Machine-Verified Network Controllers

Arjun Guha (UMass)
Mark Reitblatt (Cornell)
Nate Foster (Cornell)

[Originally presented at PLDI ’13]
Networks today

hosts
Networks today

hosts

switches
Networks today

hosts

switches

servers
Networks today
Networks today
Networks today

- hosts
- load balancers
- switches
- firewalls
- routers
- servers
Networks today
Networks today

- Load balancers
- Switches
- Firewalls
- Routers
- Wireless authentication server
- Wireless access points
- Servers
- Hosts
Challenges:

• Devices execute specialized, vendor-defined software...
• ...and must be configured independently via command-line interfaces

Extraordinarily difficult to answer simple questions about global network behavior
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 networks (SDN)
Software-defined networks (SDN)
Software-defined networks (SDN)
Software-defined networks (SDN)

Standardized programmable devices
Logically-centralized control
let forward (sw, pt, pk) = match (sw, pk.dst_ip) with
| (1, H1) -> [(1, pk)]
| (1, H2) -> [(2, pk)]
| (2, H1) -> [(1, pk)]
| (2, H2) -> [(2, pk)]
| _ -> []
let forward (sw, pt, pk) =  
  match (sw, pk.dst_ip) with  
  | (1, H1) -> [(1, pk)]  
  | (1, H2) -> [(2, pk)]  
  | (2, H1) -> [(1, pk)]  
  | (2, H2) -> [(2, pk)]  
  | _ -> []
let forward (sw, pt, pk) = 
  match (sw, pk.dst_ip) with
  | (1, H1) -> [(1, pk)]
  | (1, H2) -> [(2, pk)]
  | (2, H1) -> [(1, pk)]
  | (2, H2) -> [(2, pk)]
  | _        -> []

\[ f : \text{switch} \times \text{port} \times \text{packet} \rightarrow \{ (\text{port}_1, \text{packet}_1), \ldots, (\text{port}_n, \text{packet}_n) \} \]
let forward (sw, pt, pk) = 
  match (sw, pk.dst_ip) with 
  | (1, H1) -> [(1, pk)] 
  | (1, H2) -> [(2, pk)] 
  | (2, H1) -> [(1, pk)] 
  | (2, H2) -> [(2, pk)] 
  | _ -> []
let forward (sw, pt, pk) = 
match (sw, pk.dst_ip) with 
| (1, H1) -> [(1, pk)] 
| (1, H2) -> [(2, pk)] 
| (2, H1) -> [(1, pk)] 
| (2, H2) -> [(2, pk)] 
| _ -> [] 

\[ f : \text{switch} \times \text{port} \times \text{packet} \rightarrow \{ (\text{port}_1, \text{packet}_1), ..., (\text{port}_n, \text{packet}_n) \} \]
let forward (sw, pt, pk) =
  match (sw, pk.dst_ip) with
  | (1, H1) -> [(1, pk)]
  | (1, H2) -> [(2, pk)]
  | (2, H1) -> [(1, pk)]
  | (2, H2) -> [(2, pk)]
  | _ -> []

\[ f : \text{switch} \times \text{port} \times \text{packet} \to \{(\text{port}_1,\text{packet}_1), ..., (\text{port}_n, \text{packet}_n)\} \]
let forward (sw, pt, pk) =  
    match (sw, pk.dst_ip) with  
    | (1, H1) -> [(1, pk)]  
    | (1, H2) -> [(2, pk)]  
    | (2, H1) -> [(1, pk)]  
    | (2, H2) -> [(2, pk)]  
    | _ -> []

let block_ssh (sw, pt, pk) =  
    if sw = 1 && pk.dst_port = SSH  
    then []  
    else forward (sw, pt, pk)

\[ f : \text{switch} \times \text{port} \times \text{packet} \rightarrow \{ (\text{port}_1, \text{packet}_1), \ldots, (\text{port}_n, \text{packet}_n) \} \]
let forward (sw, pt, pk) =
  match (sw, pk.dst_ip) with
  | (1, H1) -&gt; [(1, pk)]
  | (1, H2) -&gt; [(2, pk)]
  | (2, H1) -&gt; [(1, pk)]
  | (2, H2) -&gt; [(2, pk)]
  | _ -&gt; []

let block_ssh (sw, pt, pk) =
  if sw = 1 && pk.dst_port = SSH
  then []
  else forward (sw, pt, pk)

Key challenges:
• Low-level programming interface for switches
• Distributed programming

\[ f: \text{switch} \times \text{port} \times \text{packet} \rightarrow \{ (\text{port}_1, \text{packet}_1), \ldots, (\text{port}_n, \text{packet}_n) \} \]
Must encode function $f$ into flow tables, one per switch.
Low-Level Interface

Must encode function $f$ into flow tables, one per switch

$$f|_{sw} = 1$$

$$f|_{sw} = 2$$
Low-Level Interface

Must encode function $f$ into flow tables, one per switch

<table>
<thead>
<tr>
<th>Priority</th>
<th>Predicate</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Priority</th>
<th>Predicate</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Low-Level Interface

Must encode function $f$ into flow tables, one per switch

Challenges
- Predicates are a conjunction of positive literals
- Limited collection of primitive actions
Switch flow tables cannot be updated atomically

- Flow tables are huge
- Instructions only add/delete individual entries
- Must update the table live, while it is processing packets

\[ \text{table}_{sw} \subseteq f|_{sw} \]
Non-Atomic Updates

Priorities and Predicates:

<table>
<thead>
<tr>
<th>Priority</th>
<th>Predicate</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>SSH</td>
<td>Drop</td>
</tr>
<tr>
<td>5</td>
<td>dst_ip = H1</td>
<td>Fwd 1</td>
</tr>
<tr>
<td>5</td>
<td>dst_ip = H2</td>
<td>Fwd 2</td>
</tr>
</tbody>
</table>
Non-Atomic Updates

update re-ordering
Key programming challenges:
- Encoding high-level functions as tables
- Low-level, non-atomic table manipulation
There is a cottage industry in configuration-checking tools...

- FlowChecker [SafeConfig ‘10]
- AntEater [SIGCOMM ‘11]
- NICE [NSDI ‘12]
- Header Space Analysis [NSDI ‘12]
- VeriFlow [HotSDN ‘12]
- and many others...
Existing Tools

There is a cottage industry in configuration-checking tools...

- FlowChecker [SafeConfig ‘10]
- AntEater [SIGCOMM ‘11]
- NICE [NSDI ’12]
- Header Space Analysis [NSDI ’12]
- VeriFlow [HotSDN ’12]
- and many others...

These are all great tools!

But they are expensive to run, and limited to finding bugs
Existing Tools

There is a cottage industry in configuration-checking tools...

- FlowChecker [SafeConfig ‘10]
- AntEater [SIGCOMM ‘11]
- NICE [NSDI ‘12]
- Header Space Analysis [NSDI ‘12]

*ad hoc, unsound foundations*

These are all great tools!

But they are expensive to run, and limited to finding bugs...
Our Approach

• Write programs in a high-level declarative network programming language
• Use a compiler and run-time system to generate low-level instructions
• Certify the compiler and run-time system using the Coq proof assistant
• Extract to OCaml and run on real hardware
Our Approach

- Write network programs in a high-level declarative programming language
- Use a compiler and run-time system to generate low-level instructions
- Certify the compiler and run-time system using the Coq proof assistant
- Extract to OCaml and execute on real network hardware
Our Approach

• Write network programs in a high-level declarative programming language
• Use a compiler and run-time system to generate low-level instructions
• Certify the compiler and run-time system using the Coq proof assistant
• Extract to OCaml and execute on real network hardware
Certified Software Systems

Recent success stories
• seL4 microkernel [SOSP ’09]
• CompCert compiler [CACM ’09]

Tools

Textbooks
Certified Software Systems

Recent success stories
• seL4 microkernel [SOSP ’09]
• CompCert compiler [CACM ’09]

Tools

ACL2

Textbooks

Inductive pred : Type :=
  OnSwitch : Switch -> pred
  InPort : Port -> pred
  DlSrc : EthernetAddress -> pred
  DlDst : EthernetAddress -> pred
  DlVlan : option VLAN -> pred
  ...
  And : pred -> pred -> pred
  Or : pred -> pred -> pred
  Not : pred -> pred
  All : pred
  None : pred

Inductive act : Type :=
  FwdMod : Mod -> PseudoPort -> act
  ...

Inductive pol : Type :=
  Policy : pred -> list act -> pol
  Union : pol -> pol -> pol
  Restrict : pol -> pred -> pol
  ...
Recent success stories

- seL4 microkernel [SOSP '09]
- CompCert compiler [CACM '09]

Tools

Isabelle

ACL2

Textbooks

Certified Programming with Dependent Types

Write code

Prove correct
Certified Software Systems

Recent success stories
• seL4 microkernel [SOSP ’09]
• CompCert compiler [CACM ’09]

Tools

Textbooks
Certi/fi/ed Software Systems

Recent success stories
- seL4 microkernel [SOSP '09]
- CompCert compiler [CACM '09]

Tools
- ACL2
- Isabelle

Textbooks
- Software Foundations
- Certified Programming with Dependent Types

Write code
Prove correct
Extract code
Certified executable

Inductive pred : Type :=
  | OnSwitch : Switch -> pred
  | DlSrc : EthernetAddress -> pred
  | DlDst : EthernetAddress -> pred
  | DlVlan : VLAN -> pred
  ...
  | And : pred -> pred -> pred
  | Or : pred -> pred -> pred
  | Not : pred -> pred
  | All : pred
  | None : pred

Inductive act : Type :=
  | FwdMod : Mod -> PseudoPort -> act
  ...
  | Pol : pred -> list act -> pol
  | Union : pol -> pol -> pol
  | Restrict : pol -> pred -> pol

Lemma inter_wildcard_other :forall x, Wildcard_inter WildcardAll x = x.
Proof.
intros;
destruct x;
auto.
Qed.

Lemma inter_wildcard_other1 :forall x, Wildcard_inter x WildcardAll = x.
Proof.
intros;
destruct x;
auto.
Qed.

Lemma inter_exact_same :forall x, Wildcard_inter (WildcardExact x) (WildcardExact x) = WildcardExact x.
Proof.
intros.\nunfold Wildcard_inter.
destruct (eqdec x x);
intuition.
Qed.

nettleServer :: ControllerRec -> IO ()
nettleServer controller = do
  nettle <- startOpenFlowServer Nothing 6633
  switchMsgs <- newChan
  forkIO (handleOFMsgs controller switchMsgs nettle)
  forever $ do
    (switch, switchFeatures) <- retryOnExns "nettle bug" (acceptSwitch nettle)
    writeChan switchMsgs (Left $ toInteger $ handle2SwitchID switch)
    putStrLn stderr "switch: " ++ (show (handle2SwitchID switch))
    hFlush stderr
  return ()
  forkIO (handleSwitch switch switchMsgs)
  closeServer nettle
pred ::= switch = sw  
  | srcIP = n  
  | dstIP = n  
  | protocol = n  
  | pred1 && pred2  
  | pred1 || pred2  
  | !pred  
  | true  
  | false

pol ::= fwd(pt)  
  | drop  
  | if pred then pol  
  | pol1 + pol2

Example

fwd_pol = 
  if dstIP = H1 then fwd(1) + 
  if dstIP = H2 then fwd(2)

sw1_pol = 
  if !(protocol = SSH) then 
    fwd_pol
Abstract Network Semantics

**Semantics**

- $p$ – NetCore program
- $M$ – multiset of “located” packets
- $lp$ – located packet processed in this step

\[
[p](lp) = M'
\]

\[
p \vdash \{lp\} \cup M \xrightarrow{lp} M \cup M'
\]

**Features**

- Models hop-by-hop forwarding behavior of the network
- Abstracts away from the underlying distributed system
- Enables simple reasoning about network-wide properties
Compiler Correctness

**Theorem** compile_correct :
\[
\forall \text{pol} \; \text{sw} \; \text{pt} \; \text{pk},
\text{netcore_eval} \; \text{pol} \; \text{sw} \; \text{pt} \; \text{pk} = 
\text{table_eval} \; (\text{compile} \; \text{pol} \; \text{sw}) \; \text{pt} \; \text{pk}.
\]

**Formalization Highlights**

- Library of algebraic properties of tables
- New tactic for proving equalities on bags
- General-purpose table optimizer
- Key invariant: all synthesized predicates are well-formed (w.r.t. protocol types)
OpenFlow: an open, standardized protocol for programming switches
*Dell, HP, NEC, Pica8, OpenVSwitch, etc.*
OpenFlow Specification

42 pages...

...of informal prose

...diagrams and flow charts

...and C struct definitions
Models all features related to packet forwarding, and all essential asynchrony.
/ Fields to match against flows */

struct ofp_match {
    uint32_t wildcards;  /* Wildcard fields. */
    uint16_t in_port;   /* Input switch port. */
    uint8_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address. */
    uint8_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address. */
    uint16_t dl_vlan;  /* Input VLAN. */
    uint8_t dl_vlan_pcp; /* Input VLAN priority. */
    uint16_t dl_type;  /* Ethernet frame type. */
    uint8_t nw_tos;    /* IP ToS (DSCP field, 6 bits). */
    uint8_t nw_proto;  /* IP protocol or lower 8 bits of ARP opcode. */
    uint8_t pad1[1]; /* Align to 64-bits. */
    uint8_t nw_src;    /* IP source address. */
    uint8_t nw_dst;    /* IP destination address. */
    uint16_t tp_src;   /* TCP/UDP source port. */
    uint16_t tp_dst;   /* TCP/UDP destination port. */
    uint8_t pad2[2]; /* Align to 64-bits. */
};

OFP_ASSERT(sizeof(struct ofp_match) == 40);
/* Fields to match against flows */

struct ofp_match {
  uint32_t wildcards;  /* Wildcard fields. */
  uint16_t in_port;   /* Input switch port. */
  uint8_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address. */
  uint8_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address. */
  uint16_t dl_vlan;  /* Input VLAN. */
  uint8_t dl_vlan_pcp; /* Input VLAN priority. */
  uint16_t dl_type; /* Ethernet frame type. */
  uint8_t dl_type[OFP_ETH_ALEN]; /* Ethernet frame type. */
  uint8_t nw_tos; /* IP ToS (DSCP field, 6 bits). */
  uint8_t nw_proto; /* IP protocol or lower 8 bits of ARP opcode. */
  uint8_t nw_proto[OFP_ETH_ALEN]; /* IP protocol or lower 8 bits of ARP opcode. */
  uint8_t pad1[1];  /* Align to 64-bits. */
  uint8_t pad2[2];  /* Align to 64-bits. */
  uint32_t nw_src; /* IP source address. */
  uint32_t nw_dst; /* IP destination address. */
  uint16_t tp_src; /* TCP/UDP source port. */
  uint16_t tp_dst; /* TCP/UDP destination port. */
};

OFP_ASSERT(sizeof(struct ofp_match) == 40);
/* Fields to match against flows */
struct ofp_match {
    uint32_t wildcards; /* Wildcard fields. */
    uint16_t in_port; /* Input switch port. */
    uint8_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address. */
    uint8_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address. */
    uint16_t dl_vlan; /* Input VLAN. */
    uint8_t dl_vlan_pcp; /* Input VLAN priority. */
    uint16_t dl_type; /* Ethernet frame type. */
    uint8_t nw_tos; /* IP ToS (DSCP field, 6 bits). */
    uint8_t nw_proto; /* IP protocol or lower 8 bits of ARP opcode. */
    uint8_t pad1[1]; /* Align to 64-bits. */
    uint8_t nw_src; /* IP source address. */
    uint32_t nw_dst; /* IP destination address. */
    uint16_t tp_src; /* TCP/UDP source port. */
    uint16_t tp_dst; /* TCP/UDP destination port. */
};
OFP_ASSERT(sizeof(struct ofp_match) == 40);
Forwarding

/* Fields to match against flows */
struct ofp_match {
  uint32_t wildcards; /* Wildcard fields. */
  uint16_t tp_dst; /* Output port. */
  uint16_t tp_src; /* Output port. */
  uint32_t nw_dst; /* Destination IP address. */
  uint32_t nw_src; /* Source IP address. */
  char arp_opcode; /* ARP opcode. */
  char dl_type; /* Ethernet frame type. */
  char dl_vlan; /* VLAN tag. */
  char in_port; /* Input port. */
  uint32_t wildcards; /* Wildcard fields. */
};
OFP_ASSERT(sizeof(struct ofp_match) == 40);

Definition Pattern_inter (p : Pattern) :=
let d1Src := Wildcard EtherAddress.eqdec (ptrnD1Src p) (ptrnD1Src p') in
let d1Dst := Wildcard EtherAddress.eqdec (ptrnD1Dst p) (ptrnD1Dst p') in
let d1Type := Wildcard Word16.eqdec (ptrnD1Type p) (ptrnD1Type p') in
let d1Vlan := Wildcard Word16.eqdec (ptrnD1Vlan p) (ptrnD1Vlan p') in
let d1VlanPcp := Wildcardword8.eqdec (ptrnD1VlanPcp p) (ptrnD1VlanPcp p') in
let nwSrc := Wildcard Word32.eqdec (ptrnNwSrc p) (ptrnNwSrc p') in
let nwDst := Wildcard Word32.eqdec (ptrnNwDst p) (ptrnNwDst p') in
let nwProto := Wildcard Word8.eqdec (ptrnNwProto p) (ptrnNwProto p') in
let nwTos := Wildcard Word8.eqdec (ptrnNwTos p) (ptrnNwTos p') in
let tpSrc := Wildcard Word16.eqdec (ptrnTpSrc p) (ptrnTpSrc p') in
let tpDst := Wildcard Word16.eqdec (ptrnTpDst p) (ptrnTpDst p') in
let inPort := Wildcard EtherAddress.eqdec (ptrnInPort p) (ptrnInPort p') in
MkPattern d1Src d1Dst d1Type d1Vlan d1VlanPcp
  nwSrc nwDst nwProto nwTos
  tpSrc tpDst inPort.

Definition exact_pattern (pk : Packet) (pt : Word16.T) : Pattern :=
MkPattern
  (WildcardExact (pktD1Src pk)) (WildcardExact (pktD1Dst pt))
  (WildcardExact (pktD1Type pk))
  (WildcardExact (pktD1Vlan pk)) (WildcardExact (pktD1VlanPcp pk))
  (WildcardExact (pktNwSrc pk)) (WildcardExact (pktNwDst pt))
  (WildcardExact (pktNwProto pk)) (WildcardExact (pktNwTos pt))
  (WildcardExact (pktTpSrc pk)) (WildcardExact (pktTpDst pt))
  (WildcardExact pt).

Definition match_packet (pt : Word16.T) (pk : Packet) : bool :=
  negb (Pattern_is_empty (Pattern_inter (exact_pattern pk pt) pt)).
Forwarding

```c
/* Fields to match against flows */
struct ofp_match {
    uint32_t wildcards;    /* Wildcard fields. */
    uint16_t in_port;      /* Input switch port. */
    uint8_t dl_src[OFP_ETH_ALEN];  /* Ethernet source address. */
    uint8_t dl_dst[OFP_ETH_ALEN];  /* Ethernet destination address. */
    uint16_t dl_vlan;       /* Input VLAN. */
    uint16_t dl_vlan_pcp;   /* Input VLAN priority. */
    uint16_t dl_type;       /* Ethernet frame type. */
    uint8_t dl_tos;         /* IP TOS (DSAP/PSAP field, 8 bits). */
    uint8_t dl_proto;       /* IP protocol or lower 8 bits of
                            ARP Opcode. */
    uint8_t pad2[2];        /* Align to 64-bits. */
    uint32_t nw_src;        /* IP source address. */
    uint32_t nw_dst;        /* IP destination address. */
    uint16_t nw_proto;      /* IP protocol or lower 8 bits of
                            IP protocol. */
    uint8_t nw_tos;         /* IP type of service. */
    uint8_t pad1[3];        /* Align to 64-bits. */
    uint16_t tp_src;        /* TCP/UDP source port. */
    uint16_t tp_dst;        /* TCP/UDP destination port. */
};
OFP_ASSERT(sizeof(struct ofp_match) == 40);
```

Detailed model of matching, forwarding, and flow table update
Asynchrony

“In the absence of barrier messages, switches may arbitrarily reorder messages to maximize performance.”

“There is no packet output ordering guaranteed within a port.”
Asynchrony

“In the absence of barrier messages, switches may arbitrarily reorder messages to maximize performance.”

“There is no packet output ordering guaranteed within a port.”

**Definition**

\[
\text{InBuf} := \text{Bag Packet}.
\]

\[
\text{OutBuf} := \text{Bag Packet}.
\]

\[
\text{OFInBuf} := \text{Bag SwitchMsg}.
\]

\[
\text{OFOutBuf} := \text{Bag CtrlMsg}.
\]
Asynchrony

“In the absence of barrier messages, switches may arbitrarily reorder messages to maximize performance.”

“There is no packet output ordering guaranteed within a port.”

Essential asynchrony: packet buffers, message reordering, and barriers

Definition $\text{InBuf} := \text{Bag Packet}$.
Definition $\text{OutBuf} := \text{Bag Packet}$.
Definition $\text{OFInBuf} := \text{Bag SwitchMsg}$.
Definition $\text{OFOutBuf} := \text{Bag CtrlMsg}$. 
Controller Verification

pred ::= switch = sw
  | srcIP = n
  | dstIP = n
  | protocol = n
  | pred1 && pred2
  | pred1 || pred2
  | !pred
  | true
  | false

pol ::= fwd(pt)
  | drop
  | if pred then pol
  | pol1 + pol2
## Controller Bugs

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Incorrect controller function</td>
<td></td>
<td>☒</td>
<td></td>
</tr>
<tr>
<td>Incorrect flow tables</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>Update reordering bugs</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>Incorrect flow table optimizations</td>
<td>☒</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Weak Bisimulation

$(H_1,)$
Weak Bisimulation

\((H_1, \text{letter}) \xrightarrow{} (S_1, pt_1, \text{letter})\)
Weak Bisimulation

\((H_1, \text{✉}) \xrightarrow{\cdot} (S_1, pt_1, \text{✉}) \xrightarrow{\cdot} (S_2, pt_1, \text{✉})\)
Weak Bisimulation

\[(H_1, \text{message}) \to (S_1, pt_1, \text{message}) \to (S_2, pt_1, \text{message}) \to (H_2, \text{message})\]
Weak Bisimulation

\[(H_1, \text{email}) \rightarrow (S_1, pt_1, \text{email}) \rightarrow (S_2, pt_1, \text{email}) \rightarrow (H_2, \text{email})\]
Weak Bisimulation

\[(H_1, m) \rightarrow (S_1, pt_1, m) \rightarrow (S_2, pt_1, m) \rightarrow (H_2, m)\]
Weak Bisimulation

\((H_1, m) \rightarrow (S_1, pt_1, m) \rightarrow (S_2, pt_1, m) \rightarrow (H_2, m)\)
Theorem: NetCore abstract semantics is weakly bisimilar to Featherweight OpenFlow + NetCore controller
Parameterized Weak Bisimulation

Invariants

• **Safety**: at all times, the rules installed on switches are a *subset* of the controller function
• **Liveness**: the controller eventually processes all packets diverted to it by switches

Correctness Theorem

```plaintext
Module RelationDefinitions ::=  
  FwOF.FwOFRelationDefinitions.Make (AtomsAndController).
...
Theorem fwoff_abst_weak_bisim : 
  weak_bisimulation 
  concreteStep 
  abstractStep 
  bisim_relation.
```
Experience

Source
- ~12,000 lines of Coq
- ~1,600 lines of OCaml

Deployments
- Rack in Cornell systems lab
- Pronto 3290 Gigabit switch
- Arjun’s apartment
- TP-Link Wifi Router

Applications
- Host discovery*
- Shortest-path routing
- Spanning tree
- Traffic monitoring*
- Access control

* uses partially-verified NetCore

Experiments

Faster than POX, a popular Python controller
Thank You

Arjun Guha
Postdoc→UMass

Mark Reitblatt
PhD student

www.github.com/frenetic-lang