NDN iptables match extension

L. Bracciale, A. Detti, P. Loreti, G. Rossi, N. Blefari Melazzi

May 3, 2017

This module implements a match extension for netfilter[1] to match only certain NDN packets according to the specific parameters passed to the kernel module. The module is build using `xtables-addons`[2] so to avoid to patch or recompile iptables or the whole kernel.

This document is a technical annex to the paper:
L. Bracciale, A. Detti, P. Loreti, G. Rossi, N. Blefari Melazzi, **Cluster-based Scalable ICN Router**.

1 Basic usage

Example of usage:

```
iptables -t mangle -A FORWARD -m ndn --only-interest
-j LOG --log-prefix "An interest passed" --log-level 4
```

This iptables rule will log every interest that is being forwarded by the machine

```
iptables -t mangle -A FORWARD -m ndn --hash 3:5 -j LOG
```

This iptables rule will log every NDN interest (whose MaxSuffix is > 1) or data packet whose name hashed in the range 1-5 is equal to 3. For example when an interest passes with the name `CONTENT1`, the name `CONTENT1` will be hashed producing a number from 1 to 5; then if this number is equal to 3, we have a match.
The syntax of the parameters of the `ndn` matching module is the following:

`--exact-match`	Match against NDN packets with MaxSuffix > 1
`--hash n:nmax`	Match the n-th hash over nmax hashes. n must be major equals to 1, and minor-equal to nmax
`--only-interest`	Match only interest packets
`--only-data`	Match only data packets
`--prefix [prefix-name]`	Match packets whose name start with a given prefix-name

[1] `https://www.netfilter.org/`
[2] `http://xtables-addons.sourceforge.net/`

2 Compiling

First we need to compile `xtables addons`, following the instruction on the related website [3]. On Ubuntu 16.04 we needed to install the following packages:

```
git build-essential libtool autoconf  autotools-dev  libxtables11 libxtables-dev
pkg-config conntrack iptables-dev
```

Then we need to clone the `xtables addons` repository. In this case we can clone our fork containing the ndn match extension through the command:

```
git clone https://lorenzobracciale@bitbucket.org/icn2020/scalable-router.git
```

Then we need to install the `xtables addons` using the usual autoconf procedure:

```
autoconf
./configure
make
make install
```

Then we need to reload the kernel modules and load the xt_ndn module together with conntrack for NAT and fragment reconstruction:

```
depmod -a
modprobe xt_ndn
modprobe ip_conntrack
modprobe nf_conntrack_ipv4
```

3 Example: creating a cluster of 2 Nodes

In the follow example we refer to the network architecture depicted in figure 1. In this example we consider the case of 2 internal routers (192.168.209.201 and 192.168.209.202) whose NFD instances are bound respectively on the UDP ports 6364 and 6365. The following commands must be executed on the load balancer.

First we need to load the kernel modules (if not already loaded), and we also need to set the tc qdisc to make stateless SNAT or DNAT [4].

```
# load modules
modprobe xt_ndn
modprobe -i act_nat
modprobe xt_mark

# create qdiscs
```

[3] `http://xtables-addons.sourceforge.net/`
[4] `http://man7.org/linux/man-pages/man8/tc-nat.8.html`

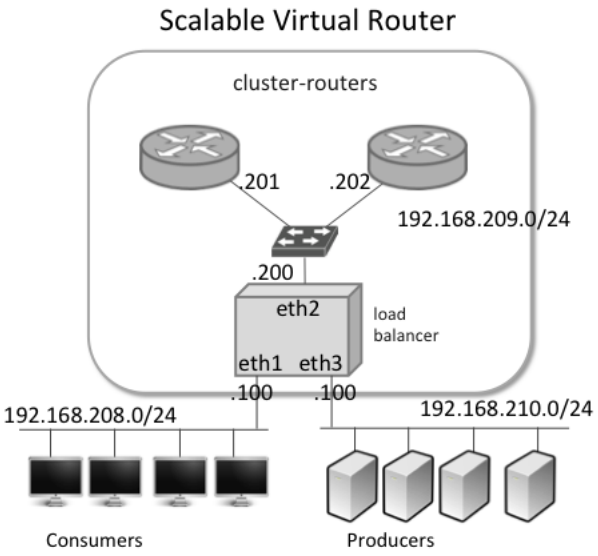

Figure 1: Reference network architecture for the presented example

```
tc qdisc add dev eth1 parent root handle 1: htb
tc qdisc add dev eth3 parent root handle 1: htb
tc qdisc add dev veth0 parent root handle 1: htb
```

Then we need to insert a set of rules to DNAT toward a different IP address all the packets arriving from the Internet and directed to the load balancer on the selected ports. In this case we used 192.168.209.253 that corresponds to a fake address in the cluster network (there is no machine associated with that IP). This is a way we process the packet in the FORWARD chain and not in the INPUT chain. Then in the *mangle* table we mark all the interest packet with 1 or 2 using the *NDN kernel module* that matches if the name hash match respectively 1 or 2 in the hash domain $[1, 2]$. As for the data packets, we mark with 1 those who come with a destination port 6364 and with 2 the ones coming with a destination port 6365. All the packets marked with 1 will be redirected (DNAT) to 192.168.209.201, and all the packet marked with 2 will be redirected to 192.168.209.202. This is done by the last two `tc` instructions that perform the stateless NAT.

```
###### Packet from internet ######

iptables -t nat -A PREROUTING -i eth1 -p udp --destination-port 6363 \
```

```
                        -j DNAT --to-destination 192.168.209.253
iptables -t nat -A PREROUTING -i eth3 -p udp --destination-port 6363 \
                        -j DNAT --to-destination 192.168.209.253
iptables -t nat -A PREROUTING -i eth1 -p udp --destination-port 6364 \
                        -j DNAT --to-destination 192.168.209.253
iptables -t nat -A PREROUTING -i eth3 -p udp --destination-port 6364 \
                        -j DNAT --to-destination 192.168.209.253
iptables -t nat -A PREROUTING -i eth1 -p udp --destination-port 6365 \
                        -j DNAT --to-destination 192.168.209.253
iptables -t nat -A PREROUTING -i eth3 -p udp --destination-port 6365 \
                        -j DNAT --to-destination 192.168.209.253

# load balancing
# we force each packet in the forward chain to pass to the ndn filter
# module that marks accordingly the packet.
# Packet with a given mark will be d-natted to the assigned cluster
# node by egress qdisc (tc nat).

iptables -t mangle -A FORWARD -d 192.168.209.253  -p udp --destination-port 6363 \
                        -m ndn --only-interest --hash 1:2 -j MARK --set-mark 1
iptables -t mangle -A FORWARD -d 192.168.209.253  -p udp --destination-port 6363 \
                        -m ndn --only-interest --hash 1:2 -j ACCEPT

iptables -t mangle -A FORWARD -d 192.168.209.253  -p udp --destination-port 6363 \
                        -m ndn --only-interest --hash 2:2 -j MARK --set-mark 2
iptables -t mangle -A FORWARD -d 192.168.209.253  -p udp --destination-port 6363 \
                        -m ndn --only-interest --hash 2:2 -j ACCEPT


iptables -t mangle -A FORWARD -d 192.168.209.253  -p udp --destination-port 6364 \
                        -m ndn --only-data -j MARK --set-mark 1
iptables -t mangle -A FORWARD -d 192.168.209.253  -p udp --destination-port 6364 \
                        -m ndn --only-data -j ACCEPT

iptables -t mangle -A FORWARD -d 192.168.209.253  -p udp --destination-port 6365 \
                        -m ndn --only-data -j MARK --set-mark 2
iptables -t mangle -A FORWARD -d 192.168.209.253  -p udp --destination-port 6365 \
                        -m ndn --only-data -j ACCEPT

tc filter add dev veth0 parent 1: protocol ip prio 1 handle 1 fw action \
                        nat ingress 192.168.209.253 192.168.209.201
tc filter add dev veth0 parent 1: protocol ip prio 1 handle 2 fw action \
                        nat ingress 192.168.209.253 192.168.209.202
```

Then, we just need to insert the rules that apply a source NAT on all the packets coming from the internal routers and directed towards the Internet, substituting the source IP address with the one of the public interface of the load balancer (in this case 192.168.208.100 or 192.168.210.100, depending on the output interface).

```
###### Packet from cluster to the internet ######
```

```
iptables -t mangle -A FORWARD  -i veth0 -p udp --source-port 6364 \
                        -j MARK --set-mark 100
iptables -t mangle -A FORWARD  -i veth0 -p udp --source-port 6365 \
                        -j MARK --set-mark 100

iptables -t mangle -A FORWARD  -j ACCEPT

iptables -t nat -A POSTROUTING -m mark --mark 100 -o eth1 \
                        -j SNAT --to-source 192.168.208.100
iptables -t nat -A POSTROUTING -m mark --mark 100 -o eth3 \
                        -j SNAT --to-source 192.168.210.100
```

`veth0` is a virtual interface that we used to circumvent the problem that `TC` stateless destination natting performed on the `INGRESS` does not change the destination MAC address of the packet. We solve this issue by using linux network namespaces and setting up two virtual interfaces, namely `veth0` and `veth1` on two different namespaces. We set the IP address of those interfaces to be respectively 10.0.0.1 and 10.0.0.2. In the first namespace we executed all the rules reported so far and we set a routing entry 192.168.209.0/24 to the gateway 10.0.0.2. In the other namespace we have two network interfaces: `veth1` and `eth2` (the "real" network interface towards the subnet of the internal routers). In that namespace we just need to pass the packets between the two interfaces without any further packet manipulation or iptables/tc rules.

```
ip netns add mynamespace
ip link add veth0 type veth peer name veth1
ip netns set mynamespace veth1
ip link set veth1 netns mynamespace
ip a a 10.0.0.1/30 dev veth0
ip link set veth0 up
ip netns exec mynamespace ip a a 10.0.0.2/30 dev veth1
ip netns exec mynamespace ip link set veth1 up
ip link set eth2 netns mynamespace
ip netns exec mynamespace ip link set eth2 up
ip netns exec mynamespace ip a a 192.168.209.200/24 dev eth2
ip r a 192.168.209.0/24 via 10.0.0.2
ip netns exec mynamespace ip r a default via 10.0.0.1
```

Finally, on each internal router we just set the following rule, respectively :

```
iptables -t nat -A PREROUTING -p udp -d 192.168.209.201 --destination-port 6363 \
                        --j REDIRECT --to-port 6364
```

and

```
iptables -t nat -A PREROUTING -p udp -d 192.168.209.202 --destination-port 6363 \
                       --j REDIRECT --to-port 6365
```