The Frenetic Network Controller

http://github.com/frenetic-lang/frenetic/contributors

June 18, 2013

Abstract

Frenetic is a software-defined networking controller platform. With Frenetic, developers describe the intended behavior of the network in a high-level language, and the compiler and run-time system generate the low-level code needed to execute programs efficiently in hardware. This talk will introduce the key programming abstractions provided in Frenetic, and present highlights of our experiences implementing those abstractions in OCaml.

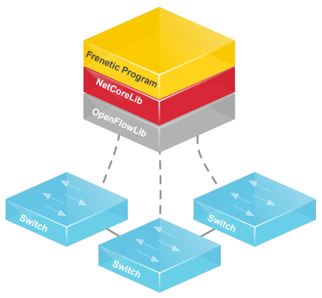
1 Overview

Software-defined networking (SDN) is an emerging network architecture in which a logically-centralized *controller machine* manages the behavior of a collection of *programmable switches*. This design simplifies network algorithms and also makes it easy to extend the network with new functionality. It is in stark contrast to traditional architectures, in which network algorithms must be expressed using complicated distributed programs that execute on proprietary devices. Most SDN controller platforms provide low-level programming interfaces that closely mirror the capabilities of the underlying hardware. Frenetic is unique in that it provides a high-level, declarative interface that abstracts away from the details of the hardware and allows developers to focus on the essential features of network programs.

Figure 1 depicts Frenetic's architecture and core policy language. Each policy denotes a function from sets of packets to sets of packets. The policy `Id` denotes the identity function, while `Drop` denotes the constant function that produces the empty set. The policy `Switch(sw)` is a filter that retains packets on switch `sw` and drops other packets. Likewise, `Match(pat)` and `NoMatch(pat)` retains packets whose header fields match and do not match `pat` respectively. The `Union` and `Sequence` operators, often abbreviated as `(+)` and `(;)`, denote union and sequential composition. The `Forward(out)` policy forwards packets to the location `out`, which must be adjacent in the network topology. The policy `Controller(h)` invokes the handler `h` on each input packet. Finally, `Query(f,h)` computes the number and total size of all input packets and invokes the handler `h` every `f` seconds. To illustrate, consider the following program, which implements a simple firewall and monitoring policy:

```
  NoMatch(icmp); Forward(all)
+ Query(60.0, print)
```

It blocks ICMP (ping) traffic and behaves like a repeater on all other traffic, and also prints the total amount of traffic to the console every minute.



(a)

```
type pol =
  | Id
  | Drop
  | Switch of switch
  | Match of pattern
  | NoMatch of pattern
  | Union of pol * pol
  | Sequence of pol * pol
  | Forward of output
  | Controller of (packet -> unit)
  | Query of float * (stats -> unit)
```

(b)

Figure 1: Frenetic (a) architecture (b) core syntax.

2 System

The Frenetic implementation [1] has several distinct components:

OpenFlowLib: Provides datatypes, parsers, and serializers for OpenFlow, the most popular SDN framework. This library makes heavy use of the `cstruct` package, which provides constructs for manipulating C-style structures in OCaml and greatly simplifies the task of writing binary parsers and serializers.

PacketLib: Provides datatypes, parsers, and serializers for Ethernet, IP, ARP, TCP, and UDP packets. This library also relies heavily on the `cstruct` package. To ensure good performance, the parsers are lazy, which allows programmers to efficiently access particular fields of packets as needed.

NetCoreLib: Implements the Frenetic policy language. It defines the abstract syntax, as well as a compiler and run-time system that implements this language using the lower-level interface provided by OpenFlowLib.

Verification: Although not included in our main development branch, we have also implemented and mechanically verified a compiler and run-time system in Coq. In addition, we have built a tool that automatically checks network reachability properties using Z3.

Main: Provides a number of additional features including natural surface syntax, support for dynamic and stateful policies using `Lwt`, and integrated testing and debuging facilities.

This talk will introduce the key programming abstractions provided in Frenetic, and present highlights of our experiences implementing those abstractions in Coq and OCaml.

References

[1] Arjun Guha, Mark Reitblatt, and Nate Foster. Machine-verified network controllers. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Seattle, WA*, June 2013. To appear.