Dr. Nick Feamster
Professor

Software Defined Networking

In this course, you will learn about software defined networking and how it is changing the way communications networks are managed, maintained, and secured.

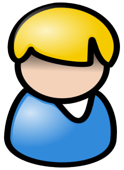
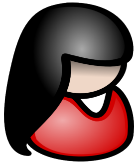
Kinetic: New Capability for SDN Control

- How to implement a network control program
  - How to provide ***dynamic control*** that handles arbitrary network events
  - E.g, Intrusion detection, traffic load shift, etc
- **Verification** and guarantees of program's correctness
- Huge missed opportunities in software

Different Types of Network Events

- Network traffic
  - Traffic load increase/decrease, security incidents
- User-specific
  - User authentication, excessive data usage
- Data-plane events
  - Topology change, switch/link failures
- …

Different Reactions to an Event

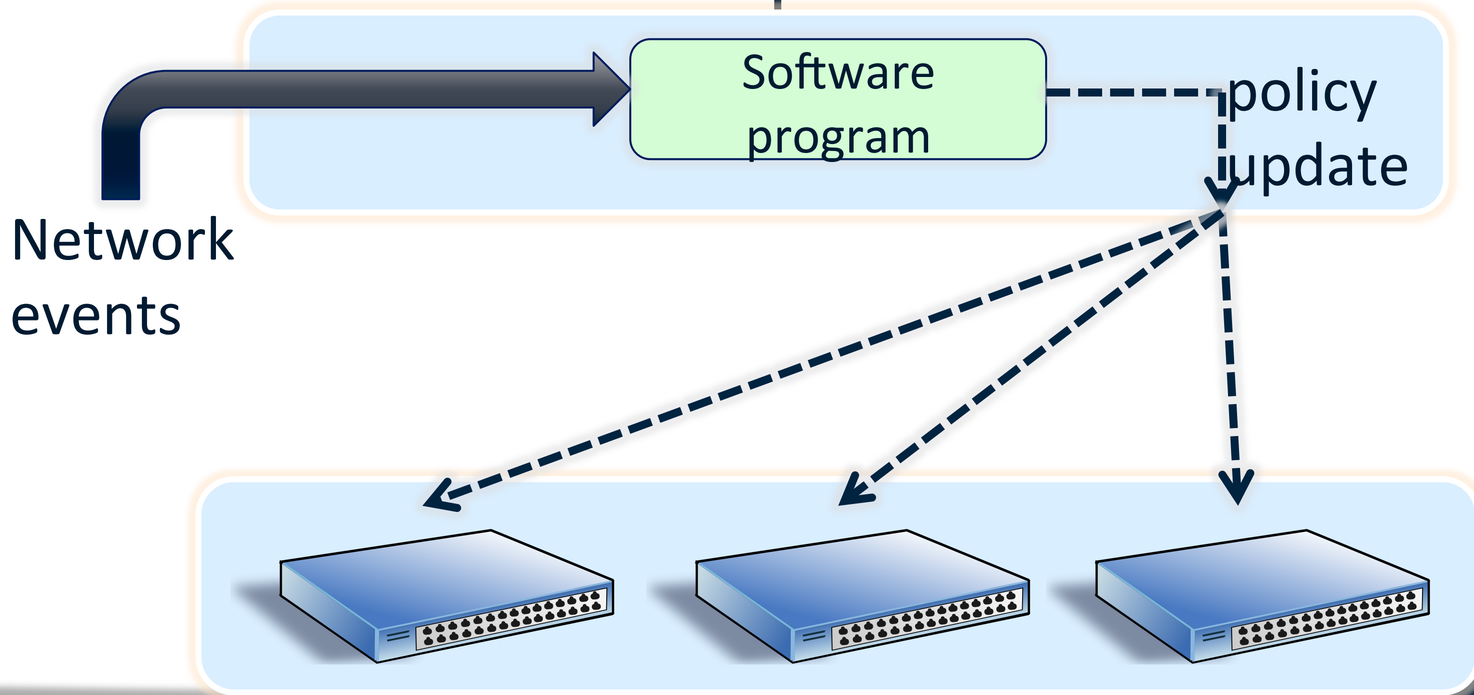
Event	Operators	Reaction
		"Only block that infected host"
Host is infected!		"Block all communications in the network!"
		"Direct communication to our internal honeypot"

Main Insight

Network events and **dynamic reactions** to them should be **programmatically encoded** in the **network control program** by operators

Dynamic Network Control Program

- **Software program** that embeds event – reaction relationships



Unanswered Questions

How to **embed event-reaction logic** in software?

How to **verify** that the program will make **changes correctly**?

Kinetic tackles these questions

Kinetic

- **Domain specific** language and control platform

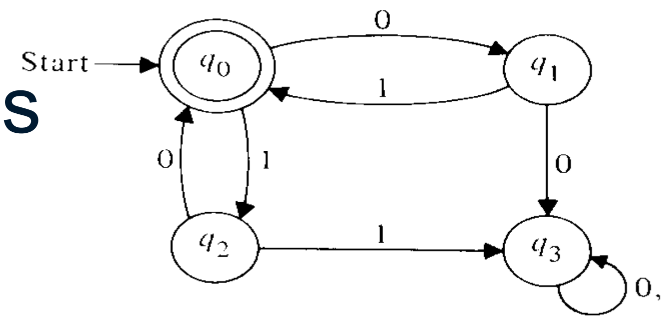
- Helps create SDN control programs that **embed custom event-reaction** relationships

- **Verifies** program's **correctness**

Kinetic Approach

- Domain specific language
  - Constrained, but structured

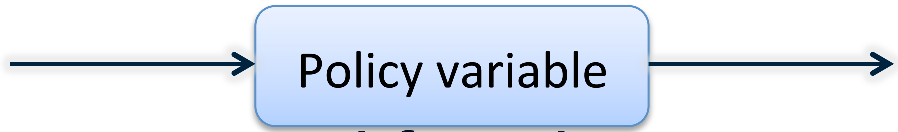
- Express changing behavior as a *finite state machine*



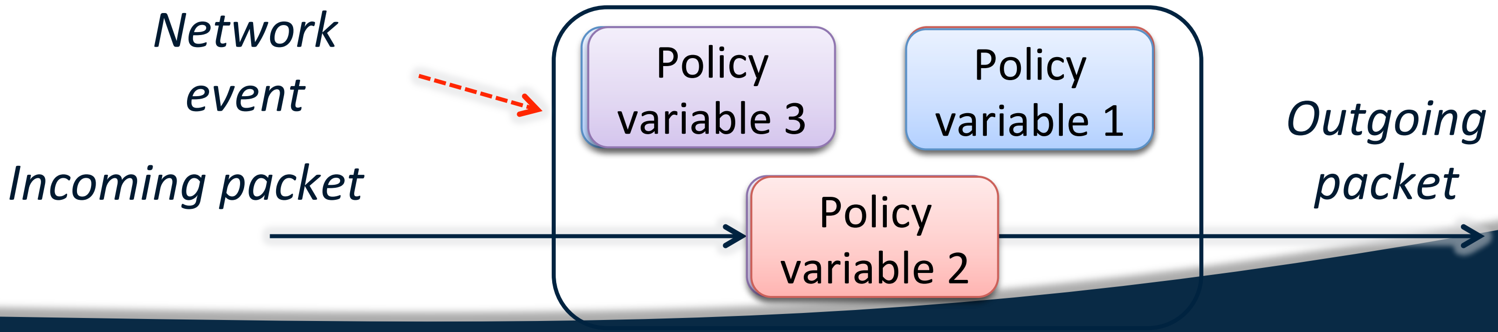
- Verify program's correctness with a model checker (NuSMV)

Kinetic's Domain Specific Language

- Borrows some abstractions from Pyretic
  - Encodes forwarding behavior in a *policy variable*

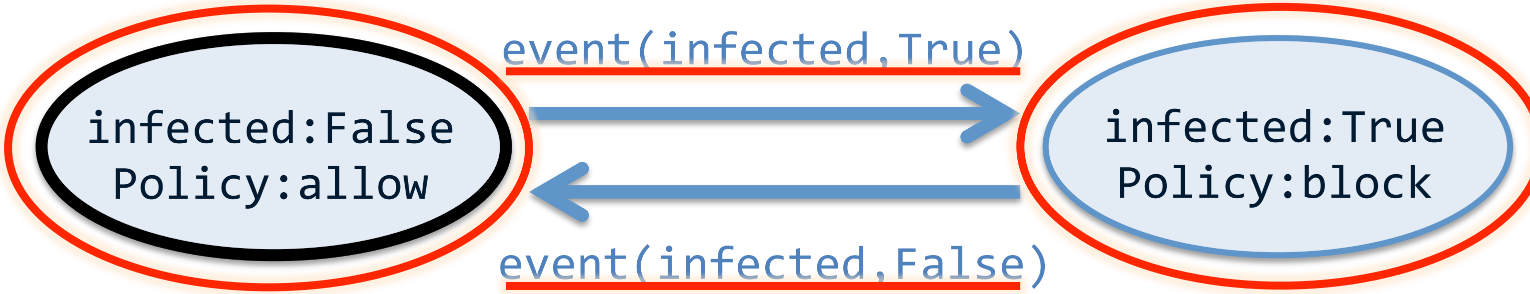


- New constructs and functions to express policies that respond to *changing conditions*



IDS Example in Kinetic

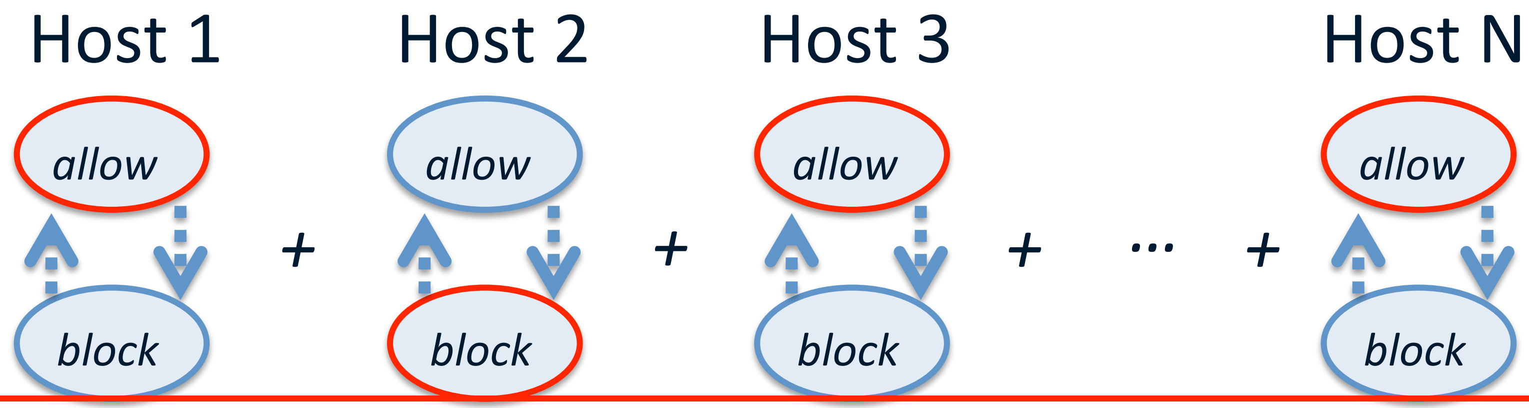
- **Event**: *infected*
- **State**: policy variable’s value
  - *allow* or *block* packet



independent FSM

Decomposing to multiple FSMs

- FSM instance is instantiated per flow



# of hosts: N *Total # of states: 2N* *Total # of transitions: 2N*

State representation is ***Linear*** in N
(instead of geometric)

LPEC: Abstraction to Define a flow

- In IDS example, flow is defined by source IP address (host)
- Other policies may require more flexibility (e.g., need to group packets by location)
- **Located Packet Equivalence Class (LPEC)**
  - Programmer abstraction to define *flow*

```
def lpec(pkt):
        return match(dstip=pkt['dstip'])
```

Kinetic Verification Process

- Kinetic verifies correctness of the program
  - User-specified temporal properties
  - Verifies **current and future** forwarding behavior based on network events
- Verification process is **automated**
  - Constrained but structured language allows automatic parsing and translation of program
- Verification runs **before program's deployment**

Verification Process

Kinetic program

```
@transition
def infected(self):
  self.case(occured(self.event),self.event)

@transition
def policy(self):
  self.case(is_true(V('infected')),C(drop))
  self.default(C(identity))

self.fsm_def = FSMDef(
  infected=FSMVar(type=BoolType(),
                  init=False,
                  trans=infected),
  policy=FSMVar(type=Type(Policy,{drop,
      identity}),
                init=identity,
                trans=policy))
```

NuSMV FSM model

```
MODULE main
  VAR
    policy    : {identity,drop};
    infected  : boolean;
  ASSIGN
    init(policy)    := identity;
    init(infected)  := FALSE;
  next(policy) :=
    case
      infected  : drop;
      TRUE      : identity;
    esac;
  next(infected) :=
    case
      TRUE     : {FALSE,TRUE};
  esac;
```

User-specified temporal properties

True or False

(w/ counter-example)

Examples of Temporal Properties

- If a host is infected, drop packets from that host

$$AG\ (\text{infected} \rightarrow AX\ \text{policy}=\text{drop})$$

For all possible transitions from current state,

For all current and future states,

For all possible transitions from current state,

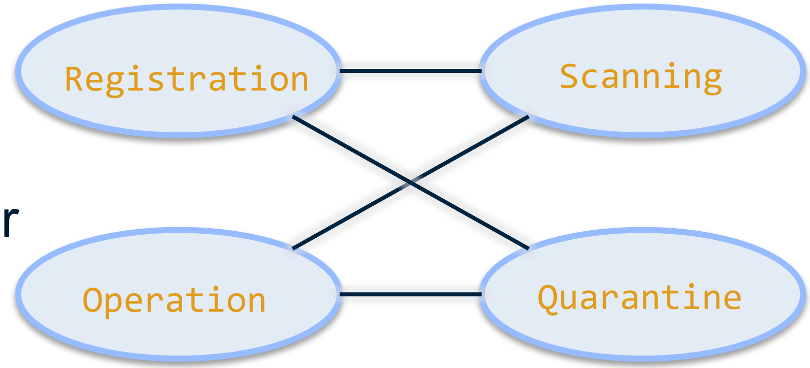
For the next state,

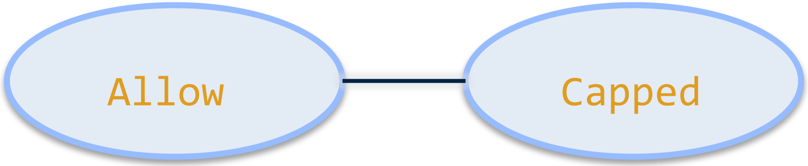
- If host is authenticated either by Web or 802.1X, and is not infected, packets should never be dropped.

$$AG\ (\ (\text{authenticated\_web} \mid \text{authenticated\_1x})\ \&\ !\text{infected} \rightarrow AX\ \text{policy}!=\text{drop}\ )$$

Kinetic: Real Deployments

- Campus network
  - Functional access control system
  - Deployed SDN-enabled switches over 3 buildings
- Home network
  - Usage-based access control
  - Deployed 21 SDN-enabled wireless routers over 3 continents
  - Jul., 2012 – Feb., 2014





Kinetic Summary

- **Domain specific** language and control platform
  - Program **encodes event-reaction logic**
- **Extensive user study** shows that
  - Much **easier to express dynamics** in the network
  - Helps to **reduce lines of code**
- **Scales well** to large networks and lots of events
- **Verification process reduces bugs** in programs