Socket Issues in MOS Communications

A description of known potential issues with generic socket communications, how these can effect the Media Object Server Protocol and recommended methods of remediation

Sean Cullinan, Systems Engineer mailto:Sean_Cullinan@enps.com
Associated Press Broadcast Technology / ENPS May 9, 2001

Issue:

An improperly closed socket connection between an NCS and a Media Object Server can result in loss of MOS communications or a state where an application restart is necessary to restore communication between the NCS or MOS.

Effect on Users:

When communications between the NCS and Media Object Server are lost the users will report MOS Based functionality will either stop or behave abnormally. These functions would be related to running order creation, story send functionality, or other operations that require a connection from the NCS to the Media Object Server.

What can be observed on the NCS and Media Object Server:

The state of the TCP/IP sockets on the NCS and Media Object Servers can be examined and can be indicative of when the problem occurs. The command "netstat –an" will display a detailed list of the state of all sockets on a server. When you run this command on the NCS after an improper socket disconnect you will observe the socket used for communications to the Media Object Server has a status of FIN_WAIT_2. On the Media Object Server the "netstat –an" command will also show the socket used to listen for connections from the NCS now has the status of CLOSE_WAIT.

Detailed Description of the Problem:

Connections between the NCS and the Media Object Server need not be persistent. In an environment with multiple Media Object Servers the NCS may choose to disconnect with one Media Object Server before communicating with another. When this happens the NCS initiates a closing of the socket to the first server's listening port. This begins the standard TCP/IP socket breakdown process. We have been able to determine and document that during this disconnection process, Media Object Servers from some manufacturers have had difficulty with the TCP/IP socket breakdown process.

TCP/IP Breakdown Process

The TCP/IP breakdown process consists of 4 parts. The bulk of this process may be handled automatically by the TCP control used by the programmer, and many programmers never delve down into this low of a level in TCP programming. Below is a description of the low level TCP events that occur when a socket is closed by the NCS:

1) The NCS sends a close command to the MOS server to close down the socket on port 10540 or 10541. At this point the status of the NCS's socket connection is FIN_WAIT_1.

2) The MOS system sends an ACK back to the NCS acknowledging that it has received the close command. The status of the socket on the MOS side is now CLOSE_WAIT.

3) Upon receiving the ACK from the MOS system, the NCS sends another ACK out to the MOS system. It's a little tough to follow here but essentially the NCS is acknowledging that it knows that the MOS system knows about the close command. The status of the socket on the NCS is now FIN_WAIT_2.

4) At this point the MOS system should send a final ACK to the NCS completing the breakdown of the socket. If this doesn't happen, the MOS will remain stuck in a CLOSE_WAIT status (from step 2) and the NCS in a FIN_WAIT_2 status from step 3.

The problem we have documented happens in the Media Object Server during the 4th part of this breakdown procedure. When the NCS closes the socket in step 1, step 2 and 3 happen immediately and automatically. The 4th part of the procedure, however, is where things break. For the 4th part to take place the programmer of the MOS server may need to issue a command to the listening socket control to close the socket, otherwise this part of the breakdown process will not occur. The results, as stated in part 4 in the above description, is that the NCS will have a status of FIN_WAIT_2 on the MOS Protocol port being closed, and the MOS server will have a status of CLOSE_WAIT on the MOS Protocol port being closed.

Effects on the System

By leaving the sockets in FIN_WAIT_2 on the NCS and CLOSE_WAIT on the MOS server the connection between the two machines is never truly closed. The connection is still partially up but is in a stagnant state in which no data can be transferred between the two systems even though they are still connected. However, the NCS is now free to connect to other Media Object Servers.

A potential problem will arise if the NCS attempts to re-use the old connection the next time it wishes to communicate with the Media Object Server to which the last socket was not cleanly broken. This state should be handled by application code in the NCS, but it is avoidable altogether if the Listening sockets in the Media Object Server are closed properly.

If the Media Object Server is not closing Listening sockets properly the TCP/IP stacks of both the Media Object Server and the NCS can melt down. The Media Object Server will become overwhelmed with sockets in the CLOSE_WAIT status, a condition that will not clear until the sockets are programmatically closed by the Media Object Server application, most often when the application is closed. The NCS can also become overwhelmed with sockets that are stuck in the FIN_WAIT_2 status.

Although Microsoft O/S clears the FIN_WAIT_2 status after 4 minutes this can still lead to problems. Essentially, high numbers of FIN_WAIT_2's and CLOSE_WAIT's will eventually bring a TCP/IP stack to its knees and must be avoided.

Finding and Avoiding Improperly Closed Sockets

So how do we get insight into what is going on with a systems socket states? The first tool that we recommend, "netstat," comes with every flavor of Windows NT and UNIX.

The following command will show you the status of ports on the system:

netstat –an

This command will display a listing of all sockets on the system. The states you will see for sockets are described as follows:

Listening: This is a socket that has an active listener on it that is waiting for incoming connections

Established: This is a socket that has an established connection. The IP address of the machine that the connection is made with is listed in the "Foreign Address" column.

TIME_WAIT: This is a socket that has been brought down cleanly. These will be cleared by the O/S's TCP/IP stack. In the Windows environment these remain for 4 minutes.

FIN_WAIT_1: This status indicates that this server started the socket closure procedure. It has sent a close command to a remote server and is waiting for the ACK to come back.

CLOSE_WAIT: This status indicates that the socket has received a close command and has sent back an ACK to acknowledge that it has received the close command.

FIN_WAIT_2: This status indicates that the socket has sent the close command, received an ACK back from the other server, and has sent an ACK back to the other server. At this point this machine is waiting for a final ACK to come back from the other end signifying that the socket is fully closed.

Testing your MOS Application:

The easiest way to test your MOS application is to telnet from another machine to the Media Object Server on the MOS protocol ports. Once the telnet connection is established simply bring it down. Typing the following at the command line will do this:

telnet <name of MOS server> 10540

or

telnet <name of MOS server> 10541

Once the telnet session is established you should get a blank screen. You should then close this telnet session by closing the window it is running in. After doing this run the command "netstat –an". If the MOS server has not correctly reset the listening socket you will see a FIN_WAIT_2 description on the machine you ***telneted from*** and a CLOSE_WAIT on the machine you ***telneted to***. If a TIME_WAIT status is observed on both systems then the socket was brought down correctly.

How to avoid this problem:

Fixing the problem will vary based on the programming language and socket control used by the developer. Below is a very basic VB example that avoids the FIN_WAIT_2 problem for a listening MS winsock control. On the close event for the winsock control the following code closes the listening socket whenever a close event is received (assume the name of the listening winsock control is tcpServer):

```
Private Sub tcpServer_Close()
        tcpServer.Close
End Sub
```

The socket is closed correctly by issuing the close command to the listening socket control during a close event.

References

The following reference were used in putting together this paper:

Microsoft Knowledge Base article Q137984
This article describes TCP connection states and how to read Netstat (NETSTAT.EXE) output.
http://support.microsoft.com/support/kb/articles/Q137/9/84.asp

Winsock Source Code Website
Website contains a plethora of information on Winsock programming.
http://www.stardust.com/winsock/ws_src.htm

Winsock Programmers FAQ
This website answers many questions that a winsock programmer may have. Specifically addresses FIN_WAIT and socket breakdown questions.
http://www.cyberport.com/~tangent/programming/winsock/articles/debugging-tcp.html

Microsoft Knowledge Base article Q198663
This article addresses what happens when the TCP/IP stack exhausts all of the socket resources as a result of too many CLOSE_WAIT's or FIN_WAIT_2's.
http://support.microsoft.com/support/kb/articles/Q198/6/63.ASP