Software Defined Networking

Dr. Nick Feamster
Professor

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

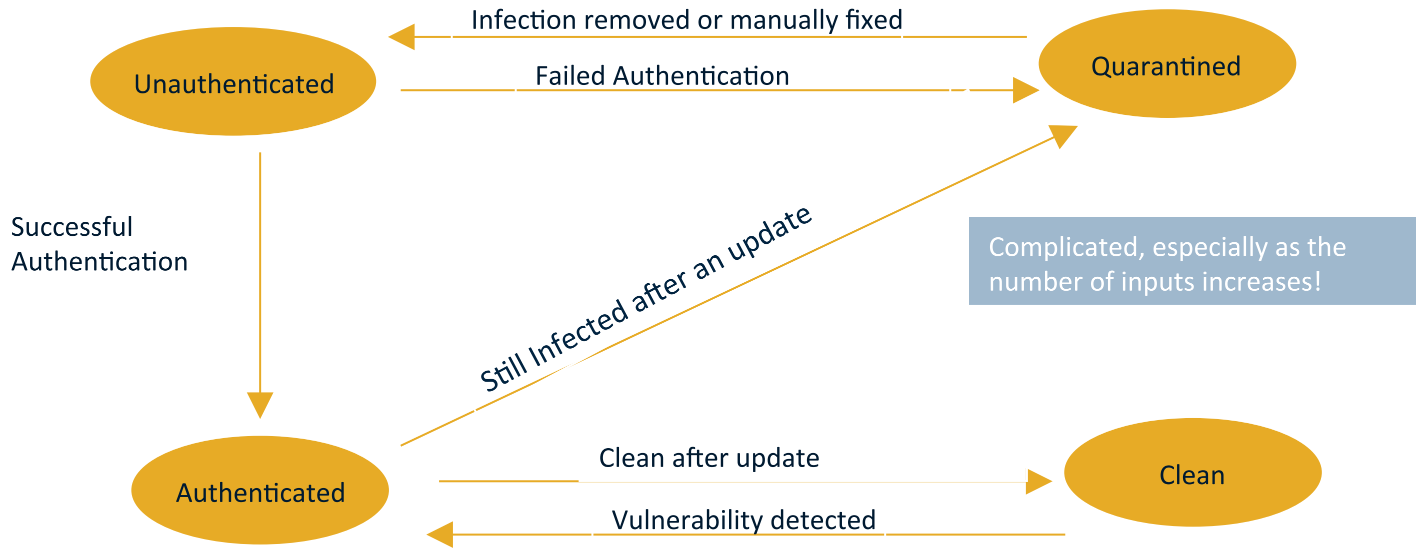
This Module: Verification

- **Motivation:** How do you know the network is doing the right thing?
- Verification techniques
  - Configuration Verification: rcc (pre-SDN)
  - **Control Plane Verification: Kinetic**
  - Data Plane Verification
    - Header Space Analysis
    - Veriflow

Configuration Changes are Common

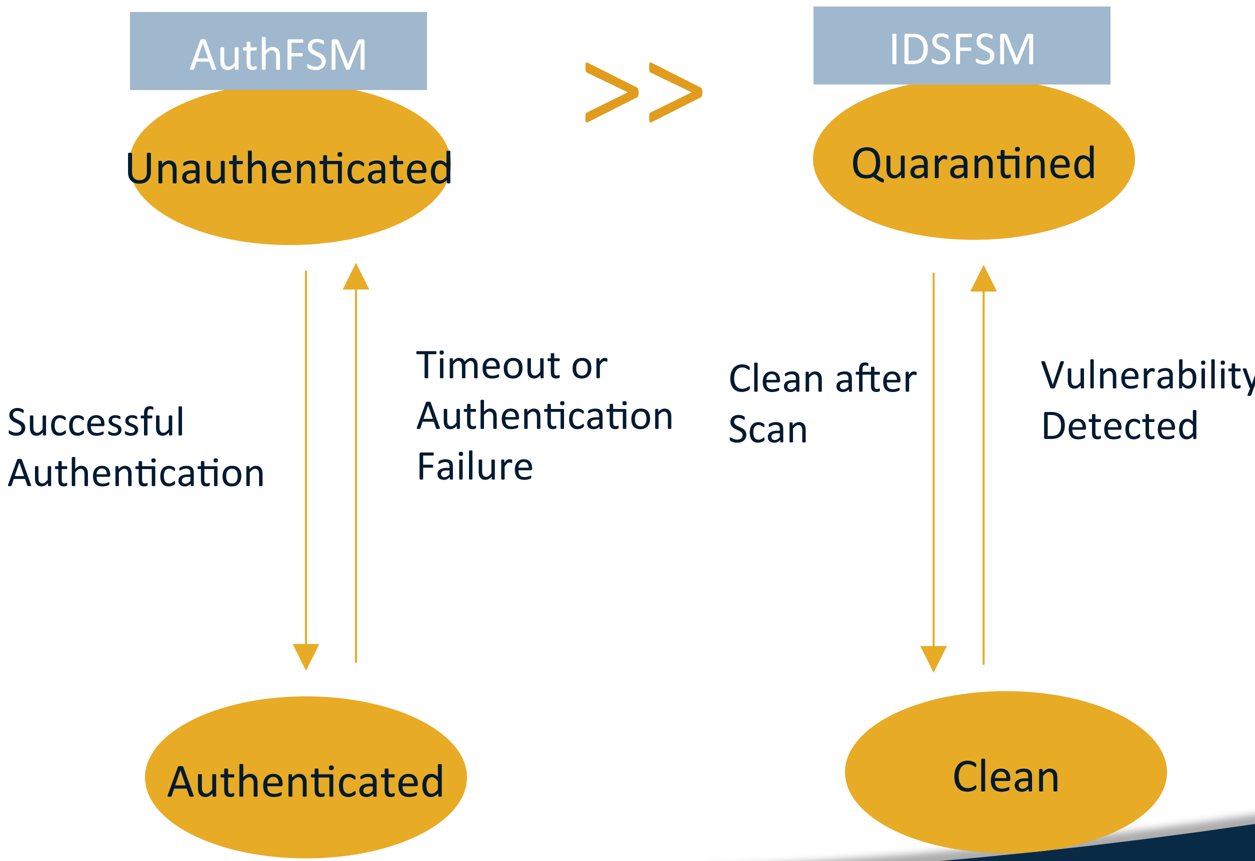
- Anywhere from 1,000-18,000 configuration changes per month
- Some of these changes can be automated, but there is no way to reason about them
- Need: Domain-specific support for expressing controller dynamics

Kinetic: Verifiable Event-Based Network Control



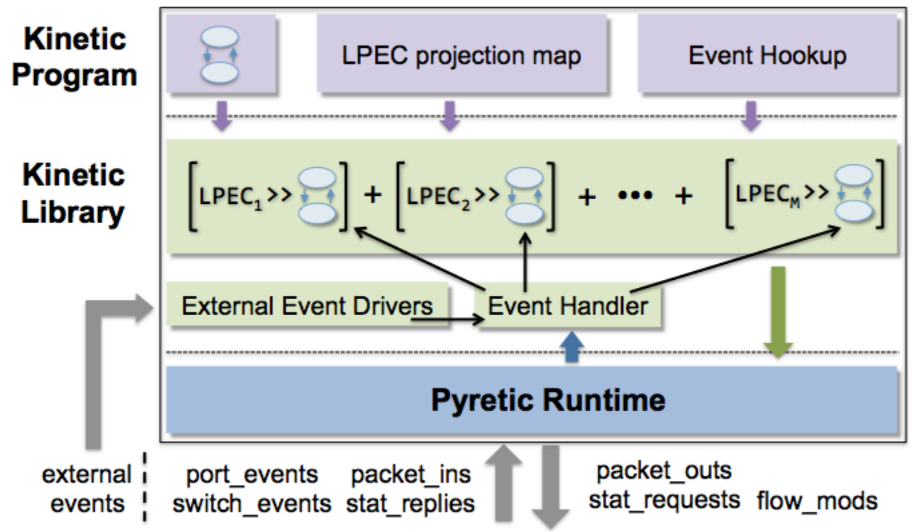
- Network policies represented as FSMs
- FSMs are verifiable!

Simpler: Sequential Composition



Simpler: Use Pyretic to sequentially compose FSMs!

Kinetic System Architecture



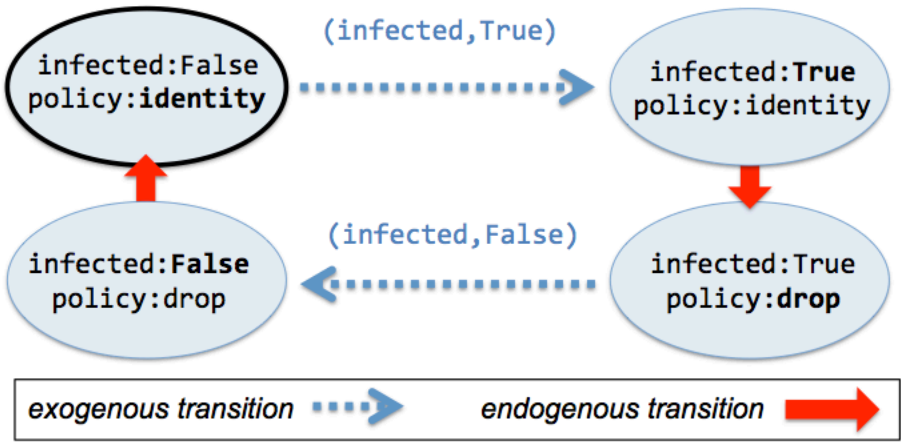
- LPEC projection map divides located packets into equivalence classes
- Event hookup for external events

Kinetic Language Architecture

Kinetic	K ::=	P | FSMPolicy(L,M) | K + K | K >> K
	L ::=	*f* : *packet* -> F
	M ::=	FSMDef([*var_name*=V])
	V ::=	VarDef(*type*,*init_val*,T)
	T ::=	[case(S,D)]
	S ::=	D==D | S & S | (S | S) | ! S
	D ::=	C(*value*) | V(*var_name*) | event
Pyretic	P ::=	Dynamic() | N | P + P | P >> P
Static Pyretic	N ::=	B | F | modify(h=v) | N + N | N >> N
	F ::=	A | F & F | (F | F) | ~F
	A ::=	identity | drop | match(h=v) |
	B ::=	FwdBucket() | CountBucket()

- Extensions to Pyretic
- Special dynamic policy class FSMPolicy
- FSM descriptions and basic values

Example: Intrusion Detection System




```
@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=PolType({drop,identity}),
               init=identity,
               trans=policy))
```

- Two types of transitions
  - **Exogenous:** Triggered by external event
  - **Endogenous:** Triggered by change in internal variable

LPEC: Define the Granularity of the FSM

Step (1)

```
match(srcip=IPAddr('10.0.0.1'))
```

Step (2)

```
def ids_lpec_pm(pkt):
    return match(srcip=IPAddr('10.0.0.1'))
```

Step (3)

```
def ids_lpec_pm(pkt):
    return match(srcip=pkt['srcip'])
```

- Specify Located Packet Equivalence Class (“LPEC”)
- Define projection MAP
- Parameterize using input packet

Conversion to NuSMV

```
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};
      TRUE    : infected;
  esac;
```

- FSMs translate directly to NuSMV model checker
- Can check properties in CTL

CTL Examples for Kinetic IDS

NuSMV	Description
`AG infected → (policy=drop)`	If infection event arrives, the system should drop the packet.
`AG !infected → (policy=identity)`	If infection is cleared, the system should allow the packet.
`AG EF policy=identity`	From any state, it is possible to go to allowed state again.
`A [ policy=identity U infected ]`	For all paths, policy allows packet until an infection occurs.

- Rules expressed using CTL
- Branching-time logic, time as a tree structure

Summary

- Event-based control is a common idiom
- Need to verify dynamic properties of network control, not only data-plane properties
- Kinetic: Verifiable dynamic network control
  - Policies expressed as FSMs
  - FSMs map naturally to model checking
  - Properties can be checked in CTL