**Chapter 5**

# Five Common HTTP Analysis Recommendations

Having been involved in many profiling and troubleshooting analyses of web applications over the years, I have noticed that many of the same issues keep coming up, and therefore, require the same or very similar recommendations. The following are some common recommendations I have provided for web applications, in addition to what was discussed above:

## 1. Investigate WAN optimization deployment

The area of WAN optimization has been around for a number of years. This service usually involves adding two devices between the client and the server that work to optimize traffic before going over the WAN and then de-optimize that traffic when it arrives at a local network. One of the first vendors to offer this service was Riverbed in 2002. Now, on the other hand, there are more than ten vendors in this space, with Riverbed and Cisco being the two largest.

This recommendation is usually given to the application team, suggesting that they look into how WAN optimization can positively or negatively impact their application. Many organizations have long since utilized optimization techniques for improved performance, but not everyone is always aware. Such devices are not always deployed everywhere on a network. They are most often between remote locations, where the users are, and the datacenters, where servers tend to be. An application team needs to understand where optimization devices are deployed and how or whether their application could be impacted by this.

Such a recommendation is particularly needed if an application resides in an area of the network where there is no optimization in place, but there are expansion plans for this application to more locations. For example, an application in which its servers and users only reside in a datacenter location or campus is not going through any optimization devices. Therefore, there would be no concern about the impact of optimization. However, if this application has global usage within your organization, user transactions will go through these devices, and the application team should understand the potential impact.

In general, WAN optimization positively impacts applications, especially HTTP. However, I have seen instances in the past when this was not always the case, albeit more so for non-HTTP data. For example, some optimization techniques can adversely affect some database traffic. In a three-tier application architecture, it is always important to understand the deployment of the database server with respect to the application or web server with which it communicates. Therefore, it is important to recommend this to the application team so that they are not surprised later by any slowness after application deployment. Furthermore, this is another reason why you, as the analyst, should ask many of the questions I previously discussed.

## 2. Increase TCP Receive Window Size

Since HTTP runs on top of the TCP protocol, one of the common reasons for a web application to perform slowly, particularly during the download of a file, is a TCP receive window that is too small. When a user requests the download of a PDF file, for example, the user's PC and the server specify their receive window sizes.

The default maximum TCP window size is 65,535 bytes. Current Windows and Linux operating systems support this. However, sometimes network settings are implemented incorrectly on a server or changes are made on a PC causing a deviation from the default. Furthermore, applications have the ability to set the TCP congestion window for both receive and send with a socket call that can limit TCP's performance. Therefore, when looking at a web application that users are saying is slow, one thing to check for is the TCP window sizes, and make the appropriate recommendation to update them.

However, be careful with this recommendation on PCs and servers with limited RAM. Data stored for the TCP congestion window is kept in RAM, and a machine with low memory will be adversely affected from increasing the window size too much.

## 3. Optimize Database Queries with Multiple Row Requests

Most web applications today are multi-tiered. Much of the data is stored in a database that is talking to a web or application server. At many organizations, most HTTP applications contain four tiers – client, web server, application server, and database server.

When a user makes a request to the web server for a particular file, some or all of that data may be stored in a database. The application server relays the request from the web server to the database server. However, because the database is often near an application or web server during the testing and quality assurance (QA) phases, application developers may not always focus on how the two communicate.

Furthermore, much of the communication is inherent in how databases work, and the developers may not have a whole lot of control over this for various reasons. Therefore, the requests are not very efficient, leading to a lot of chattiness.

There are also cases when the client makes direct requests to the database with this same type of inefficiency. One of the biggest reasons for this is that the SQL queries used request data one database row at a time. This is typical of database communication without explicit configuration changes.

You can tell whether this is the case through a couple of methods. One method is to look at the decode summary in your packet analyzer and see the data transfer between the two tiers. If your analyzer does not have the ability to decode database traffic, you should consider using Wireshark. At any rate, for an Oracle database, for example, the client requests will state “fetch a row” in the decodes. This is telling you that the database client is asking for data one row at a time.

A second method is to look at your packet analyzer time sequence diagram. Most come with one, but may have a different name for it. Whatever the name your analyzer uses, look at the sequence diagram and pay attention to the amount of interaction between the tiers. The busier the interaction, the chattier the communication. Chatty communication very likely means the database requests are for one row at a time.

If the above two methods used show single-row requests, you should recommend this be changed, if possible.

# 4. Investigate remote desktop deployment

Remote desktops have been in use for many years. Windows RDP and Citrix ICA have been two protocols and related solutions that organizations have used for many years to allow remote users to access a web application that would otherwise be terribly slow if run over the WAN. There are many other players today, but this option is something that can help.

This recommendation is usually a last-resort option. Many organizations already have server farms dedicated for remote desktop services. However, it may not always be easy or cost-advantageous for an application to do this.

There are times when the application team cannot or will not accept any of the other recommendations to remedy a slow application. This is most often the case because the application is vendor-based, and changes are either not likely occur or may not occur for a long time. If that is the case, testing the viability of remote desktop solutions should be considered.

# 5. Enable TCP window scaling

TCP window scaling is a capability of the TCP protocol to has a receive window that is larger than 65,535 bytes. The reason for this is the number of bits preserved for the window, which is 16 bits. Since the maximum this provides is 65,535, a scaling factor is used so TCP can specify the true size of its window. Once scaled, this allows the TCP protocol to send more data over the network with less acknowledgements, allowing for more network throughput and faster download times.

Window scaling is supported by practically every operating system. However, it is only implemented by default in certain Windows versions such as Windows 8 and Windows Server 2012. If your organization is still running older versions of Windows, double-check your TCP settings. Linux and other Unix-based operating systems have had this implemented by default since 2004.

When scaling is enabled, the TCP protocol can theoretically increase the receive window size to as much as 1,073,725,440 bytes, or about 1GB. This is done using a scaling factor from 0 to 14. To get the scaled window size, you multiply the current window size by 2^(scaling factor). For example, if you want to double the TCP receive window size from 65,535 to 131,070, a scaling factor of 1 is used (65,535*2^1=131,070).

It is always a good thing to send more data if the network can support it, which many, but certainly not enough, organizational WANs can do today with Gigabit speeds. TCP scaling allows for this, and therefore,

should be recommended as often as possible, especially for HTTP applications that are file-download intensive.

However, an adverse effect is that this increases memory use on a PC or server. Since the increased window size in bytes is stored in RAM, enabling TCP window scaling can degrade PC or server performance on a machine with low memory, especially if the scaling factor is too high. This is not likely to be an issue on many of today's client machines, since many contain multiple gigabytes of RAM built-in, but there are still companies running older server hardware for continuity purposes. If your company is one, this is something you must be cognizant of.

For scaling to work, like other options in TCP and HTTP, both sides must support it. To determine whether TCP scaling is enabled, you can check the TCP options in the protocol decodes in your packet analyzer. When looking at the decodes for the TCP protocol, you should see a "Window scale:" field with the value of the scaling factor.