Part 3, Monitoring your network packets and tuning the network

Martin Brown and Ken Milberg
Published on January 11, 2011
While running commands (such as netstat) can provide useful information, sometimes you need to drill down more to the packet level. This is where tracing tools come in handy, such as iptrace, ipreport, and tcpdump. You'll also learn how you can tune a network using tools such as no. The no command is similar to vmo and ioo, but no is the network flavor. This article focuses on tcp workload tuning, udp workload tuning, and some other noteworthy parameters with the no utility. The article also addresses ARP cache tuning and how to monitor and tune ARP statistics. You will also look at name resolution and how you can easily increase performance by making small adjustments to resolve hostnames.
Monitoring network packets

In this section, you'll see an overview of tools available to help you monitor your network packets. These tools allow you to troubleshoot a performance problem quickly and capture data for historical trending and analysis.

Part 1 of this series addressed some of the very basic flags, such as -in, that you typically use with netstat. Using netstat, you can also monitor more detailed information about the packets themselves. For example, the -D option shows the overall number of packets received, transmitted, and dropped in your communication subsystem. The results are sorted by device, driver, and protocol (see Listing 1).

Listing 1. netstat with the -D option

```
1488pp065_pub[/tmp] > netstat -D

Source                         Ipkts                Opkts
Idrops     Odrops
--------------------------------------------------------------
------------------
ent_dev1                    71306227          337207          0          0
ent_dev0                   203313084          82292          0          0

--------------------------------------------------
-------------
Devices Total              274619311          419499          0          0
```

---

```
1488pp065_pub[/tmp] > netstat -D

Source                         Ipkts                Opkts
Idrops     Odrops
--------------------------------------------------------------
------------------
ent_dev1                    71306227          337207          0          0
ent_dev0                   203313084          82292          0          0

--------------------------------------------------
-------------
Devices Total              274619311          419499          0          0
```
<table>
<thead>
<tr>
<th>Drivers Total</th>
<th>274619311</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Demuxer Total</th>
<th>273174517</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>1444794</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IP</th>
<th>204276236</th>
<th>1063977</th>
</tr>
</thead>
<tbody>
<tr>
<td>899839</td>
<td>828213</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IPv6</th>
<th>70588</th>
<th>70588</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6208</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TCP</th>
<th>714368</th>
<th>785630</th>
</tr>
</thead>
<tbody>
<tr>
<td>72</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Protocol</td>
<td>Total Bytes</td>
<td>Total Packets</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td><strong>UDP</strong></td>
<td>202697468</td>
<td>319900</td>
</tr>
<tr>
<td></td>
<td>202172157</td>
<td>0</td>
</tr>
</tbody>
</table>

Protocols Total

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Total Bytes</th>
<th>Total Packets</th>
<th>Total Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Protocols Total</strong></td>
<td>407688072</td>
<td>2169507</td>
<td></td>
</tr>
<tr>
<td></td>
<td>203072068</td>
<td>828213</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interface</th>
<th>Total Bytes</th>
<th>Total Packets</th>
<th>Total Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>en_if1</strong></td>
<td>70327759</td>
<td>337207</td>
<td>0</td>
</tr>
<tr>
<td><strong>en_if0</strong></td>
<td>202846780</td>
<td>82315</td>
<td>0</td>
</tr>
<tr>
<td><strong>lo_if0</strong></td>
<td>780891</td>
<td>780890</td>
<td>12</td>
</tr>
</tbody>
</table>

Net IF Total

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Total Bytes</th>
<th>Total Packets</th>
<th>Total Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Net IF Total</strong></td>
<td>273955430</td>
<td>1200412</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Total Bytes</th>
<th>Total Packets</th>
<th>Total Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NFS/RPC Client</strong></td>
<td>24</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>NFS/RPC Server</strong></td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>NFS Client</strong></td>
<td>279</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>
Another useful flag is the -s, which shows detailed statistics for all protocols used, including packets sent, received and dropped. If you only want to view tcp, you can also use the -p flag (see Listing 2).

Listing 2. netstat with the -p option

```
1488pp065_pub[/tmp] > netstat -p tcp:

785684 packets sent
227657 data packets (13800898 bytes)
0 data packets (0 bytes) retransmitted
279285 ack-only packets (509 delayed)
0 URG only packets
0 window probe packets
```
69732 window update packets

418020 control packets

0 large sends

0 bytes sent using largesend

0 bytes is the biggest largesend

714418 packets received

360033 acks (for 14009902 bytes)

69728 duplicate acks

0 acks for unsent data

217241 packets (1660096 bytes) received in-sequence

72 completely duplicate packets (72 bytes)

0 old duplicate packets

0 packets with some dup. data (0 bytes duped)

69632 out-of-order packets (0 bytes)

0 packets (0 bytes) of data after window

0 window probes

69733 window update packets

0 packets received after close
0 packets with bad hardware assisted checksum

0 discarded for bad checksums

0 discarded for bad header offset fields

0 discarded because packet too short

0 discarded by listeners

0 discarded due to listener's queue full

5241 ack packet headers correctly predicted

75327 data packet headers correctly predicted

69655 connection requests

69712 connection accepts

139364 connections established (including accepts)

139417 connections closed (including 12 drops)

0 connections with ECN capability

0 times responded to ECN

2 embryonic connections dropped

429685 segments updated rtt (of 429463 attempts)
0 segments with congestion window reduced bit set

0 segments with congestion experienced bit set

0 resends due to path MTU discovery

1 path MTU discovery termination due to retransmits

3 retransmit timeouts

0 connections dropped by retransmit timeout

0 fast retransmits

0 when congestion window less than 4 segments

0 newreno retransmits

0 times avoided false fast retransmits

0 persist timeouts

0 connections dropped due to persist timeout

0 keepalive timeouts

0 keepalive probes sent

0 connections dropped by keepalive

0 times SACK blocks array is extended

0 times SACK holes array is extended
0 packets dropped due to memory allocation failure

0 connections in timewait reused

0 delayed ACKs for SYN

0 delayed ACKs for FIN

0 send_and_disconnects

0 spliced connections

0 spliced connections closed

0 spliced connections reset

0 spliced connections timeout

0 spliced connections persist timeout

0 spliced connections keepalive timeout

0 TCP checksum offload disabled during retransmit

0 Connections dropped due to bad ACKs

There are actually so many different ways of using netstat that the best place to start is by looking at the man page and go from there. Don't be afraid to run these commands, because they won't eat up disk space or affect performance. The tracing tools that are provided within AIX 7 are used to record detailed information about the packets. Use them with more caution. These tools are extremely helpful when trying to determine the root cause of network performance problems.
First, look at iptrace and ipreport. The iptrace command records all the packets received from the network interfaces. The ipreport command formats the data that is generated from iptrace into a readable trace report. Further, you can also use ipfilter to sort the output file created from ipreport. Try starting the trace and keep it going for one minute (see Listing 3).

### Listing 3. Starting the trace

1. `l488pp065_pub[/tmp] > /usr/sbin/iptrace -a -i en0 iptrace.out &`  
2. `[1] 12845206`  
3. `l488pp065_pub[/tmp] > [8126632]`  
4. `[1] + Done /usr/sbin/iptrace -a -i en0 iptrace.out &`  
5. `l488pp065_pub[/tmp] > ps -ef | grep iptrace`  
6. `root 8126632 15 05:54:55 - 0:00 /usr/sbin/iptrace -a -i en0 iptrace.out`  
7. `root 14221424 7012524 7 05:55:17 pts/1 0:00 grep iptrace`

When you are done with the trace, you need to kill the process (see Listing 4).

### Listing 4. Killing the process
Now, examine the output (see Listing 5):

**Listing 5. Examining the output**

```bash
l488pp065_pub[/tmp] > more /ipreport.network

IPTRACE version: 2.0

ETH: ====( 114 bytes transmitted on interface en0 )==== 05:54:55.151599119


IP: < SRC = 172.29.148.225 > (l488pp065_pub)

IP: < DST = 172.29.131.16 >

IP: ip_v=4, ip_hl=20, ip_tos=16, ip_len=100, ip_id=49399, ip_off=0 DF
```
Listing 5 shows the captured information about each packet, including packet size and IP address information. As you can imagine, the trace file can get very large quickly. The example file grew to 40MB in less than one minute! Be very careful when running these traces, because you will run out of disk space really fast if you don't have the disk bandwidth for these files.

You can also start the trace using the System Resource Controller (SRC). See Listing 6.
Listing 6. Starting the trace using SRC

1 l488pp065_pub[/tmp] > startsrc -s iptrace -a "-i en1 /home/testing/iptrace/iptracelog"

0513-059 The iptrace Subsystem has been started. Subsystem PID is 12845270.

5 l488pp065_pub[/tmp] > stopsrc -s iptrace

0513-004 The Subsystem or Group, iptrace, is currently inoperative.

What about tcpdump? tcpdump prints out headers of the packets, which are captured for each NIC. One important difference with tcpdump is that, unlike iptrace, it can look at only one network interface at a time. And, because iptrace examines the entire packet from the kernel space, the results can offer lots of dropped packets. With tcpdump, you can also limit the amount of data to be traced. Also, you do not need to use an ipreport type of command to format binary data, because tcpdump does the trace and the output. See Listing 7 for an example.

Listing 7. Using tcpdump

1 l488pp065_pub[/tmp] > tcpdump > tcp.outtcpdump: listening on

2 en0, link-type 1, capture size 96 bytes
tcpdump continues to capture packets until you hit Ctrl+C. If any packets were dropped due to a lack of buffer space, it reports that, too.

**Listing 8** shows what you see when you end the example trace and view the file.

**Listing 8. End of the trace**

```
1 28 packets received by filter
0 packets dropped by kernel

3 l488pp065_pub[/tmp] > cat tcp.out

4

5 06:00:21.003328 IP l488pp065_pub.ssh > 172.29.131.16.54678:

6 P 47609416:47609464(48) ack 3002357700 win 65522 <nop,nop,timestamp 1330145971

7 1 1284351989>

8 06:00:21.003387 IP l488pp065_pub.ssh > 172.29.131.16.54678: P 48:208(160)

9 ack 1 win 65522 <nop,nop,timestamp 1330145971

10 1284351989>

11 06:00:21.003628 IP l488pp065_pub.ssh > 172.29.131.16.54678: . ack 208 win

12 32761 <nop,nop,timestamp 1284351989

13 1330145971>
```
06:00:21.238937 ARP, Request who-has
172.29.173.186 tell 172.29.133.221, length 46

06:00:21.239110 ARP, Request who-has
172.29.173.236 tell 172.29.129.59, length 46

06:00:21.325060 ARP, Request who-has
172.29.175.252 tell 172.29.129.58, length 46

06:00:21.464383 IP6 fe80::4464:ceff:fe65:4f0c >
ff02::1:ff41:34d0: ICMP6,
neighbor solicitation, who has
fe80::221:5eff:fe41:34d0, length

32

06:00:21.505281 ARP, Request who-has
172.29.175.60 tell 172.29.133.223, length 46

06:00:22.013530 ARP, Request who-has
172.29.174.66 tell 172.29.132.222, length 46

06:00:22.054164 ARP, Request who-has
172.29.173.237 tell 172.29.129.59, length 46

06:00:22.076819 ARP, Request who-has
172.29.122.25 tell 172.29.122.2, length 46

06:00:22.393898 IP 172.29.148.116.32852 >
239.255.255.253.svrlc: UDP, length 56

06:00:22.464355 IP6 fe80::4464:ceff:fe65:4f0c >
ff02::1:ff41:34d0: ICMP6, neighbor
solicitation, who has fe80::221:5eff:fe41:34d0, length
06:00:22.935140 802.1d config
8000.00:16:60:f9:a8:00.8011 root
8000.00:16:60:f9:a8:00

pathcost 0 age 0 max 20 hello 2 fdelay 15

06:00:23.186380 ARP, Request who-has
172.29.122.26 tell 172.29.122.2, length 46

06:00:24.520770 ARP, Request who-has
172.29.175.60 tell 172.29.133.223, length 46

06:00:24.558139 ARP, Request who-has
172.29.175.252 tell 172.29.129.58, length 46

06:00:24.573524 ARP, Request who-has
172.29.175.5 tell 172.29.129.57, length 46

06:00:24.736838 IP 172.29.148.116.32853 >
239.255.255.253.svrloc: UDP, length 56

06:00:25.029112 IP 172.29.133.222.netbios-dgm >
172.29.191.255.netbios-dgm: UDP, length 201

06:00:25.029965 IP 172.29.133.222.netbios-dgm >
172.29.191.255.netbios-dgm: UDP, length 201
06:00:25.030751 IP 172.29.133.222.netbios-dgm >
   172.29.191.255.netbios-dgm: UDP, length 201
06:00:25.031674 IP 172.29.133.222.netbios-dgm >
   172.29.191.255.netbios-dgm: UDP, length 201
06:00:25.032636 IP 172.29.133.222.netbios-dgm >
   172.29.191.255.netbios-dgm: UDP, length 201
06:00:25.033647 IP 172.29.133.222.netbios-dgm >
   172.29.191.255.netbios-dgm: UDP, length 201
06:00:25.033732 CDP v2, ttl: 180s, Device-ID 'Switch'[cdp]
06:00:25.034738 IP 172.29.133.222.netbios-dgm >
   172.29.191.255.netbios-dgm: UDP, length 201
06:00:25.035741 IP 172.29.133.222.netbios-dgm >
   172.29.191.255.netbios-dgm: UDP, length 201
The main benefit of tcpdump is that you can specify filters so that you can select only particular protocols, sources, destinations, ports and other combinations. This is useful if you want diagnose or determine the NFS traffic, for example, between two hosts.

## Tuning network performance

In this section, you will learn how to use the `no` command to tune your network subsystem. You'll also look at other areas that can impact network performance, and you'll learn about recommended tuning methodologies where appropriate.

The most important command for tuning network parameters is the `no` command. First, take a look at all the parameters, using the `-a` flag (see Listing 9). Be warned, though, that the list is quite extensive.

### Listing 9. Viewing the parameters

```
l488pp065_pub/[tmp] > no -a
    arpqsize = 12
    arpt_killc = 20
    arptab_bsiz = 7
    arptab_nb = 149
    bcastping = 0
    clean_partial_conns = 0
    delayack = 0
    delayackports = { }
    dgd_packets_lost = 3
```
dgd_ping_time = 5

dgd_retry_time = 5

directed_broadcast = 0

fasttimo = 200

icmp6_errmsg_rate = 10

icmpaddressmask = 0

ie5_old_multicast_mapping = 0

ifsize = 256

ip6_defttl = 64

ip6_prune = 1

ip6forwarding = 0

ip6srcrouteforward = 1

ip_ifdelete_notify = 0

ip_nfrag = 200

ipforwarding = 0

ipfragttl = 2

ipignoreredirects = 0

ipqmaxlen = 100

ipsendredirects = 1
ipsrcrouteforward = 1
ipsrcrouterecv = 0
ipsrcroutesend = 1
llsleep_timeout = 3
lo_perf = 1
lowthresh = 90
main_if6 = 0
main_site6 = 0
maxnip6q = 20
maxttl = 255
medthresh = 95
mpr_policy = 1
multi_homed = 1
nbc_limit = 262144
nbc_max_cache = 131072
nbc_min_cache = 1
nbc_ofile_hashsz = 12841
nbc_pseg = 0
nbc_pseg_limit = 524288
ndd_event_name = {all}

ndd_event_tracing = 0

ndp_mmaxtries = 3

ndp_umaxtries = 3

ndpqsize = 50

ndpt_down = 3

ndpt_keep = 120

ndpt_probe = 5

ndpt_reachable = 30

ndpt_retrans = 1

net_buf_size = {all}

net_buf_type = {all}

net_malloc_frag_mask = {0}

netm_page_promote = 1

nonlocsrcroute = 0

nstrpush = 8

passive_dgd = 0

pmtu_default_age = 10

pmtu_expire = 10
Alternatively, you can also use the -L flag. It provides much more detailed information, including current, default, boot, and range of settings. This can make it much easier to determine whether a given value is working at its optimum value and whether a specific item could be improved beyond it’s current value. **Listing 10** shows you only the first few lines.

### Listing 10. Using the -L flag

```plaintext
1488pp065_pub[/tmp] > no -L

General Network Parameters

<table>
<thead>
<tr>
<th>NAME</th>
<th>CUR</th>
<th>DEF</th>
<th>BOOT</th>
<th>MIN</th>
<th>MAX</th>
<th>UNIT</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>fasttimo</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>50</td>
<td>200</td>
<td>millisecond</td>
<td>D</td>
</tr>
</tbody>
</table>
```
nbc_limit               256K  256K  256K  0
8E-1   kbyte              D

              thewall

------------------
------------------
------------------
------------------
------------------
nbc_max_cache             128K  128K  128K  1
256M   byte              D

nbc_min_cache
nbc_limit

------------------
------------------
------------------
------------------
------------------
nbc_min_cache             1      1      1      1      128K
byte              D

nbc_max_cache

------------------
------------------
------------------
------------------
------------------
nbc_ofile_hashsz          12841  12841  12841  1
999999 segment           D

------------------
------------------
------------------
------------------
------------------
nbc_pseg                  0      0      0      0      2G-1
segment           D
There are many parameters here. The thewall defines the upper limit for network kernel buffers. Today, its size is defined at installation time, depending on the amount of RAM and the kernel type. For example, if you are running AIX 5.3 on a 64-bit kernel, the parameter is set at half the size of real memory.

**Listing 11. Setting the size of the thewall parameter**

```bash
1488pp065_pub[/tmp] > no -a|grep thewall
thewall = 1048576
```

Part 1 discussed mbufs, but it's worth another mention here, because it relates to thewall. Remember that mbufs are used to store data in the kernel for both incoming and outgoing traffic. This is why determining the right amount of mbufs is extremely important. The value of the maxmbuf tunable limits
the amount of memory that the communication systems use. If the value is 0, thewall tunable is used, and it cannot be modified from its default. Changing this tunable is a way to lower the thewall limit. As the default, if maxmbuf is 0, this value is used regardless of what thewall uses. netstat -m is used to detect shortages of failures of network memory requests (see Listing 12).

### Listing 12. netstat with the -m option

<table>
<thead>
<tr>
<th>l488pp065_pub[/tmp]</th>
<th>netstat -m</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Kernel malloc statistics:</td>
</tr>
<tr>
<td></td>
<td>******* CPU 0 *******</td>
</tr>
<tr>
<td></td>
<td>By size inuse calls failed delayed free</td>
</tr>
<tr>
<td></td>
<td>hiwat freed</td>
</tr>
<tr>
<td>64</td>
<td>558 2087455 0 7 274</td>
</tr>
<tr>
<td>5240</td>
<td>0</td>
</tr>
<tr>
<td>128</td>
<td>5884 1901723 0 175 164</td>
</tr>
<tr>
<td>2620</td>
<td>0</td>
</tr>
<tr>
<td>256</td>
<td>5780 653578 0 295 2876</td>
</tr>
<tr>
<td>5240</td>
<td>500</td>
</tr>
<tr>
<td>512</td>
<td>7970 182051630 0 972</td>
</tr>
<tr>
<td>102</td>
<td>6550 0</td>
</tr>
<tr>
<td>1024</td>
<td>3159 1960612 0 794 49</td>
</tr>
<tr>
<td>2620</td>
<td>0</td>
</tr>
<tr>
<td>2048</td>
<td>1069 3462138 0 520 25</td>
</tr>
<tr>
<td>3930</td>
<td>0</td>
</tr>
<tr>
<td>4096</td>
<td>2056 2794 0 83 3</td>
</tr>
</tbody>
</table>
In the example, there are no shortages (failures).

Although there are many parameters you can change with the no utility, most of them are better left alone. The most important parameters are ones that refer to TCP streaming workload tuning.
- tcp_sendspace—This controls how much buffer space in the kernel is used to buffer application data. You really want to bump this up from the default, because if its limit is reached, the sending application suspends data transfer until TCP sends the data to the buffer.
- tcp_receive space—In addition to controlling the amount of buffer space to be consumed by receive buffers, AIX 7 also uses this value to determine the size to make its transmit window.
- udp_sendspace—With UDP, you can set this to no more than 65536, because IP has an upper limit of 65536 bytes per packet.
- udp_resvspace—This value should be greater than udp_sendspace, because it needs to handle as many simultaneous UDP packets per socket as it can. This parameter can easily be set to 10 times the value of udp_sendspace.

Now, let's make some changes. First, increase the size of udp_sendspace (see Listing 13).

**Listing 13. Increasing the size of udp_sendspace**

1
2
3

```bash
l488pp065_pub[/tmp] > no -p -o udp_sendspace=65536
Setting udp_sendspace to 65536

Setting udp_sendspace to 65536 in nextboot file
Change to tunable udp_sendspace, will only be effective for future connections
```

Next, change udp_recvspace to the recommended configuration of 10 times udp_sendspace). See Listing 14.

**Listing 14. Changing udp_recvspace**

1
2

```bash
l488pp065_pub[/tmp] > no -p -o udp_recvspace=655360
Setting udp_recvspace to
```

2
3

```bash
Setting udp_recvspace to 655360 in nextboot file
```

Change to tunable udp_recvspace, after the next boot only.
Setting `udp_recvspace` to 655360 in `nextboot` file

Change to tunable `udp_recvspace`, will only be effective for future connections

Note that the `-p` flag keeps the entries, even after a reboot. It appends the `/etc/tunables/nextboot` stanza file, as shown in Listing 15.

Listing 15. Looking at the `/etc/tunables/nextbook` file

```
no:
  udp_recvspace = "655360"
  udp_sendspace = "65536"
```

Regarding the `tcp` parameters for higher speed adapters, there is no problem setting `tcp_sendspace` to twice the value of `tcp_recvspace`. For example, you can use the settings in Listing 16.

Listing 16. Examples settings for `tcp_sendspace`

```
tcp_receivespace = 262144
tcp_sendspace= 524288
```
Other important workload parameters include rfc1323 and sb_max.

The rfc1323 tunable enables the TCP window scaling option, which allows TCP to use a larger window size. Turning it on enables the best TCP performance. The sb_max tunable sets an upper limit on the number of socket buffers queued to an individual socket, which controls the amount of buffer space consumed by buffers (queued to either a sender or received socket). This amount should usually be less than the wall and approximately 4 times the size of the largest value of the tcp or udp send and receive settings. For example, if your udp_recvspace is 655360, you can’t go wrong if you double it to 1310720.

Now look at tcp_nodelayack. This tunable prompts TCP to send an immediate acknowledgement, rather than a delayed acknowledgement. While this can add more overhead in some environments, it can greatly improve network performance in others. If you change this parameter, but it does not improve performance, you can quickly change it back.

Next look at ipqmalen. This tunable controls the length of the IP input queue. If you see an overflow counter (through the use of netstat -s), setting a maximum length of this queue can help fix the overflow.

What about ARP? When many clients are connected to the system, you might want to tune the ARP cache. You can look at the statistics using netstat (see Listing 17).

**Listing 17. Using netstat with -p arp**

```
1488pp065_pub[/tmp] > netstat -p arparp:
12 packets sent
0 packets purged
```
If you see a high purge count, increase the size of the ARP table. For the example table, this isn't needed.

Here are the no parameters that relate to ARP (see Listing 18).

**Listing 18. Using the no parameters**

```
1  l488pp065_pub[/tmp] > no -a | grep arp
2    arpqsize = 12
3    arpt_killc = 20
4    arptab_bsiz = 7
5    arptab_nb = 149
```

You can view the specific interface settings using either ifconfig or lsattr. In the example in Listing 19, look at the settings using ifconfig (look at the last line which references some of the tunables mentioned earlier).

**Listing 19. Viewing specific interface settings using ifconfig**

```
1  l488pp065_pub[/tmp] > ifconfig en0
2      en0en0: flags=1e080863,480<P,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD(ACTIVE),CHAIN>
3                        inet 172.29.148.225 netmask 0xffffc000
4                        broadcast 172.29.191.255
```
You can change these options (by interface) by using SMIT, chdev, or ifconfig. Note that ifconfig will not update the Object Data Manager (ODM). Therefore, on a reboot, it will revert to the previous value. Because of that, you should use SMIT: the fastpath of `smit tcpip>further configuration>Network interfaces>Change/Show characteristics` of an interface (see Figure 1).

Figure 1. Using SMIT to change interface settings

You might wonder why the no parameters don't apply to some interfaces. Name resolution is another area that can impact performance. If you know how you want to resolve (using DNS or the hosts file), make sure name resolution is set up correctly in the `/etc/netsvc.conf` file. Look at a piece of the file in Listing 20.

Listing 20. Piece of `/etc/netsvc.conf` file
If you're using DNS, take out the local if you are not using a host's file at all, or you can leave it in if you are using it as a backup to DNS (but make it the second entry). Alternatively, take out the bind if you're not using DNS at all, because it will only slow down your performance by first attempting (if it is the first entry in the record) to resolve using a Name Server that doesn't exist.

Summary

This article discussed how to monitor network packets on the network. You used netstat and drilled down to the packet level using tracing tools, such as iptrace and tcpdump. Further, you learned how to tune your network using the no utility. Using this utility, you explored tcp and udp workload tuning while also learning some other noteworthy parameters. You made tuning changes and read about how you might want to tune certain settings. You also examined ARP cache tuning and saw how you could monitor and tune ARP statistics. You looked at ISNO and learned how you could tune specific no tunables by interface. You also looked at name resolution and how you could easily increase performance by making small adjustments in how to resolve hostnames.

Related topics

- AIX Wiki: Get the nmon manual or downloads here.
• Improving database performance with AIX concurrent I/O: Read this white paper for more information on how to improve database performance.

• Power Architecture: High-Performance Architecture with a History: Read this white paper.

• nmon analyser -- A free tool to produce AIX performance reports (Steven Atkins, developerWorks, April 2006): You can download nmon analyser from here.

• Operating System and Device Management from IBM provides users and system administrators with complete information that can affect your selection of options when performing such tasks as backing up and restoring the system, managing physical and logical storage, and sizing appropriate paging space.

• The AIX 7.1 Information Center is your source for technical information about the AIX operating system.

• The IBM AIX Version 6.1 Differences Guide can be a useful resource for understanding changes in AIX 6.1.

• AIX and UNIX: The AIX and UNIX developerWorks zone provides a wealth of information relating to all aspects of AIX systems administration and expanding your UNIX skills.

• IBM trial software: Build your next development project with software for download directly from developerWorks.

• Future Tech: Visit Future Tech’s site to learn more about their latest offerings.

Posted - Mon, Jun 4, 2018 12:20 PM. This article has been viewed 397 times.

Online URL: http://kb.ictbanking.net/article.php?id=217