Programming Network Policies by Examples: Platform, Abstraction and User Studies

Boon Thau Loo
University of Pennsylvania
NetPL workshop @ SIGCOMM 2017

Joint work with Yifei Yuan, Dong Lin, Siri Anil, Harsh Verma, Anirudh Chelluri, Rajeev Alur, Boon Thau Loo

Programming Abstractions in SDN

General-purpose Programming Languages

- C/C++ (NOX)
- Python (POX)
- Java (Floodlight)
- …

Domain Specific Languages (DSL)

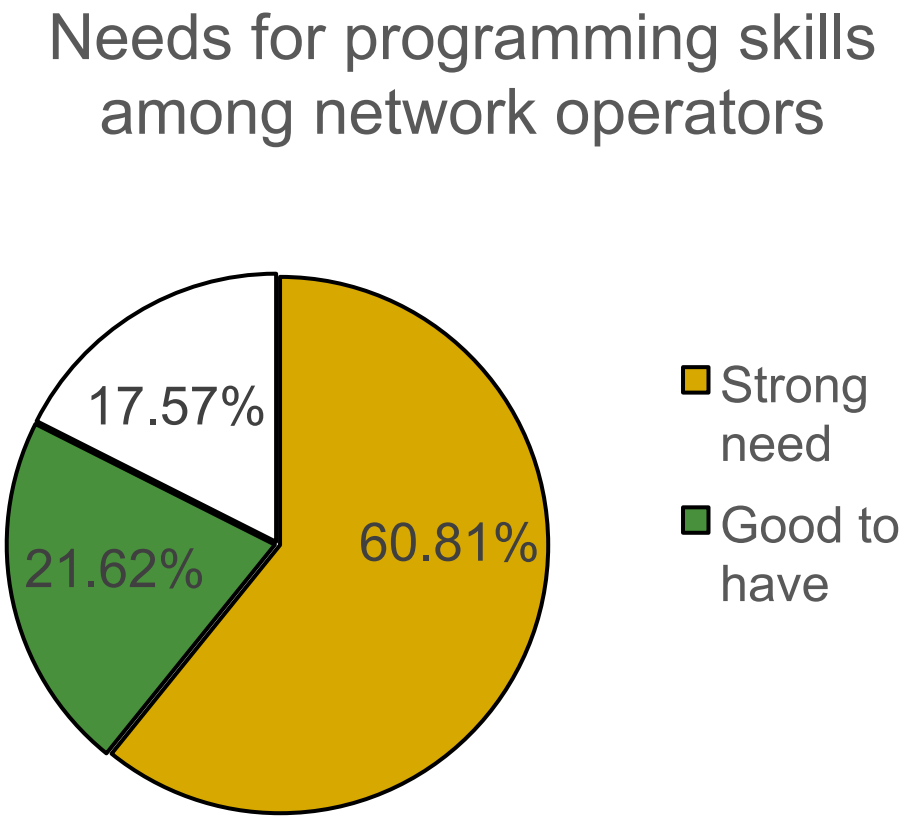
- Declarative Networking [CACM'09]
- Frenetic [ICFP'11]
- NetCore [POPL'12]
- Pyretic [NSDI'13]
- NetKAT [POPL'14]
- …

Missing: SDN programming for *non-programmers*

NSF I-Corps Program

- Interviewed 101 network operators/architects, software engineers and router vendors
  - Need for programming skills among network operators
  - Frequent network configuration changes that should be automated
  - (Lack of) programming expertise an impediment towards SDN usage.

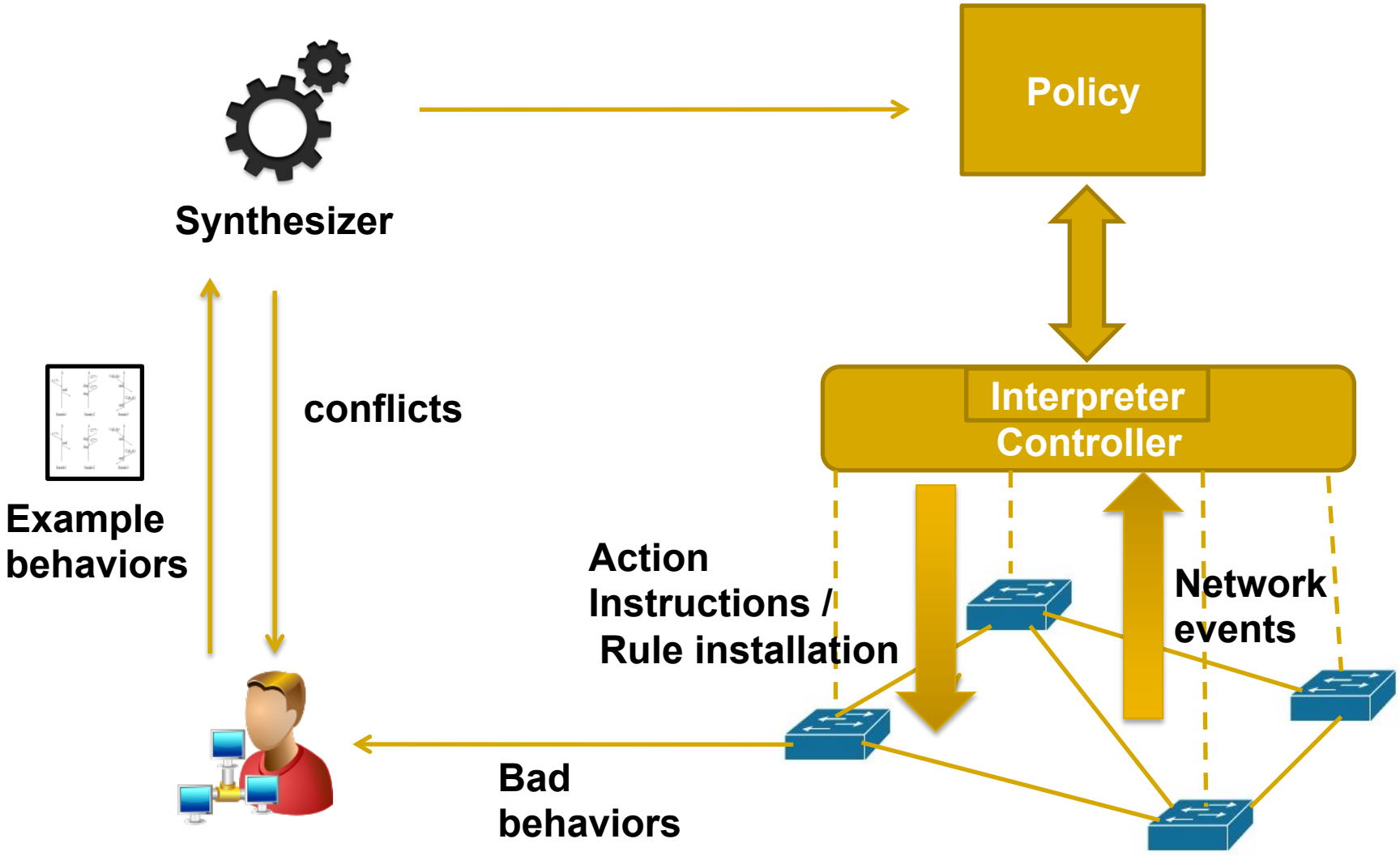
Need for Programming Capabilities



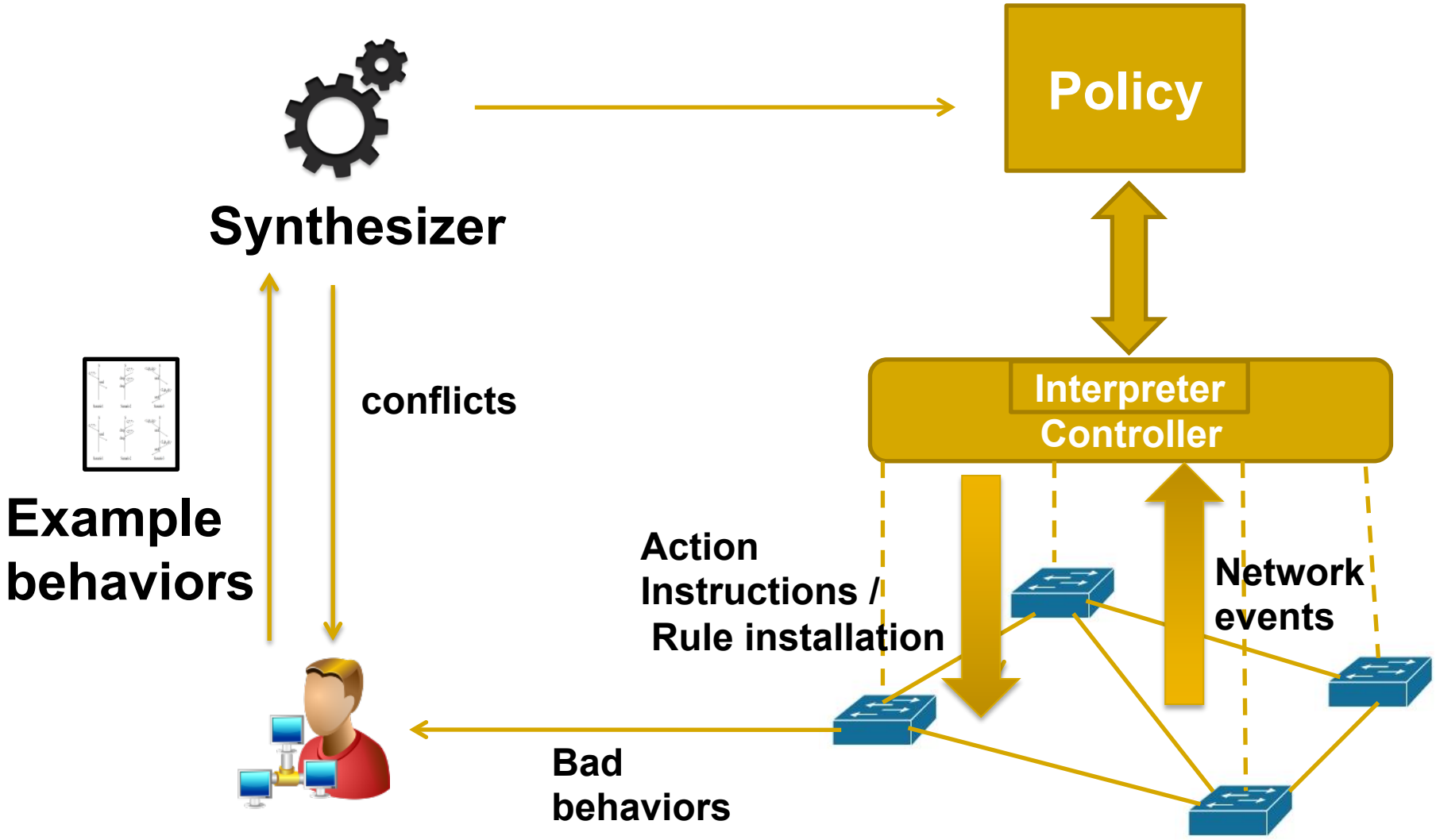
Outline

- NetEgg overview
- Controlled user study

NetEgg Overview



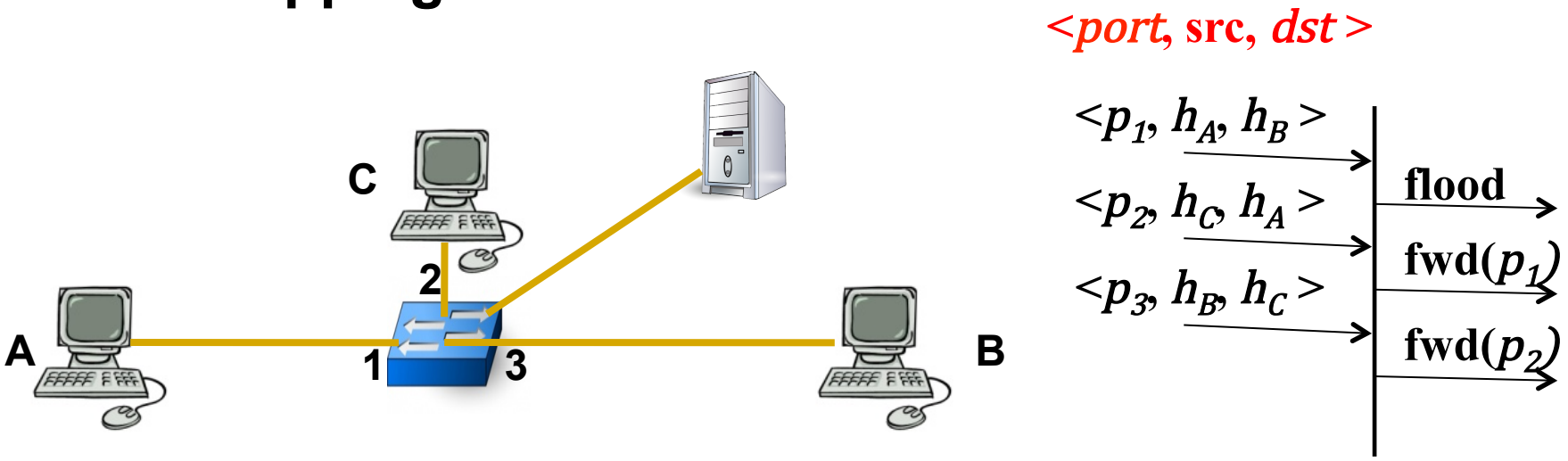
NetEgg Overview



A First Example

Learning switch:

1. **Learn the mapping between hosts and ports**
2. **Forward packets according to the learnt mapping**



A Stateful Policy Model

- Intuition: states + cases (policies)
- Learning switch:
  - States: Mapping between hosts and ports
  - Case 1. Flood packets if the destination is unknown; store the port for the source
  - Case 2. Forward packets otherwise; store port for source in packets

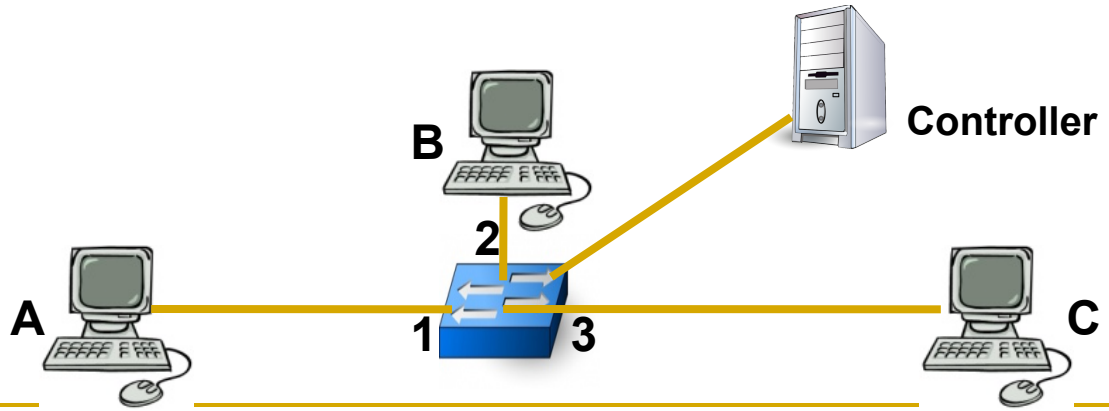
Execution Model

State Table

MAC	State	Value
A	*1*	*1*
C	*1*	*3*

Policy Table

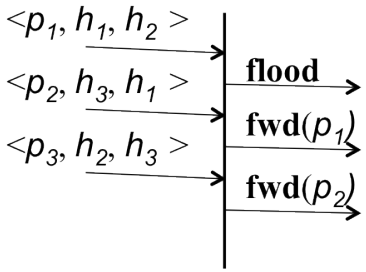
Match	Test	Action	Update
*	ST(p.dst).state=0	flood	ST(p.src):= (1,port)
*	ST(p.dst).state=1	fwd(ST(p.dst).value)	ST(p.src):= (1,port)

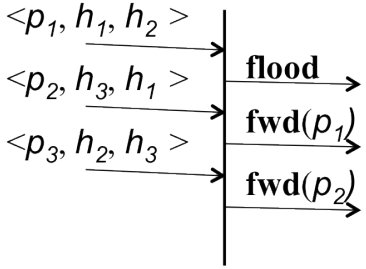


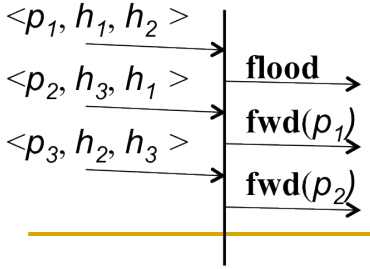
Why This Model?

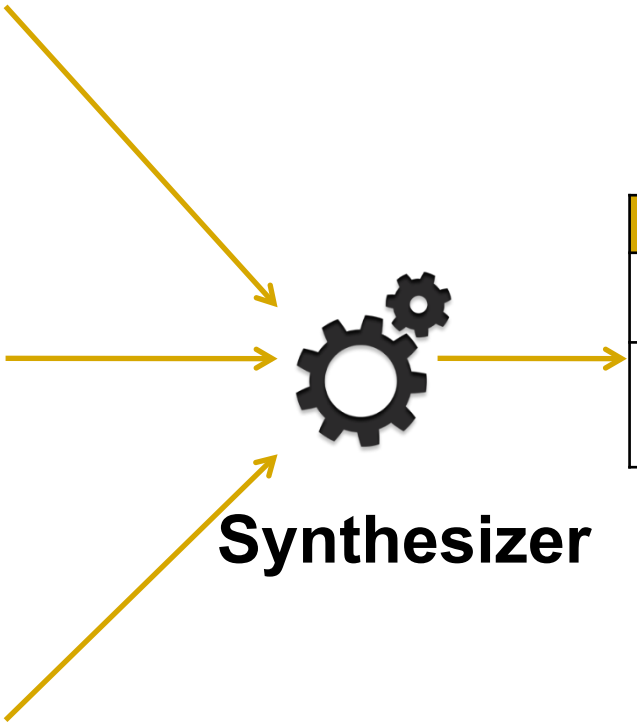
- Simple:
  - Fast execution
  - Deployment on dataplane (FAST [HotSDN'14], openState SIGCOMM CCR'14)
- Expressive:
  - Dynamic policies (Kinetic [NSDI'2015])

Synthesizer









Synthesizer

Match	Test	Action	Update
*	ST(dstmac).state=0	flood	ST(srcmac):=(1,port)
*	ST(dstmac).state=1	fwd(ST(dstmac).value)	ST(srcmac):=(1,port)

Synthesis Algorithm

- Goal: consistent, minimal

Synthesis Algorithm

- Goal: consistent, minimal
- Greedy search

Match	Test	Action	Update
*	ST(dst).state=0		

…

Match	Test	Action	Update
*	ST(dst).state=0		
*	ST(dst).state=1		

Synthesis Algorithm

- Goal: consistent, minimal
- Greedy search

Match	Test	Action	Update
*	ST(dst).state=0		

...

Match	Test	Action	Update
*	ST(dst).state=0		
*	ST(dst).state=1		

- Backtrack with pruning

Runtime Rule Installation

- Idea: Keep rules not updating state tables on the switch

MAC	State	Value
A	*1*	*2*
B	*1*	*3*

Match	Test	Action	Update
*	ST(dstmac) .state=0	flood	ST(srcmac): =(1,port)
*	ST(dstmac) .state=1	fwd(ST(dstmac) .value)	ST(srcmac): =(1,port)

<inport=3, srcmac=B, dstmac=A>

Match	Action
inport=3, srcmac=B, dstmac=A	*fwd(2)*
…	…

Evaluation

- Is scenario-based programming feasible?
  - Expressiveness
  - Intuitiveness
  - Conciseness
  - Efficiency
- Is the performance of synthesized implementations comparable to hand-crafted implementation?
  - Controller response latency
  - End-to-end performance

Breadth and Synthesis Effort/Time

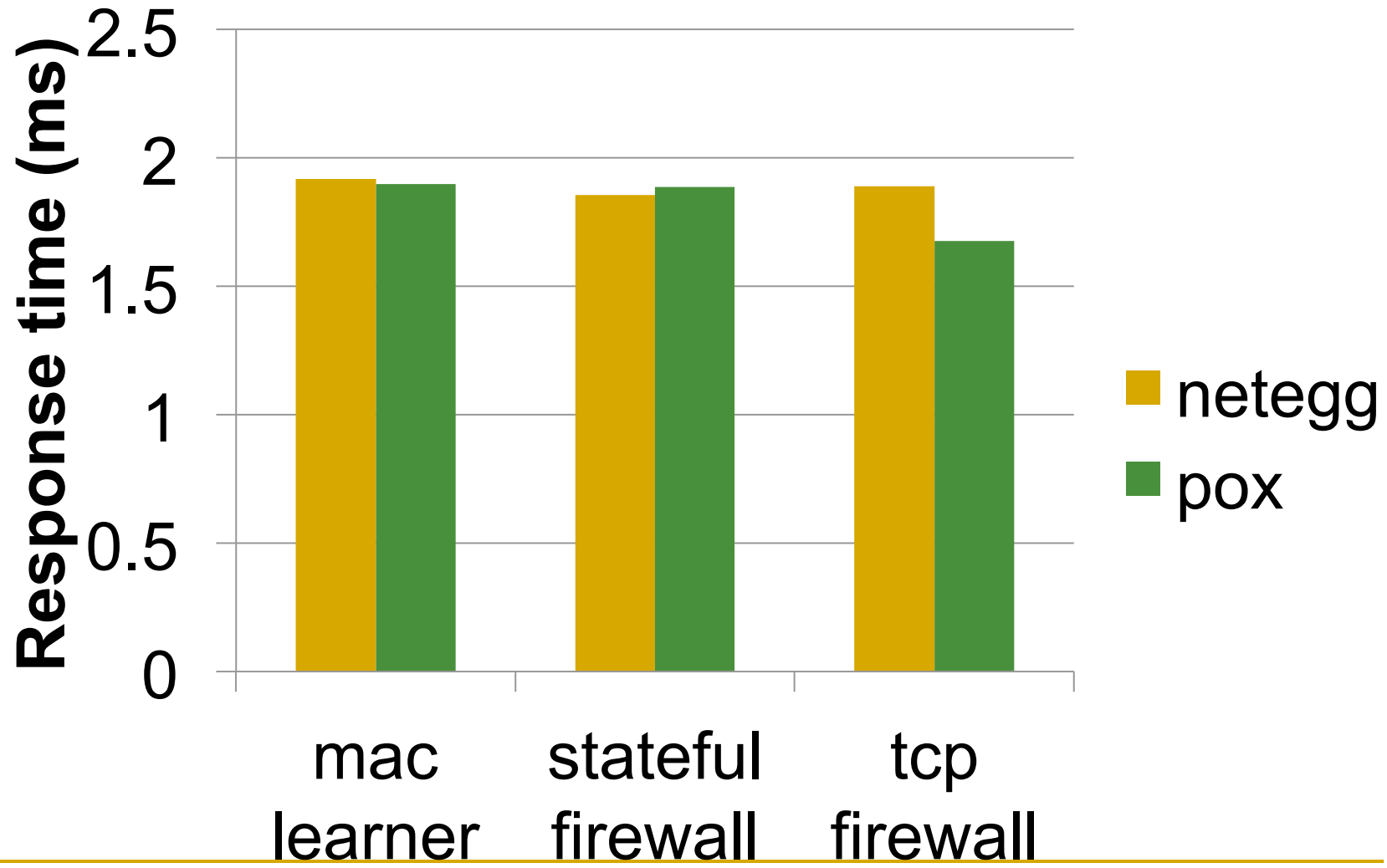
	#EV	#SC	Time
maclearner1	3	1	11 ms
maclearner2	3	1	15 ms
auth	3	2	13 ms
gardenwall	5	3	52 ms
ids	3	2	15 ms
monitor	3	2	13 ms
ratelimiter	10	5	147 ms
serverlb	7	3	143 ms
stateful firewall1	3	2	12 ms
stateful firewall2	3	2	16 ms
stateful firewall3	6	3	107 ms
trafficlb	7	3	402 ms
ucap	3	2	13 ms
vmprov	3	2	24 ms
TCP firewall	9	5	64 ms
ARP proxy	5	2	49 ms

Table 11: **Network policies generated from scenarios. #SC is the number of scenarios used to synthesize the policy, #EV is the total number of events in scenarios, Time is the running time of the synthesizer.**

Comparison in Code Size

	NetEgg	Pyretic	POX
Mac learner	3	17	29
Stateful firewall	3	21	58
TCP firewall	9	24	68

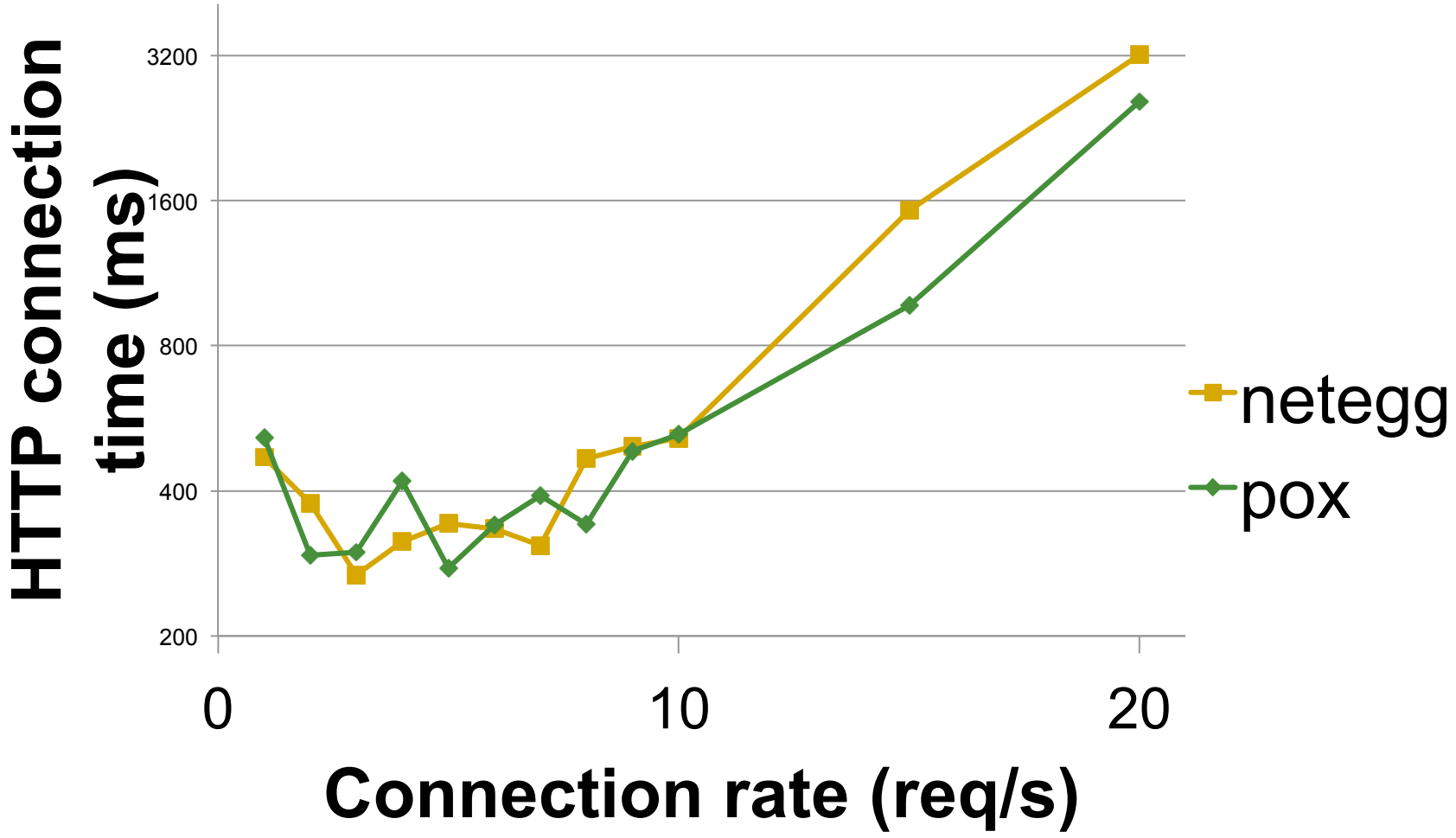
Response Time



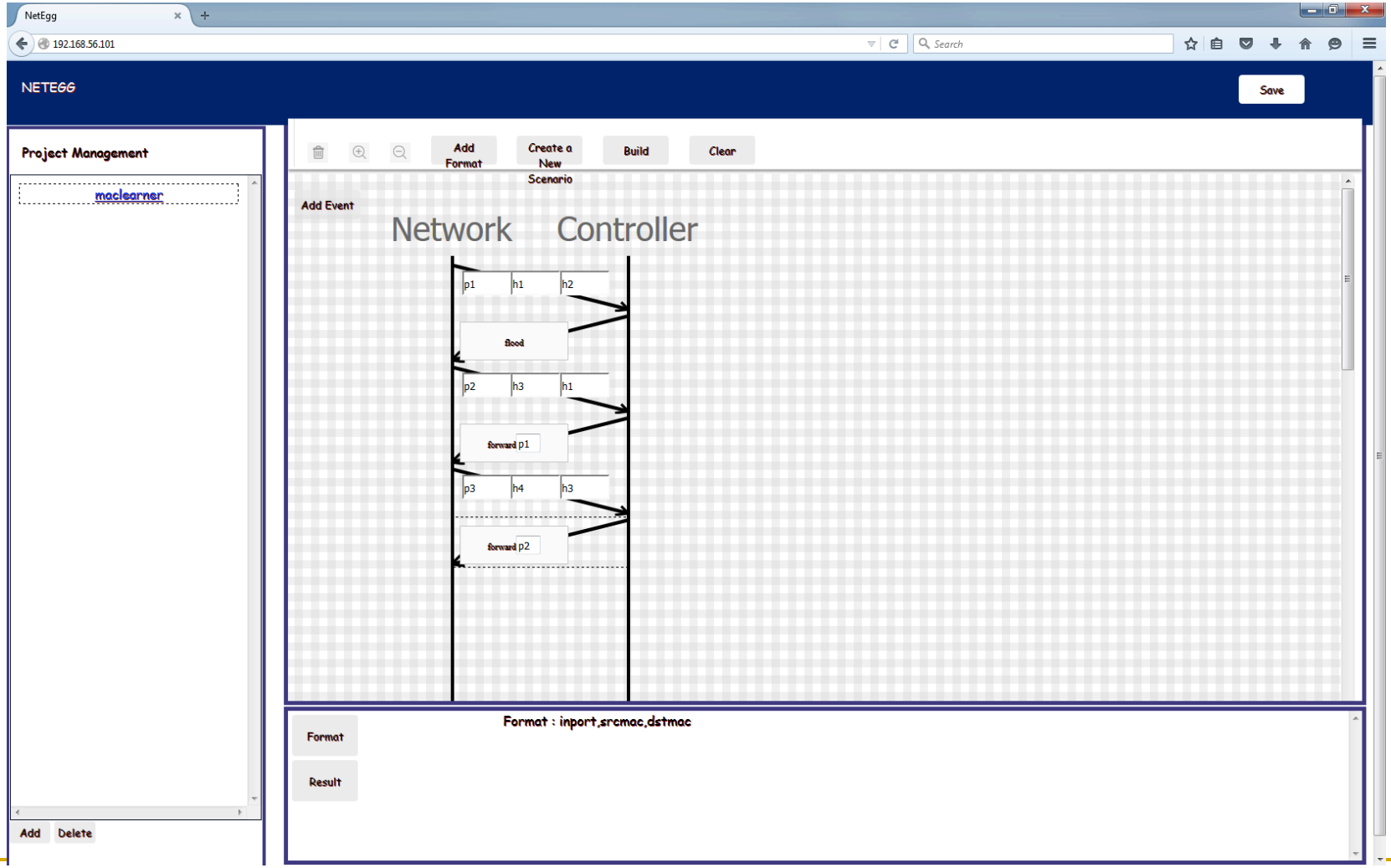
End-to-end Performance

- Topology: fattree, 20 switches, 16 hosts
- Policy: learning switch
- Setup:
  - 1 host as HTTP server
  - other hosts send HTTP requests to the server
  - benchmark connection time (i.e. time between a request is issued and it is finished.)

End-to-end Performance



NetEgg Toolkit



Outline

- NetEgg overview
- Controlled user study

Controlled User Studies (2016/17)

- Hypothesis:
  - "Programmers are more productive in programming language X compared programming language Y for solving problems type Z"
- Keep all factors as constant as possible:
  - Academic/industry backgrounds, ….
  - Solve the same problem *Z once*
  - In a controlled setting under our watch
  - Software environment is heavily instrumented
- Vary only one factor: programming language X vs Y

Controlled NetEgg user study (IRB approval was needed)

The Case for User Studies in PL Design

- Goal: identify early on the mental model of target audience and programming abstractions:
- Complementary Approaches
  - Release as open-source and measure adoption
  - Projects in large classes (>100 students) : Declarative networking (SIGCOMM'11 Education workshop), Kinetic (NSDI'15)

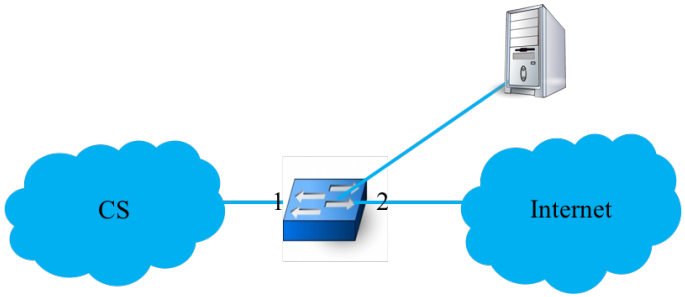
User Study

- Q1: Can NetEgg reduce the error rate of programming SDN policies?
- Q2: Can NetEgg improve user's productivity?
- Q3: How the users interact with NetEgg?

User Study Design

- Object: NetEgg VS. POX
- Users: masters/PhD students from the Engineering School of Penn
  - **With** Networking/Distributed System background
  - **No** SDN programming background
- 15 users in NetEgg, 9 users in POX
  - All users in the POX group are required to be proficient in Python programming
  - No requirement for NetEgg group
- More than 21 users in early pilot test

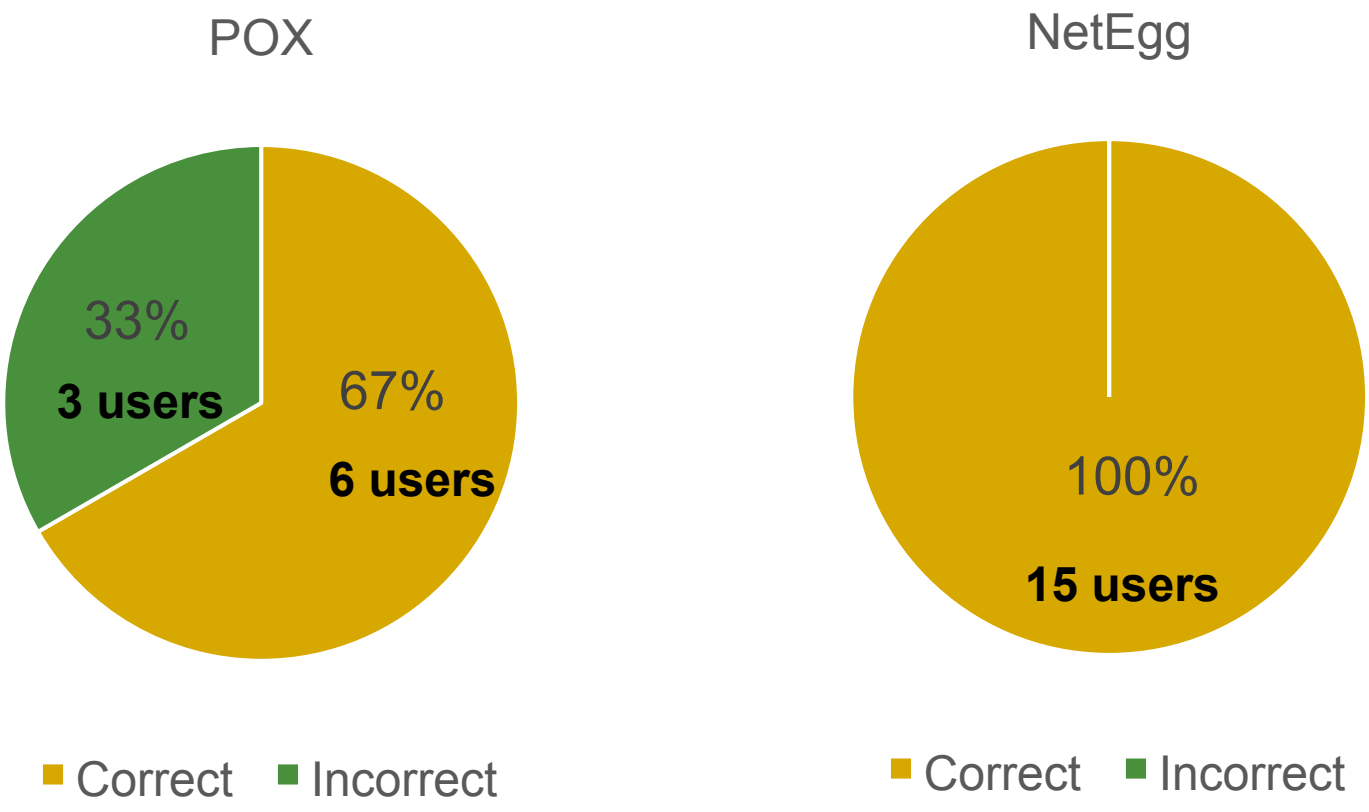
Programming Assignment: Firewall++



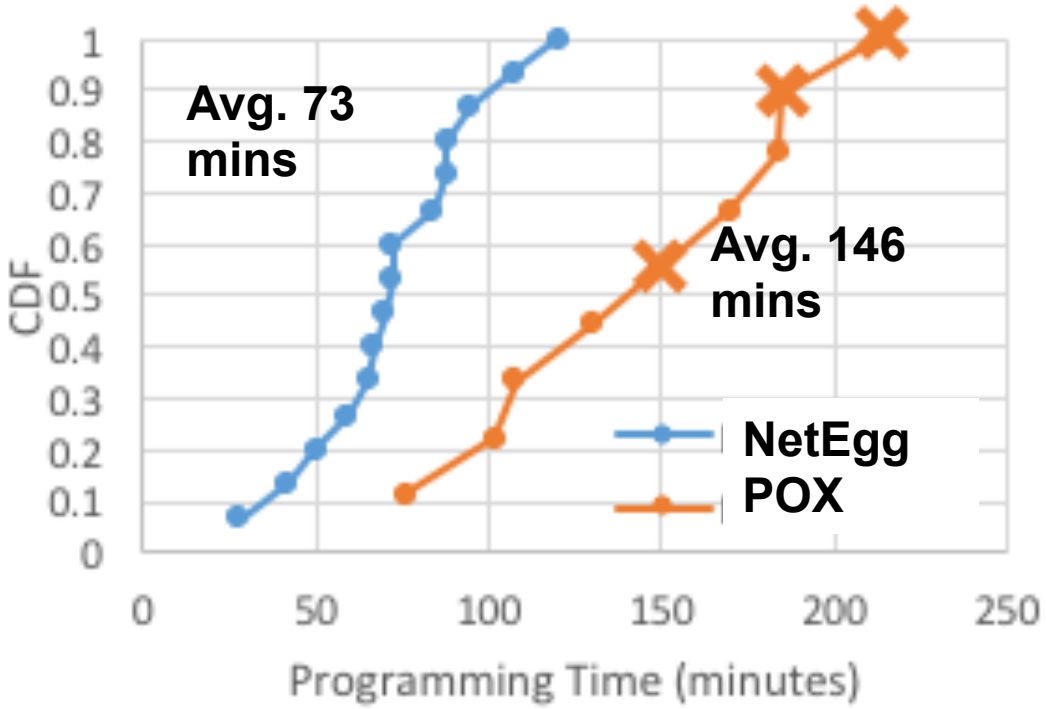
Requirment:

1. Allow all non-**IP**.
2. Block all **UDP** traffic from the Internet.
3. Allow all **UDP** traffic from the CS department.
4. Allow all **TCP** traffic from the CS department.
5. Block all **SSH** traffic from the Internet.
6. TCP traffic from any host A in the Internet at TCP port P to any host B in the CS at TCP port Q is blocked, **unless** B sends TCP traffic using TCP port Q to A at TCP port P before.

Q1: Error Rate



Q2: Programming Time



NetEgg achieves 50% reduction in programming time compared with POX.

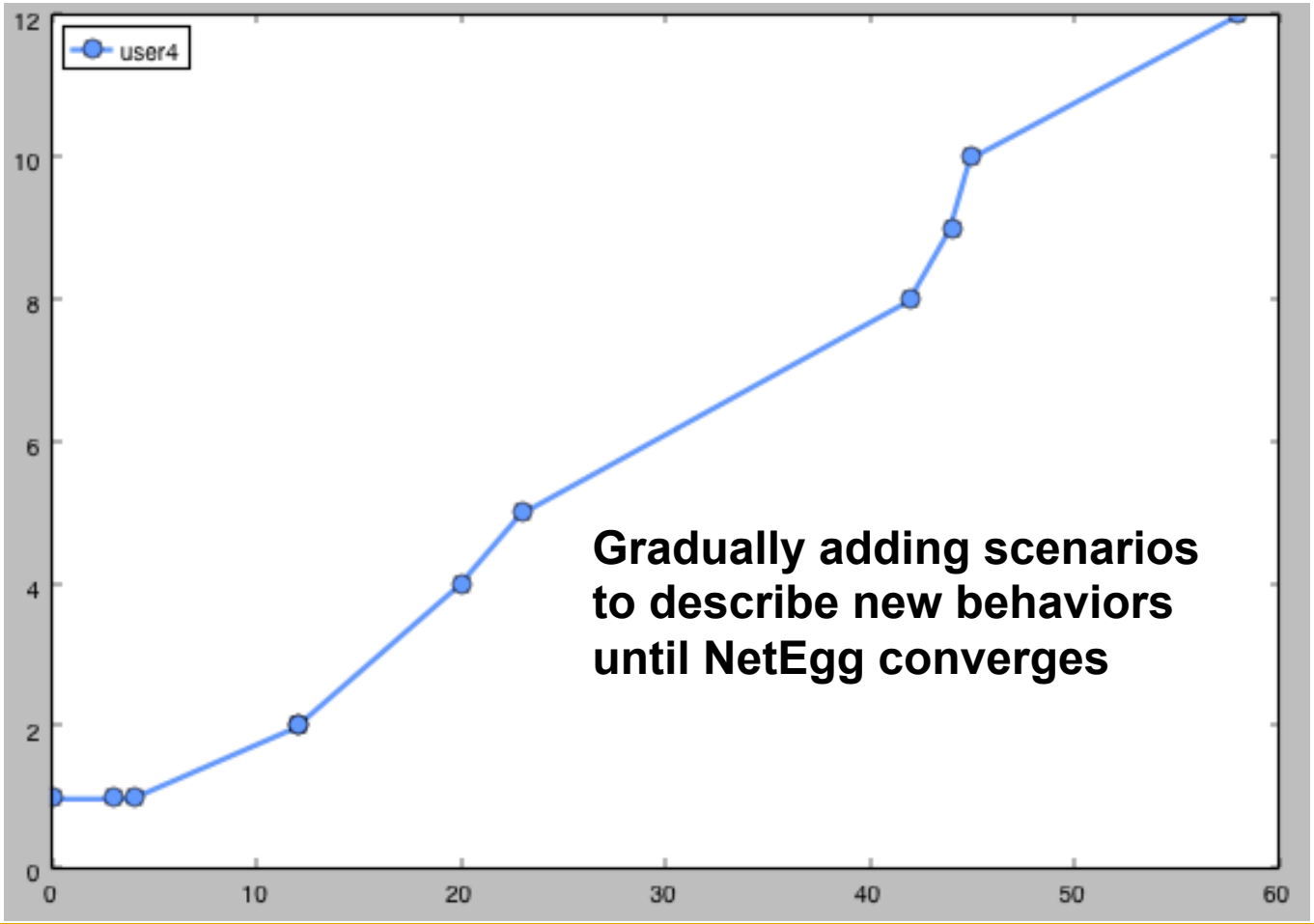
Q3: User Interaction Pattern

- Monitored the number of (unique) scenarios at each call to the synthesizer

- Identified three interaction patterns:
  - Smooth interaction
  - Back-and-forth interaction
  - Stuck interaction

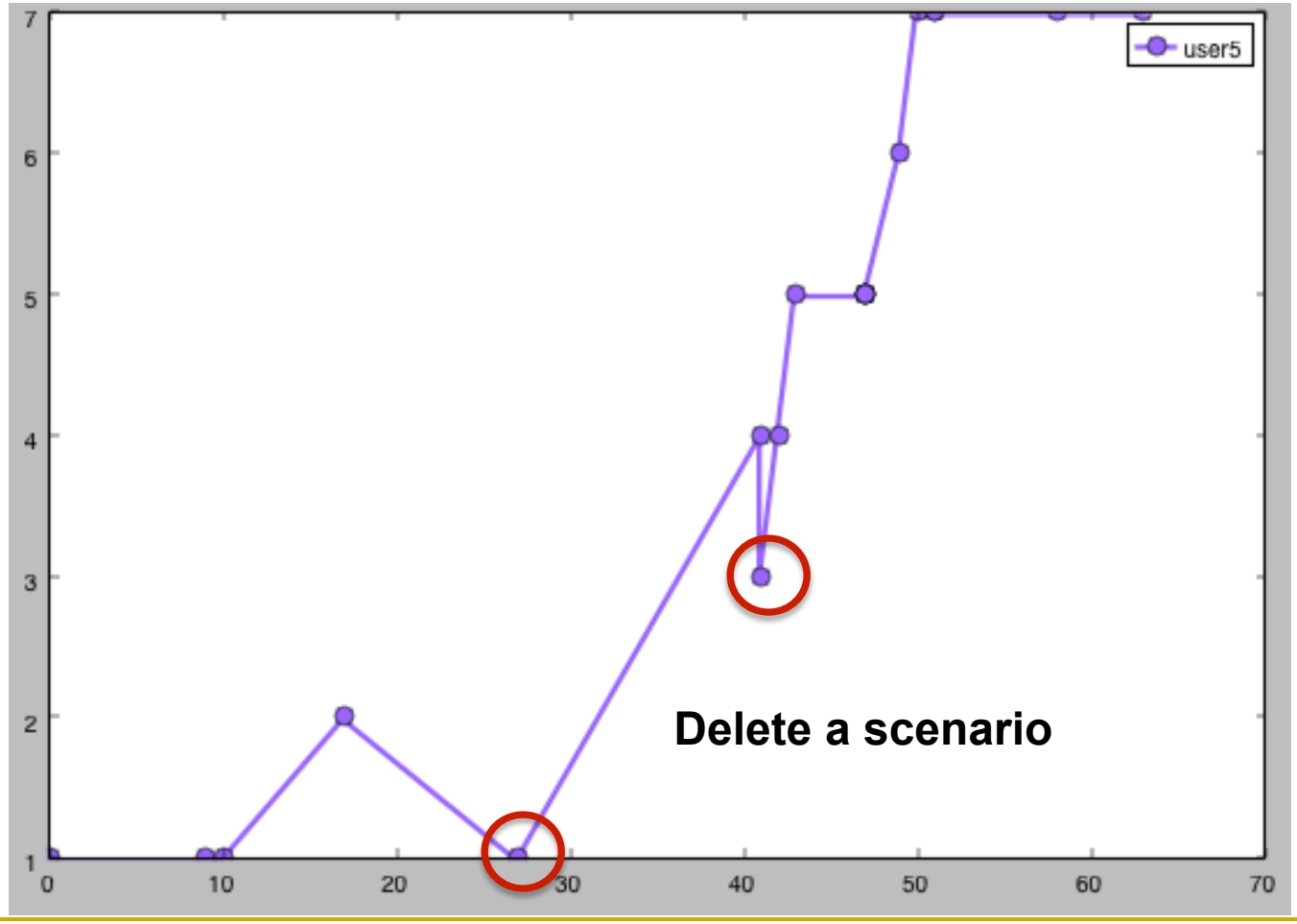
Smooth Interaction Pattern

# of Scenarios



Time (minute)

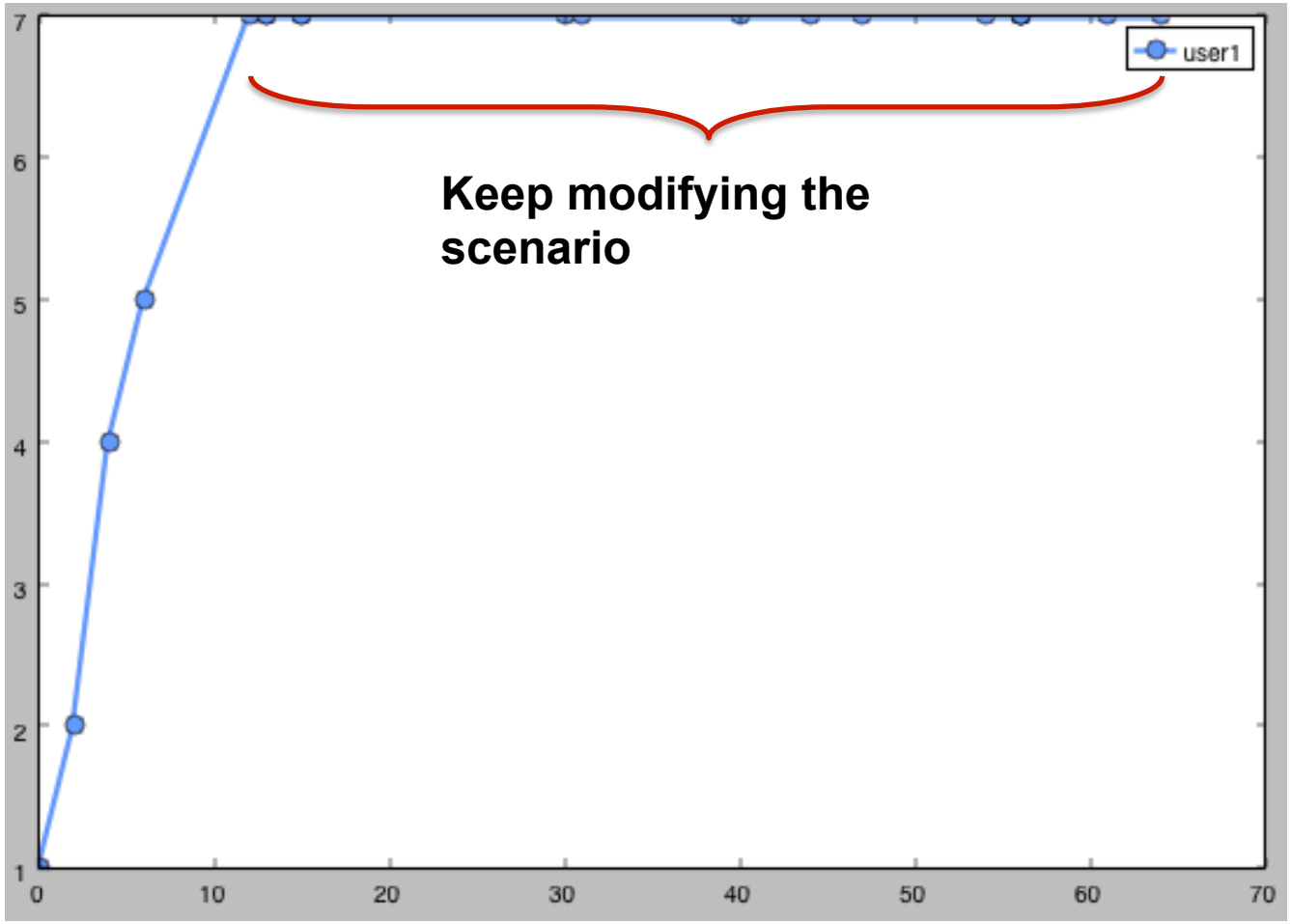
Back-and-forth Interaction Pattern



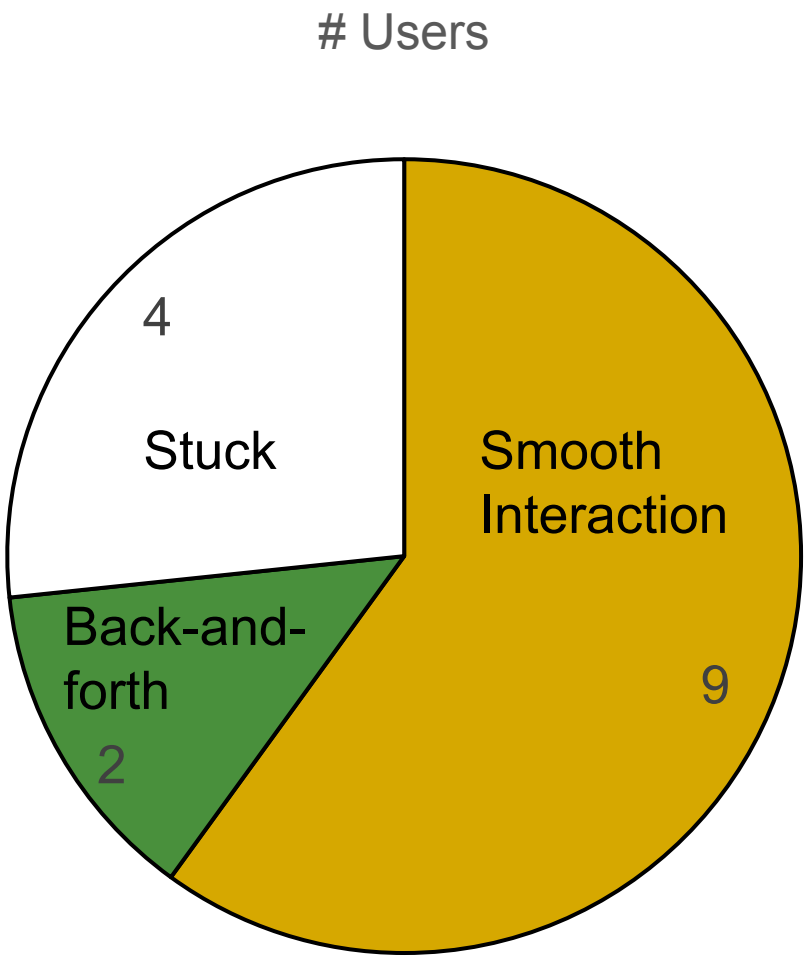
# of Scenarios

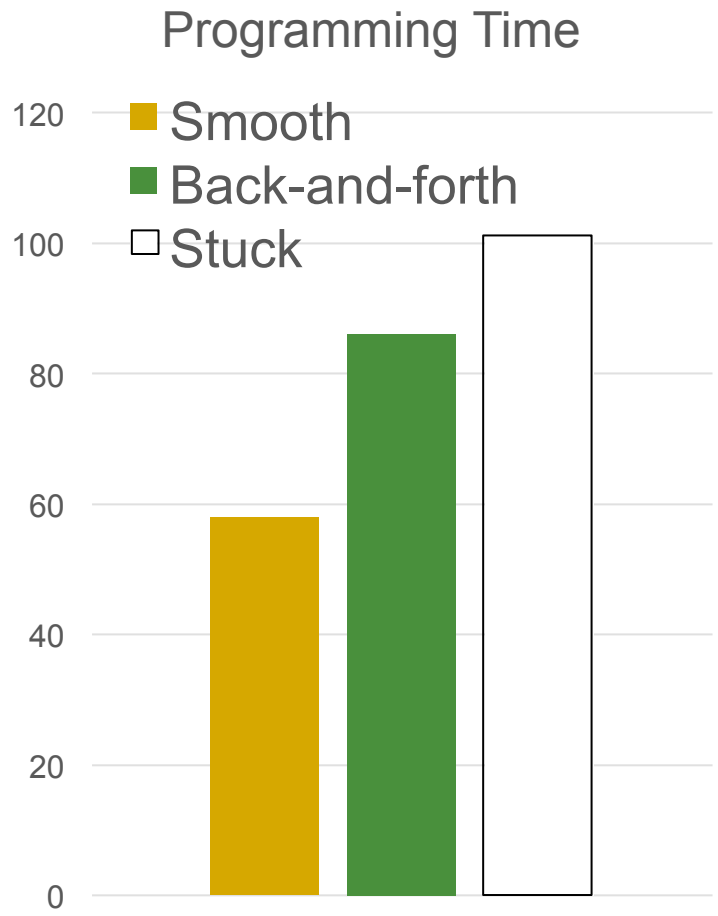
Time (minute)

Stuck Interaction Pattern



Statistics across Patterns





User Feedback

"the tool has a small learning curve and is quite intuitive . . . it is probably easier to use than a programming language."

"I am very clear about what I am doing when I use the interface, and know exactly how the packets are handled under what situations."

"(I) may get confused about how to define variables."

Some Takeaways on User Study

- Effective way to evaluate programming abstractions from the user point of view
  - Participants found the tool intuitive to use
  - Interesting observations: variables confusion and tendency to want to tweak compiled policy tables
- Finite pool of participants → need to get the design of the study correct to maximize participants' time
- Missing element: longitudinal analysis

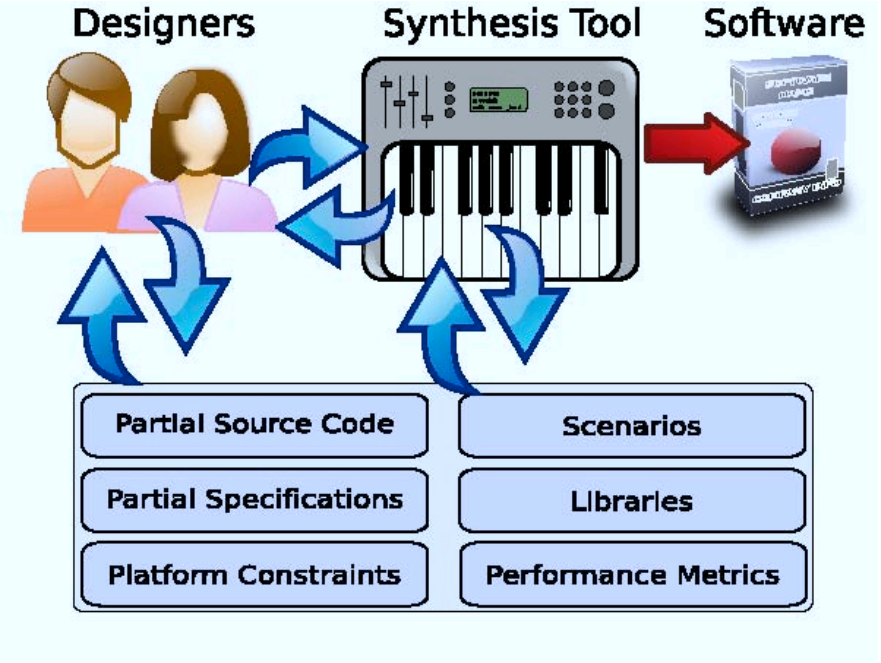
Summary

- Scenario-based programming for SDN policies:
  - Expressive to program a range of policies by examples
  - Target *non-programmers* who manage networks
  - Comparable performance to hand-crafted implementations
  - User study shows evidence of improved productivity
- Other similar opportunities in networking?
  - Measurements, data-plane processing, security attack patterns ….

Thank You!

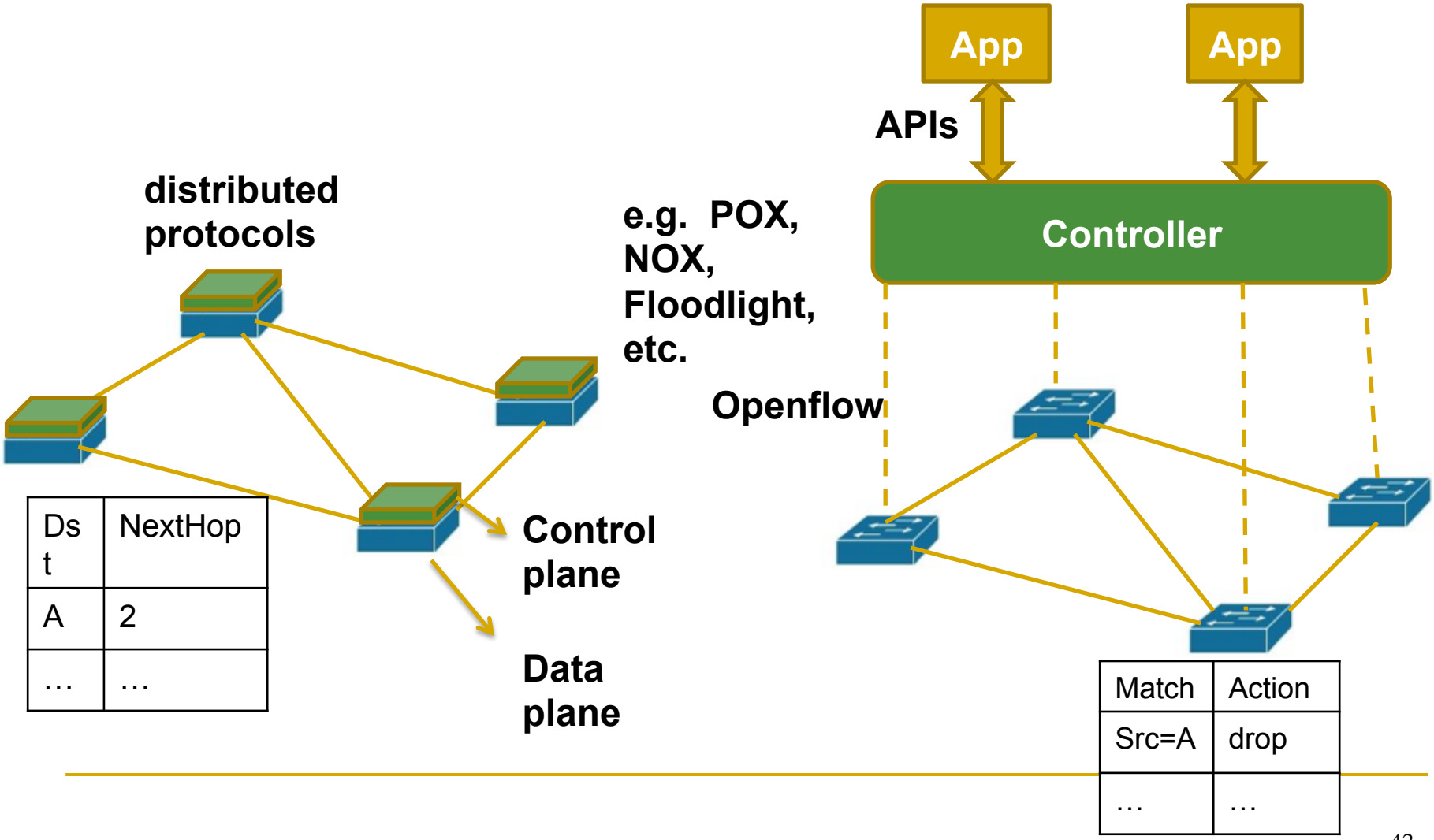
- netdb.cis.upenn.edu/netegg

NSF Expeditions in Computer Augmented Program Engineering
excape.cis.upenn.edu



- Designer expresses "what", using multiple input formats
- Synthesizer discovers new artifacts via integration
- Synthesizer solves computationally demanding problems using advanced analysis tools
- Interactive iterative design
- Integrated formal verification

Software-Defined Networking (SDN)



NetEgg Scenarios: Stateful firewall

<*port*, ether_type, srcip, *dstip* >