Network simulation in Haskell

Matei Popovici

April 2, 2015

1 Motivation

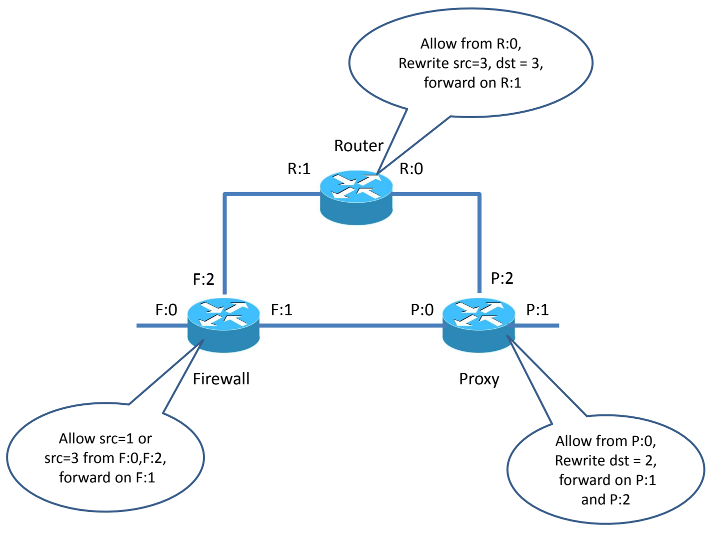

Figure 1: Toy network

Network topologies are increasingly harder to understand and reason about. Thus, it is reasonable to build tools that aid network administrators in verifying that topologies behave as intended. Consider for instance the toy topology from Figure 1. How is traffic directed and modified? For instance, if we send an arbitrary packet on port F:0 of the Firewall, what is the received traffic on port P:1 of the Proxy?

The objective of this homework is to solve *the reachability problem*: for a (i) given topology, (ii) input port, (iii) output port and (iv) traffic pattern, what is the traffic which reaches the output port, if the given traffic pattern is sent on the input port ?

2 Details

The idea that we take is to interpret the topology as an imperative program which manipulates a fixed number of packet header fields.

2.1 Traffic patterns

A packet header will contain a fixed number of header fields, which is known in advance. For simplicity, in the following examples, we shall assume a packet header contains two fields: *src* and *dst*.

A **traffic pattern** is a *description* of a set of packets. Examples:

- "$src = 1$" — the set of packets having the header field *src* equal to 1;
- "$src = any$" — the set of packets having the header field *src* equal to any possible value;
- "$src = 1$ and $dst = 0$";
- "$src = 1$ or $dst = 0$";
- "$(src = any$ and $dst = 0)$ or $src = 999$";

When solving homework, you must decide on a representation of traffic patterns, depending on the language of use. Hints (and some simplifications) will be available for each homework.

2.2 Traffic patterns through the network

The simplified example from Figure 2 shows how traffic patterns change, as they traverse the network. Assume an arbitrary traffic pattern is introduced on port F:0 of the Firewall. The filtering policy of the firewall tells us that only packet headers with the src header set to 1 will be forwarded on port F:1. Next, F:1 is connected to port P:0 of the Proxy. The policy of the proxy is to rewrite the dst field with the value of 2. Hence, on port P:1 of the Proxy, only packets having src=1 and dst=2 are reachable.

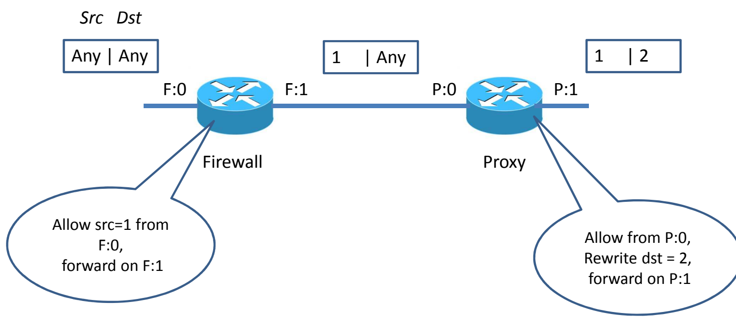

Figure 2: An example of traffic processing

Thus, if the traffic pattern which is sent on F:0 is "$src = any$ and $dst = any$", then the traffic pattern which is received on P:1 is "$src = 1$ and $dst = 2$".

In Figure 3 we have more elaborate policies implemented by the Proxy and Firewall. Hence, the possible packets reachable at port F:1 are those having either src=1 or src=3.

Hence, if the traffic pattern sent on F:0 is "$src = any$ and $dst = any$", on F:1 we receive "$src = 1$ or $src = 3$". Further on, at port P:1, the traffic pattern is P:1 "$(src = 1$ or $src = 3)$ and $dst = 2$".

2.3 Traffic pattern operations

The behaviour of each network device can be described by a combination of basic operations: *intersection*, *reunion*, *subset*, *rewrite*, which are applied traffic patterns:

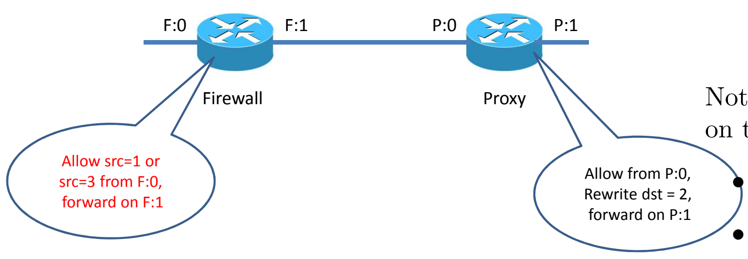
F:0
F:1
P:0
P:1
Firewall
Proxy
Allow src=1 or src=3 from F:0, forward on F:1
Allow from P:0, Rewrite dst = 2, forward on P:1

Figure 3: More complicated processing

- the *intersection* of the patterns p_1 and p_2 is the pattern: "p_1 and p_2".
- the *reunion* of patterns p_1 and p_2 is pattern: "p_1 or p_2".
- p_1 is a *subset* of p_2 if all packet headers belonging to p_1 are also in p_2.
- *rewriting* $v = e$ in pattern p means forcing header variable v to have the value e, in p.

For instance, the filtering policy of the Firewall from Figure 2 can be modelled as the intersection of the arbitrary input pattern p with the pattern "$(src = 1$ or $src = 3)$". Similarly, the policy of the Proxy can be modelled as the rewriting $dst = 2$ on the input flow.

2.4 Representing the topology

Every network processing unit (switch, router, firewall, etc.) performs some modification on the traffic pattern it receives, then it forwards the modified traffic pattern to the correct successor. In other words, a network processing unit (henceforth called NPU) must:

- *establish if the traffic pattern received on some port is subject to modification*
- *modify the traffic pattern accordingly, if this is the case*

We shall use *rules* to describe how NPUs answer these questions. A *rule* is a pair consisting of:

1. A *match* function. It takes a traffic pattern p as parameter and returns $match(p)$, which is *true/false* depending on whether or not the rule should process the pattern. The match function establishes if the rule is applicable on a pattern p.
2. A *modify* function. It takes an initial *pattern* p and returns a *modified* pattern $modify(p)$. The modify function encodes how the rule modifies p.

In what follows, to make the modelling more uniform, we shall consider that a traffic pattern also contains it's *location* in the network. Hence, each pattern will contain "$port = \langle port\_value \rangle$. Thus, all packets arriving at port F:0 are represented by the pattern "$src = any$ and $dst = any$ and $port =$ F:0".

Under this assumption, the devices from Example 3 can be modelled as follows. The Firewall is modelled as a rule with:

- $Match(p)$: check if p is a subset of "$port =$ F:0".
- $Modify(p)$: compute the intersection of p with "$(src = 1$ and $dst = any)$ or $(src = 3$ and $dst = any)$". Next, rewrite $port =$ F:1 in the result.

Notice that the modify rule also models the sending of the flow on the appropriate port. The proxy is modelled by the rule:

- $Match(p)$: check if p is a subset of "$port =$ P:0".
- $Modify(p)$: rewrite $dst = 2$ and $port = P:1$ in p.

By adhering to our convention, port connections (wires) can also be modelled as rules. For instance, the wire between ports F:1 and P:0 can be modelled by the following rule:

- $Match(p)$: check if p is a subset of "$port =$ F:1"
- $Modify(p)$: rewrite $port =$ P:0 in p.

Also, we allow building more complicated rules from simpler ones, by performing *rule composition*. Let r_1 and r_2 be rules defined by functions $match_1(p)$, $modify_1(p)$ and $match_2(p)$, $modify_2(p)$. The composition of rules r_1 and r_2 is defined by the match function $match_1(p) \wedge match_2(p)$ and the modify function $modify_1(modify_2(p))$. For instance, let r_{fwd} encode the forwarding logic of the Firewall:

- $Match(p)$: check if p is a subset of "$port =$ F:0"
- $Modify(p)$: rewrite $port =$ F:1 in p.

and r_{filt} encode the filtering policy:

- $Match(p)$: return True
- $Modify(p)$: compute the intersection of pattern p with "$(src = 1$ and $dst = any)$ or $(src = 3$ and $dst = any)$";

Then, the Firewall can be modelled by the composition of r_{fwd} with r_{filt}. The advantage of this approach is that, via composition, we can apply the same transformation logic to several devices (working on different ports) in the network.

2.5 Reachability

Having a representation for packet headers and NPUs, we attempt to address the reachability problem discussed in the introductory section.

We assume the network topology is represented as a set NT of rules, which include both NPU and wire rules. Computing reachability can be described by the following procedure:

1. Start with $A = \{p\}$, where p is the initial pattern containing the initial port. $All = \emptyset$. The set A contains all patterns which are due exploration, and All contains all previously-explored patterns.
2. Identify the set $App \subseteq NT$ of rules which are applicable on patterns from A.
   (a) If App is empty, stop. Return All.
   (b) Otherwise make $A' = \emptyset$. A' will contain the newly-created patterns resulted in this step.
   (c) For each rule r in App applicable on a pattern p in A:
      i. Compute p' by applying r on p.
      ii. Put p' in A'.
   (d) Make $All = All \cup A$. We have explored all patterns from A, thus, we move them in All.

(e) Make $A = A' \setminus All$. A now contains all newly computed patterns in the current iteration

   i. If $A = \emptyset$, then each current pattern was previously explored: we have a loop. Stop. Return All.

   ii. Otherwise, go to step 2 and repeat the process.

Once the procedure ends, it will return a set All of patterns resulting from the entire network exploration. To find out which patterns are reachable at a given destination port p_{dest}, it suffices to select from All those patterns containing $port = p_{dest}$.

To show how reachability works, we turn to the more complicated network from Figure 1. NF contains four rules for the devices and three rules for the links. The particularity is in representing the Proxy, where two rules are now needed. One rule models the policy from P:0 to 1 and the other, from P:0 to P:2. Assume the initial pattern is $p_i \equiv$"$src = any$ and $dst = any$ and $port =$ F:0".

step1 Only the rule modelling the firewall is applicable, on the initial flow. Thus, A' contains only the pattern $p_1 =$ "($src = 1$ and $dst = any$ and $port =$ F:1) or ($src = 3$ and $dst = any$ and $port =$ F:1)". At the end of this iteration (after ii.), $A = \{p_1\}$, while $All = \{p_i\}$.

step2 The wire rule between the Firewall and the Proxy is applicable. Hence, A' contains $p_2 = \{\{(\text{src},1), (\text{dst,any}), (\text{port,P:0})\}, \{(\text{src},3), (\text{dst,any}), (\text{port,P:0})\}\}$. $A = \{p_2\}$ and $All = \{p_i, p_1\}$.

step3 The two rules of the Proxy are applicable. Hence, A' contains $p_3 =$ " ($src = 1$ and $dst = 2$ and $port =$ P:1) or ($src = 3$ and $dst = 2$ and $port =$ P:1)" and $p_4 =$"($src = 1$ and $dst = 2$ and $port =$ P:2) or ($src = 3$ and $dst = 2$ and $port = P{:}2$) $A = \{p_3, p_4\}$ and $All = \{p_i, p_1, p_2\}$.

step4 Only the wire rule from the Proxy to the Router is applicable, on flow p_4. A' contains $p_5 =$ "($src = 1$ and $dst = 2$ and $port = R{:}0$) or ($src = 3$ and $dst = 2$ and $port =$ R:0)". $A = \{p_5\}$ and $All = \{p_i, p_1, p_2, p_3, p_4\}$.

step5 The router rule is applicable on flow p_5. A' contains $p_6 =$ "$src = 3$ and $dst = 3$ and $port =$ R:1". At the end of this iteration $A = \{p_6\}$ and $All = \{p_i, p_1, p_2, p_3, p_4, p_5\}$.

step6 The wire rule from the Router to the Firewall is applicable. A' contains $p_7 =$"$src = 3$ and $dst = 3$ and $port =$ F:2". At the end of this iteration $A = \{p_7\}$ and $All = \{p_i, p_1, p_2, p_3, p_4, p_5, p_6\}$.

step7 The Firewall rule is applicable. A' contains $p_8 =$ "$src = 3$ and $dst = 3$ and $port =$ F:1". At the end of this iteration $A = \{p_8\}$ and $All = \{p_i, p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$.

step8 The wire rule from the Firewall to the Proxy is applicable. A' contains $p_9 =$ "$src = 3$ and $dst = 3$ and $port =$ P:0". At the end of this iteration $A = \{p_9\}$ and $All = \{p_i, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$.

step9 Both proxy rules are applicable. A' contains $p_{10} =$"$src = 3$ and $dst = 2$ and $port =$ P:1" and $p_{11} =$"$src = 3$ and $dst = 2$ and $port =$ P:2". At the end of this iteration $A = \{p_{10}, p_{11}\}$ and $All = \{p_i, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9\}$.

step10 The wire rule from the Proxy to the Router is applicable. A' contains $p_{12} =$"$src = 3$ and $dst = 2$ and $port =$ R:0". At the end of this iteration $A = \{p_{12}\}$ and $All = \{p_i, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}\}$.

step11 The Router is applicable. A' contains $p_{13} =$"$src = 3$ and $dst = 3$ and $port =$ R:1". Note that p_{13} is already contained in All since $p_{13} = p_6$. Hence, at the end of the iteration $A = \emptyset$. The procedure returns $All = \{p_i, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}\}$.

Finally, we examine all patterns which contain P:1 as current port. p_3, p_{10} satisfy this condition. Hence, the reunion of p_3 and p_{10} is reachable on port P:1 of the Proxy.