Application Front End: The New Datacenter Architecture

Robin Layland

As a successor to server load balancers, AFEs improve application performance and security

Datacenter consolidation and Web-centric computing's goals of reducing the growth and cost of servers put new demands on networks. Network managers can no longer view consolidation as a "server" and "application" group problem. The success of consolidation projects increasingly depends on the network.

Additionally, end users are always seeking faster response time and demanding the network solve the problem. It is unacceptable for the network manager to deflect blame by pinning the problem on the application rather than the network. The fact is, the network can help solve the problem.

Finally, add to this mix the increasing creativity of hackers and attackers who put more pressure on the security of the datacenter. The servers can't provide all the protection and security that's needed; network managers must step in and help.

It is not enough to provide just the latest and fastest switch. Promising a 10-Gbps solution, while important, is not going solve the problem. A new network architecture is needed in the datacenter: A solution that meets the challenges of consolidation, fast response time and security while providing high availability, high throughput, flexibility, scalability and manageability.

Luckily a new architecture and class of products that provide it have recently emerged. The new solution is an Application Front End (AFE).

What Is An Application Front End?

First the good news. An AFE is an evolution of an old standby in the datacenter—the server load balancer (SLB). SLBs emerged in the late 1990s to solve several problems: For performance or availability reasons, applications no longer could reside on just one server; large enterprise applications and the new Web applications required multiple servers. This created a problem of balancing demand over various servers. SLBs did this by examining the application headers (Layer 7 information) and deciding which of the servers should get the request, keeping the load balanced between the different servers. SLBs also provided increased availability by intelligently monitoring servers to quickly detect which ones were having problems or not responding. It then routed traffic away from the problem servers and notified operations staff.

An AFE incorporates these functions and then takes SLBs to the next evolutionary step. Figure 1 shows the AFE's architecture. The AFE takes the server load balancer base and adds three important new functions:

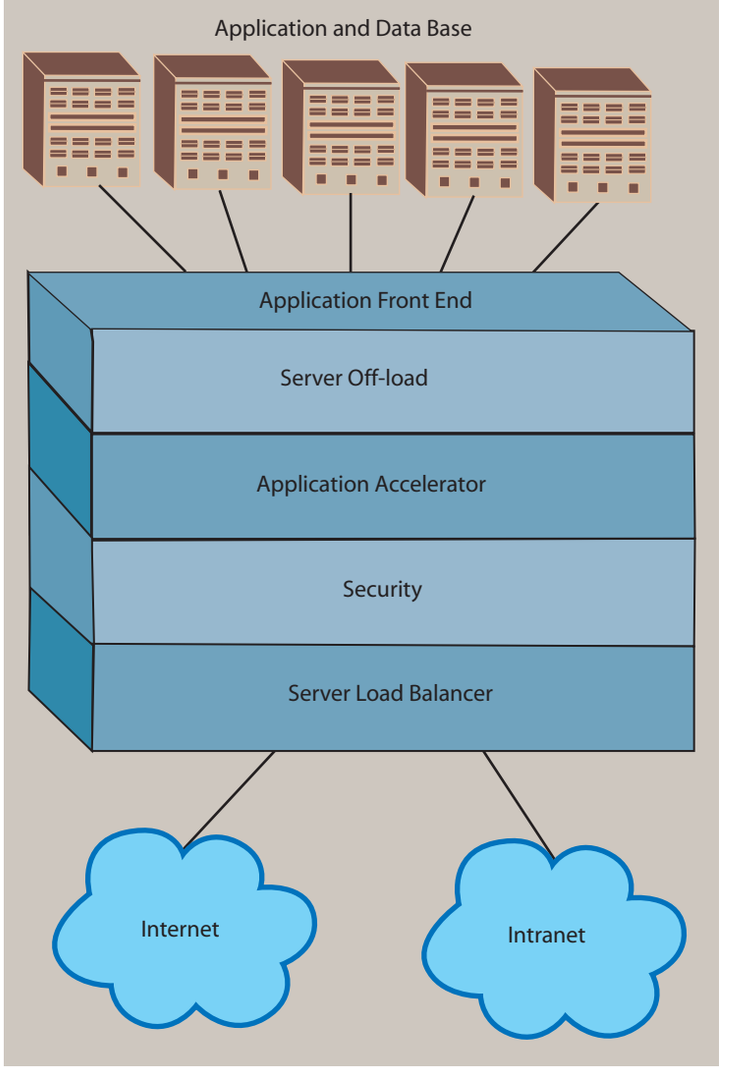
- Server offload, which reduces the number of servers needed, saving the business money.
- Application acceleration, which improves response time without having to change the underlying application.
- Security, which provides another layer of defense.

AFEs are available from several vendors including Redline Networks, Array Networks, NetScaler, Fineground Networks and from some of the existing SLB vendors including F5 Networks, Foundry and Radware. The reason all these vendors are interested in AFE is that the AFE market is growing. The SLB market has been going down recently, but with the switch to AFE, the market is estimated by Acuitive to grow to over $1.2 billion a year by 2007.

The key to adding these new features is a change from how the SLBs handle traffic. An SLB examined each packet's header when a connection was established and determined which

Robin Layland is president of Layland Consulting, a firm that specializes in network architecture and new technology. He has more than 25 years' experience in enterprise networking, including technical and management positions at American Express and Travelers.

FIGURE 1 Application Front-End Architecture



server to route the connection to. This worked very well with Web pages since they are made up of many separate TCP connections—one for each object. For example, connections for static and common objects could go to one server while dynamic objects requiring a database lookup could be sent to another server. There was no need for the SLB to interfere with the connection—just intelligently act upon it.

If an AFE is going to provide advanced security, offload the server and accelerate the application, it is going to have to get more involved with the TCP connection. Server offload means that not every request is actually going to make it to the server. And application acceleration includes acting for the server, returning the result without having to wait for the server to respond, thus cutting response time.

These advancements mean the AFE must terminate the TCP connection and act as a proxy for the server. The AFE implements a full TCP proxy function. This means that when a client or Web browser send in a connection request, the AFE intercepts the request and responds to the client. The client thinks it has an end-to-end connection with the server but really the connection is from the client to the AFE. The AFE then has a separate connection from itself to the server. This is one of the key differences between an AFE and an older server load balancer—the SLB only examined packets and did not interfere with the connection.

The AFE uses TCP multiplexing to reduce overhead

Server Offload

The proxy function is the key that allows the AFE to lessen the burden on the server. With an SLB, each connection request makes it to the server. The server then has to process the TCP overhead. While the overhead is small it can add up, especially with Web applications. This TCP overhead can easily consume 10–50 percent of server capacity.

The reason for this is the way the Web and new applications act. In the past, a client would start a connection with an older client/server application and then use the same connection for a long period of time. For example, someone would logon to a server and use that connection all morning until they logoff for lunch. Only when they went to a new application would they need another TCP connection. The TCP overhead was a very small percentage of the processing required to support the users.

With the Web and newer applications, this changed. Each item on a Web page uses its own connection to get the item. A Web page with 10 objects means that the server must process a TCP connection for each object, with each connection lasting only long enough to deliver the object. The next page meant another 10 objects with another 10 connections to process. The result is that the server is processing a lot of short-lived TCP connections. All these connections can mean that a server is spending a significant amount of resources just to bring up and take down TCP connections.

The AFE offloads this burden by handling the TCP overhead. The AFE instead establishes a set of long-lived connections to the application. When

Just increasing link speed isn't enough to accelerate application performance

a connection request comes in from a client, the AFE matches it up with a connection it already has, reusing the connection. The AFE takes the hit on bringing the connections up and down from the client. The server sees only long-lived connections to the AFE, returning it to the days when TCP connection process used few server cycles. This technique is called *TCP multiplexing* or *TCP reuse*.

The next way the AFE offloads the server is by caching server responses. For example, many Web pages feature items such as a banner with a logo. When a client request comes in for that object, traditionally the request would be forwarded to the server. The server would then spend resources to call up the banner and send it to the client. After the first time, the AFE can eliminate the request going to the server. The AFE is intelligently watching what is going on and stores, or caches, the objects. When the next request comes in for the banner, the AFE calls the object up from its memory and responds to the client. The server never sees the request and thus doesn't have to spend any time processing the request. This saves server resources plus shortens the time it takes to get the object to the client. This is called Reverse Caching (Figure 2).

One issue for the AFE is how it stores the object. AFEs can store the object in memory or on a disk. Memory is faster but a disk allows for a greater number of objects to be stored.

There are two types of objects—static and dynamic. Static content is like the banner in the above example; it has content that doesn't change. It is easy for the AFE to cache this type of item. When the AFE sees it for the first time, it stores the object and starts a timer. The timer tells the AFE how long to keep the object.

Caching dynamic objects is more complicated, but many AFEs have also addressed this. One of the reasons AFEs can cache dynamic objects is because not all dynamic content is truly dynamic. Some dynamic objects change every time; for these objects, the AFE can't provide any caching and must ask the server to send the object every time. But some dynamic objects only change occasionally. The object may change every hour or once a day, but because it does change, it was marked as dynamic. The AFE can cache these objects as long as it doesn't keep them longer than their static period.

For example, for an item that changes every hour, the AFE can reduce the load on the server by caching the object the first time it sees it and then refreshing it every half-hour. Of course, caching dynamic objects is more complicated and generally requires that the person setting up the AFE work with the application people to determine which dynamic objects are only semi-dynamic—but this effort can save a significant amount of server resources.

The AFE can also cache objects for which it's acceptable to be a little bit out of synch. While the object can constantly change, the organization may decide it is acceptable to show an older version, as a tradeoff to reducing the cost of running the servers. Again, the network manager needs to coordinate with the application group on this.

Finally, some objects are erroneously marked as dynamic—the programmer didn't know or care and just marked them incorrectly.

Another way the AFE offloads the server is by processing the SSL overhead. Since the connection is terminated, the AFE handles the authentication and encryption functions. Relieving the server of the burden of encryption can save a significant amount of server resources. While there are server NICs that can perform this function, it is best to have it done in a central place, since the AFE can apply this benefit to every server in the datacenter.

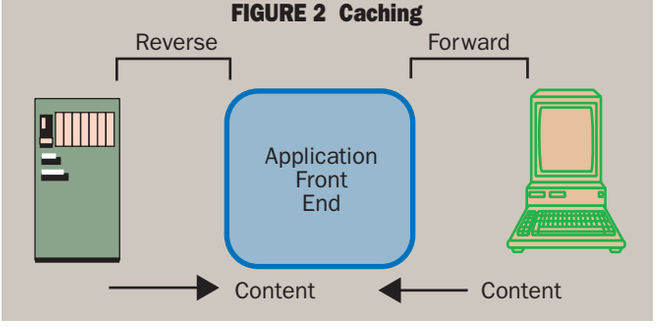
Performing the SSL function in the AFE does mean that clear text is being sent between the AFE and the server. This should not be a problem, since the hop between the AFE in the datacenter and the server should be secure. If this is a problem, vendors such as Array Networks do allow the traffic to be re-encrypted. This lessens the benefit of SSL offload but is still more efficient, since the connection is long lasting and thus avoids the key exchange for each end-user connection.

Overall, the AFE can offload a significant amount from the server. Is this important? Yes and no. If your servers are all running at a low utilization, then the offload function will have little effect. But if the servers, or some of them, are running at a high utilization, then the AFE can significantly push off the date when the servers need to be upgraded. This is especially the case for server consolidation or on-demand computing when the goal is get the most out of a server.

Application Acceleration

Users and customers are always complaining about response time: The page is never fast enough. The question CIOs face is how to accelerate delivery of the application? How to reduce response time? Delivering the page faster not only

FIGURE 2 Caching



The AFE adds another layer of security

makes for a happier user, it also can increase productivity (See this issue, pp. 8–10).

The traditional network way to reduce response time is increase the speed of the link. Someone complains and the traditional networking answer is to give then a faster connection. The problem is that this is expensive and does not always work.

Faster link speed helps the most when the link is highly utilized. A faster link, assuming the same volume of traffic, can reduce the queue time, or wait time, for a packet. For example, a link running at 75 percent utilization would see an average queue length of 3—i.e., 3 packets are waiting in line. Doubling the link speed, halving the utilization, would result in an average queue length of 6, meaning that many messages would never have to wait, and those that do would wait a significantly shorter time.

But what if the link utilization is only 30 percent or less? Would increasing the speed make much difference?

The truth is that it doesn't—most packets don't have to wait. But wouldn't the increased speed mean the packets get there faster? Yes, each individual packet arrives at its destination quicker, but it may not make much overall difference because of the way TCP works.

TCP only sends a set number of packets before requiring an acknowledgement back. The number of permitted outstanding packets increases over the life of the connection. Since many of the connections are short lived, the TCP protocol never reaches a large window size (number of packets allowed to be sent at one time). This can easily reduce the effect of the faster transmission. This is especially the case when the number of users on a link is small, and increasing the link speed may mean that the extra bandwidth is unused and doesn't significantly reduce response time. This is not to say that increasing the link speed is unhelpful; it is, and can solve some problems, just not all response-time problems.

The AFE can significantly reduce response time by attacking the problem from a different angle. The AFE reduces the number of packets sent. Sending, for example, five packets instead of 10 means it takes half the transmission time. Fewer packets means that there are fewer TCP acknowledgements, removing the wait time for the acknowledgements. Fewer packets means the object gets there faster.

So how does the AFE reduce the number of packets? The principal techniques are:

- Compression
- Caching
- Fast redirect

Compression helps by reducing the amount of data that has to be sent. Objects coming out of servers are generally uncompressed. The AFE uses standard compression techniques, such as GZIP for browser applications, to compress the data This generally results in 30–50 percent less being sent.

When the object reaches the client's browser, the browser uncompresses the object. Less being sent means fewer packets and faster response time. Compression in the AFE saves the server from having to waste cycles doing the compression. Compression also has the extra benefit of reducing link utilization, putting off the date of the next upgrade. Compression abilities can vary between vendors, including the techniques supported and what they compress.

Caching can significantly reduce response time. Caching applied between the AFE and the client is called Forward Caching. Caching reduces response time by having the AFE send the object directly to the client instead of getting the object from the server, as described above.

A potentially more important reduction in response time results from monitoring changes from page to page. Many objects are the same from page to page. When the first page arrives, the client stores the object in its own cache but what generally happens is, when the second page is sent with the same object, the object is once again sent to the client. An AFE is intelligent enough to realize this and avoids sending the object the second time. The AFE instead sends an indicator to the client telling it to use the object in its cache. In many cases this can significantly reduce the time it takes to display a page. This technique can be applied to both static and dynamic objects.

Another way the AFE can reduce response time is by fast redirect. Many times when someone enters a Web address, the client is immediately re-directed by the server. The AFE can speed this process along by automatically performing the re-direct.

One difference between different vendors' AFEs is what types of applications the application acceleration techniques apply to. Every AFE accelerates Web applications or any application that uses a browser for its user interface. The reasoning is that the industry is moving to browsers for all applications. This is an approach taken by Redline. Other vendors, such as Array Networks, F5 and NetScaler, believe that older applications are an important market, and thus their products apply many of the techniques to non-browser applications. This is more complicated, and not every technique can be applied to older client-server applications. If older client-server applications are important, care must be taken to understand what an AFE can and can't do for older client-server applications.

Security

The AFE adds a layer of security in the datacenter that the SLB didn't. Security is important because the AFE is in a unique position to detect threats. By terminating the TCP connection and acting as a full proxy, the AFE is in an ideal position to

detect denial of service (DoS) attacks and take action to prevent them from affecting the servers. Having the AFE perform this role means that the servers never see the attack, and the AFE can perform the function for the entire datacenter.

The AFE can better handle the attack—i.e., the load *it* receives during a DoS attack—due to specialized software and hardware. Additionally, the AFE can provide access control, with some AFEs such as Array Networks, F5 and NetScaler providing full VPN functionality.

It is also important to have the AFE involved in security because security increasingly needs to follow the strategy of protection in depth. Initially, security strategy was to have strong perimeter defense by installing a firewall at all connections to the outside world. This wall would keep the bad guys out.

This did not work out as well as hoped, because the bad guys figured out a way past this wall, and in addition, some attacks come from the inside. Instead, it is increasingly being recognized that security is needed in every device—a security in depth strategy. If the hacker gets by the perimeter firewall, another firewall or security device would stop them. The hacker would have to get by a series of security devices to reach the application, making it significantly less likely that the hacker would be able to successfully attack the application.

AFE vendors have embraced this strategy by implementing firewalls, VPNs and other security techniques, with the result that the datacenter is better protected and there is less chance that an application will be affected by an attack.

Conclusion

AFEs are clearly an important part of any datacenter. This was not always the case with SLBs. While SLBs provide a valuable service for any datacenter by monitoring availability, the load balancing part was only needed in datacenters where applications are spread across multiple servers. The AFE, while retaining the SLB feature, changes the equation by adding server off-load, application acceleration and security—needed features in any datacenter□

Companies Mentioned In This Article
Array Networks (www.arraynetworks.net)
F5 Networks (www.f5networks.com)
Fineground Networks (www.fineground.com)
Foundry (www.foundrynet.com)
NetScaler (www.netscaler.com)
Radware (www.radware.com)
Redline Networks (www.redlinenetworks.com)