EVERYTHING A DBA SHOULD KNOW ABOUT TCPIP NETWORKS

Chen (Gwen) Shapira,HP Software-as-a-Service

1. TCP/IP Problems that DBAs Can Face

In this paper I'll discuss some of the network problems that I've encountered in my job as a DBA. I will give some background about each problem, explain the techniques and tools I've used to debug and solve the problem and in the process I will also explain some necessary parts of how TCP works. I've attempted to keep the paper practical and therefore left out a lot of theory. I did give reference to places where more background information is given.

The problems I'll discuss are:

1. On a RAC system, a client occasionally fails to connect with ORA-12545 error.
2. Users complain that a long running batch job never finishes running, or just quits in the middle.
3. Application needs to read large amounts of data off the database. Can we reduce the time spent on network delays? And what's all this talk about SDU and MTU?
4. Redo logs need to be shipped to a remote disaster recovery site. Can we anticipate in advance how long the data shipping will take? Are there ways to make it faster?

Obviously these are not the only problems that DBAs are likely to encounter. In my experience they are common, they represent a larger class of problems and they allow me to demonstrate a wide range of tools and techniques.

Note that I am assuming quite a bit of prior knowledge of networks. I assume some familiarity with Ethernet, knowledge of what is a router and what is a firewall. I am assuming that the reader knows what is the IP protocol, what is a packet, what is TCP and what is a connection. I explain some of the more advanced issues, but if you don't know what is a packet and what is the difference between TCP and UDP, I suggest reading an introductory text. Perhaps this one: http://www.doc.ic.ac.uk/~ih/doc/pc_conn/tcpip/intro/intro2.html

Before we start discussing the problems, allow me to introduce the most useful network troubleshooting tools.

2. Wireshark and TCPDump – Network Tracing and Packet Sniffing

Wireshark is an open-source GUI tool used to troubleshoot network issues. It is multi-platform and can be used on the different Unixes, Windows and even Mac.
TCPDump is a command line packet analyzer that exists for most Unix systems. Since it is a command line tool, it is often more convenient to use when we want to capture network traffic on a server environment. Wireshark can later be used to view and analyze the results of a network capture taken by TCP Dump. Note that capturing TCP traffic on the server will require root access.

Both these tools are the most important tools in your toolkit when it comes to troubleshooting network issues. The main reason is that they show you exactly what is going on at the network level. Oracle does not do a good job of accounting for time spent in the network, and it will not report even serious network issues such as frequently lost packets in a way that can be understood and solved. Looking at the network traffic and seeing exactly what the client and the server are sending each other and when eliminates a lot of uncertainty and guesswork.

Having a good network capture also generates a much clearer discussion around network problems. I've had users complain about problems connecting to the database and sending me a TNSNAMES.ora entry that looked perfectly good. When I asked them to capture their connection attempt and send it to me, I discovered that they were attempting to connect to a completely different location. Having the real data instead of a guess can save hours of troubleshooting. Another benefit of having a good capture is that network administrators take captures seriously and will act on evidence of retransmits and lost packets much faster than they would to complaints of “slow network”.

Explaining how to use Wireshark is beyond the scope of this paper. I give few examples below, but beyond that, the documentation on http://wiki.wireshark.org/ is excellent.

And now we can continue on to troubleshooting.

3. On a RAC system, a client occasionally fails to connect with ORA-12545 error.

Symptoms:

Usually happens on a new RAC environment, or new clients connecting to an existing RAC environment. Everything looks configured correctly from the server, but some or all of the clients occasionally fail to connect with ORA-12545 error. The client that fails to connect usually succeeds on a retry, making the problem more confusing.

Debugging:

Start by verifying that the client can connect to each RAC instance separately, as if it was its own database server. If connectivity to one of the RAC nodes fails consistently, you will have to debug connectivity to this node separately. [1]

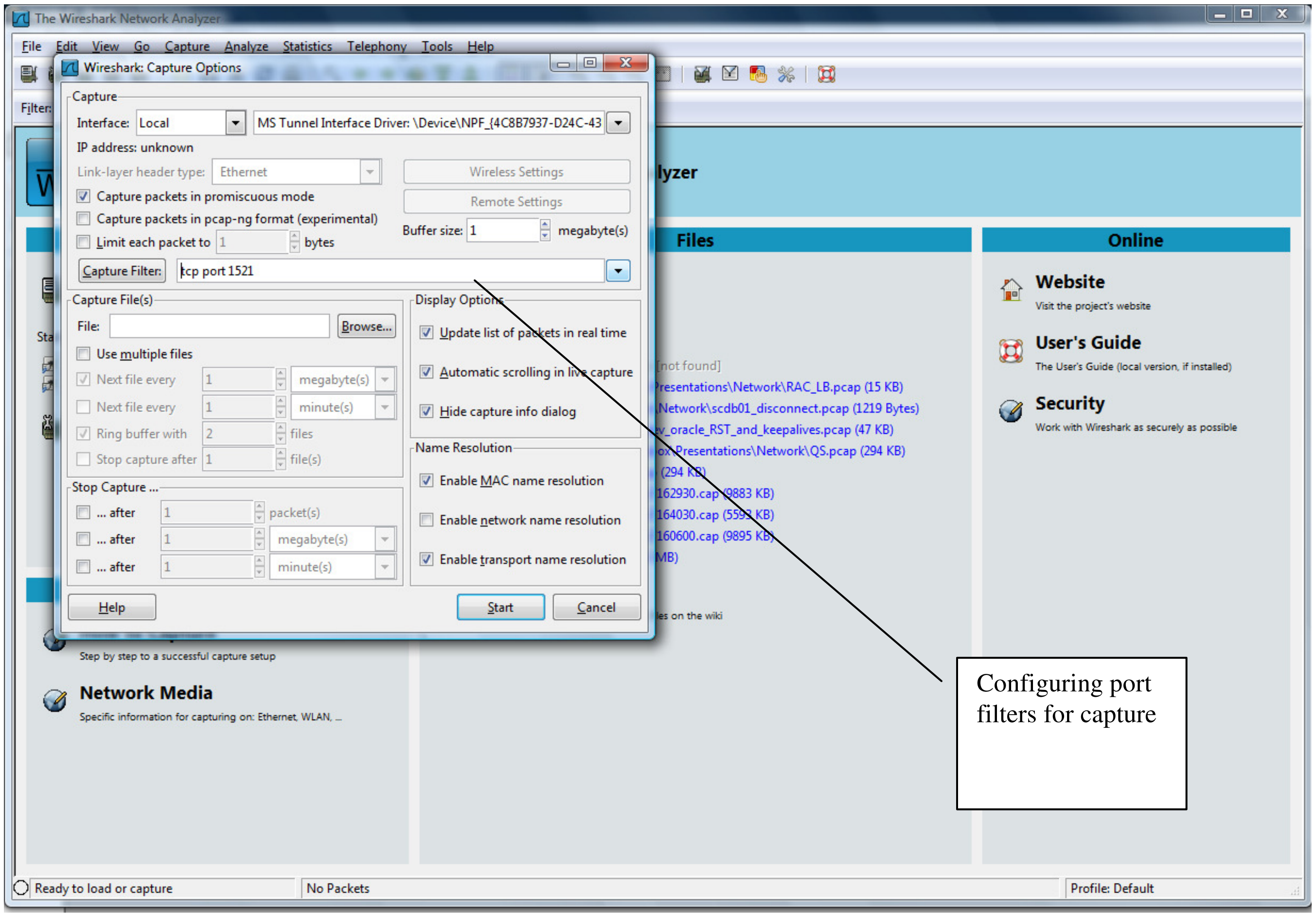
For now, we’ll assume now that connectivity to each node works perfectly, yet connecting to the RAC services sometimes fails. This calls for taking a deeper look at what happens that causes the connection to fail.

Start Wireshark and start capturing traffic. It will be easier if we set filters to only capture traffic going to port 1521.

Note that if you know you are going to capture large amounts of data, you can also configure capture to use multiple files, use the files as a ring buffer (overriding older data with newer data after the available space has been used) or to stop capture automatically after a specific amount of data was captured. This was it is safe to leave capture running overnight, knowing that you will not run into diskspace issues on the device.

It is also useful to keep in mind that it is probably better to capture too much information than too little – if you captured too much data you can filter it farther on the display and analysis level, while if you missed important information a new recording will be needed to retrieve it.

Here is what the capture configuration screen looks like:


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Wireshark: Capture Options
Wireless Settings
Remote Settings
Browse...
Help
Start
Cancel
Files
Online
Website
Visit the project's website
User's Guide
The User's Guide (local version, if installed)
Security
Work with Wireshark as securely as possible
Network Media
Specific information for capturing on: Ethernet, WLAN, ...
Step by step to a successful capture setup
Ready to load or capture
No Packets
Profile: Default
Configuring port filters for capture

Capture
Interface: Local — MS Tunnel Interface Driver: \Device\NPF_{4C8B7937-D24C-43
IP address: unknown
Link-layer header type: Ethernet
[✓] Capture packets in promiscuous mode
[ ] Capture packets in pcap-ng format (experimental)
[ ] Limit each packet to 1 bytes
Buffer size: 1 megabyte(s)
Capture Filter: tcp port 1521

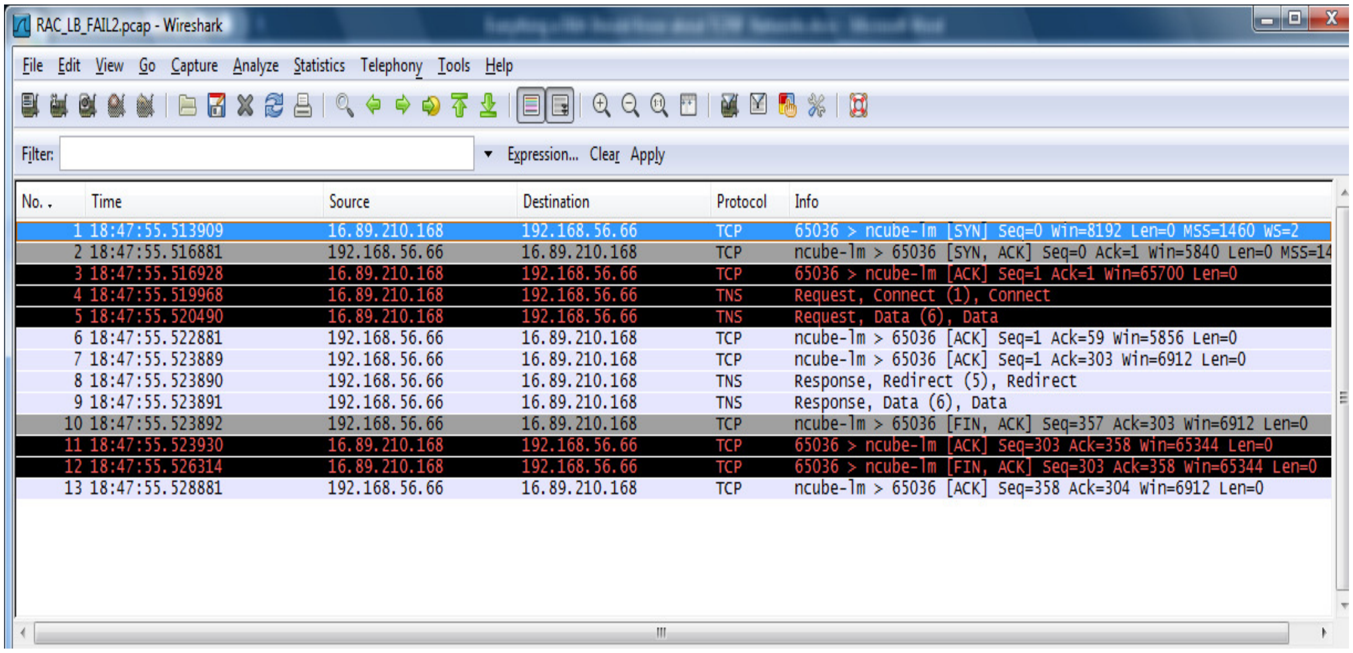
Capture File(s)
File:
[ ] Use multiple files
[✓] Next file every 1 megabyte(s)
[ ] Next file every 1 minute(s)
[✓] Ring buffer with 2 files
[ ] Stop capture after 1 file(s)

Stop Capture ...
[ ] ... after 1 packet(s)
[ ] ... after 1 megabyte(s)
[ ] ... after 1 minute(s)

Display Options
[✓] Update list of packets in real time
[✓] Automatic scrolling in live capture
[✓] Hide capture info dialog

Name Resolution
[✓] Enable MAC name resolution
[ ] Enable network name resolution
[✓] Enable transport name resolution

Here is what the results of the capture looked like:


RAC_LB_FAIL2.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression... Clear Apply

```
No. .  Time              Source          Destination     Protocol Info
     1 18:47:55.513909   16.89.210.168   192.168.56.66   TCP      65036 > ncube-lm [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
     2 18:47:55.516881   192.168.56.66   16.89.210.168   TCP      ncube-lm > 65036 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=14
     3 18:47:55.516928   16.89.210.168   192.168.56.66   TCP      65036 > ncube-lm [ACK] Seq=1 Ack=1 Win=65700 Len=0
     4 18:47:55.519968   16.89.210.168   192.168.56.66   TNS      Request, Connect (1), Connect
     5 18:47:55.520490   16.89.210.168   192.168.56.66   TNS      Request, Data (6), Data
     6 18:47:55.522881   192.168.56.66   16.89.210.168   TCP      ncube-lm > 65036 [ACK] Seq=1 Ack=59 Win=5856 Len=0
     7 18:47:55.523889   192.168.56.66   16.89.210.168   TCP      ncube-lm > 65036 [ACK] Seq=1 Ack=303 Win=6912 Len=0
     8 18:47:55.523890   192.168.56.66   16.89.210.168   TNS      Response, Redirect (5), Redirect
     9 18:47:55.523891   192.168.56.66   16.89.210.168   TNS      Response, Data (6), Data
    10 18:47:55.523892   192.168.56.66   16.89.210.168   TCP      ncube-lm > 65036 [FIN, ACK] Seq=357 Ack=303 Win=6912 Len=0
    11 18:47:55.523930   16.89.210.168   192.168.56.66   TCP      65036 > ncube-lm [ACK] Seq=303 Ack=358 Win=65344 Len=0
    12 18:47:55.526314   16.89.210.168   192.168.56.66   TCP      65036 > ncube-lm [FIN, ACK] Seq=303 Ack=358 Win=65344 Len=0
    13 18:47:55.528881   192.168.56.66   16.89.210.168   TCP      ncube-lm > 65036 [ACK] Seq=358 Ack=304 Win=6912 Len=0
```

As you can see in the capture file, the original connection to the listener has succeeded. The listener then replied with a redirect to the specific node that it wanted the client to connect to (line 8 contains the redirect and the following line contains the redirect instructions). The client attempted to follow the redirect, but the server name included in the redirect information was not a name known to the client and therefore the new connection failed, resulting in an error.

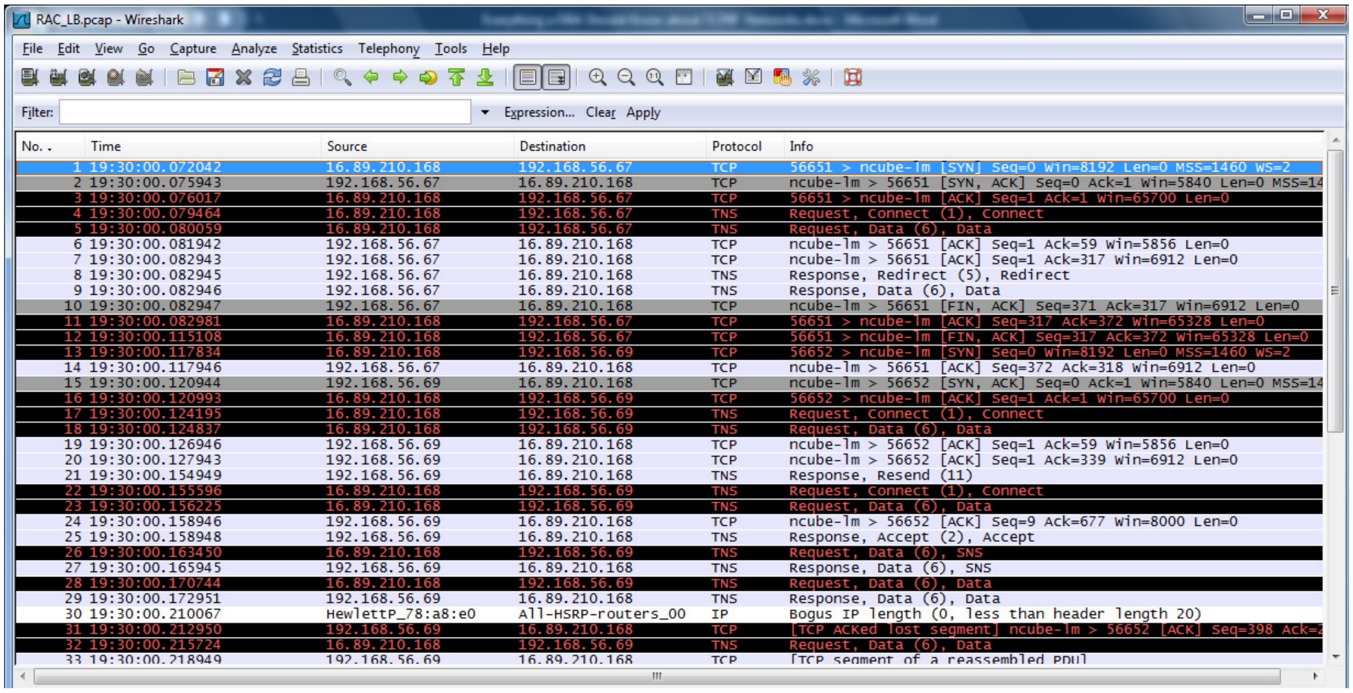
When capturing multiple connections in the same file, it is possible to "zoom" in to one of the connections by selecting one of the packets, right clicking and selecting "follow TCP stream". This has the effect of filtering just one connection attempt (uniquely identified by source host and port and target host and port) and it will also display the plain text communication in a separate screen, allowing to view the exact redirect instructions and later on the exact queries sent from the client and the replies sent from the server.

Here is what the redirect instructions contained:

```
(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=ppmdb03n2-
vip.infra.mms)(PORT=1521))).(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=ppmdb03n1)(PORT=1521))(LOAD_BAL
ANCE=yes)(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=ITG03)(CID=(PROGRAM=C:\app\shapirac\product\11
.1.0\db_1\BIN\sqlplus.exe)(HOST=CSHAPIRA3)(USER=shapirac))(INSTANCE_NAME=ITG032)))
```

Following the redirect, the client attempted to connect to host ppmdb03n2-vip.infra.mms, which could not be resolved.

We can compare this to a connection that did resolve successfully:


RAC_LB.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression... Clear Apply

```
No. .  Time                Source              Destination          Protocol  Info
  1 19:30:00.072042   16.89.210.168       192.168.56.67        TCP       56651 > ncube-lm [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
  2 19:30:00.075943   192.168.56.67       16.89.210.168        TCP       ncube-lm > 56651 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=14
  3 19:30:00.076017   16.89.210.168       192.168.56.67        TCP       56651 > ncube-lm [ACK] Seq=1 Ack=1 Win=65700 Len=0
  4 19:30:00.079464   16.89.210.168       192.168.56.67        TNS       Request, Connect (1), Connect
  5 19:30:00.080059   16.89.210.168       192.168.56.67        TNS       Request, Data (6), Data
  6 19:30:00.081942   192.168.56.67       16.89.210.168        TCP       ncube-lm > 56651 [ACK] Seq=1 Ack=59 Win=5856 Len=0
  7 19:30:00.082943   192.168.56.67       16.89.210.168        TCP       ncube-lm > 56651 [ACK] Seq=1 Ack=317 Win=6912 Len=0
  8 19:30:00.082945   192.168.56.67       16.89.210.168        TNS       Response, Redirect (5), Redirect
  9 19:30:00.082946   192.168.56.67       16.89.210.168        TNS       Response, Data (6), Data
 10 19:30:00.082947   192.168.56.67       16.89.210.168        TCP       ncube-lm > 56651 [FIN, ACK] Seq=371 Ack=317 Win=6912 Len=0
 11 19:30:00.082981   16.89.210.168       192.168.56.67        TCP       56651 > ncube-lm [ACK] Seq=317 Ack=372 Win=65328 Len=0
 12 19:30:00.115108   16.89.210.168       192.168.56.67        TCP       56651 > ncube-lm [FIN, ACK] Seq=317 Ack=372 Win=65328 Len=0
 13 19:30:00.117834   16.89.210.168       192.168.56.69        TCP       56652 > ncube-lm [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
 14 19:30:00.117946   192.168.56.67       16.89.210.168        TCP       ncube-lm > 56651 [ACK] Seq=372 Ack=318 Win=6912 Len=0
 15 19:30:00.120944   192.168.56.69       16.89.210.168        TCP       ncube-lm > 56652 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=14
 16 19:30:00.120993   16.89.210.168       192.168.56.69        TCP       56652 > ncube-lm [ACK] Seq=1 Ack=1 Win=65700 Len=0
 17 19:30:00.124195   16.89.210.168       192.168.56.69        TNS       Request, Connect (1), Connect
 18 19:30:00.124837   16.89.210.168       192.168.56.69        TNS       Request, Data (6), Data
 19 19:30:00.126946   192.168.56.69       16.89.210.168        TCP       ncube-lm > 56652 [ACK] Seq=1 Ack=59 Win=5856 Len=0
 20 19:30:00.127943   192.168.56.69       16.89.210.168        TCP       ncube-lm > 56652 [ACK] Seq=1 Ack=339 Win=6912 Len=0
 21 19:30:00.154949   192.168.56.69       16.89.210.168        TNS       Response, Resend (11)
 22 19:30:00.155596   16.89.210.168       192.168.56.69        TNS       Request, Connect (1), Connect
 23 19:30:00.156225   16.89.210.168       192.168.56.69        TNS       Request, Data (6), Data
 24 19:30:00.158946   192.168.56.69       16.89.210.168        TCP       ncube-lm > 56652 [ACK] Seq=9 Ack=677 Win=8000 Len=0
 25 19:30:00.158948   192.168.56.69       16.89.210.168        TNS       Response, Accept (2), Accept
 26 19:30:00.163450   16.89.210.168       192.168.56.69        TNS       Request, Data (6), SNS
 27 19:30:00.165945   192.168.56.69       16.89.210.168        TNS       Response, Data (6), SNS
 28 19:30:00.170744   16.89.210.168       192.168.56.69        TNS       Request, Data (6), Data
 29 19:30:00.172951   192.168.56.69       16.89.210.168        TNS       Response, Data (6), Data
 30 19:30:00.210067   HewlettP_78:a8:e0   All-HSRP-routers_00  IP        Bogus IP length (0, less than header length 20)
 31 19:30:00.212950   192.168.56.69       16.89.210.168        TCP       [TCP ACKed lost segment] ncube-lm > 56652 [ACK] Seq=398 Ack=
 32 19:30:00.215724   16.89.210.168       192.168.56.69        TNS       Request, Data (6), Data
 33 19:30:00.218949   192.168.56.69       16.89.210.168        TCP       [TCP segment of a reassembled PDU]
```

Here we can see that immediately following the redirect (lines 8 and 9) the old connection to server 192.168.56.67 was closed and a new connection was opened to 192.168.56.69 – the second node of the RAC., showing that the redirect was successful.

Solution:

The server name that the listener sends to the client in the redirect command is taken from the LOCAL_LISTENER parameter. The LOCAL_LISTENER parameter for each node contains a name of an entry in TNSNAMES.ORA and this entry is the details that will be sent to the client when it is redirected to that specific node.

One possibility is that the LOCAL_LISTENER is not configured or is configured with the wrong TNSNAME entry – one that contains the local name of the host instead of a name recognized by the DNS. In this case, the LOCAL_LISTENER parameter should be fixed [2].

Another possibility is that the LOCAL_LISTENER is configured correctly but the client is still not aware of the hostname included in the redirect instructions. This could be a problem with the DNS configuration on the client side. This can be fixed by adding the correct name and address to the local host file on the client, or asking the network admin for a DNS solution.

4. Users complain that a long running batch job never finishes running

Symptoms

Users complained that a long running batch job, one that normally takes few hours to complete now never finishes running. From SQL Plus it looks like the last function that was called continues running forever, never returning a reply.

If the code is instrumented to write its process to a log table we can see the last call that was finished successfully, we can see that the next call never ran and we can also see that the exception code never ran – ruling out any errors that could cause the job to stop in the middle.

When we try to trace the process from the server side, we can see that the code ran successfully until the end of the function.

From the client side it looks like the server never returns with a reply. From the server side it looks like everything ran successfully. What is going on?

Debugging

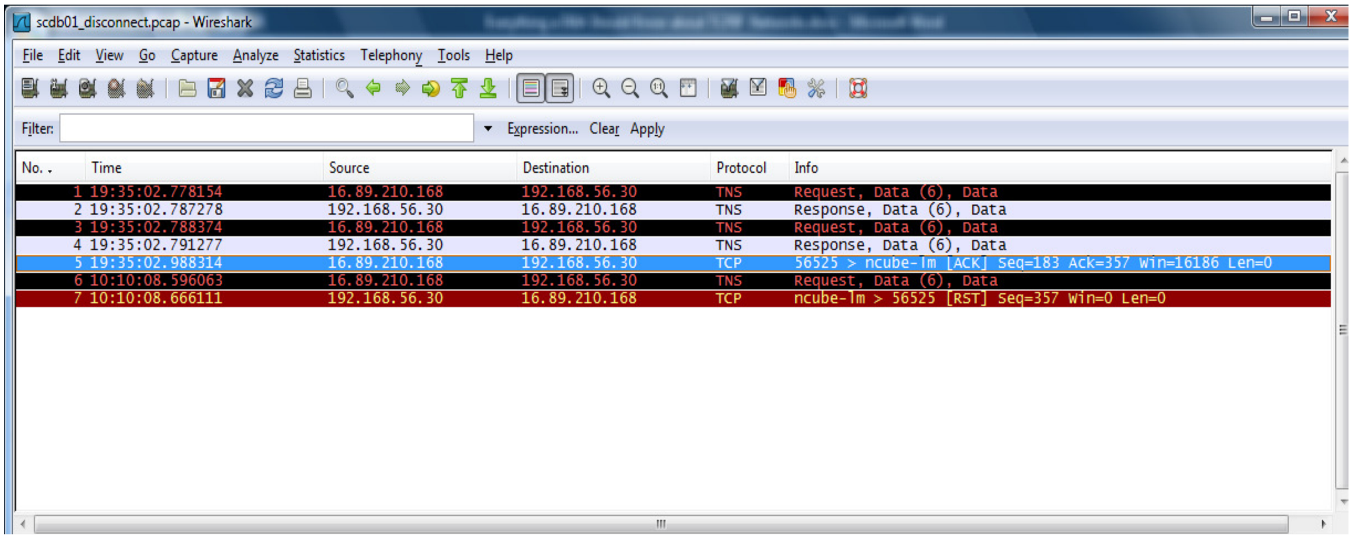
When the question is "What is really going on?" in a communication related context, the first answer should always be "Use Wireshark to see exactly what is going on". We can build many theories of what could happen, but Wireshark can shows us what happens packet by packet – eliminating the need for guess work.

In this case we can start capturing traffic from the client before starting the job. We can also simultaneously capture traffic on the server side using TCPDump, so we can compare what is going on between the client and the server.

```
tcpdump \( host 16.89.210.168 \) and tcp port 1521 -s0 -w server_wait_disc.cap
```

This runs tcp dump, capturing traffic to a specific client (my machine in this case) or a specific port (1521). The flag –s0 is very important – it means tcpdump should capture the entire packet instead of cutting it them off after a specific size was passed. The last flag, -w, tells tcpdump in which file to write the captured packets, by defaults they are sent to stdout.

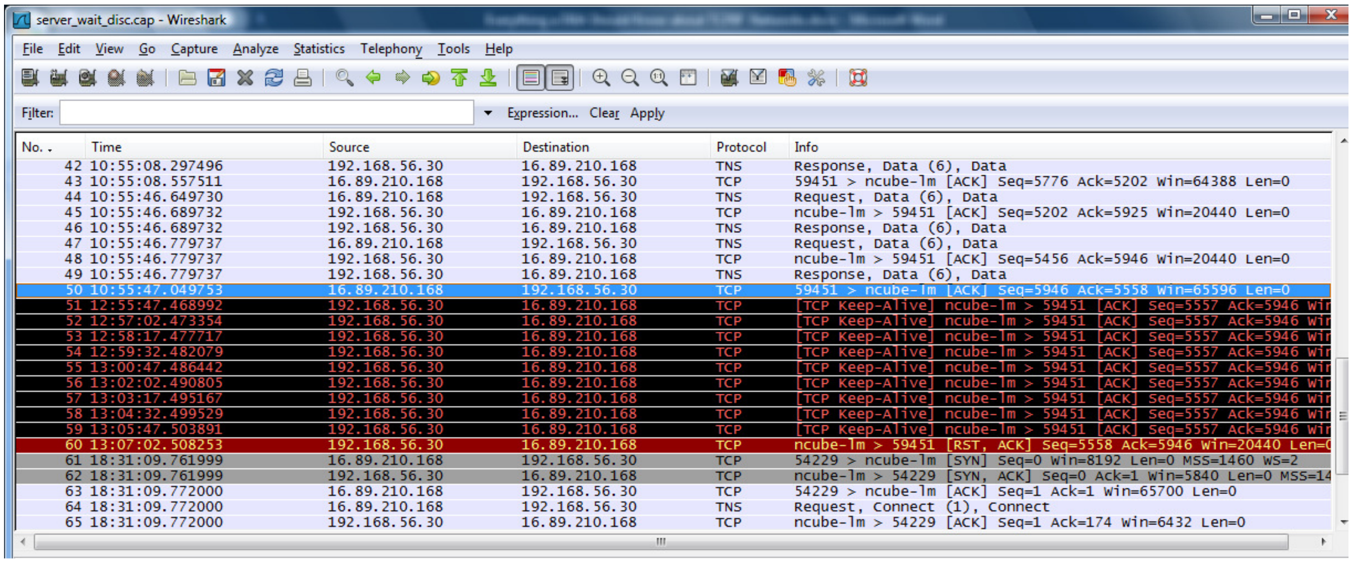
We'll start by looking at the client side of things:


scdb01_disconnect.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression... Clear Apply

```
No. Time             Source          Destination     Protocol Info
  1 19:35:02.778154  16.89.210.168   192.168.56.30   TNS      Request, Data (6), Data
  2 19:35:02.787278  192.168.56.30   16.89.210.168   TNS      Response, Data (6), Data
  3 19:35:02.788374  16.89.210.168   192.168.56.30   TNS      Request, Data (6), Data
  4 19:35:02.791277  192.168.56.30   16.89.210.168   TNS      Response, Data (6), Data
  5 19:35:02.988314  16.89.210.168   192.168.56.30   TCP      56525 > ncube-lm [ACK] Seq=183 Ack=357 Win=16186 Len=0
  6 10:10:08.596063  16.89.210.168   192.168.56.30   TNS      Request, Data (6), Data
  7 10:10:08.666111  192.168.56.30   16.89.210.168   TCP      ncube-lm > 56525 [RST] Seq=357 Win=0 Len=0
```

From the client side, things are very simple – a request what sent to the server at 19:35, the server replied and the client acknowledged the reply. Few hours later when we attempted to send another request to the server, we received a reset. SQLPLUS will show:

```
ERROR at line 1:
ORA-03135: connection lost contact
Process ID: 0
Session ID: 196 Serial number: 200
```


server_wait_disc.cap - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression... Clear Apply

```
No. .  Time                Source           Destination      Protocol  Info
    42 10:55:08.297496     192.168.56.30    16.89.210.168    TNS       Response, Data (6), Data
    43 10:55:08.557511     16.89.210.168    192.168.56.30    TCP       59451 > ncube-lm [ACK] Seq=5776 Ack=5202 Win=64388 Len=0
    44 10:55:46.649730     16.89.210.168    192.168.56.30    TNS       Request, Data (6), Data
    45 10:55:46.689732     192.168.56.30    16.89.210.168    TCP       ncube-lm > 59451 [ACK] Seq=5202 Ack=5925 Win=20440 Len=0
    46 10:55:46.689732     192.168.56.30    16.89.210.168    TNS       Response, Data (6), Data
    47 10:55:46.779737     16.89.210.168    192.168.56.30    TNS       Request, Data (6), Data
    48 10:55:46.779737     192.168.56.30    16.89.210.168    TCP       ncube-lm > 59451 [ACK] Seq=5456 Ack=5946 Win=20440 Len=0
    49 10:55:46.779737     192.168.56.30    16.89.210.168    TNS       Response, Data (6), Data
    50 10:55:47.049753     16.89.210.168    192.168.56.30    TCP       59451 > ncube-lm [ACK] Seq=5946 Ack=5558 Win=65596 Len=0
    51 12:55:47.468992     192.168.56.30    16.89.210.168    TCP       [TCP Keep-Alive] ncube-lm > 59451 [ACK] Seq=5557 Ack=5946 Wi
    52 12:57:02.473354     192.168.56.30    16.89.210.168    TCP       [TCP Keep-Alive] ncube-lm > 59451 [ACK] Seq=5557 Ack=5946 Wi
    53 12:58:17.477717     192.168.56.30    16.89.210.168    TCP       [TCP Keep-Alive] ncube-lm > 59451 [ACK] Seq=5557 Ack=5946 Wi
    54 12:59:32.482079     192.168.56.30    16.89.210.168    TCP       [TCP Keep-Alive] ncube-lm > 59451 [ACK] Seq=5557 Ack=5946 Wi
    55 13:00:47.486442     192.168.56.30    16.89.210.168    TCP       [TCP Keep-Alive] ncube-lm > 59451 [ACK] Seq=5557 Ack=5946 Wi
    56 13:02:02.490805     192.168.56.30    16.89.210.168    TCP       [TCP Keep-Alive] ncube-lm > 59451 [ACK] Seq=5557 Ack=5946 Wi
    57 13:03:17.495167     192.168.56.30    16.89.210.168    TCP       [TCP Keep-Alive] ncube-lm > 59451 [ACK] Seq=5557 Ack=5946 Wi
    58 13:04:32.499529     192.168.56.30    16.89.210.168    TCP       [TCP Keep-Alive] ncube-lm > 59451 [ACK] Seq=5557 Ack=5946 Wi
    59 13:05:47.503891     192.168.56.30    16.89.210.168    TCP       [TCP Keep-Alive] ncube-lm > 59451 [ACK] Seq=5557 Ack=5946 Wi
    60 13:07:02.508253     192.168.56.30    16.89.210.168    TCP       ncube-lm > 59451 [RST, ACK] Seq=5558 Ack=5946 Win=20440 Len=
    61 18:31:09.761999     16.89.210.168    192.168.56.30    TCP       54229 > ncube-lm [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
    62 18:31:09.761999     192.168.56.30    16.89.210.168    TCP       ncube-lm > 54229 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=14
    63 18:31:09.772000     16.89.210.168    192.168.56.30    TCP       54229 > ncube-lm [ACK] Seq=1 Ack=1 Win=65700 Len=0
    64 18:31:09.772000     16.89.210.168    192.168.56.30    TNS       Request, Connect (1), Connect
    65 18:31:09.772000     192.168.56.30    16.89.210.168    TCP       ncube-lm > 54229 [ACK] Seq=1 Ack=174 Win=6432 Len=0
```

From the server side, things look a bit different. It received the reply and started working on it. 2 hours later, the TCP Socket on the server side started sending keep-alive packets to the client. Packets that were never received by the client! After received no reply for multiple keep-alive requests, the server OS decided that the client is likely dead and closed the connection. When the database finished running the function, the connection was already closed. The database assumed that the client disconnected while it was busy, rolled back the last transaction and closed the session.

Solution

The likely root cause for the problem is that there is a firewall between the client and the server and that this firewall has a timeout on connections. After a long delay passes from the last client request, the firewall will no longer pass messages from the server and this way the TCP Keep-alive messages are lost and the connection is closed prematurely.

One solution would be to use the two captures we've gathered to explain to the network admin our problem and ask for longer firewall timeouts. Any timeout of longer than 2 hours will allow the TCP Keepalives to go through and maintain the connection.

Another solution will be to modify kernel parameter net.ipv4.tcp_keepalive_time to a value lower than the firewall timeout values. This should give the TCP keepalive a chance to keep the connection alive.

A third solution can be to use SQLNET keep-alive parameter SQLNET.EXPIRE_TIME that allows Oracle to send its own keep-alive probes from the server to the client, normally sent every 10 minutes. These probes will also keep the connection a live and not allow the firewall to disconnect it.

Beyond the scope of this specific story, keep in mind that network traffic goes through a large number of network devices – switches, routers, NATs and firewalls; each of them can be mis-configured and cause problems. Capturing traffic from both ends of the connection is an excellent way to open a discussion with the network admin about network problems you are seeing – even if you cannot know exactly which device is causing the problem and how, having a proof of an issue goes a long way.

5. Application needs to read large amounts of data off the database. Can we reduce the time spent on network delays?

Symptoms:

For some reason, our application needs to read very large amounts of data from the database - maybe a table with many rows, or rows with very large average row size. We tuned everything we could on the application side and the database side, but it looks like a lot of time is being spent moving the data over the network. Is there anything that can be done?

Debugging:

We'll start by assuming that we are talking about a LAN and that the latency between the client and server is very low. If you are talking about an application that is physically remote from the database and there is a lot of latency involved, I'll offer some advice in the next section. You can check the latency by running a simple tnsping from the application to the database – I'm talking about a situation where it takes 20ms to reply.

The first to do when trying to speed up a query is to look at wait events – where Oracle says that time is being spent. Since we are looking at the network, we need to make sure we understand the network related wait events:

The most common events will be:

SQL*Net message to client:

This event measures the time it took Oracle to put its data on the TCP stack of the server. Normally this takes milliseconds since its all local activity; nothing was done on the network yet. If you do see a significant amount of time spent here, it means that the TCP stack of the OS refuses to receive more data from Oracle – possibly because it did not yet receive ACKs from the client that the data was received. This can indicate a serious network issue and it is time to take out Wireshark and have a serious discussion with your network admin.

SQL*Net message from client

If the network is a bottleneck in the performance of a report, most of the time will be spent here. The only problem here is that this includes not only time spent in the network, but also time spent by the application before it responded to the DB. This is why DBAs traditionally know to ignore this event – because we don't want to waste time debugging application delays. Before you assume that this event indicates a network delay, you want to make sure that the application responded in a timely manner. Wireshark capture on the application side can be useful here.

SQL*Net more data to client

This indicates that Oracle attempted to send more data to the client in one call than fits in one buffer. This event rarely takes a significant amount of time.

There are also a number of statistics that are worth looking at:

Bytes sent via SQL*Net to client

This is the total amount of data sent by the server to the client. One of the first things to do when you suspect a network delay in sending data to the application is tell your network admin "Sending 1500K of data took 30 seconds. Does this make sense for our network?" and after a while you'll know the rough numbers yourself. There is no point in tuning performance for something that already shows perfectly reasonable performance.

SQL*Net roundtrips to/from client

This is the number of times the client had to go back and request more data from the server. This is one of our prime tuning candidates.

Now that we know where time is spent, it is time to see what we can tune.

Solutions

The first and in my opinion that most useful part to tune, is the number of round-trips from the client to the server. The client normally requests a specific number of rows from the server; it processes them and goes back for more. If you need to read 200,000 rows from the server, asking for 10 rows (common JDBC default), processing them and asking for ten more will be terribly inefficient. If you know you'll need to read a large number of rows, configure your application to read in larger batches. The parameter you need to tune is "arraysize" for sqlplus and "setFetchSize" for JDBC.

How high do you want to set the fetch size? I did not find any best practices for this. It is always preferred that you test several sizes and see what works best for your application. I recommend setting it to get as many rows as possible considering the amount of memory you have to spare for the data. I can't see any downside for fetching too many rows at once if you have memory to store them.

The second and somewhat less likely place to tune is the SDU size. SDU is the size of the buffer Oracle uses to store data before sending it through the network. The default size is 8K at 11gR2 and you can set it as high as 32K. If your SDU is 2K and you are fetching a 4k row, Oracle will fill the buffer and start a new one. You'll see "SQL*NET More Data to Client" event accounting for the second buffer.

There is a lot of advice going around for how to tune the SDU size, which is pretty amazing considering that no one showed significant performance improvements from tuning it.

Here is some of the advice:

- "If sufficient memory is available, using the maximum value for the SDU will minimize the number of system calls and overhead for Oracle Net Services." [3]
  This advice from Oracle seems to indicate that on a server with plenty of memory setting the default SDU to 32K will cause no harm and may do some good. Note that you will need extra 24K of RAM per client connection if you choose to do this. This advice was also repeated by Tanel Poder in his blog. [4]
- "Ideally, SDU should not surpass the size of the maximum transmission unit (MTU)." [5]
  Note that MTU on most networks is 1500bytes (you can check yours with ifconfig), which means that each packet will contain 1500bytes of data, actually a bit less if you consider packet headers and such. Even on older Oracle versions SDU size was 2k which is already larger than MTU size, which means that in any case once Oracle fills the buffer and sends it, it will be split into smaller 1500byte packets by the network layer. I believe that this advice is a result of confusion between the overheads caused by Oracle creating new buffers and the overhead caused by TCP network creating new packets.
- There is a somewhat old but very scientific and very detailed article by Sun which explains how to size the SDU to minimize the amount of packet splitting that will be done by the OS. [6]
  Since we are discussing a low latency network, I would not worry about optimizing for the number of packets sent – it will not have any effect on performance. If you are worried about number of packets sent – setting the SDU size to be a multiple of the MTU size minus the packet overheads will be a good idea.
  Note that the Sun article did report significant performance gains from tuning the SDU, but they only tested very small SDU sizes. There is no indication that changing the default from 8k to 32k is helpful.
- The best tuning advice was given by Tom Kyte: "What you have to ask is "is the amount of data divided by 24 seconds giving me a reasonable throughput rate -- what is the amount of data transmitted per second there". If you are getting what seems to be a reasonable rate already, you are done. If not, then we can look at a adjusting network settings." [7]

To conclude the review of advice – There is very little proof that fiddling with the SDU size brings any performance benefit. If you want to check whether changing the SDU can help your application then you can set it to the maximum and minimize the number of context switches that Oracle will encounter or you can set it to a multiple of the MTU and minimize the number of packets you will have to send.

6. Redo logs need to be shipped to a remote disaster recovery site. Can we anticipate in advance how long the data shipping will take? Are there ways to make it faster?

Symptoms:

In this case, we are looking at a situation where network latency is significant. Perhaps something like 500ms – 50 times the latency of the local network we were previously discussing. Note that when transferring data over the WAN we expect not just high latency but also more errors – missing and out of order packets can be expected to occur with some regularity.

We'll start by calculating the maximum rate in which data can be sent over our link. This requires asking your network manager for the expected bandwidth for the connection. You want a number in Mbits per second. You may have to explain to the network manager that you are interested in the theoretical capacity, that he won't be held to that number and that you know that in reality you will see lower throughput. I'll explain why in a minute.

Suppose that the network admin confessed to having an OC3 line to the DR site, with the theoretical capacity of 155Mb/s. This means that if our server generates 70Gbytes (155*60*60/8) of redo every hour than we our out of the game – there is no way we can get our line to carry our redo on time to the DR site. Knowing some queuing theory we know that we don't want to go over 75% utilization – because then we will not have any way to recover from backlogs that may occur if for one hour we generate more redo than expected, so on an OC3 line we probably want to limit ourselves to around 50Gbytes per hour.

What if we have less than that, can we be sure we'll make it? Well, that's easy – try copying an hour worth of redo to the remote site and check how fast this happens. Note that if you are planning to use DataGuard, use Oracle file transfer for the test – other utilities such as SCP change lots of network parameters and you may get unrealistic result.

If you are happy with the results (i.e. you finish in less than 45 minutes) than you are doing great, go tune something else. But what if you only generate 2Gbytes of data every hour and it takes the entire hour to copy them over a line with a much larger theoretical capacity?

Debugging:

Start by talking to your network administrator and ask him to measure line utilization while you are copying the data. It is obvious that you are not getting full line utilization, so either someone else is using the line and not letting you get full bandwidth or the line is not fully utilized.

If someone else is using the line then you can either get the network admin to stop him, if copying the redo logs fast is critical enough, or you may have to resign yourself to the realities of a shared line.

But it is quite likely that the network admin will say that the line is not fully utilized and that you are only using 30% of its capacity.

The mission now is to get more utilization out of the line.

Solution:

The key to understanding the reason we are not using the entire line is to understand how TCP thinks.

TCP attempts to create a reliable network line where the network is in fact unreliable. To do this, the server tracks the data it sends and it expects the client to acknowledge the data it received. If the client does not acknowledge the data, the server will send it again.

As we discussed in the previous section, a packet of data contains 1500bytes. Since on our WAN we have a round trip time of 500ms, if we waited for the client to acknowledge every packet before we send the next, we could only send 3k every second. We are trying to send 155Mb every second.

It should be obvious from this small example that the way to get more utilization out of the line is to send more packets before we receive acknowledgement for them.

The amount of data we need to have "in flight" in order to fully utilize the line is also known as "Bandwidth Delay Product" – the number of Mbs per second times the network delay. In our case:

155Mb/s *500ms = 155000 bits/s * 0.5s = 77500 bits = 9.6Mbytes

So we need 9.6Mbytes of data in the air at any given moment in order to fully utilize the line. The way to achieve this is to configure Linux to allow 9.6Mbytes of data to go without ACKs before it stops sending more data.

The way to configure that is using the kernel parameters:

net.core.wmem_max and *net.core.wmem_default* will control the maximum and default number of bytes the kernel will allow sending without ACKs. You want to set this parameter to 9.6MB at the server side.

net.core.rmem_max and *net.core.rmem_default* controls the maximum and default number of bytes the client will tell the server it can receive before it will start sending an ACK – you need to set this on the receiving side.

When installing Oracle, you normally configure these parameters to 262K. Note that when you increase this number it will increase the amount of memory each connection will take accordingly – Linux needs space to keep all the data that was sent but not acknowledged, in case it needs to retransmit it.

Oracle discusses tuning of these network parameters in the documentation [8], in SQLNET these parameters are called SEND_BUF_SIZE and REC_BUF_SIZE. Note though that tuning these parameters in SQLNET without increasing the OS maximum in the kernel is not very useful.

Also important to note that increasing the buffer size does not guarantee that a buffer of this size will really be used! Ultimately TCP decides how much "in flight" data is safe according to its own algorithms, and it is quite conservative.

You can expect each connection to start with a small window of "in flight" data, this window will grow as more data is acknowledged and the OS decides that the line is reliable. If packets are suddenly lost and retransmits are necessary then the window size will drop and will start growing slowly again. The rule is that window size grows linearly but it drops exponentially when there are problems.

Keep in mind that if you are at the point where you tune TCP windows to get more throughput from a WAN line, you may also consider purchasing a WAN accelerator – a network device that uses a number of mechanisms including caching, compression and fake ACKs to make large transfers over a WAN more efficient.

7. Conclusions

DBAs often have to work outside their area of expertise in order to do their work. While most organizations employ a full time network admin whose job is to solve network problems. In reality, network administrators are just as busy as DBAs and without conclusive proof that there is a problem and that the problem is really network related, the network administrator may be reluctant to get involved.

In this paper we went over four common network problems that DBAs can encounter and how with one simple tool, few tunable parameters and a little knowledge of networks the DBA can make significant discoveries and improve communication with the network admin.

You hopefully learned:

- How Wireshark can be used to definitively troubleshoot a connectivity issue avoiding possible guesses and misunderstandings.
- How capturing network traffic from both sides of a connection can be useful in troubleshooting network problems caused by third party devices.
- Common ways of making data transfers more efficient over the LAN and which ways do or do not work.
- How to make better use of WAN lines and in the process you also learned a bit about TCP sliding windows mechanism.

8. References

1. Metalink article: Troubleshooting ORA-12545 / TNS-12545 Connect failed because target host or object does not exist [ID 553328.1]
2. http://www.ardentperf.com/2007/04/02/local_listener-and-ora-12545/
3. Oracle® Database Net Services Administrator's Guide 11g Release 2 (11.2), Chapter 14 – Optimizing performance. http://download.oracle.com/docs/cd/E11882_01/network.112/e10836/performance.htm#i1006332
4. http://blog.tanelpoder.com/2008/02/10/sqlnet-message-to-client-vs-sqlnet-more-data-to-client/
5. http://www.dba-oracle.com/art_builder_tns.htm
6. ORACLE® Middleware Layer Net8™ Performance Tuning Utilizing Underlying Network Protocol, Gamini Bulumulle, Sun Professional Services. October 2002.
7. http://asktom.oracle.com/pls/asktom/f?p=100:11:0::::P11_QUESTION_ID:951335700013
8. Oracle® Database Net Services Administrator's Guide 11g Release 2 (11.2), Chapter 14 – Optimizing performance. http://download.oracle.com/docs/cd/E11882_01/network.112/e10836/performance.htm#i1007572