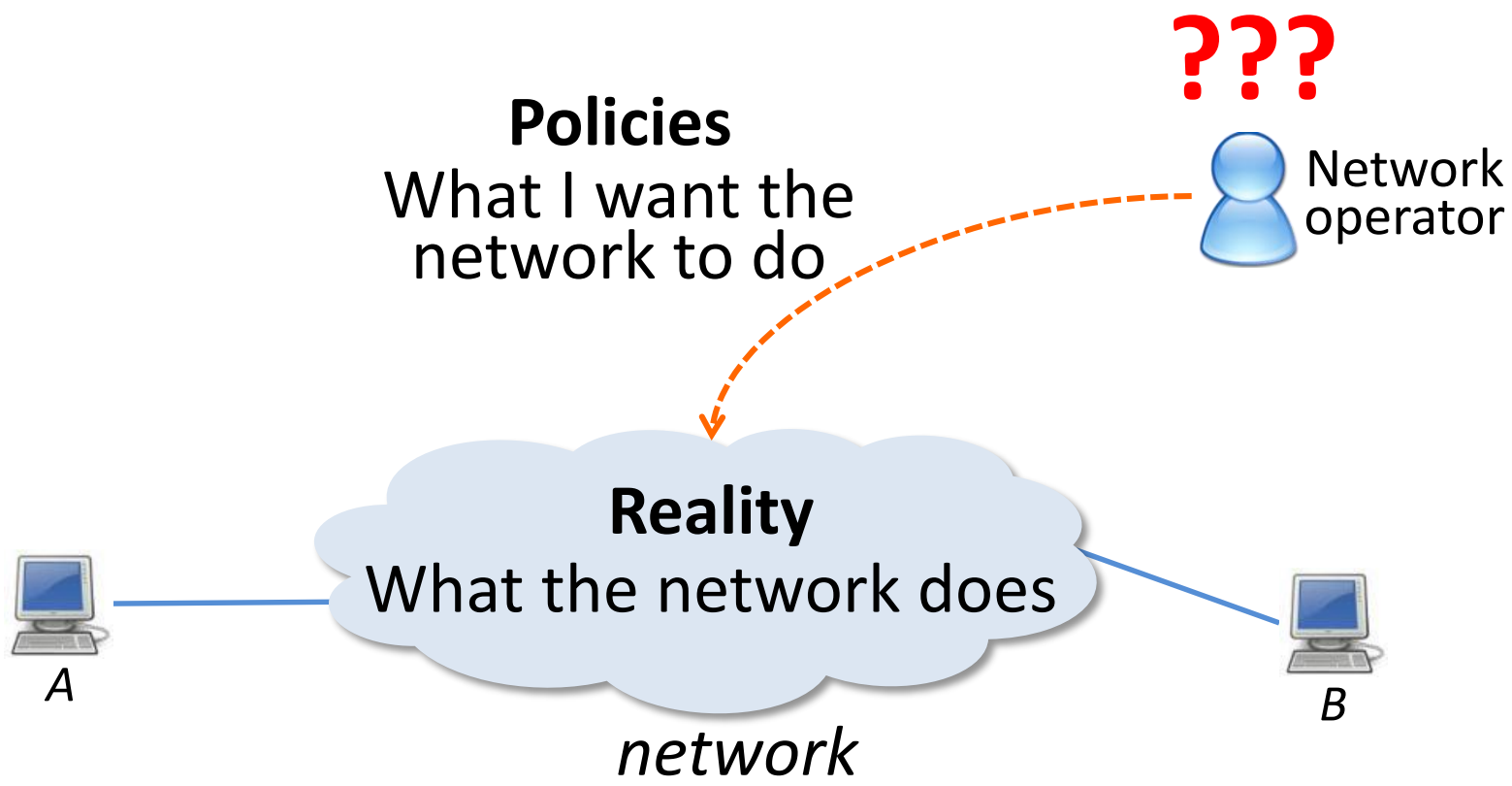
BUZZ: Testing Context-Dependent Policies in Stateful Networks

Seyed K. Fayaz, Tianlong Yu, Yoshiaki Tobioka, Sagar Chaki, Vyas Sekar

Carnegie Mellon University

Overview of checking network policies

Does the network do what I want it to do?



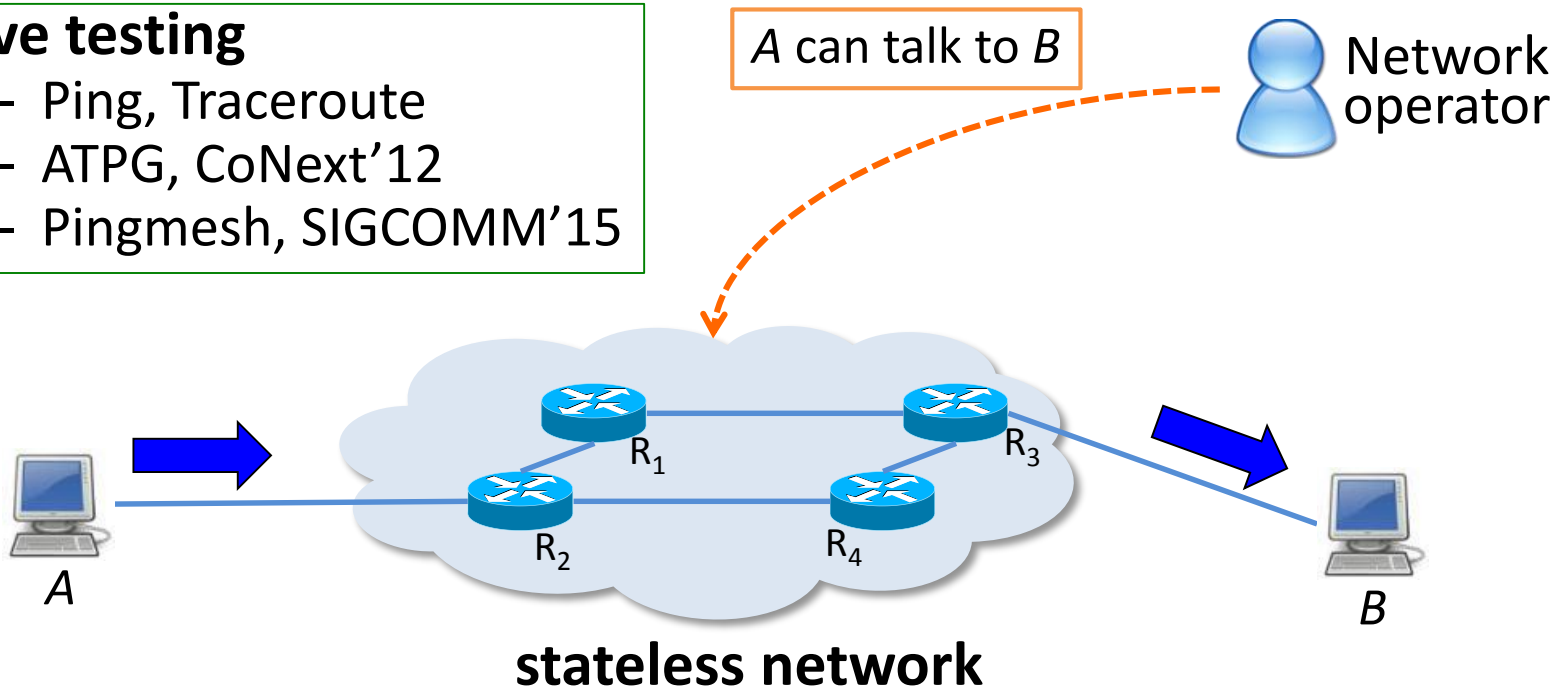
Existing work on checking network policies

Static verification
- HSA, NSDI'12
- Veriflow, NSDI'13
- NOD, NSDI'15
- Batfish, NSDI'15

Active testing
- Ping, Traceroute
- ATPG, CoNext'12
- Pingmesh, SIGCOMM'15

reachability policies

A can talk to *B*



Real networks are about more than reachability

- Reachability policies → Context-dependent policies
- Stateless networks → Stateful networks

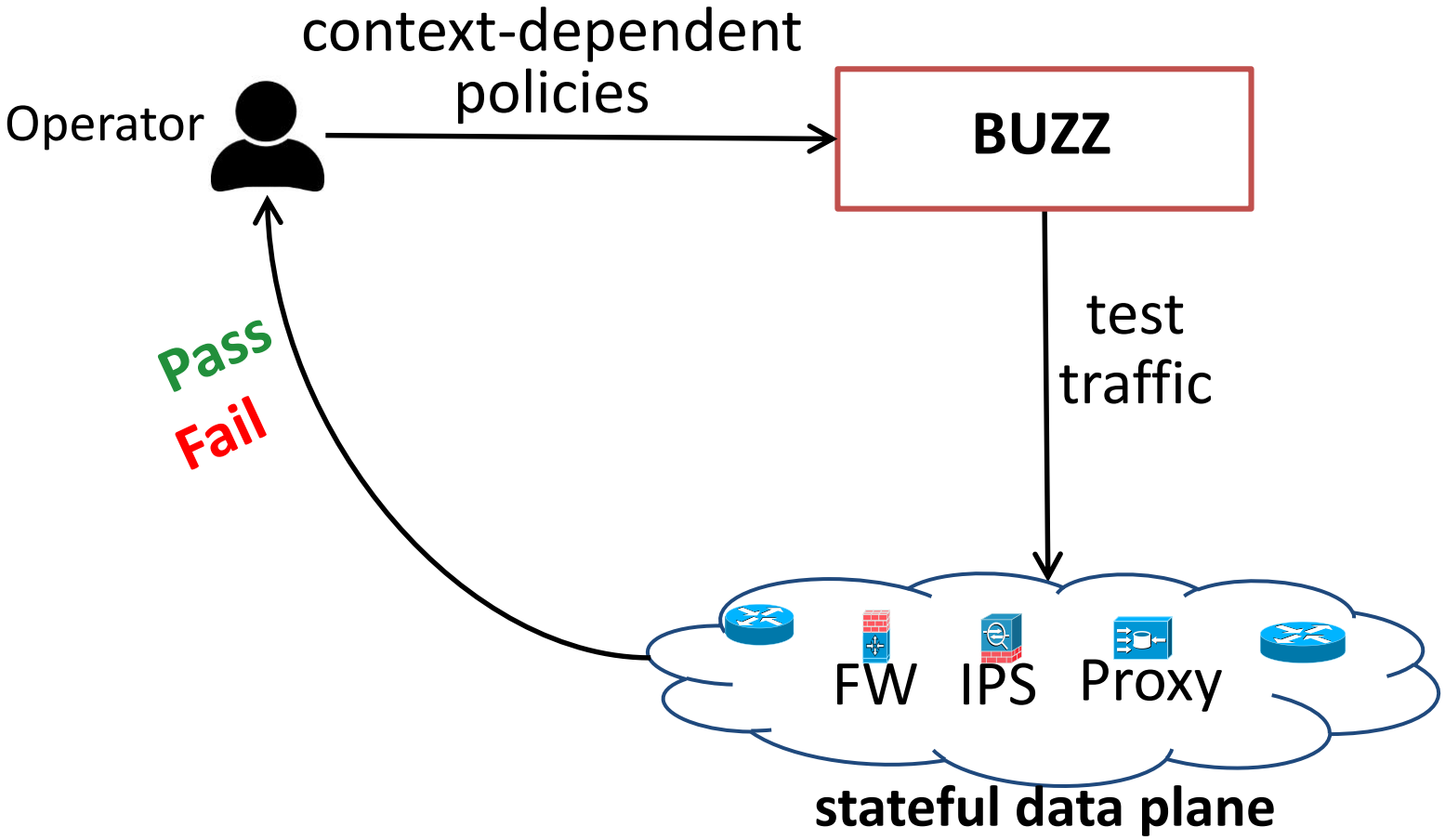
How can we check context-dependent policies in stateful networks?

Challenges:

- **Expressiveness:** How to capture stateful behaviors?
- **Scalability:** How to explore the state space?

Our solution: BUZZ

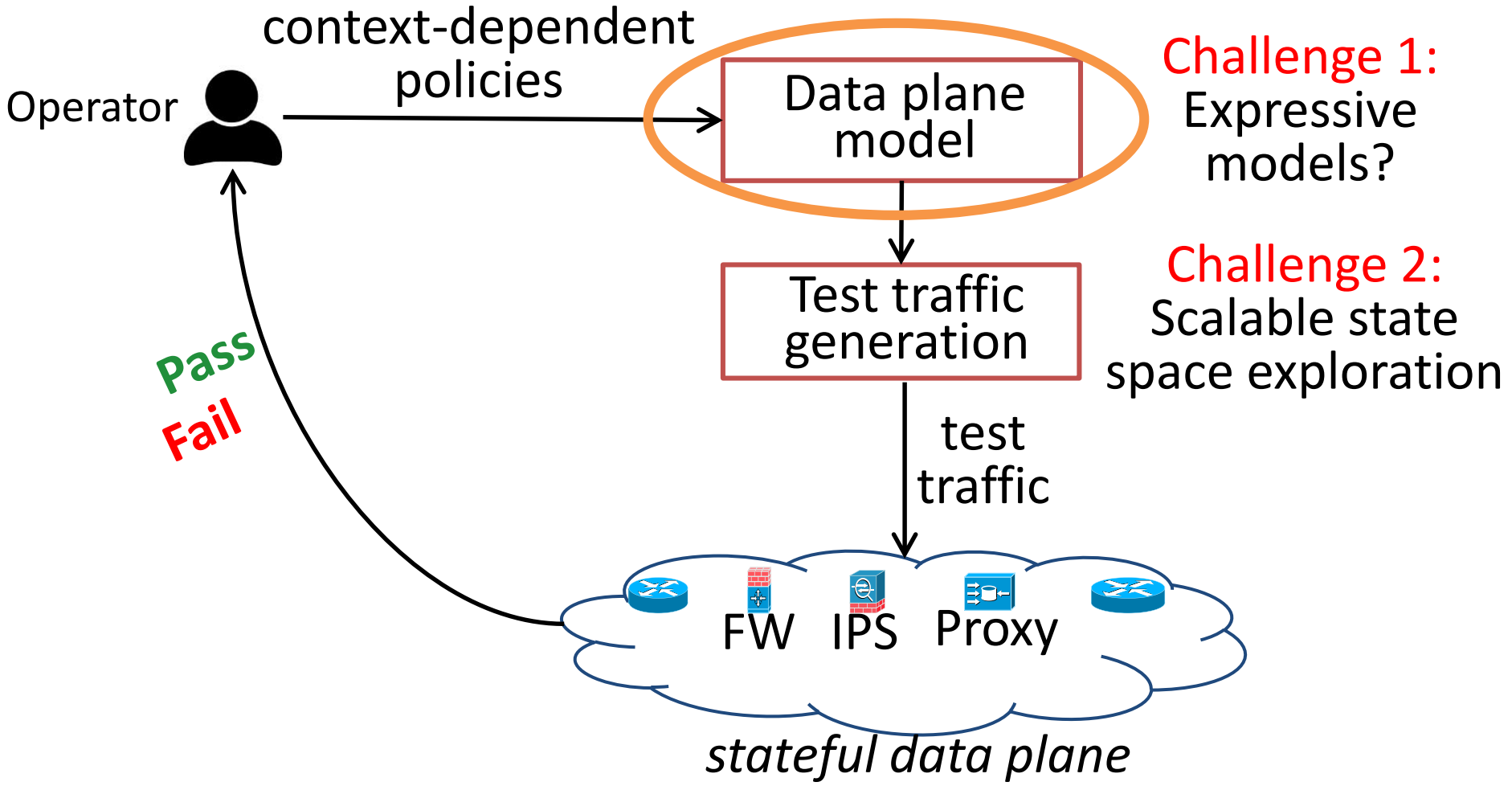
BUZZ is an active testing framework to check context-dependent policies in stateful data planes



Outline

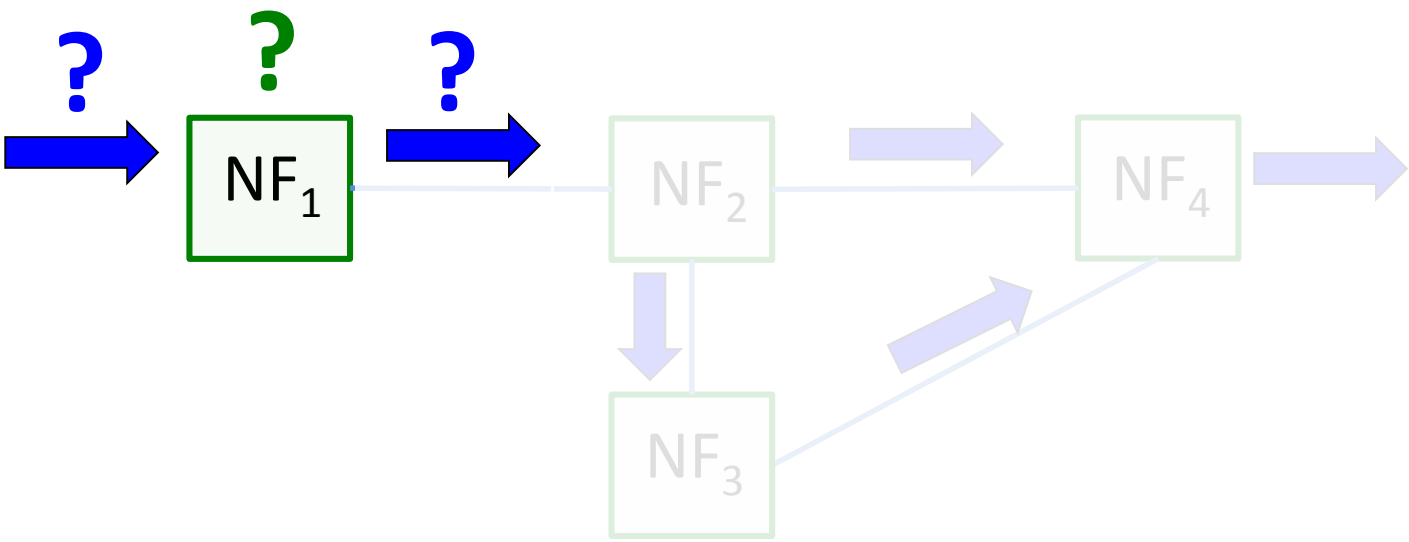
- Motivation and challenges
- Design of BUZZ
- Implementation and evaluation

Challenge 1: Expressive data plane model

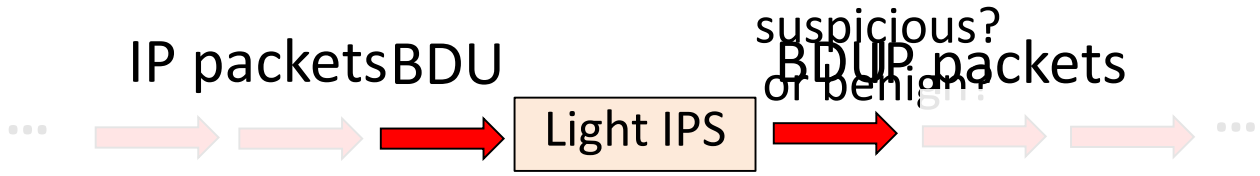


Challenge 1: Expressive data plane model

1. How to model the traffic unit?
2. How to model a network function (e.g., an IPS)?



Our idea: BDU as model of traffic unit



Located packet
(e.g., Pyretic, HSA)

```
struct locPkt {
 IPHder ipHdr;
 NetworkPort port;
};
```

Expressive ✗

Context-carrying located packet

```
struct CntxlocPkt {
 IPHder ipHdr;
 NetworkPort port;
 Context context;
};
```

Expressive ✓

Scalable ✗

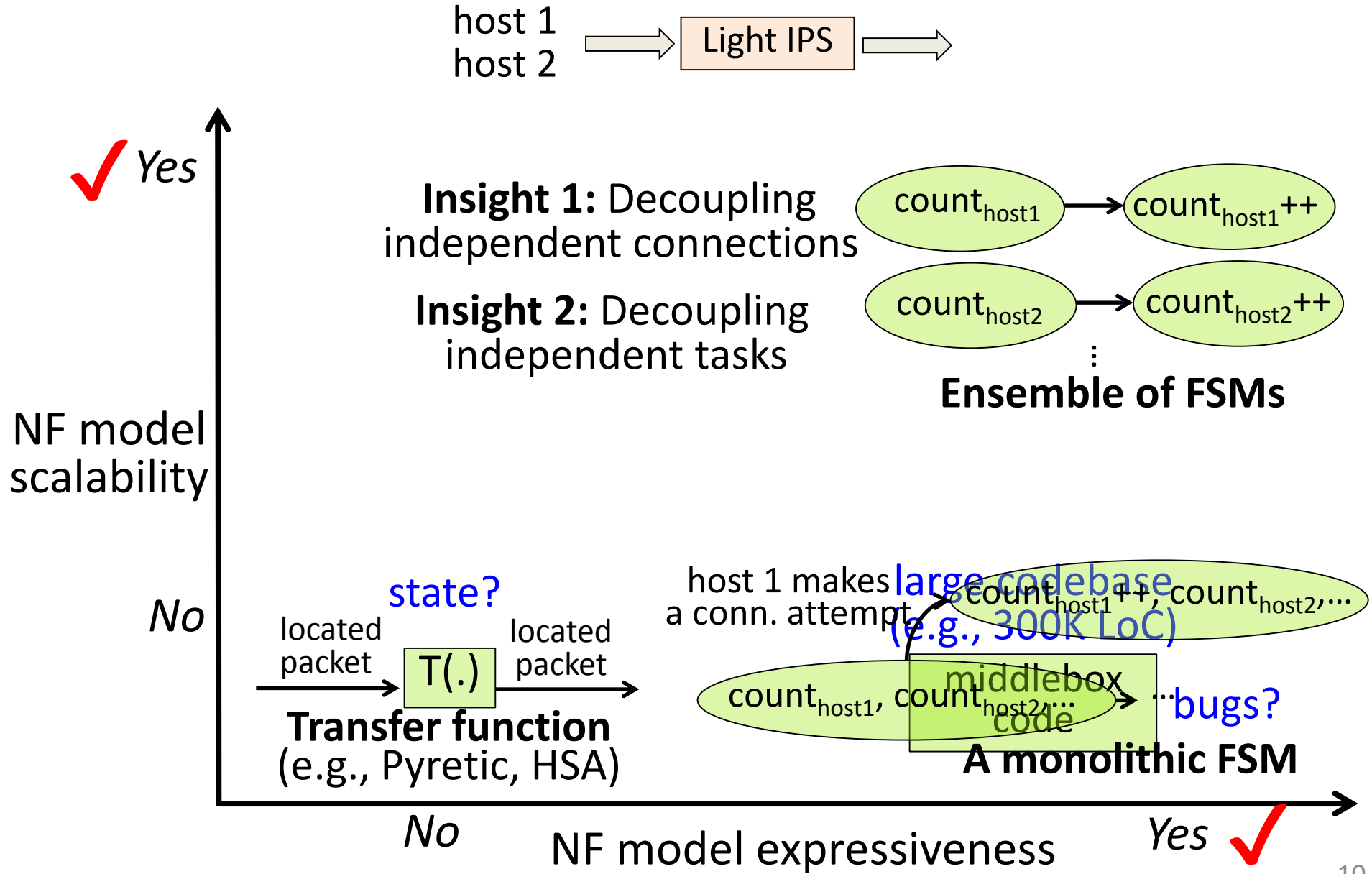
BUZZ Data Unit (BDU)

```
struct BDU{
 IPHeader ipHdr;
 NetworkPort port;
 Context context;
…
 HTTPHdr httpHdr
 …
};
```

Expressive ✓

Scalable ✓

Our idea: NF as an ensemble of FSMs



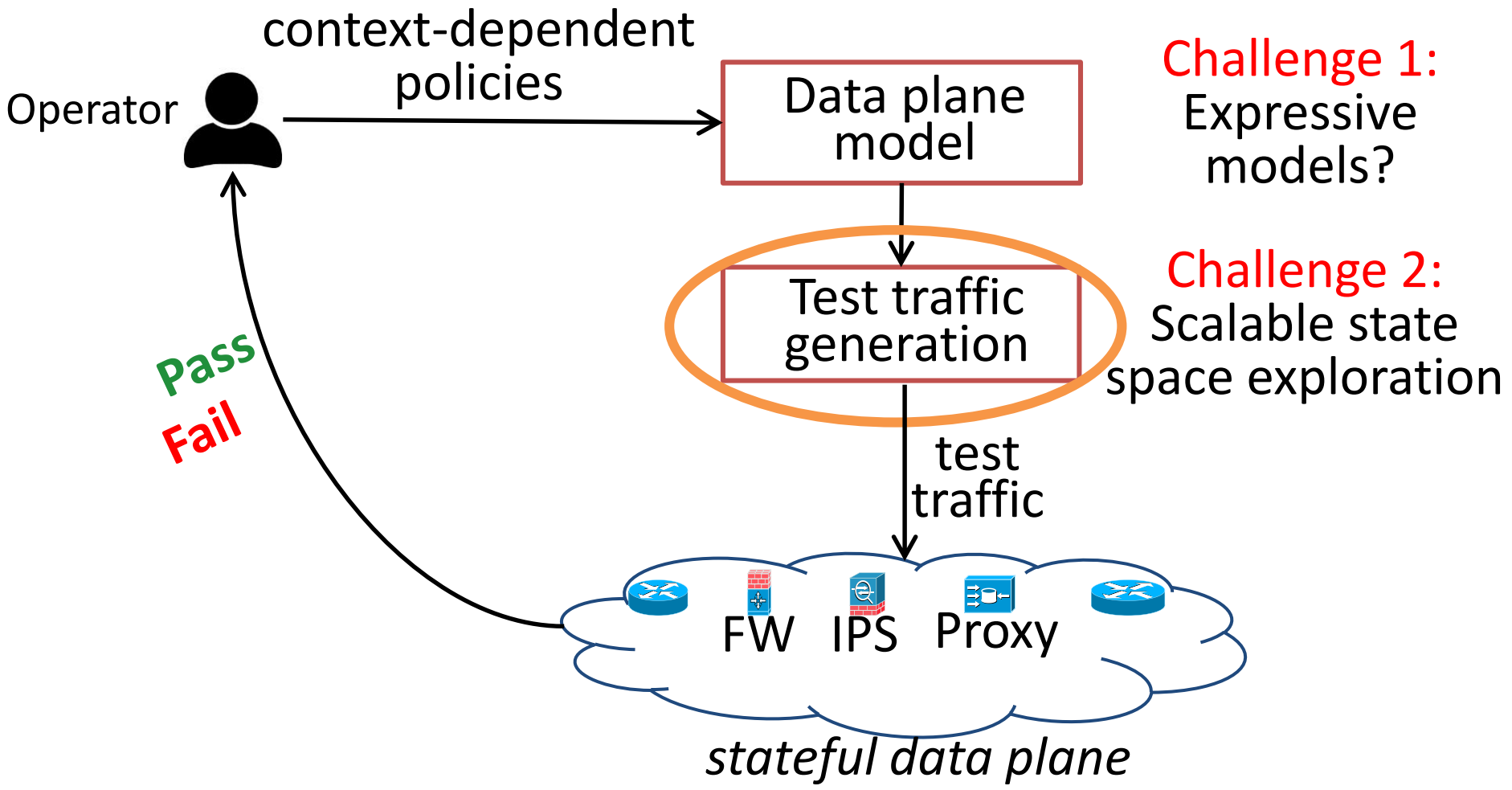
Putting it together: Composing NF models

Individual NF models

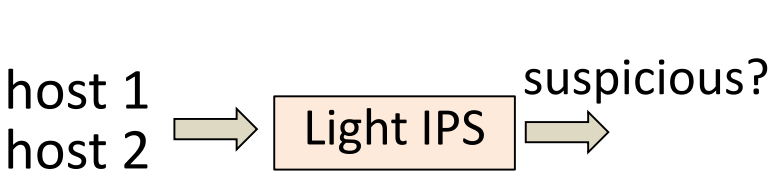
Data plane model

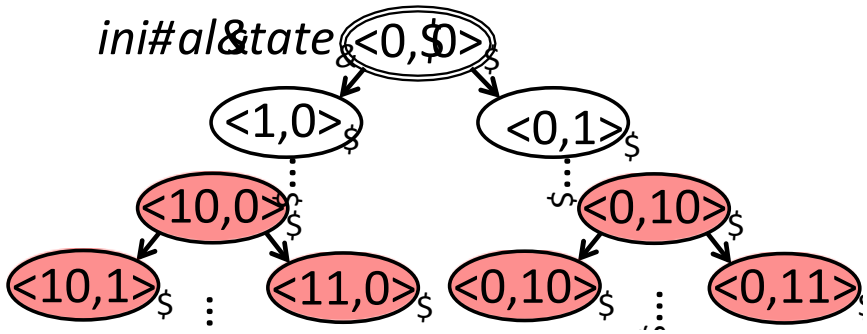
```
BDU A[20];
int objToWatch = XYZ.com;
int hostToWatch = H2;
// Global state variables
bool Cache[2][100]; // 2 proxies,  100 objects
// Model of a switch
BDU  Switch(NFId id, BDU inBDU){
    outBDU=lookUp(id, inBDU);
    return outBDU;}
// Model of a monitoring NF
BDU  Mon(NFId id, BDU inBDU){
     ...
     outBDU = inBDU;
     if (isHttp(id, inBDU)){
       takeMonAction(id, inBDU);/* if inBDU
       contains objToWatch destined to
       hostToWatch, set outBDU.dropped to 1.*/}
     ...
    return outBDU;}
// Model of a proxy NF;
BDU Proxy(NFId id, BDU inBDU){...}
main(){
  // Model of the data plane
  initializeProvenanceTags(A[]);
  for each injected A[i]
   while (!DONE(A[i])){
    Forward A[i] on current link;{
    A[i] = Next_NF(A[i]);{
    assert(
    (!(A[i].p-Tag==hostId[H2]))
    ||(!(A[i].c-Tags[cacheContext]==objToWatch));
}}}}
```

Challenge 2: Scalable test traffic generation



Challenge 2: Exploring data plane state space

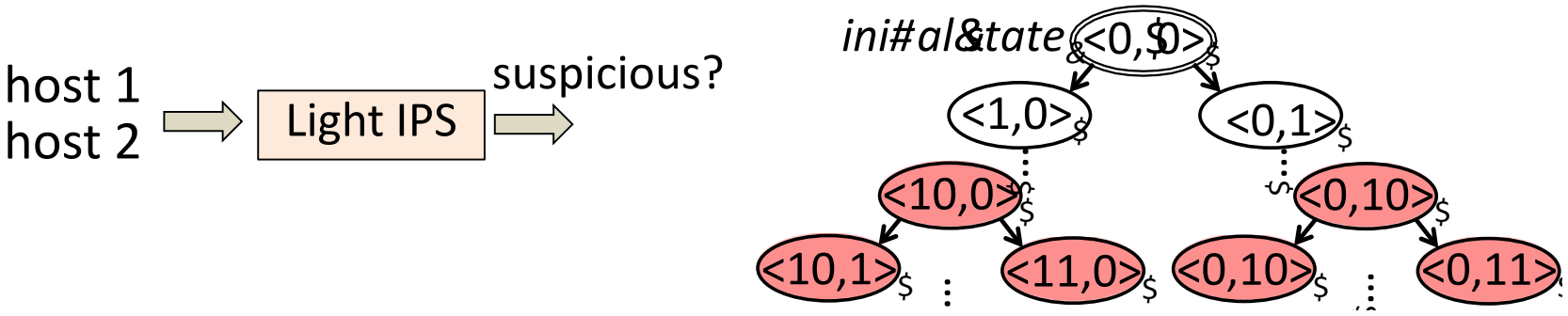




- **Conceptual view of test traffic generation**: How to reach a colored state through a sequence of traffic units?

- **Challenge of scalability** wrt traffic space and state space
  - **Strawman 1:** All possible sequences of traffic units
  - **Strawman 2:** Generate random traffic units (e.g., fuzzing)
  - **Strawman 3:** Naïve use of exploration tools (e.g., model checking)

Our idea: Test traffic generation using optimized symbolic execution



- **Our high-level approach:** Symbolic execution

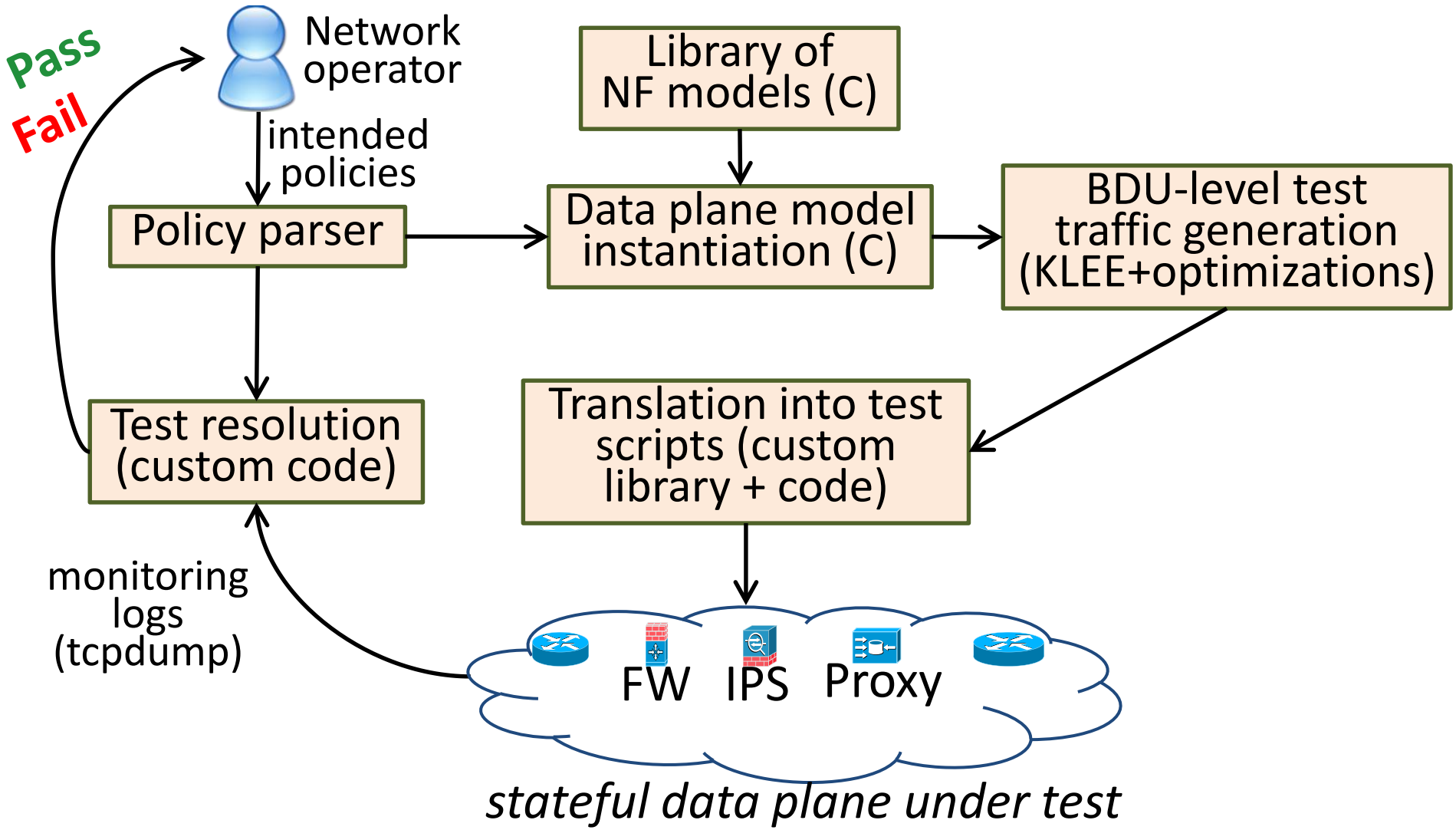
```
// Global state variables
int L_IPS_Alarm[noOfHosts];//alarm per host
int H_IPS_Alarm[noOfHosts];//alarm per host
...
//A[] is an array of symbolic BDUs
...
    assert(!(A[i].c-Tags[L_IPS_Alarm]==1))
```

- **Optimized symbolic execution**:
  - Minimize the number of symbolic BDUs
  - Scoping values of symbolic BDUs

Outline

- Motivation and challenges
- Design of BUZZ
- Implementation and evaluation

Implementation

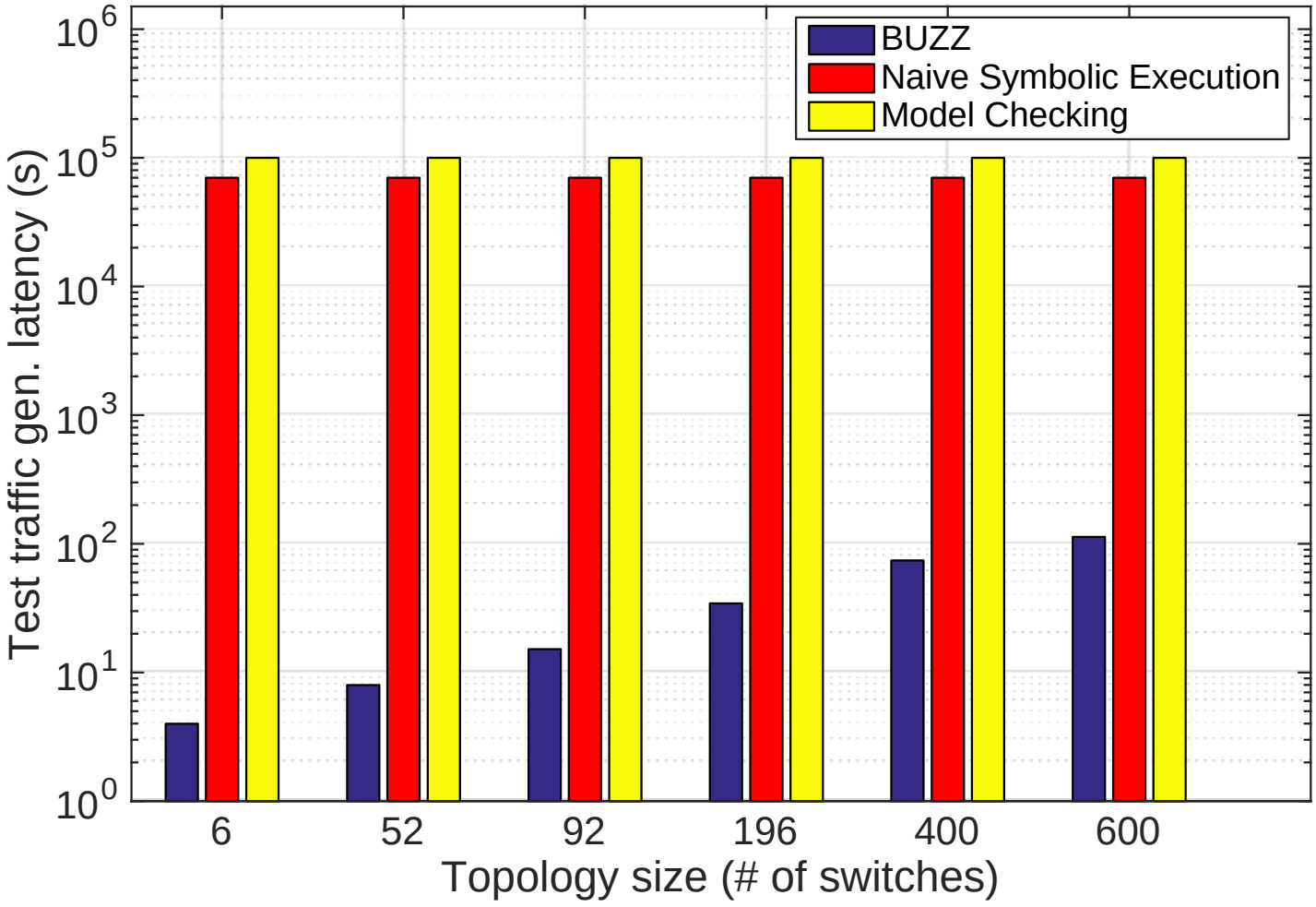


https://github.com/network-policy-tester/buzz

Evaluation: Effectiveness of BUZZ

- Found new bugs in recent SDN-based systems
  - Violations due to reactive control in Kinetic
  - Incorrect state migration in OpenNF
  - Faulty policy composition in PGA
  - Incorrect traffic tagging in FlowTags
  - …
- Found known violations
  - Broken link
  - Incorrect NAT configuration
  - SDN controller bug
  - …

Evaluation: Scalability of BUZZ



Test generation takes < 2min for a network with 600 switches and 60 middleboxes

Conclusions

- Existing work has fundamental limitations in checking context-dependent policies in stateful data planes
- **Challenges**:
  - Expressive-yet-scalable model of stateful data planes
  - Scalable state space exploration
- **Our solution is BUZZ**:
  - BUZZ Data Unit (BDU) as traffic unit model
  - Ensemble of FSMs as a network function (NF) model
  - Scalable exploration via domain-specific optimizations
- BUZZ can help find bugs and is scalable