MikroTik Solutions -n iPerf3

https://www.tangentsoft.com/mikrotik/dir/iperf3

The `Containerfile` builds a single static binary of ESNet's iperf3 tool and sets it to run in server mode by default. Its small (~0.2 MiB) size makes it ideal for testing network performance on resource constrained MikroTik RouterOS boxes running its optional container feature. As such, it's built for both 32-bit and 64-bit ARM CPUs, but also for modern Intel platforms, because why not?[1]

See the `Makefile` for further details on building, configuring, and running this container.

Simple Method

Start by installing the container package per MikroTik's docs, but in place of their overcomplicated network setup, put the veth on the bridge, then say:

```
> /container
> add remote-image=quay.io/tangent/iperf3:latest \
  interface=veth1 \
  start-on-boot=yes \
  logging=yes
> start 0
```

If your device is configured as a router with a firewall, you may need to add something like this:

```
/ip firewall filter
add place-before=0 protocol=tcp dst-port=5201 \
    action=accept chain=forward out-bridge-port=veth1
add place-before=0 protocol=udp dst-port=5201 \
    action=accept chain=forward out-bridge-port=veth1
```

Yes, believe it or not, this container is accessed through the "forward" chain even though it's bound to the main bridge and listening on an interface that belongs to the router itself. Moving this rule to the "input" chain will cause it to have no effect.

Remote Tarball Method

If you need to install the container via an image tarball, the simplest way to fetch it is:

```
$ docker pull --platform linux/arm/v7 quay.io/tangent/iperf3:latest
$ docker image save quay.io/tangent/iperf3:latest > iperf3.tar
$ scp iperf3.tar myrouter:
```

That assumes you've got a 32-bit ARM based router such as the RB4011 and it's got SSH with keys set already. For 64-bit routers, change the `--platform` argument to `linux/arm64`.

Source Method

You can instead build the container from this source repo:

```
$ fossil clone https://tangentsoft.com/mikrotik
$ cd mikrotik/iperf3
$ make PLATFORMS=linux/arm64 && scp iperf3.tar myrouter:
```

Explicitly setting the `PLATFORM` like that causes it to build for that one CPU type, not all four as it will by default. That not only makes the build go much faster,[2] it is necessary to make the tarball unpack on RouterOS, which doesn't currently understand how to disentangle multi-platform image tarballs.

You can use any platform name here supported by your container builder. We prefer Docker for this since although cross-compilation *can* be done with Podman, it doesn't work out of the box, and it's fiddly besides.[3]

Test Results

Here are the best test results I've achieved for a few representative ARM-based RouterOS boxes I have at hand here, running the above container:

Device	Forward	Reverse	Parallel
RB4011	4180	6670	9350
hAP ax lite	919	798	885
CRS328	225	499	395

The "Forward" test is a simple run of "`iperf3 -c`" against the container's veth IP, with results given in Mbit/sec.

The "Reverse" test adds the "`-R`" option, which makes the router send the packets back to the client. Notice that it sometimes helps, sometimes not.

The "Parallel" test adds "`-P2`" or "`-P4`" to that, depending on whether it's a dual or quad-core processor, bringing them all to bear on the test. Notice that this allows the RB4011 to fill the 10G pipe when the client is powerful enough to keep up its end.

Contrast the CRS328, which has a single-core CPU. Passing "`-P1`" would've simply repeated the prior test, so I passed `-P4` to show how running this test in parallel makes it worse due to all the context switching it incurs. Much the same occurs with the others: giving a quad-core CPU eight `iperf3` streams makes it perform worse.

The test setup looks like so:

MBP M1 (2020) (Sonnet TB4 to SFP+) OM4 CRS328-24P (generic MMF LC SFP+) OM4 RB4011 (generic MMF LC SFP+)

```
thickness = 0.01
color = 0x454545
fill = 0xF0F5F9
boxrad = 0.05

box "MBP M1 (2020)" "(Sonnet TB4 to SFP+)" small fit
arrow "OM4" above
box "CRS328-24P" "(generic MMF LC SFP+)" small fit
arrow "OM4" above
box "RB4011" "(generic MMF LC SFP+)" small fit
```

→ /pikchrshow

The 10G test client has an 8-core Apple M1 CPU, ensuring the device under test (the RB4011 in the diagram above) is always the bottleneck. GigE over Cat-5 is the usual cause for copper-wired routers, whereas for the fiber connected devices, it's the CPU.

If you're wondering why the CRS328 does so badly on this test even though it's in the middle for all the other tests, it's because when it is running `iperf3` itself, all of the test traffic is sourced or sunk in software running on the single-core 32-bit 800 MHz ARM CPU it's saddled with, but in the other tests, it's acting as a switch, passing the traffic in hardware, as it was designed to do.