The Frenetic Network Controller

Arjun Guha, Nate Foster, Mark Reitblatt, Cole Schlesinger, and others:

www.github.com/frenetic-lang/frenetic/contributors
hosts

networks today
networks today

hosts

switches
networks today

hosts

switches

servers
networks today

hosts

switches

routers

servers
networks today

hosts

switches

firewalls

routers

servers
networks today

hosts
load balancers
switches
firewalls
routers
servers
networks today

hosts

load balancers

switches

firewalls

routers

wireless access points

servers
networks today

hosts

switches

load balancers

firewalls

routers

wireless authentication server

wireless access points

servers
Each box:
- Runs specialized, vendor-defined software
- Configured independently via a command-line interface

Difficult to configure, difficult to reason about
Recent Network Outages

We discovered a misconfiguration on this pair of switches that caused what's called a “bridge loop” in the network.

A network change was [...] executed incorrectly [...] more “stuck” volumes and added more requests to the re-mirroring storm.

Service outage was due to a series of internal network events that corrupted router data tables.

Experienced a network connectivity issue [...] interrupted the airline's flight departures, airport processing and reservations systems.
software-defined networking (SDN)
software-defined networking (SDN)

standardized, programmable network devices ("switches")
Key Features and Advantages of SDN

- Standardized, programmable network devices
  easy to deploy new in-network features
Key Features and Advantages of SDN

- Standardized, programmable network devices easy to deploy new in-network features
- Logically centralized controller (beefy server) enables reasoning about whole-network behavior

\[ f : \text{switch} \times \text{port} \times \text{packet} \rightarrow \{ (\text{port}_1, \text{packet}_1), \ldots, (\text{port}_n, \text{packet}_n) \} \]
Lots of SDN Interest

• By startups and established players
  *can buy commercial hardware and software*

• 200+ attendees at HotSDN ’13

• Six (out of 40) papers at SIGCOMM’13 on SDN
industry-standard SDN protocol
industry-standard SDN protocol

Java

Project Floodlight
industry-standard SDN protocol

Java

Python

C++
industry-standard SDN protocol

Java

Python

C++

Haskell

The nettle-openflow package

This package provides data types that model the messages of the OpenFlow protocol, functions that implement serialization and deserialization between these data types and their binary representations in the protocol, and an efficient OpenFlow server. The library is under active development.
industry-standard SDN protocol

Java

Python

C++

Haskell

OCaml
Example
Example

let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
  let actions =
    if pktIn.port = 1 then
      [Output (PhysicalPort 2)]
    else
      [Output (PhysicalPort 1)] in
  send_packet_out sw 0 { output_payload = pktIn.input_payload;
    port_id = None;
    apply_actions = actions }
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
  let pk = parse_payload pktIn.input_payload in
  let actions =
    if Packet.dlTyp pk = 0x800 && Packet.nwProto = 6 &&
      (Packet.tpDst = 22 || Packet.tpSrc = 22) then
      [] (* no action (i.e., drop) SSH packets *)
    else if pktIn.port = 1 then
      [Output (PhysicalPort 2)]
    else
      [Output (PhysicalPort 1)] in
  send_packet_out sw 0 { output_payload = pktIn.input_payload;
                        port_id = None;
                        apply_actions = actions }
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
  let pk = parse_payload pktIn.input_payload in
  let actions =
    if Packet.dlTyp pk = 0x800 && Packet.nwProto = 6 &&
    (Packet.tpDst = 22 || Packet.tpSrc = 22) then
    [] (* no action (i.e., drop) SSH packets *)
    else if pktIn.port = 1 then
    [Output (PhysicalPort 2)]
    else
    [Output (PhysicalPort 1)] in
  send_packet_out sw 0l
  { output_payload = pktIn.input_payload;
    port_id = None;
    apply_actions = actions }
Example

```
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
  let pk = parse_payload pktIn.input_payload in
  let actions =
    if Packet.dlTyp pk = 0x800 && Packet.nwProto = 6 &&
       (Packet.tpDst = 22 || Packet.tpSrc = 22) then
    [] (* no action (i.e., drop) SSH packets *)
    else if pktIn.port = 1 then
      [Output (PhysicalPort 2)]
    else
      [Output (PhysicalPort 1)] in
  send_packet_out sw 0l
  { output_payload = pktIn.input_payload;
    port_id = None;
    apply_actions = actions }
```
Example

let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
  let pk = parse_payload pktIn.input_payload in
  let actions =
    if Packet.dlTyp pk = 0x800 && Packet.nwProto = 6 &&
    (Packet.tpDst = 22 || Packet.tpSrc = 22) then
      [] (* no action (i.e., drop) SSH packets *)
    else if pktIn.port = 1 then
      [Output (PhysicalPort 2)]
    else
      [Output (PhysicalPort 1)] in
  send_packet_out sw 0l
  { output_payload = pktIn.input_payload;
    port_id = None;
    apply_actions = actions }

Controller

Host 1

Host 2

OpenFlow Switch

Generic packet parser

OpenFlow protocol parser
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
  let pk = parse_payload pktIn.input_payload in
  let actions =
    if Packet.dlTyp pk = 0x800 && Packet.nwProto = 6 &&
    (Packet.tpDst = 22 || Packet.tpSrc = 22) then [[] (* no action (i.e., drop) SSH packets *)]
    else if pktIn.port = 1 then
      [Output (PhysicalPort 2)]
    else
      [Output (PhysicalPort 1)] in
  send_packet_out sw 0l
  { output_payload = pktIn.input_payload;
    port_id = None;
    apply_actions = actions }
ocaml-packet
ocaml-packet

- Serialization / deserialization for several packet formats
  TCP, IP, ARP, ICMP, Ethernet, 802.1Q
ocaml-packet

- Serialization / deserialization for several packet formats: TCP, IP, ARP, ICMP, Ethernet, 802.1Q
- Only depends on cstruct

by Anil Madhavapeddy et al.
ocaml-packet

- Serialization / deserialization for several packet formats
  TCP, IP, ARP, ICMP, Ethernet, 802.1Q
- Only depends on cstruct
  by Anil Madhavapeddy et al.

cstruct ip {
    uint8_t vhl;
    uint8_t tos;
    uint16_t len;
    uint16_t ident;
    uint16_t frag;
    uint8_t ttl;
    uint8_t proto;
    uint16_t chksum;
    uint32_t src;
    uint32_t dst;
    uint32_t options
} as big_endian
ocaml-packet

- Serialization / deserialization for several packet formats
  TCP, IP, ARP, ICMP, Ethernet, 802.1Q
- Only depends on cstruct
  by Anil Madhavapeddy et al.

cstruct ip {
  uint8_t vhl;
  uint8_t tos;
  uint16_t len;
  uint16_t ident;
  uint16_t frag;
  uint8_t ttl;
  uint8_t proto;
  uint16_t chksum;
  uint32_t src;
  uint32_t dst;
  uint32_t options
} as big_endian

network byte order for free
ocaml-openflow
ocaml-openflow

• Serialization for OpenFlow 1.0 and 1.3
  - OpenFlow 1.0 support based on mirage-openflow, using ideas from Nettle (Haskell)
  - OpenFlow 1.3 is less complete, but still usable
  - Cstruct-based serialization
ocaml-openflow

• Serialization for OpenFlow 1.0 and 1.3
  - OpenFlow 1.0 support based on mirage-openflow, using ideas from Nettle (Haskell)
  - OpenFlow 1.3 is less complete, but still usable
  - Cstruct-based serialization

• Runtime systems for OpenFlow 1.0 and 1.3
  - Listens for TCP connections from switches
  - Does the OpenFlow handshake, keeps connections to switches alive
ocaml-openflow

- Serialization for OpenFlow 1.0 and 1.3
  - OpenFlow 1.0 support based on mirage-openflow, using ideas from Nettle (Haskell)
  - OpenFlow 1.3 is less complete, but still usable
  - Cstruct-based serialization

- Runtime systems for OpenFlow 1.0 and 1.3
  - Listens for TCP connections from switches
  - Does the OpenFlow handshake, keeps connections to switches alive

```
type t (* A handle to an OpenFlow switch. *)

val connect : Lwt_unix.file_descr -> t option Lwt.t
val send : t -> xid -> message -> unit Lwt.t
val recv : t -> (xid * message) Lwt.t
val disconnect : t -> unit Lwt.t
val wait_disconnect : t -> unit Lwt.t
```
ocaml-openflow

• Serialization for OpenFlow 1.0 and 1.3
  - OpenFlow 1.0 support based on mirage-openflow, using ideas from Nettle (Haskell)
  - OpenFlow 1.3 is less complete, but still usable
  - Cstruct-based serialization

• Runtime systems for OpenFlow 1.0 and 1.3
  - Listens for TCP connections from switches
  - Does the OpenFlow handshake, keeps connections to switches alive

```plaintext
type t (* A handle to an OpenFlow switch. *)

val connect : Lwt_unix.file_descr -> t option Lwt.t
val send : t -> xid -> message -> unit Lwt.t
val recv : t -> (xid * message) Lwt.t
val disconnect : t -> unit Lwt.t
val wait_disconnect : t -> unit Lwt.t
```
OX

- Based on popular OpenFlow platforms
  - Inspired by POX (Python) and NOX (Python/C++)
OX

• Based on popular OpenFlow platforms
  – Inspired by POX (Python) and NOX (Python/C++)
• Simple, single-threaded OpenFlow interface
  – Uses LWT and ocaml-openflow internally
OX

• Based on popular OpenFlow platforms
  - Inspired by POX (Python) and NOX (Python/C++)
• Simple, single-threaded OpenFlow interface
  - Uses LWT and ocaml-openflow internally

[Diagram showing the relationship between OX, Simple Controllers, Advanced Controllers, ocaml-openflow, ocaml-packet, cstruct, and LWT]
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
let pk = parse_payload pktIn.input_payload in
let action =
  if Packet.dlTyp pk = 0x800 && Packet.nwProto = 6 &&
    (Packet.tpDst = 22 || Packet.tpSrc = 22) then
    [] (* no action (i.e., drop) SSH packets *)
  else if pktIn.port = 1 then
    [Output (PhysicalPort 2)]
  else
    [Output (PhysicalPort 1)] in
send_packet_out sw 0l
{ output_payload = pktIn.input_payload; port_id = None;
  apply_actions = action }
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
let pk = parse_payload pktIn.input_payload in
let action =
  if Packet.dlTyp pk = 0x800 && Packet.nwProto = 6 &&
  (Packet.tpDst = 22 || Packet.tpSrc = 22) then
    [] (* no action (i.e., drop) SSH packets *)
  else if pktIn.port = 1 then
    [Output (PhysicalPort 2)]
  else
    [Output (PhysicalPort 1)] in
send_packet_out sw 0l
{ output_payload = pktIn.input_payload; port_id = None;
  apply_actions = action }

let switch_connected (sw : switchId) : unit =
send_flow_mod sw 0l (add_flow 200 match_ssh_src []);
send_flow_mod sw 0l (add_flow 200 match_ssh_dst []);
send_flow_mod sw 0l
  (add_flow 199 match_from_1 [Output (PhysicalPort 1)]);
send_flow_mod sw 0l
  (add_flow 198 match_from_2 [Output (PhysicalPort 2)]);
send_flow_mod sw 0l (add_flow 197 match_all [])
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
let pk = parse_payload pktIn.input_payload in
let action =
  if Packet.dlTyp pk = 0x800 && Packet.nwProto = 6 &&
    (Packet.tpDst = 22 || Packet.tpSrc = 22) then
    [] (* no action (i.e., drop) SSH packets *)
  else if pktIn.port = 1 then
    [Output (PhysicalPort 2)]
  else
    [Output (PhysicalPort 1)]
in
send_packet_out sw 0l
{ output_payload = pktIn.input_payload; port_id = None;
  apply_actions = action }

<table>
<thead>
<tr>
<th>Priority</th>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>dlType:0x800, nwProto: 6, tpSrc: 22</td>
<td>drop</td>
</tr>
<tr>
<td>200</td>
<td>dlType:0x800, nwProto: 6, tpDst: 22</td>
<td>drop</td>
</tr>
<tr>
<td>199</td>
<td>inPort: 1</td>
<td>Fwd 2</td>
</tr>
<tr>
<td>198</td>
<td>inPort: 2</td>
<td>Fwd 1</td>
</tr>
</tbody>
</table>
let packet_in (sw : switchId) (xid : xid) (pktIn : packetIn) : unit =
  let pk = parse_payload pktIn.input_payload in
  let action =
    if Packet.dlTyp pk = 0x800 && Packet.nwProto = 6 &&
      (Packet.tpDst = 22 || Packet.tpSrc = 22) then
      [] (* no action (i.e., drop) SSH packets *)
    else if pktIn.port = 1 then
      [Output (PhysicalPort 2)]
    else
      [Output (PhysicalPort 1)]
  in
send_packet_out sw 0l
{ output_payload = pktIn.input_payload; port_id = None;
  apply_actions = action }

<table>
<thead>
<tr>
<th>Priority</th>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>dlType:0x800, nwProto: 6, tpSrc: 22</td>
<td>drop</td>
</tr>
<tr>
<td>200</td>
<td>dlType:0x800, nwProto: 6, tpDst: 22</td>
<td>drop</td>
</tr>
<tr>
<td>199</td>
<td>inPort: 1</td>
<td>Fwd 2</td>
</tr>
<tr>
<td>198</td>
<td>inPort: 2</td>
<td>Fwd 1</td>
</tr>
<tr>
<td>Priority</td>
<td>Pattern</td>
<td>Actions</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>---------</td>
</tr>
<tr>
<td>65535</td>
<td>dstIP: H1</td>
<td>Forward</td>
</tr>
</tbody>
</table>

+ 

<table>
<thead>
<tr>
<th>Priority</th>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>65535</td>
<td>SSH</td>
<td>Monitor</td>
</tr>
</tbody>
</table>

*Monsanto et al., POPL 2012*
<table>
<thead>
<tr>
<th>Priority</th>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>65535</td>
<td>dstIP: H1</td>
<td>Forward</td>
</tr>
</tbody>
</table>

+ 

<table>
<thead>
<tr>
<th>Priority</th>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>65535</td>
<td>SSH</td>
<td>Monitor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Priority</th>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>65535</td>
<td>SSH, dstIP: H1</td>
<td>Forward, Monitor</td>
</tr>
<tr>
<td>65534</td>
<td>dstIP: H1</td>
<td>Forward</td>
</tr>
<tr>
<td>65533</td>
<td>SSH</td>
<td>Monitor</td>
</tr>
<tr>
<td>Priority</td>
<td>Pattern</td>
<td>Actions</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>65535</td>
<td>dstIP: H1</td>
<td>Forward</td>
</tr>
</tbody>
</table>

+  

<table>
<thead>
<tr>
<th>Priority</th>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>65535</td>
<td>SSH</td>
<td>Monitor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Priority</th>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>65535</td>
<td>SSH, dstIP: H1</td>
<td>Forward, Monitor</td>
</tr>
<tr>
<td>65534</td>
<td>dstIP: H1</td>
<td>Forward</td>
</tr>
<tr>
<td>65533</td>
<td>SSH</td>
<td>Monitor</td>
</tr>
</tbody>
</table>

**Other issues:**
- Well-formedness criteria on patterns; rules applied non-deterministically in certain situations *(Guha et al., PLDI 2013)*
- Cannot atomically update all switches *(Rietblatt et al., SIGCOMM 2012)*
• DSL for programming OpenFlow networks
  - Boolean predicates to match packets
  - Several policy composition operators
  - All compile to OpenFlow tables
  - Abstractions address several fundamental problems of SDN
• DSL for programming OpenFlow networks
  - Boolean predicates to match packets
  - Several policy composition operators
  - All compile to OpenFlow tables
  - Abstractions address several fundamental problems of SDN
• Embedded in OCaml
  - Access to standard OCaml data structures and functions
  - Must use LWT for concurrency
• DSL for programming OpenFlow networks
  - Boolean predicates to match packets
  - Several policy composition operators
  - All compile to OpenFlow tables
  - Abstractions address several fundamental problems of SDN

• Embedded in OCaml
  - Access to standard OCaml data structures and functions
  - Must use LWT for concurrency

• Surface syntax
  
  if tpSrc = 22 || tpDst = 22 then
  drop
  else if inPort = 1 then
    fwd(2)
  else
    fwd(1)
• Implements key ideas from several published papers
  - ICFP’11, POPL’12, SIGCOMM’11, PLDI’13, HotSDN’12, HotSDN’13
• Open source development of current work
• Implements key ideas from several published papers
  - ICFP’11, POPL’12, SIGCOMM’11, PLDI’13, HotSDN’12, HotSDN’13
• Open source development of current work
• Frenetic and OpenFlow tutorial in OCaml
From Haskell to OCaml :)
From Haskell to OCaml :)

- Core compiler and runtime system done in Coq
  - Extracting to Haskell is much more painful
  - Haskell extraction made us abandon some Coq functors
From Haskell to OCaml :)

• Core compiler and runtime system done in Coq
  - Extracting to Haskell is much more painful
  - Haskell extraction made us abandon some Coq functors

• Functors helped us discover key algebraic properties
  - Helped establish connection to Kleene Algebra with Tests (KAT)
  - Basis of a paper in submission
From Haskell to OCaml :) 

• Core compiler and runtime system done in Coq
  - Extracting to Haskell is much more painful
  - Haskell extraction made us abandon some Coq functors
• Functors helped us discover key algebraic properties
  - Helped establish connection to Kleene Algebra with Tests (KAT)
  - Basis of a paper in submission
• Easier for our networking collaborators to grok OCaml
  - Design discussions over .mli files with networking collaborators
From Haskell to OCaml :) 

- Core compiler and runtime system done in Coq
  - Extracting to Haskell is much more painful
  - Haskell extraction made us abandon some Coq functors

- Functors helped us discover key algebraic properties
  - Helped establish connection to Kleene Algebra with Tests (KAT)
  - Basis of a paper in submission

- Easier for our networking collaborators to grok OCaml
  - Design discussions over .mli files with networking collaborators

- OPAM and oasis are great
  - All our packages are on OPAM
  - OPAM overlays provide stability
From Haskell to OCaml :(
From Haskell to OCaml :( 

- Lwt programming is mind-boggling for beginners
  - Syntax extension helps a lot
  - E.g., very difficult to reason about exceptions
From Haskell to OCaml :

• Lwt programming is mind-boggling for beginners
  - Syntax extension helps a lot
  - E.g., very difficult to reason about exceptions

• No more multicore
  - Needed for large networks (Voellmy et al. at Haskell ’13)
  - Needed to compile fast-changing policies (Ferguson et al. at SIGCOMM’13)
Hack your Network in OCaml

Frenetic
Hack at any layer

- Packet serialization
- OpenFlow serialization
- Ox controller
- Frenetic
Hack your Network in OCaml

Hack at any layer
- Packet serialization
- OpenFlow serialization
- Ox controller
- Frenetic

Ongoing work in OCaml
- Network Hypervisor
  Laurent Vanbever, et al.
- Datalog-based SDN language
  Tim Nelson, et al.
- Fault Tolerant Frenetic
  Mark Reitblatt, et al.
- Property-checking Frenetic
  Rebecca Coombes, Matthew Milano, et al.