Network Router Performance Testing How-To

Ray Patrick Soucy
rps@maine.edu
Senior Cyber Security Engineer
University of Maine System

While dedicated network test equipment is ideal, it is often cost-prohibitive. Further, the misuse of available open source tools, and misinterpretation of test data provided, often creates a situation where similar or even identical systems are described to have vastly different performance characteristics.

This how-to provides an overview of performance testing for network routers and firewalls using commodity hardware and available open source tools. The goal of this how-to is to provide network engineers and open source developers with a standard testing methodology which can be used for reproducible and relevant results for comparison of FOSS solutions against established commercial solutions.

Background

> ***Note:*** *It is important that this section is read and understood, as network performance testing can be performed using different methodologies and presented using different interpretations.*

When measuring network performance there are several measurements which represent different aspects of performance. The most common difference is what layer of the **OSI model** is being tested (Layer-2, Layer-3, or Layer-4-7).

The speed tests that most users are familiar with are typically a measurement of maximum data transfer. This is a Layer-4-7 test, measuring the data transfer over time without taking protocol overhead into account, and using the most efficient method for data transfer (large packets and, if using TCP, window scaling).

Similarly the throughput measurement reported by network devices can be different for the same flow(s) depending on the type of device or interface. A Layer-2 device or interface, such as a network switch port, will report a measurement of bits per second measured at Layer-2, while a network router or routed interface will report measurement at Layer-3. For large packet lengths the difference can be minimal, but for smaller packets the difference can be significant.

Test results for a network router or firewall are always a measurement of Layer-3 **bits per second** (bps) and will be the focus of this how-to.

As each packet requires processing on a router or firewall, a higher **packet per second** (pps) rate is required for smaller packets than larger ones to achieve the same throughput. There are several methods used for how to represent performance for different packet lengths.

Because normal network traffic is comprised of a mix of packets of various length, there have been efforts over time to represent this as an average for what is considered typical performance. The most common formula is known as **IMIX**, or Internet mix, an example of which can be seen in the table below.

Commonly used reference for Simple IMIX

Packet Len	Count	Distribution (Packet)	Bytes	Distribution (Byte)
40	7	58.3%	280	7%
576	4	33.3%	2304	56%
1500	1	8.3%	1500	37%

This results in an average packet length of 340-byte, but does not take into account the widespread adoption of TCP window scaling and explosion of video content on modern network traffic (which significantly reduces small packet count).

It should be noted that while the above reference is the most commonly cited, there is no agreed upon standard for IMIX. As one example, Cisco made use of average 410-byte packets for IMIX testing in 2010.

Today, a standard mix of traffic can be observed to have an average packet length of up to 740-bytes for normal usage, which is much different from the model described above.

More useful test results provide a range of packet lengths to allow network engineers to make decisions based on their individual use case. The most common measurements are for 64-byte, 128-byte, 256-byte, 512-byte, 1024-byte, and 1500-byte packets. Generating test results across a range of packet lengths will be the focus of this how-to.

Unfortunately, it has become common for marketing materials from major network equipment vendors to simply list test results of “Maximum” and “Typical”, often with the characteristics of the test methodology used for the latter not being disclosed.

Another standard practice for representing network performance in marketing literature is to represent throughput as the aggregate of all TX and RX taking place on the system, during the test. In effect, this counts each unidirectional flow twice, once as ingress traffic on an interface, and again as egress traffic on a different interface, representing what most would consider a 300 Mbps flow, as an 600 Mbps flow.

While the argument can be made that the goal is to measure the ability of the system to process packets, regardless of the interfaces used, it can be misleading to those who do not understand the methodology used, and often is not disclosed in marketing literature. As it is impossible to provide a comparison to vendor reported test results using undisclosed and independent test methodologies, this how-to will focus on performance testing of unidirectional flows through a router or firewall. These results can typically be doubled for comparison with industry marketing. Similarly, this how-to presents 512-byte test results to represent “typical” and 9000-byte packets “maximum”.

Test Environment

The most accessible and reliable utility for performance testing is **iperf3** on Linux, which will be the focus of this how-to.

Our test environment will be comprised of two nodes, a sender (client) and a receiver (server) to generate traffic through a router or firewall.

Hardware Selection

Hardware for the test environment must be carefully considered, and is the primary limiting factor.

A minimum of 8 x CPU cores is recommended for 10G testing. Ideally, the OS used for the test has all non-essential processes terminated to avoid CPU and IO contention during the test.

As a reference the test environment used for this how-to is a pair of Supermicro 5018A-FTN4 servers (8-core 2.4 GHz) outfitted with Intel 82599ES -based PCIe x8 10GbE NICs. The OS used is Ubuntu Server 16.04 LTS (Linux 4.4), and the standard Ubuntu package for iperf3 is used (version 3.0.11-1). As will be shown below, this test environment is not sufficient to perform 10 Gbps measurement at small packet lengths; this can be compensated with the addition of a network switch and additional test units, but is beyond the scope of this how-to.

System Tuning

For tuning purposes, network interfaces are optionally set to use an MTU of 9000 for jumbo-frame testing, and default buffers are increased by applying the following configuration to `sysctl.conf`:

```
net.core.rmem_max = 67108864
net.core.wmem_max = 67108864
net.ipv4.tcp_rmem = 4096 87380 33554432
net.ipv4.tcp_wmem = 4096 65536 33554432
net.ipv4.tcp_congestion_control = htcp
net.ipv4.tcp_mtu_probing = 1
```

These changes can be applied either using **`sysctrl -p`** or a reboot of the system.

Note that no kernel options are set for UDP. Our test environment uses TCP rather than UDP due to design choices for UDP processing on Linux which limit throughput in favor of reducing latency for applications (this is not the case for forwarding and filtering).

Baseline Test Results (Direct Node-to-Node)

Our test environment allows for testing using 1 Gbps interfaces or 10 Gbps interfaces. The results below show direct node-to-node testing to establish the maximum performance possible to measure using the test environment.

It should be noted that networking performance on Linux depends largely on interrupt processing. At 100% CPU core utilization, which is common for small packet lengths, test results will not be reliable. Counter-intuitively, the introduction of latency (even measured in the 1-10 microsecond range) can have a positive impact on these results, as the number of interrupts will be throttled, allowing for less CPU contention. For this reason, we will see better results than the baseline test for any test which has saturated CPU.

Also note the absence of 64-byte testing. This is a limitation of iperf3 in the inability to generate 64-byte packets when using TCP testing, which would require a 12-byte payload.

Because TCP is used for testing, only successful transfer will be shown in test results, with any policing or corrupted data being excluded (and significantly impacting results). This is the desired behavior for this test scenario.

Baseline Node-to-Node for Gigabit Interfaces (1000Base-T)

Packet Len (L3)	**L3 Throughput**	**L2 Throughput**
128	778.1	863.2
256	873.4	921.2
512	933.8	959.4
1024	966.8	980.0
1500	980.0	989.1
9000	1000.7	1002.3

Baseline Node-to-Node for 10 Gigabit Interfaces (10Base-SR)

** near 100% CPU utilization for testing at this packet length*

Packet Len (L3)	**L3 Throughput**	**L2 Throughput**
128	748.0 *	829.8 *
256	2389.3 *	2519.9 *
512	9276.9	9531.1
1024	9656.4	9788.4
1500	9765.4	9856.9
9000	9972.5	9988.4

Note that for smaller packet lengths the amount of overhead between Layer-2 and Layer-3 results becomes increasingly significant, while almost negligible for large packets.

Running and Understanding iperf3 Results

Because the most common use of iperf3 is to measure the speed of data transfer the reported results will be the bps of the data transfer (payload) rather than the Layer-3 transfer that is expected for discussing network router and firewall performance. To compensate for this overhead, at L3 and L2, the multipliers in the table below are used to include the calculated overhead for specific packet lengths.

Multipliers for conversion from iperf3 results to Layer-3 and Layer-2 measurements

Target Test (L3 length)	L3 overhead multiplier	L2 overhead multiplier
128	1.6842	1.8684
256	1.2549	1.3235
512	1.1130	1.1435
1024	1.0535	1.0679
1500	1.0359	1.0456
9000	1.0058	1.0074

Breakdown of packet overhead budget:

Ethernet Frame: 14 bytes
IP packet: 20 bytes
TCP packet: 20 bytes
TCP timestamps: 12 bytes (enabled by default on Linux)

Note that TCP timestamps are part of the TCP payload (MSS) but not part of the measured data rate. **This additional 12-byte overhead is often overlooked when interpreting iperf3 results.**

Because iperf3 will default to take advantage of large packets and TCP window scaling, it must be executed with specific options to force the desire packet length. This involves setting both the TCP MSS (option `-M`) and length (option `-l`) attributes, with the length attribute representing the payload (minus TCP timestamps) and the MSS option representing the payload and 12-bytes for the TCP timestamp option.

Configuration values for iperf3 TCP MSS (`-M`) and data length (`-l`)

Target Test	Frame Size (L2)	TCP MSS	Data Len
128	142	88	76
256	270	216	204
512	526	472	460
1024	1038	984	972
1500	1514	1460	1448
9000	9014	8960	8948

To get accurate test results, iperf3 should also be called with options to set a duration for the test (option `-t`), a reporting interval (option `-i`) matching the duration to avoid needless processing during the test, and the omit option to discard the first n seconds of test results to account for process start time (option `-O`).

For a 30 second test, we use the values `-t 31 -i 31 -O 1` (31 seconds minus 1).

A minimum of 30 seconds is recommended for test duration.

As an example the following command string would be used to call iperf3 for a 128-byte test:

```
iperf3 -f m -c $HOST -p 5201 -t 31 -i 31 -O 1 -M 88 -l 76 -P 8 -T 0 -A 0
```

Additional options:

`-f m`	Format (mbit)
`-c $HOST`	Server IP
`-p 5201`	Port Number
`-P 8`	Number of parallel client streams to run (single threaded)
`-T 0`	Report prefix (title)
`-A 0`	CPU affinity

In testing, using 8 parallel streams (part of the same loop of execution) produced the best results. Because iperf3 is single threaded, a separate instance of the client and server should be launched for each CPU core, using the `-A` option to set CPU affinity, and specifying a unique port for each process.

> ***Note:*** *Complete shell scripts to launch multiple client and server processes, and apply the appropriate multiplier to results, are included at the end of this how-to.*

The options for the corresponding server process should be as follows:

```
iperf3 -p 5201 -s -D -A 0
```

The reported result `SUM` in Mbps represents the data transfer for the payload and must be multiplied using the corresponding multiplies for L2 and L3 in the table above. For 128-byte testing the L3 multiplier is **1.6842**.

Example Test Results

Example test results using the same Supermicro 5018A-FTN4 hardware configuration as the test units running VyOS 1.1 (a Linux-based network router and firewall distribution), and entry-level routers from Cisco and Ubiquiti.

Example 1: Supermicro 5018A-FTN4

Supermicro 5018A-FTN4 using 10G interfaces (unidirectional)
** Limited by test environment*

Packet Len	Throughput (Mbps)
128	1720.51 *
256	2849.88 *
512	6626.13
1024	9648.59
1500	9762.74
9000	9971.50

Supermicro 5018A-FTN4 using 1G interfaces (unidirectional)

Packet Len	Throughput (Mbps)
128	757.72
256	872.46
512	933.81
1024	966.74
1500	981.26
9000	1002.4

Example 2: Cisco ISR 1921

Cisco 1921 with no services enabled, IP forwarding only

Packet Len	Throughput (Mbps)
128	185.84
256	371.88
512	801.68
1024	968.39
1500	979.61

Example 3: Ubiquiti EdgeRouter Lite

Ubiquiti ERL with offload enabled

Packet Len	Throughput (Mbps)
128	520.62
256	843.33
512	925.51
1024	965.12
1500	980.14

Comparison of Test Results to Commercial Specifications

As noted earlier, published performance numbers often provide the aggregate of RX and TX across all interfaces. To be more accurate in this regard, we should also be calculating and adding the throughput generated by TCP acknowledgment packets, but for most comparisons simply doubling the result to count both ingress and egress is will provide in-kind results that can be used for comparison.

For “maximum” we will use a packet length of 9000, and for “typical” a packet length of 512. Using the 10G test results from example 1, here are example results for basic forwarding and stateful packet inspection filtering:

Model	**Throughput Max**	**Throughput Typical**
Supermicro 5018A-FTN4	19.9 Gbps	13.2 Gbps

Cisco ASA model comparison (retrived from cisco.com 2016-11-14):

Model	**Throughput Max**	**Throughput Typical**
Cisco ASA 5525-X	2 Gbps	1 Gbps
Cisco ASA 5545-X	3 Gbps	1.5 Gbps
Cisco 5585-X with SSP10	4 Gbps	2 Gbps
Cisco 5585-X with SSP20	10 Gbps	5 Gbps
Cisco 5585-X with SSP40	20 Gbps	10 Gbps
Cisco 5585-X with SSP60	40 Gbps	20 Gbps

Here we can demonstrate that Linux on commodity hardware, can be competitive with high-end commercial solutions in terms of basic throughput and filtering, and the 5018A-FTN4 is comparable to the 5585-X SSP40 in performance (based on published information).

Appendix A: Client Test Script

```
#!/bin/bash

HOST="10.0.0.2"
SC="8"
TIME="30"
OMIT="3"
TMPFILE="/tmp/iperf3_results.txt"

case "$1" in
128)
    echo "Testing TCP using 128-byte IP packets (76-byte data, 142-byte Ethernet), $SC streams per CPU."
    TCPMSS="88"
    LEN="76"
    OVERHEAD_L3="1.6842"
    OVERHEAD_L2="1.8684"
    ;;
256)
    echo "Testing TCP using 256-byte IP packets (204-byte data, 270-byte Ethernet), $SC streams per CPU."
    TCPMSS="216"
    LEN="204"
    OVERHEAD_L3="1.2549"
    OVERHEAD_L2="1.3235"
    ;;
512)
    echo "Testing TCP using 512-byte IP packets (460-byte data, 526-byte Ethernet), $SC streams per CPU."
    TCPMSS="472"
    LEN="460"
    OVERHEAD_L3="1.1130"
    OVERHEAD_L2="1.1435"
    ;;
1024)
    echo "Testing TCP using 1024-byte IP packets (972-byte data, 1038-byte Ethernet), $SC streams per CPU."
    TCPMSS="984"
    LEN="972"
    OVERHEAD_L3="1.0535"
    OVERHEAD_L2="1.0679"
    ;;
1500)
    echo "Testing TCP using 1500-byte IP packets (1448-byte data, 1514-byte Ethernet), $SC streams per CPU."
    TCPMSS="1460"
    LEN="1448"
    OVERHEAD_L3="1.0359"
    OVERHEAD_L2="1.0456"
    ;;
9000)
    echo "Testing TCP using 9000-byte IP packets (8948-byte data, 9014-byte Ethernet), $SC streams per CPU."
    TCPMSS="8960"
    LEN="8948"
    OVERHEAD_L3="1.0058"
    OVERHEAD_L2="1.0074"
    ;;
*)
    echo "Usage $0 [128|256|512|1024|1500|9000]"
    exit 1
    ;;
esac
```

```
echo "" > $TMPFILE

echo "CPU 0 port 5201"
iperf3 -f m -c $HOST -p 5201 -t $(($TIME+$OMIT)) -i $(($TIME+$OMIT)) -l $LEN -M $TCPMSS -O $OMIT -A 0 -T 0 -P
$SC | grep SUM >> $TMPFILE &

echo "CPU 1 port 5202"
iperf3 -f m -c $HOST -p 5202 -t $(($TIME+$OMIT)) -i $(($TIME+$OMIT)) -l $LEN -M $TCPMSS -O $OMIT -A 1 -T 1 -P
$SC | grep SUM >> $TMPFILE &

echo "CPU 2 port 5203"
iperf3 -f m -c $HOST -p 5203 -t $(($TIME+$OMIT)) -i $(($TIME+$OMIT)) -l $LEN -M $TCPMSS -O $OMIT -A 2 -T 2 -P
$SC | grep SUM >> $TMPFILE &

echo "CPU 3 port 5204"
iperf3 -f m -c $HOST -p 5204 -t $(($TIME+$OMIT)) -i $(($TIME+$OMIT)) -l $LEN -M $TCPMSS -O $OMIT -A 3 -T 3 -P
$SC | grep SUM >> $TMPFILE &

echo "CPU 4 port 5205"
iperf3 -f m -c $HOST -p 5205 -t $(($TIME+$OMIT)) -i $(($TIME+$OMIT)) -l $LEN -M $TCPMSS -O $OMIT -A 4 -T 4 -P
$SC | grep SUM >> $TMPFILE &

echo "CPU 5 port 5206"
iperf3 -f m -c $HOST -p 5206 -t $(($TIME+$OMIT)) -i $(($TIME+$OMIT)) -l $LEN -M $TCPMSS -O $OMIT -A 5 -T 5 -P
$SC | grep SUM >> $TMPFILE &

echo "CPU 6 port 5207"
iperf3 -f m -c $HOST -p 5207 -t $(($TIME+$OMIT)) -i $(($TIME+$OMIT)) -l $LEN -M $TCPMSS -O $OMIT -A 6 -T 6 -P
$SC | grep SUM >> $TMPFILE &

echo "CPU 7 port 5208"
iperf3 -f m -c $HOST -p 5208 -t $(($TIME+$OMIT)) -i $(($TIME+$OMIT)) -l $LEN -M $TCPMSS -O $OMIT -A 7 -T 7 -P
$SC | grep SUM >> $TMPFILE

echo "Done"
sleep 1

TOTAL_DATA=$(cat $TMPFILE | grep receiver | gawk '{print $7}' | paste -s -d + - | bc)
TOTAL_L3=$(echo "$TOTAL_DATA*$OVERHEAD_L3" | bc)
TOTAL_L2=$(echo "$TOTAL_DATA*$OVERHEAD_L2" | bc)

echo "Total: $TOTAL_L3 Mbps IP traffic ($TOTAL_DATA data, $TOTAL_L2 Ethernet)"
```

Appendix B: Server Test Script

```
#!/bin/bash

case "$1" in
start)
    echo "Starting iperf3:"

    echo "CPU 0 port 5201"
    iperf3 -p 5201 -s -D -A 0

    echo "CPU 1 port 5202"
    iperf3 -p 5202 -s -D -A 1

    echo "CPU 2 port 5203"
    iperf3 -p 5203 -s -D -A 2

    echo "CPU 3 port 5204"
    iperf3 -p 5204 -s -D -A 3

    echo "CPU 4 port 5205"
    iperf3 -p 5205 -s -D -A 4

    echo "CPU 5 port 5206"
    iperf3 -p 5206 -s -D -A 5

    echo "CPU 6 port 5207"
    iperf3 -p 5207 -s -D -A 6

    echo "CPU 7 port 5208"
    iperf3 -p 5208 -s -D -A 7

    echo "Done"
    ;;
stop)
    echo "Stopping iperf3:"
    killall -9 iperf3
    echo "Done"
    ;;
*)
    echo "Usage: $0 [start|stop]"
    exit 1
    ;;
esac
```