most difficult to achieve, and distribute the routing lookups across lower speed edge routers. The ability to control where and how traffic is routed on your network helps to manage capacity, prioritize different services, and prevent congestion.

<u>MPLS header</u>

MPLS is Layer 2.5 because it clubs best of both layers. Gives speed of Layer 2 for forwarding a packet but preserves the scalability and dynamic capabilities of Layer 3. Usually Layer 2 is homogeneous network whereas Layer 3 is heterogeneous. Figure 4 depicts the header format of MPLS and Figure 5 summarizes the meaning of each field.

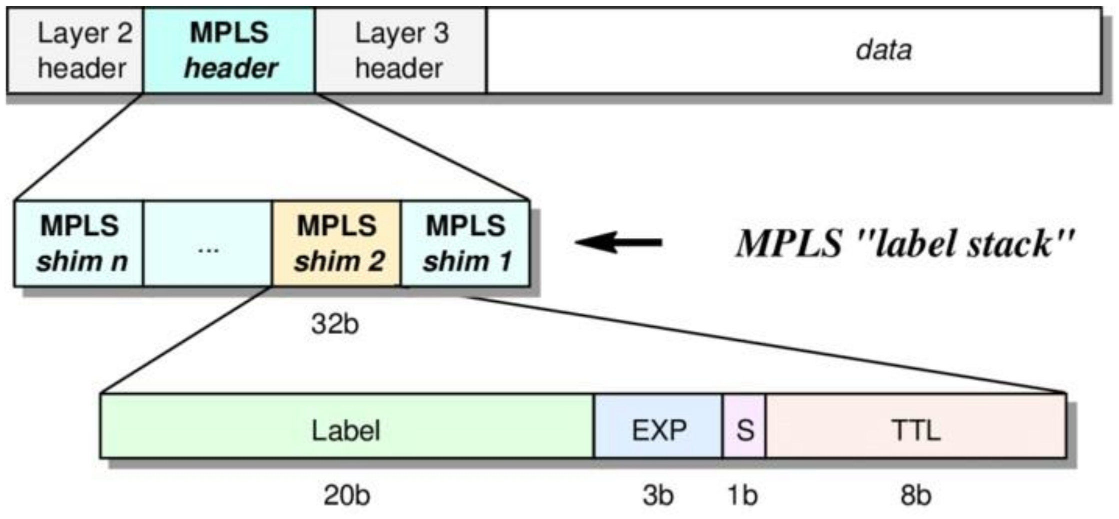

Figure 4

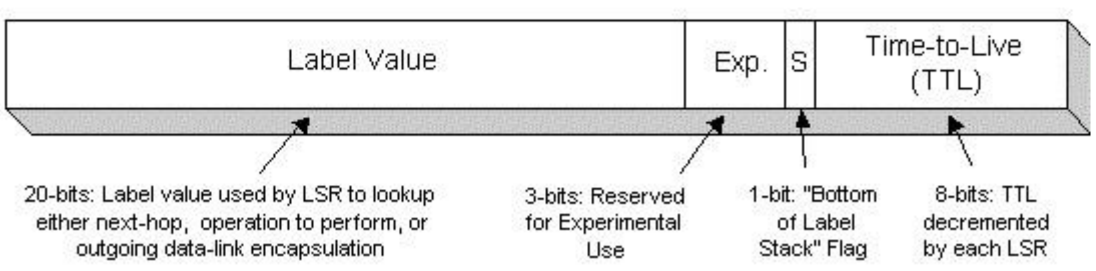
Label Value
Exp.
S
Time-to-Live (TTL)
20-bits: Label value used by LSR to lookup either next-hop, operation to perform, or outgoing data-link encapsulation
3-bits: Reserved for Experimental Use
1-bit: "Bottom of Label Stack" Flag
8-bits: TTL decremented by each LSR

Figure 5

MPLS Working – Basic Concepts

Label Switched Path ("LSP")

• One of the most important concepts for the actual use of MPLS.

• Essentially a unidirectional tunnel between a pair of routers, routed across an MPLS network.

• An LSP is required for any MPLS forwarding to occur.

Label Edge Router ("LER") or "ingress node".

• The router which first encapsulates a packet inside an MPLS LSP.

• Also, the router which makes the initial path selection.

Label Switching Router ("LSR") or "transit node"

• A router which only does MPLS switching in the middle of an LSP.

Egress Node

• The final router at the end of an LSP, which removes the label.

P – Provider Router

• A core/backbone router which is doing label switching only.

• A pure P router can operate without any customer/Internet routes at all.

• This is common in large service provider networks.

PE – Provider Edge Router

• A customer facing router which does label popping and imposition.

• Typically has various edge features for terminating multiple services like Layer 3- Virtual Private Network (L3VPN), L2VPN / Pseudo wires and Virtual Private LAN Service (VPLS).

CE is the "Customer Edge", the customer device a PE router talks to.

MPLS Signaling Protocols

To use an LSP, it must be signaled across your routers. An LSP is a network-wide tunnel, but a label is only a link-local value. An MPLS signaling protocol maps LSPs to specific label values.

There are two main MPLS routing protocols in use today:

- Label Distribution Protocol ("LDP"): A simple non-constrained (doesn't support traffic engineering) protocol.
- Resource Reservation Protocol with Traffic Engineering ("RSVP-TE"): A more complex protocol, with more overhead, but which also includes support for traffic-engineering via network resource reservations.

Most complex networks will need to use both protocols. LDP is typically used by MPLS VPN (data transport) services. But RSVP-TE is necessary for traffic engineering features. Most networks will configure LDP to tunnel inside RSVP.

MPLS Label Stacking

MPLS labels can also be stacked multiple times. The top label is used to control the delivery of the packet. When destination is reached, the top label is removed (or "popped"), and the second label takes over to direct the packet further. This is illustrated in Figure 6.

Some common stacking applications are:

- VPN/Transport services, which use an inner label to map traffic to specific interfaces, and an outer label to route through the network.
- "Bypass" LSPs, which can protect a bundle of other LSPs to redirect traffic quickly without having to completely re-signal every LSP, in the event of a router failure.

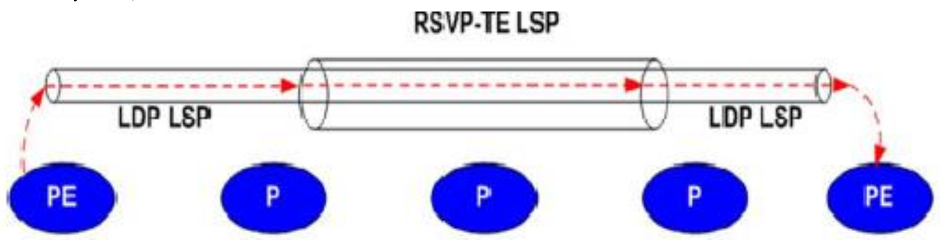

Figure 6

Traffic trunks

One feature of traffic engineering in IP networks is the traffic trunk concept. A traffic trunk is an aggregation of traffic belonging to the same class. It is essentially an abstract description of traffic that allows certain attributes of the traffic to be parameterized. It is independent of the underlying technologies. The problem of mapping traffic trunks onto a given network topology is a central issue of traffic engineering. In MPLS networks, traffic trunks are mapped onto the Network topology through the selection of routes for explicit LSPs. The terms LSP tunnel and traffic engineering tunnel (te-tunnel) are commonly used to refer to the combination of traffic trunk and explicit LSPs in MPLS.

LSP tunnels allow the performance of an operational network to be optimized in various ways. For example, if congestion problems caused by suboptimal routing are detected, LSP tunnels can be rerouted to alleviate the problem. LSP tunnels can be parameterized, and network resources can be allocated to them based on those parameters. Multiple LSP tunnels can be created between two nodes, and the traffic between the nodes divided among the tunnels according to some local policy. LSP tunnels permit the introduction of flexible and cost-effective survivability options.

Statistics derived from LSP tunnels can be used to construct a rudimentary traffic matrix. LSP tunnels can be used to redistribute traffic to address congestion problems caused by shortest path IGPs.

- Remember, an LSP is a “tunnel” between two points in the network. Under RSVP, each LSP has a bandwidth value associated with it. Using constrained routing, RSVP-TE looks for the shortest path with enough available bandwidth to carry a particular LSP.
- If bandwidth is available, the LSP is signaled across a set of links. The LSP bandwidth is removed from the “available bandwidth pool”. Future LSPs may be denied if there is insufficient bandwidth. They’ll ideally be routed via some other path, even if the latency is higher.
- Existing LSPs may be “preempted” for new higher priority LSPs. You can create higher and lower priority LSPs, and map certain customers or certain traffic onto each one. This isn’t traditional QoS, no packets are being dropped when

MPLS Fast Reroute

MPLS Fast Reroute improves convergence during a failure. By pre-calculating backup paths for potential link or node failures.

In a normal IP network

The best path calculation happens on-demand when a failure is detected. It can take several seconds to recalculate best paths and push those changes to the router hardware, particularly on a busy router. A transient routing loop may also occur, as every router in the networks learns about the topology change.

With MPLS Fast Reroute

The next best path calculation happens before the failure occurs. The backup paths are pre-programmed into the router FIB awaiting activation, which can happen in milliseconds following failure detection. Because the entire path is set within the LSP, routing loops cannot occur during convergence, even if the path is briefly suboptimal.

DNS and Global Server Load Balancing (GSLB)

Load balancing is a technique that is used for distributing the workload evenly across computing machines, networks, processing units etc. It basically enhances utilization of resources and enables maximum throughput with minimum response time hence avoiding overloading of a single server.

An extension of the same technology is the Global Server Load Balancing (GSLB). With this technology the Internet traffic can be distributed among different data-centers located at different locations of the planet. For instance, large organizations such as Facebook, eBay, and Microsoft use this technology to load-balance their web services. This might be for proximity reasons, because a user might be redirected to the closest available resource, or to keep redundant services available in case of datacenter failures.

This technology is highly efficient in avoiding local downtimes. There is a Master GSLB which monitors the health and responsiveness of the Slave sites, which in turn identifies the sites that is available and can offer the best response.

Ideally GSLB would include:

1. Redirection of requests to other closest sites (domain controller -DC) in case the usual DC or server does not respond.
2. Forwarding the visitor requests to the site that is closer to the place from where the request is raised from a geographic point of view.
3. In case a threshold is reached at a site, the requests are forwarded to other site which may be at a different geo-location. Its algorithm would calculate the shortest distance from when the request has been raised.

GSLB is based upon the use of DNS. So for instance when a user wants to go to *www.citrix.com*, that user's DNS client will do a number of queries against the different DNS servers, until it gets a response from an authoritative source for that domain.
An example might be where NetScaler, which is the authoritative nameserver for that domain, responds with an A-record to that user that is one of the load-balanced vServers attached to the domain. Before that record is handed out, NetScaler has done an evaluation of the health state of the different services the user is trying to access, using SNMP based load monitors, **Metric Exchange Protocol** (**MEP**), and explicit monitors.

GSLB also consists of different components:

- GSLB sites (that represent a geographical location or datacenter)
- GSLB services (that are linked to a load-balanced vServer)
- GSLB vServers (that consist of multiple GSLB services served from GSLB sites)
- Domains (that on NetScaler are either authoritative or act as a proxy on its behalf)
- MEP used between nodes in each site to exchange information about the state and load on the site

MEP is a proprietary protocol used by NetScaler to communicate different GSLB site metrics, network metrics and persistence info to the other GSLB sites. Communication with MEP happens on port 3011 or 3009 for secure communication. If we want to use load-balancing methods such as proximity or RTT, we need to have MEP enabled; if not, it will fall back to round robin.

Figure 7 depicts an example with a small design layout of GSLB using MEP.

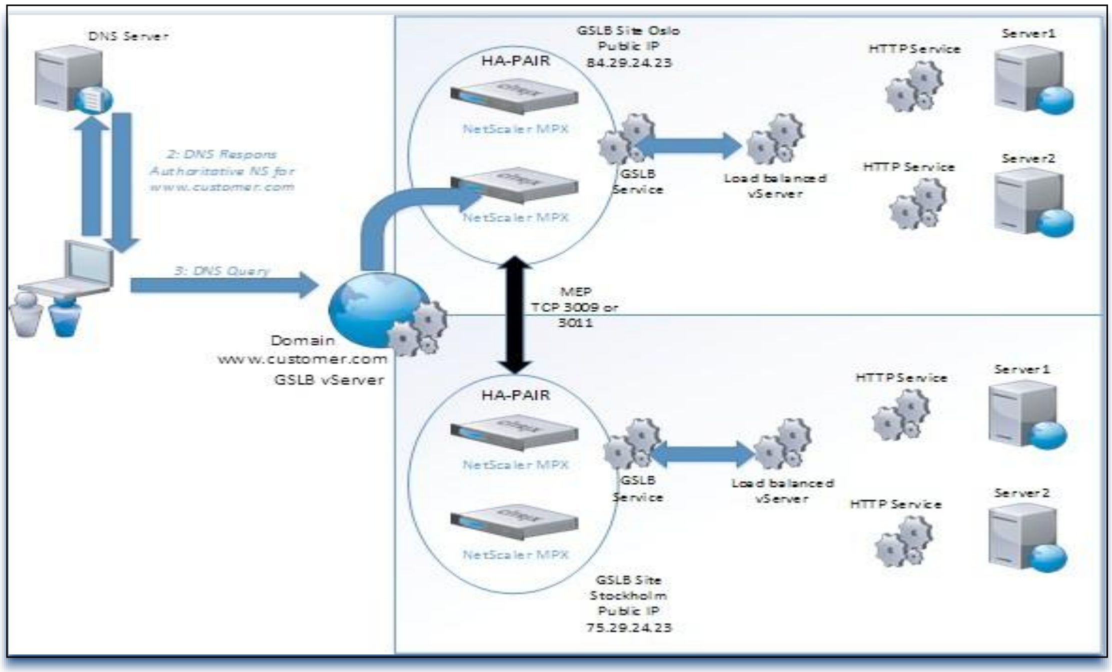

Figure 7

GSLB Scenarios and Requirements:

In GSLB, a website or services are hosted on different servers located at different geo-locations. Let's consider, a visitor is browsing www.sample-uk.com from US. In usual web hosting, this request would be sent straight to the server where it is hosted, and the request would be served from the UK server only. But in Global Load Balancing architecture, since the website is hosted on servers across different geo-locations, the user trying to access the website from US, would be served from the server that is in US. The GSLB in a regular time interval checks the health of the slave servers for providing the best response hence directing the traffic to those servers.

There aren't any special requirements for implementing GSLB for your website. You simply need to choose a server as you do in the usual way, but since you intend to apply load balancing across the globe, you are required to select servers in different locations. Your website data is placed on servers of your choice in all the selected locations.

Note: For Global Load Balancing, it is recommended to choose servers of equivalent configurations.

There are ideally two methods how GSLB can be configured:

A. GLB – Global Load Balancing
B. DNS Load Balance

A. GLB – Global Load Balancing:

In the technique of GLB, we use a primary server also known as the Master Server and is connected to Slave Servers located in different geo-locations i.e., different Domain Controllers (DC's). Whenever a request is raised, the Master Server checks the location from where it was initiated. Then cookies are set, and it redirects the request to the nearest server location.

Process Flow:

1) The user (in this case user is in US) types the website URL www.sample-uk.com in the browser. Since global load balancing is applied, the request would first go to the Master serverwhich is in UK datacenter.
2) The Master Server would send a query to the requester for querying IP address, upon confirmation of the users IP address; cookies are set which contains every information on the user system.
3) This information is forwarded to the Master Server which is in the meantime checking the health and level of responsiveness of the slave servers.
4) Now the Master Server in the GSLB architecture would redirect the request to nearest Slave Server (i.e., in US).
5) The users get a response and can view the website which is served from the US server and not the UK servers.

All these processes happen in fraction of seconds without the knowledge of the users. A constant synchronization takes place between Master and Slave servers across all the DC's.

B. Global DNS Load Balance:

Load Balancing can be fulfilled using DNS. In this the response to the DNS requests depends on requester's geographic IP location and the response from DNS. Let's consider a similar example as above: If a user requests for www.sample-uk.com, it is sent to GLB DNS Server which would check the IP of the visitor. Then it would identify the server that is closest to the location or to the one that has the minimum response time, and the traffic is forwarded or redirected.

Process Flow:

1) The user based in Malaysia tries to access the site www.sample-uk.com. In this case the DNS

Request is sent to the ISP and further to the GSLB DNS Server.

2) Then, the GSLB DNS Server forwards the request to Name Servers of all the Countries or to the GSLB Server located various geo-locations.
3) Within fraction of second, the GSLB receives the response from every server, it then analysesthe response time and identifies the Closest Server Location from the Users Request and then redirects it to that server.
4) The users in this way get connected to the closest server. In this scenario, the user request is initiated from Malaysia, hence the closest server as identified by GSLB is in India, and hence the website gets served from server located in the Indian DataCenter.

Balancing algorithms for GSLB

There are two options that are specific for GSLB: `STATICPROXIMITY` and `RTT`. `STATICPROXIMITY` can use a custom location database to determine where to send clients, or we create our own custom entries.

RTT is a dynamic proximity load balancing method that measures the delay between a client's local DNS server and the data resource it is trying to access. After it has measured the RTT, NetScaler then exchanges the metric with the other sites using MEP and then sorts out the RTT between the different sites and the client. Then it makes its load balancing decision and will dynamically redirect the client request to the GSLB site that has the lowest RTT. NetScaler uses different probes to determine the RTT. By default, it uses PING first; if that fails it tries a DNS UDP probe. If that does not work, it tries a DNS TCP probe.

If all probes fail or RTT metrics are not available, NetScaler will fall back to round-robin load-balancing.

If we want to change the order of the probes, we can do so using the following command:

```
set gslb parameter -ldnsprobeOrder DNS PING TCP
```

This might be necessary for instance if we do not allow external PING traffic via the firewall or UDP.

After we have configured the load balancing method, we must add a domain to the vServer. That can be done under the domain pane in the vServer options.

GSLB is a complex feature that might often lead to headaches if not properly configured. A good way to start troubleshooting a configuration is to look at the domains configured in the GSLB setup and look at the **visualizer**; Visualizer will give a graphical overview of how the configuration is for that domain name.

DataStream

The DataStream feature provides us with an intelligent mechanism for load balancing SQL traffic to backend endpoints. This might for instance be based upon READ and WRITE requests, where READ requests are forwarded to some dedicated backend SQL servers while WRITE requests are dedicated to a master cluster.

This can also be performed down to a database level, where we can load-balance specific databases, or can even be based upon other attributes such as usernames, packet size, and so on. The DataStream feature is only supported for MySQL and MS SQL databases.

Content switching

Content switching is the ability to direct requests sent to the same URL to different servers with different content. Consider the scenario where, when you type the URL http://url1.com, you are redirected to one backend service and, if you type another URL `http://url1.com/videos`, you are redirected to another backend service even though it shares the same URL. There are also other ways besides looking at the URL to do content switching. Some examples are:

- Language
- Cookie
- IP port source/destination
- IP address source/destination
- The HTTP method POST/GET

This feature is commonly used when, for instance, delivering web services for mobile devices. If we think about accessing a website with a mobile device, we often use the same URL as we normally do but are being redirected to a mobile-friendly website on another backend resource. This is of course more user-friendly but gives us a lot of flexibility as well to split mobile traffic with regular desktop-based traffic. This also allows us to do TCP optimization based upon device connections.

In essence, content switching consists of a vServer, policies, and actions. The policies define expressions that are then evaluated to see if they trigger an action. The actions always point to a target load-balanced vServer.

Let's look at this from a design perspective.

We have a website called url1.com that is accessed by two types of devices: mobiles and regular computers. We therefore created a content-switching vServer that has two policies. The first policy checks if the user-agent is a mobile device; if so, it redirects the request to a vServer that is load-balancing the mobile device front-end for that service. The other policy is there to handle all other traffic, which is redirected to another load-balanced vServer that has the regular website traffic. This is depicted in Figure 8.

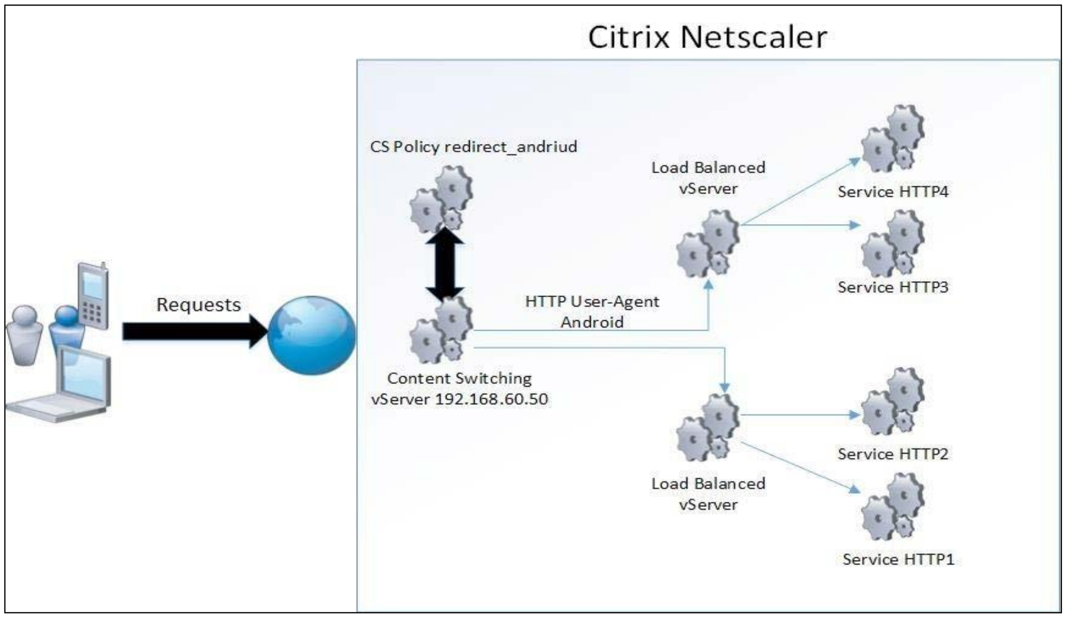

Figure 8

So, let's set up a content switching vServer for the first scenario. First, we need to enable the content switching feature; this is done by typing the command at CLI:

```
enable feature contentswitching
```

Next, we need our two load-balanced vServers in place before setting up content switching. In this scenario we just set up some vServers by using CLI.

```
Add  server  hostname  192.168.60.20   Add
server  hostname  192.168.60.21 Add  server
hostname  192.168.60.22 Add  server  hostname
192.168.60.23
```

Then we create two different services, one for mobile and another for regular HTTP traffic:

```
Add service mobile-http-1 192.168.60.20 http 80

Add service mobile-http-2 192.168.60.21 http 80

Add service http-1 192.168.60.22 http 80

Add service http-2 192.168.60.23 http 80
```

Next, we need to load-balance the vServer:

```
add lb vserver HTTP-mobile-lb http 192.168.60.24 80add lb
vserver HTTP-lb http 192.168.60.25 80
```

It is important to note that we do not need to add IP addresses to the load balanced vServers if we intend to use them with content switching, since the communication between the content switching vServer and the lb vServer is only internal. To add a lb vServer without an IP address you need to do it using the CLI, by using the earlier example but without the IP address.

Lastly, we need to bind the services to the load-balanced vServers:

```
bind lb vserver HTTP-mobile-lb mobile-http1 bind lb
vserver HTTP-mobile-lb mobile-http2 bind lb vserver
HTTP-lb http-1
bind lb vserver HTTP-lb http-2
```

It is important to note that by default the content switching server does not check whether the backend load-balanced resources are up or not. This is because if we enable the status check this means that, for instance, if one of four load-balanced vServers is down, the content switching vServer goes down.

Now we need to define the policies. Now policies, as I mentioned earlier, consist of different expressions to do an evaluation of the request. This can also be done using CLI but in this case I am going do to it using the Web GUI.

There are many ways to do it this. A simple way is to create an expression that checks the user-agent string and then redirects the users to the connected backend service.

Therefore, we can use the expression:

```
HTTP.REQ.HEADER("User-Agent"). CONTAINS("Android")
```

If we want to combine multiple expressions—for instance, a policy that binds all mobile user-agents in one expression—we can use logical operators to combine multiple expressions.

AppQoE

AppQoE is a combination of different features on NetScaler—more precisely, HTTPDoS, priority queuing, and SureConnect.

This feature allows us to prioritize traffic based upon different parameters—for instance, where the endpoint is coming from or what kind of endpoint is connecting with the use of expressions. So, for instance, let's say we have an e-commerce website that serves both mobile and desktop users. From experience, we see that desktop devices are more likely to buy something from the site than mobile devices. In that case, if the e-commerce site reaches its maximum threshold or bandwidth, we need to prioritize the traffic and then we want desktop users to get access before mobile users.

To configure AppQoE, we need to define a policy that contains an expression and an action. For instance, this might be an expression containing Android devices:

```
HTTP.REQ.HEADER("User-Agent"). CONTAINS("Android")
```

That will then have an action attached to it; the action might be what to do with the traffic that matches the expression. This might for instance be to prioritize the traffic, or to display a custom wait page, and so on.

In order to set up AppQoE, we first need to enable the feature either using CLI:

```
Enable feature appqoe
```

Then we start by creating an action, by going into the **Actions** pane and clicking **Add**. So, let's create an action that will apply to Android devices.

Here there are some actions that can be set:

- **Action Type**
  - ACS
  - NS
  - No action
- **Priority**
- **Policy Queue Depth**
- **Queue Depth**
- **DOS Action**

Action Type defines what action to take when the threshold is reached. There are three types of action we can take. **Alternative Content Service** (**ACS**) can be used to redirect requests to another vServer on the NetScaler.

We also have the NSoption under **Action Type**; this allows us to, for instance display a custom HTML page to end users trying to connect to the vServer. In order to choose a custom HTML page, we must first upload one to the NetScaler.

We also have the option to not set an action type; this will not allow NetScaler to display any custom content and any connections that are over the threshold will be placed in the lowest priority queue.

Priority defines what priority the traffic should have. There are four values to choose from here: `HIGH`, `MEDIUM`, `LOW`, `LOWEST`. Traffic will always be processed in that order.

As an example, we could have a policy aimed at mobile users, where we have defined a mobile traffic with the priority set to `MEDIUM` by default, and another policy with priority set to `HIGH` for desktop users. This would mean that all desktop user traffic would be processed first, and mobile traffic would be processed afterwards.

Policy Queue Depth is a value we define on how many connections fall within this policy. Subsequent connections are assigned to the `LOWEST` priority rights.

Queue Depth is a value that defines how many connections fall within this policy on this particular priority; subsequent connections are assigned to the `LOWEST` priority rights.

Maximum Connections defines how many concurrent connections can be active before NetScaler triggers an action; likewise, delay is defined in microseconds before NetScaler triggers an action.

Lastly, we have **DOS Action**, which is in essence the HTTPDoS feature; this feature will be covered in much greater detail. In order to define a HTTPDoS action within AppQoE, we just have to define an expression that might be filtering on an HTTPuser agent and define an action type that can be `SimpleResponse` and `HICresponse`.

It is important to note that we do not define a value for when the DOS action will trigger; this is done globally for AppQoE. Here we can define the average number of client connections that can queue up before the DOS action triggers.

After we have created an action, we have to bind it to a policy. Policies need to contain an expression that can be using HTTP user agents, IP source, and so on. After we have created the policy, we need to bind it to a vServer.

When no priority is set, NetScaler will fall back to the lowest priority.

We can look at the statistics for an AppQoE policy by using the CLI command:

```
Stat appqoe policy policyname
```

TCP profiles

Since most of the traffic going through the Internet today is based upon TCP, it is important that it is properly configured. Since endpoints might come from many different locations—such as a PC connected to a fast Ethernet connection or a mobile device using 3G/4G—the connection needs to behave differently.

TCP has many different parameters that by default on NetScaler are not configured. This is because NetScaler is configured to fit into most environments, and even though many of the TCP options might give a performance boost, they might also degrade performance if not properly configured.

This is where TCP profiles come in. TCP profiles allow us to configure a set of different TCP settings into a profile that we can then attach to a virtual server or a set of services. For instance, TCP traffic should behave differently for mobile devices browsing a mobile website located on a load-balanced vServer than for traffic going to backend resources connected on a fast Ethernet LAN.

Citrix has created a lot of built-in TCP profiles that are custom made for different types of connection and service. By default, NetScaler uses a built-in profile called `nstcp_default_profile`; This default profile has most of the different TCP features turned off, to ensure compatibility with most infrastructures. The profile has not been adjusted much since it was first added in NetScaler.

We also have other TCP profiles that are suited for particular services, such as `nstcp_default_XA_XD_profile`, which is intended for ICA-proxy traffic and has some differences from the default profile such as:

- **Window Scaling**

- **Selective Acknowledgement**
- **Forward Acknowledgement**
- **Use Nagle's algorithm**

Window Scaling is a TCP option that allows the receiving point to accept more data than allowed in the TCP RFC for window size before getting an acknowledgement. By default, the window size is set to accept 65,536 bytes. With **Window Scaling** enabled, it basically bitwise-shifts the window size. This is an option that needs to be enabled on both endpoints in order to be used and will only be sent in the initial three way handshake.

Selective Acknowledgement (**SACK**) is a TCP option that allows for better handling of TCP retransmission. In the scenario where two hosts communicate with SACK not enabled and suddenly one of the hosts drops out of the network and loses some packets, when it comes back online it receives more packets from the other host. In this case, the first host will ACK from the last packet it got from the other host before it dropped out. With SACK enabled, it will notify the other host of the last packet it got before it dropped out, and the other packets it received when it got back online.This allows for faster communication recovery since the other host does not need to resend all the packets.

Forward Acknowledgement (**FACK**) is a TCP option that works in conjunction with SACK and helps avoid TCP congestion by measuring the total number of data bytes outstanding in the network. Using the information from SACK, it can more precisely calculate how much data it can retransmit.

Nagle's algorithm is a TCP feature that tries to cope with small packet problems. Applications such as Telnet often send each keystroke within its own packet, creating multiple small packets containing only 1 byte of data, which results in a 41-byte packet for one keystroke. The algorithm works by combining a number of small outgoing messages into the same message, thus avoiding overhead.

ICA is a protocol that operates by sending many small packets, which might create congestion on the network; this is why Nagle is enabled in the TCP profile. Also since many might be connecting using 3G or Wi-Fi, which might in some cases be unreliable when it comes to changing channel, we need options that require the clients to be able to reestablish a connection quickly and that allow the use of SACK and FACK.

Note that Nagle might have negative performance on applications that have their own buffering mechanism and operate inside the LAN. Since ICA-proxy mostly uses TCP, except for Framehawk traffic using DTLS, this should be enabled for NetScaler Gateway vServers.

If we take a look at another profile such as `nstcp_default_lan`, we can see that FACK is disabled; this is because the resources needed to calculate the amount of outstanding data in a high-speed network might be too much for the CPU to handle.

So, let's go into some of the other different TCP parameters in slightly greater depth:

- **Maximum Burst Limit**: This setting controls the burst of TCP segments on the wire in a single attempt. A higher limit here ensures faster delivery of data in a congestion-free network. Limiting bursts of packets helps to avoid congestion.
- **Initial Congestion Window size**: Initial congestion window defines the number of bytes that can be outstanding in the beginning of a transaction. The default size is 4 (that is 4*MSS).

- **TCP Delayed ACK Time-out (msec)**: To minimize the number of ACK packets on the wire, this NetScaler feature by default sends ACK only to a sending node if the NetScaler receives two data packets consecutively or the timer expires; the default timeout is 200ms.
- **Maximum ooo packet queue size**: This feature allows out-of-order packets in TCP streams to be cached in system memory and reassembled before they are processed by NetScaler. By default, the value is 64; we can define a value of 0that means no limit, but this will put a lot of strain on the NetScaler memory usage.
- **MSS** and **Maximum Packets per MSS**: With this setting we can specify the maximum segment size to be used for TCP transactions. It is important to note that jumbo frames will override this setting, since MSS is TCP sizes and MTU is IP packets, which is lower in the network layer. So, if we specify a higher MTU size on the interface by enabling jumbo frames the MSS size will increase as well.
- **Maximum Packets Per Retransmission**: This setting controls how many packets are retransmitted in a single attempt. This is used with TCP Reno that used a partial ACK value to notify NetScaler that is has received some of the packets but not all.
- **TCP Buffer Size (bytes)**: The buffer size is the receiver buffer size on the NetScaler. The buffer size is advertised to clients and servers from NetScaler, and it controls their ability to send data to NetScaler. The default size is 8K and in most cases it will be beneficial to increment this when communicatingwith internal server farms.

Another important aspect of these profiles is the TCP congestion algorithms. For instance, `nstcp_default_mobile` uses the Westwood congestion algorithm, because it is much better at handling large bandwidth-delay paths such as wireless.

The following congestion algorithms are available in NetScaler:

- Default (based upon TCP Reno)
- Westwood (based upon TCP Westwood+)
- BIC
- CUBIC
- Nile (based upon TCP-Illinois)

What is worth noting here is that Westwood is aimed at 3G/4G connections, or other slow wireless connections. BIC is aimed at high-bandwidth connections with high latency, such as WAN connections. CUBIC is almost like BIC but not as aggressive when it comes to fast-ramp and retransmissions. It is important to note however that CUBIC is the default TCP algorithm in Linux kernels from 2.6.19 to 3.1.

Nile is a new algorithm created by Citrix and was introduced in NetScaler 11, which is based upon TCP-Illinois. This is targeted at high-speed, long-distance networks. It achieves higher throughput than standard TCP and is also compatible with standard TCP.

There are also some other parameters that are important to think about in the TCP profile.

One of these parameters is Multipath TCP. This feature permits endpoints that have multiple paths to a service, typically a mobile device that has WLAN and 3G capabilities and allows the device to communicate with a service on a NetScaler using both channels at the same time. This requires that the device support communication on both methods and that the service or application on the device support **Multipath TCP** (**MPTCP**).

As I mentioned, NetScaler includes a range of different profiles that are aimed at different connections. Here is a summary of the different TCP profiles and where they should be used:

- `nstcp_default_tcp_lfp`: This profile is useful for long fat pipe networks (WAN) on the client side. Long fat pipe networks have long delay, high bandwidth lines with minimal packet drops.
- `nstcp_default_tcp_lnp`: This profile is useful for long narrow pipe networks (WAN) on the client side. Long narrow pipe networks have considerable packet loss once in a while.
- `nstcp_default_tcp_lan`: This profile is useful for backend server connections, where these servers reside on the same LAN as a NetScalerappliance.
- `nstcp_default_tcp_lfp_thin_stream`: This profile is similar to the `nstcp_default_tcp_lfp` profile. However, the settings are tuned for small packet flows.
- `nstcp_default_tcp_lnp_thin_stream`: This profile is similar to the `nstcp_default_tcp_lnp` profile. However, the settings are tuned for small packet flows.
- `nstcp_default_tcp_lan_thin_stream`: This profile is similar to the `nstcp_default_tcp_lan` profile. However, the settings are tuned to small packet flows.
- `nstcp_default_tcp_interactive_stream`: This profile is similar to the `nstcp_default_tcp_lan` profile. However, it has a reduced delayed ACKtimer and ACK on PUSH packet settings.
- `nstcp_internal_apps`: This profile is useful for internal applications on a NetScaler appliance. This contains tuned window scaling and SACK options for the required applications. This profile should not be bound to applications other than internal applications.

Note that TCP profiles can be attached to a service, for backend communication, and to a vServer that communicates with the different client endpoints. In order to add a TCP profile just go into a service or vServer, go into the **Profiles** pane, and choose a TCP profile from the list.

SSL profiles

We also have SSL profiles that define how SSL traffic should be processed on a NetScaler. With SSL it means that we can define that cipher groups can be used and how and when NetScaler should encrypt data. The common issue with SSL traffic is that the more advanced the encryption we choose, the less performance we get; this is because it requires more compute capabilities to encrypt data using more advanced algorithms such as AES instead of, for instance, RC4.

When creating SSL profiles, we need to define if they are going to be frontend or backend. Frontend SSL profiles can only be attached to virtual servers, while backend ones can only be attached to services.

There are a few parameters that we should be aware of when configuring SSL profiles:

- **Quantum size**
- **PUSH Encryption Trigger**
- **Protocols**
- **Enable DH Param**

The **Quantum size** defines how much data in KB should be processed before it is encrypted. By default, this value is 8,192 KB; if we are hosting a service that provides media downloads, for instance, or large files in general, we will get a benefit if we change this to a larger quantum size.

By default, the **PUSH Encryption Trigger** value is used to tell NetScaler how long it should wait before consolidating the data and encrypting it.

For instance, ICA-proxy sessions have the PSH flag set, which means that NetScaler should always forward encrypted traffic without delay; if we set this value to 5ms, NetScaler will gather data for 5ms before sending it out back to the client. By default, this value is set to 1ms; in an environment with a lot of ICA-proxy sessions the network might be congested because of the high number of small packets that need to be encapsulated.

Another important aspect of the SSL profile is determining the protocol to use. By default, an SSL-based vServer can communicate with SSL 3, TLS 1, TLS 1.1, and TLS 1.2. As a recommendation, SSL 3 and TLS 1 should always be turned off because of security risks, and most clients and browsers today support the use of TLS 1.1and 1.2.

Enable DH Param enables NetScaler to regenerate a new Diffie-Hellman private- public pair after a number of transactions. This feature needs to be enabled and defined to, for instance, a value of 1000; we also need to bind it with a DH key. This will make NetScaler regenerate a new key pair after 1,000 sessions to ensure that a malicious attacker cannot use a compromised DH key to get access to information.

Another aspect of SSL is the kind of ciphers to use. These ciphers decide what kind of protocol to use, what kind of encryption algorithm should be used, the key exchange algorithm, and what kind of message authentication code algorithm to use. These ciphers are not defined in SSL profiles but are defined under SSL settings for each vServer.

The different ciphers can be found under **Traffic Management | SSL | Cipher Groups**. In order to test our SSL capabilities we can use a website called `https://www. ssllabs.com/` that can verify the level of security on our external web services.

Introduction to Optimization and Security

Client challenges Continuous high-profile incidents of organized hacking activities against corporations highlight the importance of protecting organizations against risk, while enabling them to continue to do business effectively and efficiently. A company's inability to comply with security standards is generally due to the inefficiency of their processes and not necessarily their tools.

In general, the implementation phase is accomplished by changing the configuration and operation of the organization's information systems to make them more secure. It includes changes to the following:

- Procedures (for example, through policy)
- People (for example, through training)
- Hardware (for example, through firewalls)
- Software (for example, through encryption)
- Data (for example, through classification)

Firewall Rules Optimization:

It is worth pointing out a common design feature of firewalls: the rule set's priority. The majority of (if not all) firewalls will use the first rule that exactly matches the conditions of the packet under observation. This means that in the previous example, the first rule matched the packet originating from 1.1.1.1 destined for 2.2.2.2 and the firewall did not continue to apply the remaining rules. Similarly, for the packet originating from 2.2.2.2 destined for 3.3.3.3, the last rule, DENY ALL, matched, which resulted in the firewall dropping the packet. This behavior leads to an interesting optimization problem for firewall administrators. To reduce network delay introduced by the firewall, administrators will typically move the most likely packetfiltering rule to the top of the list, but in doing so, they must ensure that the rule does not negate another rule further down the list. An example of such an optimization in which the administrator erroneously organized the packet filter list would be

```
ALLOW host 1.1.1.1 to host 2.2.2.2
DENY ALL
ALLOW host 3.3.3.3 to 1.1.1.1
```

In this example, the firewall administrator has placed an ALLOW rule after the DENY ALL rule.
This situation would prevent the processing of the last ALLOW rule.

IDS optimization

In addition to Intrusion Detection System (IDS) evasion techniques, the network environment that the IDS sensor monitors can affect the sensor's ability to detect malicious activity. Placement of the IDS sensor is key to monitoring the appropriate traffic. Overlooking sensor placement leads to visibility issues, as the sensor will not monitor the correct traffic. Placement in high-traffic areas can severely affect the performance of the IDS sensor. Sensors in high- traffic environments require a great deal of hardware to perform packet inspections. Packet inspections become more resource intensive as the amount of traffic increases; if the sensor does not have enough resources, it will fail to detect the malicious traffic. This results in the IDS not creating an alert about the malicious activity. The rule set used by the IDS sensor also affects the sensor's detection performance.

To increase performance, each IDS vendor uses different rule set optimization techniques. Despite the optimization techniques used, the sensor checks all traffic monitored for signature matches. Checking traffic with a smaller set of rules will result in faster performance but fewer rules with which to detect malicious content. Larger rule sets will perform slower than a smallerset but have more rules for detecting malicious activity. This shows the need for compromise between speed and threat coverage. Threat coverage shows the need for another compromise. An overflow of alerts will dilute critical alerts and valuable information with low-priority alerts and useless data. This dilution caused by excess noise makes triaging alert investigation difficult. The rule set for a sensor needs constant attention and custom tuning to reduce the number of alerts about legitimate traffic.

The last consideration for sensor placement involves inline devices. Inline devices physically sit between two network devices and can block malicious activity; however, legitimate traffic can also match signatures for malicious activity. This situation occurs often and results in the sensor blocking legitimate traffic. Another situation in which an IDS device can block traffic occurs when the sensors go offline or are overwhelmed with traffic. If the device does not fail to open in the event of system failure, then the device will block all trafficat its network interface. The inline device will also drop traffic if it exceeds its processing power.

Despite the issues facing IDSs, they are still beneficial to the security of a network. Proper consideration to the network environment that the IDS sensor will monitor is a must. An appropriate operating environment can reduce the issues previously discussed that plague a sensor's ability to detect malicious activity. Supplementing a proper network environment with continuous updates and tuning of the sensor's rule set will provide excellent coverage for most malicious events. IDS devices provide an invaluable stream of information to aid in security investigations and to improve the overall security of a network. IDS sensors can improve security by detecting a network's vulnerable areas and inbound attacks that can threaten the network. In cases involving an inline sensor, an IDS device can greatly improve network security by blocking malicious activity before it performs malice.

Conclusion

Complex networks need the support of detailed measurement. Measurement and Models play a crucial role in constructing a real time network wide view, detecting, diagnosing, and fixing problems. Routing optimization is one aspect from traffic diversion over links whereas GSLB is needed to service the client requests from the nearest server and not to burden single server or data center. TCP and SSL profiles are essential for performance while operating on diverse underlying networks. Security optimization ensures that the organizations are less vulnerable to threats arising from Internet, this requires configuring the security policies appropriately. Traffic Management can drastically increase the available bandwidth for mission critical applications, thus benefiting the organization tremendously.

References

https://datatracker.ietf.org/doc/html/rfc3272

http://techgenix.com/ssl-acceleration-offloading-security-implications/

http://www.kurtsalmon.com/en-us/cioadvisory/vertical-insight/362/Information-Security- Process-Optimization

https://technet.microsoft.com/en-us/library/bb821283.aspx

https://www.eukhost.com/kb/global-server-load-balancing/

http://docs.citrix.com/en-us/netscaler/11/traffic-management/ssl/supported-ciphers-list-release-11.html