W H I T E P A P E R

Accelerating And Optimizing Web-based Applications

May, 2004

Corporate Headquarters:

28 Hacharoshet St.
Or-Yehuda 60375
Israel
Tel. +972.3.634.6120
Fax. +972.3.634.6122

US Headquarters:

2440 Camino Ramon, Suite 225
San Ramon, CA 94583
Tel. +1.800.345.4461
Fax. +1.925.833.8894

Introduction

During a time when the Internet was growing at a paramount pace, network architects and infrastructure vendors focused greatly on network scale. In an age where money was of seemingly little concern, over-provisioning network resources was commonplace as more and more businesses justifiably relied on the Internet as a revenue-generating medium.

Times have changed somewhat. As the growth of the Internet has stabilized, financial issues have become more of a driving force for network architects and vendors alike. Network and infrastructure over-provisioning is no longer a readily viable alternative. Instead, network managers seek to maximize the resources available to them. In response, vendors have developed technology that allows existing resources to be used to their true potential. The need now lies in new technologies that allow the existing infrastructure to be optimally enabled and accelerated. At the same time, scale continues to be an issue as networks and applications still need to grow, but preferably with little or no resource addition, and definitely with no performance degradation.

Web infrastructures rely heavily on server resources, and as such, servers have become valuable commodity. Many technologies have been developed to address the scale of a server farm and its web applications. This document will primarily examine one of these technologies and discuss how web applications benefit from it. We will then discuss Crescendo Networks' unique approach towards server optimization and application enablement, its advantages over what's available today, and some of the complementary functionality that it provides while optimizing server performance.

Background

When it comes to discussing Internet driven applications, web traffic makes up most, if not all, of what a server has to deal with. Delving into the intricacies of web traffic first brings us to HTTP (Hyper Text Transfer Protocol), which is the communication protocol responsible for delivering web-based objects and applications from servers to clients. HTTP, in turn, relies on TCP as its IP transport protocol. TCP is a session-based protocol that provides inherent error checking and guarantees the delivery of its upper layer protocol (in this case, HTTP). Although TCP is an ideal delivery mechanism for web traffic, it is not completely void of drawbacks. Because of its robust nature, TCP possesses some inherent overhead that often leads to significant resource utilization on a web server.

To examine this point better, it's important to understand how TCP and HTTP work together. In the earliest versions of HTTP (versions 0.9 and 1.0), there was a one-to-one relationship between a web object request/response and a TCP session. In other words, each TCP session between client and server was used to carry one, and only one, object from server to client via HTTP. The figure below better illustrates this point:

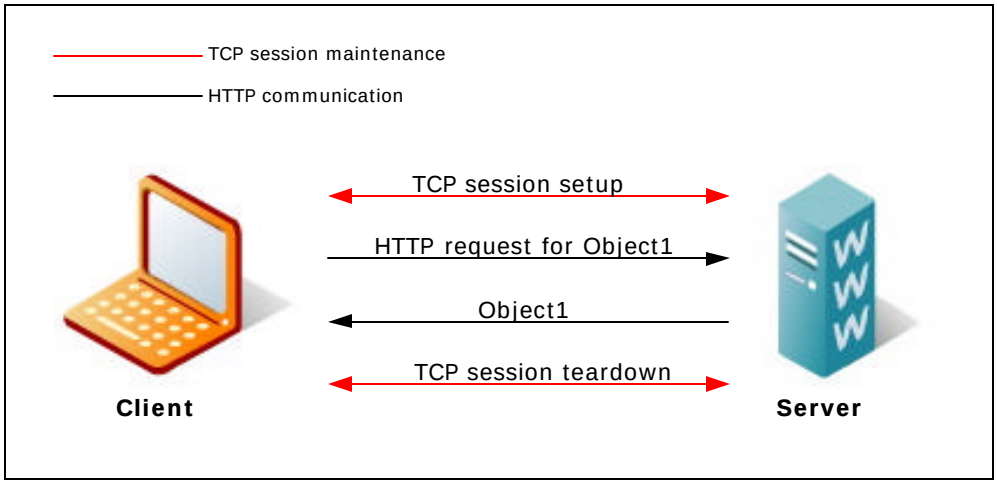

Figure 1 – A simple example of a classic HTTP request/response

Although Figure 1 is a simplified version of a web transaction, it depicts the major stages of the process. A TCP session is first opened between the client and the server. Then, an HTTP request (in this example, for "Object1") is sent from the client to the server over this newly established session. The server responds with the object and after its full delivery, the two sides close the connection. The process is repeated for every object requested from the server.

The TCP stack at each end is responsible for maintaining its side of the TCP connection, from the time the connection is established to the time that it's torn down. This is not necessarily a huge burden to the client side, since at any given time it may have only a handful of active connections. Comparatively speaking, the session maintenance on the server side is significant since it's handling a large number of connections from all its clients. As a matter of fact, using a model like that in Figure 1 causes most server resources to be spent on TCP session handling, rather than object serving.

As the prevalence of HTTP increased, it quickly became obvious that this was not an optimal way for HTTP to interact with TCP. HTTP developers recognized this and in the next version of HTTP (version 1.1, which is currently the latest version), **persistent connections** were introduced[1]. With persistent connections, the client is capable of requesting multiple objects from a server over a single TCP connection. The following figure is a simple illustration of how a persistent connection works:

[1] Actually, persistent connections were introduced as an addition to version 1.0 of the HTTP protocol, after it was originally released. However, HTTP 1.1 is where persistent connections became an official part of the HTTP specifications.

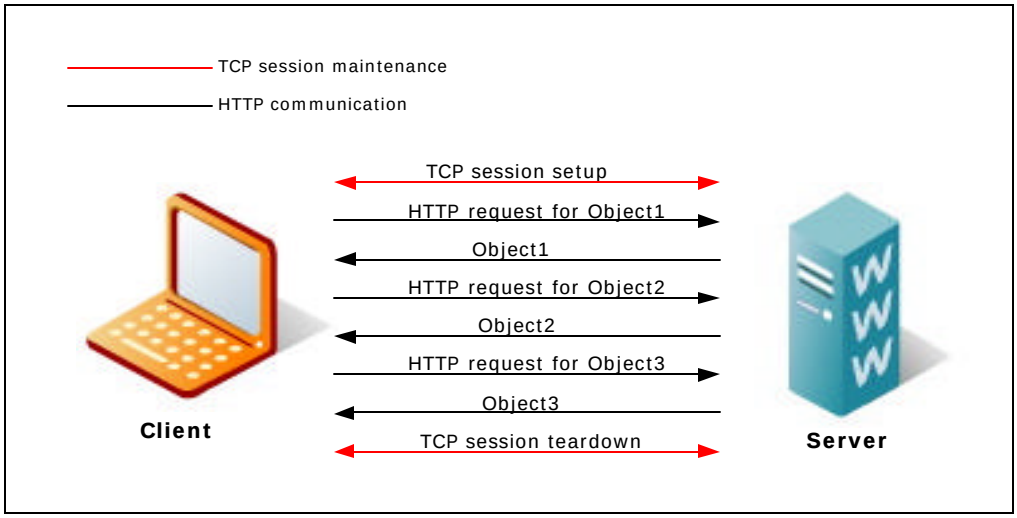

Figure 2 – A simple example of a persistent HTTP connection

As before, the client opens the TCP connection to the server. However, this time, it requests multiple objects over that single connection. The session is then torn down by either side of the connection, when deemed necessary. The number of objects fetched over a single connection is somewhat arbitrary. However, the benefits of persistent connections are immediately clear. Both sides have to deal with fewer TCP sessions, while the client ends up using less bandwidth for session maintenance and more for object retrieval. At the same time, the server minimizes the amount of resources spent on TCP session maintenance with each client. Support for persistent connections has been one of the most major improvements to the HTTP protocol, and specifically to the way it interacts with TCP.

The Problem

Although persistency clearly enhances and optimizes the interaction between HTTP and TCP, it doesn't necessary solve all TCP-related resource issues for servers, in and of itself. Persistent connections work well for the server if it has to deal with only a handful of web clients. In these cases, the total number of connections maintained by the server is greatly reduced, no matter how many objects are retrieved by the small client base. The same is true for the number of connection setups/teardowns the server has to deal with. However, the situation is much different in widespread Internet applications. When the client base is potentially the whole of the Internet, a very large number of clients will be approaching the web site and its servers. This will happen in a very unpredictable manner, which is why a server will still be burdened with opening and closing a large number of TCP sessions. This is much more prevalent if persistence is not used, but it's still an issue when it is. Furthermore, how fast a server can setup/teardown connections is only part of the problem. Now that persistency may be used, the server has to deal with sessions of a larger lifetime. This means that the server is now also responsible with maintaining a larger number of simultaneous TCP connections, which is yet another resource consuming task for that server.

These are the two main issues that concern servers, despite the use of persistent connections. Still, they are further complicated by the fact that in the real world, what a server sees is actually an arbitrary mix between persistent and non-persistent connections. This is caused by a number of reasons, such as:

Some clients still approach web sites through HTTP proxies that only support HTTP 1.0 and may not use HTTP persistence.
Automated clients, which may be used for anything from site monitoring to search engine indexing, use short-lived sessions and may never use persistence.
Some servers (such as ad servers) are only responsible for serving single objects, rather than whole pages. To them, persistence is of no advantage at all.

Finally, it is still the server's TCP stack that needs to maintain every session that it has with a client. If a client connection experiences transit problems (such as congestion or delay), it becomes a burden to both sides of the connection, including the server. So, the server continues to be hindered with TCP issues, regardless of whether the connection is persistent or not.

The bottom line is that in a web environment, servers have a significant burden when it comes to dealing with TCP session maintenance. This includes setup/teardown rate and the number of simultaneous connections each server can handle. Persistent connections improve the way HTTP and TCP work together, but they end up benefiting the client more than the server. As a matter of fact, studies have shown that in a high throughput environment, a server can spend up to 90% of its CPU cycles on dealing with TCP connections alone. This is why a solution that can offload these tasks from servers is necessary in order for them to reach their full web processing potential. Servers are utilized best if they spend the bulk of their resources on actual object serving and web processing, rather than connection management.

The Solution

Classically, there have been a number of approaches to addressing the problem of servers being over-burdened with TCP session maintenance. Load balancing presents a common solution in creating and scaling a server farm. Multiple servers are deployed (fronted by a load balancer) and share the responsibility for content delivery. Server resources are reduced on each server[2] and new servers are added, as more resources are needed. However, the problem still exists at each individual server. If we examine a single server within a server farm, it's still using most of its resources for TCP session management and not for actual object delivery and web processing. Basically, trying to grow this way just spreads the problem to more servers. This becomes expensive and not very scalable, mostly because the available resources are not being optimally allocated throughout the farm. Load balancing has many benefits of its own, but it's not the best way to solve this particular problem.

Another approach would be to use more powerful servers, perhaps with multiple processors and a significant increase in available resources. This also becomes expensive and not very scalable, while not really addressing the issue. The server may be stronger, but its resources are still not being used optimally. Upgrading to a bigger, more powerful server just means that a much more

[2] This applies to all server resources, including TCP session management and object delivery.

expensive machine is not using its available resources to their full potential. No matter how strong a server, a network administrator would still rather spend the server's resources mostly for web processing and object delivery, rather than connection maintenance overhead[3].

This leads us to the conclusion that a broader approach is necessary to address the problem properly. A successful solution is one that can significantly reduce the burden of TCP connection handling from a server. The solution must have the following qualities:

- It must be integrated into a web site infrastructure seamlessly.
- It must work within current HTTP specifications.
- It must drastically reduce the number of connection setup/teardown operations performed by the server.
- It must drastically reduce the number of simultaneous connections handled by each server.
- It must not introduce any new strains to the server or the network.

These issues are all addressed by a technology known as **Connection Consolidation**, whose main purpose is to minimize the number of connections a server handles at any given time (with regards to both setup/teardown rate and simultaneous connections handled), therefore maximizing the amount of resources it dedicates to actual object delivery. The basic concept is relatively simple to understand. First, let's look at a single server before we implement Connection Consolidation. Figure 3 shows a simplified version:

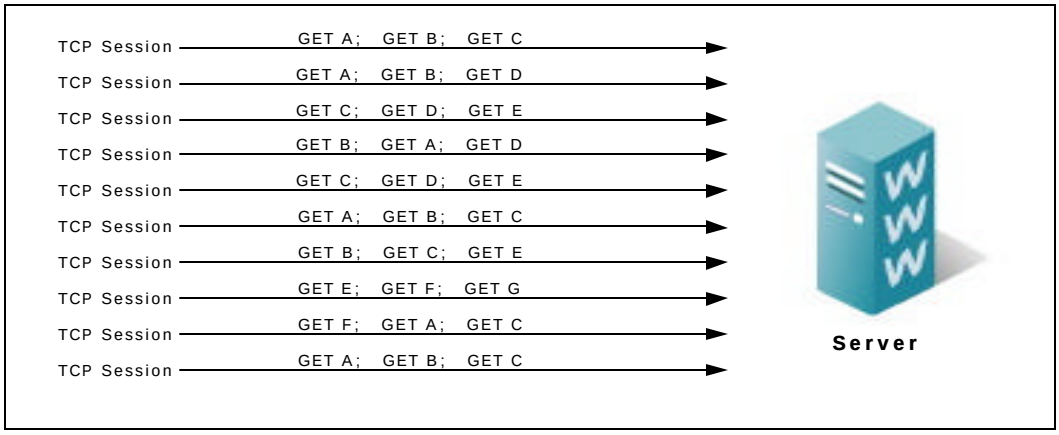

Figure 3 – A common sample of the TCP connections and HTTP requests a server sees

In Figure 3, the server has many TCP connections to maintain, while serving one or more objects per connection. These sessions can be coming from an arbitrary

[3] For reference, a recent Spec99 benchmark demonstrated that a single server with 16x1GHz processors was able to handle less than 13,000 simultaneous TCP connections; the equivalent of less than 6000 simultaneous Internet Explorer users (http://www.specbench.org/osg/web99/results/res2002q3/web99-20020819-00207.html).

number of web clients. Figure 4, below, shows the same scenario after Connection Consolidation has been implemented through an intermediary device:

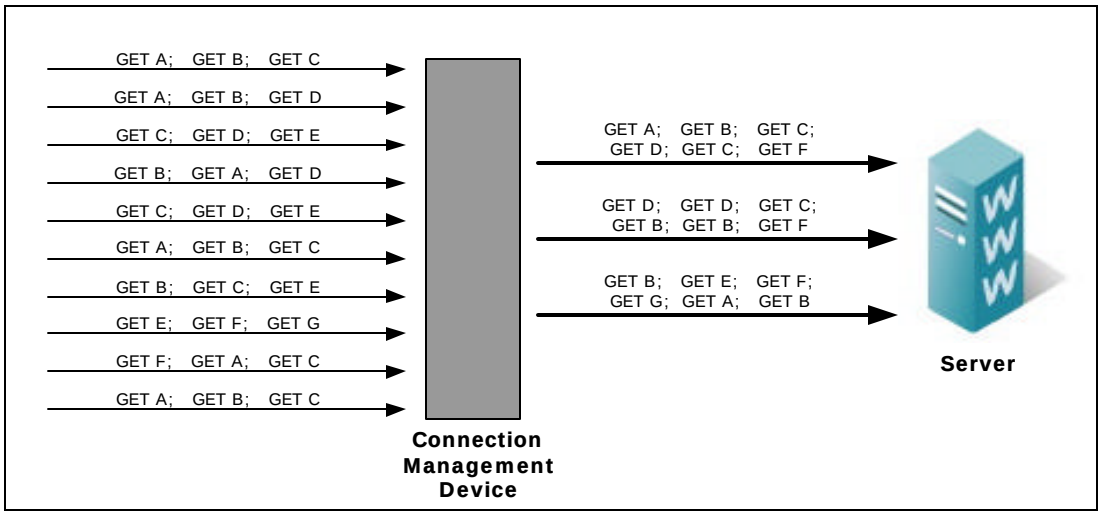

Figure 4 – Connection Consolidation

Here, the **Connection Management Device (CMD)** acts as an intermediary between the client base and the server. The task of the CMD is to act as the server towards the clients while acting as a client to the server. It then assumes the responsibility of dealing with all client-side connections. The CMD consolidates the front-end object requests from the many client-side connections to a few server-side connections. The connections on the server side are sessions with a large lifetime that are used over and over again as new object requests come in. Various algorithms may be used by the CMD to determine when a new session should be opened to the server, or which existing session a request should be carried over. The specifics of these algorithms are beyond the scope of this specific discussion. However, the benefits of Connection Consolidation through the CMD become quickly apparent:

- Since the CMD handles all client side connections, the server is no longer tasked with rapidly setting up and tearing down connections.
- Because of Connection Consolidation, the number of simultaneous TCP sessions a server has to deal with is drastically reduced.
- The CMD shields the server from client-side connection weaknesses. This means the server resources are not affected by any delay, congestion, or packet loss in the client connection. Therefore, the few connections that the server does operate through can perform at minimum overhead (and, therefore, maximum performance) for the server.

Connection Consolidation is a significant enabler for web servers and their applications. At the same time, it's important to realize that the overall solution would lose its advantage if the burden were simply displaced from the servers to the CMD. In other words, the CMD that provides Connection Consolidation as a service must have a strong enough architecture to be able to handle the

necessary TCP processing with ease. Otherwise, it's still the same problem, just in a different place.

As a final thought, it's interesting to note that Connection Consolidation is essentially a service provided by the CMD. Because of its unique location in a network, it would make sense for the CMD to complement Connection Consolidation with other functionality that fits into its specific geography within the network. Of course, this opens the door for a large number of auxiliary functionality, which warrants its own separate discussion.

Crescendo Networks' CN-5000 Product Line

To address the issues discussed so far, Crescendo Networks has developed the **CN-5000**, its flagship product line built from the ground up to provide superior server acceleration and resource optimization. The CN-5000 is built on Crescendo's **FreeFlow™** architecture. At the core of FreeFlow™ is Crescendo's powerful hardware-based platform, which is based on over 60 micro-engines, explicitly tasked with various application-specific processes. Built from the ground up, the hardware platform has been designed specifically for such application enablement and optimization. FreeFlow™ provides a very robust, powerful, and scalable foundation for the CN-5000, while allowing it to reach exciting new performance levels unmatched in the market today. In addition, FreeFlow™'s remarkably flexible architecture allows it to provide a means for easy growth. New features and applications can be implemented with market demand, while maintaining the high levels of scalability and performance.

The CN-5000 is deployed logically between the servers and the network that's responsible for delivering client requests to those servers. Usually, this puts the device between the servers and the load balancer / firewall / router. The device is deployed non-intrusively, causing no service interruptions or a need for network reconfiguration. The figure below shows a typical CN-5000 installation:

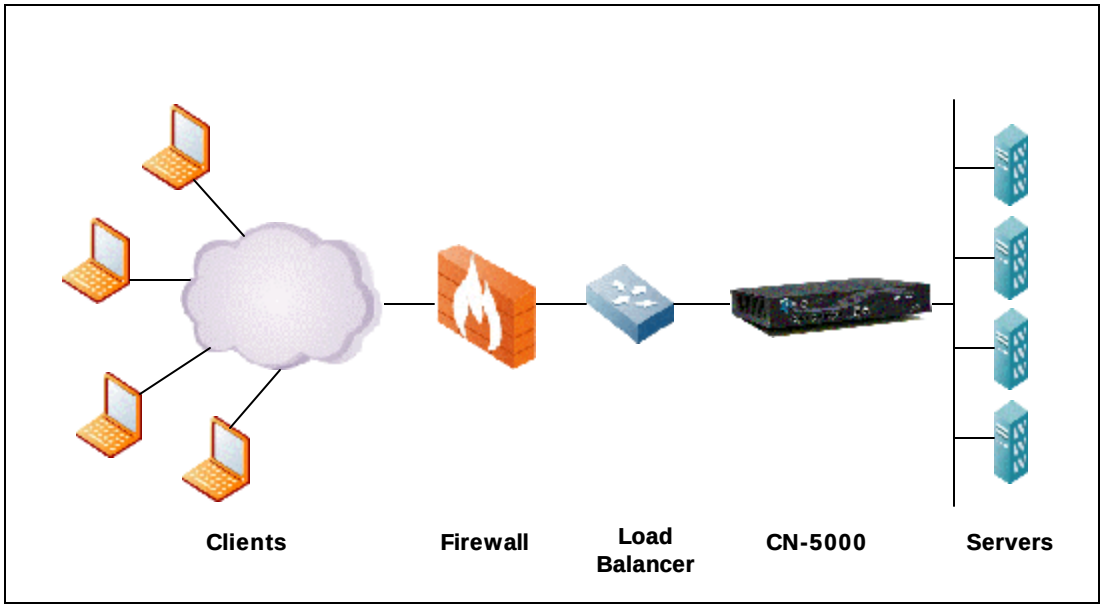

Figure 5 – A common network configuration for CN-5000 deployment

As shown in Figure 5, the CN-5000 assumes the final responsibility for delivering client requests to the servers. As such, it works together with the load balancer. The load balancer chooses the right server based on load/content/user rules. Subsequently, the CN-5000 provides optimization and acceleration services for the server chosen by the load balancer.

The core of CN-5000's functionality is based on Crescendo's Short-Lived Transaction (SLT™) technology. SLT™ has three main components that work together to provide the relevant services for the network:

- **Connection Management Algorithm:** Server-side sessions are managed through a set of advanced algorithms that provide an optimal approach to Connection Consolidation. These algorithms are dependent on a number of factors that include the type of request, client-side TCP connection performance, and an inherent knowledge of what connection profiles are best suited for the various web server operating systems. These factors provide a highly tuned approach to Connection Consolidation that allows the service to be deployed with the most impact.
- **Request Processing Algorithm:** As a session terminating intermediary, the CN-5000 is responsible for terminating client connections, processing the requests that these connections carry, and then delivering them to the server over existing, or new, server-side connections. SLT™ optimizes this process by using two unique phases for handling and delivering the requests to the server:
    - The device waits until the entire request has arrived from the client before it decides to deliver it to the server. This is incredibly beneficial in situations where long client requests are arriving over slow or problematic TCP connections. If the server were exposed to the weaknesses of these client-side TCP conditions, valuable resources would be tied up while it waited for the arrival of the complete request. By waiting for the entire request to arrive and then delivering it in whole to the server, SLT™ shields the server from client-side TCP conditions and allows it to minimize its processing time for each request.
    - Normally, a device performing Connection Consolidation would need to fully buffer an object in route from the server to the client before starting to transmit it to the client. However, at high capacity, this would require massive amounts of memory, which leads to the solution either not being very scalable or very cost effective. SLT™ addresses this issue by using partial requests on the server side, causing the server to break up large objects into smaller ones. This is coupled with proper memory management allowing high performance consolidation to occur with a reasonable amount of memory, making the CN-5000 both scalable and economical. This is completely transparent to the client who never knows or needs to worry about the way in which objects are fetched from the server by the CN-5000.
- **Response Optimization:** One of SLT™'s main objectives is to shield the server from weaknesses imposed on it by client connections that are subjected to WAN environments. These client connections experience packet loss, delay, and congestion, all of which would impact the server if it were exposed to them. By completely shielding the server from these issues, SLT™ allows the CN-5000 to communicate with the servers in a highly optimized environment. The server is already dealing with fewer

connections; and since those connections are purely managed by the CN-5000, the server can transmit its responses to the network at maximum throughput. Objects are served as optimally as possible, allowing the server to quickly move on to the next request to be processed.

Discussing SLT™ brings us to a topic that's commonly brought up alongside Connection Consolidation: *head-of-line blocking*. The concept is simple: since a single server-side connection is being used to retrieve multiple objects, it's possible that requests from two different clients are delivered to a server over the same TCP connection. Therefore, it's possible that Client1 is forced to wait until Client2's request has been processed and served[4]. Two kinds of delay can be caused by Client1's request:

- *Processing delay:* The server needs significant processing time for Client1's request, causing Client2's request to wait until Client1's request has been processed and/or sent.
- *Transmit delay:* The response to Client1's request is of significant size, causing a higher than average time to be spent by the server in transmitting the response object onto the network. So, even if Client1's request took minimal processing time, the response to Client2's request still needs to wait to be transmitted.

Head-of-line blocking is a common concern when it comes to Connection Consolidation. The basic way of dealing with it is to issue requests over idle connections, or to open a new connection if an idle one isn't available. SLT™, on the other hand, provides the CN-5000 with a **Non-blocking** architecture that addresses head-of-line blocking in many stages. This is a result of all three SLT™ components, allowing the system to complement its unique and dynamic session allocation mechanism with optimized processing of both client requests and server responses. This creates a multi-phase approach that makes head-of-line blocking of little or no concern.

SLT™ makes the CN-5000's approach towards server optimization and acceleration a unique one, far superior to the solutions commonly available today. Some of these advantages are highlighted below:

- The CN-5000 is a true TCP termination intermediary, acting as a full TCP proxy. Since it's been built from the ground up on top of the powerful FreeFlow™ architecture, no shortcuts have been taken to facilitate its operation. This means that client side session characteristics are not passed on to back-end servers. As already mentioned, this would cause client weaknesses to be exposed to the server, possibly influencing the rate or quality with which content is served. Content delivery performance from the server to the CN-5000 is truly maximized, while the CN-5000 handles all client-side connection flow control and management.
- Head-of-line blocking which can be a thorn in the side of Connection Consolidation is handled in a multi-phase approach. This all but eliminates this from ever being an issue in an application optimized by the CN-5000.

[4] This is due to the nature of HTTP, where requests have to be handled by the server in a serialized manner: request1, response1, request2, response2, etc. Response2 cannot be served until response1 is.

- True scalability is achieved through the request processing algorithm, allowing the CN-5000 to circumvent the need for massive amounts of RAM. This helps the network in two ways:
  - The efficiency with which the device operates allows the servers to deliver objects to the network with the least effort and most impact.
  - The device itself becomes highly scalable. In other words, the platform can easily handle growing amounts of traffic with no need to deploy additional devices.

The combination of FreeFlow™, its next generation platform, and the innovative algorithms and approach of SLT™ leads to significant performance improvements for servers and server farms. In-house testing results normally show a 5-fold performance increase per server. Figures 6a and 6b below illustrate these results:

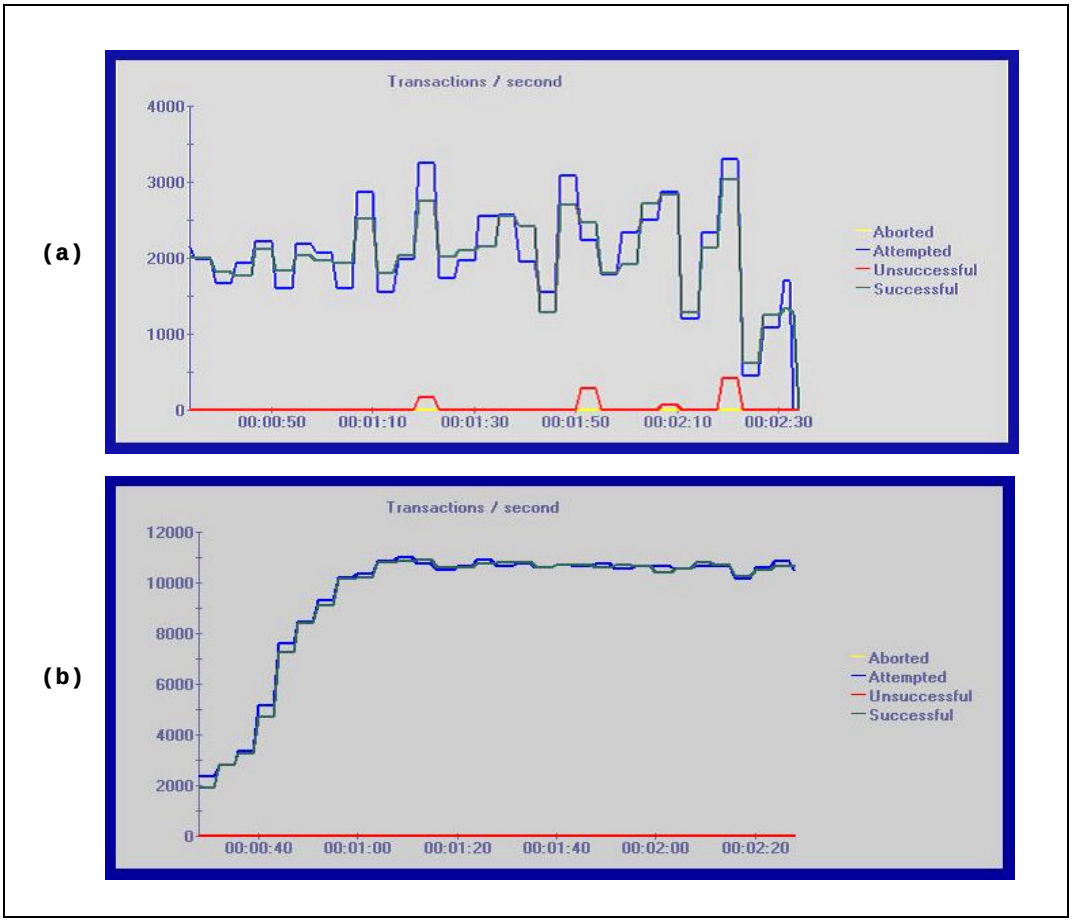

Figure 6 – Server performance before (a) and after (b) CN-5000 optimization

As Figure 6(a) shows, without CN-5000 optimization, a server processing simulated client requests serves an average of 2200 transactions per second. These transactions are generally inconsistent and also include some failures in

requests that were never served. In contrast, Figure 6(b) shows the same server optimized by the CN-5000. Here, the server can serve a consistent 11,000 transactions per server while experiencing no failures at all. Figure 6 clearly shows how dramatic the increase in server performance can be if its resources are optimized by the CN-5000.

The same optimization can be applied in the real world to a web application and its farm of servers. The result is true resource optimization for each server. The CPU cycles are used for business applications themselves, rather than the overhead associated with them. This provides an existing group of servers considerable relief and leaves them with significant room to grow into the full potential of the application.

Complementary Functions

Because of the CN-5000's unique position and scope of function within a network, it's a natural extension for it to provide some services on top of Connection Consolidation and its related algorithms. The following highlights some essential functionality that the CN-5000 offers alongside Connection Consolidation:

- **Compression:** Object compression can enhance the client experience by minimizing the number of bytes that need to traverse the path from web site to client, particularly the "last mile". Most browsers today can accept compressed data[5]. However, compressing data can be an effort for the server, especially if it's performed by purely software-based algorithms. The CN-5000 relieves the server of this task by providing compression services for objects in route from server to client. Compression is handled by a proprietary, hardware-based design that can reach a compression throughput of roughly 1Gb/sec. This is 8 to 9 times the performance currently available by any similar intermediary device.
- **SSL Acceleration and Offload:** It's a well known fact that the encryption and decryption tasks involved in handling secure content delivery through SSL (Secure Sockets Layer) are a severe burden to any web server. An intermediary device like a CMD is a natural place to offload this task from the servers. In addition, hardware-based accelerators can significantly boost SSL performance and make secure content delivery at rates close to that of non-secure content. The CN-5000 performs SSL acceleration and offload through a chipset that matches its high performance platform. The CN-5000's chipset can support more than 10,000 new SSL sessions per second, while being able to deliver 5Gbps of bulk encryption throughput. These high performance numbers are achieved because of the architectural approach of the CN-5000 towards SSL. Most SSL offload solutions today use an encryption chipset that is deployed above the standard software stack, therefore using the same resources as the rest of the system. With the CN-5000, however, the SSL functions are completely isolated, with their own dedicated resources. This accounts for the massive capacity of the CN-5000's SSL offload capabilities, making it highly scalable in such environments.

[5] Commonly used compression algorithms are gzip and deflate, both supported by common browsers and the CN-5000.

- **Distributed Denial Of Service (DDoS) Protection:** Many DDoS attacks exploit inherent weaknesses in IP stacks and/or standard web server operating systems. Because of the CN-5000's intrinsic capabilities in regards to dealing with TCP sessions and application transaction management, some of these attacks are easily prevented. Two such attacks that normally merit high concern are discussed below:
    - Because of its from-the-ground-up platform design, the CN-5000's TCP/IP engine is finely tuned to fit into web applications. This architecture can therefore inherently protect the servers from common IP stack attacks. The most significant of these is a SYN flood attack, which is meant to overburden a server's TCP resources by overloading it with new TCP connections that never fully open. The CN-5000's architecture allows it to handle more than 1 million new connections per second; enough to load a 1Gbps link with just SYN traffic. Furthermore, resources are not allocated by the CN-5000 until a TCP session is fully open. This combination makes SYN flood attacks obsolete as far as the application infrastructure is concerned.
    - Since the CN-5000 is delivering objects from server to client, it has insight into HTTP and the URLs it carries. Therefore, it becomes a natural candidate for protection against URL attacks. An example is when a large number of malicious requests are made to a server for a legitimate URL. This sort of DDoS attack could bring a server to its knees. The CN-5000, however, dynamically recognizes URLs under attack and intervenes when necessary. Intervention, in this case, is a patent-pending technique that can distinguish between man and machine, separating legitimate requests coming from actual clients from automated requests coming from malicious drones.

FreeFlow™'s flexibility allows these services to be easily added to the CN-5000's functionality. And it doesn't necessarily stop here. More and more services such as these can seamlessly be added to the CN-5000 as they become necessary.

Conclusion

The benefits of Connection Consolidation are self-evident. The FreeFlow™ architecture and SLT™ technology allow the CN-5000 to have a unique and powerful approach to this or any complementary function. The benefits of implementing such a solution in a network of servers are advantageous to both sides of web applications:

- The server side enjoys the benefits of acceleration and resource optimization. A network administrator can now use the available resources to their full potential, therefore improving productivity and consistency in the network. For example, a server that used to perform at 90Mbps can now perform at 400Mbps. The network and its applications can grow with ease, while the incremental capital and operational costs are severely minimized.
- The user side also enjoys benefits. Clients of web applications enjoy consistent response times that have been significantly improved because of the CN-5000's handling of the sessions. This leads to a superior client experience, which will ultimately lead to happier customers, more visitors, and an increase in revenue for the application.

The sum of these benefits, to both sides of the application, makes the CN-5000 an integral part of a web infrastructure, able to help any application perform at its best.

Finally, it's important to reiterate the fact that FreeFlow™ is a highly scalable and flexible architecture that allows services to be added with ease and at high performance capacity. The CN-5000 is the first step in applying FreeFlow™'s strengths. Furthermore, FreeFlow™'s hardware architecture can grow to support any network fabric. As next generation data centers are deployed, high-speed network fabrics will be used for directly connecting servers to other servers and network resources. New technologies such as RDMA (Remote Direct Memory Access) will be used to enable these networks to operate at maximum efficiency. As these technologies become available, Crescendo Networks' products can adapt to be seamlessly integrated into multi-gigabit server environments.