Saimaa University of Applied Sciences
Faculty of Technology, Lappeenranta
Information Technology, Double Degree

Ondřej Filip

OPTIMIZATION OF APPLICATION DELIVERY CONTROLLERS DEPLOYMENT

Bachelor's Thesis 2011

ABSTRACT

Ondřej Filip

Optimization of Application Delivery Controllers Deployment, 30 pages, 1 appendix

Saimaa University of Applied Sciences, Lappeenranta

Faculty of Technology, Double Degree Programme

Tutors: Pasi Juvonen, M. Sc. (Eng.), Pasi Sutinen & Taneli Lehtonen, Tieto Corporation

The Goal for the thesis was to gather knowledge about f5 BIG-IP application delivery platform deployment in a company, describing the platform, and creating guides for common usage scenarios while using virtualization to simulate real deployment.

Current configurations of devices and also manufacturer deployment manuals were examined. Interviews were done with network specialists working with f5 appliances.

There are presented drivers for application delivery devices as such in the thesis. Then the f5 BIG-IP LTM application delivery controller platform characteristics are described and configuration guides for common usage scenarios based on interviews with employees and manual study are formulated.

Keywords: ADC, Application, Application Delivery Controller, Load Balancing

CONTENTS

TERMINOLOGY

802.1q – standard defined by IEEE, describes the virtual LAN implementation

application delivery controller – appliance with advanced load balancing capabilities

CA – abbreviation of certificate authority, entity in certificate issue process. Verifies identity and issue certificates to other entities.

CRL – abbreviation of certificate revocation list, list maintained by CA which contains all certificates which are not to be used but have not expired yet

CSR – abbreviation of certificate signing request, file in certificate issue process, it contains information needed for certificate issue and is submitted to CA for signing.

guest system – in virtualization, virtualized system, system running inside host system, see *host system*.

hash calculation – mathematical function that converts larger data object to small datum, its output is called hash value. Hash value has different forms and is used in data structures, data integrity checks etc.

host system – in virtualization, system which uses virtualization application to run virtual systems

HTTP cookie – textual information contained in HTTP headers and stored on the client computer for the purpose of creating stateful session. For detailed description see Kristol (2000).

NIC – abbreviation of network interface card

pool – in networking, abstraction of the group of items: service providers, addresses and others; used for configuration simplification

1 INTRODUCTION

In the course of time we have seen how our lives have become more and more dependent or influenced by services and tools provided by computers and information systems. As those are more or less integrated into our daily routines it is inevitable that their unavailability can be a risk or – in better case – a nuisance for our lives, depending on the exact case.

Therefore there is need for high availability of such services which means a certain level of fault tolerance. With higher amounts of requests per certain time, the possibility of easy scaling of the solution is also necessary. The load balancing goal is to distribute the workload to multiple hosts with regards to their status in order to achieve the mentioned capabilities.

This work discusses load balancing mechanisms or more exactly, the use of application delivery controllers (ADC) as devices with the ability of securing scalability and high-availability. This work focuses on f5 products when it comes to application of theory. Furthermore, the mechanisms behind its operation are explained and HTTP (HTTPS) usage case scenario is presented. Scenario descriptions draw attention when configuration differs from the default or recommended process. This thesis is based on working practice in which I participated in solving issues with ADC deployment. The thesis is not dealing with basic BIG-IP LTM appliance configuration like “self IP” setup, license activation, and VLAN setup.

The practice was done in the company of “Tieto Finland Oy.” Thesis topic was formulated as there was a need of knowledge transfer from senior technical staff. In the beginning there seemed to be a testing environment available but later it came to simulating the environment on my own hardware.

2 LOAD BALANCING EVOLUTION

This chapter presents load balancing origin and shows how each technology performed relating to key properties.

In the 1990s when the Internet started to be a more and more interesting place for business, running a single server for busy sites was not sufficient any more. It introduced single point of failure (which businesses wanted to avoid) and also the hardware upgrades to keep the single machine up to the demands could not continue endlessly. Therefore the idea of spreading the workload across multiple machines came up as a solution.

2.1 DNS load balancing

DNS round-robin technique is often seen as the pioneer of the load balancing. It operates with one domain name (DNS A or AAAA record), for instance “www.example.com”, associated with multiple IP addresses. When clients would try to access a mentioned domain name, the DNS server would return the associated IP addresses and each time change their order. This means that every time a new DNS request was made the order of IP addresses was different and the first record which is used by the client was different. The way of order change is dependent on implementation and no standard exists.

Such solution offered and still offers an easy and relatively cheap way of load distributing. In respect to high availability and fault tolerance the improvements are questionable because DNS server does not know if the server behind the address is alive and operational and so it can not answer the client request with leaving the failed machine out of the response. Scalability seems fine as the only thing to change after an addition of a server is the DNS record.

We should not forget that the DNS server is not able to control other DNS servers to whom the change need to propagate first. Clients and some DNS servers may use caching and therefore limit or circumvent the round-robin technique. DNS support for load-balancing is explained in detail in RFC documents, see Brisco (1995).

2.2 Software load balancing

Because the DNS round-robin technique was not sufficient – the need existed for other technology which would fill the requirements of high-availability, scalability and other key factors. Integrating load balancing capabilities into the software became the answer.

Software solutions for load balancing problems are present in multiple implementations. They can be offered as paid proprietary applications or even as free and open-source applications, for example, Linux Virtual Server.

Even though each implementation may differ, the basis is presenting one IP address for a whole cluster of machines. All of the servers have their own IP address, and at the same time they listen for packets designated to “cluster IP address”. When a client initiates communication with a cluster, one of the servers responds first and redirects the client to its own unique IP address.

A key advantage in such an approach lies in the application developer knowledge of application internals. He is then able to fine tune the mechanism or enhance it with new features related to load balancing easily. The solution can be expanded, as when information about the load of particular node is exchanged inside the cluster, and some nodes then take precedence in conveying user requests. Software load balancing brought a certain level of predictability – developers could easily identify when user persistence was needed and repeated load balancing should be avoided.

Regarding high availability, the advantages of a software solution seems to prevail. With inaccessibility of one node, the remaining nodes continue to operate without interruption and failure could be easily identified as the application knew how to verify node status. The drawback here is that the more is added to the complexity of software which bonds (client handling, health checks etc.) nodes together, the more prone to failures the whole system becomes.

At first sight, scalability seems straight forward – prepare the new server and then add it to the cluster. However, as the number of nodes increase, the amount of inter node network traffic grows because each client needs to com-

municate with the other.

To imagine the consequences of such a situation, let us have the example when the medium for nodes and client-to-cluster communication is shared (e.g. one Ethernet port per node). In this case the increased level of traffic can lead to the state where link capacity is saturated and an even higher amount of network traffic is dedicated to inter node communication than to client-to-cluster communication. This finally results in inability to accept more clients as the medium is used at its maximum and consequent failures to deliver packets and to accept new clients arise. We can conclude that there is a risk of overloading the network infrastructure with the sum of mentioned classes of communication when we reach a certain number of servers and connections. We also should not forget that increased amounts of cluster-to-cluster communication have higher demands on node system resources where reaching limits is undesirable.

How severe the impacts of inter node communication are when traffic grows depends on how much CPU and network overhead is introduced by such communication. Please note that separating traffic with 802.1q (VLANs) is not the solution as limits are mostly imposed by media transfer rates. Although the bottleneck can be mitigated by adding another network specifically for inter node communication it also means additional hardware cost with added complexity level.

As Salchow (p. 4, 2007) has pointed out -- the tight connection of application with vendor, imposed problems in maintaining the application and further developing it because all of that was under vendor control. There was no certainty that application providing load balancing from one vendor would work with application from the other one.

The solution which sought to provide vendor-neutral load balanced software suffered from the same scalability issues plus facing additional high availability (HA) issues as they could not provide same level of integration as the vendor specific solutions. (Salchow 2007)

2.3 Hardware-based load balancing

In this chapter I discuss the step in load balancing evolution which preceded today's application delivery controllers (ADC). This step is marked by dedicated hardware appliances performing load balancing. They introduced a point of aggregation with taking the burden of load balancing decisions from the servers themselves. With no need to configure a certain number of nodes separately, one entry point brought ease of configuration and administration.

The principle of functioning is straightforward: the load balancer presents an IP address (sometimes called VIP address – virtual IP address) to the client, and then upon reception performs a destination address translation forwarding packets to appropriate servers. The server processes the request and sends the corresponding packets back to the load balancer which again changes the address, so to the client it seems that he is communicating with load balancer only (Figure 1). The exact method of presenting a single entry point for client can be different but for purposes of this work I will stick to this method.

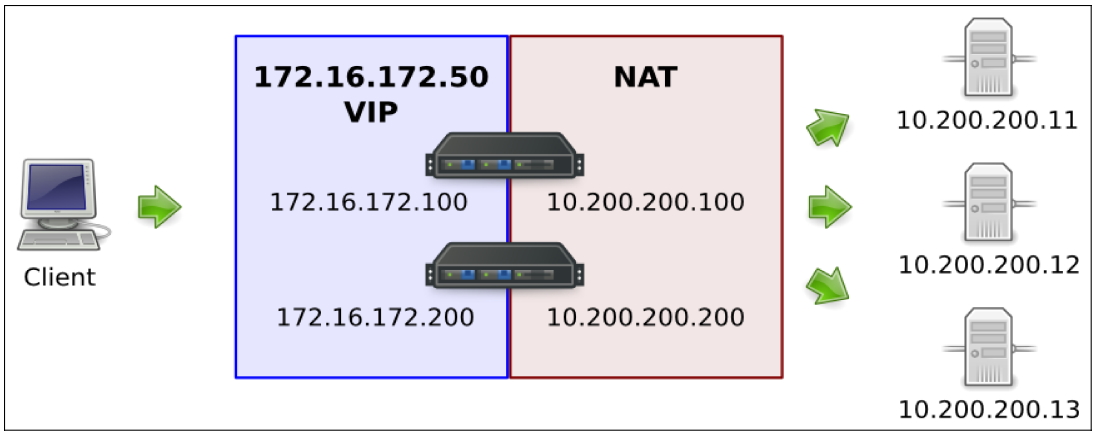

Figure 1: Hardware-based load-balancing principle

The same situation as regarding device configuration, which moved to load balancer was with the health checking and other special services for cluster, now also centralized. Therefore the growth of data related to "cluster care" was linear instead of exponential where every node needed to check with each other. From that we can see the scalability was improved over software based solutions. There is possibility for reduction of expenses and consolidating numbers of servers because no resources are dedicated for load balancing operations

and servers just take care of their own specific tasks – less computational power is needed in total.

Hardware load balancers brought also the positive approach of being vendor neutral, and even if the solutions might not have been on a par with vendor-tailored ones, the ability of load balancing every application whether it supported load balancing itself or not proved to be more viable and added to HA level provided by hardware load balancers.

With the already mentioned attributes of one point of administration, I ought to mention the aggregation of statistical data from various sources. Be it the network usage, number of requests in a given amount of time, log of potentially dangerous traffic, or status of all servers which are supplied with client connections and many more particularly important data, all of this was now readily available for administrators for checking, troubleshooting and maintaining, making all aforementioned task much simpler.

2.4 Application Delivery Controllers

Application delivery controllers (or appliances) can be viewed as descendants of hardware load balancers. We can think of them as devices which have all previously mentioned capabilities of HLB, and add more capabilities. They evolved as more and more functionality was integrated into the load balancers and their intelligence grew while capabilities for content caching, advanced packet filtering, rate shaping and other solutions commonly present in previously dedicated devices (e.g. firewalls), brought them to another level in performance concerning application delivery.

ADC offers fine-tuned traffic handling with respect to various application needs while maintaining relative ease of use.

3 DRIVING FACTORS

So far the historical aspect was explained and reasons for ADC deployment lightly sketched. Needs which are to be satisfied by ADCs are described in more detail in this chapter.

3.1 High-availability

Generally, available service can be accessed and used by the user. Availability in terms of services can be understood by mathematical expression as a percentage of time when the system can be used in a given time period.

With term high-availability (HA) I refer to a certain level (percentage) of availability which is expected by users – the system is available during certain operating hours. This means that no unplanned outages are introduced to the system and all planned outages should be announced in advance (Piedad and Hawkins, 2000). It is worth mentioning that availability can be perceived in a different way when it comes to different observers.

If a system was in fully operational state, but once its network connection to customer experienced outage – the administrator would say that “availability” was 100 percent, but the customer would disagree that only uptime was 100 percent whereas availability was just 98 %. What administrator was talking about was in fact system uptime. System was up and operational but with no connection between client and system – it was not available.

The reason why customers seek high-availability is easy to understand. They want uninterrupted work process because otherwise decreased level of productivity, loss of profit, and delays would occur. In the case that interruption of service is needed they want to know about it before it happens so the processes can be adjusted accordingly.

3.2 Fault tolerance

Fault tolerance is thought of as the property of a system which enables the system to continue to function in the case of one or more failures in its components. In the case of failure, the degradation of its quality is equal to severity of

failure. For example, failure which is not critical for the whole system will not cause the system to stop operating but merely would make some operations impossible.

Think of the mail server which could not receive a message for some user because user quota for mail is exhausted. The server would inform the sender (maybe also affected user) and continue with processing other requests instead of entirely stopping its operation.

In our case, customers seek a solution which is fault tolerant because they need high-availability. We can think of fault tolerance as a an item enhancing availability.

3.3 Predictability

Predictability is a term which is not well defined for specific scientific fields. In general, it is the ability to predict. What we are trying to predict – those are various properties of the system which depend on known and unknown variables (Grund, Reineke and Wilhelm, 2011). Therefore if we consider some system more predictable, we could predict values of watched properties with higher precision.

Think of connection oriented networks (ATM) versus connectionless networks (IP). In connection oriented networks data flow along a certain path which was created before the transfer began. We can easily predict where the packets will flow and calculate delay during the whole data transfer. On the other hand, IP protocol does not create any path but sends the packet which is travelling the net on its own and could take alternate routes regarding other packets in the same session. Calculated delay is calculated as ave rage and only after the transfer was done.

In the field of load balancing we can understand predictability in that it says with what certainty we can predict the decision of the system in server selection, health checking, or other key properties of operation. The more clearly input variables are defined, the more predictable the values of properties are. Predictability can be easily exploited for further system adjustment and risk assess-

ment.

3.4 Scalability

Scalability can be understood as ability to satisfy growing load demand gracefully with non-excessive resource usage growth and as the ability to accommodate to continuous growth with addition of resources (Bondi, 2000).

If we can satisfy 1000 users with one device and the service does not suffer when load grows to its maximum – service has similar qualities when serving 200 or 900 users – then the load was processed gracefully.

If for more users we would purchase for example another server for a total of 2000 user capacity, then our solution is able to accommodate to the demands with addition of resources. The opposite situation would happen if addition of more resources would not be possible because the technology in use does not profit from additional resources and completely new technology would be needed.

4 BIG-IP LTM PLATFORM

In this chapter I will look into implementation of ADC. I will describe the BIG-IP LTM solution from f5 company. f5 is considered as a leader in ADC market. (Gartner, 2009, 2010 cited in McGillicuddy, 2010) Their application delivery portfolio is represented by BIG-IP product line which is the place where LTM (Local Traffic Manager) lies.

4.1 Hardware

BIG-IP solutions are delivered as standalone ADC devices from 1U to 2U size or high performance blade-based solution. Devices differentiate with hardware offload possibilities, port count, computational power, memory size and traffic throughput.

All devices have management Ethernet port which is reserved for device configuration and is not to be used for routed traffic. There are also two RS-232 DE-9 ports: one is a console port while the other one is used for fail-over detection.

4.2 Software architecture

f5's product line is based on a special operating system called *Traffic Management Operating System* (TMOS) which is present in their devices along with the Red Hat Linux system. According to Salchow (2011) the TMOS is a real-time modular OS taking care of traffic related operations while running independently of Linux OS.

Notable features of TMOS include simultaneous multiple network stack usage where stacks appropriate for a certain type of application is used, hardware–software processing interchangeability where TMOS modularity enables some operations to be performed on dedicated hardware (e.g. SSL offloaded to ASIC) instead of software processing and iRules – scripts written in Tcl (Tool Command Language), which can be used to react on TMOS events and to alter connections and packets thus extending the possible usage scenarios. (Salchow 2011).

4.3 Configuration interfaces

BIG-IP devices can be accessed both through text-based interface and graphical interface. Text-based configuration is done with *bigpipe* or *tmsh (Traffic Management shell)* command tools which are run from the Linux system shell. Tmsh a is more recent tool which has wider possibilities than bigpipe's shell. Both of them can be seen as similar to Cisco's IOS shell.

The GUI is represented by a browser-based application called “Configuration utility”.

5 BIG IP VE PLATFORM

To facilitate deployment and pre-deployment operations, e. g. simulations in a lab environment, BIG IP is available in the form of a virtual appliance.

Hardware requirements for virtual appliance are quite moderate with 1 GB of RAM, one CPU and 10 GB of disk space. Disk space requirements can be lowered by not using preallocated virtual disks.

5.1 Distribution

Virtual appliance images can be downloaded from www.f5.com/trial/big-ip-ltm-virtual-edition.php After free registration, the user is able to obtain up to 4 trial licenses and also virtual appliance files. There are two more variants to choose from – the VMware ESX and the VMware Workstation version. I have chosen the Workstation variant, and for virtualization I have used the VMware Player. ESX variant would have probably been more viable for a full scale lab environment as it offers more flexibility, but I decided not to use it as my test machine was not a dedicated hardware but my own computer.

5.2 Limitations of LTM VE

The virtual edition of LTM does not support Spanning Tree Protocol (STP) nor its successor, Rapid STP. Offloading SSL to hardware is not possible. There is no information regarding virtualization of dedicated SSL processing hardware and thus making it available and functional inside an LTM VE virtual appliance.

5.3 Limitations of trial version

The trial version of LTM VE is primarily limited by its 90 day license. There are also other limitations:

- Maximum transfer rate is limited to 1Mbit and only 150 concurrent SSL connections are allowed
- Only a single CPU is supported
- Importing UCS configuration from other non virtual BIG-IP LTM may not work properly

- Applying hot-fixes and version upgrade is not possible – on the contrary, I was able to successfully do so in the case of hot-fix. It is possible that this was possible only due to the manufacturer's omission in enforcing trial restrictions.

The limitation of transfer rate is quite severe, and for any enterprise level test lab I would recommend purchase of the LTM VE license. Otherwise, any tests measuring throughput, response and other variables can hardly be seen as valid and helpful for deployment at the customer site.

5.4 Installation

The installation process is described chronologically with notes about steps which may not seem clear.

The installation file is in the form of an OVA package which contains checksum files, VMware virtual disk file, and OVF (Open Virtualization Format) file, which is used for description of the virtual appliance. For use in virtualization software I had to convert the file into the appropriate format with *ovftool* CLI tool. Ovftool is able to convert various file formats used in different virtualization solutions. It is well documented in OVF Tool Documentation. (Vmware, 2010).

When I had all necessary files ready, the appliance was imported into the VMware Player. At that point, the exact environment configuration is up to the person who creates the lab, as he needs to specify networks for specific scenarios. It is important to keep in mind that there are only two routed interfaces inside the virtual appliance, but with VLAN tagging this should not impose severe drawbacks.

5.5 Testing environment

All virtual appliances needed for this work were run in Vmware Player software version 3.1.3 on Gentoo Linux as the host operating system. In server roles there were two guest machines – one with Gentoo Linux with the Apache 2.2.17 web server, and the other with Microsoft Windows 2008 R2 SP1 with IIS 7.5.

BIG-IP LTM VE appliance version 10.1.0 Build 3341.1084 with applied hot-fix

version 3402.0 were used in redundant set-up with configuration synchronization enabled.

On the virtual servers there were four IP addresses configured so that each of them were assigned to one separate site simulating more real servers. This simplification was necessary as there was of lack of resources for running multiple instances of servers simultaneously.

Client request were performed from the host machine through virtual network 172.16.172.0/24 simulating client access. For graphical overview of the environment, see Figure 2.

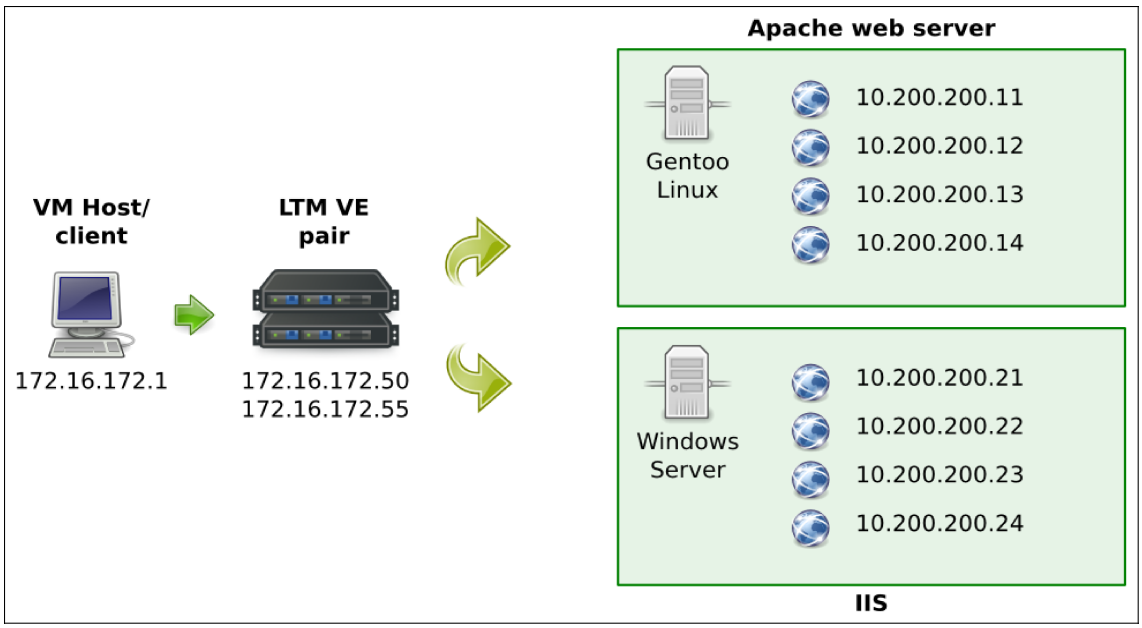

Figure 2: Test environment

I experienced issues with Microsoft Windows Server virtual appliance and VMware virtualization software as there was need for NIC to behave as a trunk port. Although support is present in the virtual NIC, the manufacturer of virtualized card does not supply drivers and software with this ability for Server 2008 R2. The problem was circumvented by extracting drivers for the Vista platform, replacing the pre-installed drivers, and installing the desired software extension which had 802.1q support.

6 USAGE SCENARIO

The scenario in this chapter represent possible deployment of f5's BIG-IP LTM for HTTP and HTTPS load balancing. It is assumed that the LTM has license applied and the interfaces are set up so the network reachability is possible. Also, the set-up of floating address which can be assigned to a virtual server is expected. Concerning web servers, they should be operational and set up for use.

The scenario is based on a site which uses two types of web application where one is run on an Apache web server and the other on an IIS platform. The desired state is that HTTP and HTTPS requests are processed and connection is persistent once load-balanced.

The site has two IP addresses (“floating addresses”) which are used to serve client requests. Every floating address is mapped to a different pool as is described in Figure 2. DNS operation is not considered here as it is out of the scope of the scenario.

In the scenarios, I conveniently used an LTM feature called “profile” which enables the administrator to use and reuse a predefined set of values for a given protocol set-up, persistence set-up and others. Therefore the profiles are a centralized method of management of various variables. There are several profiles already available for typical use cases, and these can be altered and saved as new profiles if the need for custom settings is present.

Nowadays HTTP application will most likely rely on some sort of persistence so it is crucial to set it up and keep the client request coming to the same server after the initial load balancing decision is made. Cookie persistent methods are explained in a separate chapter.

All configuration steps are performed in a web-based configuration utility (CU). The set-up of the the LTM appliances which fulfil scenario goal is in Appendix 1. Appendix 1 is formulated so that in the end, the reader is able to configure HTTP and HTTPS load balancing with cookie persistence.

6.1 Cookie persistence methods

Cookie persistence methods use well known HTTP cookies. The cookies are stored on the client computer and used for various HTTP application needs. When the client communicates with the server it includes cookies in its requests. Cookie persistence uses information inside those cookies to be able to track client sessions and send requests in one session to the same server.

Even for cookie persistence there are several methods which differ in the way the cookies are created, modified and processed.

Insert method

This method intercepts server response and adds a cookie with the name BIGIPServer. The cookie contains information about the chosen server and timeout is set according to BIG-IP values. After the interception, the response is sent to the client.

When the client send another request the request contains the BIGIPServer cookie which is read by BIG-IP, and then the client request is forwarded to the same server.

The cookie insert method advantage is that the server configuration remains the same but the fact that the processing take place inside BIG-IP could be limiting.

Rewrite method

In the rewrite method the process starts on the server side. After receipt of a load-balanced client request, the server inserts the Set-Cookie header with the name BIGipCookie containing 120 zeros. BIG-IP intercepts the server response and renames the header to BIGipCookie<pool_name> and also includes information about the server and port.

When the client sends another request it sends also the cookie which is intercepted by BIG-IP and sent to the appropriate server. The server response contains blank cookie as in the beginning, and the process repeats itself.

The workload of the BIG-IP is reduced and servers can have the same configuration, yet they need to be configured to generate the cookie beforehand.

Passive method

With this method the BIG-IP does not perform any modification to headers. Instead, servers are expected to be configured to include a cookie which identifies them. The BIG-IP only reads the HTTP headers and look for the one needed for decision.

Although the resource use is reduced as the BIG-IP does not perform any packet changes, there is still the problem that every server needs special configuration.

Hash method

As the name suggests, this method uses hash calculation. The client sends a request to the server which includes a site specific cookie. Then the result is sent back to the client. In the second request the cookie is already present and the BIG-IP finds it and calculates the hash value based on the cookie. The result of the calculation determines which node is to be selected. Persistence is not kept for the previous connection but for the subsequent ones.

6.2 HTTPS load-balancing

There are basically two way to process SSL with LTM. The difference is in the place of encrypted termination. Basically there are two methods:

Client-side SSL

The client establishes the SSL connection with LTM which offers a server certificate. For the client it still looks, like it communicates with the server but in fact the data are decrypted in LTM and then transferred unencrypted to the server. Server responses are again encrypted and then sent to the client. This methods takes all SSL tasks away from servers with the drawback of unencrypted communication. The administrator should be aware of that and take measures to avoid eavesdropping inside the server network. If risks are estimated to be too high, then the server-side method should be used.

Server-side SSL

The LTM posseses both client and server certificates. It decrypts the data for

processing and then encrypts them. Communication is encrypted from client to LTM and from LTM to server. This method has an interesting possibility when facing systems which are not able to use up-to-date encryption or certificates with a certain length, e. g. 2048 bits would consume too much resources. It is then possible to use shorter keys in the local network (128 bits or similar) and use the longer keys in client to LTM communication.

In the first place, the certificates should be obtained or at least self-signed certificates created. Certificates would probably be available in real deployment. LTM can generate certificate signing requests (CSR) which can be used to obtain certificates at a trusted CA.

In the Appendix 1, Client-side SSL configuration is used.

7 SUMMARY

Load balancing is a viable technology which, nowadays integrated with many more supportive technologies in dedicated devices, offers a comprehensive solution for application availability, scalability and fault tolerance. At the same time, the information in this field seems scattered and is not so easily available compared to long discussed networking problems like IP routing.

Because I focused on vendor specific implementation which is used in the Tieto Corporation in my work I was able to avoid over generalization. In that way the work is not so distant from practice as it would be the case if it was dealing with several vendors' equipment and general conclusions.

For purpose of the thesis I created an environment which reflected a usage scenario of HTTP load balancing in virtualization software. The set-up of web servers and the ADC appliances was performed in order to verify the process of configuration and also to verify that the configuration worked according to expectations. The possibility of having ADC virtualized was very convenient because there was no hardware available for testing purposes at the company site.

With the guide for HTTP and HTTPS load balancing set-up, I expect every network engineer with basic knowledge of HTTP and IP operation to be able to recreate the testing environment and also to deploy LTM devices at customer sites. As for LTM configuration skills, only a basic level of understanding is required.

While working in the fore mentioned environment, I experienced some issues which are related to LTM VE trial licence restrictions on throughput which effectively made testing performance more or less meaningless. Otherwise, working with the virtual version did not show any inconveniences regarding what an administrator would expect from real hardware. I can recommend purchase of LTM VE licence at least for the company test lab, or for further possibility for education of network engineering staff as its ease of use is very convenient. Also, using Vmware ESXi solution instead of the more basic Player software would make the environment more customizable and easier to implement.

Concerning support during my work, I have to admit that there are certainly things which might be improved as I felt it sometimes very difficult to approach contact persons from the Tieto Corporation for advice or support. From my point of view, there should have been more time dedicated to description of things to be done in the beginning as well as a clear definition how things are to be executed from the Tieto Corporation point of view. Time and human resources were sometimes wasted just because some problems were dealt on the fly. The thing I would like to emphasise most is an early start of work for future students, and that applies both to the business partner and to the students. [1][2][3][4][9][5][6][10][11][8][7]

8 REFERENCES

Bellaiche, F., 2011. Network Devices Kit.
http://www.quantum-bits.org/?p=48 (Accessed 20 April 2011)

Bondi, A. B, 2000, Characteristics of Scalability and Their Impact on Performance. www.win.tue.nl/~johanl/educ/2II45/Lit/Scalability-bondi%202000.pdf (Accessed 29 March 2011)

Bourke, T., 2001. Server Load Balancing. Sebastopol: O'Reilly & Associates, Inc.

Brisco, T., 1995, DNS Support for Load Balancing. http://tools.ietf.org/rfc/rfc1794.txt (Accessed 10 April 2011)

F5 Networks, Inc., 2011, BIG-IP Local Traffic Manager Virtual Edition Trial. www.f5.com/trial/big-ip-ltm-virtual-edition.php (Accessed 29 March 2011)

F5 Networks, Inc., 2011, Release Note: BIG-IP Virtual Edition Trial version 10.1.0. http://support.f5.com/kb/en-us/products/big-ip_ltm/releasenotes/product/relnotes_ve_10_1_0.html (Accessed 29 March 2011)

Grund, D., Reineke, J., Wilhelm R., 2011, A Template for Predictability Definitions with Supporting Evidence. http://drops.dagstuhl.de/opus/volltexte/2011/3078 (Accessed 10 April 2011)

Kristol, D., 2000, HTTP State Management Mechanism.
http://tools.ietf.org/rfc/rfc2965.txt (Accessed 20 April 2011)

McGillicuddy, S., 2010, Magic Quadrant for application delivery controllers: Radware ascends, newbies arrive.
http://itknowledgeexchange.techtarget.com/networkhub/magic-quadrant-for-application-delivery-controllers-radware-ascends-newbies-arrive/ (Accessed 10 April 2011)

Salchow, Ken, Jr., 2007, Load Balancing 101: The Evolution to Application Delivery Controllers. www.f5.com/pdf/white-papers/evolution-adc-wp.pdf (Accessed 2 March 2011)

Salchow, Ken, Jr., 2011, TMOS: Redefining the Solution.
http://www.f5.com/pdf/white-papers/tmos-wp.pdf (Accessed 22 March 2011)

Tango Desktop Project, 2011. Tango Icon Library
http://tango.freedesktop.org/Tango_Icon_Library (Accessed 20 April 2011)

VMware, Inc., 2010, OVF Tool User Guide. www.vmware.com/support/developer/ovf (Accessed 29 March 2011)

HTTP and HTTPS set-up

- Firstly add nodes into the LTM database – in CU navigate to *Local traffic* > *Nodes* > *Create…*
  - Enter IP address and node name. Node name is not hostname but merely identifier. For example “LINUX_1“
- After that create pool of servers – in CU navigate to L*ocal traffic* > *Pools* > *Create…*
  - Here the name of the pool should be entered so as it is easily distinguishable and understandable. For this set-up it might be “HTTP_POOL”.
  - Select health monitors which are to be used for the pool. Select “http”.
  - Choose the appropriate load balancing method and add members from the pool. It may be a good decision to keep the “Round robin” method until the persistence is configured and verified. In that case the administrator will not assume that persistence was used on your clients' requests when in fact only the load-balancing decision was performed.
- Finally the virtual server is to be created – in CU navigate to *Local traffic* > *Virtual servers* > *Create…*
  - Enter the name (as explained above) and IP address, which is a floating address
  - For service port use the port on which the HTTP server is listening, most probably 80.
  - Choose “http” in HTTP Profile list
  - In Default pool select previously created “HTTP_POOL”

Extending HTTP monitor

With “http” monitor shipped with LTM we get a monitor which accepts any re-

sponse from the web server. Even the HTTP error pages are sufficient. As this is most probably not an option for real deployment we should create our own HTTP monitor. It will be looking for a page “server_ok.html” and inside it for the string “Server is OK” so it is expected that such page is in place. Regular expressions are supported for more sophisticated testing.

This is just an example solution, it is up to the administrator and application developer or maintainer to decide how the functioning of the application should be tested.

- In CU navigate to *Local traffic* > *Monitors* > *Create…*
    - Type of monitor is HTTP, we also select to import setting from shipped “http” filter
    - In Configuration we should insert HTTP command to retrieve the previously mentioned test page: “GET /server_ok.html” That means this page is available in web site root for example at address www.example.com/server_ok.html
    - In Receive string field you enter the pattern which should be present in the returned document for the member to be considered operational. Here the “Server is OK” string is inserted.
    - Click on “Update” and apply new settings

In this moment the HTTP virtual server is ready for use but still there is no persistence in client requests which are always load balanced and sent to different nodes.

There are several persistence methods which have their pros and cons. Regarding HTTP traffic and the fact that often there are clients behind NAT, with one shared IP address, the cookie persistence methods are preferable to source address persistence.

Enabling cookie persistence

- For enabling cookie persistence – in CU navigate to *Local traffic* > *Pro-*

files > *Persistence* > *Create…*

- Persistence type is “Cookie” and parent profile would be generic “cookie” profile.
- In this example we enable the rewrite method so check “Custom” check-box and then uncheck all other check-boxes but “Cookie Method”. Select the appropriate cookie method and press “Update” to store new persistence profile permanently. For IIS you would use the insert method.

- Now the profile is created but not associated with a virtual server – in CU navigate to *Local traffic* > *Virtual servers*
  - click on the name of the HTTP virtual server you created previously
  - click on the “Resources” button at the top of the page
  - For “Default persistence profile” choose the name of new profile you created
  - Click on “Update” and apply new settings

Verifying cookie persistence

The process of persistence verification is quite straightforward. The administrator would access the web site using a web client capable of cookie handling, ideally browser. After the site has been accessed several times you can view cookies associated with the site.

To view used cookies in Internet Explorer version 8 on Windows XP SP3 – navigate to “*C:\Documents and Settings\<user login>\Local Settings\Temporary Internet Files”* where cookies are stored as text files with the name in the form “cookie:<user login>@domain”. You can easily find the cookie used by LTM here. For other browsers like Opera, or Mozilla Firefox which have more sophisticated tools for managing cookies, consult their manuals.

To verify on the server side you would simply examine the access log of web application servers in your HTTP pool. You can force your web browser to re-

move specific cookies so the LTM will handle your request as a new one.

It should not be forgotten to change the pool load-balancing method to the preferred one if “Round robin” was chosen for facilitation of cookie persistence verification.

HTTPS load-balancing

For the purpose of the testing environment the self-signed certificate is used. LTM can generate such certificate and it is also able to generate CSR file for real world deployment.

It is assumed that there is another virtual server configured which will be used for HTTPS. In this moment its configuration can be same as for HTTP server. Later you could change its SSL profile. As we are using client-side SSL (only HTTP traffic goes to server) the underlying pool and virtual IP address can be shared. It is also possible to do the whole configuration of a new virtual server after SSL set-up.

SSL related files are stored in “/config/ssl” directory in LTM file system. Directory contains subdirectories for certificates, CSR, CRL and key files.

- In first step, create the certificate – in CU navigate to *Local traffic* > *SSL Certificates* > *Create…*
    - Enter certificate name (for LTM identification only).
    - In Certificate properties choose “Self” as an issuer. In a real world situation you would choose “Certificate Authority”. When you have confirmed all certificate detail, LTM would allow you to download the CSR file so you can obtain the certificate from trusted CA and then import it in LTM.
    - In Common name field you enter the web domain or name of the server if this certificate is used in an internal network only. You can use “www.example.com” value.
    - Other values are included in the certificate so if you want to add more

information, you can do it here. At least the “Organization” field should be filled in.

- Choose Key size according to your preferences.
- Click on “Finished” which generates a new certificate and stores it inside LTM.

- Now a SSL profile would be created so it can be associated with the virtual server – *in CU navigate to Local traffic > Profiles > SSL > Client > Create…*
  - Enter name of the profile and for Parent profile choose “clientssl”.
  - Check “Custom” check-box and uncheck all subsequent check-boxes so that only Certificate and Key items are enabled. Select appropriate certificate and matching key.
  - Click on “Finished” to save the profile.
- Associate the SSL profile with virtual server – *in CU navigate to Local traffic > Virtual servers > Virtual server list*
  - Select desired server and choose previously created profile in SSL Profile (Client)
  - Click on “Update” to apply the settings.

You can go through same persistence verification as in the HTTP set-up. Now you have HTTPS virtual server with client-side SSL processing enabled and operational.