Linux network metrics: why you should use nstat instead of netstat

Loïc Pefferkorn

TL;DR: This article is about the differences between **netstat** and **nstat** regarding Linux system network metrics, and why nstat is superior to netstat *(at least for this purpose.)*

Updates

- *2016-04-12 - note about ss command*

Network metrics with netstat

netstat can be found in the net-tools software collection. Depending on your linux Distribution, it may not be installed by default, like in Archlinux since 2011.

Below is the output of netstat –statistics on my system:

```
$ netstat --statistics
Ip:
    34151 total packets received
    0 forwarded
    0 incoming packets discarded
    34108 incoming packets delivered
```

```
    38436 requests sent out
Icmp:
    6 ICMP messages received
    0 input ICMP message failed.
    ICMP input histogram:
        destination unreachable: 6
    6 ICMP messages sent
    0 ICMP messages failed
    ICMP output histogram:
        destination unreachable: 6
IcmpMsg:
        InType3: 6
        OutType3: 6
Tcp:
    365 active connections openings
    0 passive connection openings
    17 failed connection attempts
    2 connection resets received
    14 connections established
    35389 segments received
    39132 segments send out
    83 segments retransmited
    1 bad segments received.
    156 resets sent
Udp:
    655 packets received
    1 packets to unknown port received.
    0 packet receive errors
    662 packets sent
    0 receive buffer errors
    0 send buffer errors
    IgnoredMulti: 7
UdpLite:
TcpExt:
```

```
    137 TCP sockets finished time wait in fast timer
    337 delayed acks sent
    Quick ack mode was activated 47 times
    3 packets directly queued to recvmsg prequeue.
    21584 packet headers predicted
    7317 acknowledgments not containing data payload receiv
    1128 predicted acknowledgments
    2 congestion windows recovered without slow start after
    19 other TCP timeouts
    TCPLossProbes: 20
    TCPLossProbeRecovery: 2
    47 DSACKs sent for old packets
    8 DSACKs received
    46 connections reset due to unexpected data
    2 connections reset due to early user close
    5 connections aborted due to timeout
    TCPDSACKIgnoredNoUndo: 6
    TCPRcvCoalesce: 6121
    TCPOFOQueue: 2421
    TCPChallengeACK: 1
    TCPSYNChallenge: 1
    TCPSpuriousRtxHostQueues: 14
    TCPAutoCorking: 1123
    TCPSynRetrans: 26
    TCPOrigDataSent: 16502
    TCPHystartTrainDetect: 1
    TCPHystartTrainCwnd: 16
    TCPKeepAlive: 1292
IpExt:
    InMcastPkts: 27
    OutMcastPkts: 2
    InBcastPkts: 7
    InOctets: 28620819
    OutOctets: 22032907
```

```
InMcastOctets: 864
OutMcastOctets: 64
InBcastOctets: 1202
InNoECTPkts: 34992
```

Some sections are standardized and based on RFCs MIB:

- Section **Ip, Icmp**: rfc2011 SNMPv2-MIB-IP
- Section **Tcp** rfc2012 SNMPv2-MIB-TCP
- Section **Udp** rfc2013 SNMPv2-MIB-UDP

To match netstat output with RFCs variables names, I did not find another way apart from reading netstat source code, especially statistics.c, where the relation are stored in arrays, extract:

```
{"Forwarding", N_("Forwarding is %s"), i_forward | I_STATIC
{"ForwDatagrams", N_("%llu forwarded"), number},
{"FragCreates", N_("%llu fragments created"), opt_number},
{"FragFails", N_("%llu fragments failed"), opt_number},
{"FragOKs", N_("%llu fragments received ok"), opt_number},
{"InAddrErrors", N_("%llu with invalid addresses"), opt_num
{"InDelivers", N_("%llu incoming packets delivered"), numbe
```

The remaining sections (TcpExt, IpExt, ...) are less rigid, as far as I know they have been added once someone has proven them useful.

net-tools is officially obsolete in favour of **iproute2**, quote from linuxfoundation.org

> *Please keep in mind that most net-tools programs are obsolete now*

Metrics with nstat

nstat is provided by the **iproute2** collection, which is usually also the name of the package in many Linux distributions. This package also provides the most well-known command **ip**

Extract of non-zero metrics:

```
$ nstat -a
#kernel
IpInReceives                    69783                  0.0
IpInDelivers                    69469                  0.0
IpOutRequests                   68643                  0.0
IcmpInMsgs                      6                      0.0
IcmpInDestUnreachs              6                      0.0
IcmpOutMsgs                     6                      0.0
IcmpOutDestUnreachs             6                      0.0
IcmpMsgInType3                  6                      0.0
IcmpMsgOutType3                 6                      0.0
TcpActiveOpens                  1011                   0.0
TcpAttemptFails                 37                     0.0
TcpEstabResets                  27                     0.0
TcpInSegs                       71580                  0.0
TcpOutSegs                      71010                  0.0
TcpRetransSegs                  410                    0.0
TcpInErrs                       4                      0.0
TcpOutRsts                      369                    0.0
UdpInDatagrams                  1348                   0.0
```

UdpNoPorts	1	0.0
UdpOutDatagrams	1366	0.0
UdpIgnoredMulti	47	0.0
Ip6InReceives	5236	0.0
Ip6InAddrErrors	421	0.0
Ip6InDelivers	4693	0.0
Ip6OutRequests	4913	0.0
Ip6InMcastPkts	780	0.0
Ip6OutMcastPkts	200	0.0
Ip6InOctets	3743259	0.0
Ip6OutOctets	710669	0.0
Ip6InMcastOctets	71232	0.0
Ip6OutMcastOctets	14384	0.0
Ip6InNoECTPkts	5725	0.0
Icmp6InMsgs	972	0.0
Icmp6InErrors	6	0.0
Icmp6OutMsgs	709	0.0
Icmp6InDestUnreachs	148	0.0
Icmp6InEchos	102	0.0
Icmp6InRouterAdvertisements	140	0.0
Icmp6InNeighborSolicits	521	0.0
Icmp6InNeighborAdvertisements	61	0.0
Icmp6OutDestUnreachs	148	0.0
Icmp6OutEchoReplies	102	0.0
Icmp6OutRouterSolicits	2	0.0
Icmp6OutNeighborSolicits	240	0.0
Icmp6OutNeighborAdvertisements	205	0.0
Icmp6OutMLDv2Reports	12	0.0
Icmp6InType1	148	0.0
Icmp6InType128	102	0.0
Icmp6InType134	140	0.0
Icmp6InType135	521	0.0
Icmp6InType136	61	0.0
Icmp6OutType1	148	0.0

Icmp6OutType129	102	0.0
Icmp6OutType133	2	0.0
Icmp6OutType135	240	0.0
Icmp6OutType136	205	0.0
Icmp6OutType143	12	0.0
Udp6InDatagrams	51	0.0
Udp6OutDatagrams	53	0.0
TcpExtTW	349	0.0
TcpExtDelayedACKs	811	0.0
TcpExtDelayedACKLost	137	0.0
TcpExtTCPPrequeued	14	0.0
TcpExtTCPHPHits	44384	0.0
TcpExtTCPPureAcks	10490	0.0
TcpExtTCPHPAcks	4460	0.0
TcpExtTCPLossUndo	5	0.0
TcpExtTCPSlowStartRetrans	4	0.0
TcpExtTCPTimeouts	113	0.0
TcpExtTCPLossProbes	46	0.0
TcpExtTCPLossProbeRecovery	2	0.0
TcpExtTCPDSACKOldSent	136	0.0
TcpExtTCPDSACKRecv	12	0.0
TcpExtTCPAbortOnData	101	0.0
TcpExtTCPAbortOnClose	21	0.0
TcpExtTCPAbortOnTimeout	23	0.0
TcpExtTCPDSACKIgnoredNoUndo	10	0.0
TcpExtTCPRcvCoalesce	15084	0.0
TcpExtTCPOFOQueue	5832	0.0
TcpExtTCPChallengeACK	4	0.0
TcpExtTCPSYNChallenge	4	0.0
TcpExtTCPSpuriousRtxHostQueues	224	0.0
TcpExtTCPAutoCorking	1242	0.0
TcpExtTCPSynRetrans	83	0.0
TcpExtTCPOrigDataSent	23610	0.0
TcpExtTCPHystartTrainDetect	3	0.0

```
TcpExtTCPHystartTrainCwnd       48                  0.0
TcpExtTCPKeepAlive              2528                0.0
IpExtInMcastPkts                157                 0.0
IpExtOutMcastPkts               2                   0.0
IpExtInBcastPkts                47                  0.0
IpExtInOctets                   67200127            0.0
IpExtOutOctets                  24997379            0.0
IpExtInMcastOctets              5024                0.0
IpExtOutMcastOctets             64                  0.0
IpExtInBcastOctets              8252                0.0
IpExtInNoECTPkts                74074               0.0
```

In addition to absolute values of counters given by the **-a** option, **nstat** can also provide a delta since its last execution, to ease live system debugging:

```
$ nstat
#kernel
IpInReceives                    1                   0.0
IpInDelivers                    1                   0.0
IpOutRequests                   1                   0.0
TcpInSegs                       1                   0.0
TcpOutSegs                      1                   0.0
TcpExtTCPOrigDataSent           1                   0.0
IpExtInOctets                   54                  0.0
IpExtOutOctets                  58                  0.0
IpExtInNoECTPkts                1                   0.0
$
```

All values, even the zero ones with **--zero**

```
$ nstat --zero
#kernel
IpInReceives                    2                   0.0
IpInHdrErrors                   0                   0.0
IpInAddrErrors                  0                   0.0
IpForwDatagrams                 0                   0.0
IpInUnknownProtos               0                   0.0
IpInDiscards                    0                   0.0
(...)
```

Finally, metrics can be displayed in JSON format, to ease their processing by all your fancy tools:

```
::2,:4,:2,:2,"Ip6OutRequests":4,"Ip6InOctets":776,"Ip6OutOc
```

Differences

Output

netstat appears more user-friendly by describing some metrics with plain English, while **nstat** displays raw information.

This can be considered as an advantage to roughly identify the purpose of the metric, but also a drawback if you are interested in the RFC name of the variable, going through netstat source code is hence a mandatory step.

Output comparison of 3 metrics:

```
# nstat
IpInReceives                    74923
IpOutRequests                   73128
IcmpInMsgs                      6


# netstat
Ip:
    74923 total packets received
    73128 requests sent out
Icmp:
    6 ICMP messages received
```

Parsing **nstat** output is also easier, even almost done thanks to the JSON output format option.

Metrics completeness

Both **netstat** and **nstat** read the metrics provided by the kernel through the **/proc** virtual filesystem:

```
$ strace -e open nstat 2>&1 > /dev/null|grep /proc
open("/proc/uptime", O_RDONLY)          = 4
open("/proc/net/netstat", O_RDONLY)     = 4
open("/proc/net/snmp6", O_RDONLY)       = 4
open("/proc/net/snmp", O_RDONLY)        = 4

$ strace -e open netstat -s 2>&1 > /dev/null|grep /proc
open("/proc/net/snmp", O_RDONLY)        = 3
open("/proc/net/netstat", O_RDONLY)     = 3
```

However, only nstat retrieves all the metrics provided by the kernel. Netstat seems to skip some of them, breakdown of metrics number per section:

	Netstat	Nstat	Difference
Ip	6	17	+11
Ip6	14	32	+18
Icmp	6	29	+23
Icmp6	25	46	+21
Tcp	10	10	0
Udp	7	8	+1
Udp6	4	8	+4
UdpLite	0	15	+15
UdpLite6	0	7	+7
TcpExt	48	116	+68
IpExt	11	17	+6

Why? Just because netstat maintains a static table of metrics entries, while nstat parses the whole /proc files. Since netstat is obsolete, new entries are not taken into account.

Note about ss command

ss is *another utility to investigate sockets* provided by **iproute2** package, like nstat.

Unlike netstat and nstat, **ss** does not provide system-wide network statistics, but is more oriented towards analysis of established sockets connections from many families

(raw, tcp, udp, Unix domain, dccp)

The only overall statistics option **--summary** is limited to the opened sockets:

```
$ ss --summary
Total: 433 (kernel 0)
TCP:   31 (estab 17, closed 1, orphaned 0, synrecv 0, timew

Transport Total     IP        IPv6
*         0         -         -
RAW       2         0         2
UDP       22        10        12
TCP       30        18        12
INET      54        28        26
FRAG      0         0         0
```

However **ss** is way more comprehensive when it comes to TCP connection internals, by reading */proc/net/tcp*.

For instance, for an established TCP connection you can retrieve almost every number that characterize the state of an established TCP connection:

```
$ ss --info --tcp|tail -1
         cubic wscale:7,7 rto:223.333 rtt:22.325/0.746 ato:
```

Every field will be explained in another blog post, but here you can recognize the congestion control algorithm **cubic**, various TCP timers **rto, rtt, ...**

Another super feature of **ss** is its filters based on the states of a connection, more handy than grepping *netstat* output:

```
STATE-FILTER
       STATE-FILTER allows to construct arbitrary set of st
       fier of state.
```

In addition to all the TCP states, others grouping keywords are possible:

```
       Available identifiers are:

              All standard TCP states: established, syn-sen
              and closing.

              all - for all the states

              connected - all the states except for listen

              synchronized - all the connected states excep

              bucket - states, which are maintained as mini

              big - opposite to bucket
```

The manpage provides useful examples:

```
       ss -o state established '( dport = :ssh or sport = :
              Display all established ssh connections.
```

```
ss -x src /tmp/.X11-unix/*
       Find all local processes connected to X serve

ss -o state fin-wait-1 '( sport = :http or sport = :
       List all the tcp sockets in state FIN-WAIT-1
```

Try that with netstat :)

Summary

- **nstat** offers all the linux network metrics provided by the kernel, but without any knowledge of the aforementioned RFCs their names might look more or less cryptic.
- **netstat** is obsolete and does not provide all the available metrics, but many are described with plain English, which is easier to understand when looking for simple metrics.
- If you want to extract every possible information on your established connections, **ss** is what you are looking for.

Plan

I plan to write another article to describe **every** metric provided by nstat, if you are interested please leave a comment.