SDN Foundations

Nate Foster
Cornell University
Networks Today

Reasoning about network behavior is extremely difficult…
Networks Today

Reasoning about network behavior is extremely difficult…
Reasoning about network behavior is extremely difficult…

…due to the proliferation of devices, protocols, languages
Reasoning about network behavior is extremely difficult…

Does correctness matter? The Internet is best effort…

…due to the proliferation of devices, protocols, languages
Networks Today

Reasoning about network behavior is extremely difficult…

Does correctness matter? The Internet is best effort…
…the end-to-end principle says that hosts are best equipped to deal with failures!

…due to the proliferation of devices, protocols, languages
Example: 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**
Example: 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.

Even technically sophisticated companies are struggling to build networks that provide reliable performance.

**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.
Example: Cloud Computing

Would you relocate critical infrastructure to the cloud…
Example: Cloud Computing

Would you relocate critical infrastructure to the cloud…

…if your traffic was *not* guaranteed to be isolated from other tenants during periods of routine maintenance?
Example: Cloud Computing

Would you relocate critical infrastructure to the cloud...

...if your traffic was not guaranteed to be isolated from other tenants during periods of routine maintenance?
Example: Cloud Computing

Would you relocate critical infrastructure to the cloud…

Networks are critical for ensuring the security of many systems… so it is important they function as expected

…if your traffic was not guaranteed to be isolated from other tenants during periods of routine maintenance?
Software-Defined Networking

Controller Platform
Frenetic, OpenDaylight, etc.

OpenFlow API

OpenFlow-enabled switches
Pica8, Dell, NEC, HP, and many others
Software-Defined Networking

Controller Platform
- Frenetic, OpenDaylight, etc.

OpenFlow API

OpenFlow-enabled switches
- Pica8, Dell, NEC, HP, and many others

Your code goes here!
Software-Defined Networking

Enabling use of reasoning techniques typically associated with the programming languages and verification communities

Your code goes here!

Controller Platform
Frenetic, OpenDaylight, etc.

OpenFlow API

OpenFlow-enabled switches
Pica8, Dell, NEC, HP, and many others
Software-Defined Networking

Enabling use of reasoning techniques typically associated with the programming languages and verification communities

Examples:
- HSA, ATPG, NetPlumber
- Anteater, VeriFlow
- VeriCon
- FlowLog
- NetKAT

Controller Platform
Frenetic, OpenDaylight, etc.

OpenFlow API

OpenFlow-enabled switches
Pica8, Dell, NEC, HP, and many others
System Model

Application

Configurations

Run-Time System
System Model

Application

Configurations

Run-Time System

High-level application logic
Often expressed as a finite-state machine on network events (topology changes, new connections, etc.)
System Model

Network-wide packet-processing function

Expressed in terms of a set of forwarding tables, one per switch in the network
System Model

Configurations

Run-Time System

Application

Code that manages the rules installed on switches

Translate configuration updates into sequences of OpenFlow instructions
System Model

Forwarding elements that implement packet-processing functionality efficiently in hardware
This Talk

Machine Model [PLDI ’13]

Language Model [POPL’12, NSDI ’13, POPL ’14]

Run-Time Model [SIGCOMM ’12, SYNT’13]
Machine Model
Forwarding elements that implement packet-processing functionality efficiently in hardware.
Implementing Static Functions

Suppose we are given a packet-processing function:
\[ f \in \text{Packet} \rightarrow \text{Packet Set} \]

What code should we write on the controller to implement \( f \) correctly?

Even this simple task turns out to be non-trivial, due to features of the hardware.

Your code goes here!
Controller Platform
Frenetic, OpenDaylight, etc.

OpenFlow API

OpenFlow-enabled switches
Pica8, Dell, NEC, HP, and many others
OpenFlow API

Controller

OpenFlow Switch
OpenFlow API

Controller to switch:
- `flow_mod`
- `packet_out`
- `barrier_request`
- `stats_request`
OpenFlow API

Controller to switch:
- `flow_mod`
- `packet_out`
- `barrier_request`
- `stats_request`

Switch to controller:
- `switch_connected`
- `switch_disconnected`
- `port_status`
- `packet_in`
- `barrier_reply`
- `stats_reply`
module MyApplication = struct

  include OxStart.DefaultTutorialHandlers

  let switch_connected (sw : switchId) : unit =
    send_flow_mod sw 0l (del_flow 0 any [])
    send_flow_mod sw 0l (add_flow 0 any [Flood])

  let packet_in (sw : switchId) (xid : xid) (pk : packetIn) : unit =
    send_packet_out sw 0l
    { output_payload = pk.input_payload;
      port_id = None;
      apply_actions = [Flood] }

module Controller = OxStart.Make (MyApplication)
open OxPlatform
open OpenFlow0x01_Core

module MyApplication = struct
    include OxStart.DefaultTutorialHandlers

    let switch_connected (sw : switchId) : unit =
        send_flow_mod sw 0l (del_flow θ any [])
        send_flow_mod sw 0l (add_flow θ any [Flood])

    let packet_in (sw : switchId) (xid : xid) (pk : packetIn) : unit =
        send_packet_out sw 0l
        { output_payload = pk.input_payload;
          port_id = None;
          apply_actions = [Flood] }

end

module Controller = OxStart.Make (MyApplication)
Repeater Hub in Ox

open OxPlatform
open OpenFlow0x01_Core

module MyApplication = struct

include OxStart.DefaultTutorialHandlers

let switch_connected (sw : switchId) : unit =
send_flow_mod sw 0l (del_flow 0 any [])
send_flow_mod sw 0l (add_flow 0 any [Flood])

let packet_in (sw : switchId) (xid : xid) (pk : packetIn) : unit =
send_packet_out sw 0l { output_payload = pk.input_payload;
port_id = None;
apply_actions = [Flood] }

end

module Controller = OxStart.Make (MyApplication)
open OxPlatform
open OpenFlow0x01_Core

module MyApplication = struct

include OxStart.DefaultTutorialHandlers

let switch_connected (sw : switchId) : unit =
  send_flow_mod sw 0l (del_flow 0 any [])
  send_flow_mod sw 0l (add_flow 0 any [Flood])

let packet_in (sw : switchId) (xid : xid) (pk : packetIn) : unit =
  send_flow_mod sw 0l (add_flow 0 any [Flood])
  send_packet_out sw 0l
  { output_payload = pk.input_payload;
    port_id = None;
    apply_actions = [Flood] }
end

module Controller = OxStart.Make (MyApplication)
Distributed Programming

Even updating a single switch is challenging:

- Flow tables are huge
- OpenFlow only provides instructions for adding and removing single entries
- Must update tables while they are live and processing packets
Instruction Reordering

<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>
Instruction Reordering

Update re-ordering
OpenFlow Specification

42 pages...

...of informal prose

...diagrams and flow charts

...and C struct definitions
Recent success stories

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

Tools

- ACL2

Textbooks

- Software Foundations
- Certified Programming with Dependent Types

Certified executable

Write code
Prove correct
Extract code

Inductive `pred` : Type :=
  OnSwitch : Switch > pred
  InPort : Port > pred
  DlSrc : EthernetAddress > pred
  DlDst : EthernetAddress > pred
  DlVlan : VLAN > pred
  All
  Not
  None

Inductive `act` : Type :=
  FwdMod : Mod > PseudoPort > act

Inductive `pol` : Type :=
  Policy : pred > list act > pol
  Union : pol > pol > pol
  Restrict : pol > pred > pol

Lemma `inter_wildcard_other` : forall `x`, `Wildcard_interWildcardAll 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. unfold `Wildcard_inter`. destruct (eqdec `x x`) & intuition. Qed.

nettleServer :: ControllerRec -> IO ()
nettleServer controller = do
nettle <- startOpenFlowServer Nothing 6633
switchMsgs <- memChan
forkIO (handleOFMsgs controller switchMsgs nettle)
forever $ do
  (switch,
   switchFeatures) <- retryOnExns
    "nettle bug" (acceptSwitch nettle)
  writeChan switchMsgs
    (Left $ toInteger $ handle2SwitchID switch)
  hPutStrLn stderr ("switch: 
    " ++ (show (handle2SwitchID switch)))
  hFlush stderr
  return ()
forkIO (handleSwitch switch switchMsgs) closeServer nettle
Featherweight OpenFlow

Syntax

Models all features related to packet forwarding, and all essential asynchrony

Switch Components
- Rule table
- Rule table modifier
- Ports on switch
- Consumed packets
- Produced packets
- Messages from controller
- Messages to controller

Switch-Recv-Barrier

Switch-PktOut

Switch-FlowMod

(Pkt-Process)

(Send-Wire)

(Recv-Wire)

(Switch-FlowMod)

(Switch-PktOut)

(Switch-Recv-Ctrl)

(Switch-Ctrl-Send)

(Switch-Ctrl-Recv)

(Switch-Ctrl-Bar)

(Switch-Send-Ctrl)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)

(Switch-Ctrl-Bar)
/** Fields to match against flows */

```c
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. */
    uint8_t pad1[1]; /* Align to 64-bits. */
    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 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);
```
Forwarding

/* 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_5 dl_type;  /* Ethernet frame type. */
    uint8_t dl_vlan;     /* Input VLAN. */
    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 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. */
  uint8_t pad1[1]; /* Align to 64-bits. */
  uint16_5 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 pad2[2]; /* Align to 64-bits. */
  uint16_t nw_src; /* IP source address. */
  uint16_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);

Record Pattern : Type := MkPattern {
  dlSrc : Wildcard EthernetAddress
  dlDst : Wildcard EthernetAddress
  dlType : Wildcard EthernetType
  dlVlan : Wildcard VLAN
  dlVlanPcp : Wildcard VLANPriority
  nwSrc : Wildcard IPAddress
  nwDst : Wildcard IPAddress
  nwProto : Wildcard IPProtocol
  nwTos : Wildcard IPTypeOfService
  tpSrc : Wildcard TransportPort
  tpDst : Wildcard TransportPort
  inPort : Wildcard Port
}.
/* 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. */
    uint8_t pad1[2]; /* Align to 64-bits. */
    uint16_t d1_type; /* Ethernet frame type. */
    uint8_t pad2[2]; /* Align to 64-bits. */
    uint16_t nw_type; /* IP protocol or lower 8 bits of ARP opcode. */
    uint8_t nw_proto; /* IP protocol or lower 8 bits of */
    uint16_t nw_tos; /* IP ToS (DSCP field, 6 bits). */
    uint8_t nw_proto2; /* IP protocol or lower 8 bits of */
    uint16_t nw_dst; /* IP destination address. */
    uint8_t nw_dst2; /* IP destination address. */
    uint8_t nw_src; /* IP source address. */
    uint8_t nw_src2; /* IP source address. */
    uint16_t tp_dst; /* TCP/UDP destination port. */
    uint16_t tp_src; /* TCP/UDP source port. */
    uint8_t pad3[2]; /* Wildcard fields. */
    uint8_t pad4[2]; /* Wildcard fields. */
    uint8_t pad5[2]; /* Wildcard fields. */
    uint8_t pad6[2]; /* Wildcard fields. */
    uint8_t pad7[2]; /* Wildcard fields. */
    uint8_t pad8[2]; /* Wildcard fields. */
    struct /* Fields to match against flows */
    
    OFP_ASSERT(sizeof(struct ofp_match) == 40);

    Record Pattern : Type := MkPattern {
        d1Src := Wildcard EtherentAddress
        d1Dst := Wildcard EthernetAddress
        d1Type := Wildcard EthernetType
        d1Vlan := Wildcard VLAN
        d1VlanPcp := Wildcard VLANPriority
        nwSrc := Wildcard IPAddress
        nwDst := Wildcard IPAddress
        nwProto := Wildcard IPProtocol
        nwTos := Wildcard IPTypeOfService
        tpSrc := Wildcard TransportPort
        tpDst := Wildcard TransportPort
        inPort := Wildcard Port
    }.

Definition Pattern_inter (p : Pattern) :=
    let d1Src := Wildcard_inter EthernetAddress
    let d1Dst := Wildcard_inter EthernetAddress
    let d1Type := Wildcard_inter Word16
    let d1Vlan := Wildcard_inter Word16
    let d1VlanPcp := Wildcard_inter Word16
    let nwSrc := Wildcard_inter Word32
    let nwDst := Wildcard_inter Word32
    let nwProto := Wildcard_inter Word8
    let nwTos := Wildcard_inter Word8
    let tpSrc := Wildcard_inter Word16
    let tpDst := Wildcard_inter Word16
    let inPort := Wildcard_inter Word16
    let MkPattern d1Src d1Dst d1Type d1Vlan d1VlanPcp
    let nwSrc nwDst nwProto nwTos
    let tpSrc tpDst
    inPort.

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

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

Detailed model of matching, forwarding, and flow table update

Record Pattern : Type := MkPattern {
  d1Src : Wildcard EthernetAddress ·
  d1Dst : Wildcard EthernetAddress ·
  d1Type : Wildcard EthernetType ·
  d1Vlan : Wildcard VLAN ·
  d1VlanPcp : Wildcard VLANPriority ·
  nwSrc : Wildcard IPAddress ·
  nwDist : Wildcard IPAddress ·
  nwProto : Wildcard IPPROTOProtocol ·
  nwTos : Wildcard IPTypeOfService ·
  tpSrc : Wildcard TransportPort ·
  tpDst : Wildcard TransportPort ·
  inPort : Wildcard Port ·
}.

let d1Dist := Wildcard_inter_word16.eqdec (ptrnD1Dist p) (ptrnD1Dist p') in
let d1Type := Wildcard_inter_word16.eqdec (ptrnD1Type p) (ptrnD1Type p') in
let d1Vlan := Wildcard_inter_word16.eqdec (ptrnD1Vlan p) (ptrnD1Vlan p') in
let d1VlanPcp := Wildcard_inter_word8.eqdec (ptrnD1VlanPcp p) (ptrnD1VlanPcp p') in
let nwSrc := Wildcard_inter_word32.eqdec (ptrnNwSrc p) (ptrnNwSrc p') in
let nwDist := Wildcard_inter_word32.eqdec (ptrnNwDist p) (ptrnNwDist p') in
let nwProto := Wildcard_inter_word8.eqdec (ptrnNwProto p) (ptrnNwProto p') in
let nwTos := Wildcard_inter_word8.eqdec (ptrnNwTos p) (ptrnNwTos p') in
let tpSrc := Wildcard_inter_word16.eqdec (ptrnTpSrc p) (ptrnTpSrc p') in
let tpDst := Wildcard_inter_word16.eqdec (ptrnTpDst p) (ptrnTpDst p') in
let inPort := Wildcard_inter_word16.eqdec (ptrnInPort p) (ptrnInPort p') in
MkPattern d1Src d1Dist d1Type d1Vlan d1VlanPcp
  nwSrc nwDist nwProto nwTos
  tpSrc tpDst
  inPort.

Definition exact_pattern (pk : Packet) (pt : Word16.T) : Pattern :=
  MkPattern
    (WildcardExact (pktD1Src pk))
    (WildcardExact (pktD1Dist pk))
    (WildcardExact (pktD1Type pk))
    (WildcardExact (pktD1Vlan pk))
    (WildcardExact (pktD1VlanPcp pk))
    (WildcardExact (pktNwSrc pk))
    (WildcardExact (pktNwDist pk))
    (WildcardExact (pktNwProto pk))
    (WildcardExact (pktNwTos pk))
    (WildcardOf_option (pktTpSrc pk))
    (WildcardOf_option (pktTpDst pk))
    (WildcardExact pt).

  negb (Pattern_is_empty (Pattern_inter (exact_pattern pk pt) pat)).
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**

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

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

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

\[
\text{OFOOutBuf} := \text{Bag CtrlMsg.}
\]
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 \( \text{OFOutBuf} := \text{Bag CtrlMsg} \).
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

Theorem

```plaintext
Module RelationDefinitions :=
   FwOF.FwOFRelationDefinitions.Make (AtomsAndController).
...
Theorem fwof_abst_weak_bisim :
   weak_bisimulation
   concreteStep
   abstractStep
   bisim_relation.
```
Weak Bisimulation

\((H_1, \text{message})\)
Weak Bisimulation

\((H_1, \text{\includegraphics{envelope}}) \rightarrow (S_1, \text{pt}_1, \text{\includegraphics{envelope}})\)
Weak Bisimulation

\((H_1, \text{envelope}) \longrightarrow (S_1, pt_1, \text{envelope}) \longrightarrow (S_2, pt_1, \text{envelope})\)
Weak Bisimulation

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

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

\((H_1, \text{Mail}) \rightarrow (S_1, pt_1, \text{Mail}) \rightarrow (S_2, pt_1, \text{Mail}) \rightarrow (H_2, \text{Mail})\)
Weak Bisimulation

\[(H_1, \text{packet}) \rightarrow (S_1, pt_1, \text{packet}) \rightarrow (S_2, pt_1, \text{packet}) \rightarrow (H_2, \text{packet})\]
Theorem: packet-processing function $f$ is weakly bisimilar to a machine comprising OpenFlow switches and a controller.
Language Model
Network-wide packet-processing function

Expressed in terms of a set of forwarding tables, one per switch in the network

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcip=1.2.3.4, tcpdst = 22</td>
<td>Count, Drop</td>
</tr>
<tr>
<td>srcip=1.2.3.4,</td>
<td>Forward 1, Count</td>
</tr>
<tr>
<td>srcip=1.2.3.4,</td>
<td>Forward 2, Count</td>
</tr>
<tr>
<td>srcip=1.2.3.4</td>
<td>Count</td>
</tr>
<tr>
<td>tcpdst = 22</td>
<td>Drop</td>
</tr>
</tbody>
</table>
Machine Language

OpenFlow is a machine language

Programmers must think in terms of low-level concepts such as:

- Flow tables
- Matches
- Priorities
- Timeouts
- Events
- Callbacks

This complicates programs and makes reasoning difficult
Programming Language
Programming Language

**Packets**
A record comprising the location of the packet and a collection of header values

\[
\{\text{switch}=n_1, \ \text{port}=n_2, \ \text{ethSrc}=n_3, \ldots\}
\]
Programming Language

**Packets**
A record comprising the location of the packet and a collection of header values

\{\text{switch}=n_1, \text{ port}=n_2, \text{ ethSrc}=n_3, \ldots\}

**Functions**
A program describes a total function from packets to sets of packets
Programming Language

**Packets**
A record comprising the location of the packet and a collection of header values

\{\text{switch}=n_1, \text{port}=n_2, \text{ethSrc}=n_3, \ldots\}

**Functions**
A program describes a total function from packets to sets of packets

**Predicates**
Restrict the behavior of a program to a particular set of packets using predicates
### Programming Language

#### Packets
A record comprising the location of the packet and a collection of header values

\[
\{\text{switch}=n_1, \text{port}=n_2, \text{ethSrc}=n_3, \ldots\}
\]

#### Functions
A program describes a total function from packets to sets of packets

#### Predicates
Restrict the behavior of a program to a particular set of packets using predicates

#### Combinators
Operators for combining smaller programs into larger ones
Programming Language

**Packets**
A record comprising the location of the packet and a collection of header values

\{\text{switch}=n_1, \text{port}=n_2, \text{ethSrc}=n_3, \ldots\}

**Functions**
A program describes a total function from packets to sets of packets

*Functional “see every packet” abstraction*

**Predicates**
Restrict the behavior of a program to a particular set of packets using predicates

**Combinators**
Operators for combining smaller programs into larger ones
Network-Wide Programming

What features should an SDN language provide?
Network-Wide Programming

What features should an SDN language provide?

- Packet predicates
- Packet transformations
Network-Wide Programming

What features should an SDN language provide?

- Packet predicates
- Packet transformations
- Path construction
Network-Wide Programming

What features should an SDN language provide?

• Packet predicates
• Packet transformations
• Path construction
• Path concatenation
What features should an SDN language provide?

- Packet predicates
- Packet transformations
- Path construction
- Path concatenation
- Path union
Network-Wide Programming

What features should an SDN language provide?

- Packet predicates
- Packet transformations
- Path construction
- Path concatenation
- Path union
- Path iteration
NetKAT Language

\[ f ::= \text{switch} \mid \text{inport} \mid \text{srncmac} \mid \text{dstmac} \mid \ldots \]

\[ a, b, c ::= 0 \quad (* \text{false} *) \]
\[ 1 \quad (* \text{true} *) \]
\[ f = n \quad (* \text{test} *) \]
\[ a_1 + a_2 \quad (* \text{disjunction} *) \]
\[ a_1 \cdot a_2 \quad (* \text{conjunction} *) \]
\[ !a \quad (* \text{negation} *) \]

\[ p, q, r ::= a \quad (* \text{filter} *) \]
\[ f := n \quad (* \text{modification} *) \]
\[ p_1 + p_2 \quad (* \text{union} *) \]
\[ p_1 \cdot p_2 \quad (* \text{sequence} *) \]
\[ p^* \quad (* \text{iteration} *) \]
\[ \text{dup} \quad (* \text{duplication} *) \]
NetKAT Language

\[
f ::= \text{switch} | \text{inport} | \text{srcref} | \text{dstref} | ... \\
a, b, c ::= 0 \quad (* \text{false} *) \\\n\quad | 1 \quad (* \text{true} *) \\\n\quad | f = n \quad (* \text{test} *) \\\n\quad | a_1 + a_2 \quad (* \text{disjunction} *) \\\n\quad | a_1 \cdot a_2 \quad (* \text{conjunction} *) \\\n\quad | !a \quad (* \text{negation} *) \\\np, q, r ::= a \quad (* \text{filter} *) \\\n\quad | f ::= n \quad (* \text{modification} *) \\\n\quad | p_1 + p_2 \quad (* \text{union} *) \\\n\quad | p_1 \cdot p_2 \quad (* \text{sequence} *) \\\n\quad | p^* \quad (* \text{iteration} *) \\\n\quad | \text{dup} \quad (* \text{duplication} *)
\]

\[
\text{if } a \text{ then } p_1 \text{ else } p_2 \triangleq (a \cdot p_1) + (!a \cdot p_2)
\]
Basic Primitives

if !(dstport = 22) then
  if srcip = 10.0.0.0/8 then port := 1
  else port := 2
else drop

Firewall

Controller Platform
Basic Primitives

if !(dstport = 22) then
    if srcip = 10.0.0.0/8 then port := 1
    else port := 2
else drop

Firewall

Controller Platform

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstport=22</td>
<td>Drop</td>
</tr>
<tr>
<td>srcip=10.0.0.0/8</td>
<td>Forward 1</td>
</tr>
<tr>
<td>*</td>
<td>Forward 2</td>
</tr>
</tbody>
</table>
if srcip = 1.2.3.4 then port := 3

if dstip = 10.0.0.1 then port := 1
else if dstip = 10.0.0.1 then port := 2
if srcip = 1.2.3.4 then port := 3

if dstip = 10.0.0.1 then port := 1
else if dstip = 10.0.0.1 then port := 2

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcip=1.2.3.4, dstip=10.0.0.1</td>
<td>Forward 1, Forward 3</td>
</tr>
<tr>
<td>srcip=1.2.3.4, dstip=10.0.0.2</td>
<td>Forward 2, Forward 3</td>
</tr>
<tr>
<td>srcip=1.2.3.4</td>
<td>Forward 3</td>
</tr>
<tr>
<td>dstip=10.0.0.1</td>
<td>Forward 1</td>
</tr>
<tr>
<td>dstip=10.0.0.2</td>
<td>Forward 2</td>
</tr>
</tbody>
</table>
if srcip = *0 then dstip := 10.0.0.1
else if srcip = *1 then dstip := 10.0.0.2

if dstip = 10.0.0.1 then port := 1
else if dstip = 10.0.0.2 then port := 2
if srcip = *0 then dstip := 10.0.0.1
else if srcip = *1 then dstip := 10.0.0.2

if dstip = 10.0.0.1 then port := 1
else if dstip = 10.0.0.2 then port := 2
Encoding Tables

Forwarding tables can be expressed as NetKAT policies

*OpenFlow Normal Form (ONF)*

\[
\text{fwd} ::= f_1 := n_1 \cdot \ldots \cdot f_k := n_k + \text{fwd}
\]

\[
| 0 \\
\]

\[
\text{pat} ::= f = n \cdot \text{pat}
\]

\[
| 1 \\
\]

\[
\text{tbl} ::= \text{if pat then fwd else tbl}
\]

\[
| 0 \\
\]
Encoding Tables

Forwarding tables can be expressed as NetKAT policies

OpenFlow Normal Form (ONF)

\[ \text{fwd} ::= f_1 := n_1 \cdot \ldots \cdot f_k := n_k + \text{fwd} \]

\[ | 0 \]

\[ \text{pat} ::= f = n \cdot \text{pat} \]

\[ | 1 \]

\[ \text{tbl} ::= \text{if pat then fwd else tbl} \]

\[ | 0 \]

Pattern | Actions
---|---
\text{dstport}=22 | Drop
\text{srcip}=10.0.0.0/8 | Forward 1
\* | Forward 2

\[ \text{if dstport}=22 \text{ then } 0 \]
\[ \text{else if srcip}=10.0.0.1 \text{ then } \text{port} := 1 \]
\[ \text{else if 1 then } \text{port} := 2 \]
\[ \text{else 0} \]
Encoding Tables

Forwarding tables can be expressed as NetKAT policies

**OpenFlow Normal Form (ONF)**

\[
\begin{align*}
\text{fwd} & := f_1 := n_1 \cdot \ldots \cdot f_k := n_k + \text{fwd} \\
| & 0 \\
\text{pat} & := f = n \cdot \text{pat} \\
| & 1 \\
\text{tbl} & := \text{if pat then fwd else tbl} \\
| & 0
\end{align*}
\]

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstport=22</td>
<td>Drop</td>
</tr>
<tr>
<td>srcip=10.0.0.0/8</td>
<td>Forward 1</td>
</tr>
<tr>
<td>*</td>
<td>Forward 2</td>
</tr>
</tbody>
</table>

NetKAT compiler rewrites (local) policies into tables

This encoding also facilitates using NetKAT as the “composition substrate” for other platforms
Encoding Topologies

Links can be modeled as simple policies that forward packets from one end to the other, and topologies as unions of links.
Links can be modeled as simple policies that forward packets from one end to the other, and topologies as unions of links.
Encoding Topologies

Links can be modeled as simple policies that forward packets from one end to the other, and topologies as unions of links.

**Topology Normal Form**

\[
\begin{align*}
\text{lpred} & ::= \text{switch} \equiv n \cdot \text{port} \equiv n \\
\text{lpol} & ::= \text{switch} \equiv n \cdot \text{port} \equiv n \\
\text{link} & ::= \text{lpred} \cdot \text{lpol} \\
\text{topo} & ::= \text{link} + \text{topo} \\
& | \text{0}
\end{align*}
\]
Encoding Topologies

Links can be modeled as simple policies that forward packets from one end to the other, and topologies as unions of links.

**Topology Normal Form**

```
lpred ::= switch=n • port=n
lpol ::= switch:=n • port:=n
link ::= lpred • lpol
topo ::= link + topo | 0
```

```
switch=A • port=1 • switch:=B • port:=2 +
switch=B • port=2 • switch:=A • port:=1 +
switch=B • port=1 • switch:=C • port:=2 +
switch=C • port=2 • switch:=B • port:=1 +
0
```
Encoding Networks

Putting all these pieces together, an entire network can be modeled by interleaving policy and topology processing steps.
Encoding Networks

Putting all these pieces together, an entire network can be modeled by interleaving policy and topology processing steps.
Encoding Networks

Putting all these pieces together, an entire network can be modeled by interleaving policy and topology processing steps.
Encoding Networks

Putting all these pieces together, an entire network can be modeled by interleaving policy and topology processing steps.
Encoding Networks

Putting all these pieces together, an entire network can be modeled by interleaving policy and topology processing steps.
Encoding Networks

Putting all these pieces together, an entire network can be modeled by interleaving policy and topology processing steps

\[
\text{id} + (\text{policy} \cdot \text{topo})
\]
Encoding Networks

Putting all these pieces together, an entire network can be modeled by interleaving policy and topology processing steps.

\[
\text{id} + (\text{policy} \cdot \text{topo}) + (\text{policy} \cdot \text{topo} \cdot \text{policy} \cdot \text{topo})
\]
Encoding Networks

Putting all these pieces together, an entire network can be modeled by interleaving policy and topology processing steps.

\[
\text{id} + (\text{policy} \cdot \text{topo}) + (\text{policy} \cdot \text{topo} \cdot \text{policy} \cdot \text{topo}) + (\text{policy} \cdot \text{topo} \cdot \text{policy} \cdot \text{topo} \cdot \text{policy} \cdot \text{topo})
\]
Encoding Networks

Putting all these pieces together, an entire network can be modeled by interleaving policy and topology processing steps.

\[
\text{id} + \ (\text{policy} \cdot \text{topo}) + \ (\text{policy} \cdot \text{topo} \cdot \text{policy} \cdot \text{topo}) + \ (\text{policy} \cdot \text{topo} \cdot \text{policy} \cdot \text{topo} \cdot \text{policy} \cdot \text{topo}) + \ \ldots + \ (\text{policy} \cdot \text{topo})^* 
\]
Semantic Foundations

Unlike previous network programming languages, the design of NetKAT is not an accident!

Its foundations rest upon canonical mathematical structure:

- Regular operators (+, ⋅, *) encode paths through topology
- Boolean operators (+, ⋅, !) encode forwarding tables

Such structures are called *Kleene Algebras with Tests (KAT)* [Kozen ’96]

KAT has an accompanying proof system for establishing equivalences of the form $p \sim q$

Many reasoning tasks can be reduced to checking equivalences between terms
**NetKAT Proof System**

### Kleene Algebra Axioms

- \( p + (q + r) \sim (p + q) + r \)
- \( p + q \sim q + p \)
- \( p + 0 \sim p \)
- \( p + p \sim p \)
- \( p \cdot (q \cdot r) \sim (p \cdot q) \cdot r \)
- \( p \cdot (q + r) \sim p \cdot q + p \cdot r \)
- \( (p + q) \cdot r \sim p \cdot r + q \cdot r \)
- \( 1 \cdot p \sim p \)
- \( p \sim p \cdot 1 \)
- \( 0 \cdot p \sim 0 \)
- \( p \cdot 0 \sim 0 \)
- \( 1 + p \cdot p^* \sim p^* \)
- \( 1 + p^* \cdot p \sim p^* \)
- \( p + q \cdot r + r \sim r \Rightarrow p^* \cdot q + r \sim r \)
- \( p + q \cdot r + q \sim q \Rightarrow p \cdot r^* + q \sim q \)

### Boolean Algebra Axioms

- \( a + (b \cdot c) \sim (a + b) \cdot (a + c) \)
- \( a + 1 \sim 1 \)
- \( a + !a \sim 1 \)
- \( a \cdot b \sim b \cdot a \)
- \( a \cdot !a \sim 0 \)
- \( a \& a \sim a \)

### Packet Axioms

- \( f := n \cdot f':= n' \sim f' := n' \cdot f := n \) if \( f \neq f' \)
- \( f := n \cdot f' = n' \sim f' = n' \cdot f := n \) if \( f \neq f' \)
- \( f := n \cdot f := n \sim f := n \)
- \( f := n \cdot f := n \sim f := n \)
- \( f := n \cdot f := n' \sim f := n' \)
- \( f := n \cdot f := n' \sim 0 \) if \( n \neq n' \)
- \( \text{dup} \cdot f := n \sim f := n \cdot \text{dup} \)
Network-Wide Reachability

Given:
- Ingress predicate: switch = $s_1$
- Egress predicate: switch = $s_{21}$
- Topology: $t$
- Switch program: $p$

Check:
- $switch = s_1 \land switch := s_{21} + (p \cdot t)^* \sim (p \cdot t)^*$
- $switch = s_1 \land (p \cdot t)^* \land switch = s_{21} \sim 0$
Application: Rule Optimization

Given a program and a topology:

We want to be able to answer questions like:

“Will my network behave the same if I put firewall rules on A or B (or both)?”
Code Motion Proof

\[\text{in} \cdot \text{SSH} \cdot (p_A \cdot t)^* \cdot \text{out} \]
\[\equiv \{ \text{KAT-INVARINT, definition } p_A \} \]
\[\text{in} \cdot \text{SSH} \cdot ((a_A \cdot \neg \text{SSH} \cdot p + a_B \cdot p) \cdot t \cdot \text{SSH})^* \cdot \text{out} \]
\[\equiv \{ \text{KA-SEQ-DIST-R} \} \]
\[\text{in} \cdot \text{SSH} \cdot (a_A \cdot \neg \text{SSH} \cdot p \cdot t \cdot \text{SSH} + a_B \cdot p \cdot t \cdot \text{SSH})^* \cdot \text{out} \]
\[\equiv \{ \text{KAT-COMMUTE} \} \]
\[\text{in} \cdot \text{SSH} \cdot (a_A \cdot \neg \text{SSH} \cdot \text{SSH} \cdot p \cdot t + a_B \cdot p \cdot t \cdot \text{SSH})^* \cdot \text{out} \]
\[\equiv \{ \text{BA-CONTRA} \} \]
\[\text{in} \cdot \text{SSH} \cdot (a_A \cdot 0 \cdot p \cdot t + a_B \cdot p \cdot t \cdot \text{SSH})^* \cdot \text{out} \]
\[\equiv \{ \text{KA-SEQ-ZERO/ZERO-SEQ, KA-PLUS-COMM, KA-PLUS-ZERO} \} \]
\[\text{in} \cdot \text{SSH} \cdot (a_B \cdot p \cdot t \cdot \text{SSH})^* \cdot \text{out} \]
\[\equiv \{ \text{KA-UNROLL-L} \} \]
\[\text{in} \cdot \text{SSH} \cdot (1 + (a_B \cdot p \cdot t \cdot \text{SSH})^* \cdot (a_B \cdot p \cdot t \cdot \text{SSH}))^* \cdot \text{out} \]
\[\equiv \{ \text{KA-SEQ-DIST-L, KA-SEQ-DIST-R, definition out} \} \]
\[\text{in} \cdot \text{SSH} \cdot a_B \cdot a_2 + \]
\[\text{in} \cdot \text{SSH} \cdot a_B \cdot p \cdot t \cdot \text{SSH} \cdot (a_B \cdot p \cdot t \cdot \text{SSH})^* \cdot a_B \cdot a_2 \]
\[\equiv \{ \text{KAT-COMMUTE} \} \]
\[\text{in} \cdot a_B \cdot \text{SSH} \cdot a_2 + \]
\[\text{in} \cdot a_B \cdot \text{SSH} \cdot p \cdot t \cdot \text{SSH} \cdot (a_B \cdot p \cdot t \cdot \text{SSH})^* \cdot a_B \cdot a_2 \]
\[\equiv \{ \text{Lemma 1} \} \]
\[0 + 0 \]
\[\equiv \{ \text{KA-PLUS-IDEM} \} \]
\[0 + 0 \]
\[\equiv \{ \text{Lemma 1, Lemma 2} \} \]
\[\text{in} \cdot a_B \cdot \text{SSH} \cdot a_2 + \]
\[\text{in} \cdot \text{SSH} \cdot (a_A \cdot p \cdot t \cdot \text{SSH})^* \cdot p \cdot \text{SSH} \cdot a_A \cdot t \cdot \text{out} \]
\[\equiv \{ \text{KAT-COMMUTE, definition out} \} \]
\[\text{in} \cdot \text{SSH} \cdot \text{out} + \]
\[\text{in} \cdot \text{SSH} \cdot (a_A \cdot p \cdot t \cdot \text{SSH})^* \cdot a_A \cdot p \cdot t \cdot \text{SSH} \cdot \text{out} \]
\[\equiv \{ \text{KA-SEQ-DIST-L, KA-SEQ-DIST-R} \} \]
\[\text{in} \cdot \text{SSH} \cdot (1 + (a_A \cdot p \cdot t \cdot \text{SSH})^* \cdot (a_A \cdot p \cdot t \cdot \text{SSH})) \cdot \text{out} \]
\[\equiv \{ \text{KA-UNROLL-R} \} \]
\[\text{in} \cdot \text{SSH} \cdot (a_A \cdot p \cdot t \cdot \text{SSH})^* \cdot \text{out} \]
\[\equiv \{ \text{KA-SEQ-ZERO/ZERO-SEQ, KA-PLUS-ZERO} \} \]
\[\text{in} \cdot \text{SSH} \cdot (a_A \cdot p \cdot t \cdot \text{SSH} + a_B \cdot 0 \cdot p \cdot t)^* \cdot \text{out} \]
\[\equiv \{ \text{BA-CONTRA} \} \]
\[\text{in} \cdot \text{SSH} \cdot (a_A \cdot p \cdot t \cdot \text{SSH} + a_B \cdot \neg \text{SSH} \cdot \text{SSH} \cdot p \cdot t)^* \cdot \text{out} \]
\[\equiv \{ \text{KAT-COMMUTE} \} \]
\[\text{in} \cdot \text{SSH} \cdot (a_A \cdot p \cdot t \cdot \text{SSH} + a_B \cdot \neg \text{SSH} \cdot p \cdot t \cdot \text{SSH})^* \cdot \text{out} \]
\[\equiv \{ \text{KA-SEQ-DIST-R} \} \]
\[\text{in} \cdot \text{SSH} \cdot ((a_A \cdot p + a_B \cdot \neg \text{SSH} \cdot p) \cdot t \cdot \text{SSH})^* \cdot \text{out} \]
\[\equiv \{ \text{KAT-INVARINT, definition } p_B \} \]
\[\text{in} \cdot \text{SSH} \cdot (p_B \cdot t)^* \cdot \text{out} \]
Metatheory

**Soundness:** If $\vdash p \sim q$, then $\llbracket p \rrbracket = \llbracket q \rrbracket$

**Completeness:** If $\llbracket p \rrbracket = \llbracket q \rrbracket$, then $\vdash p \sim q$
Soundness: If $\vdash p \sim q$, then $\llbracket p \rrbracket = \llbracket q \rrbracket$

Completeness: If $\llbracket p \rrbracket = \llbracket q \rrbracket$, then $\vdash p \sim q$

Established previously for KAT [Kozen & Smith ’96]…

… but NetKAT’s packet histories add extra structure
Metatheory

**Soundness:** If $\vdash p \sim q$, then $\llbracket p \rrbracket = \llbracket q \rrbracket$

**Completeness:** If $\llbracket p \rrbracket = \llbracket q \rrbracket$, then $\vdash p \sim q$

Established previously for KAT [Kozen & Smith ’96]…
… but NetKAT’s packet histories add extra structure

**Idea:** develop an alternate semantics based on a language model, and leverage completeness of Kleene Algebra over regular sets [Kozen ’94]

**Proof outline:**
- Reduced NetKAT
- Regular interpretation
- Normal form
Completeness Proof

$p$ and $q$ such that $\lceil p \rceil = \lceil q \rceil$
Completeness Proof

\[ \tilde{p} \text{ and } \tilde{q} \text{ such that } \llbracket p \rrbracket = \llbracket q \rrbracket \]

\[ \vdash p \equiv \tilde{p} \text{ and } \vdash q \equiv \tilde{q} \]

\[ \llbracket \tilde{p} \rrbracket = \llbracket \tilde{q} \rrbracket \]

\[ G(\tilde{p}) = G(\tilde{q}) \]

\[ R(\tilde{p}) = R(\tilde{q}) \]

\[ \vdash \tilde{p} \equiv \tilde{q} \]

\[ \vdash p \equiv q \]

Reduce and Normalize

Soundness

Language Model

Normal Forms

Kleene Algebra Completeness

Transitivity

Reduce and Normalize
NetKAT Automata

Can construct an automaton from a NetKAT program by generalizing the Brzozowski derivative
NetKAT Automata

Can construct an automaton from a NetKAT program by generalizing the Brzozowski derivative

### Continuation Map:

\[
\begin{align*}
D_{\alpha\beta}(f = n) &= 0 \\
D_{\alpha\beta}(\text{dup}) &= \alpha \cdot [\alpha = \beta] \\
D_{\alpha\beta}(f := n) &= 0 \\
D_{\alpha\beta}(p + q) &= D_{\alpha\beta}(p) + D_{\alpha\beta}(q) \\
D_{\alpha\beta}(p \cdot q) &= D_{\alpha\beta}(p) \cdot q + \sum_{\gamma} E_{\alpha\gamma}(p) \cdot D_{\gamma\beta}(q) \\
D_{\alpha\beta}(p^*) &= D_{\alpha\beta}(p) \cdot p^* + \sum_{\gamma} E_{\alpha\gamma}(p) \cdot D_{\gamma\beta}(p^*)
\end{align*}
\]

### Observation Map:

\[
\begin{align*}
E_{\alpha\beta}(f = n) &= [\alpha = \beta \leq f = n] \\
E_{\alpha\beta}(\text{dup}) &= \alpha \cdot [\alpha = \beta] \\
E_{\alpha\beta}(f := n) &= [f := n = p_{\beta}] \\
E_{\alpha\beta}(p + q) &= E_{\alpha\beta}(p) + E_{\alpha\beta}(q) \\
E_{\alpha\beta}(p \cdot q) &= \sum_{\gamma} E_{\alpha\gamma}(p) \cdot E_{\gamma\beta}(q) \\
E_{\alpha\beta}(p^*) &= [\alpha = \beta] + \sum_{\gamma} E_{\alpha\gamma}(p) \cdot E_{\gamma\beta}(p^*)
\end{align*}
\]
Can construct an automaton from a NetKAT program by generalizing the Brzozowski derivative.

**Continuation Map:**

\[
\begin{align*}
D_{\alpha\beta}(f = n) &= 0 \\
D_{\alpha\beta}(\text{dup}) &= \alpha \cdot [\alpha = \beta] \\
D_{\alpha\beta}(f := n) &= 0 \\
D_{\alpha\beta}(p + q) &= D_{\alpha\beta}(p) + D_{\alpha\beta}(q) \\
D_{\alpha\beta}(p \cdot q) &= D_{\alpha\beta}(p) \cdot q + \Sigma_{\gamma} E_{\alpha\gamma}(p) \cdot D_{\gamma\beta}(q) \\
D_{\alpha\beta}(p^*) &= D_{\alpha\beta}(p) \cdot p^* + \Sigma_{\gamma} E_{\alpha\gamma}(p) \cdot D_{\gamma\beta}(p^*)
\end{align*}
\]

**Observation Map:**

\[
\begin{align*}
E_{\alpha\beta}(f = n) &= [\alpha = \beta \leq f = n] \\
E_{\alpha\beta}(\text{dup}) &= \alpha \cdot [\alpha = \beta] \\
E_{\alpha\beta}(f := n) &= [f := n = p_{\beta}] \\
E_{\alpha\beta}(p + q) &= E_{\alpha\beta}(p) + E_{\alpha\beta}(q) \\
E_{\alpha\beta}(p \cdot q) &= \Sigma_{\gamma} E_{\alpha\gamma}(p) \cdot E_{\gamma\beta}(q) \\
E_{\alpha\beta}(p^*) &= [\alpha = \beta] + \Sigma_{\gamma} E_{\alpha\gamma}(p) \cdot E_{\gamma\beta}(p^*)
\end{align*}
\]

Intuitively, these automata recognize the (guarded) strings denoted in NetKAT's language model.
NetKAT Automata

Can construct an automaton from a NetKAT program by generalizing the Brzozowski derivative

**Continuation Map:**

\[ D_{\alpha \beta}(f = n) = 0 \]
\[ D_{\alpha \beta}(\text{dup}) = \alpha \cdot [\alpha = \beta] \]
\[ D_{\alpha \beta}(f:=n) = 0 \]
\[ D_{\alpha \beta}(p + q) = D_{\alpha \beta}(p) + D_{\alpha \beta}(q) \]
\[ D_{\alpha \beta}(p \cdot q) = D_{\alpha \beta}(p) \cdot q + \Sigma_{\gamma} E_{\alpha \gamma}(p) \cdot D_{\gamma \beta}(q) \]
\[ D_{\alpha \beta}(p^*) = D_{\alpha \beta}(p) \cdot p^* + \Sigma_{\gamma} E_{\alpha \gamma}(p) \cdot D_{\gamma \beta}(p^*) \]

**Observation Map:**

\[ E_{\alpha \beta}(f = n) = [\alpha = \beta \leq f = n] \]
\[ E_{\alpha \beta}(\text{dup}) = \alpha \cdot [\alpha = \beta] \]
\[ E_{\alpha \beta}(f:=n) = [f := n = p_{\beta}] \]
\[ E_{\alpha \beta}(p + q) = E_{\alpha \beta}(p) + E_{\alpha \beta}(q) \]
\[ E_{\alpha \beta}(p \cdot q) = \Sigma_{\gamma} E_{\alpha \gamma}(p) \cdot E_{\gamma \beta}(q) \]
\[ E_{\alpha \beta}(p^*) = [\alpha = \beta] + \Sigma_{\gamma} E_{\alpha \gamma}(p) \cdot E_{\gamma \beta}(p^*) \]

Intuitively, these automata recognize the (guarded) strings denoted in NetKAT’s language model

Automata can be represented compactly using sparse matrices, yielding an efficient decision procedure based on bisimulation
Experiments

Networks:
- Topology Zoo
- FatTree
- Stanford Backbone

Programs:
- Shortest Paths
- Stanford Policy

Queries:
- Reachability
- All-Pairs Connectivity
- Loop Freedom
- Translation Validation
Results

Topology Zoo

Connectivity

Loop Freedom

Translation Validation

FatTree

Scalability

Relative Performance

Stanford Backbone

Basic reachability in 0.67s (vs 13s for HSA)
Run-Time Model
Run-Time Model

Application

Configurations

Run-Time System

Code that manages the rules installed on switches

Translate configuration updates into sequences of OpenFlow instructions
Example: Access Control

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>🧢</td>
<td>Web</td>
<td>Allow</td>
</tr>
<tr>
<td>🧢</td>
<td>Non-web</td>
<td>Drop</td>
</tr>
<tr>
<td>🧢</td>
<td>Any</td>
<td>Allow</td>
</tr>
</tbody>
</table>

Traffic
Example: Access Control

Security Policy

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>🧢</td>
<td>Web</td>
<td>Allow</td>
</tr>
<tr>
<td>🧢</td>
<td>Non-web</td>
<td>Drop</td>
</tr>
<tr>
<td>🧢</td>
<td>Any</td>
<td>Allow</td>
</tr>
</tbody>
</table>

Configuration A

- Process black-hat traffic on F1
- Process white-hat traffic on \{F2,F3\}
Example: Access Control

Security Policy

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>🧙‍♂️</td>
<td>Web</td>
<td>Allow</td>
</tr>
<tr>
<td>🧙‍♂️</td>
<td>Non-web</td>
<td>Drop</td>
</tr>
<tr>
<td>🧙‍♂️</td>
<td>Any</td>
<td>Allow</td>
</tr>
</tbody>
</table>

Configuration A

- Process black-hat traffic on F1
- Process white-hat traffic on {F2,F3}
**Example: Access Control**

**Security Policy**

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>🧛‍♂️</td>
<td>Web</td>
<td>Allow</td>
</tr>
<tr>
<td>🧛‍♂️</td>
<td>Non-web</td>
<td>Drop</td>
</tr>
<tr>
<td>🧛‍♂️</td>
<td>Any</td>
<td>Allow</td>
</tr>
</tbody>
</table>

**Configuration A**

- Process black-hat traffic on F1
- Process white-hat traffic on {F2,F3}

**Configuration B**

- Process black-hat traffic on {F1,F2}
- Process white-hat traffic on F3
Network Updates

Challenges
• Network is a distributed system
• Can only update one rule at a time

Our Approach
• Provide programmers with constructs for updating the entire network at once
• Engineer semantics to guarantee “reasonable” behavior during transition
• Run-time system:
  - Constructs update protocols
  - Applies optimizations automatically
Atomic Updates

- Seem sensible...
- but costly to implement...
- and difficult to reason about, due to behavior on in-flight packets
Update Semantics

**Atomic Updates**
- Seem sensible...
- but costly to implement...
- and difficult to reason about, due to behavior on in-flight packets

**Per-Packet Consistent Updates**
Every packet processed with old or new configuration, but not a mixture of the two

**Per-Flow Consistent Updates**
Every set of related packets processed with old or new configuration, but not a mixture of the two
Update Semantics

**Atomic Updates**
- Seem sensible...
- but costly to implement...
- and difficult to reason about, due to behavior on inflight packets

**Per-Packet Consistent Updates**
Every packet processed with old or new configuration, but not a mixture of the two

**Per-Flow Consistent Updates**
Every set of related packets processed with old or new configuration, but not a mixture of the two

---

**Theorem (Universal Property Preservation)**
An update is per-packet consistent if and only if it preserves all safety properties.
Update Implementations

**Two-phase update**
- Build versioned internal and edge switch configurations
- Phase 1: Install internal configuration
- Phase 2: Install edge configuration

**Pure Extension**
- Update strictly adds paths

**Pure Retraction**
- Update strictly removes paths

**Sub-space updates**
- Update modifies a small number of paths

**New: software synthesis**
- Use an model checker to generate correct sequences of updates automatically
Update Synthesis

**Inputs**
- Old configuration
- New configuration
- Logical invariant (e.g., LTL formula)

**Assumptions**
- Configurations loop free
- Pause between each switch

**Algorithm**
- Search through the space of updates
- Consider paths that do not violate property
- Halt when new configuration reached

**Optimizations**
- Learn from counterexamples
- Incremental model checker
- Search heuristics
Update Synthesis

**Inputs**
- Old configuration
- New configuration
- Logical invariant (e.g., LTL formula)

**Assumptions**
- Configurations loop free
- Pause between each switch

**Algorithm**
- Search through the space of updates
- Consider paths that do not violate property
- Halt when new configuration reached

**Optimizations**
- Learn from counterexamples
- Incremental model checker
- Search heuristics

**Performance**
Wrapping Up
Ongoing Work

Proof-Carrying Code

- Tenants generate proofs of important network properties
- Administrators verify proofs from untrusted tenants using simple checkers

Probabilistic Programming

- Encode randomized routing schemes
- Establish fault/congestion bounds

Deeper Foundations

- Understand SDN in terms of classic concepts like {safe, regular, atomic} registers
- Develop analogs of non-blocking algorithms
Conclusion

- Programming languages and formal methods have a key role to play in software-defined networks.

- By carefully engineering the right programming abstractions, effective reasoning about network behavior becomes possible.
  - Featherweight OpenFlow
  - NetKAT
  - Consistent updates
Thank you!

**Collaborators**
- Carolyn Anderson (Swarthmore)
- Pavol Cerny (CU Boulder)
- Jean-Baptiste Jeannin (CMU)
- Dexter Kozen (Cornell)
- Jed McClurgh (CU Boulder)
- Matthew Milano (Cornell)
- Jennifer Rexford (Princeton)
- Mark Reitblatt (Cornell)
- Cole Schlesinger (Princeton)
- Alexandra Silva (Nijmegen)
- Laure Thompson (Cornell)
- Dave Walker (Princeton)

**Papers, Code, etc.**

http://frenetic-lang.org/