FrequentlyAskedConnectionManagementQuestions

Connection Management FAQ

This document addresses questions about connection management which have been raised repeatedly on the HttpClient and HttpComponents mailing lists.

Connections in TIME_WAIT State

After running your HTTP application, you use the netstat command and detect a lot of connections in state **TIME_WAIT**. Now you wonder why these connections are not cleaned up.

What is the TIME_WAIT State?

The TIME_WAIT state is a protection mechanism in TCP. The side that closes a socket connection orderly will keep the connection in state TIME_WAIT for some time, typically between 1 and 4 minutes. This happens *after* the connection is closed. It does *not* indicate a cleanup problem. The TIME_WAIT state protects against loss of data and data corruption. It is there to help you. For technical details, have a look at the Unix Socket FAQ, section 2.7.

Some Connections Go To TIME_WAIT, Others Not

If a connection is orderly closed by your application, it will go to the TIME_WAIT state. If a connection is orderly closed by the server, the server keeps it in TIME_WAIT and your client doesn't. If a connection is reset or otherwise dropped by your application in a non-orderly fashion, it will not go to TIME_WAIT.

Unfortunately, it will not always be obvious to you whether a connection is closed orderly or not. This is because connections are pooled and kept open for re-use by default. HttpClient 3.x, HttpClient 4, and also the standard Java HttpURLConnection do that for you. Most applications will simply execute requests, then read from the response stream, and finally close that stream.

Closing the response stream is *not* the same thing as closing the connection! Closing the response stream returns the connection to the pool, but it will be kept open if possible. This saves a lot of time if you send another request to the same host within a few seconds, or even minutes.

Connection pools have a limited number of connections. A pool may have 5 connections, or 100, or maybe only 1. When you send a request to a host, and there is no open connection to that host in the pool, a new connection needs to be opened. But if the pool is already full, an open connection has to be closed before a new one can be opened. In this case, the old connection will be closed orderly and go to the TIME_WAIT state.

When your application exits and the JVM terminates, the open connections in the pools will *not* be closed orderly. They are reset or cancelled, without going to TIME_WAIT. To avoid this, you should call the `shutdown` method of the connection pools your application is using before exiting. The standard Java HttpURLConnection has no public method to shutdown it's connection pool.

Running Out Of Ports

Some applications open and orderly close a lot of connections within a short time, for example when load-testing a server. A connection in state TIME_WAIT will prevent that port number from being re-used for another connection. That is not an error, it is the purpose of TIME_WAIT.

TCP is configured at the operating system level, not through Java. Your first action should be to increase the number of ephemeral ports on the machine. Windows in particular has a rather low default for the ephemeral ports. The PerformanceWiki has tuning tips for the common operating systems, have a look at the respective Network section.

Only if increasing the number of ephemeral ports does not solve your problem, you should consider decreasing the duration of the TIME_WAIT state. You probably have to reduce the maximum lifetime of IP packets, as the duration of TIME_WAIT is typically twice that timespan to allow for a round-trip delay. Be aware that this will affect *all* applications running on the machine. Don't ask us how to do it, we're not the experts for network tuning.

There are some ways to deal with the problem at the application level. One way is to send a "Connection: close" header with each request. That will tell the server to close the connection, so it goes to TIME_WAIT on the other side. Of course this also disables the keep-alive feature of connection pooling and thereby degrades performance. If you are running load tests against a server, the untypical behavior of your application may distort the test results. [[BR] Another way is to not orderly close connections. There is a trick to set SO_LINGER to a special value, which will cause the connection to be reset instead of orderly closed. Note that the HttpClient API will not support that directly, you'll have to extend or modify some classes to implement this hack.

Yet another way is to re-use ports that are still blocked by a connection in TIME_WAIT. You can do that by specifying the SO_REUSEADDR option when opening a socket. Java 1.4 introduced the method Socket.setReuseAddress for this purpose. You will have to extend or modify some classes of HttpClient for this too, but at least it's not a hack.

Further Reading

Unix Socket FAQ

java.net.Socket.setReuseAddress

Discussion on the HttpClient mailing list in December 2007

PerformanceWiki

netstat command line tool