trilogy 2

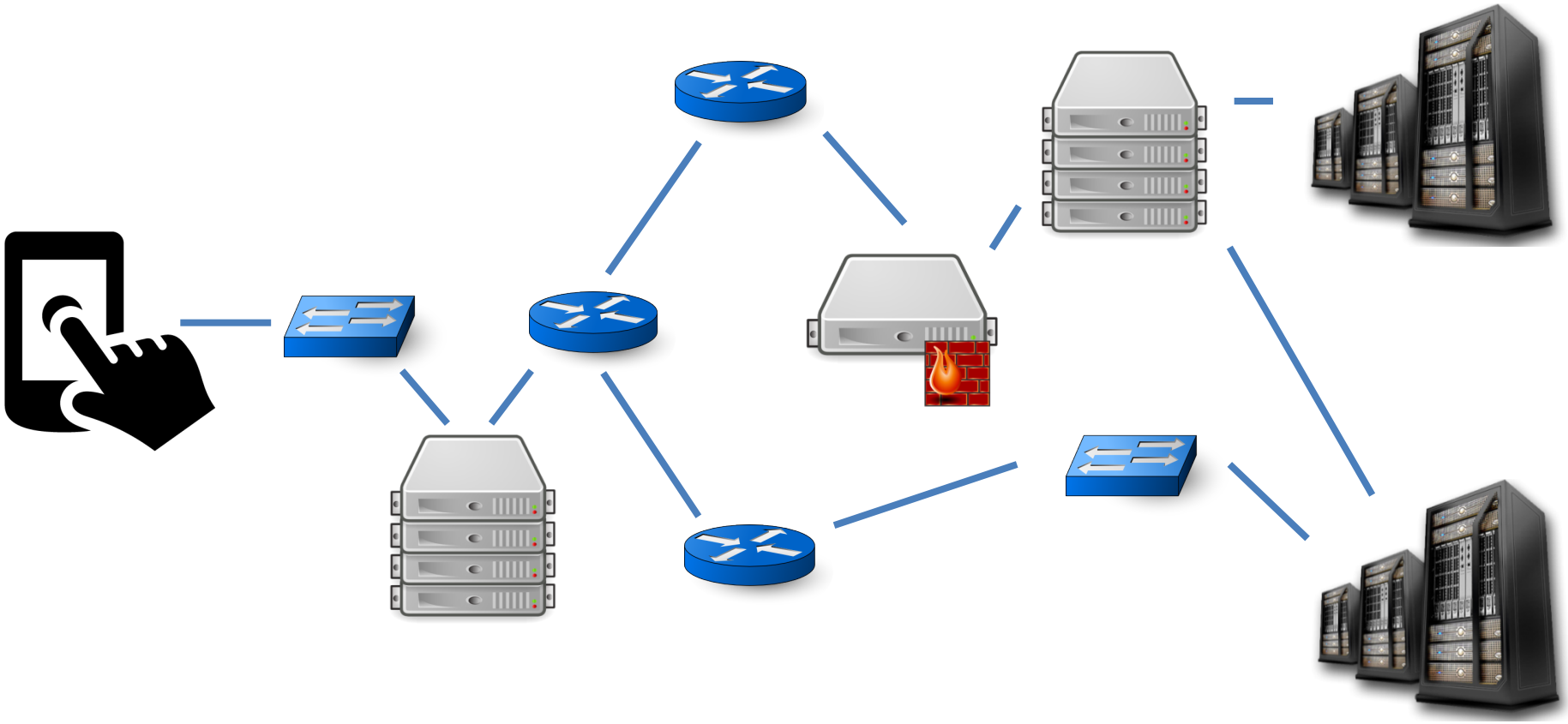
SUPERFLUIDITY

Symnet: scalable symbolic execution for modern networks

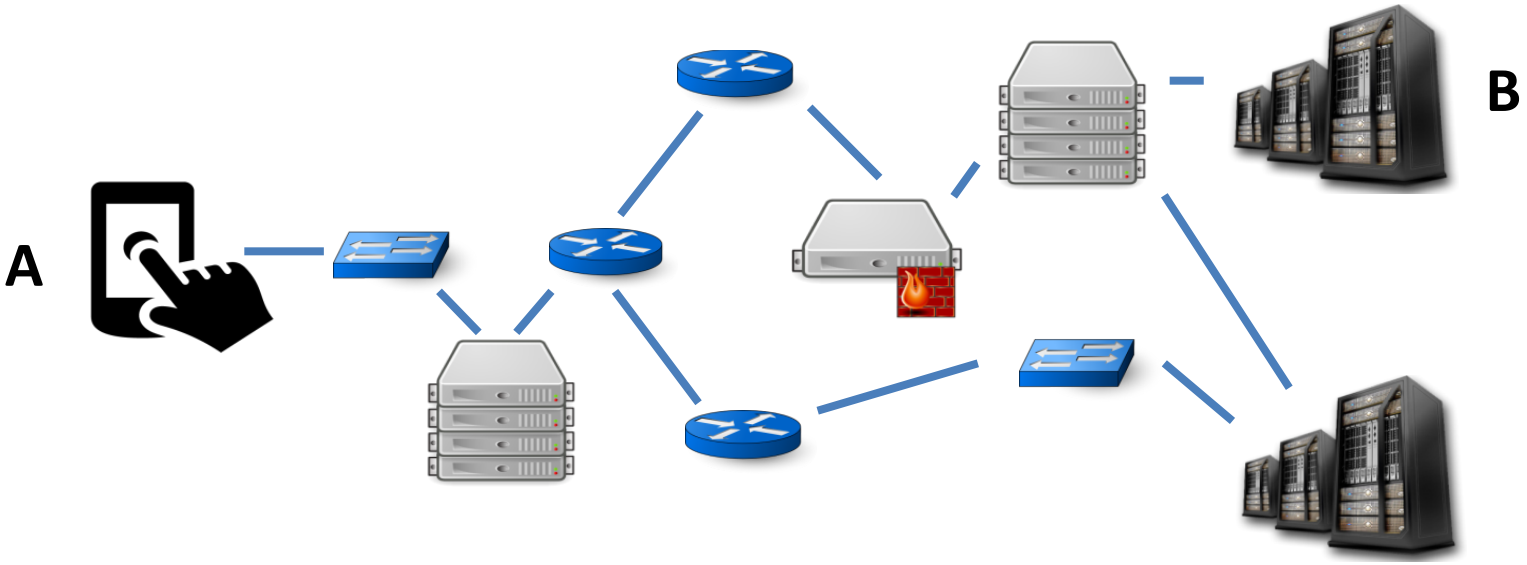
University Politehnica of Bucharest

Radu Stoenescu, Matei Popovici, Lorina Negreanu and Costin Raiciu

Networks are increasingly complex



Understand the network

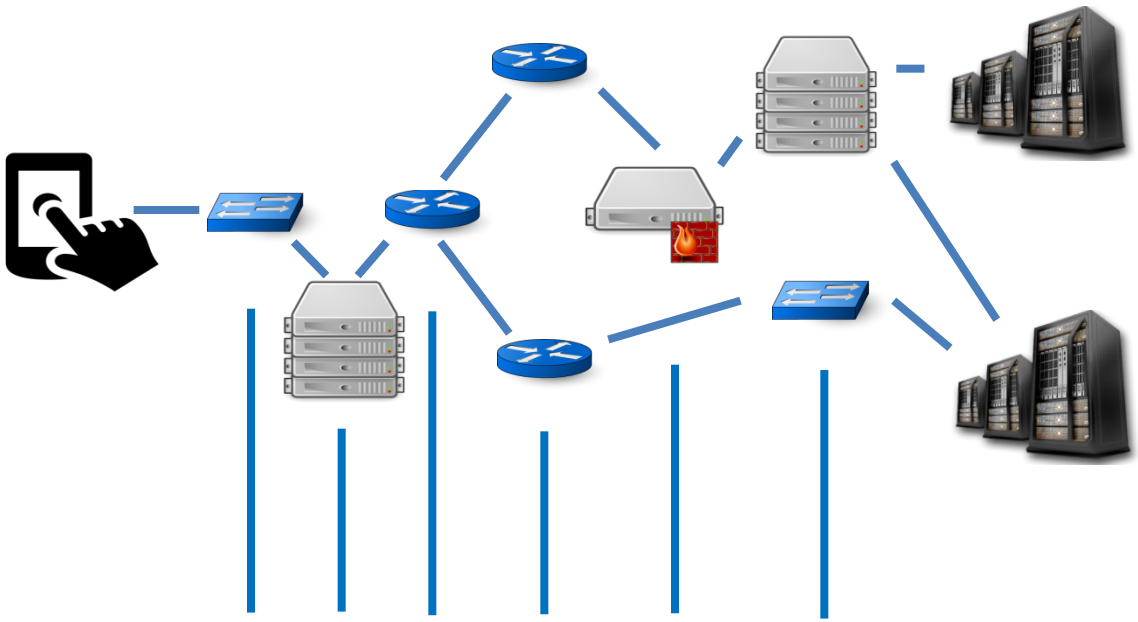


Reachability

Packet modifications

Security policy violations

Static verification to the rescue



Data plane snapshot

↓

Network model

↓

Verification engine

Symbolic Execution Friendly Language (SEFL) - Network model

Symnet - Verification engine

Choosing a modeling language

C code

- Expressive, well understood
- Symbolic execution captures many properties
- Very expensive to verify

Middle ground

Header Space Analysis

- Cheap, scalable
- No arbitrary protocol layering
- Only captures reachability

Symbol execution of firewall - C code

```
1: packet* filter(packet* p){
2:     if (p->dst_port==80)
4:         return p;
5:     else {
6:         free p;
7:         return NULL;
8:     }
9:}
```

Symbol execution of firewall - C code

Path 1

*p=**

```
1: packet* filter(packet* p){
2:     if (p->dst_port==80)
4:         return p;
5:     else {
6:         free p;
7:         return NULL;
8:     }
9:}
```

Symbol execution of firewall - C code

Path 1

*p=**

```
1: packet* filter(packet* p){
2:     if (p->dst_port==80)
4:         return p;
5:     else {
6:         free p;
7:         return NULL;
8:     }
9:}
```

Symbol execution of firewall - C code


```
1: packet* filter(packet* p){
2:     if (p->dst_port==80)
4:         return p;
5:     else {
6:         free p;
7:         return NULL;
8:     }
9:}
```

p->dst_port=80 *p->dst_port!=80*

Symbol execution of firewall - C code

Path 1 Path 2

```
1: packet* filter(packet* p){
2:     if (p->dst_port==80)
4:         return p;
5:     else {
6:         free p;
7:         return NULL;
8:     }
9:}
```

p->dst_port=80 *p->dst_port!=80*

Symbol execution of firewall - C code

Path 1 Path 2

```
1: packet* filter(packet* p){
2:     if (p->dst_port==80)
4:         return p;
5:     else {
6:         free p;
7:         return NULL;
8:     }
9:}
```

p->dst_port!=80

p->dst_port=80

filter = p

Symbol execution of firewall - C code

Path 1 Path 2

```
1: packet* filter(packet* p){
2:     if (p->dst_port==80)
4:         return p;
5:     else {
6:         free p;
7:         return NULL;
8:     }
9:}
```

p->dst_port!=80

p->dst_port=80

filter = p

Symbol execution of firewall - C code

Path 1 Path 2

```
1: packet* filter(packet* p){
2:     if (p->dst_port==80)
4:         return p;
5:     else {
6:         free p;
7:         return NULL;
8:     }
9:}
```

p->dst_port!=80

p->dst_port=80

filter = p

Symbol execution of firewall - C code

Path 1 Path 2

```
1: packet* filter(packet* p){
2:     if (p->dst_port==80)
4:         return p;
5:     else {
6:         free p;
7:         return NULL;
8:     }
9:}
```

p->dst_port=80
filter = p

p=NULL

Symbol execution of firewall - C code

Path 1 Path 2

```
1: packet* filter(packet* p){
2:     if (p->dst_port==80)
4:         return p;
5:     else {
6:         free p;
7:         return NULL;
8:     }
9:}
```

p->dst_port=80
filter = p

p=NULL

Symbol execution of firewall - C code

Path 1 Path 2

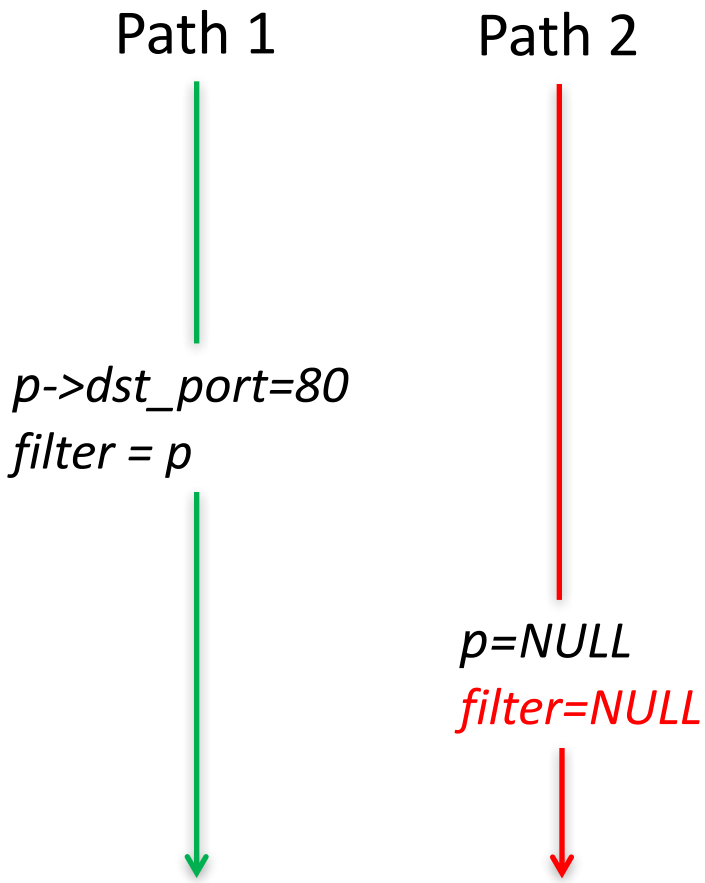
```
1: packet* filter(packet* p){
2:     if (p->dst_port==80)
4:         return p;
5:     else {
6:         free p;
7:         return NULL;
8:     }
9:}
```

p->dst_port=80
filter = p

p=NULL
filter=NULL

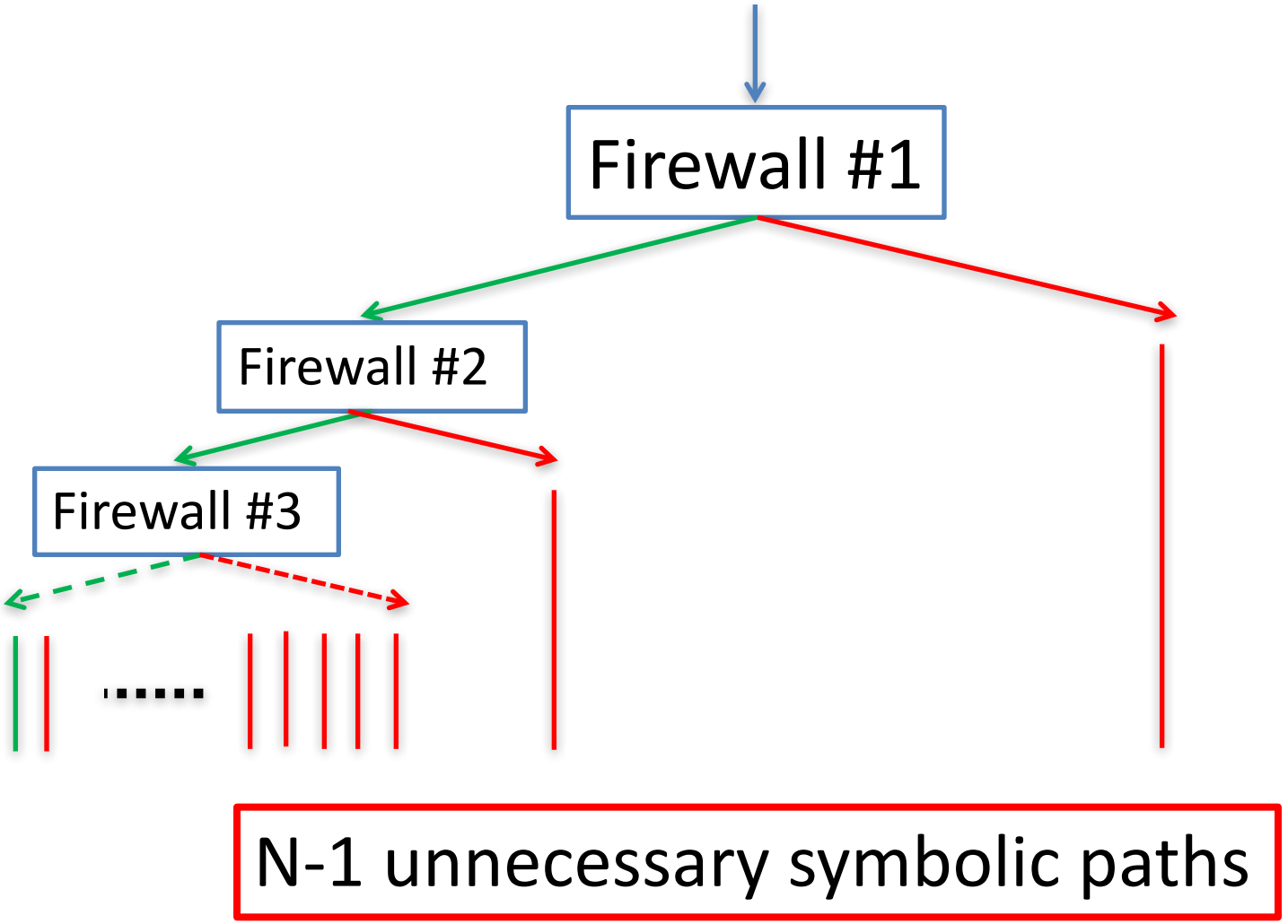
Symbol execution of firewall - C code

```
1: packet* filter(packet* p){
2:     if (p->dst_port==80)
4:         return p;
5:     else {
6:         free p;
7:         return NULL;
8:     }
9:}
```



Two symbolic paths vs. one viable in the network
Non-packet processing being executed

Symbol execution of firewalls - C code



N-1 unnecessary symbolic paths

Symbolic execution of network data plane implementations does not scale

- A core IP router results in hundreds of thousands of paths
- For a TCP options-parsing middlebox, runtime depends on option length (<40):
  - 6B ~ 1 hour, 7B ~ 3 hours

Principles for scalable data plane symbolic execution

Fundamental tradeoff between fast symbolic execution and runtime efficiency [Wagner‘13]

=> Use models of networks instead of real code

Only analyze relevant code

=> 1 execution path == 1 network packet

Complex data structures kill symbolic execution

=> Use symbolic-execution friendly data structures

Loops + conditionals are dangerous

=> Careful looping semantics with low branching factor

Our solution

SEFL symbolic execution friendly language

Symnet symbolic execution tool

Memory safety by design

- The memory space is the packet
- No pointers
- Memory access via concrete offsets; validated

Symbolic execution constructs part of the language

- Explicit forking of new execution paths
- Explicit stating of path constraints

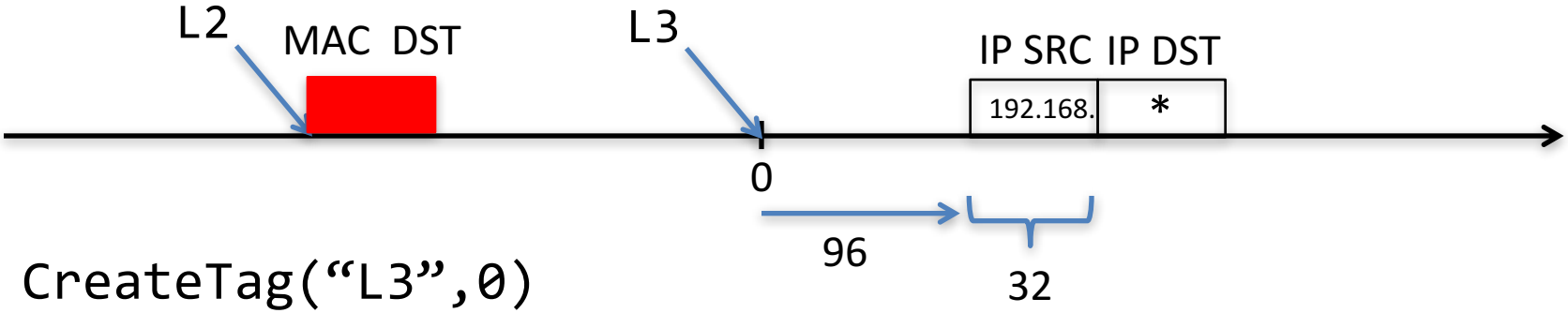
No arbitrary data structures

- Only a map data structure

SEFL symbolic execution friendly language

- Variables are packet headers or metadata
  - **Packet headers** allocated at specific addresses in the packet header
  - **Metadata** are key/value pairs in a map data structure

The packet header in SEFL




```
CreateTag("L3",0)
Allocate(IpSrc,32)
Assign(IpSrc,"192.168.1.1")
Allocate(IpDst,32)
Assign(IpDst,Symbolic)
CreateTag("L2",Tag("L3)-112)
Assign(DstMac,Symbolic)
```

Allocate(IpSrc,32) ; IpSrc = Tag("L3")+96

ERROR

Firewall

C

```
1:packet* filter(packet* p){
2: if (p->dst_port==80)
4:    return p;
5: else {
6:    free p;
7:    return NULL;
8: }
9:}
```

SEFL

```
1: filter(){
2:    constrain(IpDst==80);
3: }
```

Only relevant paths explored
Concise

Symnet symbolic execution tool

- 10K LOC of Scala; Z3 for constraint solving

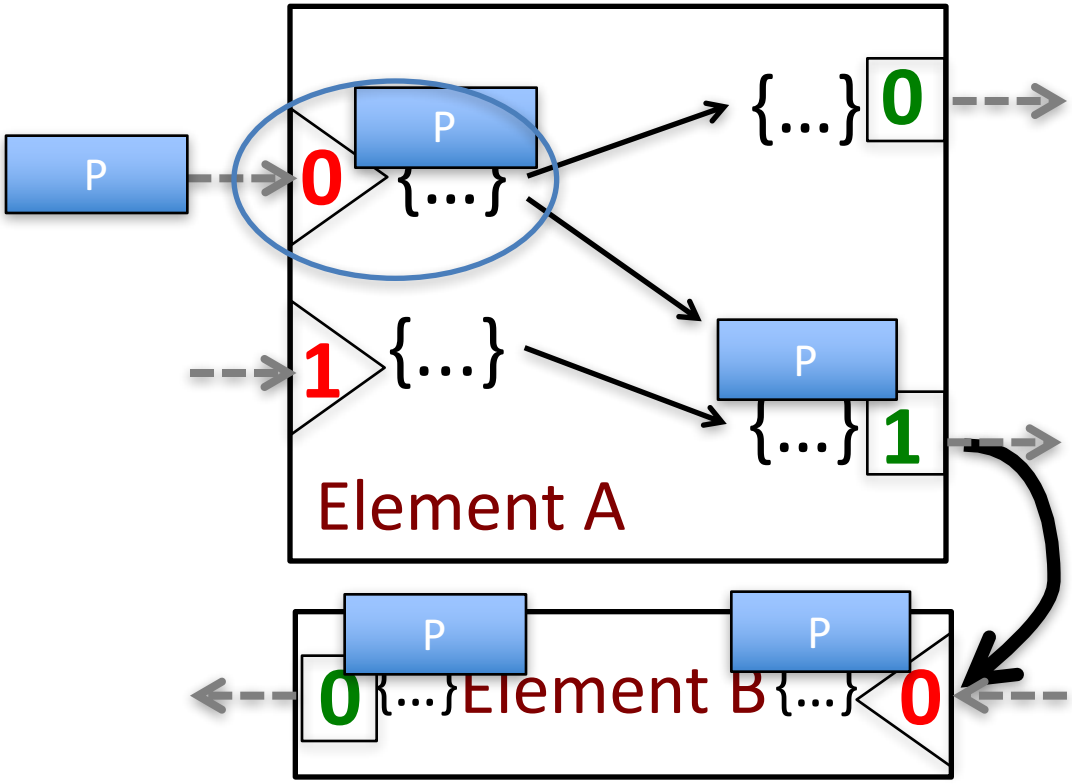
Input: SEFL network model

- SEFL models of individual network elements
- Connections between elements

Output: all feasible symbolic paths

- Values of header and metadata fields
- Path constraints

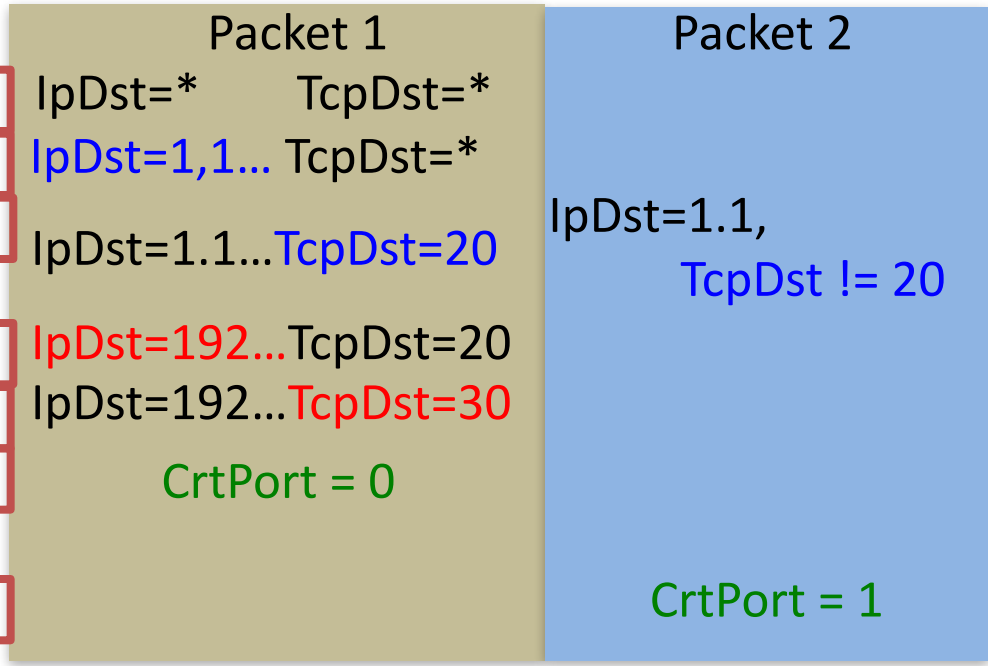
SEFL Network Models



Symbolic execution of filter + DNAT

Element A model

```
InputPort(0):
 Constrain(IPDst==1.1.1.1),
 If (Constrain(TcpDst==20),
   InstructionBlock(
     Assign(IPDst,192.168.0.1),
     Assign(TcpDst,30),
     Forward(OutputPort(0))
   ),
   Forward(OutputPort(1)),
```



- Reachability
- Loop detection
- Invariant header fields
- Header memory safety

Ready-made network models

Modeling network boxes is fairly difficult

We have developed parsers that output SEFL code from:

- Router/switch forwarding table snapshots
- CISCO ASA firewall configuration
- Click modular router configurations
- Openstack Neutron network configurations

Evaluation

Model correctness

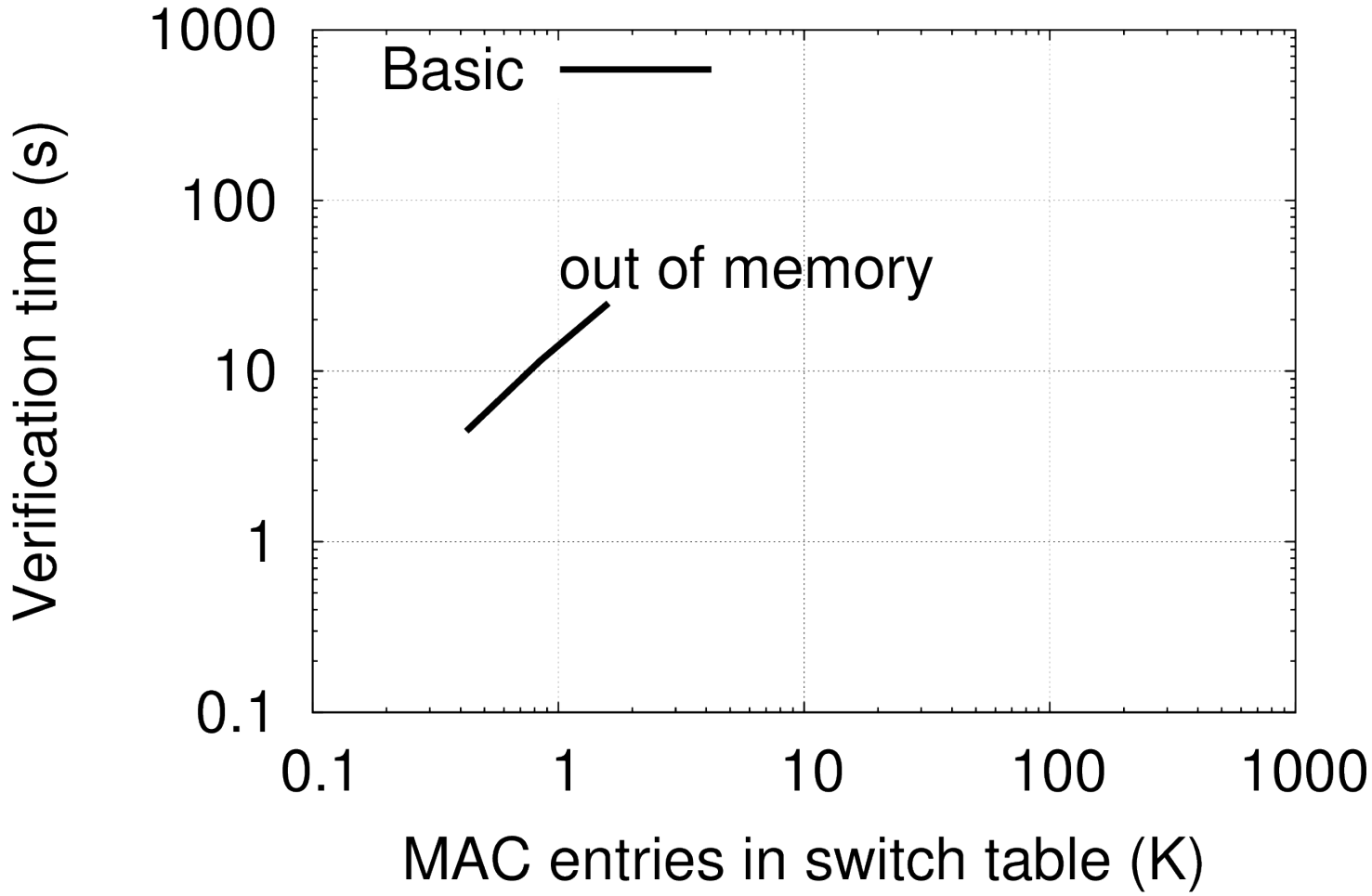
Functionality

Scalability

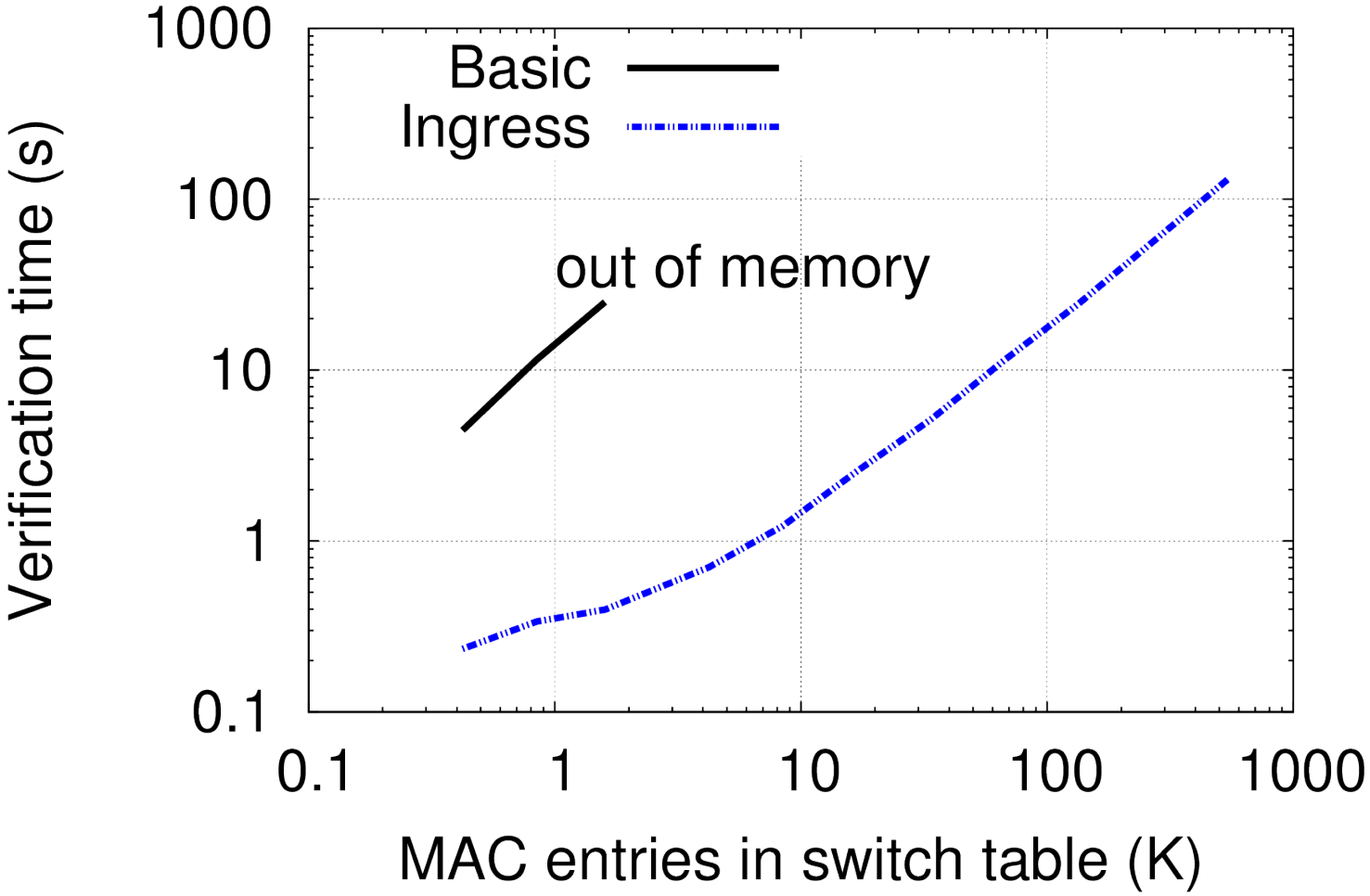
Verifiable properties

Property	HSA	NoD	SymNet
Reachability	✔	✔	✔
Loop Detection	✔	✖	✔
Header Field Invariance	✖	✖	✔
Arbitrary Packet Layout	✖	✔	✔
Tunneling	✖	✖	✔
Stateful Data Plane Processing	✖	✔	✔
Payload-sensitive Processing	✖	✖	✖
Properties Across Multiple Flows	✖	✖	✖

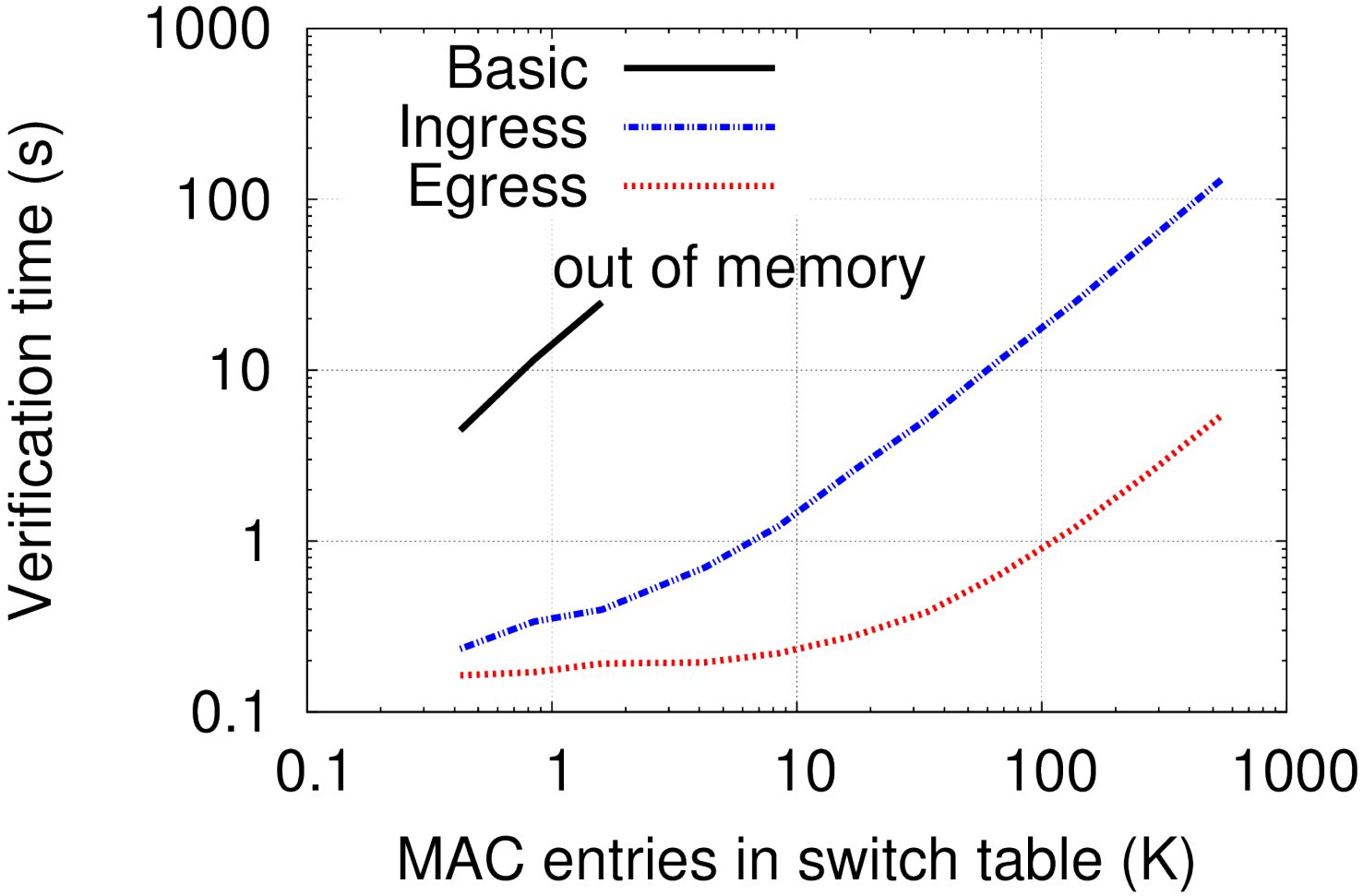
Does Symnet scale?



Does Symnet scale?



Does Symnet scale?



Analyzing bigger networks

- Stanford university backbone network
- Switches, routers and VLANs
  - Two-layer topology
  - Core routers have 180.000 entries in their FIBs

	HSA	Symnet
Model Generation Time	3.2 min	8.1 min
Runtime	24s	37s

Conclusions

SEFL + Symnet offers a deeper understanding of modern data planes at a low price.

Symnet is open-source
Check demo session tomorrow

Backup slides

TCP options parsing

Symbolic variable

Path 1

```
int crt = 0;
while (crt>=0 && crt<length &&
                        options[crt]){
    switch(options[crt]){
        case 1:
            crt++; break;
        case 2://MSS
        case 3://WINDOW SCALE
        case 4://SACK PERMITTED
        case 8://TIMESTAMP
            crt += options[crt+1]; break;
        default:
            //unknown options, scrub
            int len = options[crt+1];
            for (i=crt;i<crt+len;i++)
                options[i] = 1;
            crt += len; break;
    }
}
```

TCP options parsing

```
int crt = 0;
while (crt>=0 && crt<length &&
                        options[crt]){
    switch(options[crt]){
        case 1:
            crt++; break;
        case 2://MSS
        case 3://WINDOW SCALE
        case 4://SACK PERMITTED
        case 8://TIMESTAMP
            crt += options[crt+1]; break;
        default:
            //unknown options, scrub
            int len = options[crt+1];
            for (i=crt;i<crt+len;i++)
                options[i] = 1;
            crt += len; break;
    }
}
```

Path 1	Path 2	Path 3
options[0]==1	options[0] in {2,3,4,8}	options[0] not in {1,2,3,4,8}

TCP Options parsing

Leave the TCP options header outside of symbolic execution

Model TCP options as metadata instead

- "OPT-x" models the presence of option x
- "SZ-x" size of the option in bytes
- "DATA-x" value of the option

```
Assign("OPT30",ConstantValue(0)),
If (Constrain(TcpDst==80),Assign("OPT4",0),NoOp),
Assign("OPT2",1),
Assign("SIZE2",4),
If (Constrain("VAL2">1380),Assign("VAL2",1380),NoOp)
```

Does Symnet scale?

Symbolic execution of a core router

Prefixes	Basic	Ingress	Egress
1600	25s	2.1s	0.4s
62500	DNF	23.1s	5.6s
188500	DNF	DNF	18s

Running Klee for options parsing

Length	1	2	3	4	5	6
Number of paths	4	67	140	464	1095	3081
Runtime (s)	0.3	8	20	420	1500	9120