The ten commandments of TCP/IP performance are a distillation of hard-won experience. Monitoring and tuning TCP networks on the mainframe is complex for the basic reason that each network is a mixture of many applications and pieces of hardware. Each connection contains layers of protocols and subprotocols which must be decoded to make sense of the traffic patterns. Making sense of it all is the first step to tuning and improving performance.

For example, if you are trying to find a problem with an IP printer, you may need to understand the IP protocol, the TCP protocol, the LPR/LPD protocol, and then the hardware implementation of the printer. The following ten commandments of TCP/IP performance diagnostics have served us well to tune the TCP stack and to find performance problems.

1. Thou shalt monitor thy application backlog queue
2. Thou shalt not kill thy network by many short connections
3. Thou shalt drop unused connections
4. Thou shalt honor thy TCP duplicate ACKs and thy retransmissions
5. Thou shalt relate thy TCP reset to the cause.
6. Thou shalt not fail in watching thy TCP attempt fails
7. Thou shalt delve deeply into UDP no ports errors
8. Thou shalt address the reason for your IP address errors
9. Thou shalt not convert thy applications directly from multi-dropped SDLC
10. Thou shalt not use two packets when one will do
Commandment 1: Thou shalt monitor thy application backlog queue

Looking at the application backlog queue is ‘low-hanging fruit’. It is one of the easiest things to do, it can help users a great deal and it can solve problems which may appear to be quite difficult. How it works is that the TCP application may have a maximum number of connections which can be active at the same time, let’s say 5. Then, there is another maximum number of connections defined, let’s say 10, which are waiting for those 5 to finish. This is the backlog queue. When the 16th user tries to make a connection, he can’t. The connection is dropped.

So, you have the following:
- Current Backlog: The current number of connections in backlog.
- Maximum in Backlog: The maximum number of connections allowed in backlog at one time.
- Exceed Backlog: The total number of connections dropped by the listener due to backlog exceeded.

Let’s look at a case study where we worked with an installation using an accounting application. Sometimes the user received a ‘Connection Refused’ message when they tried to connect to the application, other times, the session initiation would just hang.

Technical support was stumped! The users were angry and the problem has gotten escalated to quite a high level. Then we were called in. When we looked at the application backlog queues, we see that the backlog queues are being exceeded and connections are dropped. You can easily see the backlog queues in the Netstat All display. See Figure 2.

![Figure 2: Netstat All Showing Connections Dropped](image)

When the user connection was in the backlog queue, the session initiation appeared to ‘hang’. When the user connection was dropped because the queue was exceeded, the ‘Connection Refused’ message was received. The real problem is that the application took so long to process and did not complete connections in a timely manner.
Commandment 2: Thou shalt not kill thy network by many short connections

The way TCP works is by creating a virtual circuit between the two ends of the connection: the remote host and the local host. The remote and local hosts are also known as ‘client’ and ‘server’ or ‘local address’ and ‘foreign address’. When the two ends need to talk to each other using the TCP protocol, a connection is established which lasts for some period of time. In fact, the connection lasts for the period of time bounded by the Open and Close. To allow us to talk about this connection, it is called a ‘virtual circuit’. All the TCP protocol functions take place in the context of this virtual circuit.

During the open sequence, TCP packets flow back and forth with various bits of the header turned on. The header is the first 20 or so bytes of the TCP packet which tell various pieces of control information. Of course, IP adds a header as well. In the open sequence, first, a TCP packet is sent from one side. Then, in response, the other side allocates buffers and other resources. This is called the SYN -- SYN/ACK sequence or the TCP 3-way handshake. At the end of the handshake, if it is properly concluded, then the connection or virtual circuit is ready for data transmission. During the close sequence, a number of packets flow back and forth as well.

Now, you can see that if you have many short connections, you are adding overhead for the open and close handshakes to the data transmissions. You can tune your TCP usage by checking the applications which have many connections from the same address pairs in a short amount of time. Using an option to have persistent connections may decrease the CPU usage and network traffic.

Commandment 3: Thou shalt drop unused connections

This seems like a ‘no-brainer’. If you are finished using a TCP connection, then the application should close it! Just as I tell my daughter, when you are done with finding things in the kitchen cupboard, close the drawers! What is so hard about this?

Actually, quite a bit. Many times applications are developed using code-generators which ‘shield’ the programmer from the perceived intricacies of raw sockets code. So, it can be quite difficult to tell in the program if the socket is actually being closed or not.

We have seen applications, such as LDAP servers, run out of sockets and abend because connections were not closed. There may be parameters for the application to time out idle connections or to do keep-alive. The keep-alive probe will drop unused connections. In our experience, dropping unused connections and eliminating errors on the TCP network has saved CPU time used by the TCP stack on the mainframe.
**Commandment 4: Thou shalt honor thy TCP duplicate ACKs and thy TCP retransmissions**

What is a duplicate ACK? If a packet is lost, then TCP will send the same acknowledgment again. Generally, when TCP gets 3 duplicate acknowledgments, then it will retransmit the packet. Look at Figure 3 for a diagram of a packet loss scenario. You will notice that Segment 2 was lost, duplicate acks with the ACK number 100 were sent, and then finally Segment 2 was retransmitted.

---

![TCP Duplicate ACKs](image)

**Figure 3: TCP Duplicate Acknowledgments**

You may find in monitoring your TCP network some counters which may be called parameters TCP Retransmits, TCP Retransmit Timers and TCP Duplicate Acknowledgments. These counters may appear in the output of the Netstat STATS command or you may see them while interrogating the SNMP MIB. These counters may all be related and indicate problems with congestion on the network. Duplicate acknowledgements indicate that packets are either lost or received out of sequence. When three duplicate acks are received, then the packet is retransmitted.

If there are a considerable number of duplicate acks, you may want to find out exactly which addresses and subnets may be having the problems. Duplicate acks can impact
network response time - called Round Trip Time. You may want to see if either Round Trip Time or, more likely, Round Trip Variance is affected by duplicate acks.

If there is excessive Round Trip Variance, then the user may be frustrated by erratic response time. You need to find which remote addresses have duplicate acknowledgments. After you find which addresses are having problems, then you may want to see if they all have something in common. The things which may be in common are:

- The same subnet
- The same time of day
- The same socket application
- The same route - set of hardware

We have seen situations where one device had over 49,000 duplicate acks. This was an IP printer which was out of paper. Consider the impact on your network of many such devices!

Commandment 5: Thou shalt relate thy TCP resets to the cause.

A RESET packet is sent by TCP to abort a connection. The fact that you have resets may or may not indicate a problem on the network. For example, a RESET segment is set to terminate a connection. A user may have gone away and left the connection idle. The application may have a keep-alive process which terminates the connection after a period of idle time. In this instance, the RESET to close the connection would be proper and indicate no problem. On the other hand, if an application is refusing connections because it is out of resources then you may see many RESETs.

You may find in monitoring your TCP network some counters which may be called Established Resets and Resets Out. Established Resets is the number of connections which were reset and Resets Out is the number of segments sent with the RESET flag on. Investigating the cause of resets can help you to find many types of problems. Let’s look at the next commandment for an example.
Commandment 6: Thou shalt not fail to watch your TCP attempt fails

You may find in monitoring your TCP network some counters which may be called Connection Attempts Failed, Connection Attempts Dropped, or Connection Attempts Discarded. These counters may appear in the output of the Netstat STATS command or you may see them while interrogating the SNMP MIB.

These counters mean a remote host IP address has tried to connect to an application on the mainframe and the connection was not successful. It could be that the application the remote users want to get to is not active or does not exist. One cause of degradation on TCP networks is unneeded traffic. Sometimes PC’s or other types of hardware on the network do ‘broadcast’ type queries to many devices on the LAN and even to the mainframe to ask for applications which are PC based.

The next figure will show SYN packets sent to start a connection to a port 445 which did not even exist on the mainframe. A SYN packet will be responded to by a SYN-ACK packet if the open of the application is successful. In this case, each SYN packet was responded to by a RST packet. The RST packet indicates that the session could not be established. Each time this sequence happens, the Connection Attempts Dropped or Connection Attempts Discarded counters will increment.

![Figure 4: SYN Packets to Port 445](image)

![Figure 5: RST Packets From TCP](image)

Notice that in Figures 3 and 4 you see SYN packet 75 responded to by RST packet 76, SYN packet 148 responded to by RST packet 149, and SYN packet 225 responded to by RST packet 226. Port 445 is used for resource sharing on Microsoft Windows 2000, XP, 2003 and other Samba based connections. If you are using Samba on the mainframe, this
may be fine. Otherwise, if this is just a mistake and happens thousands of times a day because some PC’s are not properly configured, then I will leave it to your imagination how much unnecessary CPU time the TCP stack may be taking for needless error recovery.

**Commandment 7: Thou shalt delve deeply into UDP no ports errors**

The UDP equivalent of TCP Attempts Failed, is UDP No Ports. These counters may appear in the output of the Netstat STATS command or you may see them while interrogating the SNMP MIB. UDP No Ports means that some packets were sent for a UDP port which was not available. It may be that there is a UDP application which is not active. If all UDP sockets are active, then it may be that UDP traffic is coming in at too high a rate for a particular port. We have seen this error to be correlated with ICMP Destination Unreachable Sub Type Port Unreachable error.

The ICMP Destination Unreachable message has a number of subtypes which indicate the type of error. These subtypes are as follows:

0 = Network Unreachable
1 = Host Unreachable
2 = Protocol Unreachable
3 = Port Unreachable
4 = Fragmentation Needed but Do Not Fragment Set
5 = Source Route Failed
6 = Destination Network Unknown
7 = Destination Host Unknown
8 = Source Host Isolated
9 = Network Access Prohibited
10 = Host Access Prohibited
11 = Network Unreachable for Type of Service
12 = Host Unreachable for Type of Service

The subtype we get in this case is sub type 3 or Port Unreachable.

<table>
<thead>
<tr>
<th>Packet Number</th>
<th>Date</th>
<th>Time</th>
<th>E/F</th>
<th>Interface</th>
<th>Device</th>
<th>Source Address</th>
<th>Source Port</th>
<th>Destination Address</th>
<th>Dest Port</th>
<th>Packet type</th>
</tr>
</thead>
<tbody>
<tr>
<td>671</td>
<td>2005-04-27</td>
<td>11:18:29:608346</td>
<td>F</td>
<td>LOOPBACK</td>
<td>Loopback</td>
<td>10.3.32.37</td>
<td>10139</td>
<td>10.3.32.37</td>
<td>161</td>
<td>UNREACH_PORT</td>
</tr>
<tr>
<td>735</td>
<td>2005-04-27</td>
<td>11:18:32:608541</td>
<td>F</td>
<td>LOOPBACK</td>
<td>Loopback</td>
<td>10.3.32.37</td>
<td>10139</td>
<td>10.3.32.37</td>
<td>161</td>
<td>UNREACH_PORT</td>
</tr>
</tbody>
</table>

**Figure 6: ICMP Port Unreachable**

This ICMP packet is generated because the IP address 10.3.32.37 port 10139 tried to access port 161 on the same machine. No application was listening on port 161, so this generated this ICMP error. Since this port happened to be for UDP, then it also generated
a UDP No Ports error. As with improperly configured access to TCP ports, if this is just a mistake and happens thousands of times a day because some application is not properly configured…

Commandment 8: Thou shalt address the reason for your IP address errors

Let’s take a look now at problems which may appear in IP traffic. You may find in monitoring your TCP network some counters which may be called IP Address Errors. Again, you may see this with the Netstat Stats command or from the SNMP MIB. One especially suspicious circumstance is when the IP address errors and IP discards in counters are the same.

Apparently, packets are coming in with an ‘unknown’ address and are being discarded. What kind of packets might these be? Take a look at Figure 7. You will see many packets coming into the mainframe with an address of 255.255.255.255. This is a broadcast address - by setting the address to all ones (255.255.255.255), all hosts on the network receive the broadcast. These packets may not contain data which the mainframe can understand or is interested in, so the packets are discarded. Why send packets just to have them discarded?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Interface</th>
<th>Device</th>
<th>Source Address</th>
<th>Source Port</th>
<th>Destination Address</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>OSD4</td>
<td>QDIO</td>
<td>10.30.200.8</td>
<td>34438</td>
<td>255.255.255.255</td>
<td>60001</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>OSD4</td>
<td>QDIO</td>
<td>10.30.2.11</td>
<td>14247</td>
<td>255.255.255.255</td>
<td>14247</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>OSD4</td>
<td>QDIO</td>
<td>10.30.1.200</td>
<td>2683</td>
<td>255.255.255.255</td>
<td>164</td>
</tr>
</tbody>
</table>

Figure 7: IP Address Errors
Commandment 9: Thou shalt not convert thy applications directly from multi-dropped SDLC

When many short segments are sent, there is overhead associated with the traffic – each packet contains at least 40 bytes in IP and TCP headers. If you send a very short segment, then a high proportion of the packet is overhead. Take a look at Figure 8 – you will see an application which was ideal for a multi-dropped SDLC link. It was converted directly to a TCP application keeping the same message lengths. On a TCP virtual circuit, this kind of application is quite resource intensive and may have poor response time.

<table>
<thead>
<tr>
<th>Packet Number</th>
<th>Date</th>
<th>Time</th>
<th>F</th>
<th>T</th>
<th>Interface</th>
<th>Device</th>
<th>Source Port</th>
<th>Dest Port</th>
<th>Data Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2005-03-30</td>
<td>07:55:36.693322</td>
<td>F</td>
<td>T</td>
<td>LOSABP1</td>
<td>QDIO</td>
<td>1197</td>
<td>23</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>2005-03-30</td>
<td>07:55:36.696255</td>
<td>T</td>
<td>T</td>
<td>LOSABP1</td>
<td>QDIO</td>
<td>23</td>
<td>1197</td>
<td>86</td>
</tr>
<tr>
<td>8</td>
<td>2005-03-30</td>
<td>07:55:36.703131</td>
<td>F</td>
<td>T</td>
<td>LOSABP1</td>
<td>QDIO</td>
<td>1197</td>
<td>23</td>
<td>8</td>
</tr>
<tr>
<td>349</td>
<td>2005-03-30</td>
<td>07:55:37.223269</td>
<td>F</td>
<td>T</td>
<td>LOSABP1</td>
<td>QDIO</td>
<td>1197</td>
<td>23</td>
<td>10</td>
</tr>
<tr>
<td>355</td>
<td>2005-03-30</td>
<td>07:55:37.226401</td>
<td>T</td>
<td>T</td>
<td>LOSABP1</td>
<td>QDIO</td>
<td>23</td>
<td>1197</td>
<td>68</td>
</tr>
</tbody>
</table>

Figure 8: Many Short Segments

Commandment 10: Thou shalt not use two packets when one will do

Our final commandment goes along with everything else we have been talking about in this article. I have seen applications send two packets for each transmission with the 2nd packet having only a protocol flag set on! The protocol flag could have been combined with the flags in the first packet. This is a small mistake, but when you make it a million times a day, it becomes a big mistake.

Conclusion

Tuning TCP/IP is like death by a thousand cuts. There can be thousands of small mistakes. When we do tuning, we work little by little fixing each one the best we can. The results have been quite satisfying. As problems are fixed, we can see overhead CPU usage for the TCP stack on the mainframe reducing. Throughput and response time for the applications should also improve. It is a task well worth undertaking.