Network simulation in Haskell

Matei Popovici

February 25, 2014

1 Motivation

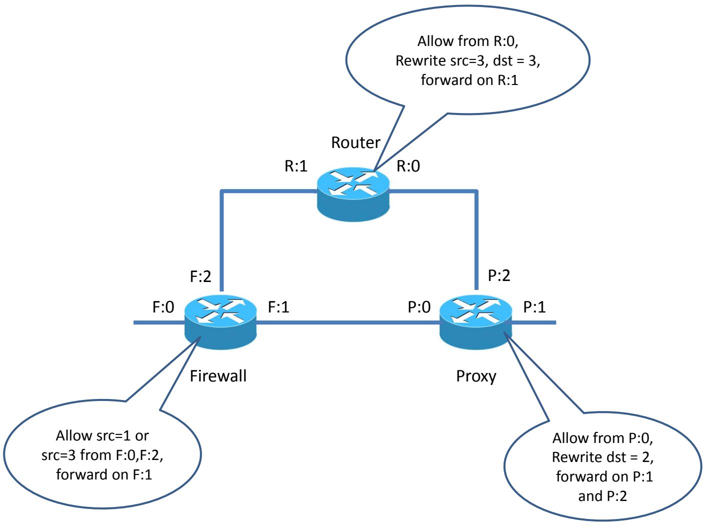

Figure 1: Toy network

Networks are nowadays becoming more and more elaborate and harder to understand and reason about. Thus, it is reasonable to build tools that aid network administrators in understanding the behavior of the networks they build. Consider for instance the toy network in Figure 1. How does this network modify traffic ? What are the properties of the traffic resulted after the processing ? 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?

In this homework, we shall be preoccupied in answering questions of the latter sort. We shall call this, the *reachability* problem. In what follows, we provide with a high-level description on how the traffic and the network are to be represented in an implementation.

2 The approach

One idea is to consider the network as a large imperative computer program in which the manipulated data (variables) are the header fields of the packet.

2.1 Representing packets

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*. The simplified example from Figure 2 shows how header fields are manipulated by the network. Assume an arbitrary packet header 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.

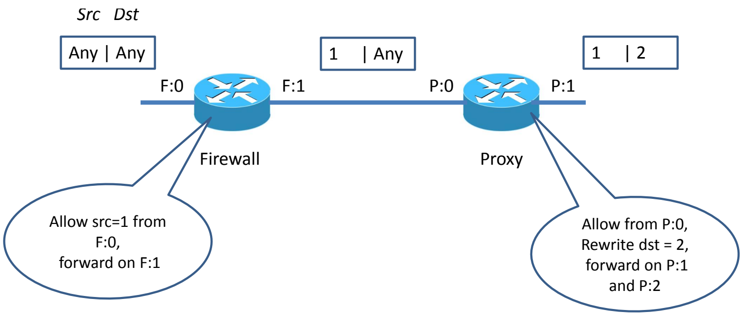

Figure 2: An example of traffic processing

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.

In general, we can represent packet headers which are accessible at some point in the network as sets *cf* of pairs (v, e) where v is a header field and e is an expression which may be a value, or it may be the special expression called ”*any*“. For instance, in Figure 2 the possible packets which can reach P:0 of the Proxy is $cf = \{(\text{src,1}), (\text{dst,any})\}$ We say such a *cf* is a *compact flow*.

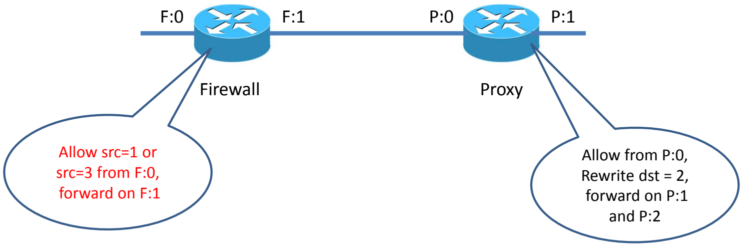

Figure 3: More complicated processing

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. We cannot represent such a set of packets using a compact flow.

Hence, we introduce the notion of *flow*, which is a set f of compact flows: $f = \{cf_1, \ldots, cf_n\}$. For instance, the flow which is reachable at port F:1 is $\{\{(\text{src,1}), (\text{dst,any})\}, \{(\text{src,3}), (\text{dst,any})\}\}$. Similarly, the flow reachable at port P:1 is: $\{\{(\text{src,1}), (\text{dst,2})\}, \{(\text{src,3}), (\text{dst,2})\}\}$. One can interpret a flow f as *a representation for the reunion* of compact flows $cf_1, \ldots, cf_n$.

2.2 Manipulating flows

As already intuitive from the previous examples, the behaviour of each network device can be described by a combination of basic operations: *intersection*, *reunion*, *subset*, *rewrite*, which are applied on flows. Their behaviour is as follows:

- the *intersection* of the flows f_1 and f_2 is the flow containing packet headers from both f_1 and f_2.
- the *reunion* of the flows f_1 and f_2 is the flow containing packet headers from either f_1 or f_2.
- f_1 is a *subset* of f_2 if all packet headers from f_1 are also in f_2.
- *rewriting* (v, e) in flow f means forcing header variable v to have the value e, in f.

For instance, the filtering policy of the Firewall from Figure 2 can be modelled as the intersection of the arbitrary input flow f with the flow $\{\{(\text{src},1), (\text{dst},\text{any})\}\}$. Similarly, the policy of the Proxy can be modelled as the rewriting (dst,2) on the input flow.

2.3 Representing the topology

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

- *Should I process a packet received on some port?*
- *If the previous answer is* Yes*, how does the processed packet should look like?*

We shall use *rules* to describe how NPUs answer these questions. A *rule* is a pair of two functions (in the mathematical sense):

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

In what follows, to make the modelling more uniform, we shall consider that a flow also contains the *location* of the packet. Hence, each flow will contain $(port, portValue)$, which tells us where the port is. Thus, we represent the flow of all packets arriving at port F:0 as $\{\{(\text{src},1), (\text{dst},\text{any}), (\text{port},\text{F:0})\}\}$.

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

- $Match(f)$: check if f is a subset of $\{\{(\text{port},\text{F:0})\}\}$.
- $Modify(f)$: compute the intersection of f with the flow $\{\{(\text{src},1), (\text{dst},\text{any})\}, \{(\text{src},3), (\text{dst},\text{any})\}\}$. 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:

- $Match(f)$: check if f is a subset of $\{\{(\text{port},\text{P:0})\}\}$.
- $Modify(f)$: rewrite (dst,5) and (port,P:1) in f.

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:

- $Match(f)$: check if f is a subset of $\{\{(\text{port},\text{F:1})\}\}$
- $Modify(f)$: rewrite (port,P:0) in f.

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 $match_1(f)$, $modify_1(f)$ and $match_2(f)$, $modify_2(f)$. The composition of rules r_1 and r_2 is defined by the match function $match_1(f) \wedge match_2(f)$ and the modify function $modify_1(modify_2(f))$. For instance, let r_{fwd} encode the forwarding logic of the Firewall:

- $Match(f)$: check if f is a subset of $\{\{(\text{port},\text{F:0})\}\}$
- $Modify(f)$: rewrite (port,F:1) in f.

and r_{filt} encode the filtering policy:

- $Match(f)$: return True
- $Modify(f)$: compute the intersection of flow f with the flow $\{\{(\text{src},1), (\text{dst},\text{any})\}, \{(\text{src},3), (\text{dst},\text{any})\}\}$

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.

2.4 Reachability

Having a representation for packet headers and NPUs, the question we attempt to answer is: *Having a certain flow (on a certain port), what is (are) the reachable flow(s) on a given destination port?*

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 $A = \{f\}$, where f is the initial flow containing the initial port. $All = \emptyset$. The set A contains all flows which are pending to be explored, and All contains all flows explored in previous steps.
2. Identify the set $App \subseteq NT$ of rules which are applicable on flows from A.
   (a) If App is empty, stop. Return All.
   (b) Otherwise make $A' = \emptyset$. A' will contain the newly-created flows resulted in this step.
   (c) For each rule r in App applicable on a flow f in A:
      i. Compute f' by applying r on f.
      ii. Put f' in A'.
   (d) Make $All = All \cup A$. We have explored all flows from A, thus, we move them in All.
   (e) Make $A = A' \setminus All$. Thus, A now contains all newly computed flows, and which were not computed in a previous step.
      i. If $A = \emptyset$, then each flow computed in this step has already been constructed previously, and 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 flows resulting from the entire network exploration. To find out which flows are reachable at a given destination port p_{dest}, it suffices to select from All those flows 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 flow is $f_i = \{\{(\text{src,any}), (\text{dst,any}), (\text{port,F:0})\}\}$.

- step1 Only the rule modelling the firewall is applicable, on the initial flow. Thus, A' contains only the flow $f_1 = \{\{(\text{src,1}), (\text{dst,any}), (\text{port,F:0})\}, \{(\text{src,3}), (\text{dst,any}), (\text{port,F:0})\}\}$. At the end of this iteration (after ii.), $A = \{f_1\}$, while $All = \{f_i\}$.

- step2 The wire rule between the Firewall and the Proxy is applicable. Hence, A' contains $f_2 = \{\{(\text{src,1}), (\text{dst,any}), (\text{port,P:0})\}, \{(\text{src,3}), (\text{dst,any}), (\text{port,P:0})\}\}$. $A = \{f_2\}$ and $All = \{f_i, f_1\}$.

- step3 The two rules of the Proxy are applicable. Hence, A' contains $f_3 = \{\{(\text{src,1}), (\text{dst,2}), (\text{port,P:1})\}, \{(\text{src,3}), (\text{dst,2}), (\text{port,P:1})\}\}$ and $f_4 = \{\{(\text{src,1}), (\text{dst,2}), (\text{port,P:2})\}, \{(\text{src,3}), (\text{dst,2}), (\text{port,P:2})\}\}$. $A = \{f_3, f_4\}$ and $All = \{f_i, f_1, f_2\}$.

- step4 Only the wire rule from the Proxy to the Router is applicable, on flow f_4. A' contains $f_5 = \{\{(\text{src,1}), (\text{dst,2}), (\text{port,R:0})\}, \{(\text{src,3}), (\text{dst,2}), (\text{port,R:0})\}\}$. $A = \{f_5\}$ and $All = \{f_i, f_1, f_2, f_3, f_4\}$.

- step5 The router rule is applicable on flow f_5. A' contains $f_6 = \{\{(\text{src,3}), (\text{dst,3}), (\text{port,R:1})\}\}$. At the end of this iteration $A = \{f_6\}$ and $All = \{f_i, f_1, f_2, f_3, f_4, f_5\}$.

- step6 The wire rule from the Router to the Firewall is applicable. A' contains $f_7 = \{\{(\text{src,3}), (\text{dst,3}), (\text{port,F:2})\}\}$. At the end of this iteration $A = \{f_7\}$ and $All = \{f_i, f_1, f_2, f_3, f_4, f_5, f_6\}$.

- step7 The Firewall rule is applicable. A' contains $f_8 = \{\{(\text{src,3}), (\text{dst,3}), (\text{port,F:1})\}\}$. At the end of this iteration $A = \{f_8\}$ and $All = \{f_i, f_1, f_2, f_3, f_4, f_5, f_6, f_7\}$.

- step8 The wire rule from the Firewall to the Proxy is applicable. A' contains $f_9 = \{\{(\text{src,3}), (\text{dst,3}), (\text{port,P:0})\}\}$. At the end of this iteration $A = \{f_9\}$ and $All = \{f_i, f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8\}$.

- step9 Both proxy rules are applicable. A' contains $f_{10} = \{\{(\text{src,3}), (\text{dst,2}), (\text{port,P:1})\}\}$ and $f_{11} = \{\{(\text{src,3}), (\text{dst,2}), (\text{port,P:2})\}\}$. At the end of this iteration $A = \{f_{10}, f_{11}\}$ and $All = \{f_i, f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9\}$.

- step10 The wire rule from the Proxy to the Router is applicable. A' contains $f_{12} = \{\{(\text{src,3}), (\text{dst,2}), (\text{port,R:0})\}\}$. At the end of this iteration $A = \{f_{12}\}$ and $All = \{f_i, f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, f_{10}, f_{11}\}$.

- step11 The Router is applicable. A' contains $f_{13} = \{\{(\text{src,3}), (\text{dst,3}), (\text{port,R:1})\}\}$. Note that f_{13} is already contained in All since $f_{13} = f_6$. Hence, at the end of the iteration $A = \emptyset$. The procedure returns $All = \{f_i, f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, f_{10}, f_{11}, f_{12}\}$.

Finally, we examine all flows which contain P:1 as current port. f_3, f_{10} satisfy this condition. Hence, the reunion of f_3 and f_{10} is reachable on port P:1 of the Proxy.