Random Walks Using IP Forwarding

Antonio Carzaniga

29 March 2021

I should start by admitting that this is a hack. But hey, hacking is fun, and you learn a lot. So, here we go. What we want to do is to sample the nodes in the network (routers). This means selecting a node at random. More specifically, we want to do that through a random walk. Of course we could do this at the application level, but we are interested in experimenting with *advanced networking,* so we want the network itself to do the sampling for us.

In practice, we want to "ping" a special "random" address, and we want the network—meaning the router that perform IP forwarding—to treat that SMTP Echo request as a random walk.

The main ingredients of our hack will be two interesting features of the networking layer of the Linux operating system, namely iptables (one word) and policy routing, and a basic feature of IP forwarding, namely ICMP Time Exceeded messages.

As usual, we do this over an emulated "mininet" network.

Random Forwarding

We start by reserving a special address as the "random" destination. In the examples in this document we use address `10.255.255.255`.

We therefore want to forward datagrams addressed to `10.255.255.255` so as to make them follow a random walk throughout the network. If we could somehow program the routers—which we will do some other time—we would write a simple program that says something line this: "if the destination is `10.255.255.255`, then pick an output port at random, and then forward the packet through that port." This would be a principled way to implement a random walk in an IP network. But we don't have this flexibility in programming, and therefore we will have to hack something up using the 9networking tools we have at our disposal, which means the standard packet processing features provided by our routers, which happen to be Linux hosts.

The Linux kernel implements a sophisticated packet processing system. However, things are simple enough at a very high-level. If a packet needs to be forwarded along—that is, if the packet isn't destined to a local network address—then the kernel looks up the destination address, finds an appropriate matching destination network in a forwarding table, and forwards the packet to the corresponding port specified in the table.

In addition to this basic processing, the kernel applies series of processing rules before and after the forwarding lookup, and furthermore allows for special routing "policies" to take effect. In essence, all those feature are made up of match-action rules. These are the feature we use to implement our random walks.

Iptables

Iptables are a mechanism designed to filter and/or modify packets. The main purpose is to implement network firewalls and network-address translation. However, other extensions allow for randomized matching conditions, as well as actions that attach a special label to the packet that can be later used for policy-based routing.

There are several "tables" that do different things. We will use a table called `mangle`, which allows for packet marking. Each table has different "chains" of rules. The one that interests us is the `PREROUTING` chain. As the name suggests, the rules in the `PREROUTING` chain are applied *before* the forwarding lookup. We therefore add rules with a command:

`iptables -t mangle -A PREROUTING` *match-conditions* `-j` *action*

The matching conditions must specify some fixed parameters. We want to match ICMP packets with destination address `10.255.255.255`. So, we write `-p icmp -d 10.255.255.255`. Then we want to choose at random whether to mark the packet with label `1` to then forward to next-hop `1` , or `2` for next-hop `2`, etc. We can do that with a special matching extension called `statistics` in `random` mode, so we write `-m statistic --mode random --probability 0.5`. With all these conditions, we want to execute (or "jump") to a `MARK` action (or "target").

To put everything together, imagine we want to configure router `R1` to choose at random between router `R2` and `R3`. We therefore write the following two iptables rules:

```
# ip netns exec R1 \
    iptables -t mangle -A PREROUTING \
      -p icmp -d 10.255.255.255 -m statistics --mode random --probability 0.5 \
      -j MARK --set-mark 2
# ip netns exec R1 \
    iptables -t mangle -A PREROUTING \
       -p icmp -d 10.255.255.255 -m mark --mark 0x0 \
       -j MARK --set-mark 3
```

Notice that the first rule matches an ICMP packet with destination `10.255.255.255` with probability 0.5, and when it matches it marks the packet with label `2`. The second rule, which is considered right after the first one, matches the same ICMP packets with the same destination, but only if they are marked `0` (default). In other words, the second rule matches all ICMP packets with the "random" destination that did not match the previous rule. As a result, each ICMP packets with the "random" destination gets marked `2` with probability 0.5, or it gets marked `3` (with probability 0.5).

Policy Routing

Once we mark the packets, we need to apply some special lookup tables so that the ones marked `0` are forwarded normally (default) while the ones marked `2` and `3` are sent to neighbor routers `R2` and `R3`, respectively. We must therefore create two special forwarding tables, and then add a forwarding "policy" rule that selects those special tables with the corresponding marking. The commands for neighbor `R2` are as follows:

```
# ip netns exec R1 \
   ip route add 10.255.255.255/32 via 10.1.2.2 dev R1-eth2 table 2
# ip netns exec R1 \
   ip rule add fwmark 2 lookup 2
```

and similarly for `R3`:

```
# ip netns exec R1 \
   ip route add 10.255.255.255/32 via 10.1.3.3 dev R1-eth3 table 3
# ip netns exec R1 \
   ip rule add fwmark 3 lookup 3
```