Benchmarking NFTables

Red Hat
Phil Sutter <psutter@redhat.com>

NFWS 2017

Testing Environment

- Two netns on same system
- Add non-matching rules on ingress side
- Run iperf (single stream TCP, affinity on single core)
- Repeat runs for better results
- Add rules for scalability dimension

Tested Setups

1. Match single source IP per rule
2. Match multiple destination ports per rule
3. Match both source IP and destination port
4. Have user-defined chain per IP
5. Drop source IPs efficiently

Tested Setups

1. Match single source IP per rule

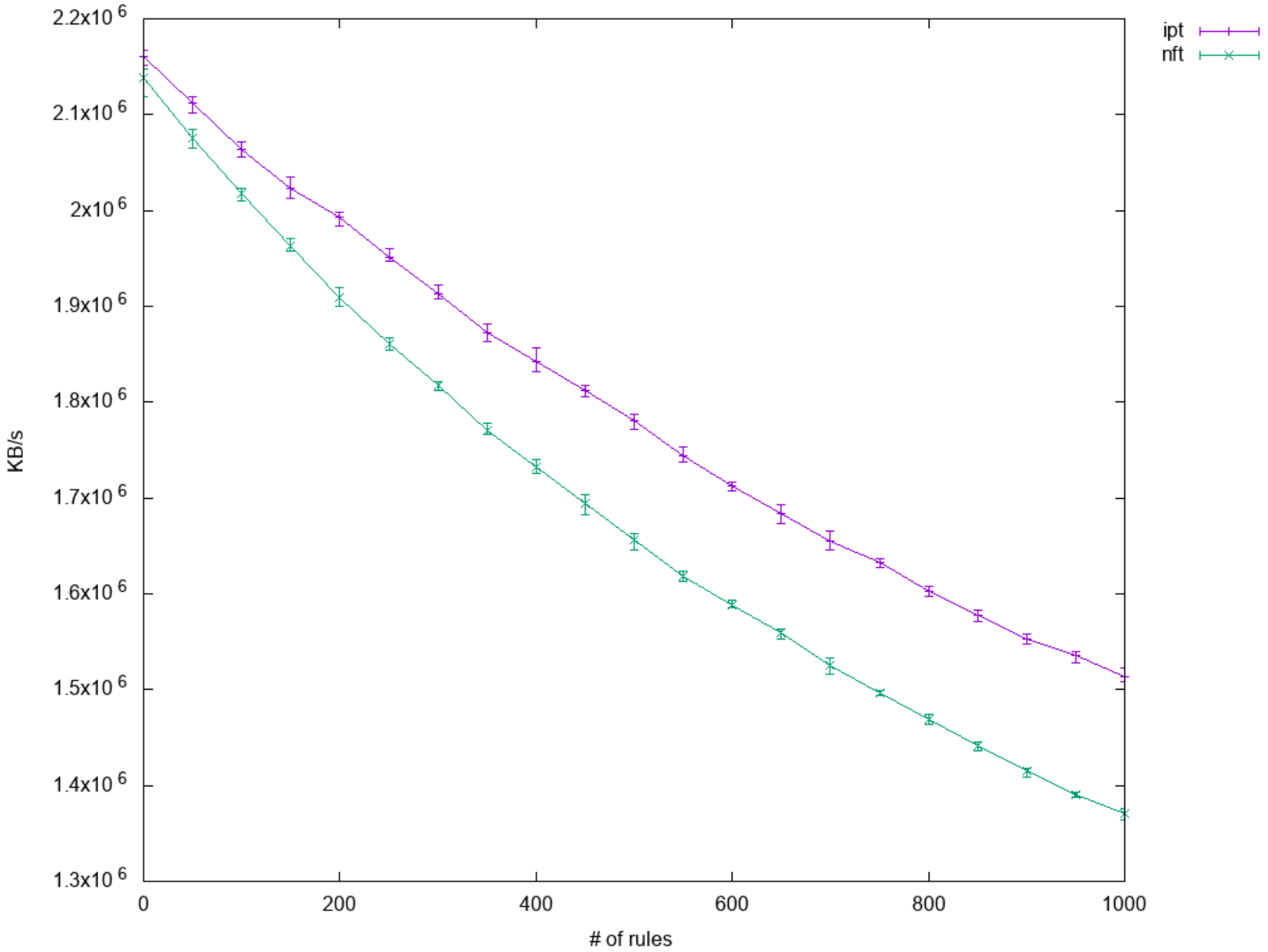
1) iptables Variant

```
if [[ $SCALE -eq 0 ]]; then
    iptables -N test
    iptables -A INPUT -j test
    iptables -A INPUT -j ACCEPT
else
    for j in {1..10}; do
        iptables -A test -s 10.11.$SCALE.$j -j DROP
    done
fi
```

1) nftables Variant

```
if [[ $SCALE -eq 0 ]]; then
    nft add table ip t
    nft add chain ip t c { \
        type filter hook input priority 0; \
    }
    nft add chain ip t test
    nft add rule ip t c jump test
    nft add rule ip t c accept
else
    for j in {1..10}; do
        nft add rule ip t test \
            ip saddr 10.11.$SCALE.$j drop
    done
fi
```

1) Results


2.2x10 6
2.1x10 6
2x10 6
1.9x10 6
1.8x10 6
1.7x10 6
1.6x10 6
1.5x10 6
1.4x10 6
1.3x10 6
KB/s
0
200
400
600
800
1000
# of rules
ipt
nft

Tested Setups

2. Match multiple destination ports per rule

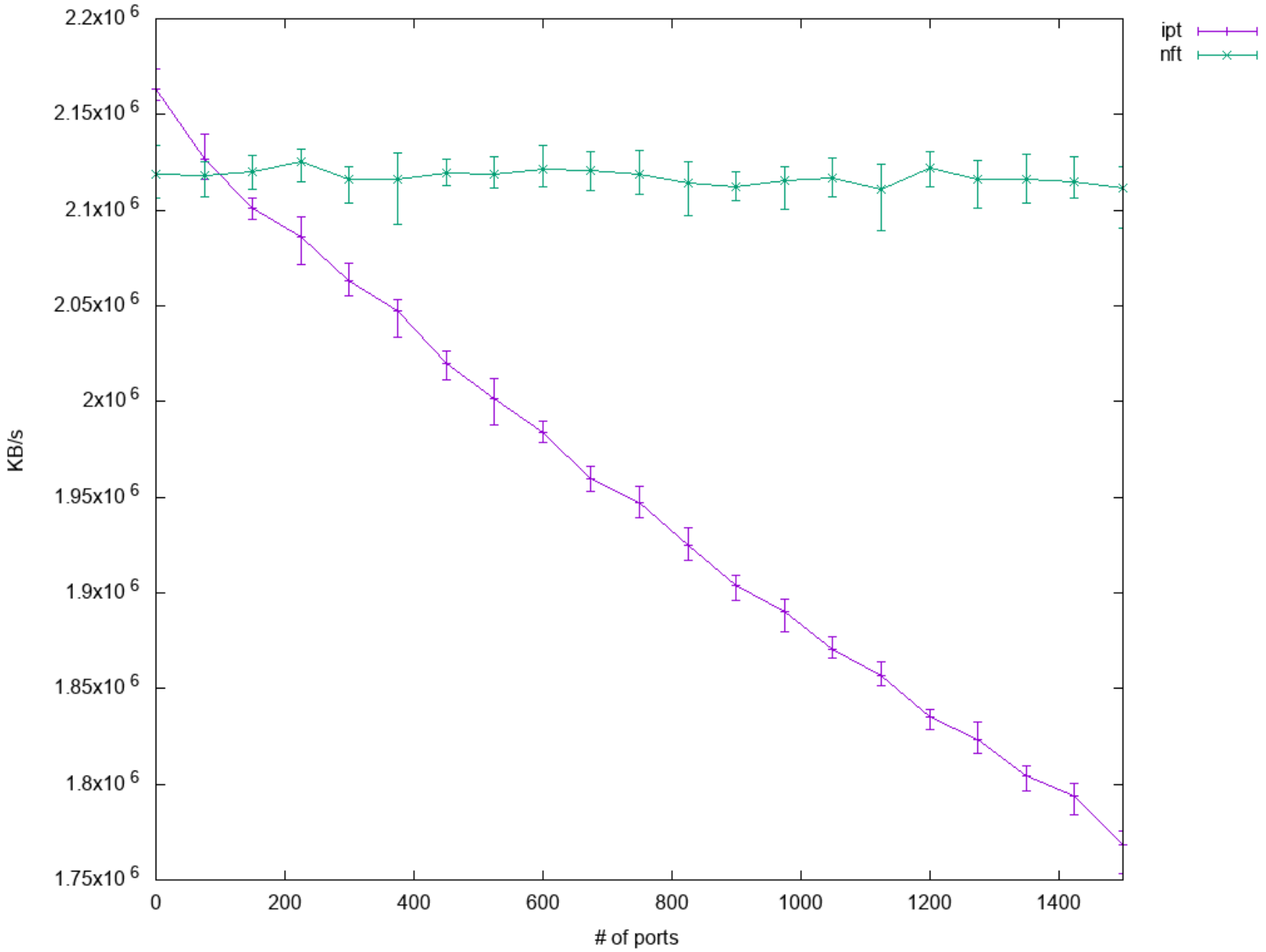
2) iptables Variant

```
for ((j = 100; j < 1600; j += 100)); do
    ports+="${sep}$((SCALE + j))"
    sep=","
done
iptables -A test -p tcp \
    -m multiport --dports $ports -j DROP
```

2) nftables Variant

```
if [[ $SCALE -eq 0 ]]; then
    [...]
    nft add set ip t ports '{ type inet_service; }'
    nft add rule ip t test tcp dport @ports drop
else
   sep=""
   ports=""
   for ((j = 100; j < 1600; j += 100)); do
           ports+="${sep}$((SCALE + j))"
           sep=","
   done
   nft add element ip t ports '{' $ports '}'
fi
```

2) Results


ipt
nft
KB/s
2.2x10 6
2.15x10 6
2.1x10 6
2.05x10 6
2x10 6
1.95x10 6
1.9x10 6
1.85x10 6
1.8x10 6
1.75x10 6
0
200
400
600
800
1000
1200
1400
# of ports

Tested Setups

3. Match both source IP and destination port

3) iptables Variant

```
for j in {1..10}; do
    iptables -A test -s 10.11.$SCALE.$j \
        -p tcp --dport $j -j DROP
done
```

3) `nftables` Variant (Bad)

```
for j in {1..10}; do
    nft add rule ip t test ip saddr . tcp dport \
        '{' 10.11.$SCALE.$j . $j '}' drop
done
```

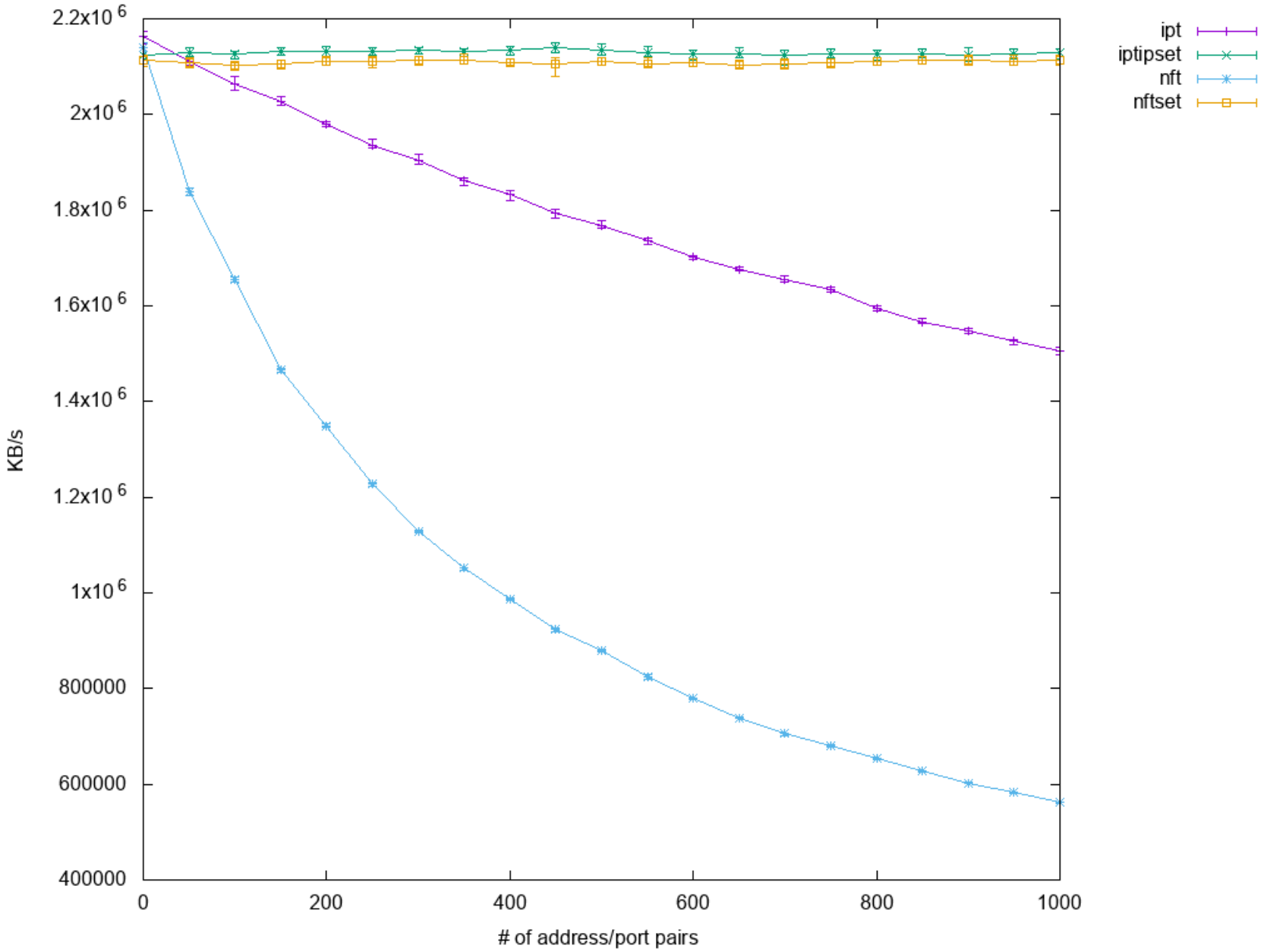
3) nftables Variant (Better)

```
if [[ $SCALE -eq 0 ]]; then
    [...]
    nft add set ip t saddr_port \
        '{ type ipv4_addr . inet_service; }'
    nft add rule ip t test \
        ip saddr . tcp dport @saddr_port drop
else
    for j in {1..10}; do
        nft add element ip t \
            saddr_port '{' 10.11.$SCALE.$j . $j '}'
    done
fi
```

3) ipset Variant

```
if [[ $SCALE -eq 0 ]]; then
    [...]
    ipset create saddr_port hash:ip,port
    iptables -A test -m set ! --update-counters \
        --match-set saddr_port src,src -j DROP
else
    for j in {1..10}; do
        ipset add saddr_port 10.11.$SCALE.$j,$j
    done
fi
```

3) Results


2.2x10 6
2x10 6
1.8x10 6
1.6x10 6
1.4x10 6
1.2x10 6
1x10 6
800000
600000
400000
KB/s
0
200
400
600
800
1000
# of address/port pairs
ipt
iptipset
nft
nftset

Tested Setups

4. Have user-defined chain per IP

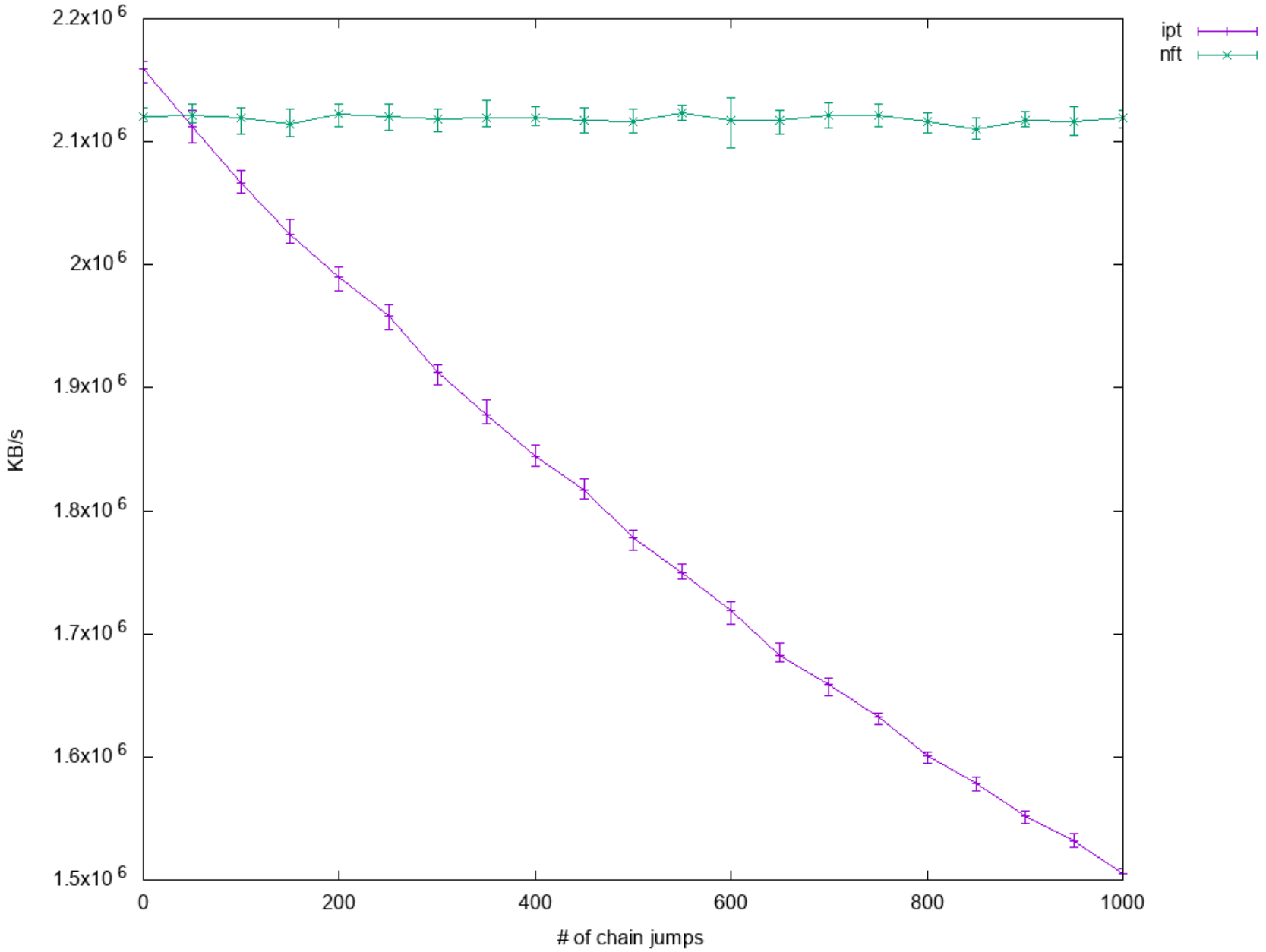
4) iptables Variant

```
for j in {1..10}; do
    iptables -N test_${SCALE}_$j
    iptables -A test_${SCALE}_$j -j DROP
    iptables -A test -s 10.11.$SCALE.$j \
        -j test_${SCALE}_$j
done
```

4) nftables Variant

```
if [[ $SCALE -eq 0 ]]; then
    [...]
    nft add map ip t testmap \
        '{ type ipv4_addr : verdict; }'
    nft add rule ip t test ip saddr vmap @testmap
else
    for j in {1..10}; do
        nft add chain ip t test_${SCALE}_$j
        nft add rule ip t test_${SCALE}_$j drop
        nft add element ip t testmap \
            '{' 10.11.$SCALE.$j : \
                jump test_${SCALE}_$j '}'
    done
fi
```

4) Results


2.2x10 6
2.1x10 6
2x10 6
1.9x10 6
1.8x10 6
1.7x10 6
1.6x10 6
1.5x10 6
KB/s
0
200
400
600
800
1000
# of chain jumps
ipt
nft

Tested Setups

5 Drop source IPs efficiently

5) Multiple Options

- `iptables`
  - `ipset` or not
  - INPUT or PREROUTING (mangle)
- `nftables`
  - named set or not
  - filter or prerouting hook
  - netdev table
- `tc`

⇒ `nftables` input dropped from tests

⇒ match interface as well

⇒ parallel stream of matched packets

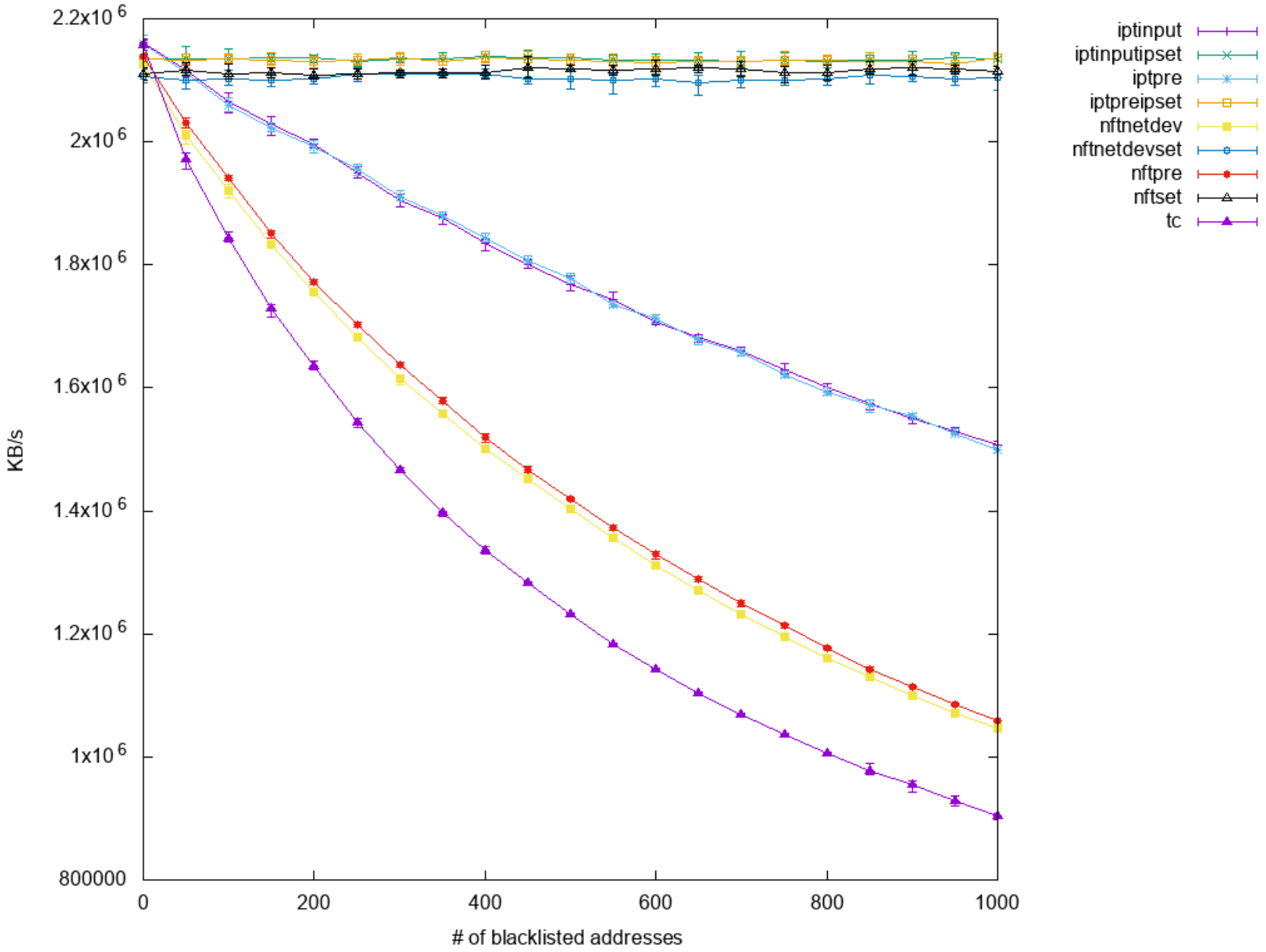
5) tc Variant

```
if [[ $SCALE -eq 0 ]]; then
    tc qd add dev enp3s0 handle ffff: ingress
else
    for j in {1..10}; do
        tc filter add dev $iface parent ffff: \
            protocol ip u32 match ip \
            src 10.11.$SCALE.$j action drop
    done
fi
```

5) nftables netdev Variant

```
if [[ $SCALE -eq 0 ]]; then
    nft add table netdev t
    nft add chain netdev t c '{ \
        type filter hook ingress \
        device enp3s0 priority 0; \
    }'
    nft add chain netdev t test
    nft add rule netdev t c jump test
    nft add rule netdev t c accept
else
    for j in {1..10}; do
        nft add rule netdev t test \
            ip saddr 10.11.$SCALE.$j drop
    done
fi
```

5) Results


2.2x10 6
2x10 6
1.8x10 6
1.6x10 6
1.4x10 6
1.2x10 6
1x10 6
800000
KB/s
0
200
400
600
800
1000
# of blacklisted addresses
iptinput
iptinputipset
iptpre
iptpreipset
nftnetdev
nftnetdevset
nftpre
nftset
tc

Conclusions

- Similar setups perform similarly, `iptables` slightly better
- `ipset` makes things scalable, `nftables` applies same concept broadly
- Using `tc` provides no benefit at all?!?