Documentation: Network Monitoring

The following is a guide to monitoring network interfaces and the network from the **Solaris** operating system.
Note: This document is not written by Sun.
Brendan Gregg, 09-Oct-2005, version 0.70.

Is your network busy?

The network often gets blamed when things are performing poorly, and perhaps this is correct - your network interfaces may be running at 100% utilisation.

What command will tell you how busy the network interface is? Many sysadmins suggest using netstat -i to find out,

```
$ netstat -i 1
    input   hme0       output           input  (Total)     output
packets errs  packets errs  colls  packets errs  packets errs  colls
70820498 6     73415337 0     0     113173825 6     115768664 0     0
242      0     149      0     0     246      0     153      0     0
1068     0     552      0     0     1075     0     559      0     0
[...]
```

This shows packet counts per second, the first line is the summary since boot. How many packets would mean the interface is busy? 100/sec, 1000/sec?

What we do know is the speed of the network interface, for this one it is 100 Mb/sec. However we have no way of telling the size of the packets - they may be 56 byte packets or 1500 bytes. This means that **the packet count is not useful**, perhaps it is useful as a yardstick of activity only. What we really need is Kb/sec...

Contents

By System

How to monitor network usage for the entire system, usually by network interface.

---

netstat

The Solaris netstat command is where a number of different network status programs have been dropped, it's the kitchen sink of network tools.

netstat -i as mentioned earlier, only prints packet counts. We don't know if they are big packets or small packets, and we can't use them to accurately determine how utilised the network interface is. There are other performance monitoring tools that plot this as a "be all and end all" value - this is wrong.

netstat -s dumps various network related counters from Kstat, the Kernel Statistics framework. This shows that Kstat does track at least some details in terms of bytes,

```
$ netstat -s | grep Bytes
        tcpOutDataSegs      =37367847   tcpOutDataBytes     =166744792
        tcpRetransSegs      =153437     tcpRetransBytes     =72298114
        tcpInAckSegs        =25548715   tcpInAckBytes       =148658291
        tcpInInorderSegs    =35290928   tcpInInorderBytes   =3637819567
        tcpInUnorderSegs    =324309     tcpInUnorderBytes   =406912945
        tcpInDupSegs        =152795     tcpInDupBytes       =73998299
        tcpInPartDupSegs    =  7896     tcpInPartDupBytes   =5821485
        tcpInPastWinSegs    =    38     tcpInPastWinBytes   =971347352
```

However the byte values above are for TCP in total, including loopback traffic that never travelled via the network interfaces.

netstat -k on Solaris 9 and earlier dumped all Kstat counters,

```
$ netstat -k | awk '/^hme0/,/^$/'
hme0:
ipackets 70847004 ierrors 6 opackets 73438793 oerrors 0 collisions 0
defer 0 framing 0 crc 0 sqe 0 code_violations 0 len_errors 0
ifspeed 100000000 buff 0 oflo 0 uflo 0 missed 6 tx_late_collisions 0
retry_error 0 first_collisions 0 nocarrier 0 nocanput 0
allocbfail 0 runt 0 jabber 0 babble 0 tmd_error 0 tx_late_error 0
rx_late_error 0 slv_parity_error 0 tx_parity_error 0 rx_parity_error 0
slv_error_ack 0 tx_error_ack 0 rx_error_ack 0 tx_tag_error 0
rx_tag_error 0 eop_error 0 no_tmds 0 no_tbufs 0 no_rbufs 0
rx_late_collisions 0 rbytes 289601566 obytes 358304357 multircv 558 multixmt 73411
brdcstrcv 3813836 brdcstxmt 1173700 norcvbuf 0 noxmtbuf 0   newfree 0
ipackets64 70847004 opackets64 73438793 rbytes64 47534241822 obytes64 51897911909 align_errors 0
fcs_errors 0   sqe_errors 0 defer_xmts 0 ex_collisions 0
macxmt_errors 0 carrier_errors 0 toolong_errors 0 macrcv_errors 0
link_duplex 0 inits 31 rxinits 0 txinits 0 dmarh_inits 0
dmaxh_inits 0 link_down_cnt 0 phy_failures 0 xcvr_vendor 524311
asic_rev 193 link_up 1
```

Great - so bytes by network interface are indeed tracked. However netstat -k was an undocumented switch that has now been dropped in Solaris 10. That's ok, as there are better ways to get to Kstat, including the C library that tools such as vmstat use - libkstat.

---

kstat

The Solaris Kernel Statistics framework does track network usage, and as of Solaris 8 there has been a /usr/bin/kstat command to fetch Kstat details,

```
$ kstat -p 'hme:0:hme0:*bytes64' 1
hme:0:hme0:obytes64     51899673435
hme:0:hme0:rbytes64     47536009231

hme:0:hme0:obytes64     51899673847
hme:0:hme0:rbytes64     47536009709
[...]
```

Now we just need a tool to present this in a more meaningful way.

---

nx.se

The SE Toolkit provides a language, SymbEL, that lets us write our own performance monitoring tools. It also contained a collection of example tools, including nx.se which lets us identify network utilisation,

```
$ se nx.se 1
Current tcp RtoMin is 400, interval 1, start Sun Oct  9 10:36:42 2005

10:36:43 Iseg/s Oseg/s InKB/s OuKB/s Rst/s  Atf/s  Ret%  Icn/s  Ocn/s
tcp       841.6    4.0  74.98   0.27  0.00   0.00   0.0   0.00   0.00
Name     Ipkt/s Opkt/s InKB/s OuKB/s IErr/s OErr/s Coll% NoCP/s Defr/s
hme0      845.5  420.8 119.91  22.56  0.000  0.000   0.0   0.00   0.00

10:36:44 Iseg/s Oseg/s InKB/s OuKB/s Rst/s  Atf/s  Ret%  Icn/s  Ocn/s
tcp       584.2    5.0  77.97   0.60  0.00   0.00   0.0   0.00   0.00
Name     Ipkt/s Opkt/s InKB/s OuKB/s IErr/s OErr/s Coll% NoCP/s Defr/s
hme0      579.2  297.1 107.95  16.16  0.000  0.000   0.0   0.00   0.00
[...]
```

Having KB/s values lets us determine exactly how busy our network interfaces are. There is other useful information printed above, including Coll% - collisions, NoCP/s - no can puts, and Defr/s defers, which may be evidence of network saturation.

---

nicstat

nicstat is a freeware tool written in C to print out network utilisation and saturation by interface,

```
$ nicstat 1
    Time   Int   rKb/s   wKb/s   rPk/s   wPk/s    rAvs    wAvs  %Util    Sat
10:48:30  hme0    4.02    4.39    6.14    6.36  670.73  706.50   0.07   0.00
10:48:31  hme0    0.29    0.50    3.00    4.00   98.00  127.00   0.01   0.00
10:48:32  hme0    1.35    4.23   14.00   15.00   98.79  289.00   0.05   0.00
10:48:33  hme0   67.73   19.08  426.00  207.00  162.81   94.39   0.71   0.00
10:48:34  hme0  315.22  128.91 1249.00  723.00  258.44  182.58   3.64   0.00
10:48:35  hme0  529.96   67.53 2045.00 1046.00  265.37   66.11   4.89   0.00
10:48:36  hme0  454.14   62.16 2294.00 1163.00  202.72   54.73   4.23   0.00
10:48:37  hme0   93.55   15.78  583.00  295.00  164.31   54.77   0.90   0.00
10:48:38  hme0   74.84   32.41  516.00  298.00  148.52  111.38   0.88   0.00
10:48:39  hme0    0.76    4.17    7.00    9.00  111.43  474.00   0.04   0.00
[...]
```

Fantastic. There is also an older Perl version of nicstat available.

The following are the switches available from version 0.90 of the C version,

```
$ nicstat -h
USAGE: nicstat [-hsz] [-i int[,int...]] | [interval [count]]

         -h                # help
         -i interface      # track interface only
         -s                # summary output
         -z                # skip zero value lines
    eg,
       nicstat             # print summary since boot only
       nicstat 1           # print every 1 second
       nicstat 1 5         # print 5 times only
       nicstat -z 1        # print every 1 second, skip zero lines
       nicstat -i hme0 1 # print hme0 only every 1 second
```

The utilisation measurement is based on the maximum speed of the interface (if available via Kstat), divided by the current throughput. The saturation measurement is a value that reflects errors due to saturation (no can puts, etc).

SNMP

It's worth mentionining that there is also useful data available in SNMP, which is used by software such as MRTG. Here we use Net-SNMP's snmpget to fetch some interface values,

```
$ snmpget -v1 -c public localhost ifOutOctets.2 ifInOctets.2
IF-MIB::ifOutOctets.2 = Counter32: 10016768
IF-MIB::ifInOctets.2 = Counter32: 11932165
```

These values are the outbound and inbound bytes for our main interface. In Solaris 10 a full description of the IF-MIB values can be found at /etc/sma/snmp/mibs/IF-MIB.txt.

Across Network

Analysing the performance of the external network.

---

ping

ping is the classic network probe tool,

```
$ ping -s mars
PING mars: 56 data bytes
64 bytes from mars (192.168.1.1): icmp_seq=0. time=0.623 ms
64 bytes from mars (192.168.1.1): icmp_seq=1. time=0.415 ms
64 bytes from mars (192.168.1.1): icmp_seq=2. time=0.464 ms
^C
----mars PING Statistics----
3 packets transmitted, 3 packets received, 0% packet loss
round-trip (ms)  min/avg/max/stddev = 0.415/0.501/0.623/0.11
```

So I discover that mars is up, and it responds within 1 ms. Solaris 10 enhanced ping to print 3 decimal places for the times.

ping is handy to see if a host is up, but that's about all. Some people use it to test whether their application server is ok. Hmm. ICMP is handled in the kernel without needing to call a user based process, so it's possible that a server will ping ok while the application either responds slowly or not at all.

---

traceroute

traceroute sends a series of UDP packets with an increasing TTL, and by watching the ICMP time expired replies can discover the hops to a host (assuming the hops actually decrement the TTL),

```
$ traceroute www.sun.com
traceroute: Warning: Multiple interfaces found; using 260.241.10.2 @ hme0:1
traceroute to www.sun.com (209.249.116.195), 30 hops max, 40 byte packets
 1  tpggate (260.241.10.1)  21.224 ms  25.933 ms  25.281 ms
 2  172.31.217.14 (172.31.217.14)  49.565 ms  27.736 ms  25.297 ms
 3  syd-nxg-ero-zeu-2-gi-3-0.tpgi.com.au (220.244.229.9)  25.454 ms  22.066 ms  26.237 ms
 4  syd-nxg-ibo-l3-ge-0-2.tpgi.com.au (220.244.229.132)  42.216 ms *  37.675 ms
 5  220-245-178-199.tpgi.com.au (220.245.178.199)  40.727 ms  38.291 ms  41.468 ms
 6  syd-nxg-ibo-ero-ge-1-0.tpgi.com.au (220.245.178.193)  37.437 ms  38.223 ms  38.373 ms
 7  Gi11-2.gw2.syd1.asianetcom.net (202.147.41.193)  24.953 ms  25.191 ms  26.242 ms
 8  po2-1.gw1.nrt4.asianetcom.net (202.147.55.110)  155.811 ms  169.330 ms  153.217 ms
 9  Abovenet.POS2-2.gw1.nrt4.asianetcom.net (203.192.129.42)  150.477 ms  157.173 ms *
```

```
10  so-6-0-0.mpr3.sjc2.us.above.net (64.125.27.54)  240.077 ms  239.733 ms  244.015 ms
11  so-0-0-0.mpr4.sjc2.us.above.net (64.125.30.2)  224.560 ms  228.681 ms  221.149 ms
12  64.125.27.102 (64.125.27.102)  241.229 ms  235.481 ms  238.868 ms
13  * *^C
```

The times may give me some idea where a network bottleneck is. We must also remember that networks are dynamic, and this may not be the permanent path to that host.

TTCP

Test TCP is a freeware tool to test the throughput between two hops. It needs to be run on both the source and destination, and there is a Java version of TTCP which will run on many different operating systems. Beware, it will flood the network with traffic to perform it's test.

The following is run on one host as a reciever. The options used make the test run for a reasonable duration - around 60 seconds,

```
$ java ttcp -r -n 65536
Receive: buflen= 8192  nbuf= 65536 port= 5001
```

Then the following was run on the second host as the transmitter,

```
$ java ttcp -t jupiter -n 65536
Transmit: buflen= 8192  nbuf= 65536 port= 5001
Transmit connection:
  Socket[addr=jupiter/192.168.1.5,port=5001,localport=46684].
Transmit: 536870912 bytes in 46010 milli-seconds = 11668.57 KB/sec (93348.56 Kbps).
```

This shows the speed between these hosts for this test is around 11.6 Megabytes per second.

---

pathchar

After writing traceroute, Van Jacobson then went on to write pathchar - an amazing tool that identifys network bottlenecks. It operates like traceroute, but rather than printing response time to each hop it prints bandwidth **between** each pair of hops.

```
# pathchar 192.168.1.10
pathchar to 192.168.1.1 (192.168.1.1)
 doing 32 probes at each of 64 to 1500 by 32
 0 localhost
 |     30 Mb/s,    79 us (562 us)
 1 neptune.drinks.com (192.168.2.1)
 |     44 Mb/s,    195 us (1.23 ms)
 2 mars.drinks.com (192.168.1.1)
2 hops, rtt 547 us (1.23 ms), bottleneck  30 Mb/s, pipe 7555 bytes
```

This tool works by sending "shaped" traffic over a long interval and carefully measuring the response times. It doesn't flood the network like TTCP does.

---

ntop

ntop is a tool that sniffs network traffic and provides comprehensive reports via a web interface. It is also available on sunfreeware.com.

```
# ntop
ntop v.1.3.1 MT [sparc-sun-solaris2.8] listening on [hme0,hme0:0,hme0:1].
Copyright 1998-2000 by Luca Deri <deri@ntop.org>
Get the freshest ntop from http://www.ntop.org/

Initialising...
Loading plugins (if any)...
WARNING: Unable to find the plugins/ directory.
Waiting for HTTP connections on port 3000...
Sniffying...
```

Now you connect via a web browser to localhost:3000.

By Process

How to monitor network usage by process. Recently the addition of DTrace to Solaris 10 has allowed the creation of the first network by process tools.

---

tcptop

This is a DTrace based tool from the freeware DTraceToolkit which gives a summary of TCP traffic by system and by process,

```
# tcptop 10
Sampling... Please wait.
2005 Jul  5 04:55:25,  load: 1.11,  TCPin:      2 Kb,  TCPout:     110 Kb

 UID    PID LADDR           LPORT FADDR           FPORT      SIZE NAME
 100  20876 192.168.1.5     36396 192.168.1.1        79      1160 finger
 100  20875 192.168.1.5     36395 192.168.1.1        79      1160 finger
 100  20878 192.168.1.5     36397 192.168.1.1        23      1303 telnet
 100  20877 192.168.1.5       859 192.168.1.1       514    115712 rcp
```

This version of tcptop will examine newly connected sessions (while tcptop has been running). In the above we can see PID and SIZE columns, this is tracking TCP traffic that has travelled on external interfaces. The TCPin and TCPout summaries also tracks localhost TCP traffic.

---

tcpsnoop

This is a DTrace based tool from the DTraceToolkit which prints TCP packets live by process,

```
# tcpsnoop
     UID    PID LADDR           LPORT DR RADDR           RPORT  SIZE CMD
     100  20892 192.168.1.5     36398 -> 192.168.1.1        79    54 finger
     100  20892 192.168.1.5     36398 <- 192.168.1.1        79    66 finger
     100  20892 192.168.1.5     36398 -> 192.168.1.1        79    54 finger
     100  20892 192.168.1.5     36398 -> 192.168.1.1        79    56 finger
     100  20892 192.168.1.5     36398 <- 192.168.1.1        79    54 finger
     100  20892 192.168.1.5     36398 <- 192.168.1.1        79   606 finger
     100  20892 192.168.1.5     36398 <- 192.168.1.1        79    54 finger
     100  20892 192.168.1.5     36398 <- 192.168.1.1        79    54 finger
     100  20892 192.168.1.5     36398 -> 192.168.1.1        79    54 finger
     100  20892 192.168.1.5     36398 -> 192.168.1.1        79    54 finger
```

```
     100  20892 192.168.1.5      36398 <- 192.168.1.1        79     54 finger
       0    242 192.168.1.5         23 <- 192.168.1.1     54224     54 inetd
       0    242 192.168.1.5         23 -> 192.168.1.1     54224     54 inetd
[...]
```

This version of tcpsnoop will examine newly connected sessions (while tcpsnoop has been running). In the above we can see a PID column and packet details, this is tracking TCP traffic that has travelled on external interfaces.

Back to Brendan Gregg's Homepage

Created: 09-Oct-2005
Last updated: 09-Oct-2005