acmqueue

A Paucity of Ports

Debugging an ephemeral problem

Dear KV,

I've been debugging a network problem in what should be a simple piece of network code. We have a small server process that listens for commands from all the other systems in our data center and then farms the commands out to other servers to be run. For each command issued, the client sets up a new TCP connection, sends the command, and then closes the connection after our server acknowledges the command.

In our original data center the controller had no problems, but then we moved to a larger data center with more client machines. Now we frequently cannot make connections when trying to execute commands, slowing down the overall system. It's such a simple design that I have no clue what could be going wrong. The controller itself is only one page of code! I suspect the network gear in the new data center is to blame, and that it cannot handle the load of incoming connections.

Connection Denied

Dear Denied,

Only one page of code and yet you can't find the bug? Perhaps the code has no bugs, and it *is* the switches. Certainly this is the usual blame game for developers: "My code worked before! You changed something!" Blame the change.

The problem is that you have to blame the correct change. Nothing is worse than venting your spleen at someone and then being proved wrong and having to admit that you were wrong. I hope you didn't vent your spleen at the network admins, because in this case you are definitely wrong.

Now I'm not going to attack your code; it is very likely as correct as you claim it to be. The blame here lies with a failure to understand TCP and how to make efficient use of it.

When your client makes a connection to the server, it uses four pieces of information to identify the connection uniquely among all the others that take place between these two hosts. These pieces of information are the source and destination IP addresses and the source and destination ports. Now, I am sure your code picks a single destination port to send to; for example, the Web uses port 80 and e-mail uses port 25. Whatever you picked, this was probably not a problem.

What do you choose for the remaining bit, the source port? Most code, and certainly code that is simple and takes up only a single page, lets the operating system decide. When you let the operating system decide the source port, it picks what's known as an ephemeral port. Note that *ephemeral* is defined in the dictionary as meaning, "lasting for a very short time," but of course *short* is in the eye of the beholder, or in this case, in the timer code for TCP. Whenever TCP closes a connection, the connection state remains active in the operating system in what is known as the TIME_WAIT state. The TIME_WAIT state lasts for at least two times the maximum segment length of the connection, but in a data center you're never going to have this state last for less than 60 seconds, which is the minimum value for 2MSL (twice the maximum segment lifetime) defined by the system.

OK, now you know that *short* in your case means one minute, so that's how long a closed socket

will stick around. While that closed socket is around, the ephemeral port number that the operating system picked cannot be reused. That should be fine, you might think, since there must be a lot of port numbers available. Unfortunately, TCP was defined at a time when the idea of having millions of connections between machines wasn't really practical, or even possible, so the number of possible ports may surprise you: it's 65,536. Ephemeral ports don't get even that full set, but a subset, and on most modern operating systems the range is about 16,384.

That limit means that you can open only 16,384 connections with ephemeral ports unless you modify some of the tuning variables in your operating system. Even if you do increase the range, say to 32,768, it is quite easy for modern computers to run through that number of connections in a minute—and remember, any connection you have closed will hang around for a minute after it is closed. The reason that this bug has not shown up till now is that you have scaled your systems to the point where they are, collectively, able to use up their ephemeral ports in one minute.

While you might now have many wild ideas about extending the TCP or changing the minimum value of 2MSL, I can tell you that you should forget them all. TCP was designed to maintain long flows in the Internet, and not to be some sort of short message service. The overhead of setting up and tearing down a TCP connection should be a clear indication that to use it efficiently requires putting more than a few bytes over it at a time, a lesson that the early designers of HTTP did not learn, but that's a story for another time.

The correct way to use TCP in your situation is to have a local process on each client that maintains a persistent connection to the central server. Without the constant opening and closing of sockets, you will not cycle through and use up your precious ephemeral ports.

KV

KODE VICIOUS, known to mere mortals as George V. Neville-Neil, works on networking and operating system code for fun and profit. He also teaches courses on various subjects related to programming. His areas of interest are code spelunking, operating systems, and rewriting your bad code (OK, maybe not that last one). He earned his bachelor's degree in computer science at Northeastern University in Boston, Massachusetts, and is a member of ACM, the Usenix Association, and IEEE. He is an avid bicyclist and traveler who currently lives in New York City.